НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

О.Г. Ревинская

Символьные вычисления в MatLab

Учебное пособие

Томск
2018

УДК 681.3.06; 044.4
ББК 32.973.26-018.1я73
Р321

Ревинская О.Г.
Р321 Символьные вычисления в MatLab : учебное пособие. – Томск : Издательский Дом Томского государственного университета, 2018. – 528 с.
ISBN 978-5-94621-745-3

Описаны возможности ядра MatLab и интегрированных в MatLab пакетов MuPAD и Symbolic Math Toolbox для выполнения символьных вычислений на компьютере в процессе аналитического и численного решения задач, относящихся к таким разделам математики, как алгебра, линейная алгебра, векторный анализ, математический анализ, комплексный анализ, дифференциальные уравнения, интерполяция и т.д. Уделено внимание способам сохранения, преобразования, использования и графического представления результатов символьных вычислений в MatLab. Изложенный материал иллюстрирован примерами и сопровождается заданиями для самостоятельного выполнения на компьютере.

Для студентов, аспирантов физико-математических и технических специальностей, а также для преподавателей и специалистов классических и технических университетов, применяющих символьные вычисления в своей профессиональной деятельности.

УДК 681.3.06; 044.4
ББК 32.973.26-018.1я73

ISBN 978-5-94621-745-3

© О.Г. Ревинская, 2018
© Национальный исследовательский Томский государственный университет, 2018

ПРЕДИСЛОВИЕ

Символьные вычисления – это тождественные аналитические преобразования различных математических выражений, выполняемые на компьютере. В этом случае под математическими выражениями понимают аналитическую запись не только различных функциональных зависимостей, но и уравнений, неравенств и их систем. Преобразования могут выполняться как с целью решения соответствующих уравнений, неравенств или систем, так и для представления этих математических выражений в виде, отвечающем определенным критериям. Примерами могут служить запись известной функциональной зависимости в наиболее компактном виде, преобразование интеграла с помощью замены переменных и т.д.

При выполнении символьных вычислений на компьютере все аналитические преобразования выполняются с символами, входящими в математическое выражение, как с числами, по правилам, принятым в математике. Например, если в уравнении необходимо перенести из левой части в правую слагаемое, записанное в виде числа, и слагаемое, обозначенное символом, то эти преобразования выполняются по одним и тем же правилам как для числа, так и для символа.

Для автоматизации аналитических преобразований математических выражений разрабатываются и постоянно совершенствуются специализированные компьютерные приложения и математические пакеты, такие как Mathematica, Maple и т.д. Такие мощные системы разработки программных решений, как Mathcad, MatLab и т.п., включают в себя не только языки программирования, но символьные вычисления в виде дополнительных пакетов, использование которых возможно только вместе с той средой, в которую данный пакет интегрирован. Использование соответствующего пакета возможно только в том случае, если это предусмотрено приобретенной лицензией.

Глава 1. ПРИЛОЖЕНИЕ MUPAD – ПАКЕТ РАСШИРЕНИЯ MATLAB ДЛЯ ВЫПОЛНЕНИЯ СИМВОЛЬНЫХ ВЫЧИСЛЕНИЙ

В MatLab символьные вычисления доступны как в отдельном приложении MuPAD, так и в командном окне (`Command Window`) MatLab. Программы для выполнения символьных вычислений, написанные в приложении MuPAD, хранятся в специальных файлах с расширением mn, а программы, содержащие символьные вычисления, выполняемые в командном окне, можно сохранять в стандартном для MatLab m-файле (в скрипте или в функции). И в том и в другом случае программист может использовать сходные по синтаксису команды и функции для организации символьных вычислений, но их результаты в командном окне MatLab и в приложении MuPAD будут представлены по-разному. В командном окне MatLab информация может быть представлена только в текстовом формате, поэтому все математические выражения (и введенные, и преобразованные в процессе символьных вычислений) записываются в строку (рис. 1.1). Это значительно затрудняет визуальный контроль результатов символьных вычислений.

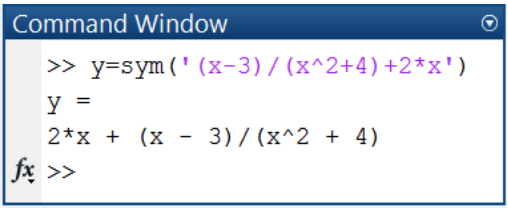


Рис. 1.1. Командное окно среды MatLab с результатами символьных вычислений

В приложении MuPAD исходное математическое выражение также необходимо записывать в строку, но результат вычислений будет представлен в графическом виде с соблюдением всех принятых в математике правил и обозначений (рис. 1.2). Такое представление результатов вычислений значительно облегчает их визуальный контроль. Кроме того, в приложении MuPAD существует специализированная панель инструментов,

содержащая шаблоны большинства команд и функций, используемых при выполнении символьных вычислений в MatLab. В редакторе m-файлов такой вспомогательной специализированной панели инструментов нет. Здесь помощь программисту в выборе нужной команды или функции может оказать только справочная система.

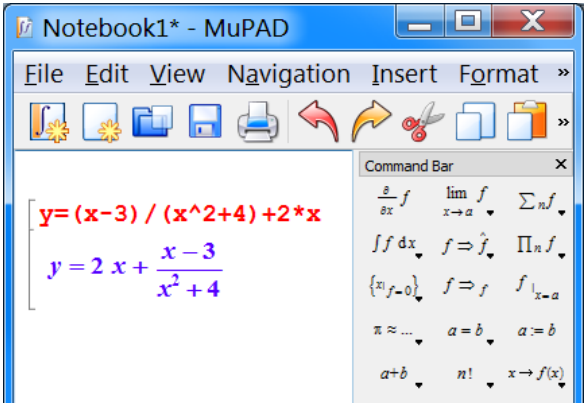


Рис. 1.2. Фрагмент окна приложения MuPAD
с результатами символьных вычислений

Особенности выполнения символьных вычислений в командном окне MatLab и тонкости, связанные с программированием этих вычислений в m-файле, рассмотрены в гл. 11.

1.1. Приложение MuPAD

При наличии соответствующей лицензии приложение MuPAD входит в комплект поставки MatLab и предназначено не только для выполнения символьных вычислений в этой среде, но и для численного решения различных математических задач.

1.2. Запуск приложения MuPAD

Запустить приложение MuPAD можно только из среды MatLab.

Начиная с версии 2012 панель инструментов основного окна среды MatLab содержит три вкладки: `Home`, `Plots`, `Apps` (рис. 1.3). Все входящие в состав MatLab (в рамках действующей лицензии) приложения

располагаются на вкладке `Apps` панели инструментов основного окна MatLab. На этой вкладке расположена раскрывающаяся панель кнопок, в первой строке которой находится кнопка `MuPAD Notebook` (рис. 1.4).

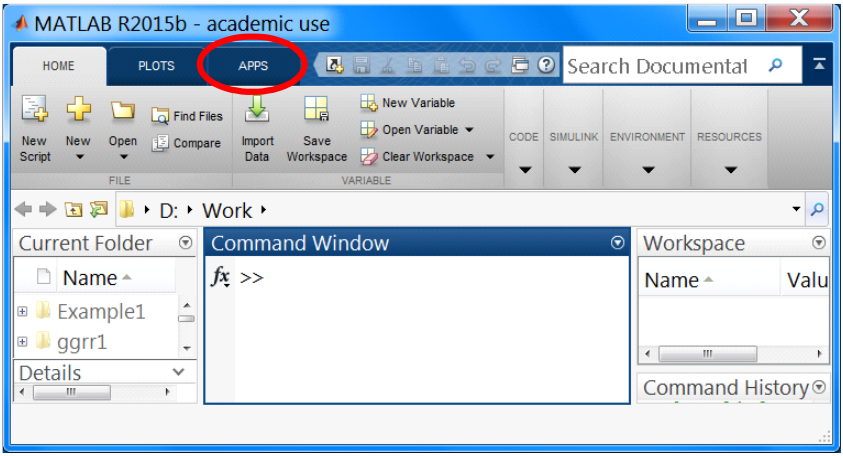


Рис. 1.3. Основное окно среды MatLab

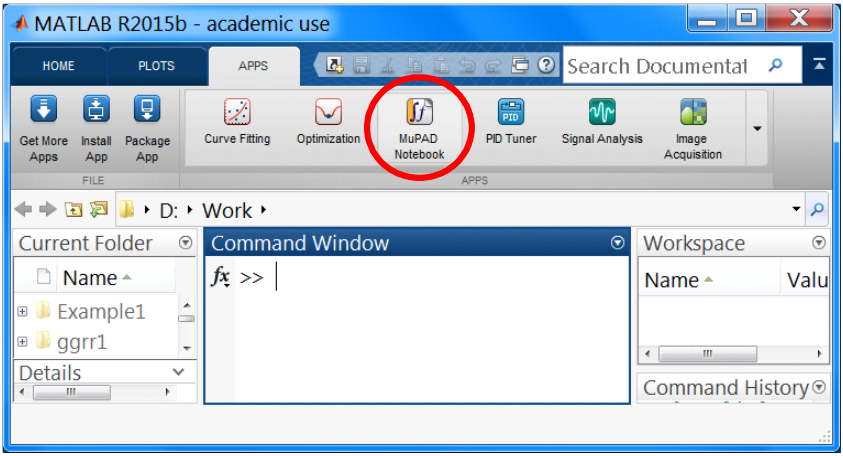


Рис. 1.4. Вкладка `Apps` панели инструментов основного окна среды MatLab

Чтобы запустить приложение MuPAD, необходимо нажать на эту кнопку. В результате появится окно приложения MuPAD.

Кроме того, приложение MuPAD можно запустить, написав его название в командном окне MatLab: `mupad`. После нажатия кнопки `Enter` также появится окно приложения MuPAD.

1.3. Интерфейс приложения MuPAD

Окно приложения MuPAD состоит из пяти различных по своему функциональному назначению частей (рис. 1.5): область редактирования, главное меню, стандартная панель инструментов, командная панель и строка состояния.

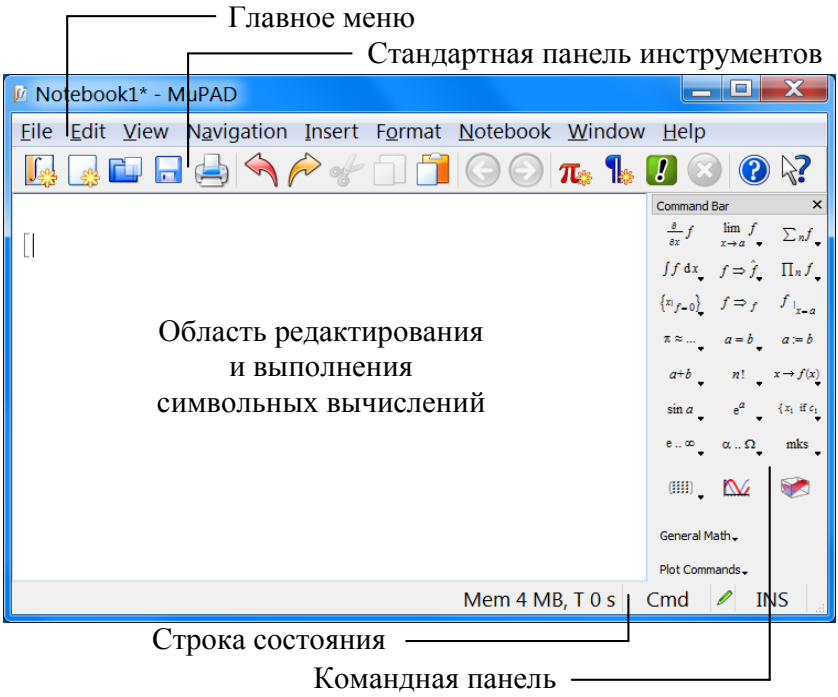


Рис. 1.5. Структура окна приложения MuPAD

Область редактирования занимает бóльшую часть окна MuPAD и предназначена для редактирования и выполнения символьных вычислений. Над областью редактирования расположена строка главного меню MuPAD, которая содержит команды управления приложением. Часто используемые команды (такие как «Создать», «Сохранить», «Скопировать» и т.д.) дублируются в виде кнопок на стандартной панели инструментов (`Standard ToolBar`), которая располагается горизонтально между строкой главного меню и областью редактирования. Справа от области редактирования располагается командная панель (`Command Bar`), которая содержит кнопки для добавления в редактируемый файл шаблонов команд и функций, предназначенных в MuPAD для выполнения символьных вычислений и численных расчетов.

В нижней части окна MuPAD расположена строка состояния, в которой контекстно отображается некоторая вспомогательная информация.

Если настройки окна MuPAD изменить, кроме стандартной и командной панелей инструментов можно отобразить также панель инструментов для форматирования (`Format ToolBar`), а также панель поиска и замены, обозревателя объектов и др. По умолчанию панель инструментов для форматирования располагается горизонтально выше области редактирования, а остальные панели – справа. Все панели (в том числе и стандартная панель инструментов и командная панель) являются вспомогательными, предназначены для повышения комфортности программирования и при необходимости их отображение в окне MuPAD можно отключить. Это никак не отразится на функциональности приложения MuPAD.

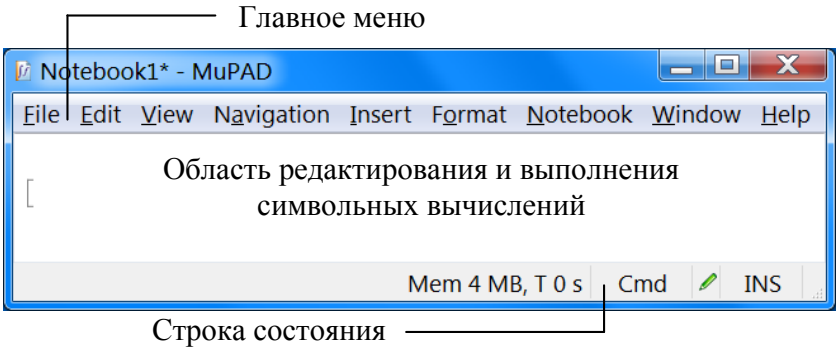


Рис. 1.6. Структура окна приложения MuPAD без вспомогательных панелей

Чтобы отобразить (отключить) какую-либо из вспомогательных панелей, необходимо выполнить однократный щелчок правой клавишей мыши на строке главного меню и выбрать в раскрывшемся списке пункт с названием соответствующей панели.

Строка главного меню, область редактирования и строка состояния отображаются в окне MuPAD всегда. Если отображение всех вспомогательных панелей отключить, то окно MuPAD будет выглядеть, как показано на рис. 1.6.

Строка главного меню MuPAD содержит команды управления приложением, сгруппированные тематически следующим образом:

- **`File`**: позволяет выполнять основные операции с файлами (создать новый файл, открыть, закрыть, сохранить, распечатать существующий файл и т.д.);
- `Edit`: позволяет выполнять основные операции редактирования текста программы;
- `View`: позволяет настраивать интерфейс окна MuPAD;
- `Navigation`: содержит дополнительные команды по навигации;
- **`Insert`**: позволяет вставлять в текст программы различного типа поля и объекты;
- `Format`: позволяет изменять цвет, размер и тип шрифтов, используемых в области редактирования;
- **`Notebook`**: позволяет управлять выполнением и отладкой программы;
- `Window`: позволяет переключаться между открытыми файлами (если их несколько);
- `Help`: служит для обращения к встроенной справочной системе MuPAD.

На стандартную панель инструментов (рис. 1.5) вынесен в виде кнопок ряд команд, считающихся часто используемыми:

создать новый mn-файл в редакторе приложения MuPAD – New Notebook (`Ctrl-N`);

создать новый m-файл в редакторе MatLab – New Editor (`Ctrl-Shift-N`);

открыть файл – Open (`Ctrl-O`);

сохранить файл – Save (`Ctrl-S`);

напечатать файл – Print (`Ctrl-P`);

отменить последнее действие – Undo (`Ctrl-Z`);

восстановить отмененное действие – Redo (`Ctrl-Y`);

вырезать – Cut (`Ctrl-X`);

скопировать – Copy (`Ctrl-C`);

вставить – Paste (`Ctrl-V`);

назад – Go Back (`Alt-Left`);

вперед – Go Forward (`Alt-Right`);

вставить командное поле – Insert Calculation (`Ctrl-I`);

вставить текстовое поле – Insert Text Paragraph (`Ctrl-T`);

выполнить вычисление – Evaluate;

остановить вычисление – Stop;

вызвать справку – Open Help (`F1`);

вызвать контекстную справку – Context Help (`Shift-F1`).

Название каждой кнопки (например, Copy, Paste и т.д.) отображается в виде всплывающей подсказки при наведении на нее курсора мыши.

В главном меню слева от вынесенных на стандартную панель инструментов команд размещается изображение, такое же как на соответствующей этой команде кнопке.

Область редактирования и выполнения программы занимает бóльшую часть окна MuPAD (см. рис. 1.5) и предназначена для ввода программного кода, поясняющего текста и вывода результатов.

Программный код вводится в командном поле (`Calculation`), отмеченном слева квадратной скобкой. Результаты выводятся в поле, тоже отмеченном слева квадратной скобкой, но пристыкованном снизу к командному (см. рис. 1.2). Это поле называется полем вывода (`Output`). Для добавления к программному коду поясняющего текста (комментария) используют текстовое поле (`Text Paragraph`).

Чтобы в тексте программы удобнее было различать поля разного типа, в них используются шрифты разного цвета и начертания: программный код отображается шрифтом красного цвета; результаты вычислений – синего; комментарии – черного цвета (рис. 1.7).

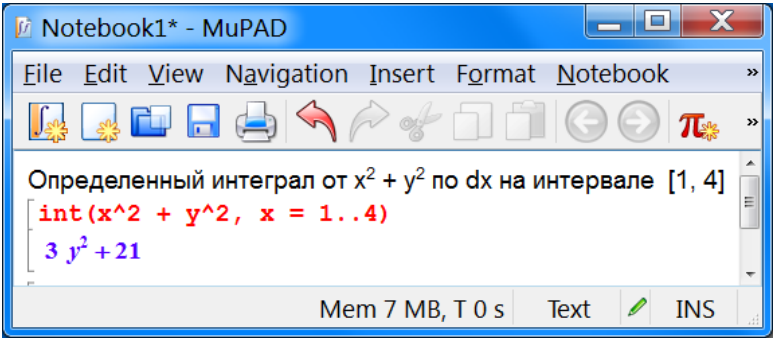


Рис. 1.7. Три типа полей в области редактирования MuPAD: текстовое, командное поле и поле вывода результатов

При создании в MuPAD нового файла в него автоматически добавляются два текстовых поля (`Text Paragraph`) и между ними одно командное (`Calculation`). Остальные поля (`Calculation` и `Text Paragraph`), необходимые для написания программы, программист должен вставлять сам. Поля, предназначенные для вывода результатов

(Output), вставляются в файл автоматически после выполнения соответствующих вычислений. После каждого командного поля добавляется одно поле вывода.

Командная панель (Command Bar) содержит шаблоны доступных из MuPAD команд и функций, предназначенных для выполнения символьных вычислений и численных расчетов. Кнопки, расположенные на этой панели, будут детально описаны в следующих главах пособия при анализе соответствующих команд и функций.

Для оптимизации редактирования и выполнения программного кода в окне приложения MuPAD кроме стандартной панели инструментов (Standard ToolBar), строки состояния (Status Bar) и командной панели (Command Bar) можно также отобразить панели Find and Replace Bar и Format Toolbars. Чтобы настроить отображение любой из этих панелей, используют одноименные команды из раздела View главного меню MuPAD или из раскрывающегося списка, для вызова которого необходимо выполнить однократный щелчок правой клавишей мыши в любом месте строки главного меню.

Чтобы закончить работу с MuPAD, достаточно закрыть приложение с помощью системных кнопок в заголовке окна или выбрав в разделе File главного меню команду Close.

1.4. Создание, редактирование, сохранение mn-файла

При запуске приложения MuPAD автоматически создается новый файл с расширением mn.

Чтобы принудительно создать новый файл, необходимо нажать на стандартной панели инструментов MuPAD, расположенной над областью редактирования, кнопку New Notebook или выбрать в разделе File главного меню команду New Notebook (см. рис. 1.4). В результате появится еще одно окно MuPAD, в области редактирования которого будет создан новый файл.

Каждый вновь созданный файл содержит два (пустых) текстовых поля и одно (пустое) командное (см. рис. 1.4, 1.5). Чтобы введенный текст имел смысл программного кода, его необходимо вводить в командных полях. Для этого следует добавить в файл необходимое количество командных полей. Поля в MuPAD добавляются по одному. Чтобы в текущий файл добавить одно командное поле, необходимо на стандартной панели инструментов,

расположенной над областью редактирования, нажать кнопку `Insert Calculation`, или в разделе `Insert` главного меню выбрать команду `Calculation`, или на клавиатуре нажать сочетание кнопок `Ctrl-I`. Чтобы в текущий файл добавить одно текстовое поле, необходимо на стандартной панели инструментов нажать кнопку `Insert Text Paragraph`, или в разделе `Insert` главного меню выбрать команду `Text Paragraph`, или на клавиатуре нажать сочетание кнопок `Ctrl-T`. Новое поле вставляется до или после того поля, в котором находился курсор до вставки. Если перед вставкой курсор находился в конце текстового поля или в любом месте поля вывода, новое поле добавляется после него. Если курсор находился в любом месте командного поля, новое поле добавляется после его поля вывода (если вывод результатов ранее был осуществлен). Если курсор находился в начале текстового поля, то новое поле добавляется перед ним.

В добавленных полях можно писать и редактировать программный код для выполнения необходимых символьных вычислений (в командных полях) или комментарии к ним (в текстовых полях). Для редактирования текста программы, как и в других редакторах, используют клавиатуру и манипулятор мышь. Выделенные с помощью мыши или клавиатуры фрагменты программного кода или последовательности полей можно перемещать, копировать и удалять, воспользовавшись кнопками `Cut`, `Copy`, `Past` стандартной панели инструментов или соответствующими командами в разделе `Edit` главного меню, а также пунктами контекстного меню, для вызова которого необходимо нажать правую клавишу мыши над выделенным фрагментом.

Для удаления отдельных символов или пустых полей в редактируемом файле, как правило, используют кнопки `Delete` и `BackSpace` на клавиатуре. Для удаления больших фрагментов текста или последовательности полей достаточно их выделить с помощью кнопок клавиатуры или мыши и нажать на клавиатуре кнопку `Delete`.

Каждое поле (командное, текстовое, поле вывода) может состоять из нескольких строк. В поле вывода строки добавляются только автоматически, так чтобы можно было полностью отобразить все результаты вычислений. В командном и текстовом полях ***дополнительные строки*** можно вставлять вручную. Для добавления строк в текстовом поле достаточно расположить курсор в соответствующем месте существующей строки этого поля и нажать на клавиатуре кнопку `Enter` необходимое количество

раз. Для добавления дополнительных строк в командное поле следует нажать на клавиатуре сочетание кнопок `Shift–Enter` соответствующее количество раз.

Если в процессе редактирования в файле необходимо оставить только программный код, все поля вывода можно удалить, выбрав в разделе `Edit` главного меню команду `Delete All Outputs`. Для удаления одного поля вывода, пристыкованного к командному полю, в котором в данный момент находится курсор, можно использовать команду `Delete Output` из раздела `Edit` главного меню.

Кнопка `Save` на стандартной панели инструментов позволяет сохранить открытый в активном окне MuPAD файл на рабочем диске компьютера. Раздел `File` главного меню позволяет также воспользоваться для сохранения файла командами `Save`, `Save as…` и `Export`.

Для быстрого выполнения сохранения команды `Save` и `Save as…` дублируются комбинациями кнопок клавиатуры: `Ctrl–S` и `Ctrl–Shift–S` соответственно. Команду `Export` можно вызвать только через раздел `File` главного меню. Каждая из команд `Save`, `Save as…` и `Export` применяется только к файлу, расположенному в активном окне.

При первоначальном сохранении программы результаты применения команд `Save` и `Save as…` не отличаются. В обоих случаях появится диалоговое окно, содержащее поле для ввода имени файла, а также информацию об уже существующих файлах и их расположении.

При повторном сохранении программы результаты применения команд `Save` и `Save as…` существенно отличаются. Повторное использование команды `Save` для сохранения программы с заданным именем файла приводит к сохранению новой версии программы с прежним именем файла. Использование команды `Save as…` для сохранения программы с уже заданным именем файла приведет к появлению еще одного файла, название которого следует ввести в появившемся диалоговом окне.

Команда `Export` из раздела `File` главного меню позволяет конвертировать текст программы вместе с полученными в ней результатами, графиками и комментариями в один из трех форматов: pdf, html или txt. Это позволяет организовать просмотр результатов выполненных в MuPAD вычислений без использования этого приложения.

1.5. Выполнение вычислений

Чтобы обеспечить сохранность текста набранной или измененной программы, перед выполнением ее необходимо сохранить на диске компьютера!

Символьные вычисления, запрограммированные в mn-файле, в приложении MuPAD выполняются построчно в порядке их следования в файле. Чтобы выполнить весь программный код файла, открытого в активном окне MuPAD, необходимо в разделе `Notebook` главного меню этого окна выбрать команду **`Evaluate All`**. Выполнение программного кода будет происходить последовательно, начиная с первого командного поля, заканчивая последним. Переход от одного командного поля к другому происходит автоматически. Текстовые поля при этом пропускаются. Перед выполнением очередного командного поля пристыкованное к нему поле вывода автоматически удаляется, а после завершения вычислений добавляется вновь, отображая полученные результаты. Далее MuPAD переходит к выполнению следующего командного поля. Если записанные в каком-то из полей вычисления выполнить не удалось, в соответствующем поле вывода появится сообщение об ошибке, начинающееся со слова «Error:», за которым следует описание ошибки. На этом выполнение вычислений прерывается.

Результаты всех успешно выполненных вычислений не только отображаются в соответствующих полях вывода, но и сохраняются в памяти MuPAD. При попытке повторного (без очистки памяти) выполнения сделанных ранее вычислений в поле вывода отображаются извлеченные из памяти MuPAD результаты, а повторные вычисления не проводятся.

Для выполнения вычислений, запрограммированных в отдельном командном поле mn-файла, необходимо поместить курсор в любое место этого поля или соответствующего ему поля вывода и нажать на клавиатуре кнопку `Enter` или на стандартной панели инструментов – кнопку `Evaluate` (или команду `Evaluate` в разделе `Notebook` главного меню). В этом случае MuPAD попытается выполнить запрограммированные в соответствующем командном поле вычисления, независимо от того, выполнялись ли вычисления, написанные выше и ниже данного командного поля, или нет. Но при выполнении этих вычислений будут учтены все полученные ранее (по времени) результаты, хранящиеся в памяти MuPAD. После успешного выполнения вычислений курсор автоматически переходит к командному полю, расположенному в файле вслед за выполненным. Это

позволяет выполнять отладку программы параллельно с ее написанием: после ввода каждого фрагмента кода нажимают кнопку `Enter` на клавиатуре для его выполнения и контроля полученных результатов, после чего вводят код в следующее поле. Следует заметить, что если, нажав кнопку `Enter`, выполнить программный код, записанный в командном поле, являющемся последним в данный момент в файле, после выполнения вычислений в файл автоматически будет добавлено еще одно командное поле вслед за выполненным.

Если в процессе выполнения программы (или ее отдельного поля) возникает необходимость прервать вычисления, можно воспользоваться кнопкой `Stop` на стандартной панели инструментов или командой `Stop` в разделе `Notebook` главного меню, а также кнопкой `Pause` на клавиатуре.

После завершения вычислений файл программы содержит и программный код, и пояснения к нему (если они были написаны в текстовых полях), и результаты вычислений. В таком виде файл может быть сохранен. Когда в дальнейшем такой файл вновь будет открыт в приложении MuPAD, он по-прежнему будет содержать не только программный код и текстовые пояснения к нему, но и последние сохраненные результаты. Обратите внимание, что при открытии mn-файла сохраненные результаты вычислений из файла в память MuPAD НЕ ЗАГРУЖАЮТСЯ. Результаты вычислений помещаются в память MuPAD только после непосредственного завершения вычислений во время текущей работы MuPAD. Поэтому, прежде чем вносить изменения в написанную ранее программу, записанные в ней вычисления необходимо выполнить (например, с помощью кнопок `Evaluate` или `Evaluate All` из раздела `Notebook` главного меню). Только после этого хранящиеся в памяти MuPAD результаты станут доступны для использования в дальнейших вычислениях.

1.6. Встроенная справочная система

В разрешении различных затруднений, возникающих при разработке программ, программисту может помочь встроенная справочная система среды MatLab (`Help`). При обращении к справочной системе MatLab из приложения MuPAD она выводит информацию о стандартных командах и функциях для выполнения символьных вычислений и численных расчетов, доступных в MuPAD, а также об используемых в этом приложении зарезервированных словах и операторах. Вызов справочной системы

осуществляется либо с помощью кнопки `Help` на стандартной панели инструментов, либо с помощью команды `Open Help` из раздела `Help` главного меню, либо с помощью кнопки `F1` на клавиатуре. Информацию о какой-либо стандартной функции (или команде) можно получить, выполнив однократный щелчок правой клавишей мыши над написанным в тексте программы названием этой функции (команды) и выбрав из раскрывшегося контекстного меню пункт `Help about '...'` (рис. 1.8).

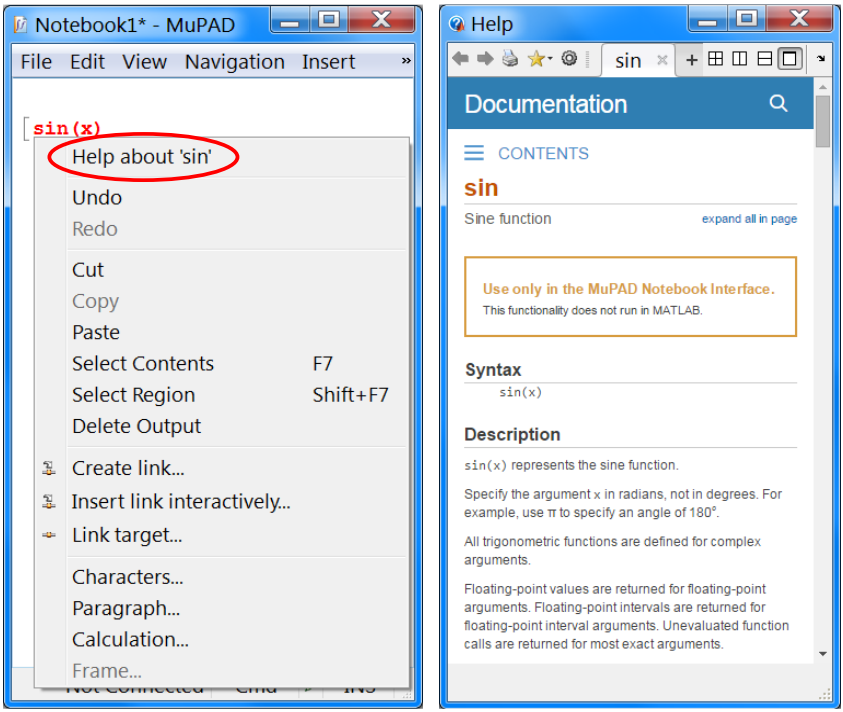


Рис. 1.8. Контекстный вызов справочной информации в MuPAD

Справочная система MuPAD также доступна на сайте компании MathWork по адресу: http://www.mathworks.com/help/symbolic/mupad-1.html

Глава 2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА И СРЕДСТВА ПРОГРАММИРОВАНИЯ В MUPAD

2.1. Структура программы

Текстовые поля служат для пояснения выполняемых программой вычислений и не влияют ни на их последовательность, ни на их результат. Поэтому под программой в MuPAD, как правило, понимают совокупность командных полей.

Структура программы, написанной в MuPAD, определяется логической последовательностью операций, необходимых для выполнения определенных аналитических преобразований и численных расчетов. При разработке программы следует помнить, что программный код каждого командного поля выполняется с учетом всех полученных ранее (по времени) результатов, хранящихся в памяти MuPAD. Технически программист может выполнять программный код, записанный в одной и той же совокупности командных полей, в различной последовательности, если будет инициировать выполнение каждого командного поля вручную (через нажатие кнопки `Enter` на клавиатуре, например). Если в каждом из командных полей запрограммированы независимые друг от друга вычисления, то их результат не зависит от последовательности этих вычислений. Если в разных командных полях программы записаны связанные друг с другом вычисления, то располагать их в файле рекомендуется в такой последовательности, в которой они должны будут выполняться. Это избавит от необходимости запоминать, в каком порядке нужно выполнять эти вычисления, и позволит использовать автоматическое выполнение всей программы с помощью команды `Evaluate All` (из раздела `Notebook` главного меню), когда командные поля выполняются последовательно одно за другим, начиная с первого. Это особенно полезно при написании больших программ.

Результаты всех вычислений могут сохраняться в памяти компьютера, пока MuPAD не будет закрыт. Некоторые из них могут учитываться при выполнении последующих вычислений. Если в каком-то месте программы (как правило, в начале) необходимо полностью удалить из памяти все полученные ранее результаты и восстановить настройки MuPAD, заданные

по умолчанию, используют оператор (команду) **`reset()`**. Оператор `reset()` легко написать, используя кнопки на клавиатуре. Кроме того, этот оператор можно вставить в текст программы, начиная с позиции курсора, если воспользоваться пунктом `Reset Session` раскрывающегося списка `General Math` на командной панели `Command Bar`. Так как выполнение программы с помощью команды `Evaluate All` (из раздела `Notebook` главного меню) начинается с первого командного поля файла, то рекомендуется программный код любых вычислений в MuPAD начинать с оператора `reset()` – размещать его в первом командном поле файла. В этом случае результаты предыдущего выполнения этой же (или любой другой) программы не смогут оказать влияние на ее повторное выполнение, если выполнение начнется с первого командного поля. Наличие нескольких операторов `reset()` в разных местах mn-файла равносильно нескольким программам, которые при использовании команды `Evaluate All` (из раздела `Notebook` главного меню) будут выполняться последовательно одна за другой, но независимо друг от друга.

В каждом командном поле можно записать несколько операторов (команд), управляющих символьными вычислениями или численными расчетами. Операторы, записанные в одном командном поле, необходимо отделять друг от друга одним из следующих символов: точка с запятой (`;`), двоеточие (`:`). Если после оператора стоит точка с запятой, то вывод результатов вычислений, инициируемых этим оператором, осуществляется в отдельной строке поля вывода. Если после оператора стоит двоеточие, полученный с его помощью результат не выводится. После последнего оператора (команды) знак препинания может отсутствовать. В этом случае результат инициируемых им вычислений будет выведен в последней строке соответствующего поля вывода. Знаки препинания (`;` или `:`) между операторами необходимо ставить, даже если они записаны в разных строках одного и того же командного поля. Знаки препинания между операторами можно опускать (не указывать), только если каждый из них записан в отдельном командном поле.

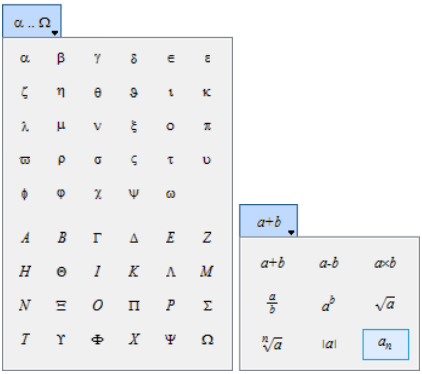
2.2. Идентификаторы

Идентификатор – уникальный набор символов, предназначенный в программировании для обозначения переменной, константы, команды, оператора, функции и т.п.

В MuPAD в качестве символов идентификатора можно использовать все буквы латинского алфавита (как строчные, так и прописные), цифры и символ _. Идентификаторы НЕ МОГУТ содержать пробел, точку, запятую, двоеточие, точку с запятой, начинаться с цифры.

Идентификаторы подразделяются на стандартные и нестандартные. Часть идентификаторов зарезервирована в MuPAD для обозначения стандартных команд, операторов и функций. Такие идентификаторы называют стандартными идентификаторами или зарезервированными словами. Использование стандартных идентификаторов не по назначению может привести к появлению ошибки при попытке выполнения программного кода.

Нестандартные идентификаторы не связаны с каким-либо фиксированным значением или объектом MuPAD, их предназначение и методы использования определяются программистом по своему усмотрению.



а *б*

Рис. 2.1. Раскрывающиеся панели кнопок на командной панели `Command Bar`:
а – `Greek letters`;
б – `Basic operators`

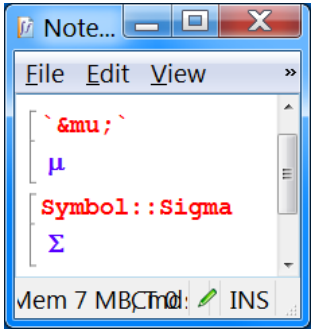


Рис. 2.2. Идентификаторы, отображающиеся в поле вывода как греческие буквы, введенные в командном поле: μ – с помощью кнопки на командной панели; Σ – с помощью указателей

Определенные последовательности символов отображаются в поле вывода как греческие буквы: α, β, γ и т.д. Чтобы вставить в командное поле набор символов, воспроизводящий в поле вывода определенную греческую букву, можно воспользоваться раскрывающейся панелью кнопок

`Greek letters` (рис. 2.1, *а*) на командной панели `Command Bar`. После выбора нужной греческой буквы в том месте командного поля, в котором располагался курсор, появится соответствующий набор символов. Например, для отображения в поле вывода греческой буквы μ в командное поле можно вставить следующий набор символов: `` `μ` ``. Идентификаторы, отображаемые в поле вывода как греческие буквы (несмотря на то, что записываются строго определенным образом) также относятся к нестандартным.

В командном поле все идентификаторы записываются в строку. Однако в поле вывода некоторые идентификаторы могут выглядеть как индексированные. Чтобы какая-то часть символов, входящих в данный идентификатор, отображалась в поле вывода как нижний индекс, используют квадратные скобки или символ _. При использовании символа _ как нижний индекс будет отображаться только одна буква или цифра, расположенная сразу же после этого символа. Чтобы несколько символов в идентификаторе интерпретировались как нижние индексы, перед каждым из них необходимо поставить символ _. Другим способом создания индексированных идентификаторов является использование квадратных скобок. В этом случае все символы, расположенные в квадратных скобках, считаются нижним индексом. Друг от друга индексы отделяются запятой. Каждый индекс может состоять из нескольких символов. Индексы, обозначенные квадратными скобками (в отличие от индексов, отмеченных символом _), интерпретируются как самостоятельные идентификаторы и поэтому могут изменяться в процессе выполнения программы. Кнопка `Indexed variable` на раскрывающейся панели кнопок `Basic operators` (рис. 2.1, *б*), расположенной на командной панели `Command Bar`, позволяет вставить в позицию курсора шаблон индексированного идентификатора, записанного с помощью квадратных скобок. Этот шаблон имеет вид: `#a[#n]`. В этом шаблоне поля, требующие заполнения, отмечены символом #. Этот символ использовать в идентификаторах запрещается, поэтому пока все символы # не будут удалены из шаблона, его не удастся использовать в качестве идентификатора.

Примеры нестандартных идентификаторов, написанных в командном поле:

```
X          X1          X_1         X[1]
a          a12         a_1_2       a[k, 2]
h2s3       h_2s_3      h_2s[k]     Y[min]
```

tau	`τ`	`τ_1`	`τ`[n]

Эти же идентификаторы в поле вывода результатов будут выглядеть следующим образом:

X	X1	X_1	X_1
a	a12	a_{12}	$a_{k,2}$
h2s3	h_2s_3	h_2s_k	$Y_{\min}$
tau	τ	τ_1	τ_n

Идентификаторы, написанные строчными и прописными буквами, считаются разными.

Кроме нижних индексов идентификатор может также содержать верхние индексы, а также надстрочные и подстрочные символы, принятые в математике. Для добавления к идентификатору в поле вывода надстрочных и подстрочных символов в командном поле необходимо использовать специальные указатели `Symbol::` (табл. 2.1).

Таблица 2.1

Правила записи идентификаторов с надстрочными и подстрочными символами

	Математика	**MuPAD**
Сопряжение	x^*	`Symbol::accentAsterisk(x)`
Тильда	$\tilde{x}$	`Symbol::accentTilde(x)`
Оператор	$\hat{x}$	`Symbol::accentHat(x)`
Вектор	$\vec{x}$	`Symbol::accentRightArrow(x)`
Производная по времени	$\dot{x}$	`Symbol::accentDot(x)`
Вторая производная по времени	$\ddot{x}$	`Symbol::accentDoubleDot(x)`
Третья производная по времени	$\dddot{x}$	`Symbol::accentTripleDot(x)`
Среднее значение	$\bar{x}$	`Symbol::accentOverBar(x)`
Нижний индекс	x_n	`Symbol::subScript(x, n)`
Верхний индекс	x^m	`Symbol::superScript(x, m)`
Нижний и верхний индексы	x_n^m	`Symbol::subSuperScript(x, n, m)`

Указатель `Symbol::` также используется для отображения в поле вывода букв греческого алфавита. В этом случае после `Symbol::` необходимо написать латинскими буквами название соответствующей греческой буквы. Идентификатором в этом случае является набор символов, записанный после указателя. Например, если в командном поле написать: `Symbol::Sigma`, то в поле вывода будет отображаться греческая буква Σ (рис. 2.2), а идентификатором считается `Sigma`. То есть указатель `Symbol::` используется только для изменения способа отображения идентификатора в поле вывода. Но в виде греческих букв могут отображаться только идентификаторы, записанные строго определенным образом (например, `phi`, `xi`, `theta` и т.д.). При использовании с указателем `Symbol::` других идентификаторов в поле вывода отображается `FAIL` – зарезервированный идентификатор MuPAD, используемый для обозначения результата в случае неудачи (с англ. *неудача*).

Указатели вида `Symbol::`, `orthpoly::`, `unit::` и т.д. позволяют организовать обращение к соответствующим библиотекам (специализированным наборам стандартных констант и функций). После указателя записывается название конкретной стандартной функции (см. разд. 2.7.1) или константы. Например: `unit::mm` – единицы измерения миллиметры; `numeric::solve(…)` – численное решение уравнений и т.д.

2.3. Константы и переменные

Переменные, константы – «контейнеры», предназначенные для хранения информации в программе. Если в процессе выполнения программы информация, хранящаяся в некотором «контейнере», изменяется, такой «контейнер» считается переменной. Если в процессе выполнения программы информация, хранящаяся в некотором «контейнере», не изменяется, такой «контейнер» считается константой. В процессе разработки программы отдельные константы могут становиться переменными, и наоборот, в зависимости от выполняемых вычислений. Может ли информация, хранящаяся в данном «контейнере», начать со временем изменяться, в MatLab (и его приложениях) никаким образом не регламентируется. Учитывая, что каждый «контейнер» (хранящаяся в нем информация) в любой момент может быть изменен, все «контейнеры», предназначенные для хранения информации в программе, написанной в любом из приложений среды MatLab, в частности в MuPAD, условно считаются ***переменными***.

Идентификаторы всех переменных в пределах данной программы должны быть уникальными. Наборы символов, начинающиеся с указателя стандартной библиотеки (например, `Symbol::`), нельзя использовать в качестве идентификаторов переменных в программе. Тип переменных в MuPAD (так же как и в MatLab) определяется автоматически в зависимости от типа и структуры информации, хранящейся в них. Так как программы, написанные в MuPAD, предназначены для выполнения символьных вычислений и численных расчетов, то в используемых в них переменных, как правило, хранятся числовые данные (целочисленные, рациональные, вещественные, комплексные) или символы. Но в зависимости от структуры хранящейся в переменной информации числа и (или) символы могут быть по-разному скомпонованы и представлены в виде числовых и символьных значений, выражений, уравнений, неравенств, списков, множеств, объектов и т.д. Способы создания переменных, в которых информация структурирована по-разному, рассмотрены в следующих главах пособия.

Таблица 2.2

Правила записи стандартных констант и переменных окружения

	Математика	**MuPAD**
Встроенное число π (3,141592653589793…)	π	`PI`
Мнимая единица ($\sqrt{-1}$)	i	`I`
Число *e* (2,718281828459046…)	e	`E`
Бесконечность	∞	`infinity`
Неизвестно		`UNKNOWN`
Истина		`TRUE`
Ложь		`FALSE`
Количество значащих цифр (для вещественных чисел, записанных в десятичном формате) – переменная окружения		`DIGITS = 10`
Количество членов ряда – переменная окружения		`ORDER = 6`

В MuPAD переменная считается ***числовой***, если в ней хранятся только числа (одно или несколько). Переменная считается ***символьной***, если ее значение нигде ранее в программе не задано или задано в виде набора

символов (например, символьного выражения). Все операции с символьными переменными выполняются так же, как и с числовыми. Операции с переменными, содержащими символьные выражения, выполняются так же, как аналитические преобразования с соответствующими выражениями в математике. В зависимости от структуры хранящихся в переменной числовых и (или) символьных данных каждой переменной в MuPAD автоматически сопоставляется один из стандартных типов.

При необходимости тип любой используемой в программе переменной можно узнать с помощью стандартной функции `type`, если в качестве аргумента этой функции указать идентификатор соответствующей переменной или константу. Например:

```
type(3)      type(1.3)   type(x)      type(x+2)
```

В MuPAD переменные автоматически создаются в памяти (выделяется необходимый для хранения информации объем памяти) при первом употреблении переменной в тексте программы и не нуждаются в специальном объявлении. При сохранении в созданную ранее переменную информации, имеющей другую структуру, тип переменной автоматически переопределяется. Чтобы удалить какую-либо ранее заданную переменную, используют команду `delete`, указав после нее идентификатор удаляемой переменной. Если необходимо удалить несколько переменных, их идентификаторы после команды `delete` указываются через запятую.

Например: `delete x, y`

При удалении индексированных переменных, записанных с помощью квадратных скобок, индексы не указывают. Команда `delete` не только удаляет информацию, хранящуюся в указанных переменных, но и освобождает память, зарезервированную для каждой из них. Если командное поле содержит только команду `delete` (со списком удаляемых переменных), после его выполнения поле вывода не появляется.

Все переменные, значения которых в программе явным образом не заданы, считаются символьными. Стандартные числовые константы (табл. 2.2), для обозначения которых в MuPAD зарезервированы специальные идентификаторы, также считаются символьными.

Для добавления в текст программы идентификатора одной из следующих стандартных констант: π, i, e, ∞, можно воспользоваться раскрывающейся панелью кнопок `Symbols`

на командной панели `Command Bar` (см. рис. 1.5).

Для отображения в поле вывода текущего значения некоторой переменной достаточно написать ее идентификатор в командном поле (рис. 2.3). В одном поле вывода можно отобразить значения нескольких переменных. Для этого идентификаторы этих переменных записывают в соответствующем командном поле последовательно, используя для их разделения один их следующих символов: точка с запятой (`;`), двоеточие (`:`), запятая (`,`). Если переменные записаны через запятую, то они образуют последовательность и их значения выводятся в одной и той же строке поля вывода через запятую (см. рис. 2.3). Если после идентификатора переменной стоит точка с запятой, то вывод ее значения (а также значений всех переменных, записанных перед ней через запятую) осуществляется в отдельной строке поля вывода. Если после идентификатора переменной стоит двоеточие, значение переменной (а также значений всех переменных, записанных перед ней через запятую) не выводится. После последней переменной знак препинания считается необязательным. В случае отсутствия знака препинания после последней переменной ее значение (а также значения всех переменных, записанных перед ней через запятую) будет выведено в последней строке поля вывода. Запятую после последней переменной ставить нельзя.

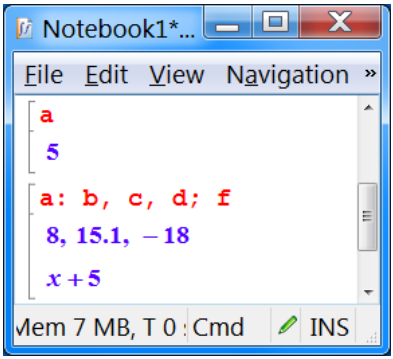


Рис. 2.3. Вывод значений переменных. Значение переменной `a` во втором поле вывода не отображается благодаря использованию двоеточия. Значения переменных `b`, `c` и `d` выведены в одной строке поля вывода, а значение переменной `f` (благодаря использованию точки с запятой) – в другой строке того же поля вывода

При необходимости при выводе перед (или между) переменными можно размещать поясняющий текст, который записывается в кавычках. Например: `"Ответ"`. Если в командном поле пояснения чередуются с идентификаторами переменных, значения которых необходимо вывести, их также необходимо отделять друг от друга символами `:`, `;` или `,` в соответствии с описанными выше правилами. Например, если в переменной `y` хранится выражение `x + 3`, то после выполнения команды `"Ответ", y` поле вывода будет содержать следующую информацию: `"Ответ", x + 3`.

2.4. Математические и логические операции

Операции – действия, которые необходимо выполнить над переменными, константами. Операции в MuPAD подразделяются на арифметические, логические, операции отношения и др. (табл. 2.3–2.5).

Операции с переменными и (или) константами записывают в строку.

Для добавления в текст программы шаблонов арифметических операций, перечисленных в табл. 2.3, можно воспользоваться раскрывающейся панелью кнопок `Basic operators` $a+b$ на командной панели `Command Bar` (см. рис. 1.5). Шаблон каждой операции содержит поля, отмеченные символом #. Например, после выбора на раскрывающейся панели кнопок `Basic operators` кнопки $a \times b$ (`Multiply`) в командном поле появится шаблон `((#a)*#b)`. Каждое поле шаблона необходимо заменить константой, переменной или выражением.

Таблица 2.3

Правила записи арифметических операций

	Математика	**MuPAD**
Сложение	$a + b$	`a + b`
Вычитание	$a - b$	`a - b`
Умножение	ab	`a*b`
Деление	$\frac{a}{b}$	`a/b`
Возведение в степень	a^b	`a^b`
Корень квадратный	$\sqrt{x}$	`sqrt(x)`

Окончание табл. 2.3

	Математика	MuPAD
Корень степени n	$\sqrt[n]{x}$	`surd(x, n)`
Модуль числа	$\|x\|$	`abs(x)`
Остаток от деления a на b		`a mod b`
Деления a на b нацело		`a div b`

Таблица 2.4

Правила записи логических операций

	Математика	MuPAD
И	$A \bigcap B$	`A and B`
Или	$A \bigcup B$	`A or B`
Не	$/A$	`not A`
Эквивалентно	$A \equiv B$	`A <=> B`
Следовательно	$A \Rightarrow B$	`A ==> B`
Принадлежит	$A \in B$	`A in B`

Таблица 2.5

Правила записи операций отношения

	Математика	MuPAD
Равно	$a = b$	`a = b`
Не равно	$a \neq b$	`a <> b`
Приближенно равно	$a \approx b$	`a ~= b`
Больше	$a > b$	`a > b`
Меньше	$a < b$	`a < b`
Больше или равно	$a \geq b$	`a >= b`
Меньше или равно	$a \leq b$	`a <= b`

2.5. Выражения, уравнения и неравенства

Под математическим ***выражением*** понимают упорядоченную совокупность переменных, констант, результатов вычислений значений функций и операций (арифметических, логических и т.д.), которые необходимо выполнить с ними. В зависимости от используемых операций выражения также называются арифметическими, логическими и т.д. соответственно.

В командном поле MuPAD выражения записываются в строку с помощью заданных ранее и незаданных символьных и числовых переменных, функций и знаков операций между ними.

Операции, входящие в выражение, выполняются последовательно с учетом установленного приоритета. Приоритет выполнения операций в MuPAD совпадает с приоритетом выполнения операций, принятым в математике. Если последовательность выполнения операций в выражении необходимо изменить, используют круглые скобки.

Например:

арифметическое выражение: `1/2 + x - sqrt(x + 2/3)`

логическое выражение: `(x > a - 3) and (x <= a + 6)`

При выполнении вычислений в выражениях автоматически приводятся подобные как для числовых, так и для символьных переменных (рис. 2.4).

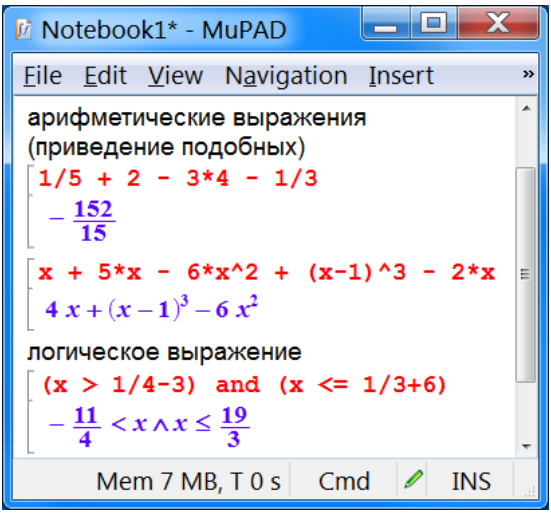


Рис. 2.4. Автоматическое преобразование выражений

Математические выражения используются при записи других математических конструкций, в частности таких, как уравнения и неравенства.

Под ***уравнением*** понимают условную запись задачи о нахождении аргументов двух функций, при которых значения этих функций равны.

Аргументы сравниваемых функций называют неизвестными, а искомые значения этих аргументов (неизвестных) – корнями уравнения.

Под ***неравенством*** понимают условную запись задачи о нахождении аргументов двух функций, при которых для значений этих функций выполняются определенные операции отношения (>, <, ≥, ≤).

Для записи функций, значения которых подлежат сравнению, используют математические выражения, между которыми указывают соответствующий знак операции отношения (=, ≠, >, <, ≥, ≤). В командном поле уравнения и неравенства также записываются в строку и могут содержать числовые значения, а также заданные и незаданные ранее символьные переменные.

Например:

уравнение: `1/2 + x = sqrt(x + 2/3)`

неравенство: `x^2 - 8 > 3 - x`

В программе отдельные выражения, уравнения, неравенства могут как использоваться самостоятельно, так и входить в состав других операторов (команд).

2.6. Присвоение значений переменным

Чтобы в некоторую переменную поместить определенную информацию, используют оператор присвоения, который записывают следующим образом:

`<Переменная> := <выражение>`

`<Переменная> := <выражение>:`

В отличие от MatLab, для присвоения значения переменной в MuPAD используется символ `:=`, слева от которого указывают идентификатор `<переменной>`. `<Выражение>` – это присваиваемое переменной значение, которое указывается справа от символа `:=` и может являться константой, переменной, выражением, уравнением, неравенством и т.д. Тип переменной определяется типом присваиваемого ей `<выражения>` (значения, уравнения, неравенства и т.д.). Например, чтобы переменной `a` присвоить числовое значение 6, в командном поле можно написать `a:= 6`; чтобы переменной `z` присвоить выражение $x^2 + 3$, можно написать `z:=x^2+3`; чтобы переменной `ur` присвоить уравнение $x + 3 = 10 \sin x$, можно соответственно написать `ur:=x+3 = 10*sin(x)`.

Конструкция вида `<переменная> := <выражение>` называется оператором (командой) присвоения.

После выполнения командного поля, содержащего присвоение, в поле вывода по умолчанию отображается не исходное `<выражение>`, а значение присвоенной переменной, но без названия этой переменной (рис. 2.5).

Если в записи выражения (уравнения, неравенства и т.д.), указанного справа от оператора присвоения, используются присвоенные ранее переменные, то их значения автоматически подставляются в выражение (уравнение, неравенство и т.д.), приводятся подобные, а результат присваивается указанной слева от символа `:=` переменной (рис. 2.6).

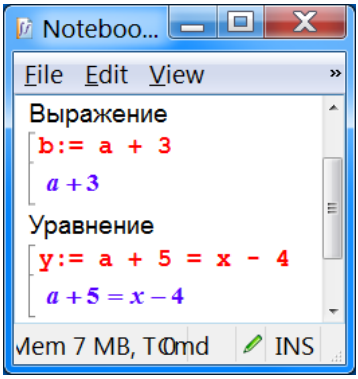


Рис. 2.5. Выполнение оператора присвоения с выводом значения переменной

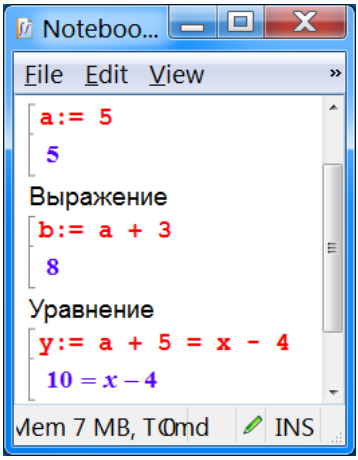


Рис. 2.6. Выполнение оператора присвоения с учетом предыдущих присвоений

Для добавления в текст программы шаблона оператора присвоения можно воспользоваться кнопкой `Assignment` $a := b$ на командной панели `Command Bar` (см. рис. 1.5). Шаблон содержит два поля, отмеченных символом #, а также символ `:=` между ними, и вставляется в командное поле в виде `#a:=#b`. Поле слева от символа `:=` необходимо заменить идентификатором присваиваемой переменной, а поле справа – или константой, или переменной, или выражением, или уравнением, или неравенством и т.д.

Когда нет необходимости выводить значение присвоенной переменной, для подавления автоматического вывода после присваиваемого выражения (значения, уравнения, неравенства и т.д.) ставят символ `:`.

В одном командном поле можно присвоить значения нескольким переменным. Для этого операторы присвоения, каждый из которых осуществляет присвоение одной переменной (`<переменная> := <выражение>`), записывают последовательно, отделяя друг от друга в соответствии с общим правилом одним из следующих символов: точка с запятой (`;`) или двоеточие (`:`). Операторы присвоения необходимо отделять друг от друга знаками препинания (`;` или `:`), даже если эти операторы расположены в разных строках одного командного поля. Если после оператора присвоения стоит точка с запятой (рис. 2.7), то вывод значения присвоенной переменной осуществляется в отдельной строке поля вывода. Если после оператора присвоения стоит двоеточие, значение присвоенной переменной не выводится. После последнего (в данном командном поле) оператора присвоения знак препинания считается необязательным. В этом случае значение последней присвоенной переменной будет выведено в последней строке поля вывода. Операторы присвоения отделять друг от друга запятой запрещается.

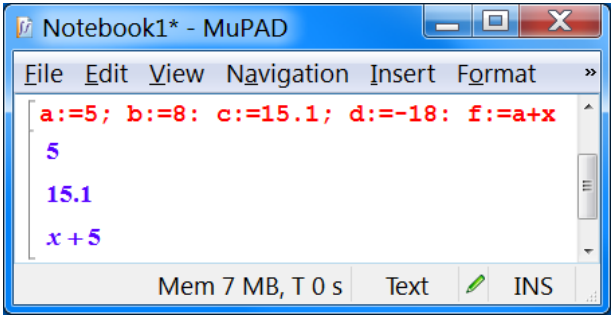


Рис. 2.7. Присвоение значений нескольких переменных в одном командном поле. Вывод присвоенных значений переменных `b` и `d` подавлен благодаря использованию двоеточия после оператора присвоения. Значения остальных переменных отображаются в отдельных строках поля вывода

Идентификатор переменной, которой присвоено числовое значение, или выражение, или уравнение, или неравенство и т.д., часто используется как условное обозначение этого выражения, уравнения, неравенства или

числа. Это позволяет организовать повторное обращение к выражению (значению, уравнению, неравенству и т.д.) через поставленную ему в соответствие переменную без необходимости дублировать текст этого выражения (значения, уравнения, неравенства и т.д.).

Выражения (уравнения, неравенства, значения) можно записывать в командном поле без присвоения. При выполнении такой команды происходит вычисление всех запрограммированных в ней доступных для выполнения операций, а в поле вывода отображаются эти выражения (уравнения, неравенства) в преобразованном виде. Например, если в командном поле написать (без присвоения) выражение `x - 6*x + 8*x`, то в поле вывода появится выражение $3x$. Однако если несколько выражений (уравнений, неравенств, значений) последовательно записаны без присвоения, то далее в программе использование (без копирования) этих выражений (уравнений, неравенств, значений) будет затруднено. Для обращения к результату последней (по времени) выполненной команды вместо идентификатора переменной можно использовать символ `%`, а для обращения к результату предпоследней выполненной команды – символ `%2` и т.д. Например, если в одном командном поле написать (без присвоения) `x+3`, в следующем командном поле можно написать `%-2*x`, тогда в результате последовательного выполнения этих команд в поле вывода первого командного поля появится выражение $x + 3$, а в поле вывода второго командного поля – выражение $3 - x$. Но при этом следует помнить, что результат выполнения команды `%-2*x` зависит от того, какой оператор был выполнен непосредственно перед ней. Поэтому если далее в программе планируется использование результатов вычислений, то НЕ РЕКОМЕНДУЕТСЯ выполнять эти вычисления без присвоения.

2.7. Функции

Функцией в программировании называют поименованную относительно самостоятельную часть программы (совокупность операторов). Обычно в виде функций оформляют логически завершенные, часто повторяющиеся фрагменты. То есть в программировании понятие функции трактуется шире, чем в математике.

В математике под функцией понимают правило, устанавливающее соответствие между элементами одного множества (аргумента) и элементами другого множества (значения функции). Если каждому значению функции ставится в соответствие элемент только из одного множества, то

такая функция называется функцией одного аргумента (переменной). Если каждому значению функции ставится в соответствие по одному элементу из двух множеств, такая функция называется функцией двух аргументов (переменных). То есть математическая функция – это последовательность действий (операторов), которые необходимо совершить, чтобы, зная значение аргумента (аргументов), получить значение функции. Эта последовательность действий может быть различной: короткой, состоящей, например, из простого математического выражения, или длинной, включающей в себя множество разнообразных вычислений.

В MuPAD используются функции, соответствующие как понятиям, принятым в программировании, так и понятиям, принятым в математике.

Таблица 2.6

Правила записи элементарных функций

	Математика	**MuPAD**
Косинус	$\cos x$	`cos(x)`
Синус	$\sin x$	`sin(x)`
Тангенс	$\mathrm{tg}\, x$	`tan(x)`
Арксинус	$\sin^{-1} x$	`arcsin(x)`
Арккосинус	$\cos^{-1} x$	`arccos(x)`
Арктангенс	$\mathrm{tg}^{-1} x$	`arctan(x)`
Косинус гиперболический	$\mathrm{ch}\, x$	`cosh(x)`
Синус гиперболический	$\mathrm{sh}\, x$	`sinh(x)`
Тангенс гиперболический	$\mathrm{tgh}\, x$	`tanh(x)`
Арксинус гиперболический	$\mathrm{sh}^{-1} x$	`arcsinh(x)`
Арккосинус гиперболический	$\mathrm{ch}^{-1} x$	`arccosh(x)`
Арктангенс гиперболический	$\mathrm{tgh}^{-1} x$	`arctanh(x)`
Экспонента	e^x	`exp(x)`
Натуральный логарифм	$\ln x$	`log(x)` или `ln(x)`
Десятичный логарифм	$\lg x$	`log10(x)`
Логарифм по основанию 2	$\log_2 x$	`log2(x)`
Знак числа		`sign(x)`
Комплексное сопряжение	x^*	`conjugate(x)`

2.7.1. Стандартные функции

MuPAD содержит большое количество встроенных (имеющих зарезервированные обозначения – идентификаторы) математических функций, начиная от элементарных, заканчивая различными специальными функциями.

Правила записи в MuPAD элементарных функций приведены в табл. 2.6. Чтобы добавить в текст программы шаблоны элементарных математических функций, перечисленные в табл. 2.6, можно воспользоваться одной из раскрывающихся панелей кнопок `Trigonometric functions` $\sin a$ ▼ или `Exponential function and logarithms` e^a ▼ на командной панели `Command Bar` (см. рис. 1.5). Например, после нажатия на раскрывающейся панели кнопок `Trigonometric functions` кнопки $\sin a$ в командном поле появится шаблон `sin(#a)`. Шаблон каждой элементарной функции содержит поле, отмеченное символом #, которое необходимо заменить константой, переменной или арифметическим выражением.

В MuPAD аргументы тригонометрических функций и результаты обратных тригонометрических функций выражаются в радианах.

Для специальных функций, которые встречаются не только в математике, но и в физике, технических науках и т.д., характерен более сложный, чем для элементарных функций, вид соответствия между аргументом (аргументами) и значением функции. Правила обращения в MuPAD к некоторым из них приведены в табл. 2.7.

Таблица 2.7

Правила записи некоторых специальных функций

Математика	**MuPAD**
Факториал	
$n!$	`n!`
Дельта-функция Дирака	
$\delta(x)$	`dirac(x)`
Дельта символ Кронекера	
$\delta_{n,m}$	`kroneckerDelta(n, m)`

Продолжение табл. 2.7

Математика	**MuPAD**
Гамма-функция	
$\Gamma(x)=\int\limits_0^\infty t^{x-1}e^{-t}dt$	`gamma(x)`
Бета-функция	
$B(x,y)=\dfrac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$	`beta(x, y)`
Биномиальные коэффициенты	
$\begin{pmatrix} n \\ k \end{pmatrix}=\dfrac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$	`binomial(n, k)`
Дигамма-функция	
$\psi(x)=\dfrac{d}{dx}\ln\Gamma(x)=\dfrac{\Gamma'(x)}{\Gamma(x)}$	`psi(x)`
Дилогарифм	
$Li_2(x)=-\int\limits_0^x \dfrac{\ln(1-t)}{t}dt$	`dilog(x)`
Модифицированная функция Бесселя первого рода	
$I_n(x)=\dfrac{(\pi/2)^n}{\sqrt{\pi}\Gamma(n+1/2)}\int\limits_0^\pi e^{x\cos t}\sin^{2n}t\,dt$	`besselI(n, x)`
Функция Бесселя первого рода	
$J_n(x)=\dfrac{(\pi/2)^n}{\sqrt{\pi}\Gamma(n+1/2)}\int\limits_0^\pi \cos(x\cos t)\sin^{2n}t\,dt$	`besselJ(n, x)`
Модифицированная функция Бесселя второго рода	
$K_n(x)=\dfrac{\pi}{2}\dfrac{I_{-n}(x)-I_n(x)}{\sin n\pi}$	`besselK(n, x)`
Функция Бесселя второго рода	
$Y_n(x)=\dfrac{J_n(x)\cos n\pi-J_{-n}(x)}{\sin n\pi}$	`besselY(n, x)`

Продолжение табл. 2.7

Математика	**MuPAD**
Функция ошибок	
$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	`erf(x)`
Дополнительная функция ошибок	
$erfc(x) = 1 - erf(x)$	`erfc(x)`
Функция Доусона	
$e^{-x^2} \int_0^x e^{t^2} dt$	`dawson(x)`
Интегральная показательная функция	
$Ei(x) = \int_{-\infty}^x \frac{e^t}{t} dt$	`Ei(x)`
Интегральный логарифм	
$li(x) = \int_0^x \frac{dt}{\ln t}$	`Li(x)`
Интегральный косинус	
$Ci(x) = -\int_x^\infty \frac{\cos t}{t} dt$	`Ci(x)`
Интегральный синус	
$Si(x) = \int_0^x \frac{\sin t}{t} dt$	`Si(x)`
С-интеграл Френеля	
$C(x) = \int_0^x \cos\left(\frac{\pi}{2} t^2\right) dt$	`fresnelC(x)`
S-интеграл Френеля	
$S(x) = \int_0^x \sin\left(\frac{\pi}{2} t^2\right) dt$	`fresnelS(x)`

Окончание табл. 2.7

Математика	**MuPAD**
Полный эллиптический интеграл первого рода	
$K(x)=\int\limits_0^{\pi/2}\frac{d\theta}{\sqrt{1-x^2\sin^2\theta}}$	`ellipticK(x)`
Эллиптический интеграл первого рода (неполный)	
$F(\varphi,x)=\int\limits_0^{\varphi}\frac{d\theta}{\sqrt{1-x^2\sin^2\theta}}$	`ellipticF(φ, x)`
Эллиптический интеграл второго рода	
$E(\varphi,x)=\int\limits_0^{\varphi}\sqrt{1-x^2\sin^2\theta}\,d\theta$	`ellipticE(φ, x)`
Эллиптический интеграл третьего рода	
$\Pi(c;\varphi,x)=\int\limits_0^{\varphi}\frac{d\theta}{(1+c^2\sin^2\theta)\sqrt{1-x^2\sin^2\theta}}$	`ellipticPi(c, φ, x)`
Многочлен Лагерра степени *n*	
$L_n(x)=\frac{e^x}{n!}\frac{d^n}{dx^n}\left(e^{-x}x^n\right)$	`laguerreL(n, x)`
Многочлен Чебышева первого рода степени *n*	
$T_n(x)=\sum\limits_{k=0}^{n/2}\binom{n}{2k}(x^2-1)^k x^{n-2k}$	`orthpoly::` `chebyshev1(n, x)`
Многочлен Чебышева второго рода степени *n*	
$U_n(x)=\sum\limits_{k=0}^{n/2}\binom{n+1}{2k+1}(x^2-1)^k x^{n-2k}$	`orthpoly::` `chebyshev2(n, x)`
Многочлен Эрмита степени *n*	
$H_n(x)=\sum\limits_{k=0}^{n/2}(-1)^k\frac{n!}{k!(n-2k)!}(2x)^{n-2k}$	`orthpoly::` `hermite(n, x)`
Многочлен Лежандра степени *n*	
$P_n(x)=\frac{1}{2^n n!}\frac{d^n}{dx^n}\left(x^2-1\right)^n$	`orthpoly::` `legendre(n, x)`

Сходные по тематике стандартные функции объединены в библиотеки. Например, библиотека `orthpoly::` содержит ортогональные полиномы, `plot::` – графические функции, `numeric::` – численные методы решения различных математических задач. В MuPAD все библиотеки доступны без подключения.

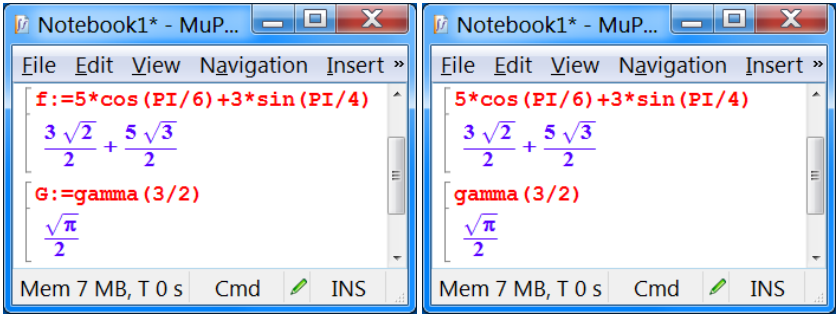
Из специальных функций, перечисленных в табл. 2.7, на командную панель `Command Bar` (см. рис. 1.5) вынесены только кнопки для добавления в программу шаблонов для вычисления факториала и биномиальных коэффициентов, которые расположены на раскрывающейся панели кнопок `Factorial and binomial coefficient` $n!$. Отмеченные символом # поля в каждом из этих шаблонов необходимо заменить константой, переменной или арифметическим выражением.

Для вычисления значения стандартной математической (элементарной, специальной) функции необходимо указать ее идентификатор и необходимое количество аргументов в соответствии с перечисленными выше правилами. Например, `gamma(3)` или `cos(PI/3)`. Если функция относится к какой-то из библиотек, то при обращении к функции название библиотеки указывается перед названием функции. Между названием функции и названием библиотеки необходимо ставить символ `::`. Например, `orthpoly::legendre(3, 0.5)` (табл. 2.7).

Следует отметить, что в поле вывода элементарные функции, значение которых не вычислено, отображаются практически так же, как принято в математике. Например, если с помощью незаданной ранее символьной переменной `x` в командном поле написать выражение `sqrt(x)+log(x)/x`, то в поле вывода это выражение будет записано в виде: $\sqrt{x}+\frac{\ln(x)}{x}$. Если вместо аргумента какой-либо специальной функции используется не числовое, а символьное выражение, то в поле вывода эта функция также отображается с помощью принятых в математике обозначений. Например, если для незаданных ранее символьных переменных `x` и `n` в командном поле написать `gamma(n), besselJ(n,x)`, то в поле вывода получится: $\Gamma(n)$, $J_n(x)$.

Стандартные математические функции можно использовать как операнды при записи выражений, уравнений, неравенств и т.д. Вычисление значений стандартных функций и выражений (уравнений, неравенств)

с их участием может осуществляться с последующим присвоением полученного значения какой-либо переменной и без присвоения (рис. 2.8). Так как с помощью символа % можно обратиться только к результату одного последнего (по времени) вычисления, НЕ РЕКОМЕНДУЕТСЯ вычислять значения стандартных математических функций и выражения, уравнения, неравенства с их участием без присвоения переменной, если далее в программе планируется использование результатов этих вычислений.



а *б*

Рис. 2.8. Вычисление значений функций
с последующим присвоением переменной (*а*) и без него (*б*)

Кроме математических (элементарных, специальных) к стандартным относят также функции, осуществляющие преобразование информации одного типа в информацию другого типа (табл. 2.8). Например, преобразование вещественных чисел в целые, рациональных в десятичные, числовой информации в строковую и т.д. Например, если переменной `z` присвоено комплексное выражение `z:=3*cos(7)+I*4*sin(9)`, то с помощью функции `real` можно получить его вещественную часть. Для этого в командном поле можно написать `real(z)`, тогда в поле вывода появится выражение: 3 cos(7), соответствующее вещественной части `z`. Аналогичным образом с помощью функции `imag` можно получить мнимую часть числа `z`.

Из функций преобразования типов, перечисленных в табл. 2.8, на командную панель `Command Bar` (см. рис. 1.5) вынесены только кнопки для добавления в программу шаблонов функции `round` для округления вещественных чисел. Эти кнопки расположены на раскрывающейся

панели кнопок `Approximate` π ≈ ⋯. Например, после нажатия на раскрывающейся панели кнопок `Approximate` на кнопку π ≈ 3.14 в командном поле появится шаблон `round(#a,2)`. Отмеченное символом # поле шаблона необходимо заменить константой, переменной или арифметическим выражением.

Таблица 2.8

Правила записи преобразования типов

	MuPAD
Округление вещественного числа до ближайшего целого	`round(x)`
Округление вещественного числа до *n* десятичных знаков после запятой	`round(x, n)`
Дробная часть вещественного числа	`frac(x)`
Преобразование числа в десятичную дробь	`float(x)` `coerce(x, Dom::Float)`
Преобразование числа в рациональную дробь	`rationalize(x, ApproximateFloats)` `coerce(x, Dom::Rational)`
Преобразование числа в целое	`coerce(x, Dom::Integer)`
Вещественная часть комплексного числа	`real(x)`
Мнимая часть комплексного числа	`imag(x)`
Округление в сторону нуля	`fix(x)`
Преобразование целого числа в его строковое представление	`int2text(x)`
Преобразование текста в целое число	`text2int(x)`
Преобразование текста в выражение	`text2expr(x)`
Преобразование выражения в текст	`expr2text(x)`
Преобразование объекта в выражение	`expr(x)`

Кроме элементарных и специальных математических функций и преобразований типов в MuPAD к стандартным также относятся функции, реализующие методы численного и аналитического решения различных математических задач. Правила использования этих функций будут подробно рассмотрены в следующих главах пособия.

Названия всех стандартных функций (и элементарных, и специальных, и преобразования типов и пр.) являются зарезервированными идентификаторами MuPAD. Использование идентификаторов стандартных функций без соблюдения правил, установленных в MuPAD, ведет к появлению сообщений об ошибках. Переопределение идентификаторов стандартных функций в программе НЕ ДОПУСКАЕТСЯ.

2.7.2. Нестандартные функции

Несмотря на многообразие стандартных функций, в программировании (и в символьных вычислениях, в частности) часто возникает необходимость использования функций, отличающихся от стандартных. Все современные языки программирования, и MuPAD в том числе, предоставляют программисту средства создания функций (отличных от стандартных) с последующим их использованием наравне со стандартными. Такие функции называются ***нестандартными***.

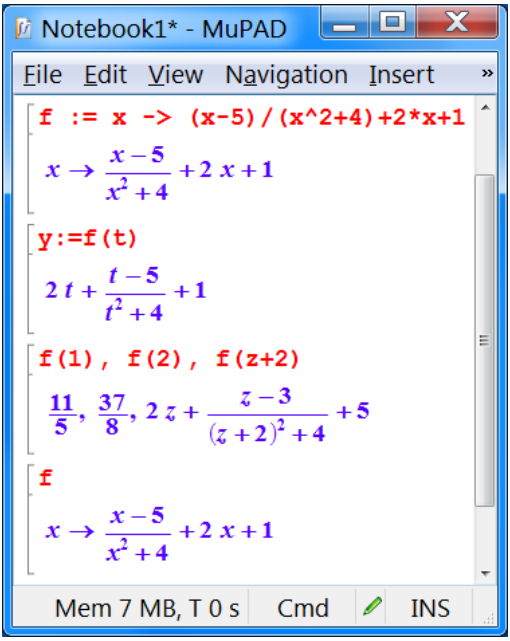


Рис. 2.9. Объявление и вычисление значений простой нестандартной функции

В отличие от стандартных нестандартные функции перед их использованием необходимо определить (объявить). В MuPAD объявить нестандартную функцию можно двумя способами. Функции, алгоритм получения значения которых из ее аргумента (аргументов) можно записать с помощью одного выражения, в MuPAD считаются простыми. Для объявления простых функций используется следующий оператор:

<Имя> **:**= (<аргумент1>, <аргумент2>, …) **->** <выражение>

Этот оператор (->) устанавливает соответствие между набором аргументов функции (<аргумент1>, <аргумент2>, …) и результатом вычисления <выражения>. В качестве <имени> указывают уникальный в пределах данной программы идентификатор, который в дальнейшем будет использоваться для обращения к данной функции. Идентификатор <имя> выступает как условное обозначение записанного после оператора присвоения соответствия между аргументами и значением функции.

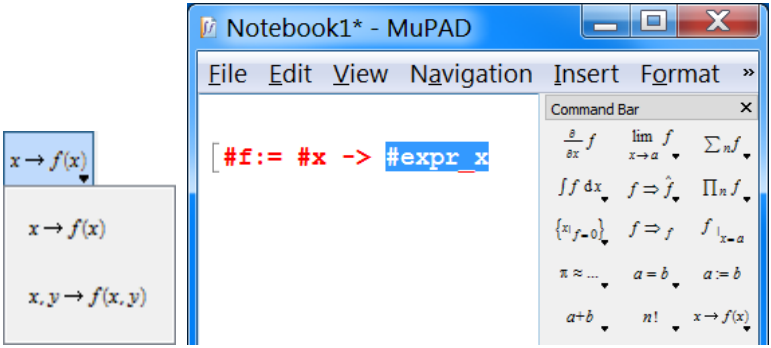
Так как способ вычисления значения функции задается одним единственным выражением, то в нем должны использоваться только константы, стандартные функции и независимые символьные переменные. Использование в <выражении> зависимых переменных при таком способе объявления функции невозможно. Поэтому все символьные переменные, использованные при записи <выражения>, считаются независимыми.

В качестве <аргументов> пред оператором -> необходимо перечислить идентификаторы всех (или некоторых) символьных переменных, использованных для записи <выражения> после оператора –>. Так же как и математическая функция, функция в MuPAD может иметь один, два или более аргументов в зависимости от количества независимых символьных переменных, использованных для записи <выражения>. Если функция имеет только один аргумент (функция одной переменной), круглые скобки между операторами **:**= и -> можно не писать. В некоторых случаях функции в MuPAD могут не иметь ни одного аргумента. В этом случае круглые скобки перед оператором -> остаются пустыми, но их наличие обязательно. Например, чтобы объявить функции $f_1(x) = x^2$ и $f_2(x, y) = 2x + 3y$, в командном поле следует написать `f1:= x -> x^2; f2:= (x,y) -> 2*x + 3*y`.

При записи <выражения> могут использоваться символьные переменные, не совпадающие ни с одним из <аргументов>.

Если при объявлении функции с помощью оператора -> после <выражения> отсутствует двоеточие, вид объявленной функции будет отображаться в поле вывода после выполнения соответствующего командного поля (рис. 2.9). При этом в <выражении>, использованном для объявления функции, все символьные переменные будут оставаться незаданными, независимо от того, были ли им ранее в программе присвоены какие-либо значения или нет.

Шаблон оператора, инициирующего объявление простой функции одного или двух аргументов, можно вставить в текст программы с помощью раскрывающейся панели кнопок `Define a function` $x \to f(x)$ на командной панели `Command Bar` (рис. 2.10). Шаблон, вставленный в командное поле с помощью кнопки $x \to f(x)$, состоит из трех полей, каждое из которых отмечено символом #, и операторов `:=` и `->` между ними и имеет вид `#f := #x -> #expr_x` (см. рис. 2.10, *б*). Каждое из полей шаблона необходимо заполнить в соответствии с описанными выше правилами. В поле `#f` необходимо указать `<имя>` функции, в поле `#x` – идентификатор `<аргумента>`, в поле `#expr_x` – `<выражение>`, в котором в качестве одного из операндов участвует идентификатор `<аргумента>` функции.



а *б*

Рис. 2.10. Раскрывающаяся панель кнопок `Define a function` (*а*) на командной панели `Command Bar`; шаблон оператора объявления функции одной переменной в программе (*б*), вставленный с помощью кнопки $x \to f(x)$

Шаблон для объявления функции двух переменных можно вставить с помощью кнопки $x, y \to f(x, y)$, расположенной на той же раскрывающейся панели кнопок (см. рис. 2.10, *а*). Оба шаблона используются аналогичным образом. Пользуясь этими шаблонами и соблюдая описанные выше правила, можно объявить в программе любую функцию нескольких переменных, задаваемую одним выражением.

Вычисление значения функции в программировании называется ее вызовом, или обращением к функции. Чтобы вычислить значение объявленной

ранее (выше) функции, необходимо указать ее имя и такое количество аргументов, которое использовалось при объявлении функции:

```
<Имя>(<аргумент1>, <аргумент2>, …)
<Переменная> := <Имя>(<аргумент1>, <аргумент2>, …)
```

Вычисление (вызов) нестандартных функций (так же как и стандартных) можно осуществлять как с присвоением вычисленного значения (выражения) какой-либо переменной, так и без присвоения (см. рис. 2.9).

Результатом вызова функции является выражение, использованное для объявления функции и преобразованное в соответствии с указанными при вызове значениями аргументов. При определенных значениях аргументов результатом вызова функции может являться числовое значение.

Аргументы функции в MuPAD служат для подстановки в указанное `<выражение>` значений символьных переменных. Если вместо аргумента функции при ее вычислении (вызове) указано числовое значение, это значение автоматически подставляется в `<выражение>`, записанное при объявлении функции, вместо той символьной переменной, на месте которой его расположили при вызове функции. Если на место одного или нескольких аргументов при вычислении функции подставить символьную переменную или выражение, соответствующая символьная переменная в выражении, записанном при объявлении функции, также автоматически заменяется этой символьной переменной или выражением. Если при объявлении функции `<выражение>` содержало кроме `<аргументов>` другие символьные переменные, то при вызове функции в это `<выражение>` подставляются не только указанные значения аргументов, но и текущие значения остальных заданных ранее символьных переменных. В полученном после подстановки выражении автоматически приводятся подобные. Если после вызова функции отсутствует двоеточие, результат этих преобразований отображается в поле вывода и, если это указано в командном поле, присваивается переменной, расположенной слева от оператора `:=`. Примеры вызова объявленной ранее функции с числовыми и символьными аргументами приведены на рис. 2.9.

Какие бы значения (числовые или символьные) не подставлялись в функцию в качестве ее аргументов при ее вызове, исходный вид установленной при объявлении функции зависимости не изменяется. В этом можно убедиться в любом месте программы (после объявления функции), если в командном поле написать `<имя>` функции без аргументов и нажать `Enter` (или кнопку `Evaluate` на стандартной панели инстру-

ментов). В результате в поле вывода вновь появится введенная ранее зависимость, связывающая <аргументы> с записанным при объявлении функции <выражением> (см. рис. 2.9). Благодаря этому свойству функции в программе можно многократно вычислять ее значения для различных значений аргумента. При этом повторно переписывать выражение, использованное для объявления функции, нет необходимости.

Если вызвать функцию, которая и не является стандартной, и не объявлена ранее (по времени) как нестандартная, то в поле вывода вызов такой функции дублируется без изменений. При этом в аргументах вызываемой функции могут быть сделаны доступные преобразования. Например, в результате выполнения команд `reset(); Fxy(x + 3), Fxy(x + x + x)` в поле вывода можно получить: Fxy(x + 3), Fxy(3x), хотя функция `Fxy(...)` не объявлена.

Кусочно-непрерывные (или ***кусочно-заданные***) функции в MuPAD можно считать простыми, благодаря наличию оператора `piecewise`, который позволяет описать все ветви кусочно-заданной функции в одном выражении. Оператор `piecewise` формирует кусочно-заданное выражение следующим образом:

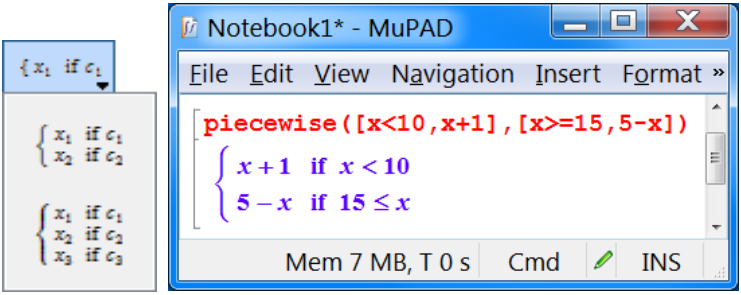
```
piecewise([<интервал1>, <выражение1>],
          [<интервал2>, <выражение2>], …)
```

В операторе `piecewise` последовательно парами через запятую перечисляются <интервалы> из области определения и соответствующие им <выражения> для вычисления значений. Каждая пара <интервал>, <выражение> должна быть заключена в квадратные скобки. Каждый <интервал> должен быть записан в виде логического выражения (или идентификатора переменной, которой это выражение ранее обозначено). Вместо одного из интервалов можно использовать стандартный идентификатор `Otherwise` (иначе), тогда соответствующее ему <выражение> будет вычисляться в тех случаях, когда все остальные логические выражения, перечисленные в операторе `piecewise`, принимают значение `False`.

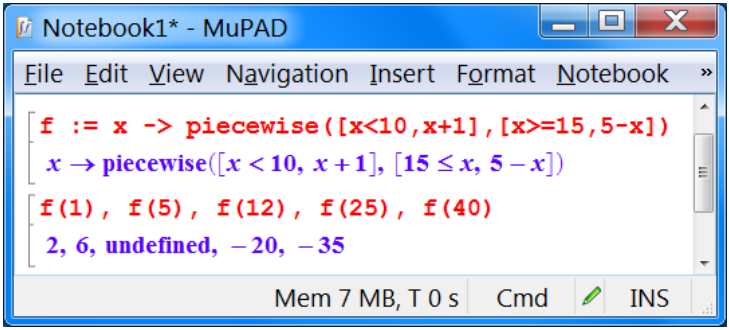
Все <интервалы> и <выражения> должны быть записаны для одной и той же независимой символьной переменной. Например:

```
piecewise([x < 10, x + 1], [x >= 15, 5 - x])
```

Если в конце оператора `piecewise` отсутствует двоеточие (`:`), то после его выполнения в поле вывода будет записано кусочно-заданное выражение (рис. 2.11, *б*).



а *б*



в

Рис. 2.11. Раскрывающаяся панель кнопок `Piecewise definition` (*а*) на командной панели `Command Bar`; использование оператора `piecewise` для записи кусочно-заданного выражения (*б*), кусочно-заданной функции (*в*)

Шаблон оператора `piecewise` можно вставить в командное поле с помощью раскрывающейся панели кнопок `Piecewise definition` на командной панели `Command Bar`. Панель кнопок `Piecewise definition` предлагает два шаблона (рис. 2.11, *а*) для создания кусочно-заданного выражения, определенного на двух или на трех отрезках области определения. Шаблон для кусочно-заданного выражения, определенного на двух отрезках, имеет вид:

```
piecewise([#cond_1, #a_1], [#cond_2, #a_2])
```

В этом шаблоне в поле #cond_1 следует записать <интервал1>, в поле #a_1 – <выражение1>, в поле #cond_2 – <интервал2>, в поле #a_2 – <выражение2>.

В символьных вычислениях редко возникает необходимость записывать кусочно-заданные выражения сами по себе. Чаще всего они используются для определения кусочно-заданных функций. Чтобы объявить кусочно-заданную простую функцию, достаточно после оператора -> вместо выражения записать оператор piecewise (рис. 2.11, *в*). В этом случае, очевидно, все интервалы и выражения оператора piecewise должны выражаться через аргумент функции, идентификатор которой расположен перед оператором ->. На рис. 2.11, *в* также представлены результаты нескольких вычислений значений кусочно-заданной функции.

Сложные функции, процесс вычисления значения которых нельзя записать с помощью одного единственного выражения, в MuPAD объявляются следующим образом:

```
<Имя> := proc(<аргумент1>, <аргумент2>, …)
begin
    <выражение или оператор 1>
    <выражение или оператор 2> …
end_proc
```

При этом способе объявления функции <имя> и <аргумент1>, <аргумент2>, … имеют тот же смысл, что и при объявлении простых функций предыдущим способом. Между служебными словами begin и end_proc записывают все операторы и (или) выражения, которые необходимо выполнить, чтобы по известным значениям аргументов функции получить ее значение. Результат последнего вычисленного между begin и end_proc выражения интерпретируется как значение функции.

Функции, объявленные этим способом, отображаются в поле вывода в виде proc <имя> (<аргумент1>, <аргумент2>, …) … end. Например, если в командном поле объявлена функция f := proc(x) begin x+sin(x) end_proc, то в поле вывода появится: proc f(x) … end.

Большинство стандартных функций в MuPAD объявлены именно этим способом. В результате выполнения некоторых команд MuPAD автоматически генерирует нестандартные функции. Эти функции всегда объявляются как сложные с помощью оператора proc.

Вычисление значений (вызов, обращение) сложной нестандартной функции, объявленной с помощью оператора `proc`, происходит так же и по тем же правилам, как и вычисление значений (вызов, обращение) простой функции, объявленной предыдущим способом (с помощью оператора `->`), т.е. в виде `<имя> (<аргумент1>, <аргумент2>, …)`. Например, если ранее в программе была объявлена функция `f := proc(x) begin x+sin(x) end_proc`, то для ее вызова можно написать следующие команды: `f(x)`, `f(PI)`, `f(PI/2)`, `f(PI+x)` и т.д.

Подробнее объявление функции с помощью оператора `proc` рассмотрено в разд. 10.8.

НЕ РЕКОМЕНДУЕТСЯ объявлять любые нестандартные функций без присвоения переменной. Это существенно затруднит их дальнейшее использование в программе.

Глава 3. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ MUPAD

Графическое представление результатов символьных вычислений и численных расчетов существенно влияет на оценку их достоверности. Наиболее часто прибегают к графическому представлению различных функциональных зависимостей, которые принято называть графиками функций. Среди графиков функций можно выделить: двумерные и трехмерные; статические и динамические (анимированные).

Для изображения в поле вывода разнообразных графических объектов используют оператор (команду) `plot`.

Графические объекты – это переменные специального типа, в которых хранится информация о некоторых изображениях, имеющая сложную структуру. Оператор `plot` может изображать как двумерные, так и трехмерные, как статические, так и динамические графические объекты. К статическим графическим объектам относятся двумерные графики функции (`plot::Function2d`), параметрические кривые (`plot::Curve2d`), наборы точек (`plot::PointList2d`), прямые линии (`plot::Line2d`), ломаные (`plot::Polygon2d`), а также аналогичные трехмерные объекты. Перед использованием оператора `plot` следует создать необходимое количество графических объектов, обозначив каждый из них уникальным в пределах данной программы идентификатором. Способы создания различных графических объектов описаны в этой главе далее.

Чтобы в поле вывода изобразить графические объекты, оператор `plot` в командном поле записывают следующим образом:

plot(<объект1>, <объект2>, …, <опция1>, <опция2>, …)

Вместо обязательных параметров <объект1>, <объект2>, … оператора `plot` необходимо обязательно указать хотя бы один идентификатор созданного ранее графического объекта. Например, для созданного ранее и обозначенного переменной `g` графического объекта (см. разд. 3.1) можно написать команду

```
plot(g)
```

Если на одном рисунке необходимо изобразить несколько графических объектов, их идентификаторы при вызове оператора `plot` перечисляют

через запятую. Например, для изображения двух созданных ранее и обозначенных переменными `f1` и `f2` графических объектов можно написать команду `plot(f1, f2)`.

Опции (`<опция1>`, `<опция2>`, …) – это необязательные параметры, которые позволяют выполнить дополнительные настройки изображения графических объектов. Опции оператора `plot` позволяют задавать (изменять) только общие для всех объектов настройки. Эти настройки должны относиться к изображению в целом, а не к отдельным объектам. При попытке в операторе `plot` указать опции, изменяющие настройки отдельного графического объекта, такие опции игнорируются. В табл. 3.1 приведены опции, чаще всего изменяемые программистами при изображении графических объектов с помощью оператора `plot`.

Таблица 3.1

Опции оператора `plot`

Название	**Описание**	**Значение по умолчанию**	**Возможные значения**
`Height`	Высота изображения	`80*unit::mm`	`8*unit::cm`
`Width`	Ширина изображения	`120*unit::mm`	`12*unit::cm`
`Header`	Заголовок	`""`	`"Название"`
`HeaderFont`	Шрифт заголовка	`["sans-serif", 8]`	`["Times New Roman", 14, Bold]`
`Background-Color`	Цвет фона изображения	`RGB::White`	RGB-цвет
`Scaling`	Масштабирование	`Unconstrained`	`Automatic, Constrained, Unconstrained`
`ViewingBox`	Видимые границы в направлениях X и Y	`[Automatic, Automatic]`	`[xmin..xmax, ymin..ymax], [Automatic, Automatic]`
`Viewing-BoxXRange`	Видимые границы в горизонтальном направлении	`Automatic.. Automatic`	`xmin..xmax`
`Viewing-BoxYRange`	Видимые границы в вертикальном направлении	`Automatic.. Automatic`	`ymin..ymax`

Продолжение табл. 3.1

Название	Описание	Значение по умолчанию	Возможные значения
CoordinateType	Тип масштаба координатных осей	LinLin	LinLin – линейные, LinLog – линейно-логарифмические, LogLin – логарифмо-линейные, LogLog – логарифмические
Axes	Тип изображения координатных осей	Automatic	None, Automatic, Boxed, Frame, Origin
AxesLineColor	Цвет координатных осей	RGB::Black	RGB-цвет
AxesLineWidth	Толщина линий координатных осей	0.18*unit::mm	0.18*unit::mm
AxesTitles	Названия координатных осей (2D)	["x", "y"]	[string, string]
AxesTitles	Названия координатных осей (3D)	["x", "y", "z"]	[string, string, string]
YAxisTitleOrientation	Направление названия вертикальной координатной оси	Horizontal	Vertical, Horizontal
AxesTips	Показывать стрелки на координатных осях?	TRUE	TRUE, FALSE
AxesVisible	Показывать оси координат?	TRUE	TRUE, FALSE
TicksNumber	Расположение чисел около меток на координатных осях	Normal	None, Low, Normal, High
TicksVisible	Показывать метки на координатных осях?	TRUE	TRUE, FALSE
XTicksVisible	Показывать метки по оси *X*?	TRUE	TRUE, FALSE

Продолжение табл. 3.1

Название	Описание	Значение по умолчанию	Возможные значения
YTicksVisible	Показывать метки по оси *Y*?	TRUE	TRUE, FALSE
TicksDistance	Расстояние между метками по координатным осям		Положительное вещественное число
XTicksDistance	Расстояние между метками по оси *X*		Положительное вещественное число
YTicksDistance	Расстояние между метками по оси *Y*		Положительное вещественное число
BorderColor	Цвет границы	RGB::Grey50	RGB-цвет
BorderWidth	Ширина границы	0	1*unit::mm
LegendVisible	Показывать легенду?	FALSE	TRUE, FALSE
LegendPlacement	Вертикальное положение легенды	Bottom	Top, Bottom
LegendAlignment	Горизонтальное положение легенды	Center	Left, Center, Right
LegendFont	Шрифт текста легенды	["sans-serif", 8]	["Arial", 12]
LineWidth	Толщина линии	0.2*unit::mm	0.2*unit::mm
LineStyle	Стиль линии	Solid	Dashed, Dotted, Solid
LineColor	Цвет линии	RGB::Blue	RGB-цвет
PointColor	Цвет точки	RGB::Blue	RGB-цвет
PointsVisible	Показывать точки?	FALSE	TRUE, FALSE
PointSize	Размер точек	1.5*unit::mm	1.5*unit::mm
PointStyle	Стиль точек	FilledCircles	Circles, Crosses, Diamonds, FilledCircles, FilledDiamonds, FilledSquares, Squares, Stars, XCrosses

Окончание табл. 3.1

Название	Описание	Значение по умолчанию	Возможные значения
GridLineColor	Цвет всех основных линий координатной сетки	RGB::Grey75	RGB-цвет
GridLineStyle	Стиль всех основных линий координатной сетки	Solid	Dashed, Dotted, Solid
GridLineWidth	Толщина всех основных линий координатной сетки	0.1*unit::mm	0.1*unit::mm
GridVisible	Показывать линии координатной сетки?	FALSE	TRUE, FALSE
VerticalAsymptotesVisible	Показывать вертикальные асимптоты?	TRUE	TRUE, FALSE
Frames	Количество кадров в анимации	50	целое число больше 0
TimeRange	Промежуток времени для анимации	0.0 .. 10.0	tmin..tmax

Все опции заданы по умолчанию. При вызове оператора plot указывают только те опции, значения которых нужно задать отличными от значений, используемых по умолчанию. Каждую опцию задают в виде: <название> = <значение>. Например, чтобы при изображении графического объекта, обозначенного ранее переменной g, на графике отобразить координатную сетку, необходимо значение опции GridVisible задать равным TRUE:

```
plot(g, GridVisible = TRUE)
```

Цвета (для опций AxesLineColor, GridLineColor, BackgroundColor, BorderColor и т.д.) необходимо задавать в формате RGB в виде набора из трех десятичных чисел от 0 до 1 каждое. Например, AxesLineColor = [0.0, 0.0, 0.0] – черный цвет осей координат. Для обозначения некоторых цветов в MuPAD выделены отдельные идентификаторы. Список часто используемых цветов и соответствующих им стандартных идентификаторов приведен в табл. 3.2.

Таблица 3.2

Стандартные идентификаторы цвета

Идентификатор	Цвет	Идентификатор	Цвет
RGB::Amethyst	Аметист	RGB::Aqua	Аква
RGB::Aquamarine	Аквамарин	RGB::Azure	Лазурный
RGB::Banana	Банан	RGB::Beige	Бежевый
RGB::Black	**Черный**	RGB::Black100	Черный 100%
RGB::Black20	Черный 20%	RGB::Black40	Черный 40%
RGB::Black60	Черный 60%	RGB::Black80	Черный 80%
RGB::Blue	**Синий**	RGB::Brass	Латунь
RGB::Brick	Кирпич	RGB::Bronze	Бронза
RGB::Brown	Коричневый	RGB::Chestnut	Каштан
RGB::Chocolate	Шоколад	RGB::Cinnamon	Корица
RGB::Cobalt	Кобальт	RGB::Copper	Медь
RGB::Coral	Коралл	RGB::Corn	Кукуруза
RGB::Crimson	Багровый	RGB::Cyan	Голубой
RGB::DarkGreen	Темно-зеленый	RGB::DarkBrown	Темно-коричневый
RGB::DarkOrange	Темно-оранжевый	RGB::DarkGray	Темно-серый
RGB::DarkRed	Темно-красный	RGB::DarkPink	Темно-розовый
RGB::DarkViolet	Темно-фиолетовый	RGB::Eggplant	Баклажаны
RGB::Fuchsia	Фуксия	RGB::Gold	Золото
RGB::Gray	**Серый**	RGB::Gray100	Серый 100%
RGB::Gray20	Серый 20%	RGB::Gray40	Серый 40%
RGB::Gray60	Серый 60%	RGB::Gray80	Серый 80%
RGB::Green	**Зеленый**	RGB::Khaki	Хаки
RGB::Lavender	Лаванда	RGB::Lemon	Лимон
RGB::LightBeige	Светло-бежевый	RGB::LightBlue	Голубой
RGB::LightGray	Светло-серый	RGB::LightGreen	Светло-зеленый
RGB::LightOrange	Светло-оранжевый	RGB::Maroon	Бордовый
RGB::Magenta	Пурпурный	RGB::Navy	Темно-синий
RGB::Mauve	Лиловый	RGB::Olive	Оливковый
RGB::Mustard	Горчица	RGB::Orchid	Орхидея

Окончание табл. 3.2

Идентификатор	Цвет	Идентификатор	Цвет
RGB::Ochre	Охра	RGB::Pear	Груша
RGB::Orange	Оранжевый	RGB::Plum	Слива
RGB::Peach	Персик	RGB::Red	**Красный**
RGB::Pink	Розовый	RGB::Salmon	Лосось
RGB::Purple	Фиолетовый	RGB::Silver	Серебро
RGB::Saffron	Шафран	RGB::Snow	Снег
RGB::Sapphire	Сапфир	RGB::Tomato	Томатный
RGB::Terracotta	Терракотовый	RGB::Ultramarine	Ультрамарин
RGB::Turquoise	Бирюзовый	RGB::White	**Белый**
RGB::Violet	Фиолетовый	RGB::Yellow	**Желтый**

Использование стандартного идентификатора RGB::random() позволяет получать случайные цвета в формате RGB.

Шаблон оператора plot можно вставить в командное поле с помощью кнопки 2D graphics and animations на командной панели Command Bar. Вставленный шаблон имеет вид plot(#f) и содержит одно поле, обозначенное #f. Вместо поля #f необходимо указать хотя бы один идентификатор созданного ранее графического <объекта> в соответствии с описанными выше правилами. Остальные параметры добавляются оператору plot по необходимости.

3.1. График функции одной переменной

Построение графика функции одной переменной – одна из наиболее распространенных задач статической графики.

Для построения графика необходимо:

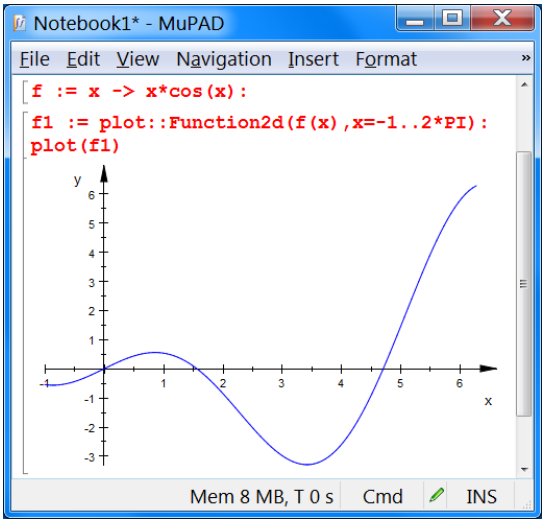
1) создать графический объект типа «двумерный график»;

2) визуализировать графический объект с помощью оператора plot.

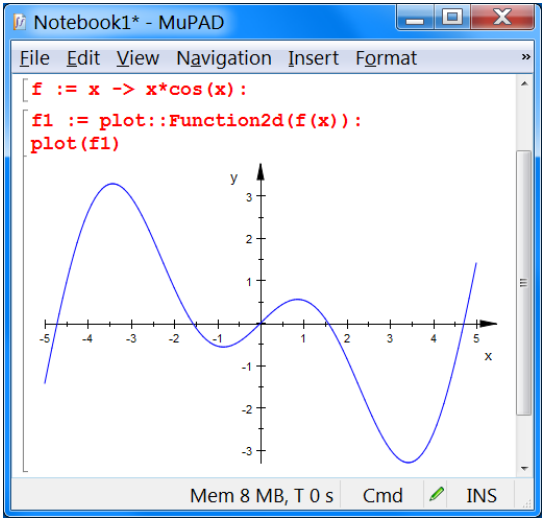
Чтобы создать объект типа «двумерный график», используют стандартную функцию plot::Function2d, которая вызывается следующим образом:

<Имя> := **plot::Function2d**(<функция(аргумент)>,
<аргумент> = <начало>..<конец>, <опция1>, <опция2>, …)

Слева от оператора присвоения необходимо указать идентификатор <имя>, которым в дальнейшем в программе будет обозначен созданный графический объект.



а



б

Рис. 3.1. Изображение (с помощью оператора `plot`) в поле вывода графика функции на разных интервалах: *а* – на интервале [–1, 2π]; *б* – интервал [–5, 5] не указан и считается заданным по умолчанию

Для функции `plot::Function2d` только один параметр является обязательным. Это параметр <функция(аргумент)>. Остальные параметры добавляются по необходимости.

В качестве параметра <функция(аргумент)> при создании графического объекта типа «двумерный график» можно использовать любую математическую функцию одной переменной, объявленную ранее (стандартную или нестандартную) и вызванную с незаданным ранее символьным аргументом, а также выражение, содержащее одну незаданную ранее символьную переменную. Например:

```
f := x -> x*cos(x);
f1 := plot::Function2d(f(x));
f2 := plot::Function2d(sin(x));
f3 := plot::Function2d(5*sin(x)+6*cos(x/2));
```

Комбинация параметров <аргумент> = <начало>..<конец> позволяет выбрать, на каком интервале необходимо строить график указанной функции. Здесь в качестве <аргумента> следует использовать тот же идентификатор, который указан при записи параметра <функция (аргумент)>. <Начало> и <конец> интервала – это числовые константы или выражения, для записи которых использованы только идентификаторы, которым ранее присвоены известные числовые значения. Если при создании графического объекта типа «двумерный график» комбинация параметров <аргумент> = <начало>..<конец> отсутствует, график будет построен на интервале [–5, 5].

Таблица 3.3

Основные опции функций
`plot::Function2d` и `plot::Curve2d`

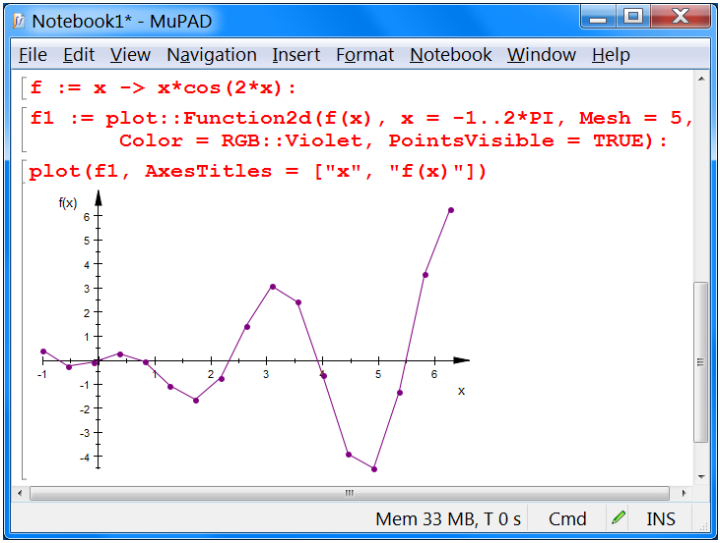
Название	Описание	Значение по умолчанию
AntiAliased	Сглаживать линии?	TRUE
Background-Color	Цвет фона	RGB::White
Color	**Основной цвет**	RGB::Blue
Frames	Количество кадров в анимации	50
LegendText	Текст для легенды	""
LegendEntry	Добавлять описание этого объекта в легенду	TRUE
LineColor	Цвет линий	RGB::Blue
LineWidth	Толщина линий	0.35*unit::mm

Окончание табл. 3.3

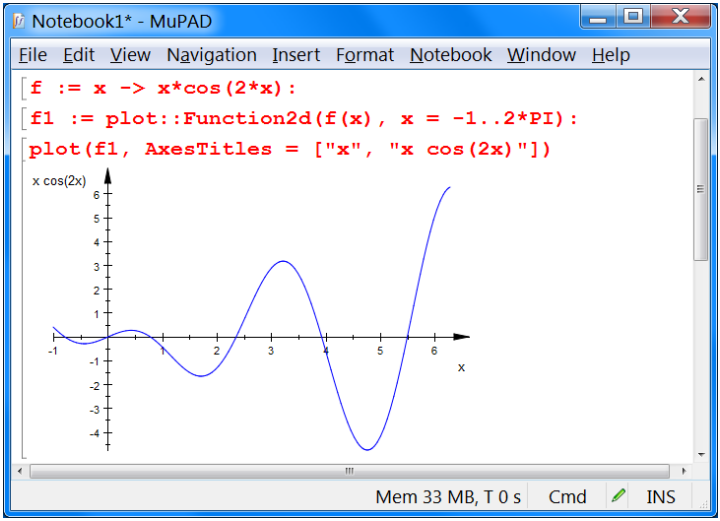
Название	Описание	Значение по умолчанию
LineStyle	Стиль линий (Solid – сплошная, Dashed – штрихами, Dotted – пунктиром)	Solid
LinesVisible	Видимые линии?	TRUE
Mesh	**Количество точек**	121
PointStyle	Стиль точек (○ Circles, ● FilledCircles, + Crosses, × XCrosses, ◊ Diamonds, ♦ FilledDiamonds, □ Squares, ■ FilledSquares, ✳ Stars)	FilledCircles
PointSize	Размер точек	1.5*unit::mm
PointsVisible	**Видимые точки?**	FALSE
TimeRange	Промежуток времени для анимации	0.0 .. 10.0
VerticalAsymptotesColor	Цвет вертикальных асимптот	RGB::Grey50
VerticalAsymptotesWidth	Толщина линии вертикальной асимптоты	0.2*unit::mm
VerticalAsymptotesStyle	Стиль линии вертикальных асимптот (Solid – сплошная, Dashed – штрихами, Dotted – пунктиром)	Dashed
VerticalAsymptotesVisible	Показывать вертикальные асимптоты?	TRUE
Visible	Видимый?	TRUE
XRange	Диапазон аргумента "x" для Function2d	-5 .. 5
URange	Диапазон аргумента "u" для Curve2d	-5 .. 5

Например, на рис. 3.1, *а* объект f1 определен как двумерный график функции $f(x)$ на интервале $x \in [-1, 2\pi]$, а на рис. 3.1, *б* для этой же функции интервал построения графика не указан. В обоих случаях объект изображается в поле вывода с помощью оператора plot.

По умолчанию функция plot::Function2d использует 121 точку для вычисления значений заданной математической функции и соединяет точки с рассчитанными координатами сплошными линиями синего цвета. При этом сами точки на графике не изображаются (не выделяются).



а



б

Рис. 3.2. Настройка двумерного графика функции типа `Function2d`: *а* – с использование опций `Color`, `Mesh`, `PointsVisible`; *б* – без опций

При создании графического объекта типа «двумерный график» <опции> позволяют изменить эти и другие настройки, заданные по умолчанию. При вызове функции plot::Function2d указывают только те опции, значения которых нужно задать отличными от значений, используемых по умолчанию. Каждую опцию задают в виде: <название> = <значение>. В табл. 3.3 приведены список опций функции plot::Function2d и их значения по умолчанию.

Цвета (для опций Color, BackgroundColor, LineColor, VerticalAsymptotesColor), так же как и для оператора plot, необходимо задавать в формате RGB в виде набора из трех десятичных чисел от 0 до 1 каждое или используя стандартные идентификаторы (см. табл. 3.2). Например:

Color = [1.0, 0.0, 0.0] – красный цвет графика

или Color = RGB::Red – тоже красный цвет графика.

Опция PointsVisible позволяет управлять (TRUE, FALSE) отображением расчетных точек, используемых для построения графика, а опция Mesh – их количеством (рис. 3.2)

Шаблон вызова функции plot::Function2d можно вставить в командное поле с помощью пункта Function Plots → 2D Function из раскрывающегося списка Plot Commands на командной панели Command Bar. Этот шаблон имеет вид:

plot::Function2d(#f, #x=#a..#b)

и содержит типичный для данной функции набор полей, отмеченных символом #. Эти поля необходимо заполнить следующим образом: в поле #f указать <функцию(аргумент)>, в поле #x – <аргумент>, в полях #a и #b – <начало> и <конец> интервала в соответствии с описанными выше правилами. Остальные параметры добавляются функции plot::Function2d по необходимости.

3.2. График параметрически заданной функции

Для построения графика параметрически заданной функции необходимо:

1) создать графический объект типа «кривая»;

2) визуализировать графический объект с помощью оператора plot.

Чтобы создать объект типа «кривая», используют стандартную функцию plot:: Curve2d, которая вызывается следующим образом:

```
<Имя> := plot::Curve2d([<X-функция(аргумент)>,
                        <Y-функция(аргумент)>],
                    <аргумент> = <начало>..<конец>,
                                <опция1>, <опция2>, …)
```

Слева от оператора присвоения указывают идентификатор <имя>, которым в дальнейшем в программе будет обозначен созданный графический объект.

Для функции plot::Curve2d обязательными являются два параметра. Это объявленные ранее или стандартные функции <X-функция(аргумент)> и <Y-функция(аргумент)>, описывающие зависимости *x*- и *y*-координат точек графика от одного и того же символьного параметра (аргумента), например от времени, соответственно. Функции перечисляются через запятую и заключаются в квадратные скобки. Остальные параметры добавляются по необходимости.

Комбинация параметров <аргумент> = <начало>..<конец> позволяет указать, на каком интервале необходимо изменять переменную, являющуюся аргументом (параметром) обоих функций. В качестве <аргумента> следует записать идентификатор этой переменной. <Начало> и <конец> интервала – это числовые константы или выражения, для записи которых использованы только идентификаторы, которым ранее присвоены известные числовые значения. Если при создании графического объекта типа «кривая» комбинация параметров <аргумент> = <начало>..<конец> отсутствует, при построении графика <аргумент> будет изменяться на интервале [–5, 5].

По умолчанию функция plot::Curve2d также использует 121 точку для вычисления значений заданных математических функций и соединяет точки с рассчитанными координатами сплошными линиями синего цвета. При этом сами точки на графике не изображаются (не выделяются).

При создании графического объекта типа «кривая» <опции> позволяют изменить эти и другие настройки, заданные по умолчанию. При вызове функции plot::Curve2d указывают только те <опции>, значения которых нужно задать отличными от значений, используемых по умолчанию. Каждую <опцию> задают в виде: <название> = <значение>. В табл. 3.3 приведены список опций функции plot::Curve2d и их

значения по умолчанию. Большинство опций функции `plot::Curve2d` по смыслу настроек и способам назначения значений аналогичны опциям функции `plot::Function2d` (см. пояснения в разд. 3.1). Пример графика параметрической кривой приведен на рис. 3.3.

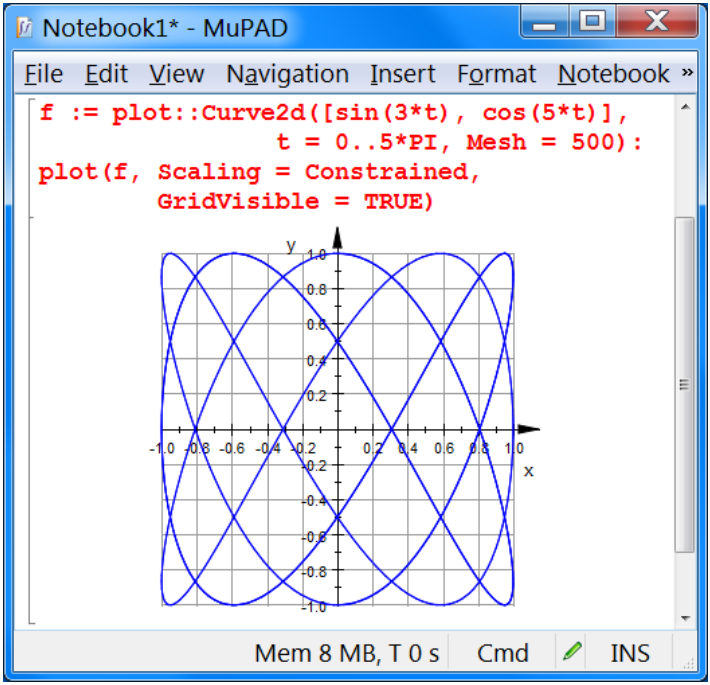


Рис. 3.3. График параметрически заданной функции

Шаблон вызова функции `plot::Curve2d` можно вставить в командное поле с помощью пункта `Curve, Planes, (Sweep) Surface` → `2D Curve` из раскрывающегося списка `Plot Commands` на командной панели `Command Bar`. Этот шаблон имеет вид:

```
plot::Curve2d (#f, #x=#a..#b)
```

и содержит типичный для данной функции набор полей, отмеченных символом `#`. Эти поля необходимо заполнить: в поле `#f` указать [`<X-функция(аргумент)>`, `<Y-функция(аргумент)>`], в поле `#x` – `<аргумент>`, в полях `#a` и `#b` – `<начало>` и `<конец>` интервала

в соответствии с описанными выше правилами. Остальные параметры добавляются функции `plot::Curve2d` по необходимости.

3.3. Графическое представление дискретного множества точек

Для графического изображения дискретного множества точек в MuPAD существует три типа объектов: линия; ломаная; набор точек. Для визуализации любого из них используется оператор `plot`. Однако предварительно необходимо создать объект соответствующего типа.

Чтобы создать объект типа «линия», используют стандартную функцию `plot:: Line2d`, которую вызывают следующим образом:

```
<Имя> := plot::Line2d(<точка1>, <точка2>,
                      <опция1>, <опция2>, …)
```

Параметр `<точка1>` – это координаты одного из концов отрезка прямой линии, которые задаются числами или арифметическими выражениями в виде `[x, y]`. Аналогично параметр `<точка2>` – это координаты другого конца отрезка прямой линии, которые тоже задаются числами или арифметическими выражениями в виде `[x, y]`. Эти две точки автоматически соединяются прямой того цвета, который установлен с помощью опции `Color`.

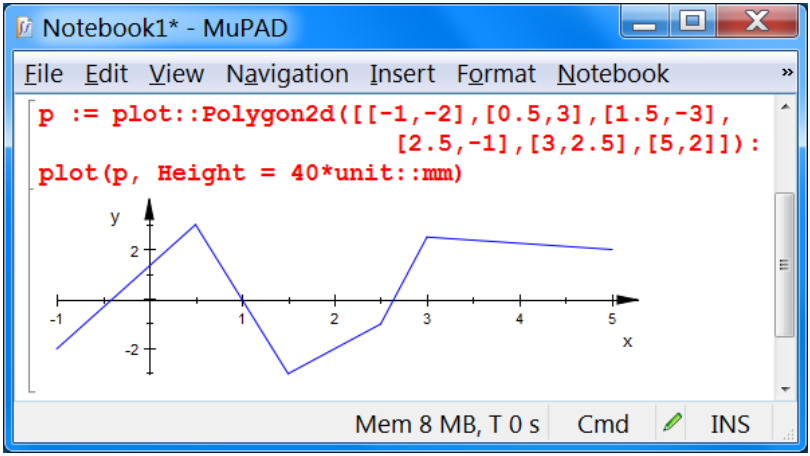
Чтобы создать объект типа «ломаная», используют стандартную функцию `plot::Polygon2d`, которую вызывают следующим образом:

```
<Имя> := plot::Polygon2d([<точка1>, <точка2>, …],
                         <опция1>, <опция2>, …)
```

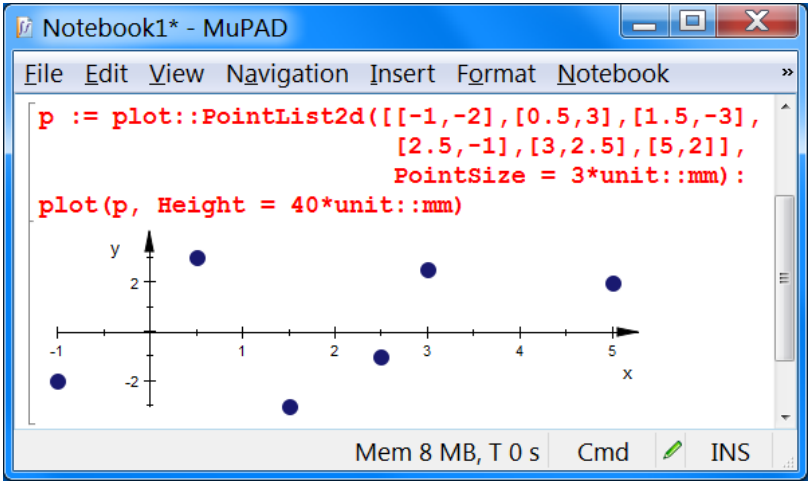
Параметры `[<точка1>, <точка2>, …]` – это последовательность координат узлов ломаной; координаты каждого из узлов задаются числами или арифметическими выражениями в виде `[x, y]`. Эти `<точки>` (узлы) автоматически соединяются отрезками прямой того цвета, который установлен с помощью опции `Color`. Если опция `Closed` задана равной `TRUE`, то последний узел соединяется с первым.

Чтобы создать объект типа «набор точек», используют стандартную функцию `plot::PointList2d`, которую вызывают следующим образом:

```
<Имя> := plot::PointList2d([<точка1>, <точка2>, …],
                           <опция1>, <опция2>, …)
```



а



б

Рис. 3.4. Изображение в поле вывода: *а* – ломаной; *б* – дискретного набора точек

Параметры `[<точка1>, <точка2>, …]` – это последовательность координат точек; координаты каждой точки задаются числами или арифметическими выражениями в виде `[x, y]`. Эти `<точки>` между собой НЕ СОЕДИНЯЮТСЯ.

При создании каждого из перечисленных выше графических объектов слева от оператора присвоения указывают идентификатор <имя>, которым в дальнейшем в программе будет обозначен созданный графический объект.

Функции plot::Line2d, plot::Polygon2d, plot::PointList2d имеют много одинаковых <опций>, которые позволяют изменить настройки, заданные по умолчанию. При вызове каждой из функций указывают только те опции, значения которых нужно задать отличными от значений, используемых по умолчанию. Как и при создании других графических объектов, каждую опцию задают в виде: <название> = <значение>. В табл. 3.4 приведены список опций и их значения по умолчанию. В этой таблице также указано, при настройке свойств каких графических объектов эти опции есть возможность использовать. Примеры построения ломаной и дискретного набора точек приведены на рис. 3.4.

Таблица 3.4

Основные опции функций plot::Line2d, plot::Polygon2d, plot::PointList2d

Название	Описание	Значение по умолчанию	Функция
Color	**Основной цвет**	RGB::Blue RGB::Blue RGB::MidnightBlue	Line2d Polygon2d PointList2d
Closed	Замкнутый?	FALSE	Polygon2d
LegendText	Текст для легенды	""	Line2d Polygon2d PointList2d
Legend-Entry	Добавлять описание этого объекта в легенду?	FALSE	Line2d Polygon2d PointList2d
LineColor	Цвет линий	RGB::Blue	Line2d Polygon2d
LineWidth	Толщина линий	0.35*unit::mm	Line2d Polygon2d
Line-Visible	Видимые линии?	TRUE	Polygon2d
LineStyle	Стиль линий (Solid – сплошная, Dashed – штрихами, Dotted – пунктиром)	Solid	Line2d Polygon2d

Окончание табл. 3.4

Название	Описание	Значение по умолчанию	Функция
Extension	Продление линии (Finite – ограниченная, Infinite – неограниченная, SemiInfinite – на половину неограниченная)	Finite	Line2d
Filled	Заполненный?	FALSE	Polygon2d
FillColor	Цвет заливки	RGB::Red	Polygon2d
PointStyle	Стиль точек (○ Circles, ● FilledCircles, + Crosses, × XCrosses, ◊ Diamonds, ♦ FilledDiamonds, □ Squares, ■ FilledSquares, ✳ Stars)	FilledCircles	Polygon2d PointList2d
PointSize	Размер точек	1.5*unit::mm	Polygon2d PointList2d
Points-Visible	Видимые точки?	FALSE	Polygon2d
TimeRange	Промежуток времени для анимации	0.0 .. 10.0	Line2d Polygon2d PointList2d
Frames	Количество кадров в анимации	50	Line2d Polygon2d PointList2d
From	Начальная точка	[0, 0]	Line2d
To	Конечная точка	[1, 0]	Line2d
Visible	Видимый?	TRUE	Line2d Polygon2d PointList2d

Шаблоны функций plot::Line2d, plot::Polygon2d можно вставить в командное поле с помощью пункта Points, Lines, Polygons → 2D Line (2D Polygon), а шаблон функций plot::PointList2d –

с помощью пункта `Statistical Utility Plots` → `2D Point List` из раскрывающегося списка `Plot Commands` на командной панели `Command Bar`. Эти шаблоны имеют вид:

```
plot::Line2d(#)
plot::Polygon2d(#)
plot::PointList2d(#)
```

и содержат по одному обязательному полю, отмеченному символом #. Вместо этого поля необходимо указать набор точек в соответствии с описанными выше правилами. Опции добавляются по необходимости.

Кроме перечисленных в разд. 3.1–3.3 объектов расположенный на командной панели `Command Bar` раскрывающийся список `Plot Commands` позволяет вставить шаблоны для создания других двумерных графических объектов самых разнообразных типов. В том числе – различных геометрических фигур (дуг, окружностей, прямоугольников, стрелок и т.д.). С правилами вызова соответствующих функций, используемых для создания этих графических объектов, можно ознакомиться в справочной системе MuPAD.

3.4. График нескольких функций одной переменной

Для отображения на графике нескольких функций, зависящих от одного и того же аргумента, необходимо создать соответствующее количество графических объектов, а затем все эти объекты изобразить в поле вывода с помощью одного оператора `plot`. При этом графические объекты могут иметь различные параметры и (или) разные типы. Например, один объект типа «двумерный график» будет изображать функцию $f_1(x) = -x^2 + x + 4$ на интервале [–3, 5], а другой объект того же типа – функцию $f_2(x) = 7\cos(2x^2)$ на интервале [–4, 4] (рис. 3.5).

На рис. 3.6 показан пример одновременного изображения на одном графике ломаной, построенной по точкам, и двумерного графика функции $3\sin x$.

Стандартная функция `plot::Hatch` позволяет создать графический объект, изображающий заштрихованную область между двумя кривыми. Вызов этой функции осуществляется следующим образом:

```
<Имя> := plot::Hatch(<график1>, <график2>,
            <начало>..<конец>, <опция1>, <опция2>, …)
```

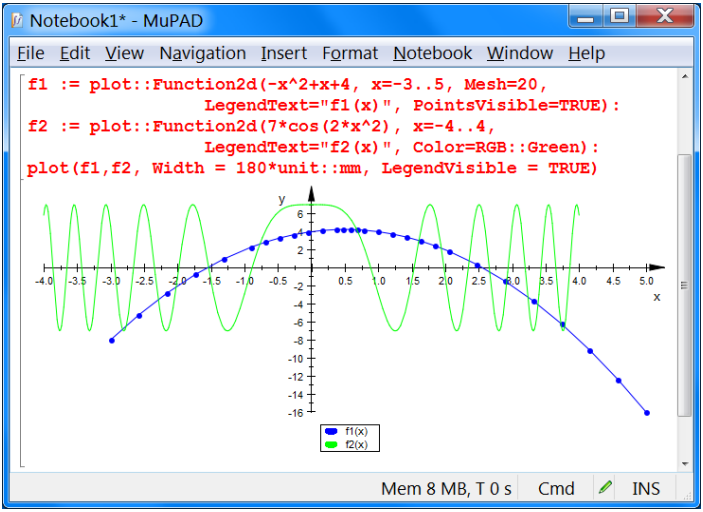


Рис. 3.5. Пример графиков двух функций, заданных на разных интервалах

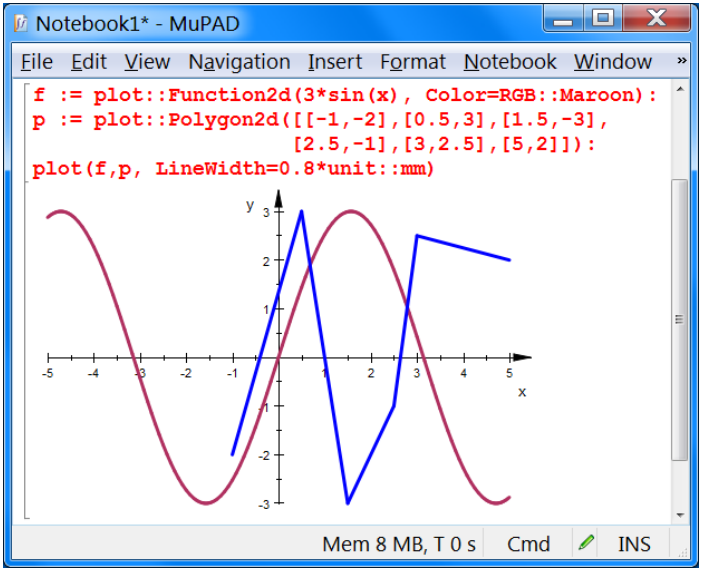


Рис. 3.6. Изображение функции и ломаной на одном графике

Слева от оператора присвоения указывают идентификатор <имя>, которым в дальнейшем в программе будет обозначен созданный графический объект.

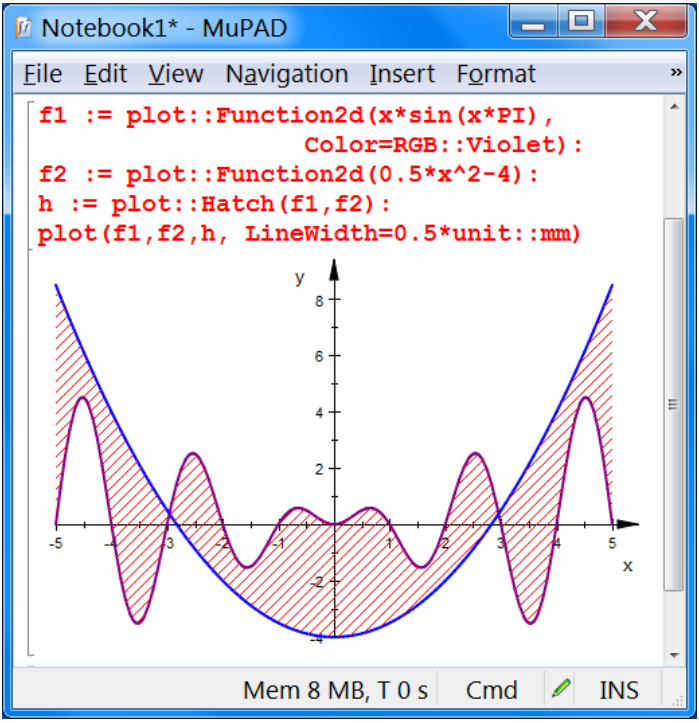


Рис. 3.7. Графики двух функций и заштрихованная область между ними

Для функции `plot::Hatch` обязательным является только параметр <график1>. Это объявленный ранее графический объект типа «двумерный график» (`plot::Function2d`). Второй параметр <график2> задает базовую линию и также должен являться объявленным ранее графическим объектом того же типа или числовой константой. Заштрихованная область будет располагаться между указанными линиями. Если второй <график> задан в виде числовой константы, заштрихованная область строится между <графиком1> и горизонтальной прямой, *y*-координата которой равна заданной константе. Если параметр <график2> не задан,

то заштрихованная область строится между <графиком1> и координатной осью *OX*.

Комбинация параметров <начало>..<конец> позволяет указать, в каких пределах по оси *OX* будет располагаться заштрихованная область. <Начало> и <конец> отрезка – это числовые константы или выражения, для записи которых использованы только идентификаторы, которым ранее присвоены известные числовые значения. Если при создании графического объекта типа «заштрихованная область» комбинация параметров <начало>..<конец> отсутствует, область строится на том же интервале, на котором построен <график1>.

Таблица 3.5

Основные опции функции `plot::Hatch`

Название	Описание	Значение по умолчанию
AntiAliased	Сглаживать линии?	TRUE
Color	**Основной цвет**	RGB::Red
FillColor	**Цвет заливки области**	RGB::Red
FillPattern	Тип заливки (CrossedLines, DiagonalLines, FDiagonalLines, HorizontalLines, Solid, VerticalLines, XCrossedLines)	DiagonalLines
Frames	Количество кадров в анимации	50
LegendText	Текст для легенды	""
LegendEntry	Добавлять описание этого объекта в легенду?	FALSE
TimeRange	Промежуток времени для анимации	0.0 .. 10.0
Visible	Видимый?	TRUE
XRange	Диапазон параметра "*x*"	-infinity .. infinity

По умолчанию заштрихованная область plot::Hatch заполняется штриховкой красного цвета без контура. Чтобы изобразить контур заштрихованной области, необходимо с помощью одного и того же оператора plot визуализировать не менее двух созданных на основе одной и той же функции объектов: plot::Hatch и plot::Function2d.

Как и для других графических объектов, при создании графического объекта типа «заштрихованная область» <опции> позволяют изменить настройки, заданные по умолчанию. При вызове функции plot::Hatch

указывают только те опции, значения которых нужно задать отличными от значений, используемых по умолчанию. Как и для других графических объектов, каждую опцию записывают в виде: <название> = <значение>. Пример построения заштрихованной области приведен на рис. 3.7. В табл. 3.5 приведены список опций функции plot::Hatch и их значения по умолчанию.

Шаблон вызова функции plot::Hatch можно вставить в командное поле с помощью пункта Hatches and Integrals → 2D Hatched Area из раскрывающегося списка Plot Commands на командной панели Command Bar. Этот шаблон имеет вид plot::Hatch(#f) и содержит минимальный для данной функции набор полей, отмеченных символом #. Вместо поля #f необходимо указать идентификатор графического объекта <график1> в соответствии с описанными выше правилами. Остальные параметры добавляются в шаблон функции plot::Hatch по необходимости.

3.5. График функции двух переменных

Все перечисленные выше функции, предназначенные для создания двумерных графических объектов, кроме plot::Hatch, имеют аналоги в трехмерном пространстве: Function3d, Curve3d, Line3d, Polygon3d, PointList3d и т.д.

Принципы использования этих функций для создания соответствующих трехмерных графических объектов аналогичны принципам создания двумерных графических объектов, подробно рассмотренным ранее. Для функций Line3d, Polygon3d, PointList3d отличие состоит в том, что все точки задаются не двумя, а тремя координатами; параметрическая кривая Curve3d – не двумя, а тремя зависимостями координат от общего параметра; для Function3d математическая <функция(…)>, для которой создается графический объект, должна зависеть не от одной, а от двух незаданных ранее символьных переменных. Функции создания трехмерных графических объектов имеют практически такой же набор опций, как и соответствующие функции создания двумерных объектов.

Для визуализации трехмерных графических объектов используется тот же оператор plot, но с обязательным указанием атрибута #3D. Если этот атрибут не указан, то вместо трехмерного изображения можно получить анимированное.

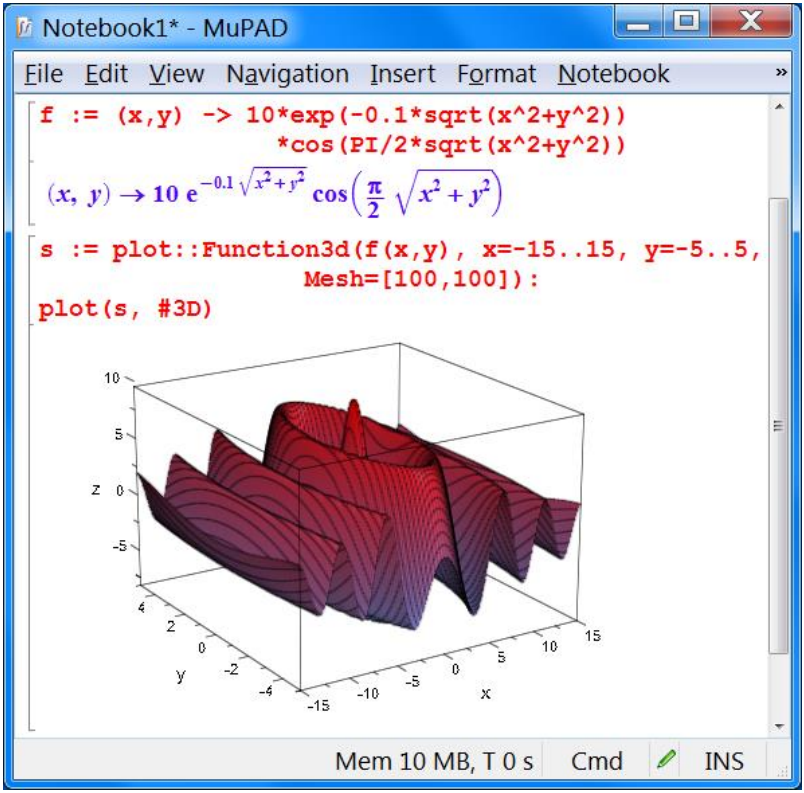


Рис. 3.8. Пример построения графика трехмерной поверхности в MuPAD

Шаблоны вызова всех функций, предназначенных для создания трехмерных объектов, можно найти в раскрывающемся списке `Plot Commands` на командной панели `Command Bar`, а шаблон оператора `plot` с атрибутом `#3D` можно вставить с помощью кнопки `3D graphics and animations` , расположенной на той же панели.

На рис. 3.8 приведен пример построения трехмерного графика функции $f(x,y) = 10e^{-0{,}1\sqrt{x^2+y^2}} \cos\left(\frac{\pi}{2}\sqrt{x^2+y^2}\right)$ для $x \in [-15, 15]$, $y \in [-5, 5]$.

Здесь с помощью опции `Mesh = [100, 100]` задано, что при создании графического объекта `s` необходимо рассчитать по 100 значений каждого из аргументов функции $f(x,y)$ и для каждой пары аргументов вычислить значение функции $f(x,y)$.

3.6. Анимация

Примером динамической графики является компьютерная анимация – последовательность сменяющих друг друга статических кадров, изображение на каждом из которых немного отличается от предыдущего. Последовательное воспроизведение таких кадров создает иллюзию движения (анимацию).

Для создания последовательности кадров в MuPAD используют те же функции, что и для создания статических графических объектов: `Function2d`, `Curve2d`, `Line2d`, `Polygon2d`, `PointList2d` и т.д. Каждая из этих функций имеет опцию `Frame`, которая указывает, сколько кадров будет создано для анимации, а опция `TimeRange` задает длительность анимации. Такие же опции есть и у оператора `plot`.

Каждая из функций, создающих графические объекты, начинает генерировать анимационные кадры только в том случае, если ее обязательный параметр (параметры) содержит больше незаданных ранее символьных переменных, чем необходимо для создания статического изображения. Например, если при создании графического объекта с помощью функции `Function2d` использовать функцию $f(x, \varphi)$ не одной, а двух переменных (рис. 3.9), то вторая переменная будет использоваться как изменяемый параметр для создания анимационных кадров. То есть для одного кадра будут вычислены значения функции $f(x, \varphi)$ при фиксированном значении φ и разных значениях x.

Для каждого последующего кадра будет выбрано новое значение φ из указанного диапазона и вновь вычислены значения функции $f(x, \varphi)$ при разных значениях x. Поэтому дополнительный параметр, вводимый в функцию для анимации, должен обеспечивать плавное постепенное изменение вида функции от кадра к кадру. В рассмотренном на рис. 3.9 примере дополнительный параметр φ функции $f(x,\varphi) = \cos\varphi \cdot \sin(0{,}5x - \varphi)$ изменяет амплитуду и начальную фазу синуса. В результате смены кадров с изображениями графиков такой функции, построенных для разных значений φ на одном и том же интервале по x, будет возникать иллюзия смещения функции синуса вдоль оси OX (за счет изменения

начальной фазы) при периодическом уменьшении (увеличении) амплитуды колебаний.

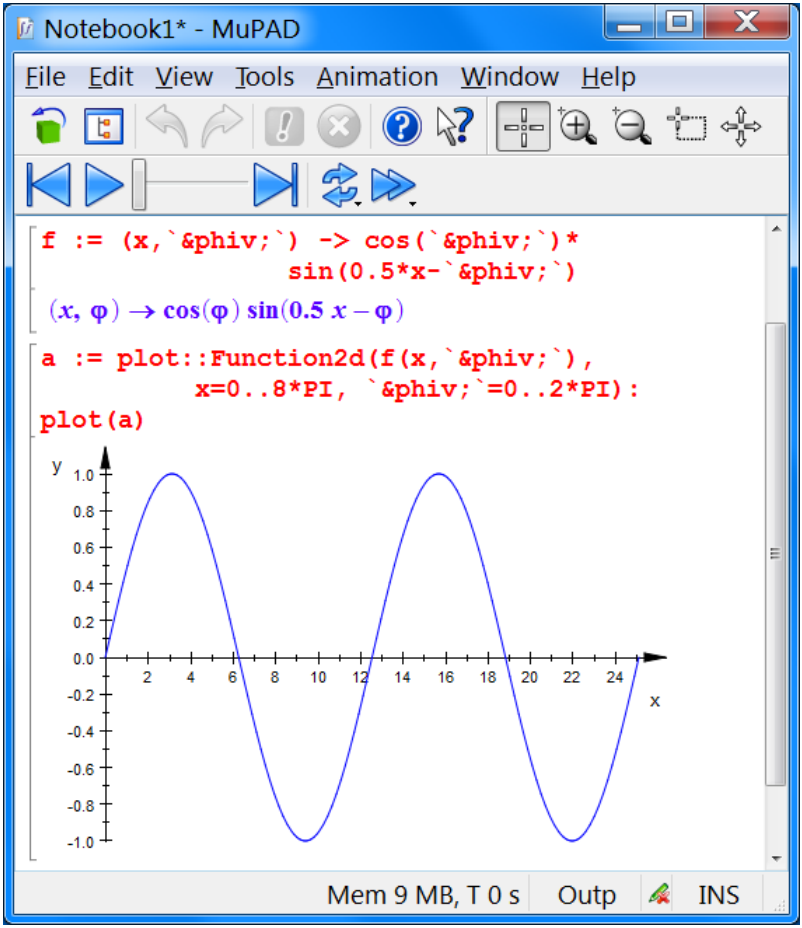


Рис. 3.9. Пример анимации в MuPAD

После выполнения (`Evaluate`) функции `plot`, воспроизводящей графический объект, состоящий из нескольких кадров, в поле вывода появится статическое изображение, соответствующее первому кадру анимации. Чтобы воспроизвести анимацию полностью, достаточно сделать

однократный щелчок левой клавишей мыши по этому изображению. При этом на стандартной панели инструментов MuPAD появятся дополнительные кнопки управления анимацией. Для повторного воспроизведения анимации достаточно нажать кнопку `Play` на этой панели, а для остановки воспроизведения – кнопку `Stop`.

Анимация трехмерных объектов организуется аналогичным образом.

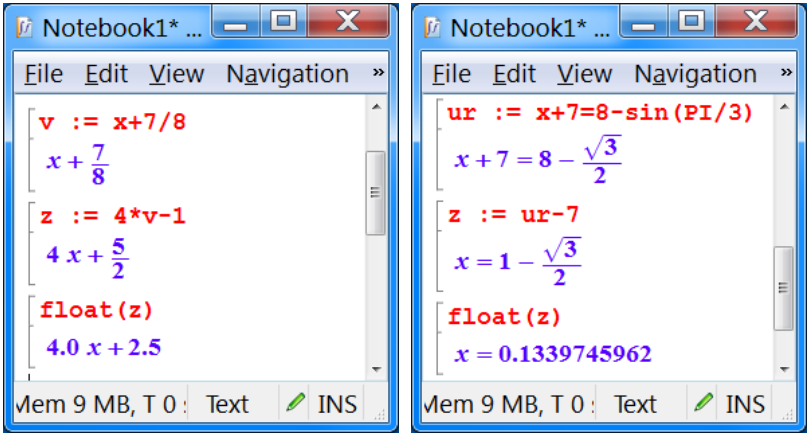
Глава 4. АНАЛИТИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ МАТЕМАТИЧЕСКИХ ВЫРАЖЕНИЙ, УРАВНЕНИЙ, НЕРАВЕНСТВ

Ранее заданные (и обозначенные некоторой переменной) выражения, уравнения, неравенства в MuPAD можно изменять, выполняя с ними различные арифметические операции и математические преобразования. Эти преобразования совершаются с символьными и числовыми выражениями как с целым, а в уравнениях и неравенствах применяются к каждой из их частей по отдельности. В частности, к любому математическому выражению, уравнению или неравенству можно прибавить (вычесть) константу или другое выражение (рис. 4.1). При этом константа (выражение) добавляется одновременно как к правой, так и к левой части уравнения (неравенства). В полученном выражении (уравнении, неравенстве) автоматически приводятся подобные. Аналогично математическое выражение, уравнение или неравенство можно домножить (разделить) на константу или другое выражение. Это позволяет, например, решать линейные уравнения и неравенства без привлечения специальных операторов (см. гл. 5). Например, если переменной `ur` ранее в программе было присвоено уравнение $x - 5 = 7$, то, прибавив к переменной `ur` число 5, можно (так же как в математике) получить уравнение $x = 12$, запись которого по форме совпадает с записью решения этого уравнения. Для этого достаточно выполнить следующие команды: `ur:=x-5=7; result:= ur+5`. Однако следует помнить, что функциональные преобразования, такие как извлечение квадратного корня, вычисление тригонометрических функций и т.д., автоматически применяются к уравнению (неравенству) в целом, а не к его частям по отдельности.

Далее в этой главе описаны функции MuPAD, предназначенные для выполнения других типичных аналитических преобразований математических выражений, уравнений, неравенств.

Обратите внимание! Значения, возвращаемые каждой из описанных в этой главе функций, рекомендуется присваивать отдельным переменным. Это облегчит их дальнейшее использование в программе. Если

полученные результаты далее в программе использовать не планируется, то для всех описанных ниже функций допускается вызов без присвоения.



а *б*

Рис. 4.1. Простейшие преобразования:
а – математического выражения; *б* – уравнения

Кроме того, следует помнить, что если в процессе математического преобразования выражение (уравнение, неравенство) будет представлено в таком виде, что какая-либо из его частей или отдельный операнд может быть вычислена (т.е. преобразована в числовое значение), то такие преобразования автоматически выполняются. Если специальных указаний сделано не было, по умолчанию все числовые результаты, а также числовые фрагменты выражений, уравнений, неравенств представляются в рациональном виде (если такое представление для данного числового значения возможно). Чтобы числовые значения представить в десятичном виде используют функцию `float` (см. рис. 4.1). Для этого после идентификатора функции `float` в круглых скобках необходимо указать выражение или переменную, значение которой необходимо преобразовать к десятичному виду. Например, в результате выполнения команды `float(1/4+ sin(PI/3))` в поле вывода получится 1.116025404, а в результате выполнения команд `z := 10*cos(PI/4); float(z)` сначала получится $5\sqrt{2}$, а потом 7.07106812. Если в качестве параметра функции `float` указать не числовое, а символьное выражение

(или уравнение, или неравенство), то в десятичный вид будут преобразованы только его числовые операнды (фрагменты). Например, в результате выполнения команды `float(x^2+3/4)` в поле вывода получится $x^2 + 0.75$. Шаблон для вызова функции `float` можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Evaluate` → `Numerically`, расположенной на командной панели `Command Bar`.

4.1. Упрощение выражений, уравнений, неравенств

Под упрощением выражения, уравнения, неравенства понимают процесс его последовательного тождественного преобразования. Целью этих преобразований чаще всего является получение более компактной формы записи исходного выражения, уравнения или неравенства.

В MuPAD, как и в других системах автоматизированных символьных вычислений, упрощение выражений (уравнений, неравенств) происходит путем последовательного применения правил тождественных математических преобразований. В процессе применения этих правил формируется совокупность выражений (уравнений, неравенств), тождественных исходному, из которой по определенному алгоритму выбирается лучшее.

В MuPAD существуют две функции, выполняющие упрощение выражений, уравнений и неравенств: `simplify` и `Simplify` (рис. 4.2). Эти функции отличаются принципом выбора наилучшего выражения (уравнения, неравенства), тождественного исходному. Функция `simplify` выполняет поиск линейно и сравнивает полученное выражение только с предыдущим, а функция `Simplify` – со всеми выражениями, полученными ранее.

Функцию `simplify` вызывают следующим образом:

```
<Переменная> := simplify(<исходное выражение>,
                                        <опции>)
```

В качестве `<исходного выражения>` можно использовать как выражения, уравнения, неравенства, списки, множества, состоящие из выражений, уравнений, неравенств, так и идентификаторы, которыми они обозначены. Например, в результате упрощения тригонометрического выражения $10 \sin x \cos x$ с помощью команды `simplify(10*sin(x)*cos(x))` в поле вывода можно получить $5 \sin(2x)$, а в результате упрощения логического

выражения $x > 9 \bigcap x > 3$ с помощью команды `simplify(x > 9 and x > 3)` в поле вывода можно получить $9 < x$.

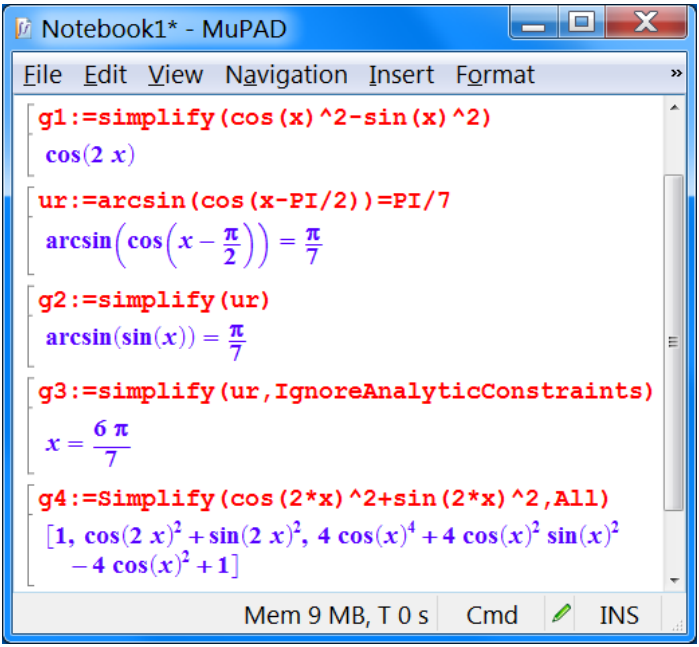


Рис. 4.2. Упрощение выражений с помощью функций `simplify` и `Simplify`

Обратите внимание, что в качестве `<исходного выражения>` при вызове функции `simplify` нельзя указывать последовательность выражений, уравнений, неравенств. Если в качестве `<исходного выражения>` используется список (или множество), то функция `simplify` выполняет упрощение каждого выражения, уравнения или неравенства, входящего в этот список (множество). `<Опции>` указываются при необходимости. В качестве опции можно указать либо `IgnoreAnalyticConstraints` (разрешает применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами, например считать тождеством arcsin(sin x) = x и т.д.), либо `Seconds = n` (ограничивает максимальное время `n` в секундах, выделяемое для выполнения

упрощения), либо `Steps = m` (ограничивает максимальное количество `m` шагов преобразования выражения).

Функция `Simplify` имеет такой же синтаксис, но позволяет использовать значительно больше опций (рис. 4.2). В частности, опция `All` позволяет показать список всех полученных в процессе упрощения выражений (уравнений, неравенств), тождественных исходному. Это позволяет выбрать наиболее удобный для дальнейшего использования вариант упрощенного выражения. В функциях `simplify` и `Simplify` реализованы разные алгоритмы поиска, поэтому в некоторых случаях они могут возвращать неодинаковые результаты упрощения одного и того же `<исходного выражения>`.

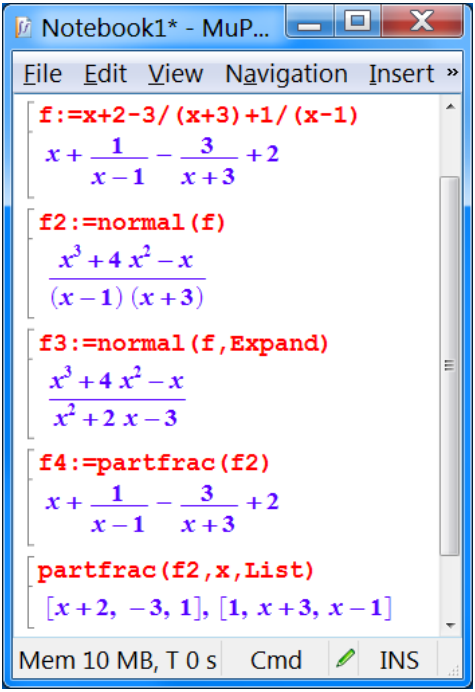


Рис. 4.3. Преобразование выражений, содержащих дроби

Кроме функций `simplify` и `Simplify` – функций общего назначения, позволяющих упрощать выражения любого вида, в MuPAD существуют

функции для тождественного преобразования выражений специального вида: содержащих дроби и радикалы. Для тождественного преобразования дробей в выражениях, уравнениях, неравенствах используют функции normal, simplifyFraction и partfrac (рис. 4.3). Функции normal и simplifyFraction приводят исходное выражение, содержащее хотя бы одну дробь, к общему знаменателю, а функция partfrac позволяет разложить исходное выражение на простые дроби.

Функции normal и simplifyFraction имеют одинаковый синтаксис и возвращают одинаковый результат. Эти функции вызывают следующим образом:

<Переменная> := **normal**(<исходное выражение>, <опции>)

<Переменная> :=
simplifyFraction(<исходное выражение>, <опции>)

Здесь параметр <исходное выражение> имеет тот же смысл, что и при вызове функции simplify. <Опции> так же указываются по необходимости. В качестве <опций> можно использовать: Expand, List, NoGcd, Rationalize, Recursive, Iterations. Опция Expand позволяет раскрыть скобки в числителе и знаменателе преобразованного выражения; опция List создает список, состоящий из числителя и знаменателя преобразованного выражения. С назначением остальных опций можно ознакомиться в справочной системе MuPAD.

Функцию partfrac вызывают следующим образом:

<Переменная> := **partfrac**(<исходное выражение>,
<аргумент>, <опции>)

Здесь параметр <исходное выражение> имеет тот же смысл, что и при вызове функции simplify. Параметр <аргумент> указывает, относительно какой из независимых переменных (если их в <исходном выражении> несколько) будут записываться искомые простые дроби. Если <исходное выражение> содержит только одну независимую символьную переменную, то <аргумент> можно опустить (не указывать). В качестве <опций> можно использовать: Full, List, MaxDegree, Adjoin, Domain, Mapcoeffs.

Опция Adjoin записывается в виде Adjoin = [<значение1>, <значение2>, …] и позволяет принудительно добавить в разложение знаменателя дроби множители вида <аргумент> ± <значение1>, <аргумент> ± <значение2>, … При использовании опции Adjoin параметр <аргумент> перед ней считается обязательным, даже если

<исходное выражение> содержит только одну независимую символьную переменную, но в записи самой опции `Adjoin` параметр <аргумент> участвовать не должен. Например, в результате выполнения команд `q := 2*x^3/(x^2 + 1); partfrac(q, x, Adjoin = [I])` в поле вывода получатся два тождественных выражения:

$$\frac{2x^3}{x^2+1} \text{ и } 2x - \frac{1}{x-\mathrm{i}} - \frac{1}{x+\mathrm{i}}.$$

Если указан <аргумент>, опция `List` создает два списка, в первом из которых перечислены все числители, а во втором – соответствующие им знаменатели дробей. С назначением остальных опций можно ознакомиться в справочной системе MuPAD самостоятельно.

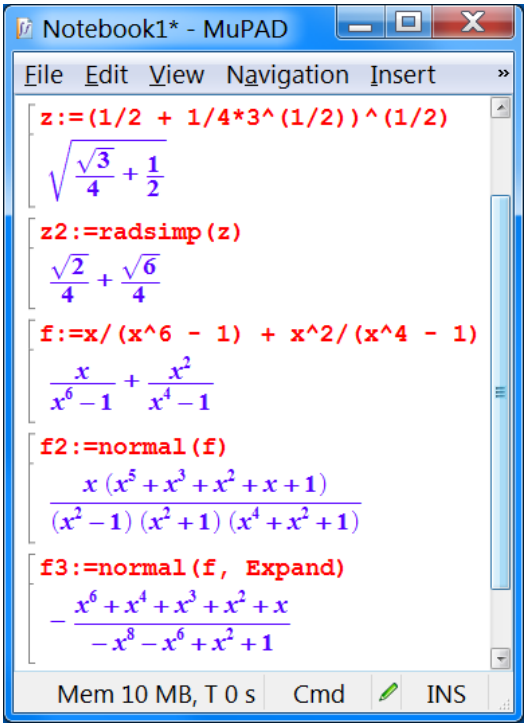


Рис. 4.4. Преобразование выражений, содержащих радикалы

Для тождественного преобразования радикалов в выражениях, уравнениях, неравенствах используют функции `radsimp` и `simplifyRadical` (рис. 4.4). Обе функции имеют одинаковый синтаксис и возвращают одинаковый результат. Функции `radsimp` и `simplifyRadical` вызывают следующим образом:

<Переменная> **:= radsimp**(<исходное выражение>)

<Переменная> **:= simplifyRadical**(<исходное выражение>)

Эти функции не имеют опций. Для их использования необходимо указать только <исходное выражение> – символьное выражение, уравнение, неравенство или переменную, которой оно обозначено ранее.

Результат, полученный с помощью тождественных преобразований любой из перечисленных выше функций, рекомендуется присвоить какой-либо <переменной>. Это облегчит дальнейшее использование в программе полученного результата. Если далее в программе результат преобразования использовать не планируется, то любую из перечисленных выше функций можно вызывать без присвоения. Тогда параметр <переменная> и оператор присвоения := можно опустить.

С помощью командной панели `Command Bar` в текст программы можно добавить шаблон функции `Simplify`. Для этого следует воспользоваться кнопкой `Simplify` $f \Rightarrow_f$. Вставленный шаблон имеет вид: `Simplify(#f)`, и содержит одно обязательное поле, отмеченное символом #. Кроме того, с помощью пунктов и подпунктов раскрывающегося списка `General Math`, расположенного на той же командной панели `Command Bar`, в текст программы можно вставить следующие шаблоны: шаблон `Simplify(#)` – выбрав последовательно `General Math` → `Simplify` → `General`; шаблон `normal(#)` – выбрав `General Math` → `Normalize`; шаблон `partfrac(#, #)` – выбрав `General Math` → `Calculus` → `Partial Function`. После добавления каждого из этих шаблонов в текст программы поля, отмеченные символом #, необходимо заполнить в соответствии с описанными выше правилами.

4.2. Подстановка

Все вычисления в MuPAD выполняются с учетом введенных ранее обозначений (значений присвоенных ранее переменных). То есть после присвоения значения переменной это значение автоматически подставляется

далее во все выражения, уравнения, неравенства, списки и т.д., записанные с использованием данной переменной. В получившихся при этом выражениях (уравнениях, неравенствах) автоматически приводятся подобные. Результаты предшествовавших присвоению вычислений при этом не изменяются. В данном случае под значением переменной следует понимать не только числовые и символьные константы и выражения (рис. 4.5), но и любую информацию, сохраненную ранее в соответствующей переменной (например, уравнение, неравенство, список, множество и т.д.).

Таким образом, чтобы в какое-то выражение, уравнение, неравенство и т.д. подставить значение одной из символьных переменных, которая использована при его записи, достаточно присвоить это значение соответствующей переменной. Чтобы убедиться, что подстановка выполнена, достаточно вывести соответствующее выражение, уравнение, неравенство и т.д. Например, после выполнения команд `delete x; y := x + 2` переменная `y` будет содержать выражение $x + 2$. Чтобы далее в это выражение подставить значение $x = 3$ и вывести результат, можно написать следующие команды: `x := 3: y`. В итоге в поле вывода получится 5.

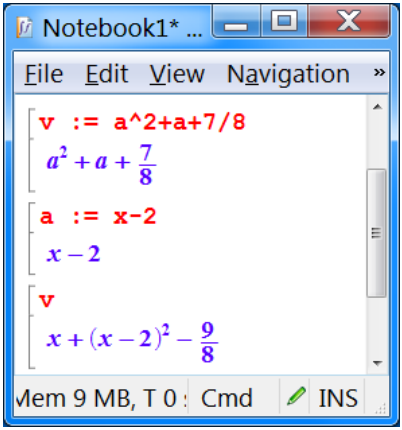


Рис. 4.5. Простейшая подстановка с изменением исходного выражения

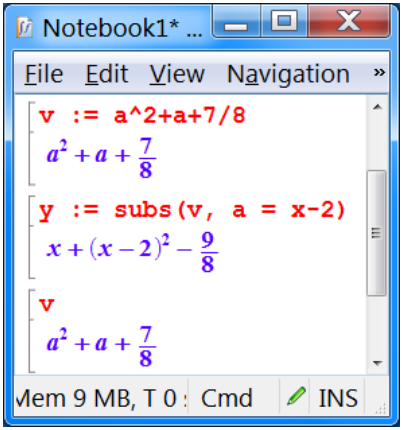


Рис. 4.6. Подстановка без изменения исходного выражения

Если необходимо в некоторое выражение (уравнение, неравенство и т.д.) подставить значения всех заданных ранее переменных без приведения

подобных, используют функцию val, которую записывают следующим образом:

<Переменная> := **val** (<исходное выражение>)

Например, если ранее в программе переменной a было присвоено значение 4, то после выполнения команды: y := val(sqrt(a)+a-3) в поле вывода получится: $\sqrt{4}+4-3$. Если ту же команду записать без использования функции val, т.е. в виде: y := sqrt(a)+a-3, то в поле вывода получится: 3.

При подстановке в исходное выражение (уравнение, неравенство) значений переменных через присвоение или функцию val изменяется само исходное выражение (уравнение, неравенство). После такой подстановки вернуть выражение (уравнение, неравенство) к исходному виду, как правило, затруднительно. Поэтому этот прием используют только в том случае, если в дальнейшем в программе исходное выражение (уравнение, неравенство) с другими параметрами использоваться не будет.

Подстановку значений символьных переменных можно осуществлять и иначе: без присвоения им конкретных значений. Для этого в MuPAD существует ряд средств для выполнения таких подстановок. Рассмотрим их подробнее.

Как было показано ранее, если в программе необходимо последовательно подставлять в некоторое выражение разные значения какой-либо переменной, то это выражение можно оформить в виде функции. Тогда в функцию можно подставлять различные значения аргумента (константы, выражения), при этом сама функция останется неизменной (см. рис. 2.9).

Для подстановки значения в выражение (уравнение, неравенство), не являющееся функцией, можно использовать функции evalAt и subs. Функция evalAt позволяет подставлять в выражения только значения отдельных переменных, а функция subs позволяет заменять не только отдельные переменные, но и целые выражения. При этом исходное выражение (уравнение, неравенство) копируется, в нем выполняется замена, а исходное выражение (уравнение, неравенство) остается неизменным (рис. 4.6).

Функция evalAt может быть записана двумя способами:

<Переменная> := **evalAt**(<исходное выражение>,
<старое1> = <новое1>, <старое2> = <новое2>, …)

<Переменная> := <исходное выражение> **|**
(<старое1> = <новое1>, <старое2> = <новое2>, …)

В качестве <исходного выражения> можно использовать как символьные выражения, уравнения, неравенства, списки, множества и т.д., так и идентификаторы, которыми они обозначены. <Старое1>, <старое2> и т.д. – переменные, встречающиеся в <исходном выражении>, которые необходимо заменить. <Новое1>, <новое2> и т.д. – переменные или выражения, на которые будут заменяться указанные перед знаком = переменные. Количество подстановок вида <старое> = <новое> может быть произвольным (не меньше одной). Если подстановок несколько, то при использовании второй формы записи этой функции их необходимо перечислять в круглых скобках. Если подстановка одна, то круглые скобки можно опустить. Например: `y := f | (x = 1, y = 2)`. Эта команда эквивалентна команде: `y := evalAt(f, x = 1, y = 2)`.

Функцию `subs` вызывают следующим образом:

<Переменная> := **subs** (<исходное выражение>,
<старое1> = <новое1>, <старое2> = <новое2>, …,
<опция>)

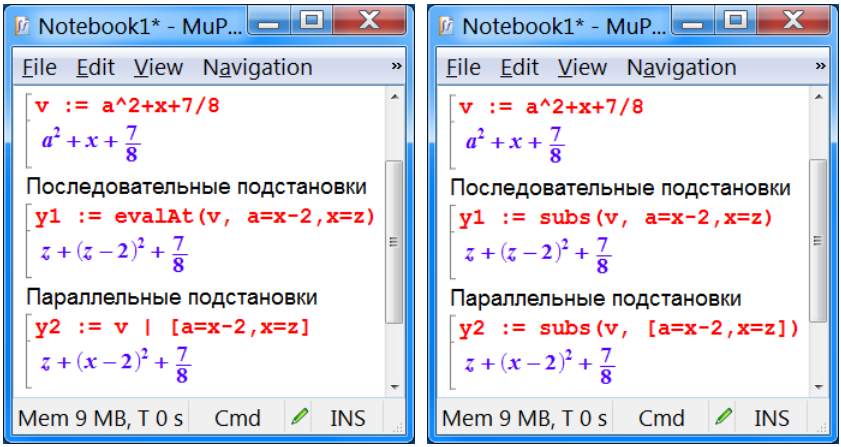
Здесь в качестве <исходного выражения> также можно использовать как символьные выражения, уравнения, неравенства, списки, множества и т.д., так и идентификаторы, которыми они обозначены. Но <старое1>, <старое2> и т.д. – это не только переменные, но и фрагменты формулы, встречающиеся в исходном выражении, которые необходимо заменить. <Новое1>, <новое2> и т.д. – соответственно переменные или выражения, на которые будут заменяться указанные перед знаком = переменные или фрагменты формулы. Количество подстановок вида <старое> = <новое> также может быть произвольным (не меньше одной). <Опция> указывается при необходимости. В качестве опции можно использовать либо `EvalChanges` (после подстановки выполнить все вычисления), либо `Unsimplified` (не упрощать полученное выражение).

По умолчанию (без опций) и функция `evalAt`, и функция `subs` приводят подобные и упрощают (в смысле разд. 4.1) выражения (уравнения, неравенства), полученные в результате указанной подстановки (замены). Например, если ранее в программе выражение $(x + t)^2 - x^2 - t^2$ обозначено через переменную `q := (x + t)^2 - x^2 - t^2`, то в результате замены $t = 3x$ получится $6x^2$ и при использовании функции `evalAt`, и при использовании функции `subs`. Чтобы в этом убедиться, можно написать

следующие команды: `evalAt(q, t = 3*x)` и `subs(q, t = 3*x)`. Но при выполнении подстановки без опций функция `evalAt` не только осуществляет указанную замену, но и ПЫТАЕТСЯ вычислить значение каждого операнда в полученном выражении (уравнении, неравенстве и т.д.), только после этого упрощает его и приводит подобные. В отличие от функции `evalAt`, функция `subs` НЕ ПЫТАЕТСЯ вычислять значения отдельных операндов в полученном выражении (уравнении, неравенстве и т.д.) перед его упрощением. Поэтому если при записи `<исходного выражения>` используются стандартные (специальные или элементарные) математические функции, то после подстановки в такое `<исходное выражение>` вместо символьных переменных *числовых значений* с помощью функции `evalAt` (без опций) значения этих стандартных функций будут вычислены и (если это возможно) записаны в рациональном виде. Например, если в выражении $5e^x$ заменить переменную x значением $2\cdot\ln(3)$ с помощью функции `evalAt` (без опций), в поле вывода получится 45; для этого можно написать команду `evalAt(5*exp(x), x = 2*ln(3))`. Так как без опций функция `subs` (в отличие от функции `evalAt`) не пытается вычислить значения отдельных операндов после подстановки, то если (как в предыдущем примере) в выражении $5e^x$ заменить переменную x значением $2\cdot\ln(3)$ с помощью функции `subs` (без опций), в поле вывода получится не 45, а $5e^{2\ln(3)}$; для этого можно написать команду `subs(5*exp(x), x = 2*ln(3))`. Чтобы завершить неоконченные функцией `subs` вычисления, можно воспользоваться функцией `eval` (или `simplify`) или сразу вызывать функцию `subs` с опцией `EvalChanges`. Таким образом, для приведенного примера команда `evalAt(5*exp(x), x = 2*ln(3))` и последовательность команд `z := subs(5*exp(x), x = 2*ln(3)): eval(z)` позволяют получить один и тот же результат: 45.

Несколько подстановок (вида `<старое> = <новое>`), указанных одновременно в функциях `evalAt` или `subs`, могут выполняться как последовательно, так и параллельно. Если подстановка осуществляется ***последовательно***, то сначала выполняется первая указанная в функции `evalAt` (или `subs`) замена `<старое1> = <новое1>`, в полученном выражении приводятся подобные, затем выполняется вторая замена `<старое2> = <новое2>`. При этом поиск происходит как в замененных фрагментах выражения, так и в остальной формуле. После этого аналогичным образом выполняются остальные подстановки. Если

подстановки выполняются ***параллельно***, то поиск фрагментов <старое1>, <старое2> и т.д. происходит только в исходной формуле (результат предыдущей подстановки игнорируется). Различия между параллельными и последовательными подстановками существенны, только если эти подстановки являются зависимыми.



а *б*

Рис. 4.7. Параллельные и последовательные подстановки, выполненные с помощью функций: *а* – `evalAt`; *б* – `subs`

Чтобы подстановки функциями `evalAt` или `subs` осуществлялись параллельно, все замены вида <старое> = <новое> необходимо перечислить в квадратных скобках: [<старое1> = <новое1>, <старое2> = <новое2>]. Если квадратные скобки отсутствуют (или вместо квадратных скобок используются круглые), то подстановки осуществляются последовательно (рис. 4.7).

При вызове функции `subs` следует помнить, что если в <исходном выражении> необходимо ЗАМЕНИТЬ ФРАГМЕНТ ФОРМУЛЫ, операнды в соответствующем параметре <старое> должны быть записаны в том же порядке, как в <исходном выражении>. В <исходном выражении> между этими операндами не должно быть других переменных или операций. Функция `subs` заменяет только фрагменты формул, полностью тождественные указанному параметру <старое>.

Если необходимо заменить фрагмент, операнды которого в исходном выражении разделены другими переменными или операциями, то вместо функции subs следует использовать функцию subsex. Функция subsex имеет такой же синтаксис, как и функция subs:

<Переменная> := **subsex**(<исходное выражение>,
<старое1> = <новое1>, <старое2> = <новое2>, …,
<опция>)

Например, в результате выполнения команды subsex(a+b+c+d, a+d=5) в поле вывода получится: $b + c + 5$. При использовании функции subs исходное выражение в данном примере не изменится.

Если подстановка вида <старое> = <новое> (или несколько подстановок <старое1> = <новое1>, <старое2> = <новое2> и т.д.) ранее присвоена какой-либо переменной, то ее идентификатор можно использовать при вызове любой из функций evalAt, subs или subsex вместо соответствующего параметра. Например, в результате выполнения команд p := x = z+3: subs(x^2+x, p) в поле вывода получится выражение $z + (z + 3)^2 + 3$. Тот же результат можно получить, выполнив команду subs(x^2+x, x = z+3).

Полученное в результате подстановки с помощью любой из перечисленных выше функций выражение (уравнение, неравенство и т.д.) рекомендуется присвоить какой-либо <переменной>. Это облегчит дальнейшее использование в программе полученного результата. Если далее в программе результат подстановки использовать не планируется, то любую из перечисленных выше функций можно использовать без присвоения. Тогда параметр <переменная> и оператор присвоения := можно опустить.

Из всех описанных ранее в этом разделе функций с помощью кнопок на командной панели Command Bar в текст программы можно добавить только шаблон функции evalAt. Для этого следует воспользоваться кнопкой Evaluate with x=a $f|_{x=a}$. Вставленный шаблон имеет вид: (#f) | #x=#a и содержит три поля, отмеченные символом #. В этом шаблоне оператор присвоения := отсутствует (при необходимости его следует дописать вручную). Поле #f слева от символа | необходимо заменить на <исходное выражение>, а в полях #x и #a записать соответственно <старую> и <новую> переменные или выражения (согласно описанным выше правилам).

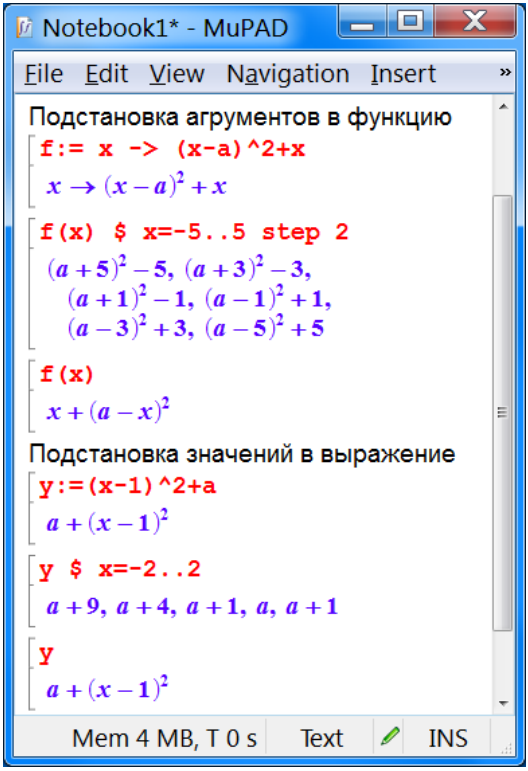


Рис. 4.8. Подстановка в выражение нескольких значений одной переменной

Все описанные выше функции (`evalAt`, `subs`, `subsex`) позволяют подставить в исходное выражение (уравнение, неравенство и т.д.) по одному значению каждой (одной или нескольких) из содержащихся в нем переменных. Чтобы в какое-то выражение последовательно подставить несколько значений одной переменной (рис. 4.8), используют оператор `$` следующим образом:

`<Результат> := <выражение>` **$** `<переменная> <значения>`

В качестве параметра `<выражение>` можно написать выражение, (уравнение, неравенство и т.д.) либо явным образом, либо в виде идентификатора переменной, которой оно ранее обозначено, либо в виде идентификатора объявленной ранее функции с символьными аргументами.

В качестве параметра <переменная> можно указать одну из символьных переменных, явно содержащихся в <выражении>, или целое положительное число. Если вместо <переменной> указано целое число (параметр <значения> при этом отсутствует), то оператор $ дублирует <выражение> указанное количество раз.

Если в качестве параметра <переменная> использована символьная переменная, то после него с помощью параметра <значения> необходимо написать, какие значения этой переменной необходимо подставлять в <выражение>. Параметр <значения> можно указать двумя способами:

1) в виде интервала

<переменная> = <начало> .. <конец> step <шаг>

В этом случае <переменная> будет автоматически изменяться в заданных пределах с указанным шагом. Если <шаг> равен единице, то его можно не указывать (опустить). Синтаксический элемент step в рассматриваемой конструкции в этом случае тоже не пишется. Границы интервала и <шаг> можно задавать как в виде чисел, так и в виде выражений, но таких, чтобы, изменяя <начало> несколько раз на <шаг>, можно было получить <конец>. Например: k = 1..8 или x = a .. a + 5 step 1/2 (a – незаданная ранее символьная переменная). Для каждого значения <переменной> будет вычислено соответствующее значение <выражения>;

2) в виде множества

<переменная> in <множество>

В качестве параметра <множество> можно либо явно перечислить в фигурных скобках элементы множества (например: x in {1, 3, 5}), либо использовать идентификатор переменной, которой ранее было присвоено соответствующее множество (например, x in z, если ранее в программе z := {1, 3, 5}). Подробнее о множествах см. в гл. 10. Каждый элемент <множества> подставляется в <выражение> вместо <переменной>.

Из выражений, полученных после подстановок значений <переменной>, оператор $ формирует последовательность вида:

<Выражение1>, <выражение2>, <выражение3>, …

Элементы последовательности отделяются друг от друга запятыми. Эта последовательность присваивается переменной <результат>. При этом исходное <выражение> не преобразуется (остается неизменным).

В этом можно убедиться, если после подстановки значений вновь вывести исходное <выражение> (см. рис. 4.8).

Следует отметить, что если при использовании оператора $ опустить (не указывать) параметры <выражение> и <переменная>, то можно получить последовательность чисел. Например, результатом выполнения команды `$ 3..9 step 2` является последовательность целых чисел 3, 5, 7, 9.

Если в выражение (уравнение, неравенство и т.д.) необходимо подставить по несколько значений для двух, трех и более переменных, то оператор $ последовательно используют несколько раз. Например, в результате выполнения команды `x*y $ x=1..3 $ y=2..5` получится следующая последовательность чисел: 2, 4, 6, 3, 6, 9, 4, 8, 12, 5, 10, 15.

Полученную с помощью оператора $ последовательность <выражений> можно преобразовать в список или множество, заключив либо в квадратные (для преобразования в список), либо в фигурные скобки (для преобразования во множество). Например, в результате выполнения команд `p := x^k $ k=1..5; s := [p]` сначала получится последовательность, которая преобразуется в список [x, x^2, x^3, x^4, x^5], присвоенный переменной `s`. Тот же результат можно получить, выполнив команду `s := [x^k $ k=1..5]`. Подробнее о списках и множествах см. в гл. 10.

4.3. Запись выражения, уравнения, неравенства через другие элементарные функции

Когда необходимо исходное выражение (уравнение, неравенство и т.д.), записанное с помощью одних элементарных или специальных функций, переписать через другие элементарные или специальные функции, в MuPAD используют функцию `rewrite` (рис. 4.9), которая применяет к исходному выражению (уравнению, неравенству и т.д.) правила тождественных преобразований между соответствующими функциями, если они существуют. Функцию `rewrite` вызывают следующим образом:

```
<Переменная> := rewrite(<исходное выражение>,
                                <целевая функция>)
```

В качестве <исходного выражения> можно использовать как выражения, уравнения, неравенства, списки, множества и т.д., так и идентификаторы, которыми они обозначены. <Целевая функция> указывает, через какие элементарные или специальные функции необходимо переписать <исходное выражение>. В качестве параметра <целевая функция> можно использовать следующие идентификаторы: `and-or`, `arccos`, `arccosh`, `arccot`, `arccoth`, `arcsin`, `arcsinh`, `arctan`, `arctanh`, `arg`, `bernoulli`, `cos`, `cosh`, `cot`, `coth`, `diff`, `D`, `erf`, `erfc`, `erfi`, `exp`, `fact`, `gamma`, `harmonic`, `heaviside`, `inverf`, `inverfc`, `lambertW`, `ln`, `max`, `min`, `piecewise`, `psi`, `sign`, `sin`, `sincos`, `sinh`, `sinhcosh`, `tan`, `tanh`. Большинство этих обозначений совпадает с названиями соответствующих элементарных или специальных функций, перечисленных в табл. 2.6, 2.7, а также с некоторыми из функций, которые будут рассмотрены в пособии далее (например, в гл. 6).

Если исходное выражение записать через указанные функции не удалось, то функция `rewrite` возвращает его в неизменном виде.

Выражение, полученное с помощью функции `rewrite`, рекомендуется присвоить какой-либо <переменной>. Это облегчит дальнейшее использование в программе полученного результата. Если далее в программе результат преобразования использовать не планируется, то функцию `rewrite` можно вызывать без присвоения. Тогда параметр <переменная> и оператор присвоения `:=` можно опустить.

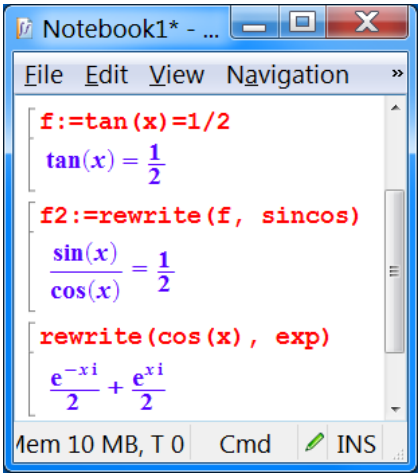


Рис. 4.9. Запись выражения через другие функции

Последовательность раскрывающихся списков `General Math` → → `Rewrite`, расположенная на командной панели `Command Bar` позволяет вставить в текст программы следующие шаблоны для вызова функции `rewrite` (с некоторыми из перечисленных выше целевых функций): `rewrite(#, diff)`, `rewrite(#, exp)`, `rewrite(#, fact)`, `rewrite(#, gamma)`, `rewrite(#, heaviside)`, `rewrite(#, ln)`, `rewrite(#, sign)`, `rewrite(#, sincos)`. В каждом из этих шаблонов поле, отмеченное символом #, необходимо заменить `<исходным выражением>` в соответствии с описанными выше правилами.

4.4. Раскрытие скобок и вынесение общих множителей

Важную роль в преобразовании математических выражений, уравнений, неравенств играет вынесение общих множителей и раскрытие компактно записанных выражений.

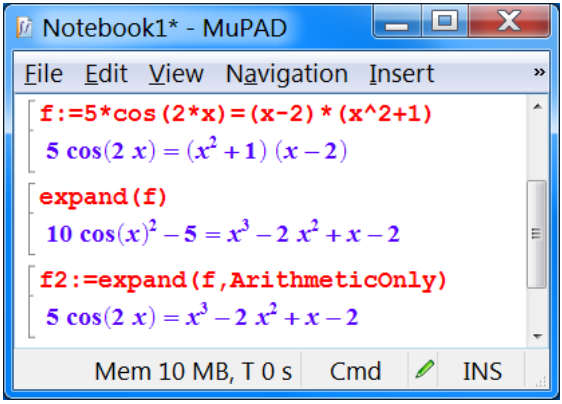


Рис. 4.10. Преобразование выражений с помощью функции `expand`

Когда необходимо раскрыть выражение (раскрыть скобки, выполнить арифметические преобразования, расписать тригонометрические, гиперболические, логарифмические и специальные функции от сложных аргументов через функции простых аргументов), в MuPAD используют функцию `expand`, которая применяет к исходному выражению правила тождественных преобразований соответствующих функций (рис. 4.10).

Функцию expand вызывают следующим образом:

<Переменная> := **expand**(<исходное выражение>,
<ограничение>, <опции>)

В качестве <исходного выражения> можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они обозначены. <Опции> и <ограничения> указываются при необходимости. В качестве <опции> можно использовать либо IgnoreAnalyticConstraints (разрешает применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами, например считать тождеством $\arcsin(\sin x) = x$ и т.д.), либо ArithmeticOnly (раскрывает только арифметическую часть выражения без тригонометрических, гиперболических, логарифмических преобразований, а также без преобразования специальных функций), либо MaxExponent = n (запрещает раскрывать степени выше n).

<Ограничение> – это выражение, которое необходимо оставить неизменным (не раскрывать) в процессе преобразования <исходного выражения>. При использовании функции expand можно указать несколько <ограничений>. В этом случае они перечисляются через запятую.

Примеры использования функции expand с опциями и без опций приведены на рис. 4.10.

Для выполнения операций, обратных операциям, осуществляемым функцией expand, в MuPAD существует несколько функций: factorout, factor – выносят общие множители; collect – группирует слагаемые при одинаковых степенях символьных переменных; combine – переписывают выражение, содержащее тригонометрические, гиперболические, логарифмические функции простых аргументов, через те же функции составных аргументов.

Функция factorout позволяет вынести из алгебраического выражения ОДИН общий множитель, вид которого известен. Эта функция выполняет деление исходного выражения на предложенный общий множитель. Операция выполняется независимо от того, действительно ли исходное выражение можно поделить на предложенный множитель без остатка. Функцию factorout вызывают следующим образом:

<Переменная> := **factorout**(<исходное выражение>,
<множитель>)

В качестве <исходного выражения> можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они обозначены. А в качестве <множителя> можно использовать только константы, математические выражения или идентификаторы, которыми они ранее обозначены. Если в качестве <исходного выражения> указано уравнение или неравенство, то на указанный <множитель> делится каждая его часть (и правая, и левая).

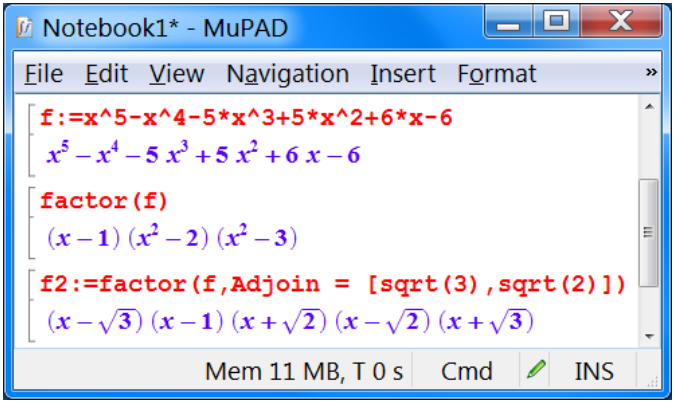


Рис. 4.11. Преобразование выражений с помощью функции `factor`

Функция `factor` позволяет вынести из алгебраического выражения ВСЕ общие множители (разложить на простые множители). Вид этих множителей функция `factor` подбирает сама (рис. 4.11). Если функция не нашла ни одного множителя, на который исходное выражение делится без остатка, она возвращает исходное выражение непреобразованным. Функцию `factor` вызывают следующим образом:

<Переменная> := **factor**(<исходное выражение>, <опции>)

Здесь в качестве <исходного выражения> можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они обозначены. Если в качестве <исходного выражения> указано уравнение или неравенство, то на простые множители разлагается каждая его часть (и правая, и левая) по отдельности. <Опции> указываются при необходимости. В качестве <опций> можно использовать `MaxDegree`, `Adjoin`. Опция `Adjoin` записывается в виде `Adjoin = = [<значение1>, <значение2>…]` и позволяет принудительно добавить

в разложение множители вида <аргумент> ± <значение1>, <аргумент> ± <значение2> и т.д. Но ни идентификатор <аргумента>, ни знак ± при записи опции `Adjoin` указывать не нужно. Например, в результате выполнения команды `factor(x^2+4, Adjoin = [2*I])` в поле вывода получится выражение: $(x - 2i)(x + 2i)$.

Другие примеры использования функции `factor` (с опциями и без опций) для разложения выражения на простые множители приведены на рис. 4.11.

Функция `ifactor` позволяет разложить целое число на простые множители. Ее вызывают следующим образом:

<Переменная> **:= ifactor**(<исходное число>)

В качестве <исходного числа> можно указать целое число или арифметическое выражение, результатом вычисления которого является целое число, а также идентификатор, которым оно обозначено. Например, если в командном поле написать: `ifactor(990)`, то в поле вывода получится: $2\ 3^2\ 5\ 11$. Результат, возвращаемый функцией, – это невычисленное произведение целых чисел, которое можно использовать в программе наравне с <исходным числом>. При необходимости из полученного невычисленного произведения целых чисел можно снова получить <исходное число> с помощью функции `expand`.

Если исходное выражение зависит от нескольких символьных переменных, то при решении некоторых задач возникает потребность сгруппировать входящие в него слагаемые, содержащие одинаковые степени одной из них. В MuPAD для этого можно использовать функцию `collect`. Для функции одной переменной эта операция эквивалентна приведению подобных, которое во всех вычислениях с MuPAD выполняется автоматически. Поэтому применение функции `collect` к выражению, содержащему одну независимую символьную переменную, не имеет смысла. Так как слагаемые в выражении можно также группировать не только по степеням переменных, но и по степеням входящих в выражение функций (стандартных или нестандартных), функцию `collect` вызывают следующим образом:

<Переменная> **:= collect**(<исходное выражение>,
[<аргумент1>, <аргумент2>, …,
<функция1>, <функция2>, …])

В качестве <исходного выражения> можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они

обозначены. После <исходного выражения> необходимо указать хотя бы один <аргумент> или название (идентификатор) <функции>, по степеням которой будут сгруппированы слагаемые. Если указан один <аргумент> (или только одна <функция>), квадратные скобки можно опустить. Если указано несколько <аргументов>, слагаемые будут группироваться относительно разных степеней произведений этих <аргументов> (рис. 4.12). Если указано несколько <функций>, слагаемые будут группироваться относительно разных степеней произведений этих <функций>.

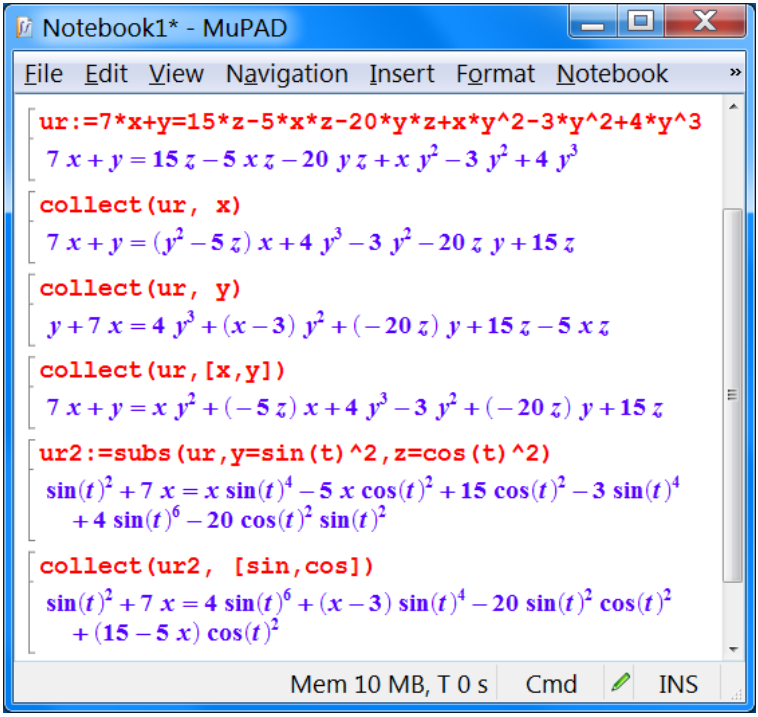


Рис. 4.12. Группировка слагаемых при разных степенях аргумента или функции

Если исходное выражение содержит несколько тригонометрических (гиперболических, логарифмических) функций от разных аргументов, то, согласно соответствующим тождествам, в некоторых случаях их комбинации

можно представить в виде одной функции составного аргумента. Например, $2 \sin x \cos x = \sin 2x$. Иногда такие преобразования позволяют записать исходное выражение в более компактном виде. Функция `combine` позволяет выполнять такие преобразования. Функцию `combine` вызывают следующим образом:

```
<Переменная> := combine(<исходное выражение>,
                                    <цель>, <опция>)
```

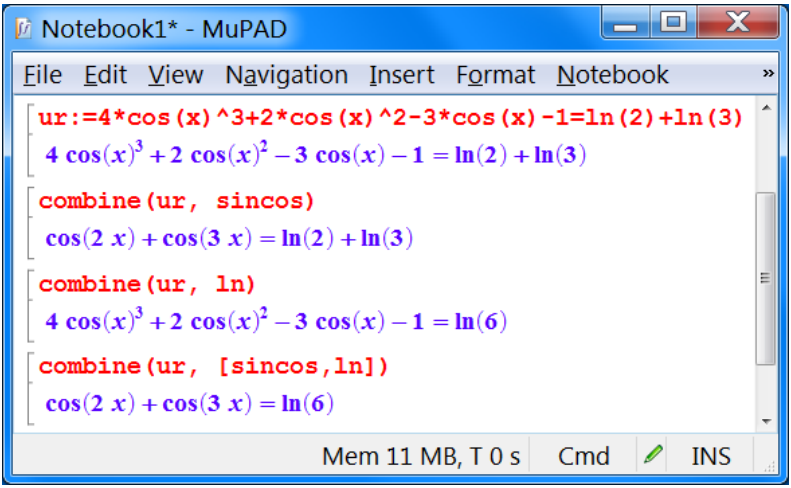


Рис. 4.13. Преобразование выражений с помощью функции `combine`

В качестве `<исходного выражения>` можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они обозначены. После `<исходного выражения>` необходимо указать, какие функции будут являться `<целью>` преобразования (правило преобразования каких функций необходимо использовать). В качестве параметра `<цель>` можно использовать следующие идентификаторы: `arctan`, `exp`, `int`, `gamma`, `ln`, `sincos`, `sinhcosh`. Например, если в качестве `<цели>` указано `sincos`, то инициируется выполнение тригонометрических преобразований. Если в `<исходном выражении>` участвуют функции разных типов, то в качестве `<цели>` можно использовать комбинацию указанных выше обозначений целевых функций. В этом случае соответствующие идентификаторы записывают в квадратных скобках

через запятую (рис. 4.13). Например: `[sincos, ln]`. В качестве <опции> функции `combine` можно использовать только `IgnoreAnalyticConstraints` (разрешает применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами; например считать тождеством $\arcsin(\sin x) = x$ и т.д.)

Выражения, полученные в результате применения функций `expand`, `factorout`, `factor`, `ifactor`, `collect` и `combine`, рекомендуется присваивать каким-либо <переменным>. Это облегчит дальнейшее использование в программе полученных результатов. Если далее в программе соответствующий результат преобразований использовать не планируется, то любую из перечисленных выше функций можно вызвать без присвоения. Тогда параметр <переменная> и оператор присвоения `:=` можно опустить.

С помощью пунктов и подпунктов раскрывающегося списка `General Math`, расположенного на командной панели `Command Bar`, в текст программы можно вставить шаблоны для вызова некоторых из перечисленных выше функций: шаблон `expand(#)` – выбрав последовательно `General Math` → `Expand`; шаблон `factor(#)` – выбрав `General Math` → `Factor`; шаблон `collect(#f, #x)` – выбрав `General Math` → `Polynomials` → `Collect`; шаблоны `combine(#)`, `combine(#, arctan)`, `combine(#, exp)`, `combine(#, ln)`, `combine(#, _power)`, `combine(#, sincos)`, `combine(#, sqrt)`, `combine(#, sinhcosh)` – выбрав соответствующий пункт из списка `General Math` → `Combine`. Поля, отмеченные символом `#`, необходимо заполнить в соответствии с описанными выше правилами.

4.5. Преобразование отдельных составляющих частей выражений, уравнений, неравенств

Нередко отдельные составляющие части крупных выражений, уравнений, неравенств нуждаются в различных преобразованиях. Например, правую часть уравнения требуется упростить, а левую – разложить на простые множители. Поэтому в системах, позволяющих выполнять символьные вычисления на компьютере, таких как MuPAD, вводится ряд функций, позволяющих из исходных выражений, уравнений, неравенств, списков, множеств и т.д. извлекать отдельные части (операнды, элементы). Как правило,

исходные по отношению к этим функциям выражения, уравнения, неравенства и т.д. в программах имеют смысл промежуточных или окончательных результатов преобразований. Именно для таких выражений, уравнений, неравенств и т.д. возникает необходимость, например, проконтролировать количество и вид независимых символьных переменных. Периодический контроль списка независимых символьных переменных в процессе длительных расчетов позволит, в частности, своевременно избавляться от случайных опечаток. Чтобы получить список независимых символьных переменных (рис. 4.14), используют функцию `indets`, которую вызывают следующим образом:

<Переменная> **:= indets** (<исходное выражение>, <опции>)

В качестве <исходного выражения> здесь можно использовать как символьные выражения, уравнения, неравенства, списки, множества и т.д., так и идентификаторы, которыми они обозначены. Функция `indets` возвращает множество, состоящее из независимых символьных переменных, входящих в <исходное выражение> (элементы множества заключены в фигурные скобки и упорядочены). <Опции> указываются при необходимости. В качестве <опций> можно использовать: `All` (добавляет к списку независимых символьных переменных список используемых в <исходном выражении> функций и операций), `PolyExpr` (интерпретирует <исходное выражение> как многочлен, а все функции, входящие в исходное выражение, но не являющиеся многочленами, считает независимыми переменными), `RatExpr` (интерпретирует <исходное выражение> как рациональное, а все функции, входящие в <исходное выражение>, но не являющиеся рациональными, считает независимыми переменными).

Чтобы определить количество операндов (например, слагаемых или сомножителей) в выражении, уравнении, неравенстве и т.д., используют функцию `nops`, которую вызывают следующим образом:

<Переменная> **:= nops** (<исходное выражение>)

В качестве <исходного выражения> можно указать выражение, уравнение, неравенство, список `[ ]`, множество `{ }` или строку, а также идентификатор, которым он (она) обозначен. Функция возвращает количество (целое число) операндов в выражении, количество частей (две) в неравенстве, уравнении, количество элементов в списке, множестве, количество символов в строке (см. рис. 4.14). Функцию `nops` можно применять только к одному <исходному выражению>.

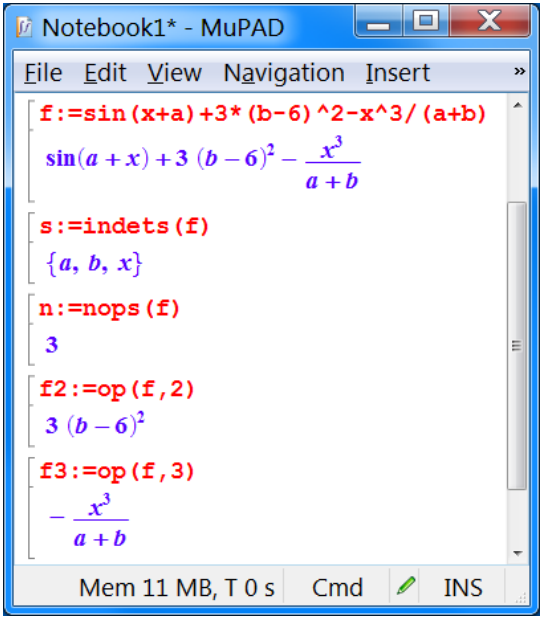


Рис. 4.14. Поиск в выражении независимых символьных переменных, определение количества операндов и извлечение одного из них

Чтобы извлечь (создать копию) *один из операндов* (например, одно из слагаемых) из выражения, часть из уравнения, неравенства, элемент из списка и т.д., используют функцию `op`, которую вызывают следующим образом:

`<Переменная> :=` **`op`** `(<исходное выражение>, <номер>)`

Чтобы извлечь (создать копию) *несколько расположенных подряд операндов* (например, несколько слагаемых) из выражения, элементов из списка и т.д., функцию `op` вызывают следующим образом:

```
<Переменная> := op (<исходное выражение>,
                    <номер1>..<номерN>)
```

В качестве `<исходного выражения>` можно использовать выражение, уравнение, неравенство, список `[ ]`, множество `{ }` или строку, а также идентификатор, которым он (она) обозначены. После `<исходного выражения>` необходимо указать хотя бы один `<номер>` операнда. Операнды нумеруются, начиная с единицы. Под нулевым операндом понимают функцию или операцию, выполняемую в `<исходном выражении>` над

операндами. Если из <исходного выражения> необходимо извлечь несколько расположенных подряд операндов, параметр <номер> можно записать в виде диапазона <номер1>..<номерN>. Если необходимо извлечь все операнды, содержащиеся в <исходном выражении>, параметр <номер> можно не указывать (опустить). Если при вызове функции op указан <номер> одного операнда, то она возвращает выражение или переменную, соответствующую данному операнду (см. рис. 4.14). Если при вызове функции op параметр <номер> отсутствует или указан в виде диапазона, то она возвращает последовательность выражений и (или) переменных, разделенных запятыми. Например, если ранее в программе выражение $5x + 3x^2 + 7$, содержащее три слагаемых, было присвоено переменной z:=5*x+3*x^2+7, то в результате выполнения команды op(z,2) получится выражение $3x^2$ (второе слагаемое), в результате выполнения команды op(z,1..3) или или команды op(z) – последовательность выражений $5x$, $3x^2$, 7 (все слагаемые), а чтобы получить последовательность $5x$, 7, состоящую из первого и третьего слагаемых, можно написать команду op(z,1), op(z,3) или op(z,k) $ k in {1,3}, использовав оператор $. Если операнд с указанным <номером> в <исходном выражении> отсутствует, функция op возвращает FAIL (с англ. *неудача*). Например, в результате выполнения команды op(a+b,3) получится FAIL.

Каким бы ни был результат, возвращаемый функцией op, <исходное выражение> при этом не изменяется.

Если интересующий программиста операнд сам имеет сложную структуру (состоит из нескольких операндов), то чтобы из выражения, уравнения, неравенства и т.д. извлечь (создать копию) *один операнд из операнда* (например, второй сомножитель из третьего слагаемого), также используют функцию op, которую вызывают следующим образом:

```
<Переменная> := op(<исходное выражение>,
                                    [<номер1>, <номер2>])
```

Здесь <номер1> – номер операнда в <исходном выражении>, <номер2> – номер операнда этом в операнде с <номером1> и т.д. Эта запись эквивалентна последовательному применению функции op:

```
<Переменная> := op(op(<исходное выражение>, <номер1>),
                                          <номер2>)])
```

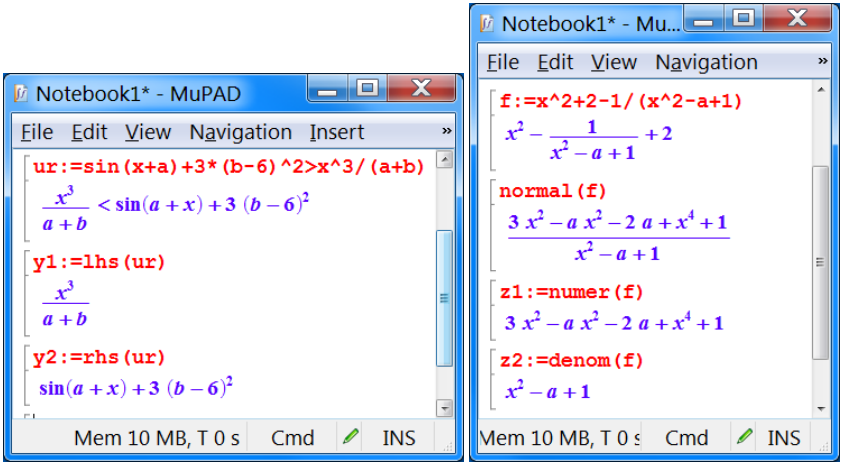
Кроме функций indets, nops, op, которые применяются к исходным выражениям любого типа, в MuPAD существуют функции, предназначенные *для извлечения (создания копии) частей уравнений и неравенств* (lhs, rhs) и *для извлечения числителя и знаменателя дроби* (numer, denom).

Функции `lhs` и `rhs` имеют одинаковые правила использования:

<Переменная> := **lhs** (<исходное выражение>)

<Переменная> := **rhs** (<исходное выражение>)

При вызове этих функций в качестве <исходного выражения> можно указывать только уравнения `f1 = f2`, неравенства `f1 > f2`, интервалы `a...b`, диапазоны `a..b` и идентификаторы, которыми они обозначены. Применять эти функции к математическим выражениям или спискам нельзя. Каждая из этих функций возвращает выражение или переменную (рис. 4.15), но для функции `lhs` это выражение соответствует левой части <исходного выражения>, а для функции `rhs` – правой.



а *б*

Рис. 4.15. Извлечение частей неравенства (*а*); числителя и знаменателя дроби (*б*)

Функции `numer` и `denom` также имеют одинаковые правила использования:

<Переменная> := **numer** (<исходное выражение>)

<Переменная> := **denom** (<исходное выражение>)

При вызове этих функций в качестве <исходного выражения> можно указывать только математические выражения и идентификаторы, которыми они обозначены. Применять эти функции к уравнениям, неравенствам, спискам и тому подобному нельзя. Каждая из этих функций возвращает выражение или переменную (см. рис. 4.15), но для функции `numer` это

выражение соответствует числителю дроби, а для функции denom – знаменателю. Если в исходное выражение дробь входит как один из операндов, то каждая из функций сначала приводит все выражение к общему знаменателю и только потом у полученной дроби определяет соответственно числитель или знаменатель.

<Исходное выражение> является обязательным параметром всех перечисленных выше функций.

Выражение, полученное в результате применения любой из перечисленных выше функций, рекомендуется присвоить какой-либо <переменной>. Это облегчит дальнейшее использование в программе полученного результата. Если далее в программе соответствующий результат использовать не планируется, то любую из перечисленных выше функций можно использовать без присвоения. Тогда параметр <переменная> и оператор присвоения := можно опустить.

4.6. Отложенные вычисления

По умолчанию все преобразования формул и вычисления автоматически выполняются сразу же (незамедлительно), как только выполняется команда Evaluate для того командного поля, в котором они записаны. При этом подставляются значения всех заданных ранее переменных и вычисляются значения функций, если их аргументы таковы, что эти вычисления выполнить возможно.

Но не все преобразования в символьных вычислениях необходимо выполнять незамедлительно. В некоторых случаях сначала требуется преобразовать, например, только аргумент функции, а ее значение предполагается вычислять позже. Или сначала выполняют преобразования с функциями в общем виде, используя какие-то их свойства, а не значения, и только в полученное после этого преобразования выражение подставляют значения функции. В этих и других аналогичных случаях некоторые вычисления необходимо временно отложить (отсрочить). Задержка вычислений полезна также для контроля правильности введенных формул, так как отложенные вычисления отображаются в поле вывода без преобразования.

Самым простым способом отсрочить некоторые вычисления является использование для этого рассмотренной ранее функции val (см. разд. 4.1). Эта функция вызывается следующим образом:

<Переменная> := val(<исходное выражение>)
и позволяет отсрочить выполнение всех операций и вычисление всех функций, указанных в <исходном выражении>. Однако значения всех ранее заданных переменных в <исходное выражение> подставляются.

Функция hold является более гибким инструментом управления задержкой (отсрочкой) вычислений. Она позволяет предотвратить не только выполнение операций и вычисление значений функций, но и подстановку присвоенных ранее переменных. Функцию hold используют одним из следующих способов:

<Переменная> := **hold(**<выражение>**)**
<Переменная> := **hold(**<функция>**)** (<аргумент>, …)
<Переменная> := <функция>(**hold(**<аргумент>**)**, …)

В первом случае откладывается вычисление всего <выражения>, включая все входящие в него операции, функции и подстановку значений ранее заданных переменных. Во втором случае отложенным является только вычисление <функции>, а ее <аргумент> и остальная часть выражения вычисляются. В третьем случае отложенным является вычисление <аргумента>, но этот <аргумент> подставляется в <функцию> (рис. 4.16). Остальная часть выражения также вычисляется.

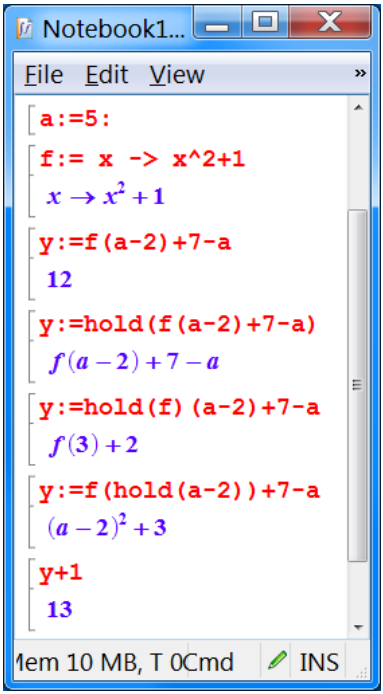


Рис. 4.16. Отложенные вычисления

Здесь в качестве параметра <выражение> можно использовать как выражения, уравнения, неравенства, так и идентификаторы, которыми они обозначены. В качестве <функции> можно использовать идентификаторы любых стандартных и нестандартных функций MuPAD.

В качестве <аргумента> используют переменные или выражения, значения которых совместимы по типу с аргументом соответствующей функции при ее объявлении. Если функция имеет несколько аргументов, то можно отложить (отсрочить) вычисление каждого из них аналогичным образом.

С помощью функции hold можно также отсрочить вычисление части <выражения>.

И с помощью функции val, и с помощью функции hold вычисления откладываются только на время выполнения данного оператора (команды). Если в следующем операторе использовать переменную, которой был присвоен результат отложенных вычислений, то все отложенные вычисления автоматически выполняются (см. рис. 4.16). Если все же необходимо принудительно выполнить отложенные вычисления, используют функцию eval:

<Переменная> := **eval** (<исходная переменная>)

Если при вызове этой функции <исходной переменной> обозначено выражение (уравнение, неравенство и т.д.), содержащее отложенные вычисления, то после выполнения функции eval эти вычисления будут реализованы, в полученном выражении (уравнении, неравенстве и т.д.) будут приведены подобные.

Функция eval отменяет ограничения, наложенные функцией hold. Если <исходная переменная> содержит несколько ограничений, наложенных с помощью нескольких функций hold, то однократное применение к ней функции eval отменяет действие только одной функции hold, являющейся внешней по отношению ко всем остальным. Повторное применение к этой переменной функции eval приведет к отмене ограничений, наложенных функцией hold следующего уровня вложения.

Функции hold и val накладывают ограничение на выполнение вычислений только в пределах одного оператора (команды). Если необходимо, чтобы некоторые вычисления НЕ ВЫПОЛНЯЛИСЬ на протяжении нескольких операторов, для отсрочки вычислений используют функцию freeze, которая создает неактивную копию исходной функции:

<Переменная> := **freeze** (<исходная функция>)

Здесь в качестве <исходной функции> указывают идентификатор любой стандартной или нестандартной функции, неактивную копию которой необходимо создать. Далее в программе <переменная>,

указанная слева от знака присвоения в этом операторе, используется в качестве имени (идентификатора) неактивной функции. Функция с этим именем будет оставаться неактивной на протяжении нескольких операторов (команд) программы, пока действие функции `freeze` не будет отменено.

Функция `freeze` вместе с идентификатором исходной функции может использоваться в выражениях:

<Переменная> **:=** `freeze(<исходная функция>)(<аргумент>)`

В этом случае <переменная>, указанная слева от знака присвоения, будет содержать выражение, вычисление которого отложено.

Отменить действие функции `freeze` можно только с помощью функции `unfreeze`:

<Переменная> **:=** **`unfreeze`**`(<исходная переменная>)`

Здесь в качестве <исходной переменной> должна быть указана переменная, которой ранее присвоена неактивная копия функции, или выражение, содержащее неактивную копию функции в качестве одного из операндов. Функция `unfreeze` делает все записанные в <исходной переменной> функции и операции активными, в результате чего отложенные ранее вычисления выполняются.

Функция `eval` не может снять ограничения, наложенные функцией `freeze`, и наоборот, функция `unfreeze` не может снять ограничения, наложенные функцией `hold`.

4.7. Ограничение свойств переменных

Некоторые математические операции, а следовательно, и соответствующие им стандартные функции не являются однозначными. Например, если x является вещественным (некомплексным) числом, то $\sqrt{x^2} = \pm|x|$; если x – комплексное число, то в результате извлечения из него квадратного корня также можно получить два значения. Обратные тригонометрические функции являются однозначными только на определенных интервалах. Поэтому для корректного выполнения упрощения некоторых выражений, а в некоторых случаях для корректного выполнения математических преобразований в целом, необходимо конкретизировать свойства (например, принадлежность к определенному числовому множеству) соответствующих символьных переменных в программе.

По умолчанию большинство не заданных ранее в программе символьных переменных считается комплексными. Однако по смыслу совершаемых математических преобразований символьные переменные могут быть вещественными, положительными вещественными, отрицательными вещественными, четными целыми, нечетными целыми и т.д. Результаты математических преобразований выражений, записанных через переменные, относящиеся к разным числовым множествам, могут существенно отличаться друг от друга. Например, $\cos\frac{\pi n}{2}$ для нечетных целых значений n равен нулю, для четных равен $(-1)^{n/2}$, а для вещественных n это выражение принимает множество других значений в пределах [–1, 1]. Таким образом, явное указание свойств символьных переменных может оказать существенное влияние на полученный программой результат.

Чтобы задать свойства символьной переменной или целому выражению, в MuPAD используют функцию `assume`, которую вызывают следующим образом:

`assume``(<свойство>)`

`<Свойство>` символьной переменной или выражения, входящего в состав другого выражения, можно задать двумя способами: 1) указав принадлежность переменной (выражения) к определенному числовому множеству; 2) записав неравенство, которому удовлетворяют значения переменной (выражения). Для обозначения широко распространенных в математике числовых множеств в MuPAD введены следующие обозначения: `C_` – комплексные, `R_` – вещественные, `Q_` – рациональные, `Z_` – целые, `N_` – натуральные (положительные целые) числа. Используя линейные преобразования, из этих множеств можно получить еще ряд числовых множеств. Например, `R_*I` – мнимые числа, `2*Z_` – четные целые, `2*Z_+1` – нечетные целые, `b*Z_` – числа, кратные `b`, и т.д. При вызове функции `assume <свойство>` символьной переменной или выражения можно задать одним из следующих способов:

```
assume(<выражение>, <множество>)
assume(<выражение> in <множество>)
assume(<неравенство>)
```

Первый способ используется для указания свойств ЛЮБЫХ объектов (в том числе массивов и матриц). Два другие способа можно использовать только для указания свойств СКАЛЯРНЫХ (неиндексированных) переменных или выражений. В последнем случае `<неравенство>` должно содержать идентификатор переменной,

свойство которой задается. Например, команда `assume(n in Z_)` или `assume(n, Z_)` указывает, что `n` – положительное целое число; команда `assume(x <= 5)` указывает, что значения переменной `x` принадлежат интервалу (–∞, 5].

Кроме того, `<свойство>` можно записать с помощью нескольких неравенств, соединив их логическими союзами (`and`, `or`), или с помощью множества и неравенства, также соединенных соответствующим логическим союзом. Например, команда `assume(x > 1 and x <= 5)` указывает, что значения переменной `x` принадлежат интервалу (1, 5]; команда `assume(n in Z_ and n > 10)` указывает, что `n` – целое число, большее 10.

После того как свойства переменной или выражения заданы с помощью функции `assume`, все последующие вычисления выполняются с учетом этих свойств. Результаты предыдущих вычислений при этом остаются неизменными.

Свойства переменной или выражения в программе остаются неизменными до тех пор, пока они не будут переопределены. Переопределение свойств может осуществляться путем введения дополнительных ограничений с помощью функции `assumeAlso` следующим образом:

`assumeAlso``(<свойство>)`

Дополнительные свойства с помощью функции `assumeAlso` задаются так же, как и основные свойства с помощью функции `assume` (путем указания числовых множеств или ограничивающих неравенств). Например, команды `assume(x > 1); assumeAlso(x <= 5)` тоже указывают, что значения переменной `x` принадлежат интервалу (1, 5].

Отменить все установленные ранее свойства (ограничения свойств) переменной или выражения можно с помощью функции `unassume` следующим образом:

`unassume``(<переменная>)`

`unassume``([<переменная1>, <переменная2>, …])`

После этого все установленные ранее ограничения на свойства указанной переменной (или выражения) снимаются, и далее считается, что эта переменная принадлежит множеству комплексных чисел (по умолчанию). Например, команды `assume(k in Z_)` … `unassume(k)` указывают, что сначала k – целое число, а затем (после нескольких вычислений) – комплексное.

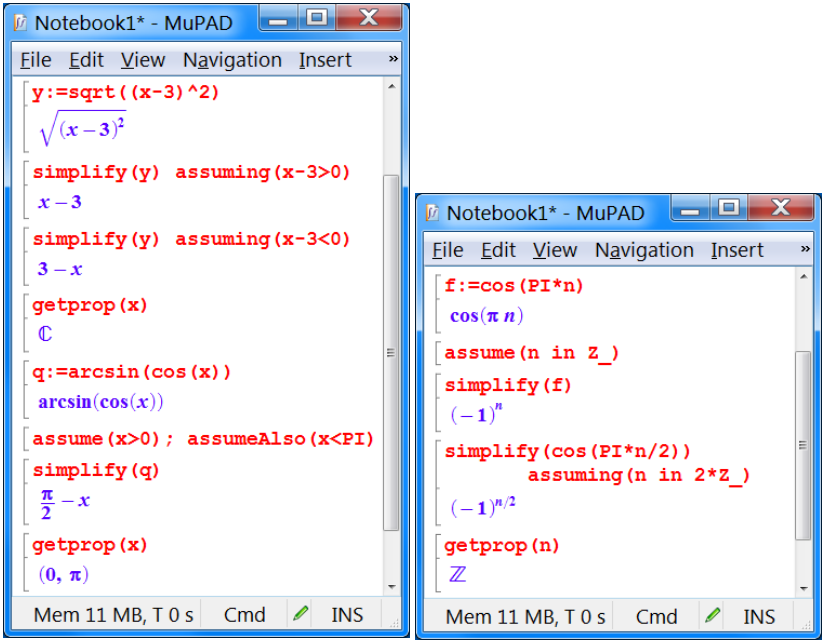


Рис. 4.17. Использование ограничения свойств переменных для упрощения результатов вычислений

С помощью функции `getprop` можно отобразить в поле вывода текущие свойства любой переменной или выражения. Для этого функцию `getprop` вызывают следующим образом:

`getprop``(<переменная>)`

В результате выполнения функции `getprop` в поле вывода отображается множество или интервал значений, которые установлены ранее в качестве свойств для указанной `<переменной>` или выражения. Например, для переменной `z`, свойства которой ранее в программе не были заданы, результатом выполнения команды `getprop(z)` будет являться множество комплексных чисел, которое в поле вывода обозначается символом $\mathbb{C}$. Следует отметить, что обозначения наиболее распространенных в математике множеств в поле вывода отличаются от зарезервированных для их обозначения идентификаторов. Идентификаторам множеств

C_, R_, Q_, Z_, N_ в поле вывода соответствуют следующие обозначения: ℂ, ℝ, ℚ, ℤ, ℕ.

В отличие от функций `assume`, `assumeAlso`, `unassume`, которые задают свойства переменных (выражений) для всех последующих математических преобразований в программе, функция `assuming` позволяет назначить переменной (выражению) некоторое свойство временно (на время выполнения какой-то одной <команды>). Для этого соответствующую <команду> записывают следующим образом:

<Переменная> **:=** <команда> **assuming**(<свойство>)

В качестве <команды> может использоваться любая функция преобразования математического выражения, уравнения, неравенства, списка и т.д. Полученное в результате выполнения <команды> выражение (уравнение, неравенство, список и т.д.) рекомендуется присвоить какой-либо <переменной>. Это облегчит дальнейшее использование в программе полученного результата. Если далее в программе результат выполнения <команды> использовать не планируется, то эту <команду> можно использовать без присвоения. Тогда параметр <переменная> и оператор присвоения **:=** можно опустить.

На рис. 4.17 приведен пример использования свойств переменных, заданных с помощью функций `assume`, `assumAlso` и `assuming`, для упрощения выражений, а также пример контроля текущих свойств переменных с помощью функции `getprop`.

Глава 5. РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ И ТРАСЦЕНДЕНТНЫХ УРАВНЕНИЙ (НЕРАВЕНСТВ) И ИХ СИСТЕМ

Под ***уравнением*** понимают краткую запись задачи о нахождении таких значений аргументов двух функций, зависящих от одной и той же независимой переменной, при которых значения этих функций равны (одинаковы).

Под ***неравенством***, соответственно, понимают краткую запись задачи о нахождении значений аргументов двух функций, зависящих от одной и той же переменной, при которых значения этих функций находятся в определенном отношении (больше, меньше, неравны и т.д.) друг к другу.

Независимая переменная, значения которой необходимо найти в процессе решения уравнения (неравенства), называется ***неизвестной***. Искомые значения неизвестной называют ***решением уравнения (неравенства)***. Решения уравнения также принято называть ***корнями уравнения***.

В этой главе описаны функции MuPAD, предназначенные для аналитического и численного решения уравнений, неравенств и их систем. Перед решением уравнения (неравенства, системы) рекомендуется обозначить (присвоить) его какой-либо переменной. Это позволит контролировать правильность записи уравнения (неравенства, системы) и облегчит дальнейшее обращение к нему. Например, чтобы обозначить уравнение $x^2 - 6 = 0$ переменной `ur`, в командном поле можно написать: `ur:= x^2-6 = 0`.

Обратите внимание! Значения, возвращаемые каждой из описанных в этой главе функций, рекомендуется присваивать отдельной переменной; это облегчит их дальнейшее использование в программе. Если полученные результаты далее в программе использовать не планируется, то для всех описанных ниже функций допускается вызов без присвоения.

5.1. Аналитическое решение уравнений и неравенств

Для получения решения уравнения (неравенства) в аналитическом виде выполняют последовательность тождественных преобразований

исходного уравнения (неравенства) так, чтобы выразить искомый аргумент через остальные параметры уравнения, считающиеся известными.

В процессе символьных вычислений эти преобразования можно запрограммировать самостоятельно, указывая необходимые операции (добавление констант, домножение и т.д.; см. гл. 4). Кроме того, можно воспользоваться стандартными функциями MuPAD, каждая из которых автоматически выполняет необходимые тождественные преобразования, параллельно анализируя дальнейшую возможность получения решения. Рассмотрим подробнее эти стандартные функции.

Функции `simplify` и `Simplify` выполняют тождественные преобразования, в том числе уравнений и неравенств (см. разд. 4.2). Если с помощью этих функций упростить линейное уравнение (неравенство), то, как правило, в качестве наилучшего упрощения выбирается такое тождественное первоначальному уравнение (неравенство), форма записи которого совпадает с формой записи его решения. Например, если линейное уравнение $2x = 3$ упростить с помощью команды `simplify(2*x = 3)`, то в командном поле получится его решение в виде: $x = \frac{3}{2}$. Получить решение уравнения (неравенства) другого вида (нелинейного) с помощью функций `simplify` или `Simplify` можно только для некоторых простых частных случаев. В остальных случаях следует использовать другие функции MuPAD.

Функция `isolate` (начиная с версии MatLab 2015) позволяет изолировать в уравнении заданную переменную или выражение. Под изоляцией переменной здесь понимается такое представление исходного уравнения, при котором заданная переменная располагается слева от знака равенства, а справа от равенства не встречается. Эти преобразования аналогичны действиям, которые обычно выполняют при нахождении корней уравнения. Чтобы MuPAD осуществил такие тождественные преобразования, функцию `isolate` вызывают следующим образом:

`<Результат> := isolate(<уравнение>, <неизвестная>)`

`<Уравнение>` необходимо записать полностью явным образом или указать идентификатор переменной, которой это `<уравнение>` ранее обозначено. В качестве `<неизвестной>` необходимо указать идентификатор одной из символьных переменных, использованных при записи `<уравнения>`. Например, в результате выполнения команды `isolate(2*x-6 = 0, x)` в поле вывода получится $x = 3$.

Следует помнить, что если уравнение имеет несколько корней, то функция `isolate` позволяет получить только один из них. Если свойства искомой переменной ограничены, то функция `isolate` находит корень, обладающий теми же свойствами. Например, если искомая `<неизвестная>` отрицательна, то найденный функцией `isolate` корень тоже будет отрицательным. Этими свойствами можно воспользоваться, чтобы найти корень уравнения, расположенный на заданном интервале. Например, если необходимо решить уравнение, обозначенное переменной `ur`, относительно переменной `x` на интервале [1, 5), в командном поле можно написать следующий оператор:

```
x2 := isolate(ur, x) assuming (x >= 1 and x < 5)
```

Функция `isolate` возвращает найденное решение в виде уравнения (рис. 5.1):

```
<Неизвестная> = <выражение или число>
```

Если функция `isolate` не может полностью изолировать указанную переменную, то все операнды, содержащие эту переменную, она переносит в левую часть уравнения.

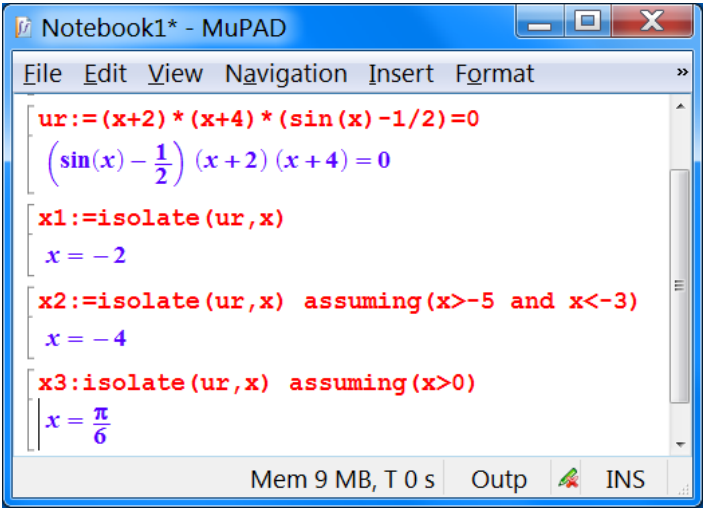


Рис. 5.1. Нахождение корней уравнения с помощью функции `isolate`

Если в исходном уравнении хотя бы одно число записано в десятичном формате, найденное решение автоматически записывается в этом же

формате. Решение, записанное в рациональном виде, можно преобразовать в десятичный формат с помощью функции `float`.

Для решения неравенств функция `isolate` НЕ ПРИМЕНЯЕТСЯ.

Функция `solve` осуществляет поиск всех решений уравнения или неравенства. Для этого функцию `solve` вызывают следующим образом:

<Результат> := **solve**(<уравнение>, <неизвестная>, <опции>)

Под <уравнением> здесь понимают либо явную запись уравнения или неравенства, либо идентификатор, которым оно было ранее обозначено. Если уравнение, которое необходимо решить, записано в виде <выражение> = 0, то вместо параметра <уравнение> при обращении к функции `solve` можно указать <выражение>. То есть если вместо <уравнения> указано <выражение>, не содержащее никаких операций отношения (=, <, >, ≠ и т.д.), то функция `solve` решает уравнение вида: <выражение> = 0.

Если <уравнение> содержит несколько незаданных ранее символьных переменных, необходимо с помощью параметра <неизвестная> указать, относительно какой из них следует решать уравнение (неравенство). Если параметр <неизвестная> не указан, то функция `solve` будет решать <уравнение> относительно всех не заданных ранее символьных переменных, встречающихся в нем. Параметр <неизвестная> задается в виде идентификатора соответствующей символьной переменной. Например, для решения уравнения $ax^2 - b = 0$ относительно переменной `x` можно написать команду: `solve(a*x^2-b = 0, x)`. Кроме идентификатора <неизвестной> можно указать также интервал, на котором необходимо искать решение <уравнения>. В этом случае параметр <неизвестная> записывается следующим образом:

<неизвестная> = <начало> .. <конец>

В качестве <начала> и <конца> интервала можно указать числовые значения или выражения, результатами вычисления которых являются числовые значения.

Если функция `solve` МОЖЕТ решить уравнение, она возвращает решение в виде множества, элементы которого перечисляются в фигурных скобках через запятую. Например: $\{1,\ -\sqrt{3}-1,\ \sqrt{3}-1\}$. Решения неравенств, как правило, записываются в виде набора интервалов.

Например: $(-\infty, -7) \cup (3, \infty)$. Если полученное решение предполагается далее в программе использовать, то его рекомендуется присвоить переменной `<результат>`. В противном случае переменную `<результат>` и оператор присвоения `:=` можно опустить. Если функция `solve` НЕ МОЖЕТ решить уравнение (неравенство), она возвращает выполняемую команду без преобразований. Например, в результате выполнения команды `solve(sin(x) = x^2, x)` в поле вывода будет написано: $\mathrm{solve}(\sin(x) - x^2 = 0, x)$.

В процессе поиска решения функция `solve` сначала преобразует исходное `<уравнение>` к виду `<выражение> = 0`, потом пытается разложить `<выражение>` на простые множители, каждый из которых приравнивает к нулю. В результате вместо одного исходного `<уравнения>` получается множество «упрощенных» уравнений, таких что решение каждого из них является корнем исходного `<уравнения>`. Далее функция `solve` решает каждое из полученных «упрощенных» уравнений. Для этого используются тождественные аналитические преобразования, характер которых зависит от типа функций, входящих в соответствующее «упрощенное» уравнение. В математике существуют методы аналитического нахождения нулей (корней) для большинства элементарных функций. Но для полиномов (многочленов), степень которых больше четырех, а также для комбинаций из нескольких разных элементарных функций аналитических методов нахождения корней не существует. Кроме того, настройки MuPAD по умолчанию таковы, что функция `solve`, вызванная без `<опций>`, ищет корни полиномов в аналитическом виде, только если их степень не выше двух (квадратные уравнения). С помощью `<опций>` в некоторых случаях удается также получить в аналитическом виде корни полиномов третьей и четвертой степени. Корни остальных полиномов находятся численно, для этого функция `solve` вызывает функцию `RootOf`.

Для численного нахождения корней полинома (`<многочлена>`) функцию `RootOf` можно вызвать (без посредничества функции `solve`) следующим образом:

```
<Результат> := RootOf(<многочлен>, <неизвестная>)
```

Если при записи `<многочлена>` используется несколько не заданных ранее символьных переменных, то параметр `<неизвестная>` указывает, относительно какой из них решается уравнение вида `<многочлен> = 0`. Если при записи `<многочлена>` используется только одна не заданная

ранее символьная переменная, то параметр <неизвестная> можно опустить. Функция RootOf возвращает корни уравнения вида <многочлен> = 0. Однако эти корни не будут записаны аналитически. Числовые значения этих корней в десятичном виде можно получить с помощью функции float. Например, в результате выполнения команды res := RootOf(x^3-2*x) переменная res будет содержать корни уравнения $x^3 - 2x = 0$, но в явном виде значения этих корней отображаться не будут. В поле вывода будет написано: RootOf($x^3 - 2x$, x). Однако, выполнив команду float(res), можно получить {–1.414213562, 0.0, 1.414213562}. Следует отметить, что если решать уравнение $x^3 - 2x = 0$ с помощью функции solve, то можно путем аналитических преобразований получить сразу все три корня $\{0, \sqrt{2}, -\sqrt{2}\}$ в рациональном виде (аналитические решения), так как многочлен $x^3 - 2x$ разлагается на множители $(x^2 - 2)\cdot x$, степень каждого из которых меньше трех.

При необходимости функция RootOf может использоваться самостоятельно (как в предыдущем примере). Но чаще всего она применяется для записи некоторых решений уравнения, полученных функцией solve. Это происходит в тех случаях, когда после разложения на простые множители <выражения> остаются множители, являющиеся полиномами степени больше двух. Например, для уравнения $x^3 - 4x^2 - 3 = 0$, обозначенного в программе переменной ur, с помощью команды solve(ur, x) можно получить решение только в виде RootOf($z^3 - 4z^2 - 3$, z).

<Опции> позволяют (если это необходимо) детализировать направление поиска решения <уравнения>. Функция solve для решения одного уравнения (неравенства) может вызываться со следующими опциями:

MaxDegree = n – используется при решении алгебраических (полиномиальных) уравнений. Параметр n может принимать значение от 1 до 4, так как только для этих степеней многочленов существуют аналитические методы вычисления корней уравнения вида <многочлен> = 0. Данная опция указывает, корни многочленов каких степеней (до n включительно) будут вычисляться аналитически. Корни многочленов степени выше n вычисляются численно, и записываются в виде RootOf(<многочлен>). По умолчанию MaxDegree = 2. Например, для уравнения $x^3 - 4x^2 - 3 = 0$,

обозначенного ранее в программе переменной `ur`, с помощь команды `solve(ur, x)` можно получить решение только в виде RootOf($z^3 - 4z^2 - 3, z$), а с помощь команды `solve(ur, x, MaxDegree=3)` это же решение будет записано следующим образом:

$$\left\{\frac{16}{9\sigma_2}+\sigma_2+\frac{4}{3},\ \frac{4}{3}-\frac{\sigma_2}{2}-\frac{8}{9\sigma_2}-\sigma_1,\ \frac{4}{3}-\frac{\sigma_2}{2}-\frac{8}{9\sigma_2}-\sigma_1\right\},$$

where $\sigma_1 = \frac{\sqrt{3}}{2}\left(\frac{16}{9\sigma_2}-\sigma_2\right)\mathrm{i}$ $\sigma_2 = \left(\frac{\sqrt{337}}{6}+\frac{209}{54}\right)^{1/3}$.

`Multiple` – используется только при нахождении корней многочленов, значения которых вычислены явным образом (без функции `RootOf`). Эта опция позволяет вывести информацию о кратности найденных корней.

`PrincipalValue` – позволяет найти единственное решение.

`Domain = <множество>` – позволяет искать только решения, принадлежащие заданному `<множеству>`, например множеству целых чисел.

`IgnoreProperties` – позволяет получить решения, значения которых не согласуются со свойствами той переменной, относительно которой решается уравнение.

`Real` – позволяет найти только вещественные решения уравнения или неравенства.

`IgnoreSpecialCases` – позволяет получить решения, игнорирующие особые случаи, например, обращение в ноль знаменателя или другие особые точки.

`IgnoreAnalyticConstraints` – разрешает применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами; например считать тождеством arcsin(sin x) = x и т.д.

`DontRewriteBySystem` – запрещает в процессе решения уравнения преобразовывать его в тождественное множество уравнений путем разложения на простые множители.

`NoWarning` – подавляет вывод предупреждающих сообщений.

Некоторые примеры использования функции `solve` с опциями и без опций приведены на рис. 5.2.

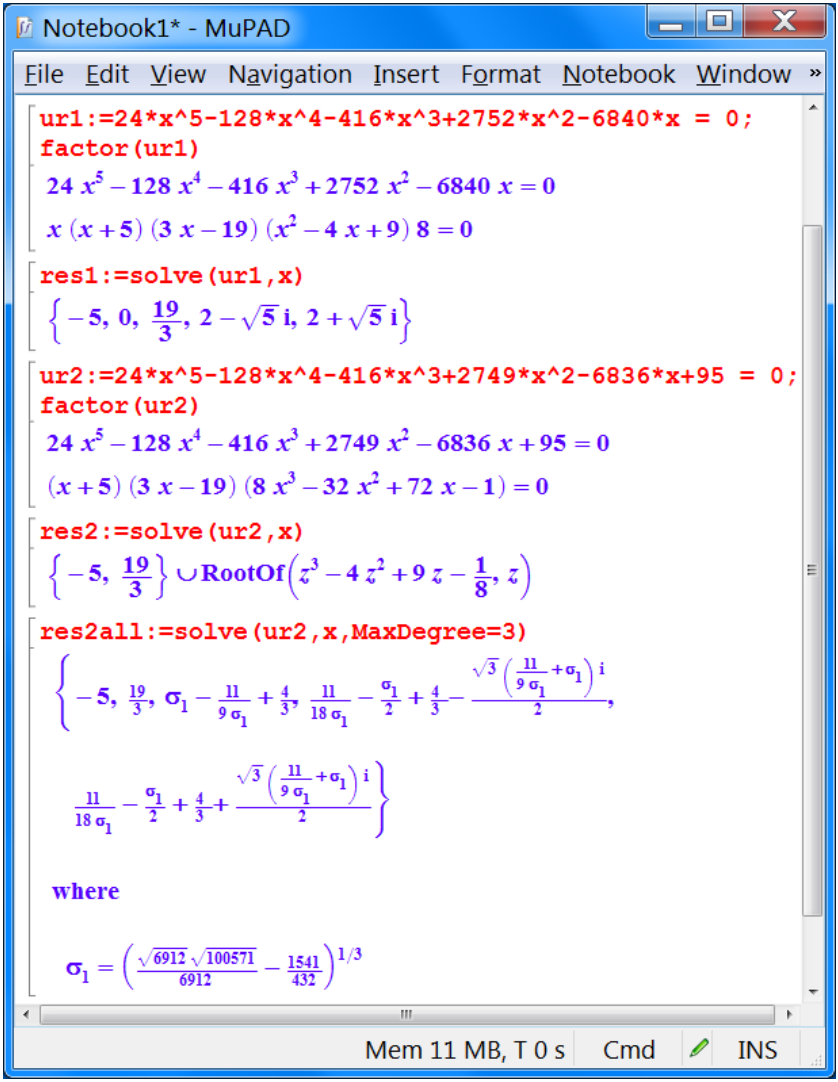


Рис. 5.2. Нахождение корней уравнения с помощью функции `solve`

Раскрывающаяся панель кнопок `Solve equation` $\{x|_{f=0}\}$ на командной панели `Command Bar` позволяет вставить в программу 9 шаблонов для вызова функции `solve`. Эти шаблоны отличаются набором наиболее распространенных опций (табл. 5.1).

Каждый шаблон, вставленный в командное поле с помощью одной из этих кнопок, состоит из нескольких полей, отмеченных символом `#`. Каждое из полей шаблона необходимо заполнить. В поле `#f` необходимо указать `<уравнение>`, в поле `#x` – идентификатор `<неизвестной>`, в поле `#condition` – предполагаемое (временное) свойство `<неизвестной>`.

Таблица 5.1

Шаблоны для вызова функции `solve`

Кнопка	Шаблон	Значение
$\{x\|_{f=0}\}$	`solve(#f, #x)`	Решить уравнение
$\{x\|_{f=0}\}_{\|x\in ?}$	`solve(#f, #x) assuming #x = #condition`	Решить уравнение с предположением
$\{x\|_{f=0}\}_{\subset}$	`solve(#f, #x, IgnoreSpecialCases)`	Решить уравнение без учета особых случаев
$\{x\|_{f=0}\}_{1}$	`solve(#f, #x, PrincipalValue)`	Найти единственное решение уравнения
$\{x\|_{f=0}\}_{\approx}$	`float(solve(#f, #x))`	Решить уравнение аналитически и представить результат в десятичном виде
$\{x\|_{f=0}\}_{\mathbb{R}}$	`solve(#f, #x, Real)`	Найти вещественные решения уравнения
$\{x\|_{f=0}\}_{\mathbb{R}\|x\in ?}$	`solve(#f, #x, Real) assuming #x = #condition`	Найти вещественные решения уравнения с предположением
$\{x\|_{f=0}\}_{\mathbb{R}\subset}$	`solve(#f, #x, Real, IgnoreSpecialCases)`	Найти вещественные решения уравнения без учета особых случаев
$\{x\|_{f=0}\}_{\mathbb{R}\,1}$	`solve(#f, #x, Real, PrincipalValue)`	Найти единственное вещественное решение уравнения

Кроме того, шаблон `solve(#)` можно вставить в текст программы с помощью последовательности раскрывающихся списков `General`

Math → Solve → Exact, расположенной на той же командной панели Command Bar.

Функция solve решает уравнения и неравенства, выполняя тождественные аналитические преобразования. С помощью этой функции можно решать как алгебраические, так и трансцендентные уравнения (неравенства). Большинство уравнений можно преобразовать к виду <функция> = 0. Если <функцию> в таком уравнении (см. рис. 5.2) можно разложить на простые множители, каждый из которых является многочленом не выше четвертой степени, то функция solve (при условии использования соответствующих опций) найдет все корни уравнения независимо от того, многочленом какой степени является <функция> в исходном уравнении. Однако когда <функцию> разложить на простые множители не удается, solve сможет найти все корни уравнения, только если <функция> является многочленом не выше четвертой степени (с использование опции MaxDegree) или не выше второй степени (без опции MaxDegree).

Все решения неравенства функция solve находит только в том случае, если оно сводится к системе неравенств относительно полиномов не выше второй степени каждое.

Так как многие трансцендентные уравнения и неравенства аналитически не решаются, использование функции solve для их решения может не дать никакого результата, либо будут найдены только некоторые из множества решений.

Проверить, действительно ли полученные решения удовлетворяют исходному уравнению (неравенству), как известно, можно, подставив каждое из решений в это уравнение (неравенство). Если при этом уравнение (неравенство) обращается в тождество, то полученные значения действительно являются решениями уравнения (неравенства).

Чтобы подставить какое-либо значение в уравнение (неравенство), можно использовать функции val, evalAt, но лучше – функцию subs (см. разд. 4.1) следующим образом:

```
<Выражение> := subs(<исходное уравнение>,
                    <неизвестная> = <значение>, <опция>)
```

После подстановки значения, соответствующего предполагаемому решению, исходное уравнение (неравенство) превращается в логическое выражение. Например, после выполнения команды subs(x-5=0, x=2) в поле вывода получится логическое выражение: –3 = 0. Логическое

выражение может принимать одно из трех значений: TRUE, FALSE или UNKNOWN (с англ. *истина*, *ложь* или *неизвестно* соответственно). Если логическое выражение, полученное при подстановке значения в исходное уравнение (неравенство), приняло значение TRUE, то данное значение действительно является решением уравнения (неравенства). В некоторых простых случаях истинность полученных логических выражений можно оценить на глаз. В остальных случаях для проверки истинности логических выражений можно воспользоваться функцией `is`, которую вызывают следующим образом:

`is``(<логическое выражение>)`

Например, в результате выполнения команд `ur:= x^2-5=0: proverka:= subs(ur, x = 2): is(proverka)` в поле вывода получится FALSE.

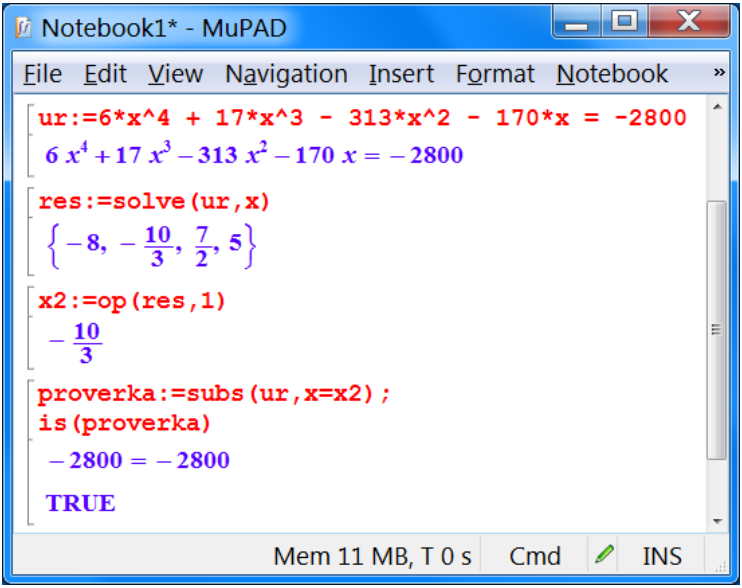


Рис. 5.3. Проверка, является ли полученное значение решением уравнения

Следует отметить, что если при нахождении решения уравнения вида `<выражение> = 0` в функцию `solve` подставлялось только `<выражение>`, при подстановке в него найденного значения получится не

логическое, а арифметическое выражение, значение которого необходимо сравнить с нулем: `is(<выражение> = 0)`.

Например: `ur := x^2 - 5: proverka := subs(ur, x = 2): is(proverka = 0)`.

Функция `solve` возвращает решения уравнения в виде множества. При необходимости элемент из этого множества можно извлечь с помощью функции `op` (см. разд. 4.5) следующим образом: `op(<множество решений>, <номер корня>)`. Следует отметить, что в памяти компьютера элементы множества могут располагаться в порядке, отличающемся от изображенного в поле вывода: в памяти компьютера корни уравнения располагаются в том порядке, в котором были найдены, а в поле вывода они упорядочены по возрастанию (рис. 5.3).

Пример, в котором для проверки, является ли одно из полученных значений решением уравнения, кроме функции `solve` использованы также функции `op`, `subs` и `is`, приведен на рис. 5.3.

Чтобы повысить эффективность решения уравнений, бывает важно предварительно локализовать каждый из корней (найти отрезки, на которых расположено по одному корню). Если все корни уравнения вида `<функция> = 0` вещественные (или необходимо найти только вещественные корни), то для локализации (отделения) корней можно построить график `<функции>` и визуально определить, на каких отрезках функция пересекает ось абсцисс. Кроме того, если `<функция>` является многочленом, то для локализации его вещественных корней можно воспользоваться функций `polylib::realroots`, которую вызывают следующим образом:

`<Переменная> :=` **`polylib::realroots`**`(<полином>)`

В результате `<переменной>`, расположенной слева от оператора `:=`, будет присвоен список интервалов, на каждом из которых располагается по одному корню. Некоторые из этих интервалов могут иметь нулевую длину. Например, для полинома $6x^4 + 17x^3 - 313x^2 - 170x + 2\,800$, обозначенного в программе переменной `y`, выполнение команды `polylib::realroots(y)` позволяет получить следующий список интервалов: [[–8, –8], [–8, 0], [0, 4], [4, 8]]. С помощью дополнительного положительного числового параметра можно ограничить максимальную длину найденных интервалов.

Каждый из полученных с помощью функции `polylib::realroots` отрезков можно далее использовать при вызове функции

`isolate` или `solve` для нахождения значения корня на этом интервале. Для этого границы соответствующих отрезков необходимо сначала извлечь из полученного списка, например с помощью функции `op` (см. разд. 4.5).

5.2. Численное решение уравнений

Для управления точностью численных расчетов в MuPAD используется переменная среды `DIGITS`. Она задает количество точных (верных) значащих цифр, которые должна получить функция, выполняющая численные расчеты, результаты которых всегда записываются в десятичном виде. По умолчанию `DIGITS = 10`. Значение этой переменной можно изменять в пределах от 1 до 2^{29} (только целые). Команда `delete DIGITS` возвращает этой переменной значение, установленное по умолчанию. В поле вывода числовые значения, записанные в десятичном формате, всегда содержат только точные (верные) значащие цифры, т.е. содержат столько значащих цифр, сколько задано с помощью переменной `DIGITS`.

Способы решения уравнения вида `<функция> = 0` существенным образом зависят от вида функции. Аналитические методы решения таких уравнений разработаны в основном для элементарных функций (sin, cos, exp и т.д.), а также для полиномов не выше четвертой степени. Уравнения, содержащие полиномы более высоких степеней или комбинации функций разного вида, решаются численно, т.е. путем многократного вычисления значений входящих в уравнение функций.

Однако для некоторых видов функций разработаны специальные методы численного поиска корней уравнения, учитывающие свойства этих функций. Так, например, для численного поиска корней полиномов существует метод Лагерра. В MuPAD этот метод реализован в функции `numeric::polyroots`, которую вызывают следующим образом:

`<Переменная> := numeric::polyroots(<полином>, <опции>)`

Здесь `<полином>` – многочлен одной переменной вида: $a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$, где a_n, a_{n-1}, a_{n-2}, … a_1, a_0 называются коэффициентами многочлена (полинома). Коэффициенты полинома могут быть комплексными и (или) вещественными. Если для записи коэффициентов полинома используются символьные переменные, то их значения должны быть ранее заданы в программе. В качестве `<полинома>` можно использовать соответствующее полиномиальное выражение или

идентификатор переменной, которой это выражение было ранее присвоено.

Например, чтобы найти корни многочлена $2x^5 - 3x^3 + 6$ можно написать команду `numeric::polyroots(2*x^5-3*x^3+6)`.

Функция `numeric::polyroots` находит ВСЕ корни многочлена (полинома), как вещественные, так и комплексные. Функция `numeric::polyroots` всегда возвращает (в виде списка `[ ]`) такое количество корней, какова степень полинома. Значения кратных корней в списке указываются столько раз, какова кратность корня. Значение каждого корня записывается в десятичном виде. Точность вычисления корней определяется переменной среды `DIGITS`, которая задает количество точных (верных) значащих цифр в найденных решениях. По умолчанию `DIGITS = 10`. Значение этой переменной можно изменять в любом месте программы независимо от используемых в ней функций. Например, для полинома $(2x - 3)(3x + 4)$ в результате выполнения команды `numeric::polyroots((2*x-3)*(3*x+4))` в поле вывода получится список [1.5, –1.333333333].

`<Опции>` не являются обязательным параметром функции `numeric::polyroots`, но позволяют детализировать поиск корней многочлена.

`FixedPrecision` – позволяет выполнить быстрый поиск корней с точностью два десятичных знака после запятой. Не рекомендуется использовать эту опцию, если многочлен имеет близко расположенные или кратные корни.

`Factor` – сначала выполняется аналитическое разложение выражения (многочлена) на простые множители, после этого выполняется численный поиск корней для каждого множителя отдельно. Эта опция позволяет исключить повторный поиск кратных корней.

`NoWarning` – позволяет блокировать вывод предупреждающих сообщений.

С помощью функции `numeric::polyrootbound` можно численно определить симметричный относительно начала координат интервал, внутри которого находятся все корни многочлена. Для этого функцию `numeric::polyrootbound` вызывают следующим образом:

`<Переменная> :=` **`numeric::polyrootbound`**`(<полином>)`

В качестве параметра <полином> можно использовать соответствующее полиномиальное выражение или идентификатор переменной, которой это выражение было ранее присвоено.

Функция `numeric::polyrootbound` возвращает ОДНО ЧИСЛО, такое, что каждый корень полинома по модулю меньше этого числа. Это значение полезно использовать при построении графика полиномиальной функции или при поиске корней многочлена с помощью функции `numeric::solve`.

Например, для полинома $(x + 5)(x - 4)(x - 3)$ в результате выполнения команды `numeric::polyrootbound((x+5)*(x-4)*(x-3))` в поле вывода получится число 6.744138176.

Если необходимо найти ТОЛЬКО ВЕЩЕСТВЕННЫЕ нули (корни) функции одной переменной произвольного вида, следует сначала отделить (изолировать) интервалы, на каждом из которых находится только один вещественный корень этой функции, а затем найти (уточнить) значение каждого корня (рис. 5.4).

Чтобы численно найти интервалы, на каждом из которых <функция> одной переменной имеет только один корень, в MuPAD можно использовать функцию `numeric::realroots`, которую вызывают следующим образом:

```
<Переменная> := numeric::realroots(<функция>,
      <неизвестная> = <начало> .. <конец>, <точность>,
                                                <опция>)
```

В качестве <функции> можно вызвать объявленную ранее функцию одной переменной с символьными аргументами или указать математическое выражение, содержащее одну не заданную ранее символьную переменную, или идентификатор переменной, которой это выражение ранее обозначено. Параметр <функция> является единственным обязательным параметром для функции `numeric::realroots`. Эту функцию следует применять только для таких <функций>, для которых справедлива интервальная арифметика, а именно для <функций>, содержащих любые арифметические операции с многочленами и любыми из следующих элементарных и специальных функций: `abs`, `arccos`, `arccosh`, `arccoth`, `arccot`, `arccsc`, `arccsch`, `arcsec`, `arcsech`, `arcsin`, `arcsinh`, `arctan`, `arctanh`, `arg`, `beta`, `ceil`, `cos`, `cosh`, `cot`, `coth`, `csc`,

csch, dirac, exp, floor, gamma, Im, ln, Re, round, sec, sech, sin, sinh, sqrt, tan, tanh, trunc.

Функция numeric::realroots возвращает список [ ] интервалов, на которых <функция> предположительно имеет корни. Гарантируется, что вне указанных интервалов <функция> корней не имеет. Если <функция> является полиномом, то на каждом из найденных интервалов <функция> гарантированно имеет по одному корню.

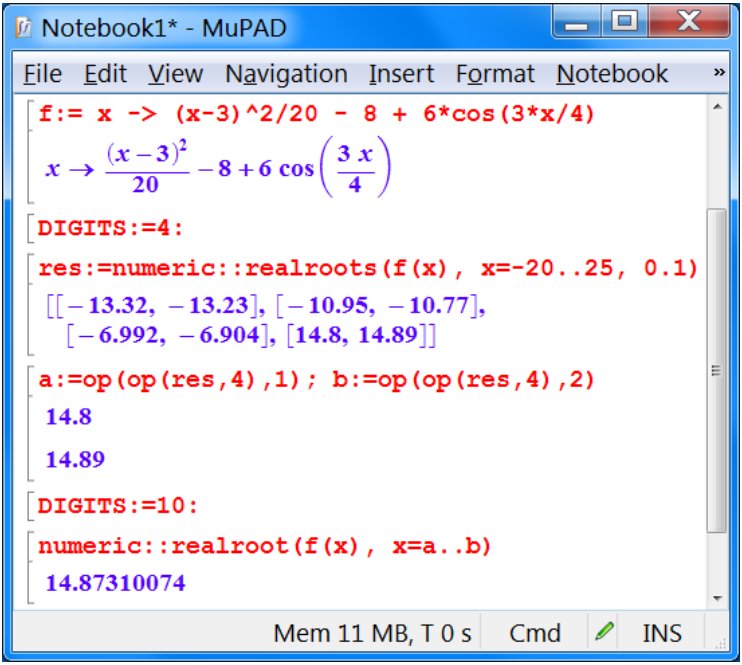


Рис. 5.4. Численное отделение и уточнение вещественных корней

Параметр <неизвестная> = <начало> .. <конец> указывает, на каком интервале следует искать корни уравнения вида <функция> = 0. <Неизвестная> должна быть единственной не заданной ранее символьной переменной, используемой для записи аналитического вида <функции>. В качестве <начала> и (или) <конца> интервала можно использовать вещественные числа, а также идентификаторы

бесконечности (-infinity или infinity соответственно). Здесь под infinity понимают число 10^5.

Параметр <точность> ограничивает максимальную длину найденных интервалов. По умолчанию (если <точность> явным образом не указана) длина найденных функцией numeric::realroots интервалов не превышает 0,01.

Функция numeric::realroots может использоваться только с одной опцией Merge (слияние), которая может принимать значение TRUE или FALSE. Если Merge = TRUE, функция numeric::realroots выполняет слияние смежных интервалов. По умолчанию для полиномиальных функций Merge = TRUE, для остальных – Merge = FALSE.

Пример использования функции numeric::realroots приведен на рис. 5.4.

Чтобы численно получить ЗНАЧЕНИЕ ОДНОГО ВЕЩЕСТВЕННОГО КОРНЯ уравнения вида <функция> = 0 с заданной точностью на заданном интервале, в MuPAD можно использовать функцию numeric::realroot, которую вызывают следующим образом:

<Переменная> := **numeric::realroot**(<функция>,
<неизвестная> = <начало> .. <конец>, <опция>)

В качестве <функции> можно вызвать объявленную ранее функцию одной переменной с символьным аргументом или указать математическое выражение, зависящее от одной не заданной ранее переменной, или идентификатор переменной, которой оно (выражение) было ранее обозначено. Параметр <неизвестная> = <начало> .. <конец> указывает, на каком интервале следует искать корень уравнения вида <функция> = 0. <Неизвестная> должна быть единственной не заданной ранее символьной переменной, используемой для записи аналитического вида <функции>. В качестве <начала> и <конца> интервала можно использовать вещественные числа, но чтобы быть уверенным, что корень на данном интервале действительно есть, в качестве <начала> и <конца> интервала лучше взять один из интервалов, полученных с помощью функции numeric::realroots.

Чтобы задать точность поиска корня на указанном интервале используется переменная среды DIGITS, которая указывает количество точных (верных) значащих цифр искомого решения уравнения.

Функция numeric::realroot возвращает одно числовое (вещественное) значение в десятичном виде с заданным количеством точных

(верных) значащих цифр. Для достижения заданной точности функция numeric::realroot использует комбинацию следующих методов: дихотомии (би-секции), метода секущих и квадратичной интерполяции.

Функция numeric::realroot может использоваться только с одной опцией SearchLevel = n, где в качестве n следует указывать только целые положительные значения. Опция SearchLevel определяет, насколько «настойчиво» необходимо искать корень на заданном интервале. Значение по умолчанию: SearchLevel = 1 (самый низкий уровень). Увеличение SearchLevel ведет к увеличению времени поиска. Не рекомендуется задавать SearchLevel больше 5.

Пример использования функции numeric::realroot приведен на рис. 5.4.

Функция numeric::fsolve численно решает неполиномиальные уравнения вида <функция> = 0 с помощью модифицированного метода Ньютона. Вызов функции numeric::fsolve осуществляется аналогично описанным ранее функциям (в том числе функции solve, предназначенной для аналитического решения уравнений и неравенств):

<Переменная> := **numeric::fsolve**(<уравнение>,
<неизвестная> = <начало> .. <конец>, <опции>)

В качестве параметра <уравнение> указывают либо полную аналитическую запись решаемого уравнения, либо идентификатор переменной, которой это уравнение было ранее присвоено. Если решаемое уравнение имеет вид <функция> = 0, то в качестве параметра <уравнение> можно указать <функцию>, нули которой необходимо найти. Прежде чем решать <уравнение>, функция numeric::fsolve анализирует его. Если <уравнение> является скалярным (нематричным), то функция numeric::fsolve вызывает numeric::realroot.

С помощью параметра <неизвестная>, который является обязательным, необходимо указать, относительно какой переменной следует решать уравнение. Параметр <неизвестная> задается в виде идентификатора соответствующей символьной переменной. Параметр <уравнение> не должен содержать других не заданных ранее символьных переменных, кроме переменной, указанной в качестве параметра <неизвестная>. Кроме идентификатора <неизвестной> можно указать также интервал, на котором необходимо искать решение <уравнения>.

В этом случае параметр <неизвестная> записывается следующим образом:
<неизвестная> = <начало> .. <конец>

В качестве <начала> и <конца> интервала можно использовать числовые значения или выражения, результатом вычисления которых являются числовые значения. Если <уравнение> является полиномиальным, указанный интервал игнорируется.

Например, для численного нахождения на интервале [0; 5] одного решения уравнения $e^x = \pi + x$ можно написать команду numeric::fsolve(exp(x)=PI+x, x=0..5).

Для поиска решения <уравнения> функция numeric::fsolve использует модифицированный метод Ньютона. Для применения этого метода необходимо задать начальное приближение искомого корня. Если для <неизвестной> указан интервал, начальное приближение автоматически выбирается из этого интервала. Если интервал не указан, то начальное приближение автоматически выбирается вблизи нуля. Кроме того, начальное приближение можно задать принудительно, если параметр <неизвестная> указать в виде:
<неизвестная> = <начало>

Помимо этого, для применения модифицированного метода Ньютона перед началом поиска решения указанного <уравнения> функция numeric::fsolve приводит уравнение к виду <функция> = 0 и аналитически вычисляет производную <функции>.

Функция numeric::fsolve возвращает ТОЛЬКО ОДНО решение указанного <уравнения> в десятичном формате.

Набор <опций> функции numeric::fsolve позволяет оптимизировать процесс поиска этого решения.

RestrictedSearch – запрещает проводить поиск за пределами указанного интервала.

UnrestrictedSearch – разрешает проводить поиск за пределами указанного интервала.

Random – выбирает начальное приближение для модифицированного метода Ньютона на указанном интервале случайным образом.

В отличие от всех описанных ранее функцию numeric::solve можно считать универсальным средством решения уравнений произвольного вида.

Она сначала анализирует функции, входящие в уравнение. Если для записи уравнения использовано несколько (больше одной) не заданных ранее символьных переменных, функция numeric::solve вызывает функцию solve, которая пытается решить уравнение аналитически. Если для записи уравнения используется только одна символьная переменная, то уравнение решается численно одним из перечисленных далее методов. Если уравнение содержит только полиномы (многочлены), функция numeric::solve вызывает функцию numeric::polyroots, которая находит все корни многочлена. Если уравнение не удается свести к поиску корней полинома, то функция numeric::solve находит только один корень (рис. 5.5). В этом случае сначала используется метод дихотомии (бисекции) совместно с квадратичной интерполяцией. Если этими методами найти корень не удается, предпринимается попытка найти корень методом Ньютона с помощью функции numeric::fsolve, для которого выполняется символьное вычисление производной. Вызов функции numeric::solve осуществляется аналогично описанным ранее функциям (в том числе функции solve, предназначенной для аналитического решения уравнений и неравенств):

<Переменная> := **numeric::solve**(<уравнение>,
<неизвестная> = <начало> .. <конец>, <опции>)

В качестве параметра <уравнение> указывают либо полную аналитическую запись решаемого уравнения, либо идентификатор переменной, которой это уравнение было ранее присвоено. Если решаемое уравнение имеет вид <функция> = 0, то в качестве параметра <уравнение> можно указать <функцию>, нули которой необходимо найти.

С помощью параметра <неизвестная>, который не является обязательным, можно указать, относительно какой символьной переменной необходимо решать уравнение. Если параметр <неизвестная> не указан, то функция numeric::solve будет решать <уравнение> относительно первой по алфавиту из не заданных ранее символьных переменных, встречающихся в нем. Параметр <неизвестная> задается в виде идентификатора соответствующей символьной переменной. Кроме идентификатора <неизвестной> можно указать также интервал, на котором необходимо искать решение <уравнения>. В этом случае параметр <неизвестная> записывается следующим образом:

<неизвестная> = <начало> .. <конец>

В качестве <начала> и <конца> интервала можно указать числовые значения или выражения, результатом вычисления которых также являются числовые значения. Если <уравнение> является полиномиальным, указанный интервал игнорируется.

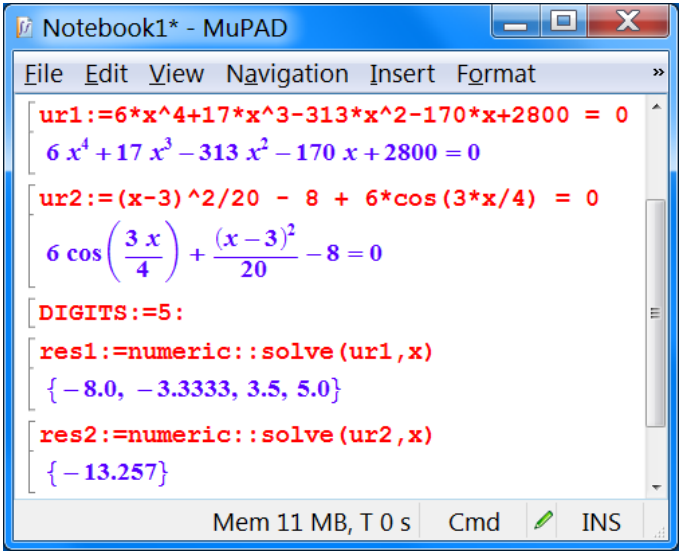


Рис. 5.5. Численное решение полиномиального и неполиномиального уравнений

Если функция `numeric::solve` может решить уравнение, она возвращает решения в виде множества, элементы (корни уравнения) которого – это числовые значения, записанные в десятичном формате в фигурных скобках через запятую.

Набор <опций> функции `numeric::solve` обеспечивает возможность вызова ею всех используемых для решения уравнения функций.

`AllRealRoots` – при использовании этой опции функция `numeric::solve` вызывает функции `numeric::realroots` и `numeric::realroot` для нахождения ВСЕХ ВЕЩЕСТВЕННЫХ КОРНЕЙ уравнения. Вместе с этой опцией можно использовать только опции `Multiple` и `NoWarning`, остальные опции игнорируются. Для ускорения поиска с этой опцией рекомендуется указывать интервал.

`Multiple` – позволяет вывести информацию о кратности найденных корней. Эта опция учитывается только для полиномиальных уравнений.

`FixedPrecision` – при использовании этой опции автоматически вызывается функция `numeric::realroots` для быстрого поиска корней с малой точностью. Эта опция учитывается только для полиномиальных уравнений.

`Factor` – перед вызовом функции `numeric::realroots` выполняется символьное разложение уравнения на простые множители. Эта опция учитывается только для полиномиальных уравнений.

`RestrictedSearch` – запрещает функции `numeric::fsolve` проводить поиск за пределами указанного интервала. Эта опция учитывается только для неполиномиальных уравнений.

`UnrestrictedSearch` – разрешает функции `numeric::fsolve` проводить поиск за пределами указанного интервала. Эта опция учитывается только для неполиномиальных уравнений.

`Random` – случайным образом выбирает начальное приближение для модифицированного метода Ньютона, используемого функцией `numeric::fsolve`. Эта опция учитывается только для неполиномиальных уравнений.

`NoWarning` – подавляет вывод предупреждающих сообщений. Эта опция учитывается только для полиномиальных уравнений.

Пример использования функции `numeric::solve` для численного нахождения корней уравнений различного типа без опций приведен на рис. 5.5.

Кнопка `Solve equation numerically` $\{x|_{f\approx 0.0}\}$, расположенная на раскрывающаяся панели кнопок `Solve equation` $\{x|_{f=0}\}$ на командной панели `Command Bar` позволяет вставить шаблон для вызова функции `numeric::solve`. Вставляемый с помощью этой кнопки шаблон содержит два поля, отмеченных символом #, и в командном поле программы имеет вид: `numeric::solve(#f, #x)`. Каждое из полей шаблона необходимо заполнить. В поле `#f` необходимо указать `<уравнение>`, в поле `#x` – идентификатор `<неизвестной>` в соответствии с описанными выше правилами.

Кроме того, шаблон `numeric::solve(#)` можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Solve` → `Numeric`, расположенной на той же командной панели `Command Bar`.

5.3. Решение систем уравнений, неравенств

Под ***системой уравнений (неравенств)*** понимают одновременное решение нескольких уравнений (неравенств), которые записаны с помощью разных функций нескольких (одних и тех же) аргументов (независимых переменных). ***Решением системы уравнений (неравенств)*** называется такое множество значений этих аргументов, при одновременной подстановке которых во все уравнения (неравенства) каждое из них обращается в тождество. Независимые переменные (аргументы), значения которых необходимо найти при решении системы уравнений (неравенств), называются ***неизвестными***.

Среди систем уравнений (неравенств) принято выделять системы линейных уравнений (неравенств). В этих системах каждое уравнение выражается только через полином первой степени относительно каждой независимой переменной. Для систем такого вида в математике разработаны особые методы как аналитического, так и численного решения. В MuPAD для аналитического решения системы линейных уравнений используют функцию `linsolve`, а для численного – функцию `numeric::linsolve`. Правила вызова этих функций одинаковы:

```
<Переменная> := linsolve(<уравнения>, <неизвестные>,
                                                <опции>)
<Переменная> := numeric::linsolve(<уравнения>,
                                <неизвестные>, <опции>)
```

И в том и в другом случае в качестве параметра `<уравнения>` необходимо указать совокупность решаемых уравнений (рис. 5.6), которую можно записать через запятую либо в виде списка (в квадратных скобках), либо в виде множества (в фигурных скобках). Например: `{x=y-1, x+y=8}` или `[x=y-1, x+y=8]`. В качестве параметра `<неизвестные>` необходимо указать совокупность независимых символьных переменных, значения которых необходимо найти в процессе решения системы. Эту совокупность также можно записать через запятую либо в виде списка (в квадратных скобках), либо в виде множества (в фигурных скобках). Например: `{x, y}` или `[x, y]`.

Каждая из функций `linsolve` и `numeric::linsolve` возвращает список значений неизвестных, соответствующих решению системы, в том же порядке, в котором неизвестные были перечислены при вызове этих функций, т.е. в виде:

```
[<неизвестная1> = <значение1>,
                    <неизвестная2> = <значение2>, …]
```

Если поиск решения выполняется аналитически, `<значение1>`, `<значение2>`, … могут быть записаны в виде математических выражений или числовых значений в рациональном формате.

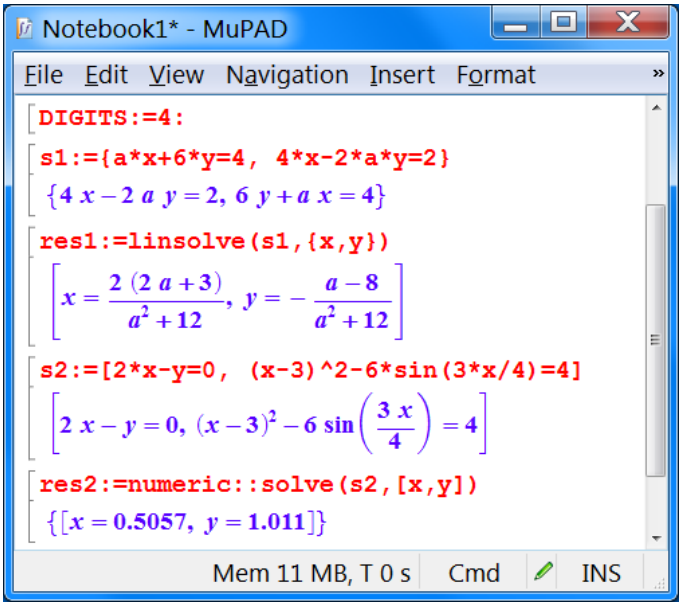


Рис. 5.6. Аналитическое решение системы линейных уравнений и численное решение системы нелинейных уравнений

Для конкретизации направления поиска аналитического решения системы линейных уравнений функцию `linsolve` можно вызвать только с тремя опциями: `Domain = <множество>` (искать только решения, принадлежащие указанному множеству); `Normal = TRUE` по умолчанию (перед выводом приводит решения к нормальному виду); `ShowAssump-`

tions (позволяет вывести расширенную информацию о полученном решении).

Шаблон linsolve({#}) для вызова функции linsolve можно вставить в текст программы с помощью последовательности раскрывающихся списков General Math → Solve → Linear System, расположенной на командной панели Command Bar. Пример использования функции linsolve для аналитического решения системы линейных уравнений (без опций) приведен на рис. 5.6.

Для конкретизации направления поиска численного решения системы линейных уравнений функцию numeric::linsolve можно вызвать с четырьмя опциями: Hard, HardwareFloats, Soft, SoftwareFloats (выполнять вычисления в десятичном формате с плавающей Soft, SoftwareFloats или с фиксированной Hard, HardwareFloats запятой); Symbolic (поиск решения в символьном виде, если коэффициенты уравнения содержат символьные параметры); ShowAssumptions (позволяет вывести расширенную информацию о полученном решении); NoWarning (подавляет вывод предупреждающих сообщений).

Поиск численных решений выполняется с точностью, которая задается переменной среды DIGITS (количеством точных значащих цифр в десятичной записи чисел при численных расчетах).

Системы уравнений, все функции в которых являются полиномами и хотя бы одна из них является полиномом степени больше единицы, называются системами полиномиальных уравнений. Решение таких систем выполняется численно и в MuPAD осуществляется с помощью функции numeric::polysysroots, которую вызывают следующим образом:

<Переменная> := **numeric::polysysroots**(<уравнения>, <неизвестные>, <опции>)

Параметры <уравнения> и <неизвестные> имеют тот же смысл и указываются по тем же правилам, как и для функций linsolve или numeric::linsolve.

Функция numeric::polysysroots возвращает множество (в фигурных скобках) всех решений системы уравнений. Каждый элемент этого множества – одно из решений системы. Например: {[x=0, y=-1], [x=-0.4534, y=-1.2056]}.

Функцию numeric::polysysroots можно также применять для численного поиска корней одного полиномиального уравнения. В этом

случае функция numeric::polysysroots возвращает множество всех корней полинома.

Поиск численных решений выполняется с точностью, которая задается переменной среды DIGITS (количеством точных значащих цифр в десятичной записи чисел при численных расчетах).

Функцию numeric::polysysroots можно вызвать только с одной опцией: NoWarning (подавляет вывод предупреждающих сообщений о нестабильности решения).

Системы уравнений, хотя бы одно из которых не является полиномиальным, можно решать численно. В MuPAD это осуществляется с помощью той же функции numeric::fsolve, которую используют и для решения неполиномиальных уравнений (см. разд. 5.2):

<Переменная> := **numeric::fsolve**(<уравнения>,
<неизвестные>, <опции>)

Здесь параметры <уравнения> и <неизвестные> имеют тот же смысл и указываются по тем же правилам, как для функций linsolve или numeric::linsolve. Кроме того, при вызове функции numeric::fsolve для каждого или некоторых <неизвестных> можно указать интервал, или начальное приближение. Например, <неизвестные> для решения системы с двумя искомыми независимыми переменными можно указать в виде: [x, y], или [x=0..1, y], или [x, y=2], или [x=-3, y =-5..5] и т.д.

Функция numeric::fsolve возвращает только одно решение указанной системы уравнений. Функцию numeric::fsolve для решения системы можно использовать с теми же <опциями>, как и для решения одного уравнения (см. разд. 5.2).

Как и для других численных расчетов, поиск решения системы также выполняется с точностью, которая задается переменной среды DIGITS (количеством точных значащих цифр в десятичной записи чисел при численных расчетах).

Функции solve и numeric::solve также применяются для решения не только уравнений произвольного вида (см. разд. 5.1–5.2), но и систем уравнений произвольного вида:

<Переменная> := **solve**(<уравнения>, <неизвестные>,
<опции>)

```
<Переменная> := numeric::solve(<уравнения>,
                                <неизвестные>, <опции>)
```

Здесь параметры `<уравнения>` и `<неизвестные>` имеют тот же смысл и указываются по тем же правилам, как для функций `linsolve` или `numeric::linsolve`. Обе функции – `solve` и `numeric::solve` – возвращают множество (в фигурных скобках) всех решений системы уравнений. Каждый элемент этого множества – одно из решений системы.

Кроме перечисленных в разд. 5.1 опций функция `solve` имеет также опцию `VectorFormat`, которая позволяет представить решение системы уравнений в матричном виде.

Пример использования функции `numeric::solve` для численного решения системы нелинейных уравнений приведен на рис. 5.6.

Глава 6. РЕШЕНИЕ ЗАДАЧ МАТЕМАТИЧЕСКОГО АНАЛИЗА

Задачи математического анализа включают вычисление сумм и произведений, вычисление пределов, дифференцирование, поиск экстремумов функции, разложение функции в ряд, интегрирование и т.д. Для контроля правильности записи и облегчения дальнейшего обращения к выражениям и функциям, являющимся исходными для этих задач, рекомендуется заранее обозначить (присвоить) их отдельным переменным.

В данной главе описаны функции MuPAD, предназначенные для решения задач математического анализа.

Обратите внимание! Значения, возвращаемые каждой из описанных ниже функций, рекомендуется присваивать отдельной переменной. Это облегчит их дальнейшее использование в программе. Если полученные результаты далее в программе использовать не планируется, то для всех описанных ниже функций допускается вызов без присвоения.

6.1. Вычисление сумм и произведений

В MuPAD функции, позволяющие вычислить сумму или произведение однотипных аналитических выражений, содержащих целочисленную переменную, принимающую разные значения для каждого из складываемых или перемножаемых выражений, имеют одинаковую структуру. Для аналитического вычисления суммы используется функция `sum`, а для произведения – функция `product`. Их вызывают следующим образом:

```
<Переменная> := sum(<выражение>, <индекс> <значения>)
<Переменная> := product(<выражение>,
                                <индекс> <значения>)
```

Параметр `<выражение>` задает аналитический вид слагаемых (или сомножителей), сумму (или произведение) которых необходимо вычислить. `<Выражение>`, как правило, содержит целочисленную переменную, которую называют индексом суммирования (перемножения). Чаще всего слагаемые в сумме (сомножители в произведении) отличаются только значениями индекса, например, `sin(PI*k/6)`. В качестве параметра `<индекс>` при вызове функций `sum` или `product` необходимо

указать идентификатор переменной, имеющей в <выражении> смысл индекса суммирования (перемножения). Например, k.

После <индекса> необходимо указать с помощью параметра <значения>, какие значения будет принимать индекс в процессе вычисления суммы, произведения. <Значения> можно указать двумя способами:

1) как интервал в виде

<индекс> = <начало> .. <конец>

В этом случае <индекс> будет автоматически изменяться в заданных пределах через единицу. Изменение индекса суммирования (перемножения) с другим шагом в MuPAD не предусмотрено. Поэтому <конец> не может быть меньше <начала>. Границы интервала можно задавать как в виде чисел, так и в виде выражений, но таких, чтобы, увеличив <начало> несколько раз на единицу, можно было получить <конец>. Например: k = a .. a + 5 (где a – не заданная ранее символьная переменная). Для изменения индекса суммирования (перемножения) до бесконечности используют стандартный идентификатор infinity. Например: k = 1..infinity;

2) в виде множества

<индекс> in <множество>

В качестве параметра <множество> можно либо явно перечислить в фигурных скобках элементы множества (например: k in {1, 3, 5}), либо использовать функцию, которая возвращает результат в виде множества (например: k in RootOf((x-2)*(x-5), x)), либо указать идентификатор переменной, которой это множество было ранее присвоено. В этом случае шаг изменения <индекса> зависит от разности значений соседних элементов множества и в процессе суммирования может отличаться от единицы, а также может оказаться переменным.

Например, чтобы вычислить сумму $\sum_{k=2}^{5} \sin\frac{\pi k}{6}$ можно в командном поле написать sum(sin(PI*k/6), k = 2..5), тогда в поле вывода получится $\sqrt{3}+\frac{3}{2}$. Если нужно посчитать сумму элементов множества $\{x-1, x, x+2, x+5\}$, то можно написать такую последовательность команд: mnog:={x-1, x, x+2, x+5}: sum(a, a in mnog); в результате в поле вывода получится $4x + 6$. Аналогично можно получить произведение элементов множества.

Если при вызове функций `sum` и `product` параметр <значение> не указан, по умолчанию считается, что индекс суммирования (произведения) изменяется от единицы до значения (<индекс> – 1). Например, результаты вычисления сумм `sum(n, n)` и `sum(k, k=1..n-1)` будут одинаковыми и равными $\sum_{k=1}^{n-1} k = \frac{1}{2} n(n-1)$.

С помощью раскрывающейся панели кнопок `Sum` $\Sigma_n f$, расположенной на командной панели `Command Bar`, в программу можно вставить один из двух шаблонов для аналитического вычисления суммы: `sum(#f, #n)` (кнопка `Indefinite sum`) или `sum(#f, #n=#a..#b)` (кнопка `Definite sum`). С помощью раскрывающейся панели кнопок `Product` $\Pi_n f$, расположенной на командной панели `Command Bar`, в программу можно вставить один из двух шаблонов для аналитического вычисления произведения: `product(#f, #n)` (кнопка `Indefinite product`) или `product(#f, #n=#a..#b)` (кнопка `Definite product`). Каждый вставленный с помощью этих кнопок шаблон содержит поля, отмеченных символом `#`. Каждое из полей шаблона необходимо заполнить. В поле `#f` необходимо указать <выражение>, в поле `#n` – идентификатор <индекса>, в полях `#a` и `#b` – <начало> и <конец> диапазона изменения <индекса> в соответствии с описанными выше правилами.

Кроме того, шаблоны `sum(#f, #n)` и `sum(#f, #n=#a..#b)` для аналитического вычисления суммы можно вставить в текст программы с помощью последовательности раскрывающихся списков `General` `Math` → `Calculus` → `Sum` (или … → `Sum (Definite)`), расположенной на той же командной панели `Command Bar`.

Если вычисление суммы или произведения необходимо отложить, функцию `hold` применяют следующим образом:

```
<Переменная> := hold(sum) (<выражение>,
                                    <индекс> <значения>)
<Переменная> := hold(product) (<выражение>,
                                    <индекс> <значения>)
```

Тогда в поле вывода отображается не результат вычислений, а соответствующая математическая запись задачи через символы Σ или Π соответственно. Этот прием можно использовать для контроля правильности

введенной команды для вычисления сумм и произведений, содержащих слагаемые сложного вида (рис. 6.1).

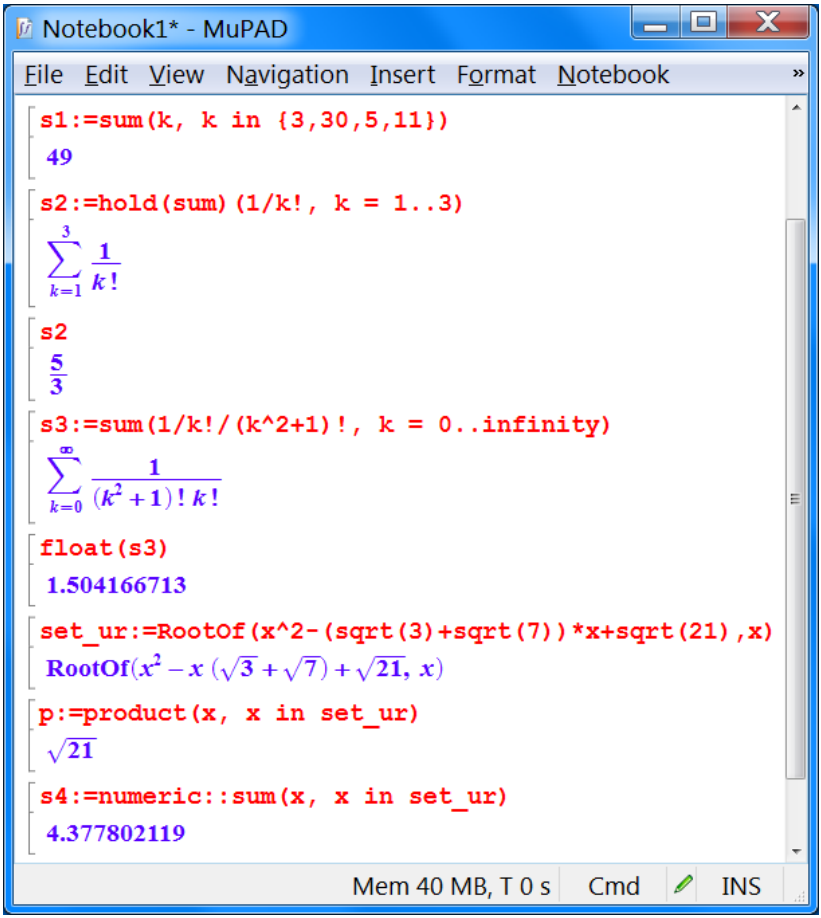


Рис. 6.1. Вычисление сумм и произведений

Если функция `sum` или `product` не может представить результат вычислений в аналитическом виде, то в поле вывода также отображается соответствующая математическая запись задачи через символы Σ или Π,

как при отложенных вычислениях. Числовое значение результата в этом случае можно получить с помощью функции `float` (см. рис. 6.1).

Для численного вычисления суммы в MuPAD используется функция `numeric::sum`, а для численного вычисления произведения – функция `numeric::product`. Их вызывают аналогично функциям `sum` и `product` по тем же правилам:

```
<Переменная> := numeric::sum(<выражение>,
                                <индекс> <значения>)
<Переменная> := numeric::product(<выражение>,
                                <индекс> <значения>)
```

Для численного вычисления суммы или произведения `<выражение>`, с помощью которого задают вид слагаемого или сомножителя, не должно содержать никаких не заданных ранее символьных переменных, кроме `<индекса>`. В этом случае и функция `numeric::sum`, и функция `numeric::product` возвращают числовое значение в десятичном виде.

Например, для численного вычисления произведения $\prod_{k=2}^{5} \sin\frac{\pi}{k}$ в командном поле можно написать `numeric::product(sin(PI/k), k=2..5);` в результате получится число 0.3599434866. Пример численного вычисления суммы приведен на рис. 6.1.

6.2. Вычисление пределов

Для вычисления предельных значений функции (пределов) в MuPAD используется функция `limit`, которую вызывают следующим образом:

```
<Переменная> := limit(<функция>,
                      <аргумент> = <значение>, <опции>)
```

В качестве параметра `<функция>` необходимо указать аналитическое выражение, предельное значение которого необходимо вычислить. Это выражение можно записать непосредственно на месте параметра `<функция>`, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение, либо с помощью вызова объявленной ранее функции с символьными аргументами.

В качестве `<аргумента>` следует использовать идентификатор одной из тех независимых символьных переменных, через которые записана `<функция>`. Параметр `<значение>` указывает, к какому значению необходимо устремить `<аргумент>` при вычислении предела. Если параметр

<значение> не указан, функция `limit` вычисляет предел, устремив <аргумент> к нулю (по умолчанию).

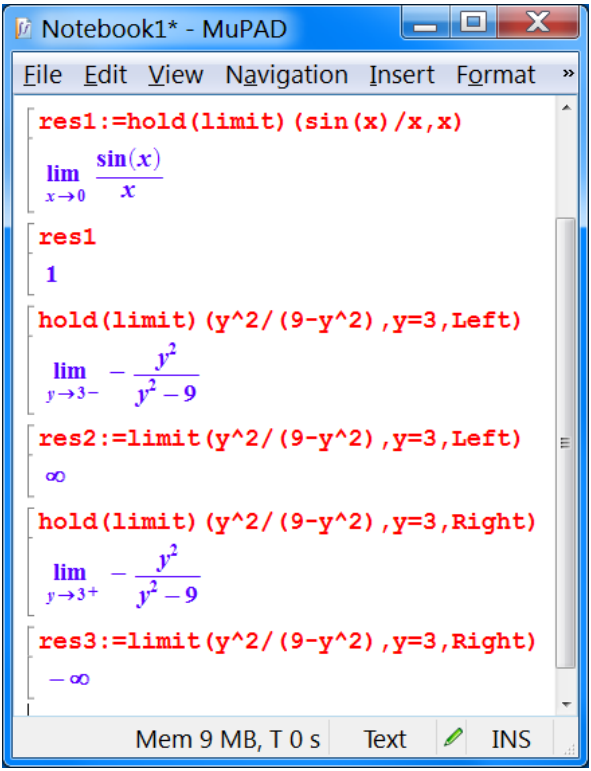


Рис. 6.2. Вычисление пределов

Если искомый предел существует и функция `limit` может его вычислить, она возвращает результат в виде математического выражения или числового значения в рациональном формате. Если искомый предел не существует, то функция `limit` возвращает undefined (с англ. *не определено*).

Например, для вычисления предела $\lim\limits_{x\to\infty}\left(1+\frac{a}{x}\right)^x$ в командном поле можно написать `limit((1+a/x)^x, x = infinity)`, тогда в поле

вывода появится следующий результат: e^a; для вычисления предела $\lim\limits_{x\to 0}\frac{1}{x}$ в командном поле можно написать `limit(1/x, x = 0)`, тогда в поле вывода появится: undefined.

Для детализации условий, при которых необходимо вычислить предел, используются <опции>.

`Left`, `Real`, `Right` – позволяет вычислять односторонний (`Left` – слева, `Right` – справа) или двусторонний предел (`Real`). Если эта опция не указана, то вычисляется двусторонний предел (по умолчанию).

`Intervals` – если <функция> при стремлении <аргумента> к указанному <значению> стремится не к одному, а к нескольким значениям, то эта опция позволяет получить интервал, на котором расположены все эти значения. Если функция стремится к единственному значению, то вместо интервала возвращается значение предела.

`NoWarning` – подавляет вывод предупреждающих сообщений.

На рис. 6.2 приведены примеры вычисления пределов с помощью функции `limit` (с опциями и без них).

С помощью раскрывающейся панели кнопок `Limit` $\lim\limits_{x\to a} f$, расположенной на командной панели `Command Bar`, в программу можно вставить один из трех шаблонов для вычисления предела: `limit(#f, #x=#a)` (кнопка `Limit` – двусторонний предел); `limit(#f, #x=#a, Left)` (кнопка `Limit, left-hand-side` – предел слева); `limit(#f, #x=#a, Right)` (кнопка `Limit, right-hand-side` – предел справа). Каждый вставленный с помощью этих кнопок шаблон содержит поля, отмеченных символом `#`. Каждое из полей шаблона необходимо заполнить. В поле `#f` необходимо указать <функцию>, в поле `#x` – идентификатор <аргумента>, в поле `#a` – <значение> в соответствии с описанными выше правилами.

Кроме того, шаблоны `limit(#f, #x=#a)`, `limit(#f, #x=#a, Left)`, `limit(#f, #x=#a, Right)` для вычисления предела можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Calculus` → `Limit` (или … → `Limit (Left)`, или … → `Limit (Right)`), расположенной на той же командной панели `Command Bar`.

Если вычисление предела необходимо отложить, функцию `hold` применяют следующим образом:

```
<Переменная> := hold(limit) (<функция>,
                              <аргумент> = <значение>, <опции>)
```

Тогда в поле вывода отображается не результат вычислений, а соответствующая математическая запись задачи через символ `lim`. Этот прием можно использовать для контроля правильности введенной команды (см. рис. 6.2).

Если результат вычисления предела невозможно представить в аналитическом виде, то в поле вывода также отображается соответствующая математическая запись задачи через символ `lim`, как при отложенных вычислениях.

6.3. Дифференцирование

Для аналитического выполнения дифференцирования в MuPAD существует две функции – `diff` и `D`, одна из которых предназначена для дифференцирования математических выражений, другая – для дифференцирования функций.

Для дифференцирования математических выражений функцию `diff` вызывают следующим образом:

```
<Результат> := diff(<выражение>, <список переменных>)
```

В качестве параметра `<выражения>` необходимо указать символьное математическое выражение, производную которого необходимо вычислить. Это выражение можно записать непосредственно на месте параметра `<выражение>`, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение. В качестве отдельных операндов в `<выражении>` можно использовать вызов объявленных и не объявленных ранее стандартных и нестандартных функций с символьными аргументами.

В качестве `<списка переменных>` следует через запятую перечислить идентификаторы всех независимых переменных, по которым необходимо продифференцировать указанное `<выражение>`. Например, команда `diff(x+y-x*y, x)` соответствует производной $\frac{\partial^2}{\partial x}(x+y-xy)$, а команда `diff(x+y-x*y, x, y)` соответствует производной $\frac{\partial^2}{\partial x \partial y}(x+$ $+y-xy)$. При этом, так же как в математическом анализе, последовательность, в которой перечисляются независимые переменные, не имеет

значения, так как $\frac{\partial}{\partial x}\frac{\partial}{\partial y}=\frac{\partial}{\partial y}\frac{\partial}{\partial x}$. Если <список переменных> отсутствует (не указан), то функция diff вычисляет производную нулевого порядка, т.е. возвращает исходное непреобразованное <выражение>. Например, в результате выполнения команды diff(x^2+5) получится исходное выражение $x^2 + 5$.

Если по какой-то переменной дифференцирование необходимо выполнить дважды, то идентификатор этой переменной необходимо дважды указать в <списке переменных>. Например, команда diff(sin(x), x, x) соответствует второй производной по dx от $\sin x$. Автоматизировать создание длинной последовательности из нескольких одинаковых переменных можно с помощью оператора $ следующим образом:
<переменная> $ <количество>

Например, команда x $ 3 создает последовательность $x, x, x,$ поэтому команда diff(sin(x), x $ 3) эквивалентна команде diff(sin(x), x, x, x) и соответствует производной $\frac{d^3}{dx^3}\sin x$.

Результатом дифференцирования, выполненного с помощью функции diff, является математическое (символьное или числовое) выражение. Следовательно, если вычисление производной с помощью этой функции выполняется с присвоением, то переменной <результат> будет присвоено полученное выражение (рис. 6.3), даже если исходное выражение записано с использованием вызова каких-либо функций (стандартных или нестандартных).

Кнопка Derivative $\frac{\partial}{\partial x} f$, расположенная на командной панели Command Bar, позволяет вставить в программу шаблон для вызова функции diff, который в командном поле имеет вид: diff(#f, #x), и содержит поля, отмеченных символом #. Каждое из полей шаблона необходимо заполнить. В поле #f следует указать <выражение>, в поле #x – идентификатор одной <переменной> или <список переменных> в соответствии с описанными выше правилами.

Кроме того, шаблоны diff(#f, #x), diff(#f, #x$#n) для вычисления производной можно вставить в текст программы с помощью последовательности раскрывающихся списков General Math → Calculus

→ `Differentiate` (или → `Differentiate n times`), расположенной на той же командной панели `Command Bar`.

Для дифференцирования функций, объявленных ранее в соответствии с правилами MuPAD, используют `D`-функцию, которую вызывают следующим образом:

`<Результат> := D([<список номеров>], <функция>)`

В качестве параметра `<функция>` можно использовать ТОЛЬКО ИДЕНТИФИКАТОР ФУНКЦИИ или математическое выражение из идентификаторов функций (без указания аргументов). При этом дифференцируемые `<функции>` могут быть как объявленными заранее, так и необъявленными.

Если `<функция>` зависит от нескольких аргументов, с помощью параметра `<список номеров>` в квадратных скобках через запятую необходимо указать порядковые номера аргументов функции, по которым нужно вычислить производную. Например, если написана команда `D([3], g)`, то будет вычисляться производная первого порядка по третьему аргументу функции `g`, каким бы идентификатором он не был обозначен (для $g(x, y, z)$ это производная $\partial g/\partial z$); если написана команда `D([1, 3], g)`, то будет вычисляться производная второго порядка – по первому, а потом по третьему аргументу функции `g` (для $g(x, y, z)$ это производная $\partial^2 g/\partial x/\partial z$). Очевидно, если по какому-то аргументу производную необходимо вычислить несколько раз, то номер этого аргумента следует в `<списке номеров>` записать несколько раз. Например, команда `D([3, 3], g)` соответствует производной второго порядка по третьему аргументу функции `g` ($\partial^2 g/\partial z^2$ для $g(x, y, z)$), а команда `D([1, 3, 3], g)` соответствует производной $\partial^3 g/\partial x/\partial z^2$ для $g(x, y, z)$ и т.д. Для автоматизации создания длинных списков из одинаковых номеров можно использовать оператор `$`. Например, команда `3 $ 4` создает последовательность 3, 3, 3, 3, а команда `2 $ 3` – последовательность 2, 2, 2, поэтому для $g(x, y, z)$ команда `D([3 $ 4], g)` соответствует производной $\partial^4 g/\partial z^4$, а команда `D([1, 2 $ 3, 3], g)` – производной $\partial^5 g/\partial x/\partial y^3/\partial z$ и т.д.

Если `<функция>` зависит только от одного аргумента, то `<список номеров>` можно опустить – сокращенная форма `D`-функции. Для этого случая существует краткая запись `D`-функции:

`<Результат> := <функция>'`

Например, для функции `f`, зависящей от одного аргумента, обе команды `D([1],f)` и `D(f)` эквивалентны команде `f'`.

Если от `<функции>`, имеющей только один аргумент, необходимо вычислить производную второго порядка, то сокращенную форму `D`-функции или ее краткую запись применяют дважды; для вычисления производной третьего порядка – трижды и т.д. Например, для функции `f`, зависящей от одного аргумента, обе команды `D([1,1],f)` и `D(D(f))` эквивалентны команде `f''`, а команды `D([1$3],f)` и `D(D(D(f)))` эквивалентны команде `f'''` и т.д. В MuPAD краткая запись `D`-функции часто используется при решении обыкновенных дифференциальных уравнений (см. гл. 8).

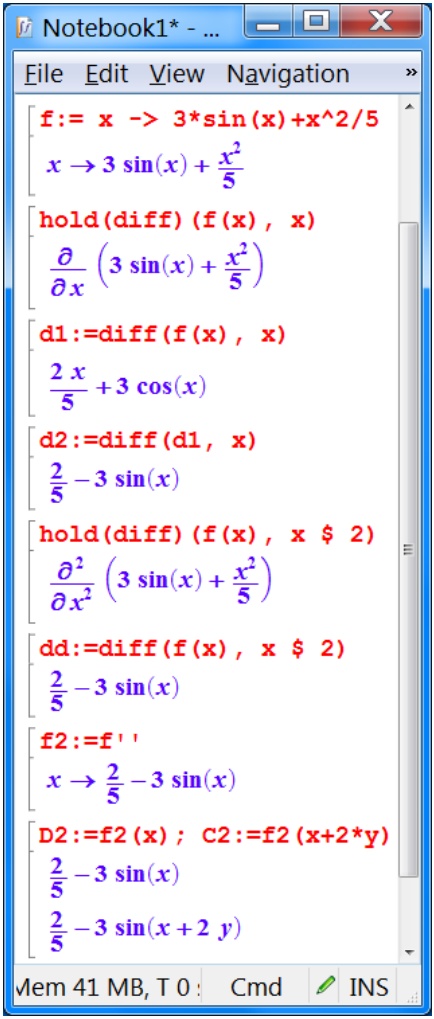


Рис. 6.3. Вычисление производных

Результатом дифференцирования, выполненного с помощью функции `D` (или ее краткой записи), является функция. Следовательно, если вычисление производной с помощью `D`-функции выполняется с присвоением, то происходит объявление новой нестандартной функции, а переменная `<результат>` становится ее идентификатором. Далее в программе параметр `<результат>` можно использовать как имя объявленной функции (см. рис. 6.3), т.е. вызывать эту

функцию следующим образом: <результат>(<список аргументов>). Подставляя разные <списки аргументов>, можно получить разные значения производной.

Можно дополнительно при вызове функции D указать список аргументов дифференцируемой функции следующим образом:

<Результат> := **D(**[<список номеров>],
<функция>**)** (<список аргументов>)

Тогда сначала происходит дифференцирование <функции>, а затем вызов полученной в результате дифференцирования функции с указанными аргументами. В итоге получится математическое выражение, которое будет присвоено переменной <результат>. В этом случае повторный вызов функции, полученной в результате дифференцирования, невозможен. Следует отметить, что результат такого использования функции D эквивалентен результату использованию функции diff. Подставить в такую производную другие значения аргументов можно с помощью функций val, evalAt, subs, subsex (см. разд. 4.1), но лишь в случае, когда <список аргументов> состоит только из незаданных ранее символьных переменных.

Например, для объявленной ранее функции $f(x)$ результаты выполнения команд diff(f(x), x, x), D([1, 1], f)(x), D(D(f))(x) и f''(x) будут одинаковыми; для объявленной ранее функции $g(x, y, z)$ результаты выполнения команд diff(g(x,y,z), x,y) и D([1,2], g)(x,y,z) тоже будут одинаковыми и т.д.

Шаблон D(#f) для вычисления производной можно вставить в текст программы с помощью последовательности раскрывающихся списков General Math → Calculus → Differential Operator, расположенной на командной панели Command Bar.

Приведем еще один пример использования функции D. Если необходимо построить графики функции и ее производной, то программа должна содержать следующую последовательность действий:

1) объявить функцию;

2) вычислить производную от исходной функции;

3) создать два графических объекта типа plot::Function2d (график исходной функции и график производной) и изобразить их одновременно с помощь одной функции plot (см. гл. 3).

Эту последовательность действий можно записать, например, с помощью следующих команд:

```
f := x -> x^3 - 13*x + 12;
f1 := f';
funcf  := plot::Function2d(f(x), Color=RGB::Blue):
funcf1 := plot::Function2d(f1(x), Color=RGB::Red):
plot(funcf, funcf1)
```

Если вычисление производной необходимо отложить, функцию `hold` применяют следующим образом:

```
<Результат> := hold(diff)(<выражение>,
                                  <список переменных>)
<Результат> := hold(D)([<список номеров>], <функция>)
```

Тогда в поле вывода отображается не результат вычислений, а соответствующая математическая запись задачи дифференцирования через символ $\frac{\partial}{\partial ...}$ или D соответственно. Этот прием можно использовать для контроля правильности введенной команды (см. рис. 6.3).

Если результат вычисления производной невозможно представить в аналитическом виде, то в поле вывода также отображается соответствующая математическая запись задачи, как при отложенных вычислениях. Например, если написать команду для вычисления производной от нестандартной функции, которая ранее в программе не была объявлена.

6.4. Исследование функции

Вычисление пределов и производных позволяет выполнять разносторонние исследования математических функций (стандартных и нестандартных), в частности определять точки экстремумов, перегибов, разрывы функции, ее особые точки, а также строить касательные и асимптоты и т.п.

6.4.1. Поиск экстремумов функции

Из математического анализа известно, что в экстремумах (максимумах, минимумах) все первые производные функции обращается в ноль. Поэтому программа поиска экстремумов математической функции, зависящей от одной переменной, должна содержать следующую последовательность действий:

1) объявить функцию;

2) вычислить производную от исходной функции;

3) составить уравнение: приравнять к нулю первую производную исходной функции;

4) решить полученное уравнение (найти все вещественные корни – абсциссы точек экстремума).

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
f := x -> x^3 - 13*x + 12;
f1 := f';
ur := f1(x) = 0;
x1x2 := solve(ur, x)
```

Если производная исходной функции не является полиномом, то для решения уравнения можно использовать функцию `numeric::solve` с опцией `AllRealRoots` (рис. 6.4). Например:

```
x1x2 := numeric::solve(ur, x, AllRealRoots)
```

Если эти точки необходимо отметить на графике функции, то далее следует для каждого значения аргумента функции (корня), найденного при решении уравнения, вычислить значение исходной функции. Если абсциссы точек экстремумов получены в виде множества `{}`, то все значения функции в этих точках можно также получить в виде множества `{}` или списка `[]`. Для этого используют оператор `$`, позволяющий создать последовательность выражений (см. разд. 4.1):

```
<Последовательность> := <выражение> $
                                <переменная> <значения>
```

Здесь `<выражение>` записывается либо явным образом, либо в виде идентификатора переменной, которой оно ранее обозначено, либо через вызов объявленной ранее функции с символьными аргументами. В качестве параметра `<переменная>` можно указать одну из символьных переменных, содержащихся в `<выражении>`.

Тогда после параметра `<переменная>` с помощью параметра `<значения>` необходимо указать, каким образом следует изменять эту переменную при составлении последовательности выражений. Напомним, что одним из способов указать `<значения>` `<переменной>` является использование множества:

```
<переменная> in <множество>
```

В качестве параметра `<множество>` можно либо явно перечислить в фигурных скобках элементы множества (например: `x in {1, 3, 5}`),

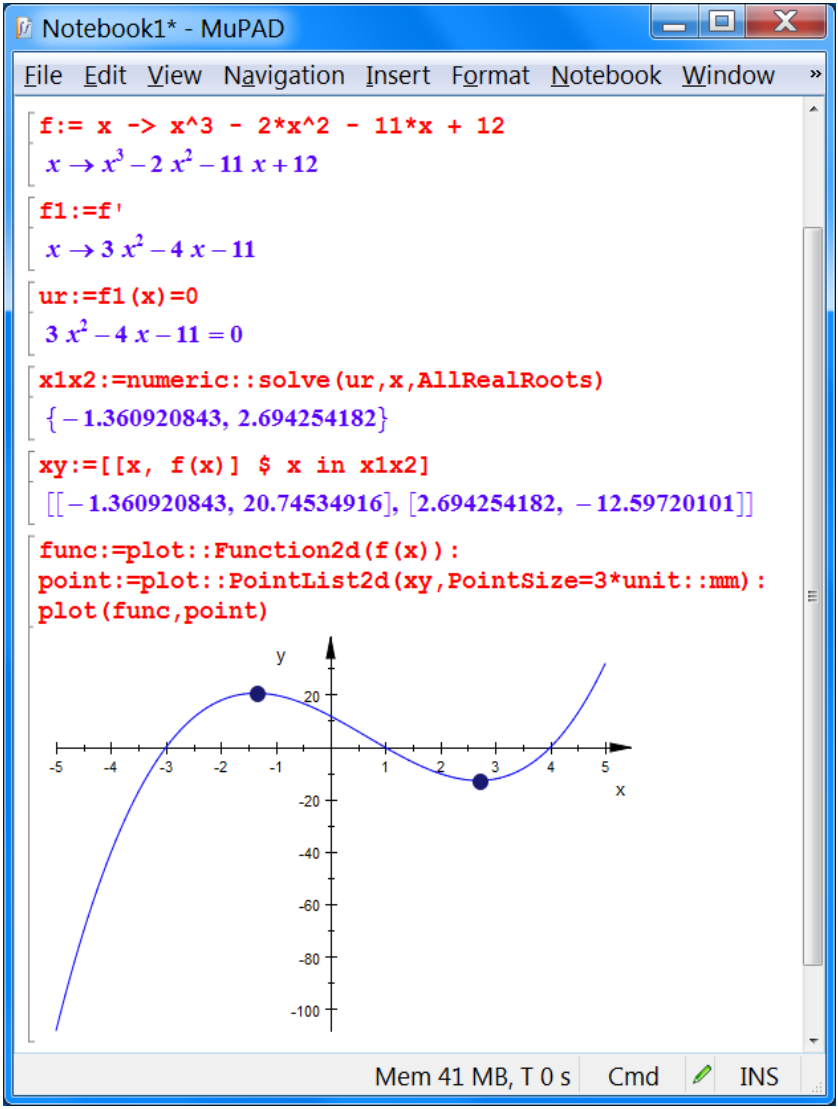


Рис. 6.4. Экстремумы функции

либо использовать идентификатор переменной, которой ранее было присвоено соответствующее множество (например: `x in x1x2`). Для каждого элемента множества вычисляется соответствующее значение `<выражения>`.

Из вычисленных значений `<выражения>` формируется последовательность вида:

`<Выражение1>, <выражение2>, <выражение3>, …`

Полученная последовательность выражений присваивается переменной `<последовательность>`.

С помощью фигурных скобок `{}` можно из `<последовательности>` получить множество следующим образом:

`<Результат> := {<последовательность>}`

Например, с помощью команды `x^2 $ x in {1,2,3}` создается последовательность 1, 4, 9, а с помощью команды `{x^2 $ x in {1,2,3}}` создается множество {1, 4, 9}.

Кроме того, с помощью квадратных скобок `[] <последовательность>` можно преобразовать в список:

`<Результат> := [<последовательность>]`

Например, с помощью команды `x^2 $ x in {1,2,3}` создается последовательность 1, 4, 9, а с помощью команды `[x^2 $ x in {1,2,3}]` создается список [1, 4, 9]. Подробнее о множествах, списках и последовательностях см. в разд. 10.2.

Таким образом, если в результате решения уравнения $f'(x) = 0$ (с помощью команд `f1 := f'; ur := f1(x) = 0; x1x2 := solve(ur, x)`) получено множество решений, которое обозначено, например, переменной `x1x2`, то далее можно получить множество, содержащее значения исходной функции `f(x)` в этих точках с помощью следующей команды:

`f1f2 := {f(x) $ x in x1x2}`

Аналогично можно получить список координат точек экстремумов, если вычислять не `f(x)`, а пары координат `[x, f(x)]` для всех значений аргумента `x` из множества `x1x2`, содержащего абсциссы экстремумов. Для этого можно написать команду:

`xy := [[x, f(x)] $ x in x1x2]`

Точки с этими координатами можно изобразить на графике функции (см. рис. 6.4). Для этого необходимо создать два графических объекта (`plot::Function2d` – график исходной функции; `plot::Point-`

`List2d` – дискретный набор точек) и воспроизвести их одновременно с помощь одной функции `plot` (см. гл. 3):

```
func := plot::Function2d(f(x)):
point := plot::PointList2d(xy, PointSize=3*unit::mm):
plot(func, point)
```

Полностью текст и результаты выполнения программы, которая ищет и изображает на графике экстремумы функции, приведены на рис. 6.4.

Очевидно, чтобы определить, какие из найденных экстремумов являются максимумами, а какие минимумами, необходимо определить знак второй производной в каждом из экстремумов.

Поиск экстремумов функций нескольких переменных выполняют аналогичным образом. При этом приравнивают к нулю все первые производные (по каждому аргументу), в результате чего получают систему уравнений, решением которой и являются координаты точек экстремума.

6.4.2. Поиск точек перегиба функции

Из математического анализа известно, что в точках перегиба выпуклость функции сменяется вогнутостью (или наоборот), а все вторые производные функции обращаются в этих точках в ноль. Таким образом, программа поиска точек перегиба математической функции, зависящей от одной переменной, должна содержать следующую последовательность действий:

1) объявить функцию;

2) вычислить вторую производную от исходной функции;

3) составить уравнение: приравнять к нулю вторую производную исходной функции;

4) решить полученное уравнение (найти все вещественные корни – абсциссы точек перегиба);

5) при необходимости рассчитать значения функции в точках перегиба и нанести их на график функции.

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
f := x -> x^3 - 2*x^2 - 11*x + 12;
f2 := f'';
ur := f2(x) = 0;
x1x2 := solve(ur, x);
xy := [[x, f(x)] $ x in x1x2];
func := plot::Function2d(f(x)):
```

```
point := plot::PointList2d(xy, PointSize=2*unit::mm):
plot(func, point)
```

Если вторая производная исходной функции не является полиномом, то при решении уравнения лучше использовать функцию `numeric::solve` с опцией `AllRealRoots`. Например:

```
x1x2 := numeric::solve(ur, x, AllRealRoots)
```

Поиск точек перегиба функции нескольких переменных выполняют аналогичным образом. При этом приравнивают к нулю все вторые производные (по каждому аргументу), в результате чего получают систему уравнений, решением которой и являются координаты точек перегиба.

6.4.3. Поиск особых точек функции

Особыми точками функции называют точки, в которых функция не определена или имеет нерегулярное поведение. Например, точка, в которой функция имеет разрыв или недифференцируема.

Среди особых точек выделяют устранимые, неустранимые и полюсы. Особая точка функции является ***устранимой***, если существует конечный предел этой функции в данной точке. Особая точка называется ***полюсом***, если в данной точке существует бесконечный предел. Особая точка считается ***неустранимой***, если функция не имеет предела в данной точке (ни конечного, ни бесконечного).

В точках разрыва предел справа и предел слева для исходной функции могут иметь разные значения.

Для поиска точек, где функция не определена или имеет разрыв, в MuPAD существует функция `discont`, которую вызывают следующим образом:

```
<Результат> := discont(<функция>,
                <аргумент> = <начало>..<конец>, <опции>)
```

В качестве параметра `<функция>` необходимо указать символьное математическое выражение, разрывы которого необходимо найти. Это выражение можно записать непосредственно на месте параметра `<функция>`, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение, либо с помощью вызова объявленной ранее функции с символьными аргументами.

В качестве `<аргумента>` указывают идентификатор одной из тех независимых символьных переменных, через которые записана `<функция>`. Если необходимо искать точки разрыва только в пределах фиксированного

интервала, то после параметра <аргумент> указывают его <начало> и <конец>. В противном случае эти параметры (вместе с символами = и . .) можно опустить.

Например, чтобы найти все точки, в которых функция $\frac{\ln(x+7)}{(x-3)(x+5)}$ терпит разрыв или не определена, можно написать команду `discont(log(x+7)/(x–3)/(x+5), x)`, в результате в поле вывода получится: {–5, 3} ∪ (–∞, –7]; а чтобы найти точки разрыва этой же функции на интервале [–4, 4], можно написать команду `discont(log(x+7)/(x–3)/ (x+5), x=-4..4)`, в результате в поле вывода получится множество: {3}.

Функцию `discont` можно вызвать с одной из следующих <опций>: `Undefined` – позволяет искать только те точки (интервалы), в которых функция не определена; `Real` – выполняет поиск только в вещественной области значений <аргумента>.

Если исходная функция определена везде, то функция `discont` находит только точки разрыва.

Для поиска только особых точек математической функции в MuPAD существует функция `poles`, которую вызывают аналогично функции `discont`:

<Результат> := **poles**(<функция>,
<аргумент> = <начало>..<конец>, <опции>)

Параметры <функция>, <аргумент>, <начало>, <конец> имеют тот же смысл, что и при вызове функции `discount`.

Функция `poles` НЕ ИЩЕТ интервалы и точки, в которых исходная <функция> не определена.

Например, чтобы найти все особые точки функции $\frac{\ln(x+7)}{(x-3)(x+5)}$, можно написать команду `poles(log(x+7)/(x-3)/(x+5), x)`, в результате в поле вывода получится множество: {–7, –5, 3}; а чтобы найти особые точки этой же функции на интервале [–6, 6], можно написать команду `poles(log(x+7)/(x-3)/(x+5), x=-6..6)`, в результате в поле вывода получится множество: {–5, 3}.

Функцию `poles` можно вызвать с одной из следующих <опций>, которые для каждой особой точки позволяют вывести дополнительную

информацию: `Multiple` – порядок полюса; `Residues` – значение вычета.

Кроме <опций> на результаты, получаемые функциями `discont` и `poles`, оказывают влияние свойства использованных для записи <функции> символьных переменных. По умолчанию все символьные переменные в MuPAD считаются комплексными, но с помощью функции `assume` (или `assuming`) можно указать другое числовое множество (см. разд. 4.7), к которому будет относиться аргумент исследуемой <функции>. Например, чтобы найти все особые точки функции $\frac{10}{(x^2-9)(x^2+25)}$, обозначенной ранее в программе переменной `y` (т.е. `y:=10/(x^2-9)/(x^2+25)`) можно написать команду `poles(y, x)`, в результате в поле вывода получится множество: $\{-3, 3, -5i, 5i\}$; а чтобы найти только вещественные особые точки этой же функции, можно написать команду `poles(y, x) assuming(x in R_)`, в результате в поле вывода получится множество: $\{-3, 3\}$.

Таким образом, программа поиска особых точек математической функции, зависящей от одной переменной, должна содержать следующую последовательность действий:

1) объявить функцию;

2) найти особые точки функции;

3) вычислив предел исходной функции в каждой из найденных точек, определить характер каждой особой точки (устранимая, неустранимая, полюс);

4) вычислив предел справа и предел слева для исходной функции в каждой из найденных точек, определить, какие из них являются точками разрыва;

5) при необходимости отметить найденные особые точки на оси абсцисс графика функции.

Эта последовательность действий может быть реализована, например, с помощью следующих команд:

```
f := x -> sin(x)/(x^3 + x^2 - x - 1);
x1x2 := poles(f(x), x);
x1 := op(x1x2, 1); x2 := op(x1x2, 2);
limit(f(x), x=x1);
limit(f(x), x=x1, Left); limit(f(x), x=x1, Right);
limit(f(x), x=x2);
```

```
limit(f(x), x=x2, Left); limit(f(x), x=x2, Right);
xy := [[x, 0] $ x in x1x2];
func := plot::Function2d(f(x)):
point := plot::PointList2d(xy, PointSize=2*unit::mm):
plot(func, point)
```

Если исходная функция не имеет других особых точек, кроме разрывов, то вместо оператора `x1x2 := poles(f(x), x)` можно использовать оператор `x1x2 := discont(f(x), x)`. Для корректного нахождения точек разрыва или особых точек других функций (отличающихся от приведенной в примере) может потребоваться использовать функции `discont` или `poles` с опциями либо ограничить свойства аргумента исследуемой функции `f(x)` с помощью оператора `assume` (или `assuming`). Например, вместо оператора `x1x2 := poles(f(x), x)` можно написать `x1x2 := poles(f(x), x) assuming(x in R_)`.

6.4.4. Построение касательных и асимптот

Касательные и асимптоты – это прямые. Поэтому в плоскости *XOY* уравнение касательной удобно записать в виде $y = k\cdot(x - x_0) + l$ – линейная функция.

Если необходимо построить касательную к функции $f(x)$, зависящей от одной переменной, в точке x_0, то при вычислении коэффициентов k и l необходимо помнить, что искомая касательная должна проходить через точку с координатами $[x_0, f(x_0)]$, лежащую на кривой $y = f(x)$, а тангенс угла наклона такой касательной должен быть равен производной исходной функции в данной точке $f'(x_0)$. Следовательно, чтобы найти коэффициенты k и l, необходимо совместно решить два уравнения:

$$f(x_0) = k\cdot(x - x_0) + l\Big|_{x=x_0} \text{ и } \frac{df(x)}{dx}\Big|_{x=x_0} = \frac{d}{dx}(k\cdot(x - x_0) + l)\Big|_{x=x_0}.$$

То есть

$$k = f'(x_0),\ l = f(x_0).$$

Таким образом, для построения проходящей через заданную точку касательной к математической функции, зависящей от одной переменной, программа должна содержать следующую последовательность действий:

1) объявить исходную функцию;
2) объявить линейную функцию;
3) вычислить производную от исходной функции;

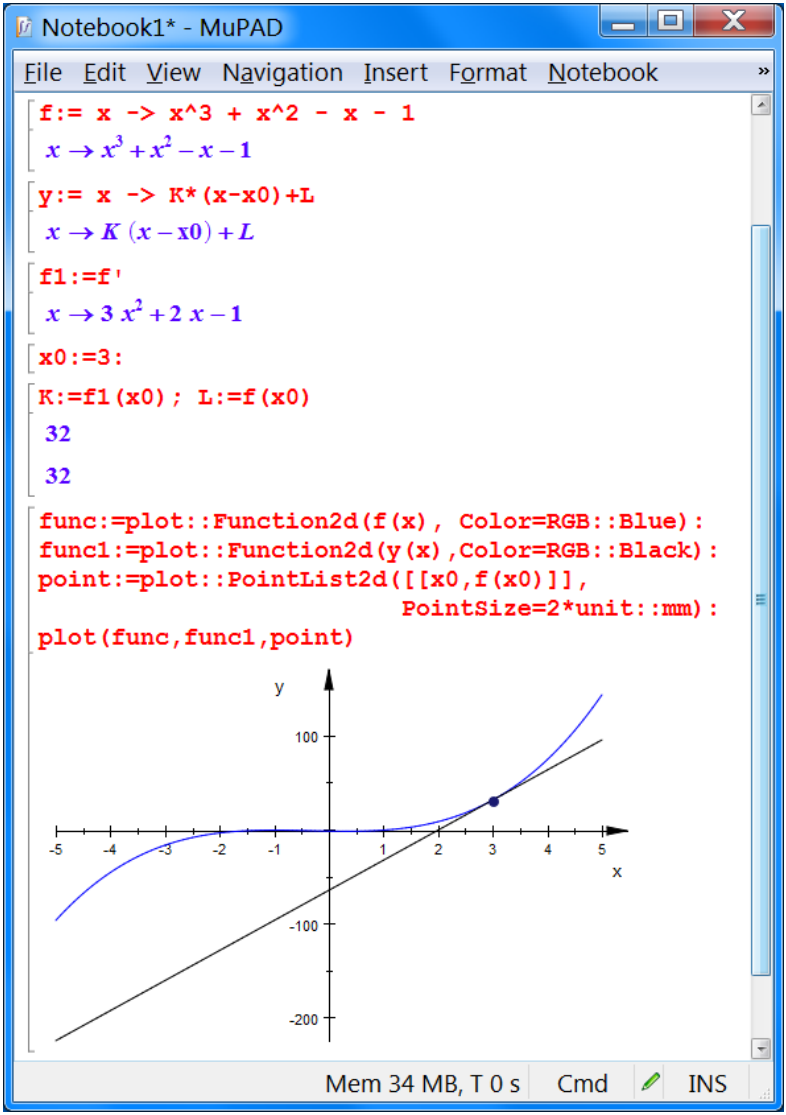


Рис. 6.5. Построение касательной к функции в заданной точке

4) задать абсциссу точки, через которую необходимо провести касательную;

5) вычислить коэффициенты линейной функции (касательной);

6) построить на одном графике исходную и линейную функции, а также отметьте точку с координатами (x_0, $f(x_0)$).

Эта последовательность действий может быть реализована, например, с помощью следующих команд, результаты выполнения которых приведены на рис. 6.5:

```
f : =x -> x^3 + x^2 - x - 1;
y := (x) -> K*(x-x0)+L;
f1 := f';
x0 := 3;
K := f1(x0); L := f(x0);
funcf := plot::Function2d(f(x), Color=RGB::Blue):
funcf1:= plot::Function2d(y(x), Color=RGB::Black):
point := plot::PointList2d([[x0, f(x0)]],
                      PointSize=2*unit::mm):
plot(funcf, funcf1, point)
```

Нормаль к функции $f(x)$ строят аналогичным образом: используют тот же вид линейной функции (как для касательной), но коэффициенты линейной функции вычисляются следующим образом:

$$k = -f'(x_0),\ l = f(x_0).$$

Уравнение асимптоты в плоскости XOY – это тоже линейная функция, но ее удобнее записать в виде: $y = k{\cdot}x + l$. Тогда коэффициенты линейной функции, являющейся асимптотой для функции $f(x)$, вычисляются следующим образом:

$$k = \lim_{x\to\infty} \frac{f(x)}{x},\ l = \lim_{x\to\infty}(f(x) - kx).$$

Если коэффициент k, вычисленный по этой формуле, является конечным, то функция $f(x)$ имеет наклонную асимптоту на $+\infty$. Если коэффициент k равен бесконечности или не существует, то функция $f(x)$ не имеет наклонной асимптоты.

На $-\infty$ функция $f(x)$ может иметь другую асимптоту, коэффициенты k и l линейной функции для которой необходимо искать, вычисляя аналогичные пределы при $x \to -\infty$.

6.5. Разложение функции в ряд

Представление функции $f(x)$ в виде бесконечной суммы однотипных слагаемых $f(x)=\sum_{k=0}^{\infty} c_k \varphi_k(x)$ (где c_k – числовые коэффициенты) называют разложением этой функции в ряд.

Ряд Тейлора – это представление функции в виде бесконечной суммы многочленов (полиномов целых положительных степеней). Для бесконечно дифференцируемой функции $f(x)$ в окрестности точки $x = a$ этот ряд имеет вид:

$$f(x)=\sum_{k=0}^{\infty} c_k (x-a)^k = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k =$$

$$= f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\ldots+\frac{f^{(n)}(a)}{n!}(x-a)^n+\ldots$$

Здесь числовые коэффициенты $c_k = \dfrac{f^{(k)}(a)}{k!}$.

Разложение функции в ряд Тейлора в окрестностях нуля ($a = 0$) называется рядом Маклорена.

Функции комплексного переменного можно также разложить в ряд Лорана, который кроме полиномов целых положительных степеней содержит полиномы целых отрицательных степеней (с другими числовыми коэффициентами c_k):

$$f(x)=\sum_{k=0}^{\infty} c_k (x-a)^k + \sum_{k=1}^{\infty} c_{-k} (x-a)^{-k}.$$

Если разложение функции выполняется по полиномам дробных (рациональных) степеней (с соответствующими числовыми коэффициентами c_k), то его можно записать в виде ряда Пюизе:

$$f(x)=\sum_{k=n_0}^{\infty} c_k (x-a)^{k/m}$$, где n_0 – целое, m – натуральное число.

Ряды Тейлора, Маклорена и Лорана являются частными случаями ряда Пюизе (при $m = 1$)

Если функция $f(x)$ имеет особые точки, то вблизи ее особых точек соответствующие полиномы не существуют, поэтому функцию $f(x)$ раскладывают в ряд не по полиномам, а по другим функциям $\psi_n(x)$, которые в

этих точках существуют и $\psi_{n+1}(x) = o(\psi_n(x))$ при стремлении x к некоторой предельной точке x_0. Такой ряд называется асимптотическим:

$$f(x) = \sum_{k=0}^{\infty} c_k \psi_k(x) \text{ при } x \to x_0.$$

В общем виде в окрестности точки $x = a$ любой из этих рядов можно записать следующим образом:

$$f(x) = \sum_{k=n_0}^{\infty} c_k \varphi_k(x),$$

где либо $\varphi_k(x) = (x-a)^{\frac{n_0+k}{m}}$ – полиномы, либо $\varphi_k(x) = \psi_k(x)$ – неполиномиальные функции (в зависимости от типа ряда) с соответствующими числовыми коэффициентами c_k, значения которых зависят и от типа ряда и от вида функции $f(x)$.

Все перечисленные выше ряды являются бесконечными (содержат бесконечное количество слагаемых). Но в каждом из них принято явным образом записывать только несколько первых слагаемых (членов ряда), а сумму остальных слагаемых называют остаточным членом и заменяют специальным обозначением. Как правило, остаточный член записывают с помощью O – символа Ландау, который характеризует асимптотическое поведение этого члена. Для ряда, записанного в общем виде, это делают следующим образом:

$$f(x) = \sum_{k=n_0}^{\infty} c_k \varphi_k(x) =$$

$$= c_{n_0}\phi_{n_0}(x) + c_{n_0+1}\phi_{n_0+1}(x) + \ldots + c_{n-1}\varphi_{n-1}(x) + O(\varphi_n(x)).$$

Значение символа Ландау для произвольной функции $f(x)$ соответствует вычислению предела $\lim\limits_{x \to x_0} f(x)$. Для вычисления символа Ландау в MuPAD можно воспользоваться функцией `O`, которую вызывают следующим образом:

`<Переменная> := O(<функция>, <аргумент> = <значение>)`

Здесь под `<функцией>` понимают математическое выражение, а под `<аргументом>` одну из независимых символьных переменных, через которые оно записано. `<Значение>` указывает, куда будет стремиться аргумент функции при вычислении предела. Если `<значение>` не указано, то по умолчанию аргумент стремится к нулю. Если параметр `<ар-`

`гумент> = <значение>` полностью опустить, то при вычислении предела все независимые символьные переменные, через которые записана `<функция>`, будут стремиться к нулю. Например, после выполнения команды `O(x, x = 1)` получится $O(1,\ x = 1)$; после выполнения команды `O(x^4 + 2*x^2)` получится $O(x^2)$; после выполнения команды `O(x^3*y^2 + x^2*y^2)` получится $O(x^2y^2)$.

Если остаточный член ряда записан в виде $O(F(x))$, это означает, что в окрестности указанной точки $x = a$ он растет не быстрее, чем $F(x)$.

Чтобы конкретизировать количество членов ряда $f(x) = c_{n_0}\varphi_{n_0}(x) + c_{n_0+1}\varphi_{n_0+1}(x) + \ldots + c_{n-1}\varphi_{n-1}(x) + O(\varphi_n(x))$, которые необходимо записать явным образом, принято указывать, до какой функции $\varphi_n(x)$ включительно должны содержать эти члены. Как показано выше, в большинстве рядов разложение выполняется по полиномам. Поэтому достаточно указать n – максимальную степень (порядок) полиномов, входящих в слагаемые, не включенные в остаточный член ряда. Кроме того, независимо от того, по каким функциям $\varphi_k(x)$ выполняется разложение, отношение каждого члена ряда (в том числе и остаточного) к первому тоже является полиномом, степень (порядок) которого принято использовать, чтобы зафиксировать количество явным образом записанных членов ряда.

Если в дальнейшем остаточный член отбрасывается (не учитывается), то исходная функция приближенно заменяется конечной суммой членов ряда, записанных явным образом: $f(x) \approx c_{n_0}\varphi_{n_0}(x) + c_{n_0+1}\varphi_{n_0+1}(x) + \ldots + c_{n-1}\varphi_{n-1}(x)$. Чем больше слагаемых включено в такую сумму, тем точнее такая приближенная замена (представление) вблизи некоторой точки. Поэтому указание максимального порядка (степени) членов ряда позволяет фиксировать не только количество явным образом записанных слагаемых (членов) этого ряда, но и точность, с которой исходную функцию можно заменить соответствующей суммой, состоящей из конечного количества членов ряда.

Для управления порядком (степенью) разложения в ряд в MuPAD используется переменная среды `ORDER`. Она задает порядок (степень), ограничивающий количество явным образом записанных членов ряда, которые должна получить функция, выполняющая разложение (`taylor`, `series`, `asympt` и т.д.). По умолчанию `ORDER = 6` – в разложение включаются члены, отношение которых к первому члену ряда является полиномом меньше шестой степени (разложение выполняется с относительной

точностью до шестого порядка). Значение переменной `ORDER` можно изменять в пределах от 1 до 2^{31} (только целые). Команда `delete ORDER` возвращает этой переменной значение, установленное по умолчанию.

Рассмотрим функции MuPAD, предназначенные для разложения в ряд заданной математической функции.

Для разложения аналитически заданной математической функции в ряд Тейлора в MuPAD можно использовать функцию `taylor` следующим образом:

```
<Переменная> := taylor(<функция>,
                <аргумент> = <значение>, <опции>)
```

В качестве параметра `<функция>` необходимо указать символьное математическое выражение, которое необходимо разложить в ряд Тейлора. На месте параметра `<функция>` это выражение можно записать либо непосредственно, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение, либо с помощью вызова объявленной ранее функции с символьными аргументами.

В качестве `<аргумента>` следует использовать идентификатор одной из тех независимых символьных переменных, через которые записана `<функция>`. Параметр `<значение>`

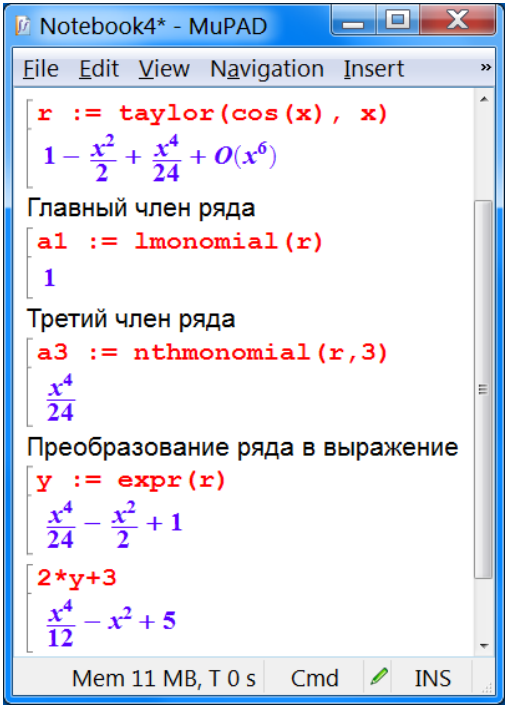


Рис. 6.6. Разложение функции cos *x* в ряд Тейлора (Маклорена)

указывает, в какой точке необходимо выполнить разложение. Если параметр <значение> не указан, функция taylor разлагает <функцию> в ряд в точке, в которой <аргумент> равен нулю (по умолчанию) – ряд Маклорена. Функция taylor возвращает результат разложения в ряд соответствующего символьного выражения в виде суммы нескольких членов ряда, записанных явным образом, и остаточного члена, записанного с помощью *O*-символа Ландау:

$$c_0 + c_1(x - a) + \ldots + c_{n-1}(x - a)^{n-1} + O((x - a)^n).$$

На рис. 6.6 приведен пример использования функции taylor (без опций) для разложения функции $\cos x$ в ряд.

Функцию taylor можно использовать с одной из следующих <опций>: AbsoluteOrder = <порядок> – позволяет получить разложение с абсолютной точностью заданного <порядка> (этот порядок будет иметь полином, указанный в *O*-символе); <порядок> или RelativeOrder = <порядок> – позволяет получить разложение с относительной точностью заданного порядка (используется по умолчанию): отношение функции, указанной в *O*-символе, к первому члену ряда будет являться полиномом указанного <порядка>, следовательно, количество членов ряда (в том числе и нулевых) перед *O*-символом будет равно заданному <порядку>. При указании каждой из <опций> в качестве параметра <порядок> следует использовать целое неотрицательное число. Если <опции> не заданы, то относительная точность разложения определяется переменной среды ORDER, которая по умолчанию равна 6.

Например, при разложении функции $\frac{\sin x}{x+1}$ в ряд Тейлора до третьего порядка с помощью команды taylor(sin(x)/(x+1), x, AbsoluteOrder = 3) получится ряд $x - x^2 + O(x^3)$, а с помощью любой из следующих команд taylor(sin(x)/(x+1), x, RelativeOrder = 3) или taylor(sin(x)/(x+1), x, 3) получится ряд $x - x^2 + \frac{5}{6}x^3 + O(x^4)$ ($\frac{O(x^4)}{x}$ – величина третьего порядка).

Как принято в математическом анализе, если в полученном разложении член указанного порядка обращается в ноль, то при записи остаточного члена используется *O*-символ для следующего члена ряда. Например, при разложении функции $\cos x$ вблизи нуля в ряд Тейлора до

третьего порядка с помощью команды `taylor(cos(x), x, 3)` получится ряд: $1-\frac{1}{2}x^2+O(x^4)$.

Для разложения в ряд Тейлора математической функции, зависящей от нескольких аргументов, в MuPAD используют функцию `mtaylor`, вызов которой отличается от вызова функции `taylor` только количеством <аргументов>, по которым необходимо проводить разложение. Список <аргументов> и (при необходимости) их <значений> записывают в квадратных скобках через запятую. Подробнее с синтаксисом функции `mtaylor` можно ознакомиться в справочной системе MuPAD.

Разложение в асимптотический ряд в MuPAD можно получить с помощью функции `asympt`, которую вызывают следующим образом:

<Переменная> := **asympt**(<функция>,
<аргумент> = <значение>, <опции>)

Здесь параметры <функция>, <аргумент>, <значение> имеют тот же смысл, как при вызове функции `taylor`. Однако если <значение> не задано, выполняется разложение на бесконечности (<аргумент> = `infinity` по умолчанию).

Функцию `asympt` можно использовать со следующими <опциями>:

<порядок> – целое неотрицательное число, которое указывает относительную точность разложения (см. опции функции `taylor`). Если эта <опция> не задана, то <порядок> определяется переменной среды `ORDER`;

`Left`, `Right` – позволяет получить разложение при стремлении <аргумента> к заданному <значению> слева (`Left`) или справа (`Right`) соответственно. Для конечного <значения> по умолчанию используется `Right`.

Функция `asympt` возвращает результат разложения <функции> в ряд в виде:

$$c_0\psi_0(x) + c_1\psi_1(x) + \ldots + c_{n-1}\psi_{n-1}(x) + O(\psi_n(x)).$$

Функция `asympt` сама подбирает функции $\psi_k(x)$, по которым выполняет разложение. Например, после выполнения команды:

```
asympt(sin(1/x+exp(-x))-sin(1/x), x, 3)
```

получится асимптотическое разложение функции $\sin\left(1/x+e^{-x}\right)-\sin\left(1/x\right)$ на бесконечности в ряд до третьего порядка:

$$e^{-x} - \frac{e^{-x}}{2x^2} + \frac{e^{-x}}{24x^4} + O\left(\frac{e^{-x}}{x^5}\right);$$

а после выполнения команды:

```
asympt(1/(exp(-x)+x-1), x=0, 3, Left)
```

получится разложение функции $\frac{1}{e^{-x}+x-1}$ вблизи нуля в ряд до третьего порядка в виде $\frac{2}{x^2} + \frac{2}{3x} + \frac{1}{18} + O(-x)$.

Функция `series` выполняет разложение математической функции в ряд одного из перечисленных выше типов. Перед разложением функция `series` анализирует исходную математическую <функцию> и точку, вблизи которой необходимо выполнить разложение, выбирает какой ряд (Тейлора, Лорана, Пюизе или асимптотический) для этого использовать. Вызов функции `series` осуществляется аналогично вызову функций `taylor` и `asympt`:

<Переменная> := **series**(<функция>,
<аргумент> = <значение>, <опции>)

Здесь параметры <функция>, <аргумент>, <значение> имеют тот же смысл, как при вызове функции `taylor`. Если параметр <значение> не задан, то выполняется разложение в нуле (<аргумент> = 0). Так же как и для функций `taylor` и `asympt`, здесь в качестве одной из опций можно указать относительный <порядок> разложения (в виде положительного целого числа). Остальные опции функции `series` можно изучить самостоятельно с помощью справочной системы MuPAD.

Функция `series` возвращает результат разложения <функции> в ряд в виде:

$$c_{n_0}\varphi_{n_0}(x) + c_{n_0+1}\varphi_{n_0+1}(x) + \ldots + c_{n-1}\varphi_{n-1}(x) + O(\varphi_n(x)).$$

Результатов, полученных с помощью функций `taylor` и `asympt` в каждом из приведенных выше примеров, можно достичь, использовав функцию `series`. В частности, в результате выполнения команды `series(sin(x)/(x+1), x, 3)`, также как при выполнении команды `taylor(sin(x)/(x+1), x, 3)`, получится ряд $x - x^2 + \frac{5}{6}x^3 + O(x^4)$.

С помощью раскрывающегося списка `General Math` → `Power Series`, расположенного на командной панели `Command Bar`, в программу можно вставить шаблон для разложения функции в ряд: `series(#, #)`,

который содержит поля, отмеченных символом #. Каждое из полей шаблона необходимо заполнить: в первом поле необходимо указать <функцию>, а во втором – идентификатор <аргумента> и при необходимости его <значение> в соответствии с описанными выше правилами.

Любая из перечисленных выше функций (taylor, asympt или series) возвращает результат в виде объекта типа Series, который не является математическим выражением. Каждый ряд (объект типа Series) состоит из нескольких членов, первый из которых считается главным, а последний – остаточным: $c_{n_0}\varphi_{n_0}(x) + c_{n_0+1}\varphi_{n_0+1}(x) + \ldots + c_{n-1}\varphi_{n-1}(x) + O(\varphi_n(x))$. Наличие остаточного члена, который записывается через символ Ландау, говорит о том, что данный объект имеет тип Series.

Каждый член ряда $c_k\varphi_k(x)$, кроме остаточного, представляет собой произведение функции $\varphi_k(x)$, по которой проводилось разложение (например, полинома: $\varphi_k(x) = (x - a)^k$), на постоянный коэффициент c_k.

Таблица 6.1

Функции для работы с рядами типа Series

Функция	Назначение
lmonomial(<ряд>)	Возвращает главный член ряда
lterm(<ряд>)	Возвращает функцию, по которой проводилось разложение, для главного члена ряда
lcoeff(<ряд>)	Возвращает коэффициент главного члена ряда
ldegree(<ряд>)	Возвращает старшую степень для полинома
nthmonomial(<ряд>, n)	Возвращает *n*-й член ряда
nthterm(<ряд>, n)	Возвращает функцию, по которой проводилось разложение, для *n*-го члена ряда
nthcoeff(<ряд>, n)	Возвращает коэффициент *n*-го члена ряда
coeff(<ряд>)	Возвращает список коэффициентов всех членов ряда, кроме остаточного
expr(<ряд>)	Преобразует в математическое выражение ряд без остаточного члена

Несмотря на то, что в поле вывода результат разложения в ряд выглядит как сумма, в действительности данный объект представляет собой совокупность степеней функций $\varphi_k(x)$, по которым выполнялось разложение, коэффициентов разложения c_k и точки a, в которой выполнялось

разложение. Поэтому извлечь (создать копии) членов ряда (слагаемых) из объекта типа `Series` с помощью функции `op` невозможно. Для извлечения отдельных членов ряда, а также содержащихся в них функций и коэффициентов в MuPAD существует несколько специальных функций (табл. 6.1), для вызова каждой из которых необходимо указать параметр <ряд> – объект типа `Series` или соответствующий ему идентификатор. Результатом выполнения каждой из перечисленных в табл. 6.1 функций (кроме `coeff`) являются математические выражения.

Пусть, например, идентификатором `r` обозначен результат разложения функции cos x в точке $x = \pi$ в ряд до третьего порядка, полученный с помощью команды `r:=series(cos(x), x=PI, 3)`, после выполнения которой в поле вывода получился объект $-1+\frac{(\pi-x)^2}{2}+O((\pi-x)^4)$. Тогда с помощью команды `lmonomial(r)` можно получить главный член ряда: –1; а с помощью команды `nthmonomial(r, 2)` – второй член ряда: $\frac{(\pi-x)^2}{2}$. Другой пример извлечения членов ряда с помощью функций, перечисленных в табл. 6.1, приведен на рис. 6.6.

Некоторые операции, такие как вычисление производной, построение графика и т.д., с объектами типа `Series` можно выполнять как с обычными математическими выражениями. Например, в результате выполнения команд `r := series(cos(x), x=PI, 3); diff(r, x)` можно получить не только разложение функции cos x в ряд в точке $x = \pi$ до третьего порядка: $-1 + (\pi - x)^2/2 + O((\pi - x)^4)$, но и производную от этого ряда: $x - \pi + O(-(\pi - x)^3)$. Однако для выполнения некоторых операций (например, подстановки `subs`) объекты типа `Series` предварительно необходимо преобразовывать в выражения. Для этого используют функцию `expr` следующим образом:

<Переменная> **:=** **expr**(<ряд>)

В качестве параметра <ряд>, как правило, используют идентификатор переменной, которой присвоен результат разложения в ряд некоторой функции – объект типа `Series`. На рис. 6.6 представлен пример преобразования в выражение разложения в ряд Тейлора вблизи нуля для функции cos x.

6.6. Интегрирование

6.6.1. Аналитическое вычисление интегралов

Для аналитического вычисления определенных и неопределенных интегралов в MuPAD используется функция `int`, которую вызывают следующим образом:

```
<Переменная> := int(<функция>,
                <аргумент> = <начало> .. <конец>, <опции>)
```

В качестве параметра `<функция>` следует указать символьное выражение, которое необходимо проинтегрировать. На месте параметра `<функция>` это выражение можно записать либо непосредственно, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение, либо с помощью вызова объявленной ранее функции с символьными аргументами.

В качестве `<аргумента>` указывают идентификатор одной из тех независимых символьных переменных, через которые записана `<функция>`. Параметры `<начало>` и `<конец>` указывают границы интервала, на котором необходимо выполнить интегрирование. Если границы интервала не указаны, функция `int` вычисляет ***неопределенный интеграл***; в противном случае – ***определенный*** (на рис. 6.7 приведены примеры вычисления определенного и неопределенного интегралов с помощью функции `int`). В качестве границ интервала можно использовать как числовые значения, так и математические выражения, записанные на месте параметров `<начало>` и `<конец>` непосредственно или через идентификаторы переменных, которым эти выражения (значения) были ранее присвоены. В процессе вычисления определенного интеграла функция `int` проверяет, имеет ли подынтегральная `<функция>` особые точки внутри и на границе интервала, а также существует ли предел первообразной в этих точках и на $\pm\infty$, если одна или обе границы интервала интегрирования являются бесконечными. Если соответствующие пределы существуют и конечны, то интеграл считается сходящимся несобственным и вычисляется следующим образом: $\int_a^b f(x)dx = \lim_{\varepsilon\to 0}\int_a^{b-\varepsilon} f(x)dx$ или $\int_a^b f(x)dx = \lim_{\varepsilon\to 0}\int_{a+\varepsilon}^{b} f(x)dx$, если особая

точка находится на границе интервала; $\int_a^b f(x)dx = \lim_{\substack{\varepsilon_1 \to 0 \\ \varepsilon_2 \to 0}} \left(\int_a^{c-\varepsilon_1} f(x)dx + \right.$

$\left. + \int_{c+\varepsilon_2}^b f(x)dx \right)$, если особая точка c находится внутри интервала;

$\int_a^{+\infty} f(x)dx = \lim_{b \to +\infty} \int_a^b f(x)dx$, $\int_{-\infty}^b f(x)dx = \lim_{a \to -\infty} \int_a^b f(x)dx$ или

$\int_{-\infty}^{\infty} f(x)dx = \lim_{\substack{a \to -\infty \\ b \to +\infty}} \int_a^b f(x)dx$ на бесконечном интервале. Если на интервале интегрирования, не являющемся бесконечным, подынтегральная функция не имеет особых точек, то интеграл вычисляется по формуле Ньютона–Лейбница.

Если функция `int` может вычислить интеграл (неопределенный, определенный собственный или несобственный сходящийся), то она возвращает результат в виде математического выражения или числового значения. Например, в результате выполнения команды `int(4*x^3-1, x)` в поле вывода появится выражение $x^4 - x$, а в результате выполнения команды `int(3*x^2, x = -1..2)` появится значение 9. Если несобственный интеграл расходится, то функция `int`, вызванная без `<опций>`, возвращает undefined.

Функцию `int` можно использовать со следующими `<опциями>`:

`IgnoreAnalyticConstraints` – разрешает применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами, например считать тождеством arcsin(sin x) = x и т.д.;

`IgnoreSpecialCases` – позволяет получить решения, игнорирующие особые случаи, например обращение в ноль знаменателя или другие особые точки;

`PrincipalValue` – позволяет вычислить расходящийся несобственный интеграл в смысле главного значения по критерию Коши. Если для несобственного интеграла $\int_a^b f(x)dx$, содержащего на интервале интегрирования особую точку c функции $f(x)$, существует предел

$$\int_a^b f(x)dx = \lim_{\varepsilon\to 0}\left(\int_a^{c-\varepsilon} f(x)dx + \int_{c+\varepsilon}^b f(x)dx\right)$$

или для несобственного интеграла $\int_{-\infty}^{+\infty} f(x)dx$ существует предел

$$\int_{-\infty}^{+\infty} f(x)dx = \lim_{b\to\infty}\int_{-b}^{b} f(x)dx,$$

то этот предел считается главным значением соответствующего интеграла (по критерию Коши). Если несобственный интеграл сходится, то его значение, полученное в обычном смысле, совпадает с главным (в этом случае функция `int` с опцией `PrincipalValue` и без нее возвращает одинаковый результат). Если несобственный интеграл расходится, то его значение в обычном смысле не существует, но может существовать в смысле главного значения. Тогда функция `int` без опции `PrincipalValue` возвращает undefined, а с этой опцией – соответствующее главному значению интеграла выражение или числовое значение. Например, для вычисления интеграла

$$\int_{-1}^{2}\frac{dx}{x} = \lim_{\varepsilon\to 0}\left(\int_{-1}^{0-\varepsilon}\frac{dx}{x} + \int_{0+\varepsilon}^{2}\frac{dx}{x}\right) =$$

(после замены переменных $x = -t$ в первом интеграле)

$$= \lim_{\varepsilon\to 0}\left(-\int_{\varepsilon}^{1}\frac{dt}{t} + \int_{\varepsilon}^{1}\frac{dx}{x} + \int_{1}^{2}\frac{dx}{x}\right) = \int_{1}^{2}\frac{dx}{x} = \ln 2$$

можно написать команду `int(1/x, x=-1..2, PrincipalValue)`, в результате выполнения которой тоже получится значение ln(2).

С помощью раскрывающейся панели кнопок `Integral` , расположенной на командной панели `Command Bar`, в программу можно вставить один из двух шаблонов для аналитического вычисления интегралов: `int(#f, #x)` (кнопка `Indefinite integral` – неопределенный интеграл); `int(#f, #x=#a..#b)` (кнопка `Definite integral` – определенный интеграл). Каждый вставленный с помощью этих кнопок шаблон содержит поля, отмеченные символом `#`, которые необходимо заполнить в соответствии с описанными выше правилами. В поле `#f` следует указать `<функцию>`, в поле `#x` – идентификатор

<аргумента>, в полях #a и #b – <начало> и <конец> интервала, на котором необходимо выполнить интегрирование. Кроме того, эти же шаблоны можно вставить в текст программы с помощью последовательности раскрывающихся списков General Math → Calculus → Integrate (или … → Integrate (Definite)), расположенной на той же командной панели Command Bar.

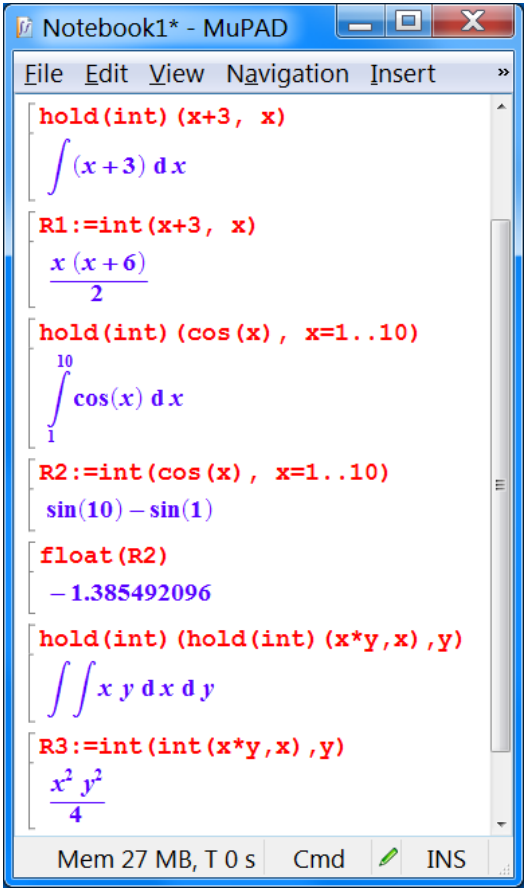


Рис. 6.7. Аналитическое вычисление интеграла

Если вычисление интеграла необходимо отложить, функцию `hold` применяют следующим образом:

```
<Переменная> := hold(int) (<функция>,
                <аргумент> = <начало> .. <конец>, <опции>)
```

Тогда в поле вывода отображается не результат вычислений, а соответствующая математическая запись задачи через символ ∫. Этот прием можно использовать, например, для контроля правильности введенной команды для интегралов, содержащих подынтегральные функции сложного вида (см. рис. 6.7).

Если функция `int` не может представить результат ни в виде числового значения, ни в виде символьного выражения, то в поле вывода так же отображается соответствующая математическая запись задачи через символ ∫, как и при отложенных вычислениях. Полученное с помощью функции `int` числовое значение интеграла с помощью функции `float` можно представить в десятичном формате (см. рис. 6.7). Применение функции `float` к результату аналитического интегрирования, полученному с помощью функции `int`, эквивалентно использованию функции `numeric::int` (см. разд. 6.6.2).

Кратные интегралы (как определенные, так и неопределенные) вычисляются аналитически путем повторного вызова функции `int`. Например, чтобы вычислить неопределенный интеграл $\iint xy\ dxdy$ достаточно написать команду: `int(int(x*y, x), y)`

Перед выполнением интегрирования MuPAD проверяет, соответствуют ли условия интегрирования (например, границы интегрирования или свойства подынтегральной функции) текущим свойствам переменных, по которым выполняется интегрирование. Установки по умолчанию таковы, что MuPAD НЕ ВЫВОДИТ сообщения об обнаруженных противоречиях. Чтобы включить появление этих предупреждающих сообщений, необходимо использовать команду `intlib::printWarnings(TRUE)`. Команда `intlib::printWarnings(FALSE)` отключает вывод предупреждающих сообщений, а команда `intlib::printWarnings()` позволяет узнать, включены или отключены соответствующие предупреждающие сообщения.

Для получения результата функция `int` использует все известные методы сведения исходного интеграла к табличным автоматически. Однако при необходимости можно принудительно выбрать метод интегрирования по частям или метод интегрирования путем замены переменных.

При вычислении интеграла по частям подынтегральную функцию необходимо представить в виде произведения двух функций: $u'(x) = \dfrac{du}{dx}$ и $v(x)$, тогда

$$\int u'(x)v(x)dx = u(x)v(x) - \int u(x)v'(x)dx .$$

Для вычисления интеграла по частям в MuPAD используют функцию `intlib::byparts`, которую вызывают следующим образом:

```
<Переменная> := intlib::byparts(<интеграл>,
                                <часть u'>)
```

В качестве параметра `<интеграл>` необходимо указать аналитическую запись невычисленного интеграла (определенного или неопределенного). В таком виде можно представить любой интеграл с помощью отложенных вычислений `hold(int)(…)`. Если параметр `<интеграл>` записать через `int(…)` без отложенных вычислений, то интеграл сначала будет вычислен, и в функцию `intlib::byparts` будет подставлено полученное математическое выражение, а не интеграл. Применить интегрирование по частям к выражению (не к интегралу) нельзя.

В качестве параметра `<часть u'>` необходимо указать тот сомножитель подынтегральной функции, который при интегрировании по частям необходимо обозначить как $u'(x)$.

В результате выполнения функции `intlib::byparts` будет один раз применен метод интегрирования по частям для указанного интеграла. Функция `intlib::byparts` возвращает выражение, состоящее из двух слагаемых, одно из которых – интеграл, вычисление которого отложено, даже если этот интеграл является табличным. Для вычисления этого отложенного интеграла следует применить функцию `eval` к полученному результату. (Пример вычисления интеграла по частям приведен на рис. 6.8.) Если вычисление интеграла по частям необходимо продолжить, то функцию `intlib::byparts` применяют повторно по перечисленным выше правилам.

Шаблон `intlib::byparts(hold(int)(#, #), #(#))` для вычисления интегралов по частям можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Calculus` → `Integrate by Parts`, расположенной на командной панели `Command Bar`. Все поля, отмеченные символом `#`, необходимо заполнить в соответствии с описанными выше правилами.

При вычислении интеграла методом подстановки подынтегральную функцию следует представить в виде произведения функции $f(g(x))$ на производную ее аргумента $g'(x) = \dfrac{dg}{dx}$, тогда

$$\int f(g(x))g'(x)dx = \left(\int f(t)dt\right)\Big|_{t=g(x)}.$$

Для вычисления интеграла по частям в MuPAD используют функцию `intlib::changevar`, которую вызывают следующим образом:

```
<Переменная> := intlib::changevar(<интеграл>,
                                  <замена>, <новый аргумент>)
```

В качестве параметра `<интеграл>` (так же как и для предыдущей функции) необходимо указать аналитическую запись невычисленного интеграла (определенного или неопределенного). В таком виде можно представить любой интеграл с помощью отложенных вычислений `hold(int)(...)`. Если параметр `<интеграл>` записать через `int(...)` без отложенных вычислений, то интеграл сначала будет вычислен, и в функцию `intlib::changevar` будет подставлено полученное математическое выражение, а не интеграл. Применить интегрирование методом подстановки к выражению (не к интегралу) нельзя.

В качестве параметра `<новый аргумент>` необходимо указать идентификатор переменной, которой необходимо обозначить переменную интегрирования в подынтегральном выражении после выполнения замены переменных. В качестве параметра `<замена>` необходимо указать уравнение, связывающее `<новый аргумент>` и переменную интегрирования в исходном интеграле.

В результате выполнения функции `intlib::changevar` будет один раз применен метод интегрирования с помощью указанной подстановки для указанного интеграла. Функция `intlib::changevar` возвращает интеграл, вычисление которого отложено, даже если этот интеграл является табличным. Для вычисления такого отложенного интеграла следует применить функцию `eval` к полученному результату. Если методом замены переменных вычисляется неопределенный интеграл, то чтобы вернуться в полученном интеграле к исходным обозначениям (переменным), следует с помощью функции `subs`, например, подставить в полученное выражение `<новый аргумент>`, выразив его из сделанной ранее `<замены>`. Пример вычисления интеграла методом замены переменных также приведен на рис. 6.8.

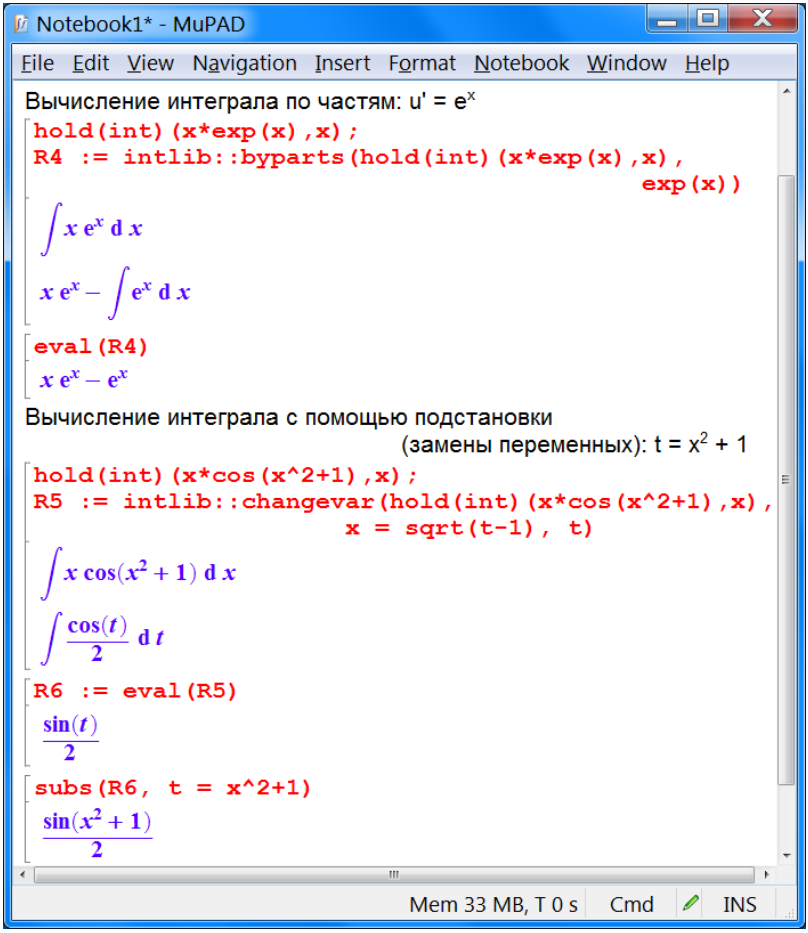


Рис. 6.8. Вычисление интеграла по частям и с помощью подстановки

Шаблон `intlib::changevar(hold(int)(#f, #x), #x = #y)` для вычисления интегралов методом замены переменных можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Calculus` → `Change Variaible`, расположенной на командной панели `Command Bar`. Все поля, отмеченные

символом #, необходимо заполнить в соответствии с описанными выше правилами.

6.6.2. Численное интегрирование

Численно выполняется только вычисление определенных интегралов. При этом и границы интервала, и все параметры подынтегральной функции, кроме переменной интегрирования, должны быть заранее заданы так, чтобы их значения можно было представить в виде чисел в десятичном формате.

Численное интегрирование осуществляется по квадратурным формулам (квадратурам) вида:

$$\int_a^b f(x)dx \approx \sum_{k=1}^{n} c_k f(x_k)$$

без вычисления первообразных. Точки x_k, в которых вычисляют значения подынтегральной функции для расчета интеграла, называют узлами квадратурной формулы, а константы c_k – весовыми коэффициентами.

Существует несколько видов квадратурных формул, которые отличаются друг от друга расположением узлов и значениями коэффициентов. Так как любой интеграл, вычисляемый на интервале конечной длины, с помощью линейного преобразования $x = a + (b - a)t$ можно свести к интегралу на интервале [0, 1] (или другой заменой – на интервале [–1, 1]), то каждый метод численного интегрирования характеризуется узлами t_k, расположенными на этом интервале, и соответствующими им весовыми коэффициентами c_k:

$$\int_a^b f(x)dx = (b-a)\int_0^1 f(a+(b-a)t)dt \approx (b-a)\sum_{k=1}^{n} c_k \cdot f(a+(b-a)t_k).$$

В MuPAD используется три метода численного интегрирования (квадратурные формулы): метод Ньютона–Котеса; метод Гаусса–Чебышева и метод Гаусса–Лежандра.

С помощью функции `numeric::ncdata` можно получить значения узлов и весовых коэффициентов на интервале [0, 1] для метода Ньютона–Котеса, который применяется для непрерывной подынтегральной функции любого вида. Для этого функцию `numeric::ncdata` вызывают следующим образом:

<Переменная> := **numeric::ncdata**(<количество узлов>)

`<Количество узлов>` – целое положительное число, указывающее, сколько значений функции предстоит вычислить, чтобы рассчитать по ним интеграл. Функция `numeric::ncdata` возвращает список `[]`, состоящий, в свою очередь, из двух списков: [список коэффициентов c_k], [список узлов t_k]. Например, в результате выполнения команды `numeric::ncdata(3)` можно получить список $\left[\left[\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\right], \left[0, \frac{1}{2}, 1\right]\right]$. В методе Ньютона–Котеса узлы располагаются на интервале [0, 1] равномерно, а коэффициенты попарно симметричны. Метод является точным для всех полиномов, степень которых меньше, чем используемое в методе количество узлов.

С помощью функции `numeric::gtdata` можно получить значения узлов и весовых коэффициентов на интервале [0, 1] для метода Гаусса–Чебышева, который применяется, если подынтегральная функция имеет вид: $f(x) = \frac{p(x)}{\sqrt{x(1-x)}}$. Для этого функцию `numeric::gtdata` вызывают следующим образом:

`<Переменная> :=` **`numeric::gtdata`**`(<количество узлов>)`

`<Количество узлов>` – целое положительное число, указывающее, сколько значений функции предстоит вычислить, чтобы рассчитать по ним интеграл. Функция `numeric::gtdata` тоже возвращает список `[]`, состоящий, в свою очередь, из двух списков: [список коэффициентов c_k], [список узлов t_k]. Например, в результате выполнения команды `numeric::gtdata(3)` можно получить список $\left[\left[\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{12}\right], \left[\frac{\sqrt{3}}{4} + \frac{1}{2}, \frac{1}{2}, \frac{1}{2} - \frac{\sqrt{3}}{4}\right]\right]$. В методе Гаусса–Чебышева узлы располагаются на интервале [0, 1] симметрично относительно середины интервала, а коэффициенты попарно симметричны. Метод является точным, если $p(x)$ – полином, степень которого не выше, чем $2n - 1$ (где n – используемое в методе количество узлов).

С помощью функции `numeric::gldata` можно получить значения узлов и весовых коэффициентов на интервале [0, 1] для метода Гаусса–Лежандра, который применяется для непрерывной подынтегральной

функции любого вида. Для этого функцию `numeric::gldata` вызывают следующим образом:

<Переменная> := **numeric::gldata**(<количество узлов>,
<точность>)

<Количество узлов> – целое положительное число, указывающее, сколько значений функции предстоит вычислить, чтобы рассчитать по ним интеграл. Параметр <точность> (тоже целое положительное число) указывает, сколько точных (верных) значащих цифр должны содержать узлы и весовые коэффициенты, записанные в десятичном формате. Чаще всего параметр <точность> задают равным переменной окружения `DIGITS`, чтобы точность вычисления интегралов и узлов была одинаковой. Функция `numeric::gldata` также возвращает список `[]`, состоящий, в свою очередь, из двух списков: [список коэффициентов c_k], [список узлов t_k]. Например, в результате выполнения команды `numeric::gldata(3, DIGITS)` можно получить список [[0.2777777778, 4/9, 0.2777777778], [0.1127016654, 1/2, 0.8872983346]]. В методе Гаусса–Лежандра узлы также располагаются на интервале [0, 1] симметрично относительно середины интервала, а коэффициенты попарно симметричны. Метод является точным, если подынтегральная $f(x)$ – полином, степень которого не выше, чем $2n - 1$ (где n – используемое в методе количество узлов).

Получив с помощью одной из перечисленных выше функций значения весовых коэффициентов и узлов, можно самостоятельно рассчитать численное значение интеграла на заданном интервале, просуммировав значения подынтегральной функции с соответствующими весовыми коэффициентами по квадратурной формуле:

$$\int_a^b f(x)dx \approx (b-a)\sum_{k=1}^{n} c_k \cdot f(a+(b-a)t_k).$$

Эти вычисления (одним из перечисленных методов) можно также выполнить с помощью функции `numeric::quadrature`. При этом она не только автоматически получает нужные весовые коэффициенты и узлы (вызвав соответствующую функцию) и высчитывает по ним соответствующую квадратуру, но и доводит вычисление интеграла до заданной с помощью переменной окружения `DIGITS` точности. Для этого исходный интервал интегрирования разбивается на части, для каждой из которых выполняется вычисление интеграла по соответствующей квадратурной формуле. Искомый интеграл рассчитывается как сумма интегралов,

каждый из которых вычислен на своей части исходного интервала. Такие расчеты выполняются для разного количества разбиений исходного интервала, пока заданное количество значащих цифр не начнет сохраняться или пока количество итераций не превысит установленный предел.

Для вычисления численного значения определенного интеграла функцию `numeric::quadrature` вызывают следующим образом:

<Переменная> := **numeric::quadrature**(<функция>,
<аргумент> = <начало> .. <конец>, <опции>)

В качестве параметра <функция> следует указать символьное выражение, которое необходимо проинтегрировать. На месте параметра <функция> это выражение можно записать либо непосредственно, либо с помощью идентификатора, которому ранее присвоено соответствующее выражение, либо с помощью вызова объявленной ранее функции с символьными аргументами. При вычислении однократного интеграла <функция> может содержать только одну не заданную ранее символьную переменную. Идентификатор именно этой переменной необходимо использовать в качестве <аргумента>. Параметры <начало> и <конец> указывают границы интервала, на котором необходимо выполнить интегрирование. В качестве границ интервала можно использовать как числовые значения, так и математические выражения, записанные на месте параметров <начало> и <конец> непосредственно или через идентификаторы переменных, которым эти выражения (значения) были ранее присвоены. Но если параметры <начало> и <конец> заданы как выражения, в них должны отсутствовать не заданные ранее символьные переменные (т.е. результатом вычисления данных выражений должны являться числовые значения, которые можно представить в десятичном виде).

Например, для численного вычисления интеграла $\int_1^{10} \cos x\,dx$ можно написать команду `numeric::quadrature(cos(x), x=1..10)`, в результате в поле вывода получится значение –1.385492096.

Функцию `numeric::quadrature` можно вызвать со следующими <опциями>:

`GaussLegendre = n`, или `GaussTschebyscheff = n`, или `NewtonCotes = n`, – с помощью этой опции можно указать, какая квадратурная формула (`GaussLegendre` – метод Гаусса–Лежандра, `GaussTschebyscheff` – метод Гаусса–Чебышева, `NewtonCotes` –

метод Ньютона–Котеса) и с каким количеством узлов (`n`) должна использоваться при вычислении интеграла. Если эта опция не указана (рис. 6.9), то по умолчанию используется метод Гаусса–Лежандра с `n = 20`, если переменная окружения `DIGITS ≤ 10` (`n = 40`, если `10 < DIGITS ≤ 50`; `n = 80`, если `50 < DIGITS ≤ 100`; `n = 160`, если `DIGITS > 100`). Обратите внимание! Метод Гаусса-Чебышева используется только с опцией `Adaptive = FALSE`;

`Adaptive = TRUE` или `Adaptive = FALSE` – позволяет включить (отключить) проверку ошибок при вычислении по квадратурной формуле;

`MaxCalls = m` – задает максимальное количество итераций (попыток вычислить интеграл с заданной точностью). Если эта опция не указана, то по умолчанию `m = MAXDEPTH·n`, где `n` – количество узлов квадратурной формулы, а `MAXDEPTH` – переменная окружения, которая задает максимальную глубину рекурсии для функции (по умолчанию `MAXDEPTH = 500`). Если заданное количество итераций превышено, а указанная точность не была достигнута, MuPAD выдает предупреждающее сообщение.

Например, чтобы численно посчитать интеграл $\int_1^{10} \frac{\cos x}{x} dx$ с помощью квадратурной формулы Ньютона–Котеса, содержащей два узла (т.е. с помощью метода трапеций), достаточно написать команду

```
numeric::quadrature(cos(x)/x, x=1..10, NewtonCotes = 2)
```

Но для достижения точности 10^{-9} (`DIGITS = 9`) оказывается недостаточно 1 000 итераций, заданных по умолчанию. Для вычисления этого интеграла методом трапеций необходимо около 20 000 итераций. Это количество итераций можно задать с помощью опции `MaxCalls` (см. рис. 6.9). В то же время методу Гаусса–Лежандра (также использующему 2 узла) потребуется меньше 1 000 итераций для вычисления этого интеграла с заданной точностью.

Использование функции `numeric::int` для численного интегрирования равносильно использованию функции `numeric::quadrature`. Функция `numeric::int` вызывает функцию `numeric::quadrature`, поэтому имеет тот же синтаксис, параметры и опции (см. рис. 6.9).

```
<Переменная> := numeric::int(<функция>,
                <аргумент> = <начало> .. <конец>, <опции>)
```

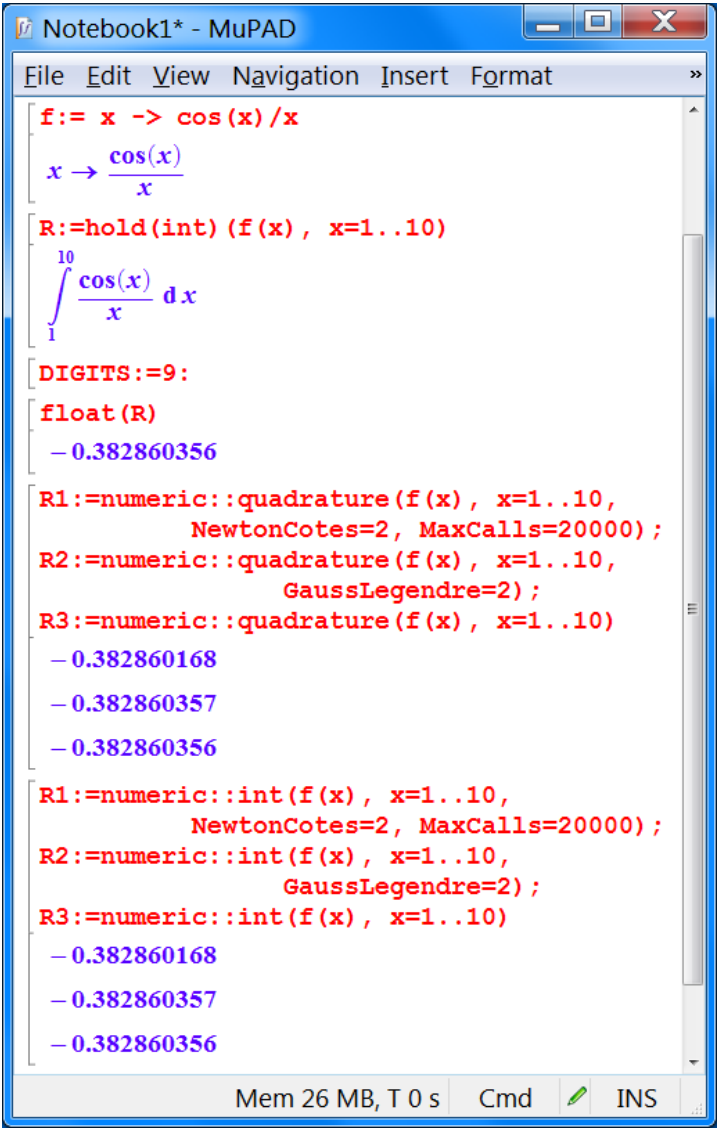


Рис. 6.9. Вычисление интеграла численно

Отличие функции `numeric::int` от функции `numeric::quadrature` состоит только в том, что функция `numeric::int` не выдает предупреждающих сообщений.

Например, чтобы численно вычислить интеграл $\int_1^{10} \frac{\cos x}{x} dx$, можно вместо команды `numeric::quadrature(cos(x)/x, x=1..10)` использовать команду `numeric::int(cos(x)/x, x=1..10)`. При выполнении обеих этих команд в поле вывода получится один и тот же результат: –0.3828603559.

Если функция `numeric::quadrature` или функция `numeric::int` не может вычислить интеграл, то в поле вывода дублируется введенная команда, как при отложенных вычислениях.

С помощью кнопки `Numerical integration` (численное интегрирование), расположенной на раскрывающейся панели кнопок `Integral` $\int f dx$ командной панели `Command Bar`, в программу можно вставить шаблон для численного вычисления определенного интеграла: `numeric::int(#f, #x=#a..#b)`. Этот шаблон содержит поля, отмеченных символом `#`. Каждое из полей шаблона следует заполнить в соответствии с описанными выше правилами: в поле `#f` необходимо указать `<функцию>`, в поле `#x` – идентификатор `<аргумента>`, в полях `#a` и `#b` – `<начало>` и `<конец>` интервала, на котором необходимо выполнить интегрирование. Кроме того, шаблон `float(hold(int) (#f, #x))` для представления результата интегрирования в десятичном формате можно вставить в текст программы с помощью последовательности раскрывающихся списков `General Math` → `Calculus` → `Integrate Numerically`, расположенной на той же командной панели `Command Bar`.

6.6.3 Площадь фигуры под кривой

В соответствии с геометрическим смыслом определенного интеграла площадь плоской фигуры под кривой $y = f(x)$ на интервале $[a, b]$ равна определенному интегралу от функции $f(x)$ на этом интервале. Поэтому программа для вычисления площади под графиком математической функции, зависящей от одной переменной, должна содержать следующую последовательность действий:

1) объявить функцию;

2) задать границы интервала;

3) вычислить интеграл от объявленной функции на заданном интервале (при необходимости представить результат вычисления в десятичном виде);

4) построить график функции, на котором заштриховать область между кривой и горизонтальной осью на заданном интервале (см. разд. 3.4).

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
f := x -> -x^4 + 2*x^3 + 11*x^2 - 12*x + 35;
a := -3: b := 4:
S := int(f(x), x=a..b);
func := plot::Function2d(f(x), x=a-1..b+1):
h := plot::Hatch(func, a..b):
plot(h, func, LineWidth=0.5*unit::mm)
```

Если интеграл аналитически не вычисляется, то для нахождения площади фигуры вместо функции `int` следует использовать функцию `numeric::int`. Например, `S := numeric::int(f(x), x=a..b)`.

Аналогичным образом можно вычислить объем под двумерной поверхностью. Для этого необходимо вычислить двумерный интеграл в заданных пределах.

6.6.4. Площадь плоской фигуры, заключенной между двумя кривыми

Площадь плоской фигуры, заключенной между двумя кривыми $y = f_1(x)$ и $y = f_2(x)$ на интервале $[a, b]$ (ниже, чем $f_1(x)$, но выше, чем $f_2(x)$), вычисляется как

$$S = \int_a^b dx \int_{f_2(x)}^{f_1(x)} dy = \int_a^b f_1(x)dx - \int_a^b f_2(x)dx .$$

Поэтому программа для вычисления площади фигуры, заключенной между графиками двух математических функций, каждая из которых зависит от одной переменной, должна содержать следующую последовательность действий:

1) объявить первую функцию;

2) объявить вторую функцию;

3) задать границы интервала;

4) вычислить соответствующие интегралы на заданном интервале;

5) построить график двух функций, на котором заштриховать область между кривыми на заданном интервале (см. разд. 3.4).

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
f1 := x -> -x^4 + 2*x^3 + 11*x^2 - 12*x + 35;
f2 := x -> -20-10*x;
a := -3: b := 4:
S := int(f1(x), x=a..b) - int(f2(x), x=a..b);
func1 := plot::Function2d(f1(x), x=a-1..b+1):
func2 := plot::Function2d(f2(x), x=a-1..b+1):
h := plot::Hatch(func1, func2, a..b):
plot(h, func1, func2, LineWidth=0.5*unit::mm)
```

Если интегралы аналитически не вычисляются, то вместо функции `int` следует использовать функцию `numeric::int`. Например:

```
S := numeric::int(f1(x), x = a..b) -
                numeric::int(f2(x), x=a..b);
```

6.6.5. Положение центра инерции плоской фигуры

Декартовы координаты центра инерции плоской (двумерной) фигуры, заключенной между двумя кривыми $y = f_1(x)$ и $y = f_2(x)$ на интервале $[a, b]$ (ниже, чем $f_1(x)$, но выше, чем $f_2(x)$), вычисляются как

$$x_C = \frac{1}{S}\int_a^b x dx \int_{f_2(x)}^{f_1(x)} dy\,,\ \ y_C = \frac{1}{S}\int_a^b dx \int_{f_2(x)}^{f_1(x)} y dy\,,$$

$$\text{где } S = \int_a^b dx \int_{f_2(x)}^{f_1(x)} dy = \int_a^b f_1(x)dx - \int_a^b f_2(x)dx\,.$$

Поэтому программа для вычисления координат центра инерции двумерной фигуры, заключенной между графиками двух математических функций, каждая из которых зависит от одной переменной, должна содержать следующую последовательность действий:

1) объявить первую функцию;

2) объявить вторую функцию;

3) задать границы интервала;

4) вычислить соответствующие интегралы на заданном интервале;

5) построить график двух функций, на котором заштриховать область между кривыми на заданном интервале и отметить положение центра инерции (см. разд. 3.4).

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
f1 := x -> -x^4 + 2*x^3 + 11*x^2 - 12*x + 35;
f2 := x -> -20 - 10*x;
a := -3: b := 4:
S := int(f1(x), x=a..b) - int(f2(x), x=a..b);
X := int(int(x, y=f2(x)..f1(x)), x=a..b):
Y := int(int(y, y=f2(x)..f1(x)), x=a..b):
XC := X/S; YC := Y/S;
func1 := plot::Function2d(f1(x), x=a-1..b+1):
func2 := plot::Function2d(f2(x), x=a-1..b+1):
h := plot::Hatch(func1, func2, a..b):
point := plot::PointList2d([[XC, YC]],
                    PointSize=2*unit::mm):
plot(h, func1, func2, point, LineWidth=0.5*unit::mm)
```

Если интегралы аналитически не вычисляются, то вместо функции `int` следует использовать функцию `numeric::int`. Например:

```
S := numeric::int(f1(x), x=a..b) -
                 numeric::int(f2(x), x=a..b);
```

Аналогично можно вычислить положение центра инерции объемной трехмерной фигуры. Для этого необходимо вычислить три тройных интеграла: $\iiint x\,dxdydz$, $\iiint y\,dxdydz$ и $\iiint z\,dxdydz$, каждый из которых разделить на объем фигуры $\iiint dxdydz$.

Глава 7. РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Линейная алгебра – раздел алгебры, изучающий линейные математические объекты, такие как линейные (векторные) пространства, отображения, системы уравнений и т.д. Одним из базовых понятий линейной алгебры является матрица. Под ***матрицей*** понимают двумерную прямоугольную таблицу, состоящую из объектов одинаковой природы (вещественных или комплексных чисел, функций, векторов и т.д.), которые принято называть элементами матрицы. Расположение каждого элемента в матрице однозначно задается номером (индексом) строки и номером (индексом) столбца, в которых он находится. Размер матрицы определяется количеством строк и столбцов в ней. В матричном представлении числа и векторы интерпретируются как частные случаи матрицы: векторы – матрицы, имеющие только одну строку (или только один столбец), а числа – матрицы, имеющие только одну строку и один столбец. Строки и столбцы матриц принято нумеровать, начиная с единицы.

В этой главе описаны функции MuPAD, позволяющие использовать матрицы и операции с ними в символьных вычислениях.

Обратите внимание! Значения, возвращаемые каждой из описанных в этой главе функций, рекомендуется присваивать отдельным переменным. Это облегчит их дальнейшее использование в программе. Если полученные результаты далее в программе использовать не планируется, то для всех описанных ниже функций допускается вызов без присвоения.

7.1. Матрицы и векторы (создание)

Матрицы содержат одновременно несколько значений (элементов). Чтобы в переменную можно было поместить несколько значений, в MuPAD ее необходимо специальным образом объявить (написать специальные указания).

7.1.1. Создание матрицы произвольного вида

В программировании матрицы являются частным случаем двумерных массивов, которые, так же как и матрицы, представляют собой совокупность однотипных данных, пронумерованных с помощью двух индексов (номера строки и номера столбца). Но у массивов нумерация строк и столбцов может начинаться с любого удобного для решаемой задачи целого числа, а у матриц – только с единицы. Поэтому в MuPAD матрицы можно объявлять так же, как двумерные массивы (см. разд. 10.5.2). Но так как использование матриц получило широкое распространение не только в линейной алгебре, но и в других разделах математики, а также в физике, инженерных науках и т.д., в MuPAD предусмотрена специальная функция `matrix`, которая позволяет создать (объявить) матрицу. Для этого функцию `matrix` вызывают следующим образом:

```
<Переменная> := matrix(<количество строк>,
                <количество столбцов>, <значения>, <опция>)
```

Параметры `<количество строк>` и `<количество столбцов>` задают размеры будущей матрицы, поэтому они должны быть записаны с помощью целых положительных чисел или арифметических выражений, результатом вычисления которых являются целые положительные числа. Количество строк (столбцов) в матрице не должно превосходить 2^{31}.

Возвращаемая функцией `matrix` матрица отображается в поле вывода в традиционном для матриц виде: как двумерная таблица в круглых скобках. Если вызов функции `matrix` выполняется с присвоением результата какой-либо `<переменной>`, то далее в программе идентификатор этой `<переменной>` будет интерпретироваться как идентификатор матрицы.

При создании матрицы в нее можно сразу же поместить значения всех или некоторых элементов. Для этого используется параметр `<значения>`. Так как линейная алгебра имеет дело в основном с числовыми матрицами, то по умолчанию функция `matrix` заполняет созданную ею матрицу нулями. Поэтому если параметр `<значения>` опустить, можно получить матрицу заданного размера, состоящую из нулей. Например, в результате выполнения команды `A := matrix(2, 3)` в поле вывода можно получить матрицу `A` в виде: $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

Если при создании матрицы необходимо задать значения ее элементов, то параметр <значения> следует указать в виде списка `[]`. Элементами списка могут являться как числовые значения, так и символьные выражения. По сути, функция `matrix` выполняет преобразование списка в матрицу. Расположение элементов в списке можно организовать по-разному.

Элементы можно перечислять в списке подряд через запятую. Например, [1, 2, 3, 4, 5, 6] или [1, x + 2, 3, x – 4, 5, 6 – x]. Тогда количество элементов должно строго соответствовать размеру матрицы. Если при вызове функции `matrix` указано, что матрица должна иметь `n` строк и `m` столбцов, то первые `m` чисел, взятые из списка, будут помещены в первую строку, следующие `m` чисел – во вторую и т.д. пока вся матрица не будет заполнена. Список такого вида можно написать вручную или создать с помощью оператора `$` (см. разд. 4.1, 6.3, 6.4). Например: `[3 $ n*m]` – создает список, состоящий из `n` × `m` троек (все элементы матрицы равны 3); `[$ 3..9 step 2]` – список, состоящий из нечетных чисел [3, 5, 7, 9]; `[x^2 $ x = -2..3]` – список [4, 1, 0, 1, 4, 9] и т.д. Кроме того, список можно создать с помощью функции, связывающей индексы `i`, `j` (номер строки и номер столбца элемента матрицы соответственно) со значением элемента: `<Имя> := (i, j) -> <a_i_j>`

Для такой функции вместо параметра `<a_i_j>` необходимо написать математическое выражение, в котором используются (или не используются) аргументы этой функции (`i` и `j`). Параметр <имя> – идентификатор функции. Например: `f := (i, j) –> i+j-1` или `h := (i, j) –> x^(j-1)`. Если идентификатор <имя> такой функции использовать вместо параметра <значения> функции `matrix`, то индексы `i` и `j` будут изменяться автоматически, начиная с единицы до указанного <количества строк> и <количества столбцов> соответственно. Более того, такую функцию можно объявить непосредственно на месте параметра <значения>. То есть если предварительно объявлена функция `f := (i, j) -> i+j-1`, то в результате выполнения двух следующих команд:
`matrix(2, 3, f)` и `matrix(2, 3, (i, j) -> i+j-1)`
в поле вывода получится одна и та же матрица: $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}$.

Рассмотрим другой способ задания параметра <значения>. Список значений элементов матрицы может состоять из списков, в каждом из которых перечислены элементы одной строки. Например: [[1, 2, 3], [4, 5, 6]] или [[3, 5], [7, 9]]. Если все элементы матрицы перечислены таким способом, параметры <количество строк> и <количество столбцов> можно опустить. Например: `matrix([[1, 2, 3], [4, 5, 6]])`. Кроме того, можно опускать (не указывать) нулевые элементы, расположенные в конце каждой из строк. Например, в результате выполнения команды `matrix([[3], [0, 4], [0, 2, 5]])` можно получить следующую матрицу:

$$\begin{pmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 2 & 5 \end{pmatrix}.$$

Список элементов каждой строки можно записать непосредственно в списке элементов или указав идентификатор переменной, которой этот список был ранее присвоен (рис. 7.1). Например, если предварительно создать два списка: `v1 := [a^i $ i=1..3]` и `v2 := [b^i $ i=0..3]`, то с помощью команды `A := matrix(2, 4, [v1, v2])` из этих списков можно составить матрицу: $\begin{pmatrix} a & a^2 & a^3 & 0 \\ 1 & b & b^2 & b^3 \end{pmatrix}$.

Если вместо параметра <значение> указать не список элементов, а одно число (например, `matrix(3, 2, 5)`), то будет создана матрица, все элементы которой равны указанному числу. При этом параметры <количество строк> и <количество столбцов> необходимо обязательно указать.

Функцию `matrix` можно использовать с одной из двух <опций> (`Diagonal`, `Banded`), только если параметр <значения> задается в виде общего списка значений элементов без явного деления на строки. При использовании каждой из этих <опций> функция `matrix` создает диагональные матрицы двух различных видов. MuPAD содержит несколько функций для создания матриц специального вида. Эти функции будут рассмотрены позже (см. разд. 7.1.2). В том же разделе будут подробно рассмотрены способы создания диагональных матриц с помощью <опций> функции `matrix`.

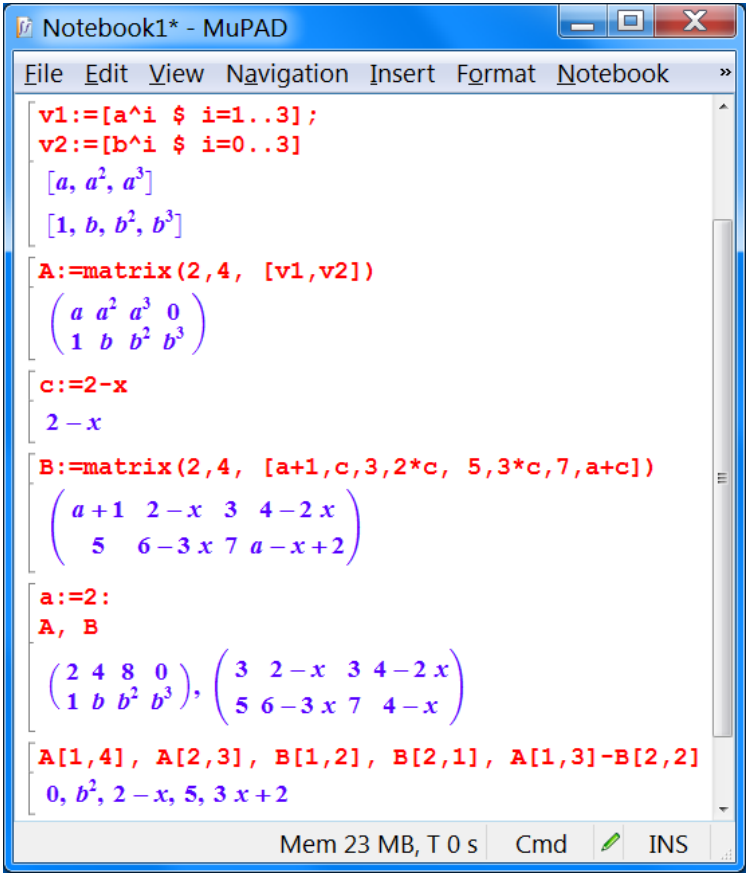


Рис. 7.1. Создание матриц произвольного вида

Списки элементов, используемые для создания матриц, могут содержать как числа, так и символьные переменные (см. рис. 7.1). Если символьной переменной ранее было присвоено какое-то значение (числовое или символьное), то это значение автоматически подставляется в список и в создаваемую с его помощью матрицу. Если значение символьной переменной присваивается после создания матрицы, то при первом же обращении к этой матрице (например, для ее вывода) значения всех заданных

на данный момент символьных переменных автоматически в нее подставляются. Если для создания матрицы вместо функции `matrix` использовать функцию `densematrix`, то такие подстановки становятся невозможными – после создания матрицы все символьные обозначения в ней будут сохраняться неизменными, несмотря на любые преобразования соответствующих символьных переменных в программе. В матрицу, объявленную с помощью функции `densematrix`, значения символьных переменных автоматически подставляются только в момент ее создания. В остальном правила использования функции `densematrix` такие же, как правила использования функции `matrix`.

С помощью раскрывающейся панели кнопок `Matrices and vectors` , расположенной на командной панели `Command Bar`, в программу можно вставить 25 шаблонов матриц размером 1×1, 1×2, …, 2×1, 2×2, 2×3, …, 5×1, …, 5×5. Каждый шаблон, вставленный с помощью любой из кнопок, расположенных на этой панели, содержит поля, отмеченных символом `#`. Например, шаблон для матрицы размером 2×3 будет иметь вид: `matrix([[#_1_1, #_1_2, #_1_3], [#_2_1, #_2_2, #_2_3]])`. Каждое из полей шаблона необходимо заполнить значениями элементов матрицы в соответствии с описанными выше правилами.

Идентификатор `<переменной>`, которой присвоена матрица при ее создании, служит как для обращения к матрице в целом, так и для обращения к ее отдельным элементам. Для этого после идентификатора `<переменной>` необходимо указать в квадратных скобках индексы (номер строки и номер столбца) соответствующего элемента матрицы. Например, если в программе с помощью команды `A := matrix(2, 4, [1, 2, 3, 4, 5, 6, 7, 8])` была объявлена матрица `A`, то команда `A, A[1, 4], A[2, 3]` позволяет вывести и матрицу целиком, и значения двух ее элементов:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}, 4, 7$$

При создании матрицы с помощью функции вида `(i, j) -> <a_i_j>`, аргументами которой являются номера строк и столбцов, при записи выражения `<a_i_j>` можно также использовать элементы другой (созданной ранее) матрицы с индексами `i` и `j` соответственно. Например, если ранее была объявлена матрица `A := matrix(2, 4, [1, 2, 3, 4, 5, 6,`

7, 8]), то из этой матрицы с помощью команды B := matrix(2, 4, (i, j) -> A[i, j] + 1) можно создать следующую матрицу:

$$\begin{pmatrix} 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{pmatrix}.$$

Под вектором в линейной алгебре понимают матрицу-строку или матрицу-столбец. Поэтому для создания векторов используют ту же функцию matrix, при вызове которой один из размеров (количество строк или количество столбцов) задают равным единице. В этом случае параметр <значения> указывают в виде общего списка без выделения элементов, относящихся к отдельным строкам. Если при этом не указывать размеры матрицы (опустить параметры <количество строк> и <количество столбцов>), то получится вектор-столбец. Например, в результате выполнения команды matrix(1, 3, [3, 4, 5]) получится вектор-строка; а в результате выполнения команд matrix(3, 1, [3, 4, 5]) или matrix([3, 4, 5]) – вектор-столбец.

Для обращения к элементам вектора (матрицы-строки или матрицы-столбца) достаточно после идентификатора <переменной> указать в квадратных скобках один индекс (номер строки или номер столбца соответственно). Например, если в программе с помощью команды B := matrix(1, 6, [6, 5, 4, 3, 2, 1]) был объявлен вектор-строка B, то команда B, B[1], B[3] позволяет вывести и вектор целиком, и значения двух его элементов:

$$\begin{pmatrix} 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}, 6, 4$$

Элементы матрицы или вектора можно использовать в записи выражений. Например, r := x+A[2, 1]−3; если ранее была объявлена матрица A, в которой $a_{21} = 5$, то в результате выполнения такой команды в поле вывода получится значение переменной r в виде: $x + 2$.

В некоторых случаях для вывода больших матриц, отдельные элементы которых равны нулю, вместо вывода по умолчанию удобно пользоваться функцией doprint. Эта функция выводит только отличные от нуля элементы матрицы в виде списка с указанием размеров матрицы и индексов каждого выведенного элемента. Например, если для матрицы $B = \begin{pmatrix} 0 & 2 & 0 & 4 \\ 0 & 0 & 7 & 0 \end{pmatrix}$ написать команду doprint(B), то в поле вывода получится: Dom::Matrix()(2, 4, [(1, 2) = 2, (1, 4) = 4, (2, 3) = 7]).

В остальных случаях для вывода заданных ранее матриц (так же как и для вывода других заданных ранее переменных) чаще всего используют вывод по умолчанию. Для этого в командном поле достаточно указать как отдельный оператор идентификатор матрицы (переменной), которую необходимо вывести (см. разд. 2.3).

При создании матрицы (с помощью функции `matrix` или `densematrix`) происходит преобразование заданного списка в матрицу (двумерный массив специального типа Dom::Matrix). При необходимости матрицу можно вновь преобразовать в список. Для этого используют оператор `$`. С его помощью составляют последовательность из всех элементов матрицы, изменяя каждый из индексов в пределах, заданных размерами матрицы. Полученную последовательность преобразуют в список с помощью квадратных скобок. Например, если необходимо преобразовать в список все элементы матрицы `A` размером `n`×`m`, можно использовать следующую команду: `[A[i, j] $ i=1..n $ j=1..m]`. Аналогичным образом можно преобразовать в список вектор или часть матрицы. Для этого достаточно соответствующим образом указать диапазон изменения индексов элементов матрицы, которые необходимо включить в список.

7.1.2. Создание матриц стандартного вида

В процессе изучения матриц было доказано, что некоторые матрицы специального вида обладают определенными свойствами. Как правило, эти матрицы имеют специальные названия. В табл. 7.1 приведены названия и характерный вид некоторых специальных матриц. Элементами большинства из этих матриц могут быть не только числовые значения, но и символьные выражения. Для быстрого создания таких матриц в MuPAD включены специальные функции или специальные опции у некоторых стандартных функций.

Нулевая матрица – матрица произвольного размера ($n \times m$), все элементы которой – нули (табл. 7.1). Такую матрицу можно создать с помощью функции `matrix`, если не указывать параметр `<значения>` или если этот параметр задан в виде одного числа (0). При этом размеры (количество строк n и столбцов m) следует обязательно указать. Например: `matrix(n, m)` или `matrix(n, m, 0)`, если числовые значения переменных `n` и `m` предварительно заданы в программе.

Диагональная матрица – матрица произвольного размера ($n \times m$), у которой отличными от нуля являются только элементы, расположенные на главной диагонали (см. табл. 7.1). Для создания такой матрицы используют функцию `matrix` с опцией `Diagonal`. При этом список `<значения>` должен содержать только значения диагональных элементов (в виде числовых или символьных выражений). Количество элементов в списке не должно превышать меньший из параметров `<количество строк>` или `<количество столбцов>`. Например, в результате выполнения команды: `matrix(3, 5,[3, 5, 1], Diagonal)` получится матрица, приведенная в табл. 7.1.

Таблица 7.1

Матрицы стандартного вида

$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ Нулевая матрица ($n \times m$)		$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ Матрица из единиц ($n \times m$)
$E = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ Единичная матрица ($n \times n$)	$\begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ Диагональная матрица ($n \times m$)	$\begin{pmatrix} 3 & 4 & 5 & 0 & 0 \\ 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$ Матрица Тёплица ($n \times m$)
$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$ Матрица Паскаля ($n \times n$)	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 & 2^4 \\ 1 & 3 & 3^3 & 3^3 & 3^4 \\ 1 & 4 & 4^2 & 4^4 & 4^4 \\ 1 & 5 & 5^2 & 5^3 & 5^4 \end{pmatrix}$ Матрица Вандермонда ($n \times n$)	$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix}$ Матрица Гилберта ($n \times n$)

Матрица из единиц – матрица произвольного размера ($n \times m$), все элементы которой – единицы (табл. 7.1). Такую матрицу можно создать

с помощью функции `matrix` (без опций), если вместо списка элементов (параметра `<значения>`) указать одно число (1). Например: `matrix(3, 3, 1)` – получится матрица из единиц размером 3×3.

Единичная матрица – диагональная квадратная ($n \times n$) матрица, все элементы которой равны нулю, кроме элементов, расположенных на главной диагонали и равных единице (см. табл. 7.1). Такую матрицу можно создать двумя способами. Если использовать функцию `matrix` с опцией `Diagonal`, то вместо списка элементов достаточно указать одно число (1). Например, в результате выполнения команды `matrix(3, 3, 1, Diagonal)` получится единичная матрица размером 3×3. Кроме того, для создания единичной матрицы существует специальная функция `matrix::identity`. Для ее вызова необходимо указать один единственный параметр – количество строк (или столбцов). Например, команда `matrix::identity(3)` тоже создает единичную матрицу размером 3×3.

Матрица Тёплица – матрица произвольного размера ($n \times m$), вдоль направлений которой, параллельных главной диагонали, расположены одинаковые элементы. Чтобы задать такую матрицу, необходимо указать список чисел или символьных выражений $[u_{-k}, \ldots, u_{-2}, u_{-1}, u_0, u_1, u_2, \ldots, u_k]$, содержащий нечетное количество элементов. При составлении списка необходимо учитывать, что значение u_0, находящееся в середине списка, будет располагаться на главной диагонали; значения, указанные перед u_0, будут располагаться ниже главной диагонали, а значения, указанные после u_0 – выше главной диагонали. Количество значений, которые можно поместить в этот список, зависит от меньшего из размеров матрицы: если количество строк n меньше, чем количество столбцов m, то список должен содержать не более чем $2n - 1$ элемент; если $m < n$, то $2m - 1$ элемент. Например, для создания матрицы Тёплица размером 3×5 потребуется список не более чем из пяти ($2 \times 3 - 1 = 5$) элементов. Имея такой список, в MuPAD можно создать матрицу Тёплица двумя способами: 1) использовать функцию `matrix` с опцией `Banded`; 2) использовать функцию `linalg::toeplitz`. Параметры обеих функций имеют одинаковый смысл. Например, матрицу, приведенную в табл. 7.1, можно получить с помощью выполнения одной из следующих команд: `matrix(3, 5, [1, 2, 3, 4, 5], Banded)` или `linalg::toeplitz(3, 5, [1, 2, 3, 4, 5])`.

Матрица Паскаля – квадратная ($n \times n$) матрица, состоящая из биномиальных коэффициентов вида: $a_{ij} = \binom{i+j-2}{i-1} = \frac{(i+j-2)!}{(i-1)!(j-1)!}$, где i – номер строки, j – номер столбца элемента матрицы. Чтобы создать такую матрицу, можно использовать функцию `linalg::pascal`, которая имеет только один параметр – количество строк (или столбцов) создаваемой матрицы. Например, для получения матрицы Паскаля, приведенной в табл. 7.1, можно использовать следующую команду:

```
A := linalg::pascal(5)
```

Матрица Гильберта – квадратная ($n \times n$) матрица, элементы которой вычисляются следующим образом: $a_{ij} = \frac{1}{i+j-1}$, где i – номер строки, j – номер столбца элемента матрицы (см. табл. 7.1). Чтобы создать такую матрицу, можно использовать функцию `linalg::hilbert`, которая имеет только один параметр – количество строк (или столбцов) создаваемой матрицы. Например, для получения матрицы Гильберта, приведенной в табл. 7.1, можно использовать команду `linalg::hilbert(5)`. С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Hilbert Matrix`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::hilbert(#)` для создания матрицы Гильберта. Поле, отмеченное символом `#`, необходимо заполнить в соответствии с описанными выше правилами.

Матрица Вандермонда – квадратная ($n \times n$) матрица, в каждой строке которой расположены разные степени одного и того же числа, начиная с нулевой (см. табл. 7.1). Чтобы создать такую матрицу, необходимо указать список из чисел или символьных выражений [u_1, u_2, …, u_n]. Тогда каждый элемент матрицы Вандермонда будет вычисляться по формуле: $a_{ij} = (u_i)^{j-1}$, где i – номер строки, j – номер столбца элемента матрицы. Чтобы создать такую матрицу, можно использовать функцию `linalg::vandermonde`, которая имеет только один параметр – список чисел (или символьных переменных). Какую длину n имеет список, такой размер ($n \times n$) будет иметь созданная функцией `linalg::vandermonde` матрица. Например, для получения матрицы Вандермонда, приведенной в табл. 7.1, можно использовать следующую команду:

```
linalg::vandermonde([1, 2, 3, 4, 5])
```

Кроме того, в MuPAD есть функции `linalg::invpascal`, `linalg::invhilbert`, `linalg::invvandermonde`, с помощью которых можно создать матрицы, обратные матрицам Паскаля, Гильберта и Вандермонда соответственно. С правилами их использования можно самостоятельно ознакомиться в справочной системе MuPAD.

Чтобы создать *матрицу*, состоящую *из случайных чисел*, используют функцию `linalg::randomMatrix`, которую вызывают следующим образом:

```
<Переменная> := linalg::randomMatrix(
            <количество строк>, <количество столбцов>,
                        <тип>, <интервал>, <опция>)
```

С помощью параметров `<количество строк>` и `<количество столбцов>` следует задать размеры создаваемой матрицы (так же как при вызове функции `matrix`).

С помощью параметра `<тип>` можно указать, в каком формате будут представлены сгенерированные случайные числа. Например, если вместо параметра `<тип>` указать: `Dom::Integer`, функция `linalg::randomMatrix` будет генерировать целые случайные числа; `Dom::Float` – вещественные случайные числа в десятичном формате; `Dom::Rational` – вещественные случайные числа в виде рациональных дробей. Для генерации случайные чисел разного типа функция `linalg::randomMatrix` использует разные статистические распределения. При генерации целых случайных чисел используется равномерное распределение, а при генерации вещественных – нормальное.

Например, чтобы создать матрицу размером 3×5, состоящую из целых случайных чисел, можно написать команду: `linalg::randomMatrix(3,5, Dom::Integer)`.

С помощью параметра `<интервал>` можно ограничить диапазон генерации случайных чисел, из которых будет создана матрица. Параметр `<интервал>` можно задать в виде: `<начало>..<конец>`. Тогда все сгенерированные ЦЕЛЫЕ ЧИСЛА будут лежать между указанными границами. Например, чтобы создать матрицу размером 3×5, элементами которой будут являться целые случайные числа из интервала [2, 10], можно написать команду `linalg::randomMatrix(3, 5, Dom::Integer, 2..10)`. Если указать только одно число (`<конец>`), то `<начало>` автоматически выбирается равным нулю. Если параметр `<интервал>` не указан, то случайные целые числа генерируются в диапазоне от –1 000 до

1 000. При генерации ВЕЩЕСТВЕННЫХ СЛУЧАЙНЫХ ЧИСЕЛ с помощью параметра <интервал> можно управлять параметрами нормального распределения: математическим ожиданием и дисперсией. Если при генерации вещественных случайных чисел параметр <интервал> не указан, то матрица будет состоять как из положительных, так и из отрицательных чисел, многие из которых расположены вблизи нуля (математическое ожидание равно нулю).

При вызове функции `linalg::randomMatrix` можно использовать одну из следующих <опций>: `Diagonal` (позволяет создать диагональную матрица заданного размера); `Unimodular` (используется только для квадратных матриц, позволяет создать квадратную матрицу с целыми элементами, определитель которой равен +1 или –1). Например, для создания диагональной матрицы размером 3×5, диагональные элементы которой будут являться случайными целыми числами из интервала [2, 10], можно написать следующую команду: `linalg::randomMatrix(3, 5, Dom::Integer, 2..10, Diagonal)`.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Random Matrix`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::randomMatrix(#, #)` для создания матрицы, состоящей из случайных чисел. Поля, отмеченные символом `#`, необходимо заполнить в соответствии с описанными выше правилами.

Обратите внимание, что, в отличие от функции `matrix`, при использовании функции `linalg::randomMatrix` создаются матрицы с элементами фиксированного типа. При последующих операциях значения элементов такой матрицы будут автоматически преобразовываться к указанному типу. Если такое преобразование осуществить невозможно, появится сообщение об ошибке. Для матриц функция `type` (см. разд. 2.3) возвращает не только тип переменной, которой присвоена матрица, но и тип ее элементов. Например, если с помощью команды `A := linalg::randomMatrix(3, 5, Dom::Integer, 2..10)` создана матрица `A`, то с помощью команды `type(A)` в поле вывода можно получить: Dom::Matrix(Dom::Integer); если матрица создана с помощью команды `B := matrix(3, 5)`, то ее тип можно определить с помощью команды `type(B)`. В результате в поле вывода получится: Dom::Matrix(). Это означает, что матрица `B` может иметь элементы произвольного типа.

Чтобы матрицу с элементами одного типа преобразовать в матрицу с элементами другого типа, можно использовать функцию `coerce`, которую вызывают следующим образом:

`<Результат> := coerce(<матрица>, <тип>)`

Здесь в качестве параметра `<матрица>` рекомендуется использовать идентификатор созданной ранее исходной матрицы, а в качестве `<результата>` – идентификатор преобразованной матрицы.

С помощью параметра `<тип>` следует указать, к какому типу необходимо преобразовать исходную `<матрицу>`. Например, если вместо параметра `<тип>` написать: `Dom::Matrix(Dom::Integer)` – в результате преобразования получится матрица, элементами которой могут быть только целые числа; `Dom::Matrix(Dom::Rational)` – получится матрица, элементами которой могут быть вещественные числа в рациональном формате; `Dom::Matrix(Dom::Float)` – матрица, элементами которой могут быть вещественные числа в десятичном формате, и т.д.; `Dom::Matrix()` – матрица с элементами произвольного типа. Например, если после создания (с помощью команды `A := linalg::randomMatrix(2, 3, Dom::Integer, 1..6)`) матрицы `A`, состоящей из целых чисел, выполнить преобразование `B := coerce(A, Dom::Matrix())`, то получится переменная `B` того же типа, как при использовании функции `matrix`, т.е. в результате выполнения команды `type(B)` в поле вывода получится Dom::Matrix(), а не Dom::Matrix(Dom::Integer). С такой матрицей можно выполнять любые преобразования, предусмотренные линейной алгеброй.

При использовании функции `coerce` следует помнить, что преобразование некоторых чисел (а значит и элементов матрицы) к указанному типу может оказаться невозможным. В этом случае в поле вывода появится сообщение об ошибке.

7.2. Изменение матрицы (вектора)

В процессе символьных вычислений может возникнуть потребность изменить созданную ранее матрицу (вектор): задать новые значения некоторым ее элементам; извлечь или удалить строку (столбец), умножить строку (столбец) на число, поменять местами две строки (два столбца), сложить две строки (два столбца), добавить к матрице дополнительные строки (столбцы) и т.д.

Для изменения значения одного элемента ранее созданной матрицы (вектора) достаточно присвоить ему новое значение:

```
<матрица>[<номер строки>, <номер столбца>] :=
                                    <новое значение>
```

Для этого достаточно перед оператором присвоения указать идентификатор <матрицы> с <номером строки> и <номером столбца> элемента, которому необходимо присвоить <новое значение>. <Номер строки> и <номер столбца> присваиваемого элемента после идентификатора <матрицы> записывают в квадратных скобках через запятую. Далее после оператора присвоения (:=) в качестве <нового значения> можно указать числовое значение, арифметическое или символьное выражение. В качестве примера можно рассмотреть следующую последовательность команд:

```
A := matrix(3, 3, [2*x, 5*x, 1], Diagonal);
A[3, 1] := 6-x; A
```

С помощью первой команды была создана диагональная матрицы A размером 3×3:

$$\begin{pmatrix} 2x & 0 & 0 \\ 0 & 5x & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Вторая команда (A[3, 1] := 6-x) заменяет нулевой элемент матрицы, расположенный в третьей строке первого столбца, на выражение $6 - x$. Третья команда выводит матрицу A:

$$\begin{pmatrix} 2x & 0 & 0 \\ 0 & 5x & 0 \\ 6-x & 0 & 1 \end{pmatrix}.$$

При необходимости можно создать матрицу (вектор), состоящую из нулей, а затем через присвоение задать все ее ненулевые элементы.

Более того, в MuPAD любую переменную можно сделать индексированной и присваивать значения отдельным ее элементам. Например, Z[2, 3] := x + 6; Z[2, 4] := 9 - x/2. Но если тип этой индексированной переменной ранее не был конкретизирован (она не была объявлена ни матрицей, ни массивом), то такая переменная будет считаться таблицей. Так как многие операции над матрицами не могут применяться к таблицам, то для корректного решения задач линейной алгебры в MuPAD необходимо сначала объявить соответствующую переменную

матрицей, а потом присваивать ее элементам значения или делать другие преобразования этой матрицы.

Для выполнения таких преобразований, как извлечение или удаление строки (столбца), умножение строки (столбца) на число, перестановка местами двух строк (двух столбцов), сложение двух строк (двух столбцов), добавление к матрице дополнительных строк (столбцов) и т.д., MuPAD предлагает богатый набор функций. Большинство из этих преобразований можно выполнить как со строками, так и со столбцами, поэтому они представлены, как правило, двумя функциями, правила использования которых аналогичны. Примеры использования некоторых из перечисленных ниже функций приведены на рис. 7.2.

Для всех перечисленных ниже функций в качестве параметра `<матрица>` (или `<матрица1>`) необходимо указать идентификатор переменной, которой ранее обозначена исходная матрица.

Для извлечения (создания копии) из исходной матрицы ***подматрицы*** используют функцию `linalg::submatrix`. Эту функцию можно вызвать следующим образом:

```
<Результат> := linalg::submatrix (<матрица>,
                    <список строк>, <список столбцов>)
<Результат> := <матрица>[<диапазон строк>,
                                <диапазон столбцов>]
```

С помощью параметров `<список строк>`, `<список столбцов>` необходимо указать, расположенные в каких строках и столбцах элементы необходимо извлечь из исходной матрицы. Каждый из этих параметров можно записать, перечислив номера строк (столбцов) через запятую в квадратных скобках. Если необходимо выбрать несколько строк (столбцов) подряд, то соответствующий список можно записать в виде диапазона: `<начало>..<конец>`. Если в обоих списках строки и столбцы исходной матрицы выбираются подряд, то для извлечения подматрицы можно обойтись без функции `linalg::submatrix`, а указать идентификатор матрицы и после него в квадратных скобках `<диапазон строк>` и `<диапазон столбцов>` (каждый в виде: `<начало>..<конец>`). Результат будет таким же, как при использовании функции `linalg::submatrix`. Например, команда `linalg::submatrix(A, [1, 3], [2, 4])` извлекает из матрицы `A` элементы, расположенные на пересечении 1-й, 3-й строк и 2-го, 4-го столбцов; команда `linalg::submatrix(A, 1..3, 2..4)` извлекает из матрицы `A` элементы, расположенные

на пересечении 1-й, 2-й, 3-й строк и 2-го, 3-го, 4-го столбцов; команда `A[1..3, 2..4]` дает тот же результат. <Результатом> выполнения функции `linalg::submatrix` всегда является матрица, размер которой не превосходит размера исходной матрицы. Исходная матрица при этом не изменяется.

Для извлечения (создание копии) из исходной матрицы ***строки*** используют функцию `linalg::row`, для извлечения ***столбца*** – функцию `linalg::col`. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::col**(<матрица>,
<список столбцов>)

<Результат> := **linalg::row**(<матрица>, <список строк>)

С помощью параметра <список строк> (или <список столбцов>) необходимо указать, какие именно строки (столбцы) необходимо извлечь из исходной матрицы. Список может содержать один или несколько номеров строк (столбцов). Функции `linalg::col` и `linalg:: row` извлекают указанные столбцы или строки целиком (все элементы от начала до конца). Если <список строк> (или <список столбцов>) содержит несколько значений, то эти значения перечисляются через запятую в квадратных скобках. Если необходимо выбрать несколько строк (столбцов) подряд, то соответствующий список можно заменить диапазоном вида: <начало>..<конец> (без квадратных скобок). Если <список строк> (или <список столбцов>) содержит только один номер строки (столбца), то вместо списка можно указать одно число, а квадратные скобки опустить. Например, команда `linalg::col(A, 3)` извлекает из матрицы `A` третий столбец; команда `linalg::col(A, 2..4)` извлекает из матрицы `A` три столбца с номерами 2, 3, 4; команда `linalg::col(A, [1, 3, 6])` тоже извлекает из матрицы `A` три столбца, но с номерами 1, 3, 6. В зависимости от того, сколько строк (столбцов) необходимо извлечь, <результатом> выполнения функций `linalg::col` и `linalg::row` может быть либо один вектор, либо список векторов (в квадратных скобках через запятую). Исходная матрица при этом не изменится.

Для удаления из исходной матрицы ***строки*** используют функцию `linalg::delRow`, а для удаления ***столбца*** – функцию `linalg:: delCol`. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::delCol**(<матрица>,
<список столбцов>)

<Результат> := **linalg::delRow**(<матрица>,
<список строк>)

С помощью параметра <список строк> (или <список столбцов>) необходимо указать, какие именно строки (столбцы) необходимо удалить из исходной матрицы. Список может содержать один или несколько номеров строк (столбцов). Если <список строк> (или <список столбцов>) содержит несколько номеров строк (столбцов), то эти номера перечисляются через запятую в квадратных скобках. Если необходимо выбрать несколько строк (столбцов) подряд, то соответствующий список можно заменить диапазоном вида: <начало>..<конец> (без квадратных скобок). Если <список строк> (или <список столбцов>) содержит только один номер строки (столбца), то вместо списка можно указать одно число, а квадратные скобки также опустить. Например, команда linalg::delCol(A, 3) удаляет из матрицы A третий столбец; команда linalg::delCol(A, 2..4) удаляет из матрицы A три столбца с номерами 2, 3, 4; команда linalg::delCol(A, [1, 3, 6]) тоже удаляет три столбца, но с номерами 1, 3, 6. <Результатом> выполнения каждой из этих функций является матрица, размер которой меньше размера исходной матрицы.

Для умножения на число строки используют функцию linalg::multRow, а для умножения на число ***столбца*** – функцию linalg::multCol. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::multCol**(<матрица>,
<список столбцов>, <число>)

<Результат> := **linalg::multRow**(<матрица>,
<список строк>, <число>)

При вызове этих функций кроме <списка столбцов> (или <списка строк>) необходимо указать <число>, на которое соответствующие столбцы (строки) предстоит умножить. В качестве параметра <число> можно использовать числовое значение или арифметическое выражение, результатом вычисления которого является числовое значение, а также в некоторых случаях – символьную переменную. <Список строк> (или <список столбцов>) может содержать один или несколько номеров строк (столбцов). Правила записи этих списков для функций linalg::multCol и linalg::multRow такие же, как и для предыдущих функций. Например, команда linalg::multCol (A, 3, 10) умножает на 10 третий столбец матрицы A; команда

`linalg::multCol(A, 2..4, 10)` умножает на 10 три столбца с номерами 2, 3, 4; команда `linalg::delCol(A, [1, 3, 6], 10)` тоже умножает на 10 три столбца, но с номерами 1, 3, 6. <Результатом> выполнения функций `linalg::multCol` и `linalg::multRow` является матрица, размер которой совпадает с размером исходной матрицы.

Чтобы поменять местами две строки, используют функцию `linalg::swapRow`, а чтобы поменять местами ***два столбца*** – функцию `linalg::swapCol`. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::swapCol**(<матрица>,
<столбец1>, <столбец2>, <список строк>)
<Результат> := **linalg::swapRow**(<матрица>,
<строка1>, <строка2>, <список столбцов>)

С помощью параметров <столбец1> и <столбец2> указывают, номера столбцов, которые необходимо поменять местами с помощью функции `linalg::swapCol`. Если далее указан <список строк>, то в этих столбцах меняются местами только элементы, расположенные в перечисленных с его помощью строках. Параметры функции `linalg::swapRow` имеют аналогичный смысл. <Список строк> (или <список столбцов>) может содержать один или несколько номеров строк (столбцов). Правила записи этих списков для функций `linalg::swapCol` и `linalg::swapRow` такие же, как и для предыдущих функций. Например, команда `linalg::swapCol(A, 3, 5)` меняет местами элементы третьего и пятого столбцов матрицы `A`; команда `linalg::swapCol(A, 3, 5, 2..4)` меняет местами элементы третьего и пятого столбцов, расположенные только в строках с номерами 2, 3, 4. <Результатом> выполнения функций `linalg::swapCol` и `linalg::swapRow` является матрица, размер которой совпадает с размером исходной матрицы.

Чтобы одну строку умножить на число и прибавить к другой, используют функцию `linalg::addRow`, а чтобы выполнить ту же операцию ***со столбцами*** – функцию `linalg::addCol`. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::addCol**(<матрица>,
<столбец1>, <столбец2>, <число1>, <число2>)
<Результат> := **linalg::addRow**(<матрица>,
<строка1>, <строка2>, <число1>, <число2>)

Notebook1* - MuPAD

File Edit View Navigation Insert Format Notebook Window »

A - исходная матрица
(заполнена целыми случайными числами от 0 до 10)

```
A:=linalg::randomMatrix(3,5, Dom::Integer,10)
```

$$\begin{pmatrix} 1 & 0 & 7 & 3 & 6 \\ 8 & 5 & 8 & 1 & 9 \\ 5 & 3 & 7 & 0 & 4 \end{pmatrix}$$

B1 - извлеченный столбец № 4 (вектор);
B2 - матрица A после удаления столбцов №1 и №3

```
B1:=linalg::col(A,4):
B2:=linalg::delCol(A, [1,3]):
B1, B2
```

$$\begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 & 3 & 6 \\ 5 & 1 & 9 \\ 3 & 0 & 4 \end{pmatrix}$$

C1 - матрица A после умножения на 5 строк №1 и №2;
C2 - матрица C1 после того, как в ней поменяли местами строки №2 и №3

```
C1:=linalg::multRow(A,1..2,5):
C2:=linalg::swapRow(C1,2,3):
C1, C2
```

$$\begin{pmatrix} 5 & 0 & 35 & 15 & 30 \\ 40 & 25 & 40 & 5 & 45 \\ 5 & 3 & 7 & 0 & 4 \end{pmatrix}, \begin{pmatrix} 5 & 0 & 35 & 15 & 30 \\ 5 & 3 & 7 & 0 & 4 \\ 40 & 25 & 40 & 5 & 45 \end{pmatrix}$$

H1 - результат объединения по горизонтали матриц A, B2 и C2

```
H1:=A.B2.C2
```

$$\begin{pmatrix} 1 & 0 & 7 & 3 & 6 & 0 & 3 & 6 & 5 & 0 & 35 & 15 & 30 \\ 8 & 5 & 8 & 1 & 9 & 5 & 1 & 9 & 5 & 3 & 7 & 0 & 4 \\ 5 & 3 & 7 & 0 & 4 & 3 & 0 & 4 & 40 & 25 & 40 & 5 & 45 \end{pmatrix}$$

H2 - результат замены в матрице H1 столбца № 11 на вектор B1

```
H2:=linalg::setCol(H1,11,B1)
```

$$\begin{pmatrix} 1 & 0 & 7 & 3 & 6 & 0 & 3 & 6 & 5 & 0 & 3 & 15 & 30 \\ 8 & 5 & 8 & 1 & 9 & 5 & 1 & 9 & 5 & 3 & 1 & 0 & 4 \\ 5 & 3 & 7 & 0 & 4 & 3 & 0 & 4 & 40 & 25 & 0 & 5 & 45 \end{pmatrix}$$

Mem 22 MB, T 0 s Cmd INS

Рис. 7.2. Преобразование матриц

Для функции `linalg::addCol` с помощью параметров <столбец1> и <столбец2> указывают номера столбцов, с которыми необходимо выполнить преобразование. В качестве параметров <число1> и <число2> можно использовать числовые значения или арифметические выражения, результатами вычисления которых являются числовые значения. Если при вызове функции `linalg::addCol` указаны оба параметра <число1> и <число2>, то <столбец1> умножается на <число1>, <столбец2> умножается на <число2>. Если <число2> не задано, то <столбец2> умножается на единицу. Далее столбцы складываются поэлементно, а полученным после этого столбцом заменяется <столбец2>. <Столбец1> остается без изменений. Параметры функции `linalg::addRow` имеют аналогичный смысл. Например, команда `linalg::addCol(A, 3, 5, 10)` третий столбец матрицы `A` умножает на 10 и прибавляет к столбцу с номером 5; команда `linalg::addCol(A, 3, 5, 10, -2)` третий столбец матрицы `A` умножает на 10 и прибавляет к столбцу с номером 5, умноженному на –2. <Результатом> выполнения функций `linalg::addCol` и `linalg::addRow` является матрица, размер которой совпадает с размером исходной матрицы.

Чтобы одну строку матрицы заменить вектором, используют функцию `linalg::setRow`, а чтобы ***столбец заменить вектором*** – функцию `linalg::setCol`. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::setCol**(<матрица>, <позиция>, <вектор-столбец>)

<Результат> := **linalg::setRow**(<матрица>, <позиция>, <вектор-строка>)

С помощью параметра <позиция> указывают, элементы какого столбца (строки) в исходной матрице необходимо заменить вектором. Для функции `linalg::setCol` параметр <позиция> – это номер столбца, для функции `linalg::setRow` – номер строки. В качестве параметра <вектор-столбец> можно использовать вектор, количество строк в котором совпадает с количеством строк в исходной матрице. В качестве параметра <вектор-строка> можно использовать вектор, количество столбцов в котором совпадает с количеством столбцов в исходной матрице. Чаще всего в качестве параметра <вектор-строка> (или <вектор-столбец>) используют идентификатор созданного ранее объекта

типа матрица-строка (или матрица-столбец соответственно). <Результатом> выполнения функций linalg::setCol и linalg::setRow является матрица, размер которой совпадает с размером исходной матрицы.

Чтобы часть матрицы заменить другой матрицей, используют функцию linalg::substitute. Эту функцию можно вызвать следующим образом:

<Результат> := **linalg::substitute**(<матрица1>,
<матрица2>, <позиция m>, <позиция n>)

В качестве параметра <матрица2> используют матрицу, элементами которой необходимо заменить часть элементов исходной <матрицы1>. С помощью параметров <позиция m> и <позиция n> необходимо указать номер строки и номер столбца элемента <матрицы1>, начиная с которого будет проводиться замена. Например, в результате выполнения команды linalg::substitute(A, B, 2, 3) вместо элемента a_{23} матрицы A с индексами 2 и 3 будет подставлен элемент b_{11} матрицы B с индексами 1 и 1 и т.д.: $a_{23} = b_{11}$, $a_{24} = b_{12}$, …, $a_{33} = b_{21}$, $a_{34} = b_{22}$, … Независимо от размера <матрицы2> <результатом> выполнения функции linalg::substitute является матрица, размер которой совпадает с размером исходной <матрицы1>.

Чтобы объединить две матрицы по горизонтали, используют функцию linalg::concatMatrix, а чтобы объединить две матрицы ***по вертикали*** – функцию linalg::stackMatrix. Эти функции можно вызвать следующим образом:

<Результат> := **linalg::stackMatrix**(<матрица1>,
<матрица2>, <матрица3>, …)
<Результат> := **linalg::concatMatrix**(<матрица1>,
<матрица2>, <матрица3>, …)
<Результат> := <матрица1>.<матрица2>

В качестве параметров функций linalg::concatMatrix и linalg::stackMatrix указывают (через запятую) матрицы, которые необходимо объединить. Функция linalg::concatMatrix присоединяет к исходной <матрице1> справа все перечисленные матрицы последовательно. Функция linalg::stackMatrix присоединяет к исходной <матрице1> снизу все перечисленные матрицы также последовательно. Функция linalg::concatMatrix имеет сокращенную форму <матрица1>.<матрица2>, которая применяется только для

объединения двух матриц. Например: `C := A.B`. В случае объединения по горизонтали <матрица1>, <матрица2>, <матрица3>, … должны иметь одинаковое количество строк, а в случае объединения по вертикали – одинаковое количество столбцов. <Результатом> выполнения функций `linalg::stackMatrix` и `linalg::concatMatrix` является матрица, размер которой превосходит размер каждой из исходных матриц.

7.3. Свойства матрицы (вектора)

Так как в линейной алгебре некоторые операции можно выполнять с матрицами (векторами), обладающими только определенными свойствами, то в процессе символьных вычислений может возникнуть необходимость проверить, является ли та или иная матрица квадратной, унитарной, эрмитовой и т.д.

Например, чтобы определить, является ли матрица квадратной, необходимо сначала определить ее размеры (`n` и `m`). Далее можно воспользоваться, например, функцией `is(n = m)`. Если матрица квадратная (имеет одинаковые размеры), то в результате выполнения этой команды получится `TRUE`.

Размер ранее заданной (созданной или полученной в результате преобразований, вычислений) ***матрицы*** можно определить с помощь функции `linalg::matdim`, для вызова которой необходимо указать только один параметр – матрицу:

<Результат> := **`linalg::matdim`**(<матрица>)

Функция `linalg::matdim` возвращает список, состоящий из двух целых чисел: количества строк и количества столбцов матрицы. В результате выполнения команды `[n, m] := linalg::matdim(A)` можно получить, например, [2, 3].

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Dimension`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::matdim(#)` для определения размера матрицы. Поле, отмеченное символом #, необходимо заполнить в соответствии с описанными выше правилами.

Количество строк матрицы можно получить с помощью функции `linalg::nrows`, а ***количество столбцов*** – с помощью функции

`linalg::ncols`. ***Длину (размерность) вектора*** можно узнать с помощью функции `linalg::vecdim`. Для вызова каждой из этих функций также необходимо указать только один параметр – матрицу или вектор соответственно:

```
<Результат> := linalg::nrows(<матрица>)
<Результат> := linalg::ncols(<матрица>)
<Результат> := linalg::vecdim(<вектор>)
```

Эти функции возвращают по одному числовому значению каждая – количество строк, столбцов матрицы или длину (размерность) вектора соответственно. В результате выполнения команды `m := linalg::ncols(A)` можно получить, например, 3.

Функция `linalg::nonZeros` возвращает ***количество элементов*** матрицы, ***значения которых отличны от нуля***. Для вызова этой функции также необходимо задать только один параметр – матрицу:

```
<Результат> := linalg::nonZeros(<матрица>)
```

Такие свойства, как эрмитовость, унитарность и т.д., имеют смысл только для квадратных матриц и несколько отличаются для комплексных и вещественных матриц (табл. 7.2). Числовая матрица (вектор) считается вещественной, если все ее элементы – вещественные числа. Если хотя бы один элемент матрицы (вектора) является комплексным числом, то матрица считается комплексной. При проверке свойств, перечисленных в табл. 7.2, для вещественной матрицы A используется ее транспонированная матрица A^{T} или транспонированный вектор X^{T}, а для комплексной матрицы A – эрмитово сопряженная матрица $A^{\mathrm{H}} = A^{*\mathrm{T}}$ (комплексно сопряженная и транспонированная) или эрмитово сопряженный вектор $X^{\mathrm{H}} = X^{*\mathrm{T}}$.

Таблица 7.2

Некоторые свойства матриц

Свойство	**Вещественная матрица**	**Комплексная матрица**
Эрмитова матрица A	$A^{\mathrm{T}} = A$	$A^{\mathrm{H}} = A$
Нормальная матрица A	$A^{\mathrm{T}}\cdot A = A\cdot A^{\mathrm{T}}$	$A^{\mathrm{H}}\cdot A = A\cdot A^{\mathrm{H}}$
Унитарная матрица A	$A^{\mathrm{T}}\cdot A = A\cdot A^{\mathrm{T}} = E$	$A^{\mathrm{H}}\cdot A = A\cdot A^{\mathrm{H}} = E$
Положительно определенная матрица A (для любого ненулевого вектора-столбца X)	$X^{\mathrm{T}}\cdot A\cdot X > 0$	$X^{\mathrm{H}}\cdot A\cdot X > 0$

MuPAD содержит несколько стандартных функций для проверки некоторых свойств квадратных матриц, например `linalg:: isHermitian`, `linalg::isUnitary`, `linalg::isPosDef`. Каждая из этих функций возвращает `TRUE` (или `FALSE`), если указанная матрица обладает (не обладает) соответствующим свойством.

Чтобы проверить, является ли матрица ***эрмитовой***, функцию `linalg::isHermitian` вызывают следующим образом:

`<Результат> := linalg::isHermitian(<матрица>)`

Чтобы проверить, является ли матрица ***унитарной***, функцию `linalg::isUnitary` вызывают следующим образом:

`<Результат> := linalg::isUnitary(<матрица>)`

Чтобы проверить, является ли матрица ***положительно определенной***, функцию `linalg::isPosDef` вызывают следующим образом:

`<Результат> := linalg::isPosDef(<матрица>)`

Для этих функций в качестве параметра `<матрица>` рекомендуется использовать идентификатор ранее заданной (созданной или полученной в результате преобразований, вычислений) матрицы. Например, если переменной `A` обозначена созданная ранее матрица Паскаля, то в результате выполнения команды `linalg::isHermitian(A)` получится `TRUE`.

Если элементы `<матрицы>` содержат не заданные ранее символьные переменные, то функциям `linalg::isHermitian`, `linalg::isUnitary`, `linalg::isPosDef` не всегда удается определить, обладает ли указанная `<матрица>` соответствующим свойством. В этом случае может потребоваться с помощью функции `assume` уточнить свойства символьных переменных, через которые записаны элементы исходной `<матрицы>`.

Шаблоны `linalg::isHermitean(#)`, `linalg::isUnitary(#)`, `linalg::isPosDef(#)` для проверки свойств матрицы можно вставить в текст программы с помощью последовательности раскрывающихся списков `General` `Math` → `Matrices` → `Tests` → `Hermitean` (или … → `Orthogonal/Unitary`, или … → `Positive Definite` соответственно), расположенной на командной панели `Command Bar`.

Для проверки остальных свойств матриц необходимо записать соответствующие логические выражения (например, $A^{T}\cdot A = A\cdot A^{T}$) и проверить их истинность с помощью функции `is(<логическое выражение>)`, которая возвращает `TRUE`, если `<логическое выражение>` истинно.

7.4. Операции с матрицами (векторами)

Элементарные операции с матрицами (векторами), такие как сложение, вычитание, умножение, в MuPAD выполняются с помощью тех же арифметических операторов (+, –, *), что и действия с числовыми и символьными значениями и выражениями:

<Результат> := <матрица1> **+** <матрица2>
<Результат> := <матрица1> **–** <матрица2>
<Результат> := <число> ***** <матрица>
<Результат> := <матрица1> ***** <матрица2>

Например, чтобы сложить две созданные ранее (см. разд. 7.1) матрицы A и B одинакового размера, в командном поле можно написать: A+B.

Несмотря на то, что синтаксис этих операций для матриц внешне не отличается от синтаксиса для числовых и символьных значений и выражений, все правила, установленные в линейной алгебре, неукоснительно соблюдаются. Так, сложение матриц (векторов) выполняется поэлементно; при попытке сложить матрицы разного размера выдается сообщение об ошибке; умножение матрицы (вектора) на число также выполняется поэлементно (на число умножается каждый элемент матрицы, вектора); под вычитанием матриц понимают сложение матрицы с другой матрицей, умноженной на (–1): $C = A - B = A + (-1)\cdot B$. Умножение двух матриц выполняется по правилу «строка на столбец», т.е. $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$. Как принято в линейной алгебре, в MuPAD перемножаются только матрицы, у которых количество столбцов <марицы1> равно количеству строк <матрицы2>. При попытке перемножить матрицы другого размера появляется сообщение об ошибке.

<Результат> := <матрица> **^** <число>

Под возведением <матрицы> в целую положительную степень подразумевается многократное перемножение <матрицы> самой на себя: $C = A^3 = A\cdot A\cdot A$. Под возведением <матрицы> в степень (–1) подразумевается вычисление обратной матрицы: $C = A^{-1}$. Следовательно, эти операции выполняются только для квадратных матриц.

<Результат> := **1 /** <матрица>

По аналогии с операциями над числами считается, что $1/A = A^{-1}$, поэтому под делением единицы на матрицу тоже подразумевают вычисление обратной матрицы. Данная операция также может быть выполнена только для квадратной невырожденной <матрицы>. Для вычисления обратной

матрицы в MuPAD используется несколько функций, которые будут подробно описаны в этом разделе далее.

<Результат> := <матрица1> **/** <матрица2>

Под делением матриц понимают умножение <матрицы1> на матрицу, обратную <матрице2>: $C = A/B = A \cdot B^{-1}$. То есть <матрица2> должна быть квадратной и невырожденной, а количество столбцов <матрицы1> должно совпадать с количеством строк <матрицы2>.

<Результат> := <матрица> **/** <число>

Под делением <матрицы> на число понимают умножение матрицы на число, обратное данному <числу>. Например: $C = A/5 = A \cdot 5^{-1}$. То есть при делении <матрицы> на <число> каждый элемент матрицы делится на это число. Например, чтобы созданную ранее матрицу B разделить на 10, в командном поле достаточно написать: B/10.

<Результат> := **diff**(<матрица>, <список переменных>)

Дифференцирование матрицы выполняется поэлементно, например:

$$C = \frac{dA}{dx} \Rightarrow c_{ij} = \frac{da_{ij}}{dx}.$$

В частности, чтобы продифференцировать все элементы ранее заданной матрицы A по dt, в командном поле можно написать diff(A, t). В результате получится матрица того же размера, как и исходная, но с другими элементами.

Для выполнения других операций с матрицами в MuPAD необходимо использовать соответствующие функции. Далее описаны правила вызова этих функций, а на рис. 7.3 приведены примеры использования некоторых из них.

Для нахождения ***комплексно сопряженной матрицы*** (A^*) используют функцию conjugate, для нахождения ***транспонированной*** (A^{T}) – функцию transpose, а для нахождения ***эрмитово сопряженной*** (A^{H}) – функцию htranspose следующим образом:

<Результат> := **conjugate**(<матрица>)

<Результат> := **transpose**(<матрица>)

<Результат> := **linalg::transpose**(<матрица>)

<Результат> := **htranspose**(<матрица>)

Функция linalg::transpose эквивалентна функции transpose. В результате выполнения функции conjugate каждый элемент исходной <матрицы> заменяется его комплексно сопряженным значением. Очевидно, если все элементы исходной матрицы вещественные, то

комплексное сопряжение такую матрицу не изменит. В результате выполнения функции `transpose` (или `linalg::transpose`) каждая строка исходной <матрицы> записывается как столбец результирующей матрицы в том же порядке. Эрмитово сопряжение включает в себя одновременно и комплексное сопряжение, и транспонирование матрицы ($A^{*T} = A^{H}$). Поэтому команды `htranspose(A)` и `transpose(conjugate(A))` позволяют получить один и тот же результат как для комплексных, так и для вещественных матриц.

Например, чтобы транспонировать заданную ранее матрицу `B`, в командном поле можно написать `transpose(B)` или `linalg::transpose(B)`.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Transpose`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::transpose(#)` для выполнения транспонирования матрицы. Поле, отмеченное символом #, необходимо заполнить в соответствии с описанными выше правилами.

След матрицы – это сумма диагональных элементов матрицы. След можно вычислять только для квадратной матрицы. Для этого в MuPAD можно использовать функцию `linalg::tr`, единственным параметром которой является исходная <матрица>:

<Результат> := **linalg::tr**(<матрица>)

<Результатом> вызова функции `linalg::tr` является числовое значение или символьное выражение, в результате вычисления которого можно получить числовое значение.

Например, чтобы вычислить след заданной ранее матрицы `A` (см. рис. 7.3), в командном поле можно написать `linalg::tr(A)`.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Trace`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::tr(#)` для вычисления следа матрицы. В поле, отмеченном символом #, необходимо указать квадратную матрицу в соответствии с описанными выше правилами.

Минором (минором матрицы) называется определитель квадратной матрицы, составленной из одинакового количества неповторяющихся строк и столбцов исходной матрицы. Размер (количество строк или столбцов) матрицы, которая составляется для вычисления минора, называется

порядком минора. Для матрицы размером $n \times n$, выбирая разные строки и столбцы, можно построить миноры порядка n, $n-1$, $n-2$, …, 2, 1. При этом минор порядка n – это определитель исходной матрицы, а минор первого порядка – это один из элементов исходной матрицы.

Рангом матрицы называют наивысший порядок отличного от нуля минора матрицы. Ранг матрицы равен количеству линейно независимых строк (или столбцов) этой матрицы. Ранг можно вычислять как для квадратной, так и для неквадратной матрицы. В MuPAD для аналитического вычисления ранга матрицы можно использовать функцию `linalg::rank`, а для численного вычисления ранга – функцию `numeric::rank`. Эти функции вызывают следующим образом:

```
<Результат> := linalg::rank(<матрица>)
<Результат> := numeric::rank(<матрица>, <точность>,
                                          <опция>)
```

Для вычисления ранга функция `linalg::rank` приводит `<матрицу>` к треугольному виду, выполняя линейные преобразования строк методом Гаусса. При этом линейно зависимые строки обращаются в ноль. Если исходная `<матрица>` содержит символьные переменные, то перед выполнением преобразований каждый элемент матрицы, содержащий символьные переменные, заменяется случайным целым числом от 1 до 10^{10}. Поэтому результат оценки ранга такой матрицы не всегда является корректным.

Функция `numeric::rank` используется только для `<матриц>`, все элементы которых являются числовыми значениями или арифметическими выражениями, результат вычисления которых можно представить в виде числа, записанного в десятичном формате. Для вычисления ранга функция `numeric::rank` сначала находит собственные значения произведения исходной матрицы на ее эрмитово сопряженную. После этого вычисляет положительные квадратные корни из каждого собственного значения. Количество ненулевых результатов таких вычислений совпадает с рангом исходной матрицы. При этом ненулевыми считаются значения, превышающие указанную при вызове функции `<точность>`. Если параметр `<точность>` не указан, то значение `<точности>` по умолчанию вычисляется через постоянную среды `DIGITS` следующим образом `<точность>` = $10^{-\mathtt{DIGITS}}$. При необходимости численное вычисление ранга можно выполнить в десятичном формате с фиксированной (с помощью `<опций>` `Hard` или `HardwareFloats`) или с плавающей

(с помощью <опций> Soft или SoftwareFloats) запятой. По умолчанию используются вычисления с плавающей запятой.

Каждая из функций linalg::rank и numeric::rank возвращает в качестве результата положительное целое число.

Например, чтобы аналитически вычислить ранг заданной ранее матрицы A (см. рис. 7.3), в командном поле можно написать linalg::rank(A).

С помощью последовательности раскрывающихся списков General Math → Matrices → Rank, расположенной на командной панели Command Bar, в текст программы можно вставить шаблон linalg::rank(#) для аналитического вычисления ранга матрицы. В поле, отмеченном символом #, необходимо указать исходную матрицу в соответствии с описанными выше правилами.

Для вычисления ***определителя квадратной матрицы*** в MuPAD можно использовать либо функцию numeric::det, либо функцию det, которая эквивалентна функции linalg::det. Эти функции вызывают следующим образом:

<Результат> := **det**(<матрица>, <опции>)
<Результат> := **linalg::det**(<матрица>, <опции>)
<Результат> := **numeric::det**(<матрица>, <опции>)

В каждом случае в качестве параметра <матрица> необходимо указать квадратную матрицу или идентификатор переменной, которой она ранее обозначена.

Если функция numeric::det вызывается с <опцией> Symbolic, она вычисляет определитель исходной <матрицы> аналитически. Если эта опция не указана, то функция переводит значения всех элементов исходной <матрицы> в десятичный формат. Если такое преобразование сделать не удалось, автоматически выбирается <опция> Symbolic. Для аналитического вычисления определителя функция numeric::det использует *LU*-разложение исходной <матрицы> на две треугольные матрицы, перемножение которых дает исходную <матрицу>. Если функция numeric::det вычисляет определитель численно, то она использует *QR*-разложение исходной <матрицы> на две матрицы, одна из которых является унитарной, а произведение полученных в результате разложения матриц равно исходной. В обоих случаях определитель исходной <матрицы> равен произведению определителей матриц, полученных в результате разложения (*LU* или *QR*).

Если использовать <опцию> MinorExpansion, функция numeric::det выполняет рекуррентное разложение определителя на миноры по первой строке. Рекуррентный алгоритм существенно увеличивает время вычисления определителя, поэтому эту <опцию> рекомендуется использовать для вычисления определителей не выше 10 порядка (как численно, так и аналитически).

<Опция> NoWarning подавляет вывод предупреждающих сообщений.

При необходимости численное вычисление определителя можно выполнить в десятичном формате с фиксированной (с помощью <опций> Hard или HardwareFloats) или с плавающей (с помощью <опций> Soft или SoftwareFloats) запятой. По умолчанию используются вычисления с плавающей запятой.

Функция det (или linalg::det) перед началом вычисления определителя анализирует исходную <матрицу>: содержит ли она символьные переменные, а также пытается выделить подматрицы, используя которые можно упростить вычисление определителя. В любом случае для получения результата функция det (или linalg::det) вызывает функцию numeric::det. Но если в процессе предварительного анализа в исходной <матрице> были обнаружены символьные переменные, то функция numeric::det вызывается с <опцией> Symbolic. Если выявлены подматрицы, позволяющие упростить вычисление определителя с помощью разложения определителя на миноры, то функция numeric::det вызывается с <опцией> MinorExpansion.

При вызове функции det (или linalg::det) можно использовать только две <опции>: MinorExpansion (принудительное использование рекуррентного разложения определителя на миноры по первой строке); Normal (после вычисления определителя результат представляется в нормальном виде; имеет смысл, если результат – символьное выражение).

Таким образом, и результатом, возвращаемым функцией numeric::det, и результатом, возвращаемым функцией det (или linalg:: det), может являться числовое значение (в рациональном или десятичном виде) или символьное выражение, результатом вычисления которого также будет числовое значение.

Например, чтобы вычислить определитель заданной ранее квадратной матрицы A (см. рис. 7.3), некоторые элементы которой являются символь-

ными выражениями, в командном поле можно написать либо `det(A)`, либо `linalg::det(A)`, либо `numeric::det(A, Symbolic)`. Результаты выполнения этих команд будут одинаковыми.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Determinant`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::det(#)` для аналитического вычисления определителя матрицы. Вместо поля, отмеченного символом `#`, необходимо указать квадратную матрицу в соответствии с описанными выше правилами.

Алгебраическим дополнением элемента a_{ij} матрицы A размером $n \times n$ называется произведение $(-1)^{i+j}$ на минор $(n-1)$-го порядка, составленный из всех строк матрицы A, кроме строки с номером i, и из всех столбцов матрицы A, кроме столбца с номером j. ***Союзная***, или ***присоединенная, матрица*** – это матрица, в которой каждый элемент a_{ij} матрицы A заменен его алгебраическим дополнением. В результате умножения исходной матрицы на союзную получится диагональная матрица. В MuPAD для получения союзной матрицы можно использовать функцию `linalg::adjoint`, которую вызывают следующим образом:

`<Результат> := linalg::adjoint(<матрица>)`

В качестве параметра `<матрица>` необходимо указать квадратную матрицу или идентификатор переменной, которой она ранее обозначена.

Функция `linalg::adjoint` возвращает матрицу, размер которой совпадает с размером исходной `<матрицы>`.

Например, чтобы для заданной ранее квадратной матрицы `A` (см. рис. 7.3) получить союзную матрицу, в командном поле можно написать

`linalg::adjoint(A)`

Обратной матрицей называется матрица, результатом умножения исходной матрицы на которую является единичная матрица. Чтобы получить обратную матрицу, достаточно союзную матрицу разделить на определитель исходной матрицы. Квадратная матрица имеет обратную, если ее определитель отличен от нуля. В MuPAD для получения обратной матрицы можно использовать либо функцию `inverse`, либо функцию `numeric::inverse`, которые вызывают следующим образом:

`<Результат> := inverse(<матрица>, <опция>)`

`<Результат> := numeric::inverse(<матрица>, <опции>)`

В качестве параметра `<матрица>` необходимо указать квадратную матрицу или идентификатор переменной, которой она ранее обозначена.

Если функция numeric::inverse вызывается с <опцией> Symbolic, она вычисляет обратную матрицу аналитически. Если эта опция не указана, то функция переводит значения всех элементов исходной <матрицы> в десятичный формат. Если такое преобразование сделать не удалось, автоматически выбирается <опция> Symbolic. Если функция numeric::inverse вычисляет обратную матрицу численно, то она использует метод Гаусса приведения матрицы к треугольному виду с выделением главного элемента (максимального по модулю). Для аналитического вычисления обратной матрицы функция numeric::inverse использует тот же метод Гаусса, но без выделения главного элемента.

<Опция> NoWarning подавляет вывод предупреждающих сообщений.

При необходимости численное вычисление обратной матрицы можно выполнить в десятичном формате с фиксированной (с помощью <опций> Hard или HardwareFloats) или с плавающей (с помощью <опций> Soft или SoftwareFloats) запятой. По умолчанию используются вычисления с плавающей запятой.

Функция inverse перед началом вычисления обратной матрицы анализирует исходную <матрицу>: содержит ли она символьные переменные. В любом случае для получения результата функция inverse вызывает функцию numeric::inverse. Но если в процессе предварительного анализа в исходной <матрице> были обнаружены символьные переменные, то функция numeric::inverse вызывается с <опцией> Symbolic.

При вызове функции inverse можно использовать только одну <опцию>: Normal = TRUE или FALSE (после вычисления обратной матрицы результат представляется в нормальном виде; по умолчанию используется TRUE).

Если обратную матрицу вычислить удалось, функции numeric::inverse и inverse возвращают матрицу того же типа и размера, как у исходной <матрицы>. Если исходная матрица является вырожденной, то обратная матрица для нее не существует. В этом случае функции numeric::inverse и inverse возвращают FAIL (с англ. *неудача*).

Если для инициации поиска обратной матрицы используется одна из следующих команд:

<Результат> := <матрица>^(-1) или

<Результат> := 1/<матрица>

(например, `1/A` или `A^(-1)`), то автоматически вызывается функция `inverse` с опциями по умолчанию, т.е. `inverse(A)`.

Например, чтобы для заданной ранее квадратной матрицы `A` (см. рис. 7.3), некоторые элементы которой являются символьными выражениями, вычислить обратную матрицу, в командном поле можно написать одну из следующих команд: `numeric::inverse(A, Symbolic)`, `inverse(A)`, `A^(-1)` или `1/A`. Все перечисленные команды позволяют получить один и тот же результат.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Inverse`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `1/#` для аналитического вычисления обратной матрицы. Вместо поля, отмеченного символом `#`, необходимо указать идентификатор квадратной матрицы в соответствии с описанными выше правилами.

Одним из широко используемых в линейной алгебре является ***метод Гаусса***. Этот метод позволяет привести матрицу произвольного размера ($n \times m$) к треугольному виду. Для приведения к треугольному виду (получения нулей под главной диагональю) последовательно одну строку матрицы складывают с другой, умноженной на соответствующий коэффициент. Коэффициенты подбирают так, чтобы при сложении строк один из элементов под главной диагональю обращался в ноль. Значение коэффициента зависит от значения элемента матрицы, расположенного на главной диагонали матрицы в соответствующем столбце. Сначала нули получают в первом столбце, потом – во втором и т.д. Перед преобразованием каждого столбца можно выбрать главный элемент – максимальный по модулю элемент, расположенный ниже главной диагонали в этом столбце. После этого две строки матрицы меняют местами так, чтобы главный элемент оказался на главной диагонали. Такие преобразования матрицы позволяют избежать деления на ноль при вычислении коэффициентов. В MuPAD привести матрицу к треугольному виду методом Гаусса можно с помощью функции `linalg::gaussElim`, которую вызывают следующим образом:

```
<Результат> := linalg::gaussElim(<матрица>, <опция>)
```

Здесь в качестве параметра `<матрица>` необходимо указать матрицу (произвольных размеров) или идентификатор, которым она была ранее обозначена.

Notebook1* - MuPAD

File Edit View Navigation Insert Format Notebook

A - исходная матрица

```
A:=matrix([[3,3,1,2],[3,1,2,3],
           [2,1,4,2],[14,5,5,13]])
```

$$\begin{pmatrix} 3 & 3 & 1 & 2 \\ 3 & 1 & 2 & 3 \\ 2 & 1 & 4 & 2 \\ 14 & 5 & 5 & 13 \end{pmatrix}$$

sledA - след матрицы A; rngA - ранг матрицы A;
opredA - определитель матрицы A

```
sledA:=linalg::tr(A):  rngA:=linalg::rank(A):
opredA:=det(A):
sledA, rngA, opredA
```

$$21,\ 4,\ 10$$

A1 - результат приведения матрицы A к треугольному виду;
A2 - союзная матрица для матрицы A;
A3 - обратная матрица для матрицы A;
AA3 - произведение матриц A и A3

```
A1:=linalg::gaussElim(A):
A2:=linalg::adjoint(A):   A3:=inverse(A):
AA3:=A*A3:
A1, A2; A3, AA3
```

$$\begin{pmatrix} 3 & 3 & 1 & 2 \\ 0 & -2 & 1 & 1 \\ 0 & 0 & \frac{17}{6} & \frac{1}{6} \\ 0 & 0 & 0 & -\frac{10}{17} \end{pmatrix}, \begin{pmatrix} -9 & -93 & 25 & 19 \\ 8 & 36 & -10 & -8 \\ -1 & -7 & 5 & 1 \\ 7 & 89 & -25 & -17 \end{pmatrix}$$

$$\begin{pmatrix} -\frac{9}{10} & -\frac{93}{10} & \frac{5}{2} & \frac{19}{10} \\ \frac{4}{5} & \frac{18}{5} & -1 & -\frac{4}{5} \\ -\frac{1}{10} & -\frac{7}{10} & \frac{1}{2} & \frac{1}{10} \\ \frac{7}{10} & \frac{89}{10} & -\frac{5}{2} & -\frac{17}{10} \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Mem 9 MB, T 0 s Cmd INS

Рис. 7.3. Операции с матрицами

Функцию `linalg::gaussElim` можно вызвать только с одной `<опцией>`: `All`. Если опция `All` не используется, то функция `linalg::gaussElim` возвращает матрицу треугольного вида, размер которой совпадает с размером исходной матрицы. Если `<опция>` `All` используется, то функция `linalg::gaussElim` кроме преобразованной матрицы возвращает ее ранг, определитель (если исходная матрица является квадратной), а также множество номеров столбцов, считающихся характеристическими.

Например, чтобы привести ранее заданную матрицу `A` к треугольному виду (см. рис. 7.3), в командном поле можно написать
`linalg::gaussElim(A)`

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Fraction-Free Gaussian Elimination`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::gaussElim(#)` для приведения матрицы к треугольному виду. Вместо поля, отмеченного символом #, необходимо указать матрицу в соответствии с описанными выше правилами.

Метод Гаусса часто используется в процессе вычисления определителя, обратной матрицы, решения системы линейных уравнений, а также в некоторых разложениях матриц.

После приведения матрицы размером $n \times m$ $(m \geq n)$ методом Гаусса к треугольному виду можно продолжить складывать строки, умноженные на числовой коэффициент, так чтобы элементы, расположенные в первых n столбцах над главной диагональю, обратились в ноль. В результате квадратная $(n \times n)$ подматрица исходной матрицы примет диагональный вид. Для приведения к такому виду получать нули над главной диагональю начинают с n-го столбца матрицы, потом переходят к предыдущему и т.д. Эти преобразования называют преобразованиями обратным ходом. Числовые коэффициенты, так же как и в методе Гаусса, зависят от значения элемента матрицы, расположенного на главной диагонали в соответствующем столбце. Но в процессе преобразований обратным ходом менять местами строки матрицы нельзя. Этот метод преобразования матрицы, при котором первые ее n столбцов приводятся к диагональному виду, называется ***методом Гаусса–Жордана***. В MuPAD привести матрицу к частично диагональному виду методом Гаусса–Жордана можно

с помощью функции `linalg::gaussJordan`, которую вызывают следующим образом:

`<Результат> :=` **`linalg::gaussJordan`**`(<матрица>, <опция>)`

В качестве параметра `<матрица>` необходимо указать матрицу (произвольных размеров) или идентификатор, которым она была ранее обозначена.

Функцию `linalg::gaussJordan` (так же как функцию `linalg::gaussElim`) можно вызвать только с одной `<опцией>`: `All`. Если `<опция> All` не используется, то функция `linalg::gaussJordan` возвращает матрицу частично диагонального вида, размер которой совпадает с размером исходной матрицы. Если `<опция> All` используется, то функция `linalg::gaussJordan` кроме преобразованной матрицы возвращает ее ранг, определитель (если исходная матрица является квадратной), а также множество номеров столбцов, которые считаются характеристическими.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Gauss-Jordan Elimination`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::gaussJordan(#)` для приведения матрицы к диагональному виду (частично диагональному). Вместо поля, отмеченного символом `#`, необходимо указать матрицу в соответствии с описанными выше правилами.

Метод Гаусса–Жордана обычно используется для нахождения обратной матрицы, решения системы линейных уравнений, а также в некоторых разложениях матриц.

Под разложением матриц понимают нахождение таких двух матриц заданного вида, произведение которых равно исходной матрице.

LU-разложение – представление исходной матрицы размером $n \times m$ в виде произведения квадратной матрицы L размером $n \times n$ и матрицы U размером $n \times m$. Обе матрицы являются треугольными, но у матрицы L нули расположены над главной диагональю, а у матрицы U – под главной диагональю (рис. 7.4). Для получения таких матриц к исходной матрице по горизонтали присоединяют единичную. Полученную матрицу приводят к треугольному виду так, чтобы в первых n столбцах нули располагались над главной диагональю (аналогично методу Гаусса). Первые n столбцов преобразованной матрицы – это матрица L, остальные – матрица U.

Если в процессе приведения матрицы к треугольному виду необходимо было выполнить перестановку строк, то произведение матриц $L \cdot U$, полученных в результате разложения, будет отличаться от исходной матрицы. В этом случае, чтобы получить исходную матрицу, необходимо не только вычислить произведение $L \cdot U$, но и выполнить обратные перестановки. Для этого одновременно с разложением записывают матрицу перестановок: единичная матрица размером $n \times n$, в которой выполняют такие же перестановки со столбцами, какие были сделаны со строками преобразованной матрицы в процессе ее приведения к треугольному виду. В этом случае исходная матрица равна произведению всех трех полученных в результате разложения матриц. Для выполнения LU-разложения в MuPAD можно использовать либо функцию `linalg::factorLU`, либо функцию `numeric::factorLU`, которые вызывают следующим образом:

<Результат> := **linalg::factorLU**(<матрица>)

<Результат> := **numeric::factorLU**(<матрица>, <опции>)

В качестве параметра <матрица> необходимо указать матрицу (произвольных размеров) или идентификатор, которым она была ранее обозначена.

Если функция `numeric::factorLU` вызывается с <опцией> `Symbolic`, она выполняет разложение исходной <матрицы> аналитически. Если эта опция не указана, то функция переводит значения всех элементов исходной <матрицы> в десятичный формат. Если такое преобразование сделать не удалось, автоматически выбирается <опция> `Symbolic`. Если функция `numeric::factorLU` выполняет разложение численно, то она использует для приведения матрицы к треугольному виду метод Гаусса с выделением главного элемента (максимального по модулю). Для аналитического разложения функция `numeric::factorLU` использует тот же метод Гаусса, но без выделения главного элемента.

<Опция> `NoWarning` подавляет вывод предупреждающих сообщений.

При необходимости численное разложение можно выполнить в десятичном формате с фиксированной (с помощью <опций> `Hard` или `HardwareFloats`) или с плавающей (с помощью <опций> `Soft` или `SoftwareFloats`) запятой. По умолчанию используются вычисления с плавающей запятой.

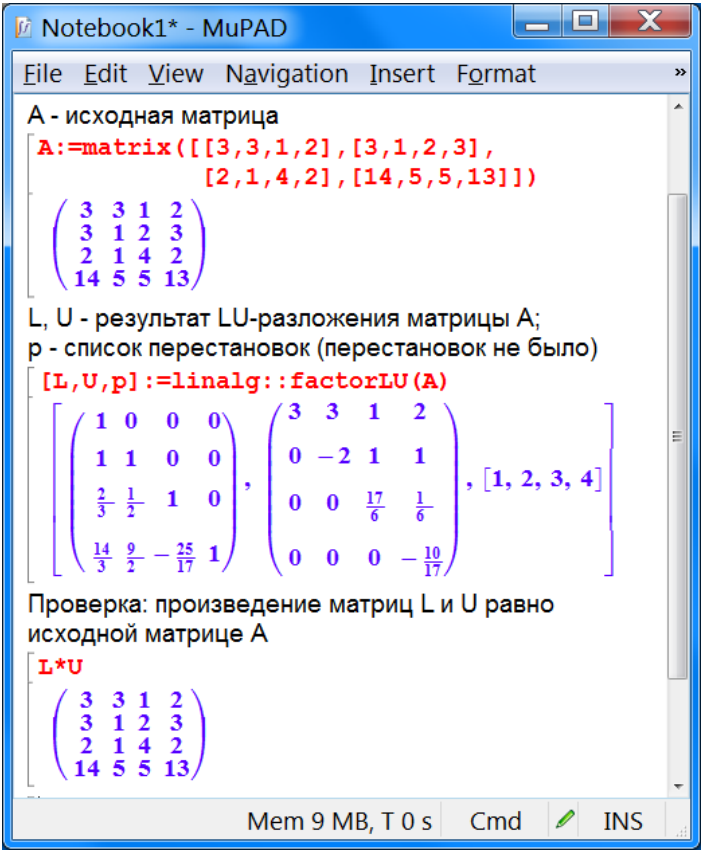


Рис. 7.4. Разложение матриц

Функция `linalg::factorLU` перед началом разложения анализирует исходную <матрицу>: содержит ли она символьные переменные. В любом случае для получения результата функция `linalg::factorLU` вызывает функцию `numeric::factorLU`. Но если в процессе предварительного анализа в исходной <матрице> были обнаружены символьные переменные, то функция `numeric::factorLU` вызывается с <опцией> `Symbolic`.

Каждая из функций `numeric::factorLU` и `linalg::factorLU` возвращает список, состоящий из трех элементов: матрицы L, матрицы U, а также списка перестановок. Список перестановок всегда содержит n элементов. Если эти элементы расположены в порядке возрастания, то перестановки строк в матрице не совершались (см. рис. 7.4). Тогда матрица перестановок совпадает с единичной, а произведение $L \cdot U$ равно исходной матрице. В противном случае, чтобы получить матрицу перестановок, необходимо в единичной матрице выполнить перестановку столбцов в порядке, указанном в полученном списке.

Пример LU-разложения матрицы с помощью функции `linalg::factorLU` приведен на рис. 7.4. Тот же результат можно получить, если вместо команды `[L,U,p]:=linalg::factorLU(A)` использовать команду `[L,U,p]:= numeric::factorLU(A, Symbolic)`.

С помощью последовательности раскрывающихся списков `General Math` → `Matrices` → `Decomposition` → `LU`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `linalg::factorLU(#)` для LU-разложения матрицы. Вместо поля, отмеченного символом `#`, необходимо указать матрицу в соответствии с описанными выше правилами.

QR-разложение – представление исходной матрицы размером $n \times m$ в виде произведения квадратной матрицы Q размером $n \times n$ и матрицы R размером $n \times m$. При этом матрица Q является унитарной, а матрица R – треугольной (нули расположены под главной диагональю). Для получения таких матриц к исходной матрице по горизонтали присоединяют единичную. С полученной матрицей выполняют такие преобразования, чтобы первые ее n столбцов видоизменились в унитарную матрицу – матрицу Q. Остальные столбцы преобразованной матрицы – это матрица R. Для выполнения QR-разложения в MuPAD можно использовать либо функцию `linalg::factorQR`, либо функцию `numeric::factorQR`, которые вызывают следующим образом:

`<Результат> :=` **`linalg::factorQR`**`(<матрица>)`

`<Результат> :=` **`numeric::factorQR`**`(<матрица>, <опции>)`

В качестве параметра `<матрица>` необходимо указать матрицу (произвольных размеров) или идентификатор, которым она была ранее обозначена.

Если функция `numeric::factorQR` вызывается с `<опцией> Symbolic`, она выполняет разложение исходной `<матрицы>` аналитически.

Если эта опция не указана, то функция переводит значения всех элементов исходной <матрицы> в десятичный формат. Если такое преобразование сделать не удалось, автоматически выбирается <опция> Symbolic. Численное и аналитическое разложения исходной <матрицы> выполняются разными методами (Грамма–Шмидта или Хаусхолдера).

<Опция> NoWarning подавляет вывод предупреждающих сообщений.

При необходимости численное разложение можно выполнять в десятичном формате с фиксированной (с помощью <опций> Hard или HardwareFloats) или с плавающей (с помощью <опций> Soft или SoftwareFloats) запятой. По умолчанию используются вычисления с плавающей запятой.

Функция linalg::factorQR перед началом разложения анализирует исходную <матрицу>: содержит ли она символьные переменные. В любом случае для получения результата функция linalg::factorQR вызывает функцию numeric::factorQR. Но если в процессе предварительного анализа в исходной <матрице> были обнаружены символьные переменные, то функция numeric::factorQR вызывается с <опцией> Symbolic.

Если разложение получить удалось, то каждая из функций numeric::factorQR и linalg::factorQR возвращает список, состоящий из двух элементов: матрицы Q и матрицы R. Если разложение получить невозможно, функции возвращают FAIL (с англ. *неудача*).

С помощью последовательности раскрывающихся списков General Math → Matrices → Decomposition → QR, расположенной на командной панели Command Bar, в текст программы можно вставить шаблон linalg::factorQR(#) для QR-разложения матрицы. Вместо поля, отмеченного символом #, необходимо указать матрицу в соответствии с описанными выше правилами.

Кроме LU- и QR-разложений MuPAD также содержит функции для выполнения ряда других разложений матриц, с правилами использования которых можно познакомиться самостоятельно в справочной системе MuPAD.

7.5. Применение матриц (векторов)

7.5.1. Вычисление миноров

Минором (минором матрицы) называется определитель квадратной матрицы, составленной из одинакового количества неповторяющихся строк и столбцов исходной матрицы. Размер (количество строк или столбцов) матрицы, которая составляется для вычисления минора, называется ***порядком минора***. Для матрицы размером $n \times n$, выбирая разные строки и столбцы, можно построить миноры порядка n, $n-1$, $n-2$, …, 2, 1. При этом минор порядка n – это определитель исходной матрицы, а минор первого порядка – это один из элементов исходной матрицы.

Следовательно, чтобы для матрицы размером $n \times n$ вычислить минор $(n-1)$-го порядка, необходимо сначала получить матрицу, которая содержит на одну строку и на один столбец меньше, чем исходная. Для этого достаточно из исходной матрицы удалить одну строку и один столбец. Далее необходимо вычислить определитель полученной матрицы.

Таким образом, для вычисления минора $(n-1)$-го порядка программа должна содержать следующую последовательность действий:

1) создать (задать) матрицу;

2) написать функцию, которая из матрицы удаляет одну строку и один столбец с заданными номерами (индексами);

3) вызвать объявленную функцию, указав в качестве ее параметров номер строки и номер столбца, которые необходимо вычеркнуть; в результате получится матрица размером $(n-1) \times (n-1)$;

4) вычислить определитель полученной матрицы;

5) при необходимости получить новую матрицу, вычеркнув другую строку и столбец, и вычислить ее определитель.

Эту последовательность действий можно реализовать, например, с помощью следующих команд:

```
A := matrix([[3, 3, 1, 2], [3, 1, 2, 3],
            [2, 1, 4, 2], [14, 5, 5, 13]]);
vych := (A, k, p) -> linalg::delRow(
                     linalg::delCol(A, p), k);
B1 := vych(A, 1, 1); minor1 := det(B1);
B2 := vych(A, 2, 1); minor2 := det(B2)
```

Для составления минора $(n-2)$-го порядка достаточно вычеркнуть из исходной матрицы две строки и два столбца. Так как в приведенном

примере при объявлении функции `vych` использовались стандартные функции `linalg::delRow` и `linalg::delCol`, то в соответствии с правилами их использования в качестве параметров `k` и `p` можно использовать не только фиксированные числовые (целочисленные) значения, но списки значений. Поэтому с помощью функции `vych`, объявленной в программе для вычисления минора (n – 1)-го порядка, можно получить также матрицы для вычисления минора (n – 2)-го порядка и т.д., задавая вместо `k` и `p` списки, содержащие по 2 и более чисел. Например, команда `C1 := vych(A, [1, 2], [1, 4])` позволяет получить матрицу `C1`, вычеркнув из матрицы `A` строки с номерами 1 и 2, столбцы с номерами 1 и 4; в результате получится матрица `C1` размером $(n-2)\times(n-2)$. Вычислив ее определитель, можно получить минор (n – 2)-го порядка. Аналогично можно получить миноры других порядков.

7.5.2. Решение систем линейных уравнений. Решение матричных уравнений

Исторически введение матриц связано с задачей решения систем линейных уравнений. Матрицы строились как таблицы коэффициентов систем линейных уравнений, а правила преобразования матриц вытекают из тождественных преобразований этих систем.

В разд. 5.3 были рассмотрены функции `linsolve`, `solve`, `numeric::linsolve` и `numeric::solve`, которые позволяют решать системы линейных уравнений. Однако при использовании этих функций не используется матричное представление системы линейных уравнений (представление в виде одной или двух матриц, составленных из коэффициентов и свободных членов уравнений). В этом разделе рассмотрены функции MuPAD, которые позволяют решать системы уравнений, представленные в матричном виде. В частности, система линейных уравнений в матричном виде записывается, как правило, следующим образом: $A\cdot X = B$ (A – матрица из коэффициентов при неизвестных; B – столбец свободных членов; X – столбец, составленный из неизвестных). Например:

$$\begin{cases} x_1 + x_2 = 1 \\ 2x_1 - x_2 = 3 \end{cases} \Rightarrow A\cdot X = B, \text{ где } A = \begin{pmatrix} 1 & 1 \\ 2 & -1 \end{pmatrix},\ B = \begin{pmatrix} 1 \\ 3 \end{pmatrix},\ X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

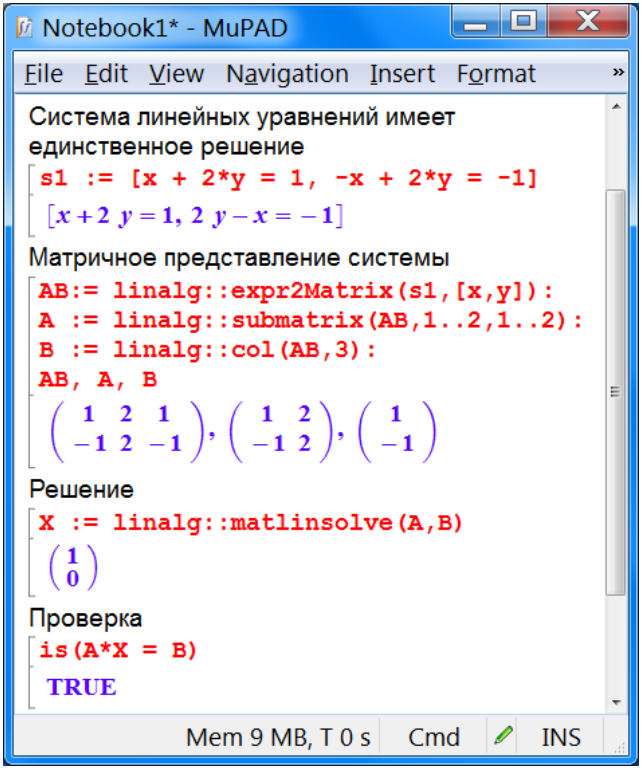


Рис. 7.5. Система линейных уравнений, имеющая единственное решение

Если система линейных уравнений записана явным образом (как множество или список уравнений через символьные переменные, выступающие в роли неизвестных; например, см. рис. 5.6), то ее можно представить в виде расширенной матрицы $(A|B)$ с помощью функции `linalg::expr2Matrix`, которую вызывают следующим образом:

```
<Матрица> := linalg::expr2Matrix(<система уравнений>,
                                 <неизвестные>, <опция>)
```

Параметр `<система уравнений>` можно задать либо в виде списка, либо в виде множества уравнений, а также с помощью идентификатора переменной, которой ранее этот список или множество было присвоено.

Каждое из уравнений должно являться линейным, т.е. содержать только линейную комбинацию неизвестных (символьных переменных) с числовыми или символьными коэффициентами. Параметр `<неизвестные>` указывает, относительно каких из использованных для записи уравнений символьных переменных следует решать в дальнейшем данную `<систему уравнений>`. Если этот параметр не указан, то по умолчанию `<неизвестными>` считаются все не заданные ранее символьные переменные, использованные для записи `<системы уравнений>`. Параметр `<неизвестные>` следует указывать в виде списка `[]` или множества `{}`, состоящего из идентификаторов соответствующих символьных переменных. Функция `linalg::expr2Matrix` из заданной `<системы уравнений>` формирует матрицу, количество строк которой равно количеству уравнений, а количество столбцов на единицу больше, чем количество `<неизвестных>`, – расширенную матрицу коэффициентов, в последний столбец которой помещаются свободные члены каждого из уравнений. Например, в результате выполнения команды

```
linalg::expr2Matrix([x + y = 1, 2*x - y = 3], [x, y])
```

получится матрица:

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & -1 & 3 \end{pmatrix}.$$

При вызове функции `linalg::expr2Matrix` можно использовать только одну `<опцию>` (`Include`), которая позволяет при составлении матрицы изменить знак на противоположный у всех свободных членов каждого из уравнений.

Если для дальнейшего использования в программе вместо расширенной матрицы нужны отдельно матрица коэффициентов и столбец свободных членов, их можно извлечь из полученной расширенной матрицы с помощью функций `linalg::col` и `linalg::submatrix` (см. разд. 7.2).

В MuPAD для решения системы линейных уравнений, записанной в матричном виде $A{\cdot}X = B$, можно использовать функции `linalg::matlinsolve` и `numeric::matlinsolve`, которые вызывают следующим образом:

```
<Результат> := linalg::matlinsolve(<матрица A>,
                                   <матрица B>, <опции>)
```

<Результат> := **numeric::matlinsolve**(<матрица A>,
<матрица B>, <опции>)

В качестве параметров <матрица A> и <матрица B> обеим функциям необходимо указать соответственно матрицу, составленную из коэффициентов при неизвестных системы линейных уравнений, и матрицу-столбец, составленную из свободных членов этих уравнений. Как правило, эти параметры задают с помощью идентификаторов переменных, которым ранее были присвоены соответствующие матрицы. Например, команда искать решение системы линейных уравнений (без опций) может иметь вид: `linalg::matlinsolve(A, B)` или `numeric::matlinsolve(A, B)`, если `A` – идентификатор заданной ранее матрицы коэффициентов, `B` – идентификатор матрицы-столбца свободных членов системы линейных уравнений.

Функция `linalg::matlinsolve` решает заданную с помощью <матрицы A> и <матрицы B> систему уравнений аналитически, а функция `numeric::matlinsolve` – численно. Если функция `numeric::matlinsolve` используется без <опции> `Symbolic`, перед началом вычисления она переводит элементы обеих исходных матриц в десятичный формат. Если хотя бы один элемент любой из этих матриц не удалось перевести в десятичный формат, то функция `numeric::matlinsolve` автоматически выбирает <опцию> `Symbolic` и ищет решение аналитически.

Если система линейных уравнений имеет единственное решение, то каждая из функций `linalg::matlinsolve` и `numeric::matlinsolve` (вызванная без использования <опций>) находит и возвращает <результат> в виде матрицы, размер которой совпадает с размером <матрицы B>. Если система имеет несколько (конечное количество) решений, то каждая из функций `linalg::matlinsolve` и `numeric::matlinsolve` возвращает список, каждый элемент которого является решением системы и записан в виде матрицы, размер которой совпадает с размером <матрицы B>. Если система не имеет решения, то функция `linalg::matlinsolve` возвращает пустой список `[]`, а функция `numeric::matlinsolve` – список в виде: `[FAIL, NIL]`. Если система имеет бесконечное количество решений, то каждая из этих функций (вызванная без использования <опций>) возвращает результат в виде списка, состоящего из двух элементов (рис. 7.6): 1) частное решение однородного матричного уравнения $A \cdot X = 0$; 2) список базисных векторов.

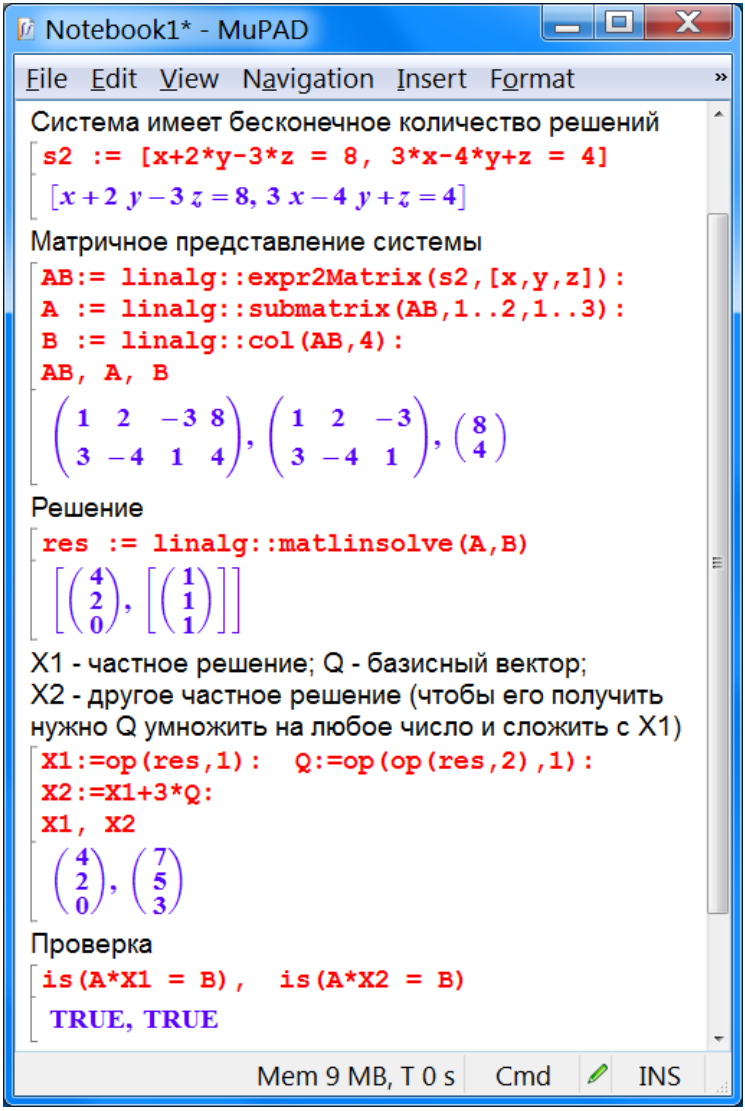


Рис. 7.6. Система линейных уравнений, имеющая бесконечное количество решений

Сумма частного решения однородного матричного уравнения и линейной комбинации базисных векторов с различными числовыми коэффициентами позволяет получать разные частные решения исходной системы уравнений $A \cdot X = B$.

Если исходная система уравнений имеет бесконечное количество решений, то можно каждой из функций `linalg::matlinsolve` и `numeric::matlinsolve` в качестве одной из <опций> задать список числовых коэффициентов для получения одного частного решения этой системы (матричного уравнения). Список коэффициентов может, например, иметь вид `[0, 0, 1, 1/2, –2]`, если решение системы уравнений содержит 5 базисных векторов. В этом случае к частному решению однородного матричного уравнения будут прибавлены базисные векторы, умноженные каждый на свой коэффициент из указанного списка – получится одно частное решение исходного матричного уравнения. В этом случае каждая из функций `linalg::matlinsolve` и `numeric:: matlinsolve` возвращает только это частное решение.

Кроме списка коэффициентов в качестве <опций> для функции `linalg::matlinsolve` можно использовать: `Normal` (приводит полученное решение к нормальному виду); `ShowAssumptions` (показывает все сделанные в процессе решения предположения как дополнительные элементы в списке <результат>); `Special` (показывает только одно частное решение); `Unique` (показывает решение, только если оно единственное).

Функцию `numeric::matlinsolve` (кроме списка коэффициентов) можно вызывать со следующими <опциями>: `Symbolic` (позволяет искать решение в символьном виде); `ShowAssumptions` (показывает все сделанные в процессе решения предположения как дополнительные элементы в списке <результат>); `Hard`, `HardwareFloats`, `Soft` или `SoftwareFloats` (позволяют выполнять операции с вещественными числами в десятичном формате с фиксированной или с плавающей запятой соответственно); `NoWarning` (подавляет предупреждающие сообщения); `Sparse` (использует разреженное представление матриц).

Функции `linalg::matlinsolve` и `numeric::matlinsolve` можно использовать не только для решения систем линейных уравнений в матричном виде, но и для решения любых матричных уравнений вида $A \cdot X = B$ (A – известная квадратная матрица; B – известная матрица,

количество строк которой равно количеству столбцов матрицы A; X – неизвестная матрица, количество строк которой также равно количеству столбцов матрицы A). Например:

$$A \cdot X = B, \text{ где } A = \begin{pmatrix} 1 & 2 \\ -2 & 3 \end{pmatrix}, \; B = \begin{pmatrix} 4 & 2 \\ 6 & 3 \end{pmatrix}, \; X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}.$$

Если такое матричное уравнение имеет единственное решение, то для его получения достаточно исходное уравнение слева домножить на матрицу, обратную матрице A:

$$A^{-1} \cdot A \cdot X = A^{-1} \cdot B \Rightarrow X = A^{-1} \cdot B$$

Если решение не является единственным, применяют другие методы.

Эти методы (в том числе и методы, использующие матрицу, обратную матрице A) и реализованы в функциях `linalg::matlinsolve` и `numeric::matlinsolve`. Для решения матричных уравнений функции `linalg::matlinsolve` и `numeric::matlinsolve` вызывают по тем же правилам и с теми же опциями, что и для решения системы линейных уравнений в матричном виде.

Кроме функций `linalg::matlinsolve` и `numeric::matlinsolve` MuPAD содержит и другие функции, позволяющие решать матричные уравнения другими методами (например, методом наименьших квадратов – `numeric::leastSquares` или методом Вейдемана – `linalg::wiedemann`). С особенностями использования этих методов можно самостоятельно познакомиться в справочной системе MuPAD.

Итак, если система линейных уравнений имеет единственное решение, то, чтобы решить такую систему в матричном виде, необходимо, чтобы программа выполняла следующую последовательность действий:

1) записать систему линейных уравнений в аналитическом виде;

2) представить систему линейных уравнений в виде двух матриц;

3) решить матричное уравнение;

4) проверить, является ли полученный вектор решением системы линейных уравнений.

Например, описанную выше последовательность действий можно реализовать с помощью следующих команд, результат выполнения которых приведен на рис. 7.5:

```
s  := [x + 2*y = 1, -x + 2*y = -1];
AB := linalg::expr2Matrix(s, [x, y]);
A  := linalg::submatrix(AB, 1..2, 1..2);
```

```
B  := linalg::col(AB, 3);
X  := linalg::matlinsolve(A, B);
is(A*X = B)
```

При необходимости для решения матричного уравнения вместо функции `linalg::matlinsolve` можно использовать функцию `numeric::matlinsolve`. В приведенном примере проверка решения выполнена в матричном виде: результат умножения `A*X` должен равняться `B`. Если это так, то функция `is` покажет `TRUE`. Проверку можно также выполнить, используя первоначальную систему. Для этого необходимо в эту систему вместо переменной `x` подставить `X[1]`, а вместо переменной `y` подставить `X[2]`, например с помощью функции `subs`.

Если система имеет несколько решений, проверку стоит выполнить для каждого. Если система имеет бесконечное количество решений, то из полученного частного решения однородного матричного уравнения и линейной комбинации базисных векторов следует сконструировать несколько частных решений исходной системы уравнений и выполнить проверку для каждого из них.

Пример решения системы линейных уравнений, имеющей бесконечное количество решений, приведен на рис. 7.6.

7.5.3. Собственные векторы и собственные значения матрицы

Всякий ненулевой вектор-столбец X называется ***собственным вектором*** матрицы A, если найдется такое число λ, что результатом умножения матрицы A на этот вектор-столбец X является этот же вектор-столбец, умноженный на число λ:

$$A \cdot X = \lambda \cdot X.$$

Тогда λ называют ***собственным значением*** матрицы A (число, умножение которого на собственный вектор матрицы эквивалентно умножению матрицы на этот собственный вектор). Учитывая правила умножения матриц, собственные векторы и собственные значения может иметь только квадратная матрица. Для нахождения собственных векторов и собственных значений связывающее их уравнение преобразуют следующим образом:

$$A \cdot X = \lambda \cdot X \Rightarrow \lambda \cdot X - A \cdot X = 0 \Rightarrow (\lambda \cdot E) \cdot X - A \cdot X = 0 \Rightarrow (\lambda \cdot E - A) \cdot X = 0.$$

Очевидно, что в таком виде это уравнение можно интерпретировать как матричную запись системы однородных линейных уравнений,

где X – столбец неизвестных, а $(A - \lambda \cdot E)$ – матрица коэффициентов при неизвестных, которая называется ***характеристической матрицей***. Известно, что система однородных линейных уравнений имеет нетривиальное (отличное от нуля) решение, только если определитель матрицы, состоящей из коэффициентов при неизвестных этой системы, равен нулю. Следовательно, чтобы матрица A имела нетривиальные собственные векторы, определитель характеристической матрицы должен быть равен нулю:

$$|\lambda \cdot E - A| = 0 \text{ или } \begin{vmatrix} \lambda - a_{11} & -a_{12} & \dots & -a_{1n} \\ -a_{21} & \lambda - a_{22} & \dots & -a_{2n} \\ \dots & \dots & \dots & \dots \\ -a_{n1} & -a_{n2} & \dots & \lambda - a_{nn} \end{vmatrix} = 0 \cdot$$

В результате вычисления определителя получается полином n-й степени относительно собственных значений λ. Этот ***полином*** также называется ***характеристическим***, так как собственные значения матрицы A – это корни характеристического полинома. Полином n-й степени имеет не более чем n корней, некоторые из которых могут оказаться кратными. Поэтому и матрица A не может иметь собственных значений больше, чем количество строк (столбцов) в ней. Собственные значения матрицы имеют ту же кратность, что и совпадающие с ними корни характеристического полинома. Таким образом, чтобы найти собственные значения матрицы, необходимо найти корни характеристического полинома. Каждому собственному значению соответствует собственный вектор. Чтобы его найти, необходимо подставить собственное значение в соответствующую однородную систему линейных уравнений и решить ее относительно вектора X.

MuPAD содержит несколько функций, позволяющих составлять характеристические матрицы, характеристические полиномы, а также искать собственные значения и собственные векторы для заданных квадратных матриц. Чтобы составить характеристическую матрицу, можно использовать функцию `linalg::charmat`, а чтобы составить характеристический полином – функцию `linalg::charpoly`. Эти функции вызывают следующим образом:

```
<Результат> := linalg::charmat(<матрица>,
                                    <переменная>)
<Результат> := linalg::charpoly(<матрица>,
                                    <переменная>)
```

В качестве параметра <матрица> необходимо указать квадратную матрицу или идентификатор, которым она была ранее обозначена. В качестве <переменной> необходимо указать идентификатор символьной переменной, которой будут обозначены искомые (неизвестные) собственные значения <матрицы>.

Например, чтобы для заданной ранее квадратной матрицы, обозначенной переменной A, составить характеристическую матрицу, можно написать команду linalg::charmat(A,x), а чтобы для этой же матрицы составить характеристический полином – команду linalg::charpoly(A,x). В полученных в результате выполнения этих команд матрице и полиноме (соответственно) с помощью символьной переменной x будут обозначены искомые собственные значения матрицы A.

Функция linalg::charmat возвращает <результат> в виде матрицы, размер которой совпадает с размером исходной <матрицы>, а на главной диагонали стоят алгебраические выражения, записанные с помощью указанной символьной <переменной>. Элементы этой матрицы являются не выражениями, а объектами специального типа – полиномами. Чтобы в дальнейшем в программе иметь возможность выполнять какие-то операции с полученной матрицей, может потребоваться преобразовать ее в матрицу с элементами произвольного типа. Это можно осуществить с помощью команды coerce(<результат>, Dom:: Matrix()) (см. разд. 7.1.2)

Функция linalg::charpoly возвращает <результат> в виде полинома, элементы которого скомпонованы по степеням указанной символьной <переменной>; степень полинома совпадает с количеством строк (столбцов) исходной матрицы. Полученный с помощью этой функции полином не является выражением, поэтому для дальнейшего его использования в программе может потребоваться преобразовать полученный полином в выражение с помощью функции expr, написав команду expr(<результат>). Корни характеристического полинома, преобразованного в выражение, можно найти с помощь функций solve или numeric:: polyroots (см. разд. 5.1, 5.2).

С помощью последовательности раскрывающихся списков General Math → Matrices → Characteristic Polynomial, расположенной на командной панели Command Bar, в текст программы можно вставить шаблон linalg::charpoly(#, #) для получения

характеристического полинома матрицы. Поля, отмеченные символом #, необходимо заполнить в соответствии с описанными выше правилами.

Функции linalg::eigenvalues и numeric::eigenvalues позволяют найти собственные значения матрицы. Для этого эти функции вызывают следующим образом:

<Результат> := **linalg::eigenvalues**(<матрица>, <опция>)
<Результат> := **numeric::eigenvalues**(<матрица>,
<опции>)

Для этих функций параметр <матрица> имеет тот же смысл, и записывается по тем же правилам, как и для функций linalg::charmat и linalg::charpoly.

Функция linalg::eigenvalues находит собственные значения исходной <матрицы>, используя аналитические методы. Если хотя бы один элемент исходной <матрицы> содержит число в десятичном формате, функция linalg::eigenvalues вызывает функцию numeric::eigenvalues, которая ищет собственные значения исходной <матрицы>, используя численные методы. Перед началом вычислений функция numeric::eigenvalues переводит все элементы исходной <матрицы> в десятичный формат.

Например, для нахождения собственных значений заданной ранее квадратной матрицы, обозначенной переменной A, можно написать одну из следующих команд (без опций): linalg::eigenvalues(A) или numeric::eigenvalues(A).

Функцию linalg::eigenvalues можно использовать только с одной <опцией> Multiple, которая позволяет получить информацию о кратности каждого собственного значения.

Функцию numeric::eigenvalues можно использовать с двумя <опциями>: NoWarning (подавляет вывод предупреждающих сообщений); опция, позволяющая управлять десятичным форматом представления вещественных чисел – с фиксированной (Hard или HardwareFloats) или с плавающей (Soft или SoftwareFloats) запятой. По умолчанию используются вычисления с плавающей запятой.

Если функция linalg::eigenvalues используется без <опции> Multiple, она возвращает найденные собственные значения исходной <матрицы> в виде множества { } – в фигурных скобках через запятую. С <опцией> Multiple эта функция возвращает список [], каждый

элемент которого содержит два заключенных в квадратные скобки числа: одно собственное значение и его кратность.

Функция numeric::eigenvalues всегда возвращает собственные значения в виде списка. Кратные собственные значения повторяются в этом списке соответствующее количество раз.

Шаблоны linalg::eigenvalues(#) и numeric::eigenvalues(#) для вызова функций, осуществляющих поиск собственных значений матрицы, можно вставить в текст программы с помощью последовательности раскрывающихся списков General Math → Matrices → Eigenvalues → Exact (или … → Numeric соответственно), расположенной на командной панели Command Bar. Вместо поля, отмеченного символом #, необходимо указать матрицу в соответствии с описанными выше правилами.

Функции linalg::eigenvalues и numeric::eigenvalues поиск собственных векторов исходной <матрицы> НЕ ВЫПОЛНЯЮТ.

Для нахождения собственных векторов матрицы можно использовать либо функцию linalg::eigenvectors, либо функцию numeric::eigenvectors. Эти функции вызывают следующим образом:

<Результат> := **linalg::eigenvectors**(<матрица>)
<Результат> := **numeric::eigenvectors**(<матрица>, <опции>)

Для этих функций параметр <матрица> имеет тот же смысл и записывается по тем же правилам, как и для функций linalg::charmat и linalg::charpoly.

Функция linalg::eigenvectors находит собственные векторы исходной <матрицы>, используя аналитические методы (метод Гаусса–Жордана). Если хотя бы один элемент исходной <матрицы> содержит число в десятичном формате, функция linalg::eigenvectors вызывает функцию numeric::eigenvectors, которая ищет собственные векторы исходной <матрицы>, используя численные методы. Перед началом вычислений функция numeric::eigenvectors переводит все элементы исходной <матрицы> в десятичный формат. Так как для нахождения собственных векторов необходимо знать собственные значения матрицы, каждая из этих функций сначала находит собственные значения, а потом приступает к поиску собственных векторов.

Например, для нахождения собственных векторов заданной ранее квадратной матрицы, обозначенной переменной A, можно написать одну из

следующих команд (без опций): `linalg::eigenvectors(A)` или `numeric::eigenvectors(A)`.

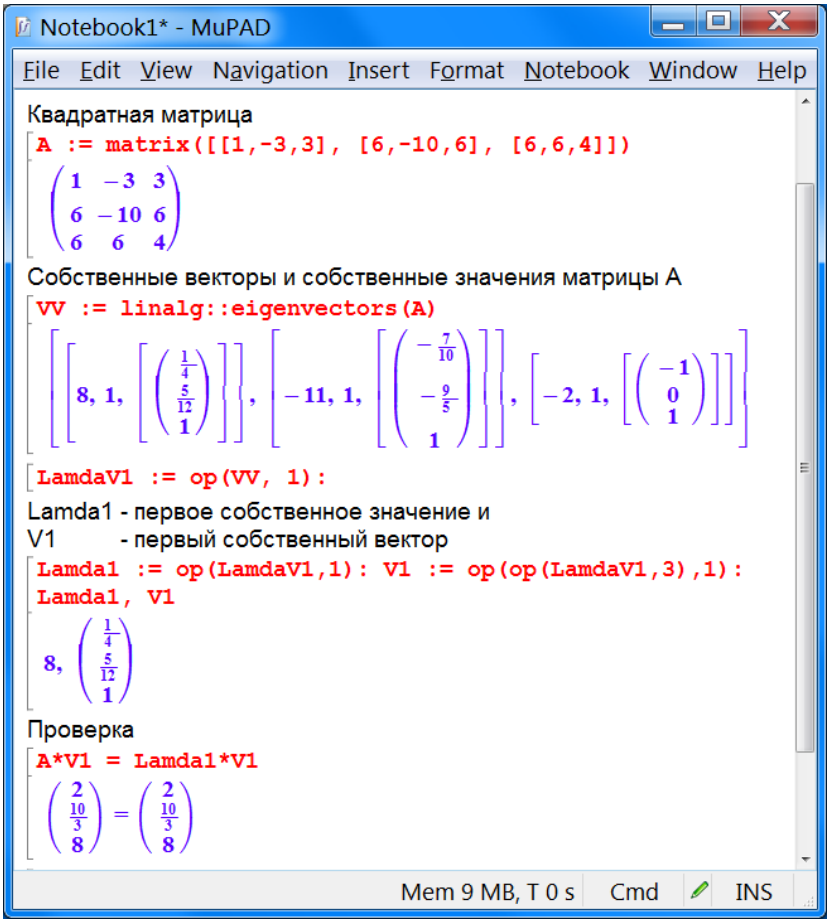


Рис. 7.7. Собственные векторы и собственные значения матрицы

Функция `linalg::eigenvectors` не имеет <опций>.

Функцию `numeric::eigenvectors` можно использовать со следующими <опциями>: `NoResidues` (позволяет не выполнять оценку погрешности вычисления собственных значений; в зависимости от версии

MuPAD эта опция также может называться `NoErrors`); `NoWarning` (подавляет вывод предупреждающих сообщений); опция, позволяющая управлять десятичным форматом представления вещественных чисел – с фиксированной (`Hard` или `HardwareFloats`) или с плавающей (`Soft` или `SoftwareFloats`) запятой. По умолчанию используются вычисления с плавающей запятой.

Функция `linalg::eigenvectors` возвращает список `[]`, каждый элемент которого, в свою очередь, тоже является списком, состоящим из трех элементов: одного собственного значения, его кратности и соответствующего этому собственному значению собственного вектора (рис. 7.7).

Функция `numeric::eigenvectors` возвращает список `[]`, состоящий из трех элементов: 1) список собственных значений; 2) матрица, составленная из собственных векторов (объединение векторов-столбцов по горизонтали); 3) список погрешностей вычисления собственных значений или `NIL`, если оценка погрешности не выполнялась. Поэтому при вызове этой функции `<результат>` сразу можно записывать в виде списка, состоящего из трех переменных. Тогда первой переменной будет присвоен список собственных значений, второй – матрица собственных векторов, а третьей – список погрешностей. Например: `[Lamda, V, pogr]:= numeric::eigenvectors(A).`

Шаблоны `linalg::eigenvectors(#)` и `numeric::eigenvectors(#)` для вызова функций, осуществляющих поиск собственных векторов матрицы, можно вставить в текст программы с помощью последовательности раскрывающихся списков `General` `Math` → `Matrices` → `Eigenvectors` → `Exact` (или … → `Numeric` соответственно), расположенной на командной панели `Command Bar`. В поле, отмеченном символом #, необходимо указать матрицу в соответствии с описанными выше правилами.

Таким образом, для нахождения собственных векторов матрицы можно либо использовать функцию `linalg::eigenvectors` (или `numeric::eigenvectors`), либо выполнить следующую последовательность вычислений:

1) задать (создать) квадратную матрицу;

2) составить характеристическую матрицу и характеристический полином (и выполнить вспомогательные операции: преобразовать характе-

ристическую матрицу в матрицу с элементами произвольного типа, а характеристический полином – в выражение);

3) найти корни характеристического полинома – собственные значения;

4) составить систему линейных однородных уравнений в матричном виде (для этого необходимо в характеристическую матрицу подставить одно из собственных значений, дополнительно создать матрицу-столбец свободных членов, состоящую из нулей) и, решив ее, получить собственный вектор;

5) если матрица имеет несколько собственных значений, для каждого из них следует найти собственный вектор, решив соответствующую систему линейных однородных уравнений.

Для преобразования характеристической матрицы в матрицу с элементами произвольного вида можно использовать функцию `coerce` (см. разд. 7.1.2), а для преобразования характеристического полинома в выражение – функцию `expr`.

В этом случае программа для нахождения собственных значений и собственных векторов матрицы может иметь вид:

```
A := matrix([[1, -3, 3], [6, -10, 6], [6, 6, 4]]);
B := matrix(3, 1, 0);
HH := linalg::charmat(A, x);
H := coerce(HH, Dom::Matrix());
pp := linalg::charpoly(A, x); p:= expr(pp);
Lamda := solve(p = 0, x);
Lamda1 := op(Lamda, 1);
H1 := subs(H, x = Lamda1);
V1 := linalg::matlinsolve(H1, B);
Lamda2 := op(Lamda, 2);
H2 := subs(H, x = Lamda2);
V2 := linalg::matlinsolve(H2, B); …
```

Тот же результат можно получить, если использовать, например, функцию `linalg::eigenvectors`. Тогда та же программа может иметь вид:

```
A := matrix([[1, -3, 3], [6, -10, 6], [6, 6, 4]]);
VV := linalg::eigenvectors(A);
LamdaV1 := op(VV, 1); LamdaV2 := op(VV, 2); …
Lamda1 := op(LamdaV1, 1); V1 := op(op(LamdaV1, 3), 1);
Lamda2 := op(LamdaV2, 1); V2 := op(op(LamdaV2, 3), 1); …
```

Чтобы убедиться, что найденные значения действительно являются собственными векторами и собственными значениями для исходной матрицы, достаточно умножить исходную матрицу на собственный вектор и сравнить полученный результат с умножением этого собственного вектора на соответствующее ему собственное значение. Например:

```
A*V1 = Lamda1*V1;  A*V2 = Lamda2*V2; …
```

Если справа и слева от равенства получатся одинаковые векторы (столбцы), то можно считать, что собственные векторы и собственные значения для данной матрицы были найдены правильно.

7.5.4. Векторы и векторные поля

В геометрии под вектором понимают направленный отрезок, а в линейной алгебре – матрицу-строку или матрицу-столбец произвольной длины (размерности) n. Элементы матрицы-строки (матрицы-столбца) интерпретируют как координаты вектора в некоторой системе координат n-мерного пространства. Под длиной (размерностью) вектора понимают количество элементов вектора: количество столбцов матрицы-строки или количество строк матрицы-столбца. Для векторов справедливы все операции, разрешенные для неквадратных матриц: сложение, вычитание, умножение, транспонирование. По правилам умножения матриц можно вычислить произведение вектора-строки на вектор-столбец или произведение вектора-столбца на вектор-строку (если они имеют одинаковую длину). Умножение двух векторов-строк (двух векторов-столбцов) друг на друга не допускается. Кроме операций, свойственных всем неквадратным матрицам, для векторов вводятся дополнительные операции: скалярное, векторное и смешанное произведения векторов.

Скалярным произведением $(\vec{a},\vec{b})$ двух векторов-столбцов $\vec{a}$ и $\vec{b}$ называется результат умножения транспонированного комплексно сопряженного вектора $\vec{a}^{*\mathrm{T}}$ на вектор $\vec{b}$:

$$(\vec{a},\vec{b}) = \vec{a}^{*\mathrm{T}} \cdot \vec{b} = \begin{pmatrix} a_1^* & a_2^* & \dots & a_n^* \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} = \sum_{i=1}^{n} a_i^* b_i \ .$$

Скалярное произведение $(\vec{a},\vec{b})$ двух векторов-строк $\vec{a}$ и $\vec{b}$:

$$(\vec{a},\vec{b}) = \vec{a}\cdot\vec{b}^{*\mathrm{T}} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix}\cdot\begin{pmatrix} b_1^* \\ b_2^* \\ \dots \\ b_n^* \end{pmatrix} = \sum_{i=1}^{n} a_i b_i^* .$$

Если все элементы векторов вещественные (некомплексные), то комплексное сопряжение можно опустить. Скалярное произведение определено только для векторов одинаковой длины (размерности).

В MuPAD скалярное произведение векторов можно вычислить с помощью функции `linalg::scalarProduct`, которую вызывают следующим образом:

`<Результат> :=` **`linalg::scalarProduct`**`(<вектор a>, <вектор b>, <опция>)`

В качестве параметров `<вектор a>` и `<вектор b>` необходимо указать идентификаторы двух матриц-строк (или двух матриц-столбцов) одинаковой размерности (рис. 7.8). Если элементы `<вектора a>` и `<вектора b>` не все являются числами, а записаны через символьные переменные, то по умолчанию эти переменные считаются комплексными. Поэтому при вычислении скалярного произведения (без `<опции> Real`) выполняется комплексное сопряжение одного из векторов. Если при вызове функции `linalg::scalarProduct` использовать `<опцию> Real`, то вычисления выполняются без комплексного сопряжения. В качестве `<результата>` функция `linalg::scalarProduct` возвращает числовое значение или математическое выражение, результатом вычисления которого является числовое значение.

Например, чтобы вычислить скалярное произведение двух ранее созданных векторов (матриц-строк), обозначенных переменными `V1` и `V2` (см. рис. 7.8), можно написать команду `linalg::scalarProduct (V1, V2)`.

В трехмерном пространстве (длина вектора равна 3) кроме скалярного определены также векторное и смешанное произведения векторов. ***Векторным произведением*** $[\vec{a},\vec{b}]$ двух векторов-строк $\vec{a}$ и $\vec{b}$ называется вектор-строка, которая вычисляется с помощью определителя матрицы, составленной из элементов векторов $\vec{a}$ и $\vec{b}$ и единичных ортов $(\hat{e}_1\ \hat{e}_2\ \hat{e}_3)$ системы координат, в которой заданы эти векторы:

$$[\vec{a},\vec{b}] = \begin{vmatrix} \hat{e}_1 & \hat{e}_2 & \hat{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}.$$

В матричной форме вектор, являющийся результатом вычисления векторного произведения, имеет следующие элементы:

$$[\vec{a},\vec{b}] = \begin{pmatrix} a_2b_3 - a_3b_2 & a_3b_1 - a_1b_3 & a_1b_2 - a_2b_1 \end{pmatrix}.$$

Аналогично, векторное произведение $[\vec{a},\vec{b}]$ двух векторов-столбцов $\vec{a}$ и $\vec{b}$ – это вектор-столбец следующего вида:

$$[\vec{a},\vec{b}] = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix}.$$

В MuPAD векторное произведение векторов можно вычислить с помощью функции `linalg::crossProduct`, которую вызывают следующим образом:

`<Результат> := linalg::crossProduct(<вектор a>, <вектор b>)`

В качестве параметров `<вектор a>` и `<вектор b>` необходимо указать идентификаторы двух матриц-строк (или двух матриц-столбцов), состоящих из трех элементов каждый. В качестве `<результата>` функция `linalg::crossProduct` возвращает матрицу-стоку или матрицу-столбец в зависимости от того, какими матрицами являлись `<вектор a>` и `<вектор b>`.

Например, чтобы вычислить векторное произведение двух ранее созданных векторов (матриц-строк), обозначенных переменными `V1` и `V2` (см. рис. 7.8), можно написать команду `linalg::crossProduct (V1, V2)`.

Под ***смешанным произведением*** $(\vec{a},\vec{b},\vec{c})$ трех векторов $\vec{a}$, $\vec{b}$ и $\vec{c}$ (размерностью 3 каждый) понимают результат вычисления определителя матрицы, составленной из этих векторов:

$$(\vec{a},\vec{b},\vec{c}) = (\vec{a},[\vec{b},\vec{c}]) = ([\vec{a},\vec{b}],\vec{c}) = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}.$$

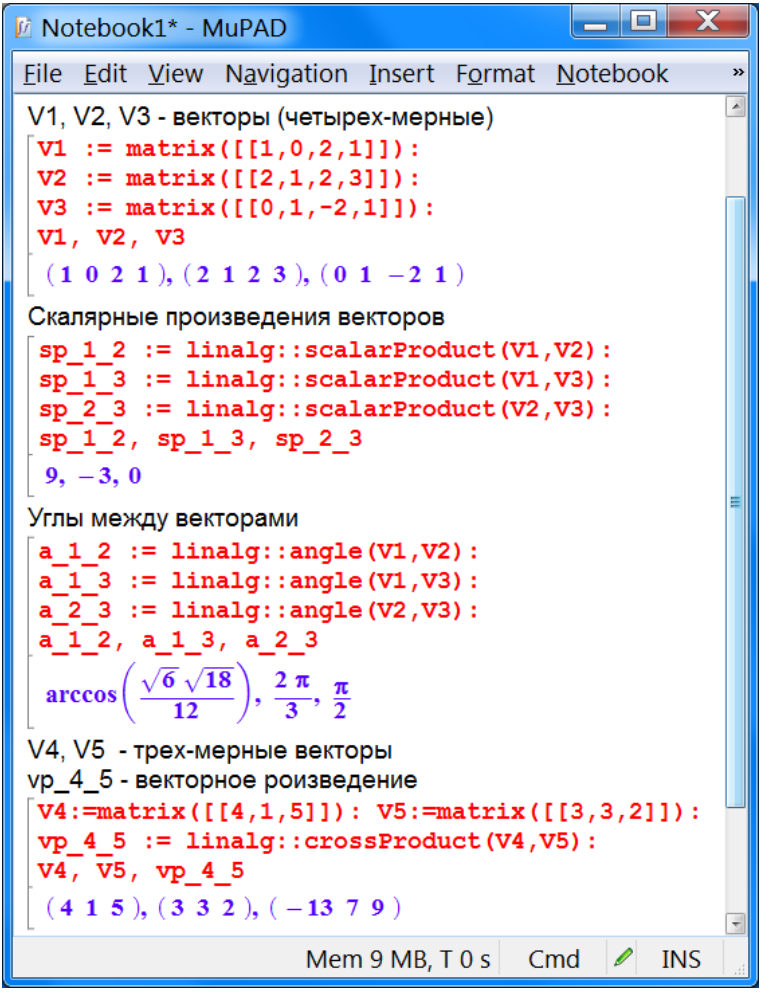


Рис. 7.8. Скалярное и векторное произведения векторов

В MuPAD отсутствует стандартная функция для вычисления смешанного произведения. Поэтому для вычисления смешанного произведения трех векторов следует либо составить из элементов этих векторов матрицу (например, объединить матрицы-строки по вертикали) и вычислить ее

определитель, либо с помощью функций `linalg::scalarProduct` и `linalg::crossProduct` вычислить скалярное произведение одного из этих векторов на векторное произведение двух других.

Учитывая геометрический смысл вектора, вводят понятия «модуль вектора» и «угол между двумя векторами».

Считается, что скалярное произведение вектора самого на себя равно квадрату его модуля. То есть, чтобы определить ***модуль*** $|a|$ ***вектора*** $\vec{a}$ произвольной длины (размерности) n, необходимо вычислить корень квадратный из скалярного произведения вектора самого на себя:

$$|a| = \sqrt{(\vec{a},\vec{a})} = \sqrt{\sum_{i=1}^{n} a_i^* a_i} \ .$$

Если элементы вектора некомплексные, то $|a| = \sqrt{\sum_{i=1}^{n} a_i^2}$.

Так как в геометрии скалярное произведение двух векторов $\vec{a}$ и $\vec{b}$ равно произведению модулей этих векторов и косинуса угла между векторами: $(\vec{a},\vec{b}) = |a|\cdot|b|\cdot\cos\varphi$, то ***угол*** φ ***между векторами*** $\vec{a}$ и $\vec{b}$ можно вычислить следующим образом:

$$\varphi = \arccos\left(\frac{(\vec{a},\vec{b})}{|\vec{a}|\cdot|\vec{b}|}\right) \text{ или } \varphi = \arccos\left(\frac{\sum_{i=1}^{n} a_i^* b_i}{\sum_{i=1}^{n} a_i^* a_i \sum_{i=1}^{n} b_i^* b_i}\right).$$

В MuPAD угол между векторами можно вычислить с помощью функции `linalg::angle`, которую вызывают следующим образом:

`<Результат> := linalg::angle(<вектор a>, <вектор b>)`

В качестве параметров `<вектор a>` и `<вектор b>` необходимо указать идентификаторы двух матриц-строк (или двух матриц-столбцов) одинаковой размерности (см. рис. 7.8). В качестве `<результата>` функция `linalg::angle` возвращает числовое значение (в радианах) или математическое выражение, результатом вычисления которого является числовое значение.

Например, чтобы вычислить угол между двумя ранее созданными векторами (матрицами-строками), обозначенными переменными `V1` и `V2` (см. рис. 7.8), можно написать команду `linalg::angle(V1, V2)`.

Если скалярное произведение двух векторов обращается в ноль, то такие векторы называют ортогональными. Чтобы проверить, являются ли два вектора ортогональными, достаточно вычислить их скалярное произведение. ***Система векторов*** $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\}$, модуль каждого из которых отличен от нуля, является ***ортогональной***, если эти векторы попарно ортогональны: $(\vec{x}_i, \vec{x}_j) = 0$ при $i \neq j$. Для ортогонализации системы векторов (построения системы ортогональных векторов) существуют различные методы, например метод Грамма–Шмидта. В соответствии с этим методом векторы, образующие ортогональную систему $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\}$, можно получить из векторов, образующих неортогональную систему $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_m\}$, следующим образом:

$$\vec{x}_k = \vec{f}_k - \sum_{i=1}^{k-1} \lambda_{ki} \vec{x}_i \text{, где } \lambda_{ki} = \frac{(\vec{f}_k, \vec{x}_i)}{(\vec{x}_i, \vec{x}_i)}.$$

Очевидно, что $\vec{x}_1 = \vec{f}_1$, $\lambda_{21} = \dfrac{(\vec{f}_2, \vec{x}_1)}{(\vec{x}_1, \vec{x}_1)} = \dfrac{(\vec{f}_2, \vec{f}_1)}{(\vec{f}_1, \vec{f}_1)}$ и т.д. Если система $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_m\}$ ортогональна, то все коэффициенты λ_{ki} обращаются в ноль. Для ортогонализации системы векторов методом Грамма–Шмидта в MuPAD можно использовать функцию `linalg::orthog`, которую вызывают следующим образом:

```
<Результат> := linalg::orthog(<набор векторов>,
                                        <опция>)
```

В качестве параметра `<набор векторов>` необходимо указать список `[]` или множество `{}` матриц-строк (или матриц-столбцов) одинаковой длины (размерности), которые необходимо ортогонализовать. Если элементы этих векторов не все являются числами, а записаны через символьные переменные, то по умолчанию они считаются комплексными. Поэтому если функция `linalg::orthog` вызвана без `<опции>` `Real`, при вычислении скалярных произведений выполняется комплексное сопряжение одного из двух перемножаемых векторов. Если при вызове функции `linalg::orthog` использовать `<опцию>` `Real`, то вычисление выполняются без комплексного сопряжения. В качестве `<результата>` функция `linalg::orthog` возвращает список `[]` или множество

{} ортогональных векторов той же размерности, что и векторы в исходном <наборе векторов>.

Например, чтобы ортоганализовать систему, состоящую из ранее созданных векторов (матриц-строк), обозначенных переменными V1, V2, V3, V4, V5, можно написать команду linalg::orthog([V1, V2, V3, V4, V5]).

Чтобы убедиться, что полученная система векторов ортогональна, достаточно вычислить все их скалярные произведения друг на друга попарно.

Два вектора $\vec{a}$ и $\vec{b}$ называются линейно зависимыми, если можно найти такое число λ (отличное от нуля), что $\vec{b} = \lambda\vec{a}$. Векторное произведение линейно зависимых векторов равно нулю.

Множество линейно независимых векторов некоторого пространства называется ***базисом***, если любой другой вектор этого пространства может быть единственным образом представлен в виде линейной комбинации векторов этого множества. Если в некотором пространстве существует набор векторов, то среди них можно выбрать базисные – линейно независимые. В MuPAD для этого можно воспользоваться функцией linalg::basis, которую вызывают следующим образом:

<Результат> := **linalg::basis**(<набор векторов>)

В качестве параметра <набор векторов> необходимо указать список [] или множество {} матриц-строк (или матриц-столбцов) одинаковой длины (размерности), из которых необходимо построить базис. В качестве <результата> функция linalg::basis возвращает список [] или множество {} базисных векторов той же размерности, что и векторы в исходном <наборе векторов>. Линейно зависимые векторы из исходного набора векторов исключаются. В этом можно убедиться, если вычислить векторные произведения каждого базисного вектора с векторами из исходного <набора векторов>.

Например, чтобы из ранее созданных векторов (матриц-строк) V1, V2, V3, V4, V5 составить базис, можно написать команду linalg::basis([V1, V2, V3, V4, V5]).

Если в каждой точке пространства (3-мерного или n-мерного) задано значение некоторой скалярной функции $F(x_1, x_2, \ldots, x_n)$ (задано числовое значение или символьное выражение, в результате вычисления которого может получиться числовое значение), то говорят, что в этом пространстве

задано ***скалярное поле***. Если в каждой точке пространства (3-мерного или n-мерного) задано значение некоторой векторной функции $\vec{F}(x_1,\ x_2,\ \ldots,\ x_n)$ (задан вектор), то говорят, что в этом пространстве задано ***векторное поле***.

Направление скорейшего возрастания n-мерного скалярного поля $F(x_1, x_2, \ldots, x_n)$ указывает ***градиент*** $\overrightarrow{\mathrm{grad}}\, F$ – n-мерный вектор, каждый элемент которого равен частной производной от скалярной функции $F(x_1, x_2, \ldots, x_n)$ по соответствующей координате:

$$\overrightarrow{\mathrm{grad}}\ F = \begin{pmatrix} \dfrac{\partial F}{\partial x_1} & \dfrac{\partial F}{\partial x_2} & \ldots & \dfrac{\partial F}{\partial x_n} \end{pmatrix} \text{ или } \left(\mathrm{grad}\, F\right)_i = \frac{\partial F}{\partial x_i}.$$

Поток векторного поля $\vec{F}\,(x_1, x_2, \ldots, x_n)$ через элементарную поверхность в n-мерном пространстве определяет ***дивергенция*** $\mathrm{div}\,\vec{F}$ – числовое значение или символьное выражение, которое равно сумме частных производных от каждого элемента вектора $\vec{F}$ по соответствующей координате:

$$\mathrm{div}\ \vec{F} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_2}{\partial x_2} + \ldots + \frac{\partial F_n}{\partial x_n} = \sum_{i=1}^{n} \frac{\partial F_i}{\partial x_i}.$$

Является ли векторное поле $\vec{F}(x, y, z)$, заданное в трехмерном пространстве, вихревым (непотенциальным) или безвихревым (потенциальным), можно определить, вычислив ***ротор*** $\mathrm{rot}\,\vec{F}$ – вектор, являющийся результатом векторного произведения вектора $\vec{\nabla} = \begin{pmatrix} \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \end{pmatrix}$ и вектора $\vec{F} = \begin{pmatrix} F_x & F_y & F_z \end{pmatrix}$:

$$\mathrm{rot}\,\vec{F} = [\vec{\nabla}, \vec{F}] = \begin{pmatrix} \hat{e}_x & \hat{e}_y & \hat{e}_z \\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \\ F_x & F_y & F_z \end{pmatrix}.$$

Если ротор $\mathrm{rot}\,\vec{F}$ равен нулю в любой точке пространства, векторное поле $\vec{F}(x, y, z)$ называется безвихревым и является потенциальным.

Поток градиента скалярного поля $F(x_1, x_2, \ldots, x_n)$ через элементарную поверхность в n-мерном пространстве называется ***Лапласианом*** ΔF и эквивалентен $\operatorname{div}\left(\overrightarrow{\operatorname{grad} F}\right)$. Лапласиан скалярной функции $F(x_1, x_2, \ldots, x_n)$ – это числовое значение или символьное выражение, которое вычисляется следующим образом:

$$\Delta F = \operatorname{div}\left(\overrightarrow{\operatorname{grad} F}\right) = \frac{\partial^2 F}{\partial x_1^2} + \frac{\partial^2 F}{\partial x_2^2} + \ldots + \frac{\partial^2 F}{\partial x_n^2} = \sum_{i=1}^{n} \frac{\partial^2 F}{\partial x_i^2}.$$

В MuPAD для вычисления градиента можно использовать функцию `gradient`, для вычисления дивергенции – функцию `divergence`, для вычисления ротора – функцию `curl`, а для вычисления Лапласиана – функцию `laplacian`. Эти функции вызываются следующим образом:

```
<Результат> := gradient(<скалярная функция>,
                                <координаты>, <опция>)
<Результат> := divergence(<векторная функция>,
                                <координаты>, <опция>)
<Результат> := curl(<векторная функция>,
                                <координаты>, <опция>)
<Результат> := laplacian(<скалярная функция>,
                                <координаты>, <опция>)
```

При использовании каждой из этих функций необходимо указать либо скалярную, либо векторную математическую функцию, которая определяет поле (скалярное или векторное) в каждой точке пространства. При вызове функций `gradient` и `laplacian` в качестве `<скалярной функции>` можно указать математическое выражение (рис. 7.9), в записи которого координаты пространства, в котором задано поле, обозначены символьными переменными. Например: `x^2 + y`. В качестве `<скалярной функции>` можно также использовать вызов объявленной ранее функции нескольких переменных с символьными аргументами. Например, `f(x, y)`, если ранее было объявлено, что `f := (x, y) -> x^2 + y`.

При вызове функций `divergence` и `curl` в качестве `<векторной функции>` необходимо указать матрицу-строку или матрицу-столбец, элементы которой являются выражениями, в записи которых координаты пространства, в котором задано поле, также обозначены символьными переменными. На практике `<векторную функцию>` удобнее задавать с помощью идентификатора присвоенного ранее вектора. Например, с помощью

идентификатора переменной `F`, если ей ранее был присвоен вектор `F := matrix([x^2, 2*y, z])`. Кроме того, можно объявить функцию, которая возвращает матрицу-строку (матрицу-столбец), и использовать ее вызов с символьными аргументами в качестве `<векторной функции>`. Например, если сначала объявить функцию `F:= (x,y,z) -> matrix([x^2, 2*y, z])`, то далее в качестве параметра `<векторная функция>` можно использовать `F(x,y,z)`.

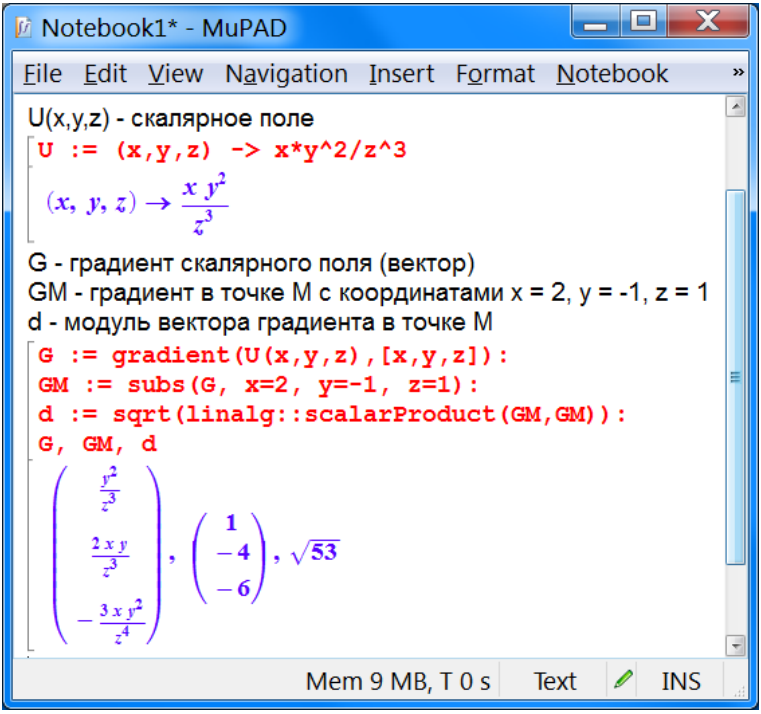


Рис. 7.9. Градиент скалярного поля

При вызове каждой из функций `gradient`, `laplacian`, `divergence` или `curl` в качестве параметра `<координаты>` необходимо указать список `[]` всех символьных переменных, которые использовались в `<скалярной функции>` или в `<векторной функции>` для обозначения координат пространства. Например: `[x, y]` или `[x, y, z]`.

Пример объявления функции `U(x,y,z)`, описывающей скалярное поле, и вычисления его градиента с помощью функции `gradient` (без опций) приведен на рис. 7.9.

`<Скалярная функция>` или `<векторная функция>` может быть представлена как в декартовых, так в полярных, сферических, цилиндрических и других координатах. По умолчанию считается, что функция поля записана в декартовых координатах. Если это не так, необходимо с помощью `<опции>` указать либо стандартный идентификатор соответствующей системы координат (`Cartesian`, `Cylindrical`, `Spherical`, `Spherical[LeftHanded]`, `EllipticCylindrical`, `ParabolicCylindrical`, `RotationParabolic`, `Torus`), либо вектор преобразования между используемыми координатами и декартовыми. Например, для цилиндрических координат следует использовать вектор `[1, r, 1]`. Для вычисления градиента скалярного поля, заданного функцией $U(r, \theta, \varphi) = r \sin\theta \cos\varphi$ в сферической системе координат, можно, например, написать команду `gradient(U(r,t,fi), [r,t,fi], Spherical)`, если предварительно объявлена функция `U:= (r,t,fi) -> r*sin(t)*cos(fi)`.

В качестве `<результата>` функции `divergence` и `laplacian` возвращают числовое значение или символьное выражение, результатом вычисления которого в каждой точке поля может являться числовое значение. В качестве `<результата>` функции `gradient` и `curl` возвращают вектор, длина (размерность) которого соответствует длине списка `<координаты>`, указанного при вызове функции, а элементы вектора – числовые значения или символьные выражения, результатами вычисления которых в каждой точке поля могут являться числовые значения.

Кроме рассмотренных выше дифференциальных характеристик векторные поля имеют также интегральные характеристики, такие, например, как ***потенциал*** – скалярная функция $U(x_1, x_2, \ldots, x_n)$, градиент от которой в каждой точке пространства равен известной векторной функции поля $\vec{F}(x_1, x_2, \ldots, x_n)$:

$$\overrightarrow{\text{grad}}\ U(x_1, x_2, \ldots, x_n) = \vec{F}(x_1, x_2, \ldots, x_n).$$

Векторный потенциал – векторная функция $\vec{U}(x_1, x_2, \ldots, x_n)$, ротор от которой в каждой точке пространства равен известной векторной функции поля $\vec{F}(x_1, x_2, \ldots, x_n)$:

$$\mathrm{rot}\ \vec{U}\ (x_1, x_2, \ldots, x_n) = \vec{F}\ (x_1, x_2, \ldots, x_n).$$

Для вычисления скалярного и векторного потенциалов в MuPAD можно использовать функции `potential` и `vectorPotential` соответственно. Вызов этих функций выполняется по правилам, аналогичным правилам вызова рассмотренных ранее функций. При необходимости с примерами использования функций `potential` и `vectorPotential` можно ознакомиться в справочной системе MuPAD.

Глава 8. РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ И ИХ СИСТЕМ

В этой главе описаны стандартные функции MuPAD, предназначенные для решения обыкновенных дифференциальных уравнений и их систем.

Обратите внимание! Значения, возвращаемые каждой из описанных далее функций, рекомендуется присваивать отдельным переменным. Это облегчит их дальнейшее использование в программе. Если полученные результаты далее в программе использовать не планируется, то для всех описанных ниже функций допускается вызов без присвоения.

Дифференциальным уравнением называют краткую запись задачи об отыскании аналитического вида функции с помощью известной зависимости между этой функцией, ее независимыми переменными (аргументами) и одной или несколькими производными искомой функции. Уравнения, не содержащие хотя бы одну производную искомой функции, не являются дифференциальными. Решением дифференциального уравнения называется функция, которая при подстановке в уравнение обращает его в тождество (удовлетворяет известной зависимости). Если искомая функция зависит от одного аргумента (переменной), то соответствующее дифференциальное уравнение называется ***обыкновенным***. Если искомая функция зависит от нескольких аргументов (переменных), то уравнение называется дифференциальным уравнением в частных производных. MuPAD позволяет решать только обыкновенные дифференциальные уравнения (для решения дифференциальных уравнений в частных производных в MatLab можно воспользоваться другим приложением – PDE).

Порядком дифференциального уравнения называется наивысший порядок производной, входящей в уравнение. Каждое обыкновенное дифференциальное уравнение n-го порядка можно записать несколькими способами:

$$F(y^{(n)}, \dots, y'', y', y, x) = 0, \text{ или } F(y^{(n)}, \dots, y'', y', y, x) = \varphi(x),$$
$$\text{или } y^{(n)} = F(y^{(n-1)}, \dots, y'', y', y, x),$$

где $y \equiv y(x)$ – искомая функция, зависящая от одного аргумента x.

В символьных вычислениях на компьютере чаще всего используется первая форма записи. Поэтому перед вызовом некоторых функций MuPAD исходное обыкновенное дифференциальное уравнение необходимо

привести к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Если уравнение приведено к такому виду, то вместо уравнения часто записывают только его левую часть $F(y^{(n)}, \ldots, y'', y', y, x)$.

Большинство обыкновенных дифференциальных уравнений n-го порядка можно свести к системе обыкновенных дифференциальных уравнений первого порядка с помощью следующей замены переменных:

$$y = u_0,\ y' = u'_0 = u_1,\ y'' = (y')' = u'_1 = u_2,\ \ldots,\ y^{(n)} = (y^{(n-1)})' = u'_{n-1}.$$

Тогда $y^{(n)} = F(y^{(n-1)}, \ldots, y'', y', y, x) \Rightarrow \begin{cases} u'_0 = u_1; \\ u'_1 = u_2; \\ \cdots\cdots\cdots \\ u'_{n-2} = u_{n-1}; \\ u'_{n-1} = F(u_{n-1}, \ldots, u_2, u_1, u_0, x). \end{cases}$

Поэтому большинство методов как аналитического, так и численного решения обыкновенных дифференциальных уравнений посвящено решению уравнений (или систем уравнений) первого порядка.

Для записи производных от не объявленной ранее искомой функции в дифференциальном уравнении или в его левой части в MuPAD в большинстве случаев можно использовать как функцию `diff`, так и функцию `D` или `'` (см. разд. 6.3). Например, дифференциальное уравнение $y''(t) + 4y(t) = \sin 2t$ можно записать: `y''(t) + 4*y(t) = sin(2*t)` или `diff(y(t), t $ 2) + 4*y(t) = sin(2*t)`; а левую часть уравнения $x^2 - y^2 + 2xy\dfrac{dy}{dx} = 0$ можно записать, например:

`x^2 - y(x)^2 + 2*x*y(x)*diff(y(x), x)`

или `x^2 - y(x)^2 + 2*x*y(x)*y'(x)`.

Но некоторые из описанных далее функций, предназначенных для анализа свойств и решения дифференциальных уравнений, накладывают ограничения на способ записи производной (либо только через `diff`, либо только через `'`) в соответствующем дифференциальном уравнении. При необходимости уравнение, производные в котором записаны с помощью оператора `'`, можно преобразовать в уравнение, производные в котором записаны с помощью оператора `diff`, с помощью функции `rewrite(<уравнение>, diff)`, указав в качестве целевой функции при ее вызове оператор `diff` (см. разд. 4.3). Здесь `<уравнение>` – дифференциальное уравнение, записанное либо явным образом, либо через

идентификатор переменной, которой оно было ранее присвоено. Например, в результате выполнения команды `rewrite(y''(x) + x*y'(x) = 0, diff)` получится то же самое уравнение $y''(x) + x \cdot y'(x) = 0$, но записанное в виде: $\frac{\partial^2 y(x)}{\partial x^2} + x\frac{\partial y(x)}{\partial x} = 0$.

Обыкновенное дифференциальное уравнение называется ***линейным***, если искомая функция $y(x)$ и ее производные входят в уравнение только в первой степени: $c_n(x)y^{(n)}(x) + \ldots + c_2(x)y''(x) + c_1(x)y'(x) + c_0(x)y(x) = \varphi(x)$. В этом случае множители $c_n(x)$, …, $c_2(x)$, $c_1(x)$, $c_0(x)$ при искомой функции $y(x)$ и ее производных называются ***коэффициентами*** линейного обыкновенного дифференциального уравнения. Коэффициенты $c_n(x)$, …, $c_1(x)$, $c_0(x)$ линейного обыкновенного дифференциального уравнения могут зависеть только от аргумента искомой функции x.

Если линейное уравнение имеет вид $c_n(x)y^{(n)}(x) + \ldots + c_2(x)y''(x) + c_1(x)y'(x) + c_0(x)y(x) = 0$, т.е. содержит только члены, пропорциональные искомой функции $y(x)$ и ее производным ($\varphi(x) = 0$), такое линейное дифференциальное уравнение называется ***однородным***. Если коэффициенты $c_n(x)$, …, $c_1(x)$, $c_0(x)$ при искомой функции $y(x)$ и ее производных не зависят от x, такое линейное дифференциальное уравнение называется линейным дифференциальным уравнением с постоянными коэффициентами. Оно также может быть однородным или неоднородным.

Чтобы задать линейное однородное обыкновенное дифференциальное уравнение, достаточно задать его коэффициенты в определенной последовательности. Например, если известно, что $a(x)$, $b(x)$, $c(x)$ – коэффициенты линейного однородного дифференциального уравнения, заданные в последовательности увеличения порядка производной, то очевидно, что соответствующее дифференциальное уравнение имеет вид: $a(x)y(x) + b(x)y'(x) + c(x)y''(x) = 0$.

В MuPAD с помощью функции `ode::mkODE` можно по заданным коэффициентам сформировать левую часть линейного однородного обыкновенного дифференциального уравнения $c_n(x)y^{(n)} + \ldots + c_2(x)y'' + c_1(x)y' + c_0(x)y = 0$. Для этого функцию `ode::mkODE` вызывают следующим образом:

```
<Результат> := ode::mkODE(<список коэффициентов>,
                          <искомая функция>, <аргумент функции>)
```

Параметр `<список коэффициентов>` – коэффициенты линейного дифференциального уравнения, которые необходимо указать в виде

списка через запятую в последовательности увеличения порядка производной искомой функции. Порядок уравнения определяется количеством элементов в <списке коэффициентов>. Например, если <список коэффициентов> содержит три элемента, то функция ode::mkODE создаст дифференциальное уравнение второго порядка. Поэтому в зависимости от вида уравнения <список коэффициентов> может содержать нулевые элементы. В качестве параметра <искомая функция> необходимо указать идентификатор символьной переменной, которой в уравнении необходимо обозначить искомую функцию, а в качестве параметра <аргумент функции> – идентификатор переменной, от которой должна зависеть искомая функция. Результатом выполнения функции ode::mkODE является левая часть линейного однородного обыкновенного дифференциального уравнения – выражение, полученное в результате линейной комбинации искомой функции и ее производных с указанными коэффициентами. Например, в результате выполнения команды ode::mkODE([x^2, 0, -2*x], y, x) получится следующее выражение: $x^2 y(x) - 2x\dfrac{\partial^2 y(x)}{\partial x^2}$. Приравняв это выражение к нулю, можно получить линейное однородное обыкновенное дифференциальное уравнение.

С помощью функции ode::isLODE можно проверить, является ли заданное выражение левой частью линейного обыкновенного дифференциального уравнения $c_n(x)y^{(n)} + \ldots + c_2(x)y'' + c_1(x)y' + c_0(x)y = 0$. Для этого функцию ode::isLODE вызывают следующим образом:

<Результат> := **ode::isLODE**(<выражение>,
<искомая функция>, <опция>)

Анализируемое <выражение> на месте первого параметра функции ode::isLODE можно указать явным образом или с помощью идентификатора переменной, которой оно было ранее присвоено. При этом все производные искомой функции в анализируемом <выражении> должны быть записаны через оператор diff (например, diff(y(x), x), а не y'(x)). В качестве параметра <искомая функция> необходимо указать идентификатор и в скобках аргумент функции, относительно которой необходимо анализировать исходное <выражение>. Например: y(x). Если функция ode::isLODE вызывается без <опций>, то она возвращает TRUE (или FALSE), если <выражение> является (не является) левой частью линейного обыкновенного дифференциального уравнения относительно указанной <искомой функции>.

Например, чтобы проверить, является ли дифференциальное уравнение $x^2 - y'(x) = 0$ линейным, можно написать команду `ode::isLODE(x^2-diff(y(x), x), y(x))`.

Функцию `ode::isLODE` можно вызвать с одной из следующих <опций>: `Homogeneous`; `HlodeOverRF`; `Hlode`; `LodeOverRF`; `Lode`. Эти <опции> позволяют определить, является ли обыкновенное дифференциальное уравнение, левая часть которого представлена указанным <выражением>: `Homogeneous` – однородным; `Lode` – линейным; `LodeOverRF` – линейным с рациональными коэффициентами; `Hlode` – однородным линейным; `HlodeOverRF` – однородным линейным с рациональными коэффициентами. Если <выражение> можно считать левой частью уравнения с указанными свойствами, то функция `ode::isLODE`, вызванная с одной из опций `HlodeOverRF`, `Hlode`, `LodeOverRF`, `Lode`, возвращает последовательность, которая состоит из исходного выражения, идентификатора искомой функции, ее аргумента и порядка соответствующего дифференциального уравнения. С <опцией> `Homogeneous` или без <опции> функция `ode::isLODE` возвращает только `TRUE` или `FALSE`. Например, в результате выполнения команды `ode::isLODE(x^2 - diff(y(x), x), y(x))` получится `TRUE`, а в результате выполнения команды `ode::isLODE(x^2 - diff(y(x), x), y(x), Lode)` получится последовательность: $x^2 - \frac{\partial y(x)}{\partial x}$, y, x, 1. Значит, выражение $x^2 - \frac{\partial y(x)}{\partial x}$ является левой частью линейного обыкновенного дифференциального уравнения порядка 1 относительно искомой функции $y(x)$.

Функция `ode::vectorize` позволяет извлечь коэффициенты из однородного линейного дифференциального уравнения $c_n(x)y^{(n)} + \ldots + c_2(x)y'' + c_1(x)y' + c_0(x)y = 0$. Для этого функцию `ode::vectorize` вызывают следующим образом:

<Результат> := **ode::vectorize**(<выражение>,
<искомая функция>, <аргумент функции>, <порядок>)

Параметр <выражение> должен содержать левую часть однородного линейного дифференциального уравнения, в которой все производные искомой функции записаны через оператор `diff` (например, `diff(y(x), x)`, а не `y'(x)`). Левая часть дифференциального уравнения

на месте параметра <выражение> может быть записана явно или через идентификатор присвоенной ранее переменной. После параметра <выражение> следует указать, по отношению к какой функции необходимо анализировать <выражение> и каким идентификатором обозначен аргумент этой функции, а также предполагаемый порядок анализируемого дифференциального уравнения. В результате функция `ode::vectorize` возвращает список, состоящий из выражений, являющихся коэффициентами при искомой функции и ее производных, в последовательности увеличения порядка производной. Например, в результате выполнения команды `ode::vectorize(2*x*diff(y(x), x$3) - y(x), y, x, 3)` получится список [–1, 0, 0, 2x].

Нормальным видом дифференциального уравнения называют такую полученную в результате тождественных преобразований форму записи этого уравнения, при которой все его операнды располагаются в левой части уравнения, а коэффициент при производной искомой функции высшего порядка равен единице. Функция `ode::normalize` позволяет преобразовать однородное линейное дифференциальное уравнение $c_n(x)y^{(n)} + \ldots + c_2(x)y'' + c_1(x)y' + c_0(x)y = 0$ к нормальному виду. Для этого функцию `ode::normalize` вызывают следующим образом:

<Результат> := **ode::normalize**(<выражение>,
<искомая функция>, <аргумент функции>, <порядок>)

Параметр <выражение> также должен содержать левую часть однородного линейного дифференциального уравнения, в которой все производные искомой функции записаны через оператор `diff` (например, `diff(y(x), x)`, а не `y'(x)`). Левая часть дифференциального уравнения на месте параметра <выражение> может быть записана явно или через идентификатор присвоенной ранее переменной. После параметра <выражение> необходимо указать, каким идентификатором в <выражении> обозначена функция, для нахождения которой может быть использовано преобразуемое уравнение, и каким идентификатором обозначен аргумент этой функции, а далее – порядок анализируемого дифференциального уравнения. В результате функция `ode::normalize` возвращает выражение, представляющее собой левую часть дифференциального уравнения, эквивалентного исходному, но приведенного к нормальному виду. Например, в результате выполнения команды `ode::`

`normalize(2*x*diff(y(x), x $ 3) - y(x), y, x, 3)` получится выражение $\frac{\partial^3 y(x)}{\partial x^3} - \frac{y(x)}{2x}$.

Функция `ode::getOrder` позволяет определить порядок (линейного или нелинейного) обыкновенного дифференциального уравнения $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Для этого функцию `ode::getOrder` вызывают следующим образом:

<Результат> := **ode::getOrder**(<выражение>,
<искомая функция>)

Параметр <выражение> должен содержать левую часть обыкновенного дифференциального уравнения вида $F(y^{(n)}, \ldots, y'', y', y, x) = 0$, в которой все производные искомой функции записаны через оператор `diff` (например, `diff(y(x), x)`, а не `y'(x)`). Левая часть дифференциального уравнения на месте параметра <выражение> может быть записана явно или через идентификатор присвоенной ранее переменной. После параметра <выражение> следует указать, по отношению к какой функции необходимо анализировать <выражение>. <Искомая функция> должна быть записана вместе с аргументом. Например: `y(x)`. В результате функция `ode::getOrder` возвращает целое положительное число – наибольший порядок производной искомой функции, встречающейся в <выражении>. Если в <выражении> ни одной производной искомой функции не встречается, функция `ode::getOrder` возвращает $-\infty$. Например, в результате выполнения команды `ode::getOrder(2*x*diff(y(x), x $ 3) - y(x), y(x))` получится 3.

Обратите внимание, что при вызове многих из описанных далее функций, предназначенных в MuPAD для работы с обыкновенными дифференциальными уравнениями (линейными и нелинейными), необходимо указывать только левую часть уравнения вида $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Кроме того существуют функции (например, `ode::solve` или `numeric::odesolve2`), для корректного вызова которых решаемое дифференциальное уравнение необходимо представить в виде специального объекта.

8.1. Аналитическое решение обыкновенного дифференциального уравнения

8.1.1. Общее и частное решение обыкновенного дифференциального уравнения

Для аналитического решения обыкновенного дифференциального уравнения в MuPAD можно использовать функцию `ode::solve` или функцию `solve`, которая, если обнаружит, что решаемое уравнение является дифференциальным, автоматически вызывает функцию `ode::solve`. В этом случае функцию `solve` следует вызывать по тем же правилам и с теми же параметрами, как и функцию `ode::solve`.

Для решения дифференциального уравнения с помощью функции `ode::solve` его необходимо представить в виде специального ode-объекта. Чтобы корректно преобразовать дифференциальное уравнение в ode-объект удобно использовать функцию `ode`, которую вызывают следующим образом:

`<Ode-объект> := ode(<уравнение>, <искомая функция>)`

В качестве параметра `<уравнение>` необходимо записать обыкновенное дифференциальное уравнение или его левую часть, если уравнение приведено к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Уравнение или его левая часть может быть записано явно или через идентификатор присвоенной ранее переменной. После параметра `<уравнение>` необходимо указать, по отношению к какой функции следует считать данное `<уравнение>` дифференциальным. `<Искомая функция>` должна быть записана вместе с аргументом. Например: `y(x)`. Производные `<искомой функции>` в `<уравнении>` можно записывать как через оператор `diff`, так и через оператор `'`. Функция `ode` приводит `<уравнение>` к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$, выделяет его левую часть и формирует `<ode-объект>`, который в поле вывода отображается следующим образом:

ode(`<выражение>`, `<искомая функция>`)

где `<выражение>` – левая часть преобразованного уравнения, в котором все производные искомой функции записаны через оператор `'` (т.е. `y'(x)`, а не `diff(y(x), x)` и т.д.). Например, в результате выполнения команды `ode(2*x*diff(y(x), x $ 3) - y(x) = sin(x), y(x))` в поле вывода получится: $\mathrm{ode}(2xy'''(x) - y(x) - \sin(x), y(x))$; в результате

выполнения команды `ode(2*x*diff(y(x), x $ 3) - y(x), y(x))` получится: ode($2xy'''(x) - y(x)$, $y(x)$); а в результате выполнения команды `ode(2*x*y'''(x) - y(x), y(x))` тоже получится: ode($2xy'''(x) - y(x)$, $y(x)$).

Если дифференциальное уравнение требуется решать при заданных начальных условиях (известных значениях искомой функции и ее производных, соответствующих фиксированному значению аргумента), то эти начальные условия необходимо включить в состав ode-объекта. Для этого функцию `ode` вызывают следующим образом:

<Ode-объект> := **ode**({<уравнение>,
<начальные условия>}, <искомая функция>)

<Начальные условия> необходимо записывать в виде <искомая функция> = <значение> или <производная искомой функции> = <значение> для фиксированного значения аргумента искомой функции. Например, если требуется решить дифференциальное уравнение $y' + 2y - y^2 = 0$ при начальном условии $y(0) = -1/4$, то для создания ode-объекта функцию `ode` вызывают следующим образом: `ode({y'(x) + 2*y(x) -y(x)^2 = 0, y(0) = -1/4}, y(x))`. Если для некоторого фиксированного значения аргумента известны и значение искомой функции, и значение ее производных, все начальные условия перечисляются через запятую. Например, если необходимо решить дифференциальное уравнение $y'' = 32 \sin^3 y \cdot \cos y$ при начальных условиях $y(1) = \pi/2$, $y'(1) = 4$, то для создания ode-объекта функцию `ode` вызывают следующим образом: `ode({y''(x) = 32*sin(y(x))^3*cos(y(x)), y(1) = PI/2, y'(1) = 4}, y(x))`.

Указывая начальные условия, следует помнить, что для получения частного решения обыкновенного дифференциального уравнения n-го порядка необходимо n начальных условий. Если при решении дифференциального уравнения использовано меньшее количество начальных условий, то искомая функция будет содержать константы интегрирования.

Если <начальные условия> заданы, то функция `ode` также возвращает ode-объект, но в поле вывода этот объект отображается с учетом указанных значений искомой функции и ее производных:

ode({<выражение>, <начальные условия>},
<искомая функция>)

Здесь <выражение> – левая часть дифференциального уравнения, преобразованного к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Например, в результате

выполнения команды `ode({y''(x) = 32*sin(y(x))^3*cos(y(x)), y(1) = PI/2, y'(1) = 4}, y(x))` в поле вывода ode-объект будет отображен в виде:

$$\mathrm{ode}\left(\left\{y'(1)=4,\, y''(x)-32\cos(y(x))\sin(y(x))^3,\, y(1)=\frac{\pi}{2}\right\},\, y(x)\right).$$

Полученный в результате использования функции `ode` объект рекомендуется присвоить какой-либо переменной, чтобы в дальнейшем при решении дифференциального уравнения ее можно было использовать в качестве идентификатора ode-объекта.

Для решения обыкновенного дифференциального уравнения, представленного в виде ode-объекта, можно использовать функцию `ode::solve` (или `solve`) следующим образом:

`<Результат> := ode::solve(<ode-объект>, <опции>)`

`<Результат> := solve(<ode-объект>, <опции>)`

Если в качестве первого параметра функции `solve` указать ode-объект, то она вызывает функцию `ode::solve` с теми же параметрами, с которыми была вызвана сама.

В качестве `<ode-объекта>` рекомендуется использовать идентификатор переменной, которой ранее был присвоен ode-объект, созданный с помощью функции `ode`. Например, если ранее в программе для некоторого дифференциального уравнения был создан ode-объект, обозначенный переменной `OdeUr`, то для решения этого уравнения можно написать команду `ode::solve(OdeUr)`. Подробный пример решения дифференциального уравнения с помощью функции `ode::solve` без опций приведен на рис. 8.1.

Функцию `ode::solve` (или функцию `solve` для ode-объекта) можно вызвать с одной из следующих `<опций>`: `Type = <тип уравнения>`, `MaxDegree = n`, `IgnoreSpecialCases`, `IgnoreAnalyticConstraints`. В процессе решения дифференциального уравнения после предварительных преобразований, как правило, выполняются промежуточные вычисления, такие как интегрирование, а в некоторых случаях и нахождение корней алгебраических или трансцендентных уравнений. Поэтому опции `MaxDegree = n`, `IgnoreSpecialCases`, `IgnoreAnalyticConstraints` позволяют регулировать правила выполнения этих промежуточных вычислений: `MaxDegree = n` – указывает,

корни многочленов до каких степеней включительно должны вычисляться аналитически (см. разд. 5.1); `IgnoreAnalyticConstraints` – разрешает при функциональных преобразованиях или интегрировании применять ко всем символьным переменным правила, справедливые только для переменных, обладающих фиксированными свойствами (см. разд. 6.6.1); `IgnoreSpecialCases` – позволяет при функциональных преобразованиях или интегрировании игнорировать особые случаи, например обращение в ноль знаменателя или другие особые точки (см. разд. 6.6.1).

Таблица 8.1

Типы решаемых функцией `ode::solve` обыкновенных дифференциальных уравнений

Тип обыкновенного дифференциального уравнения	**Стандартный идентификатор `<тип уравнения>`**
Однородное уравнение первого порядка $y' = g\left(\frac{y}{x}\right)$	`Homogeneous`
Уравнение первого порядка, разрешенное относительно производной $y' = f(y, x)$	`ExactFirstOrder`
Уравнение второго порядка, разрешенное относительно старшей производной $y'' = f(y', y, x)$	`ExactSecondOrder`
Уравнение Бернулли $y' = Q(x)y^k - P(x)y$	`Bernoulli`
Уравнение Риккати $y' = a(x)\cdot y^2 + b(x)\cdot y + c(x)$	`Riccati`
Уравнение Абеля $y' = a(x)\cdot y^3 + b(x)\cdot y^2 + c(x)\cdot y + r(x)$	`Abel`
Уравнение Чини $y' = a(x)\cdot y^n + b(x)\cdot y + c(x)$	`Chini`
Уравнение Лагранжа $y = x\cdot f(y') + g(y')$	`Lagrange`
Уравнение Клеро (частный случай уравнения Лагранжа) $y = x\cdot y' + g(y')$	`Clairaut`

Если функцию `ode::solve` вызывают без опции `Type = <тип уравнения>`, она сначала анализирует `<ode-объект>`, чтобы определить, к какому типу относится данное дифференциальное уравнение. Далее в зависимости от типа решаемого уравнения выбирается метод его решения. Опция `Type = <тип уравнения>` позволяет указать функции `ode::solve`, каким методом следует решать данное дифференциальное уравнение. Если с помощью опции `Type` указан `<тип уравнения>`, то этап анализа `<ode-объекта>` пропускается и сразу используется метод, разработанный для решения уравнений указанного типа. Если в действительности решаемое уравнение относится к другому типу дифференциальных уравнений, то решение получить не удастся (в этом случае функция `ode::solve` возвращает `FAIL`). Для обозначения `<типов уравнения>` используются стандартные идентификаторы. В табл. 8.1 перечислены типы решаемых функцией `ode::solve` уравнений и соответствующие им стандартные идентификаторы. Большинство из них соответствует дифференциальным уравнениям первого порядка. Предварительно дифференциальные уравнения высших порядков заменой переменных автоматически преобразуются в систему дифференциальных уравнений первого порядка. После этого определяется, к какому типу относятся входящие в систему уравнения.

Решение дифференциального уравнения может содержать константы интегрирования (если начальные условия не заданы или их количество недостаточно для получения единственного частного решения). В процессе решения уравнения функция `ode::solve` последовательно генерирует константы интегрирования в формате C** в количестве, необходимом для записи полученного решения. Например: C5, C6, …, C23 и т.д. Далее функция `ode::solve` пытается выразить значения этих констант через начальные условия (если они заданы).

Соответственно, если начальные условия не заданы, функция `ode::solve` находит *общее решение* дифференциального уравнения, количество констант интегрирования в котором совпадает с порядком дифференциального уравнения. Каждое добавление нового начального условия в `<ode-объект>`, используемый при вызове функции `ode::solve`, уменьшает количество констант интегрирования в его решении. Если количество начальных условий достаточно, чтобы выразить через них все константы интегрирования, то функция `ode::solve` находит *частное решение* соответствующего дифференциального уравнения.

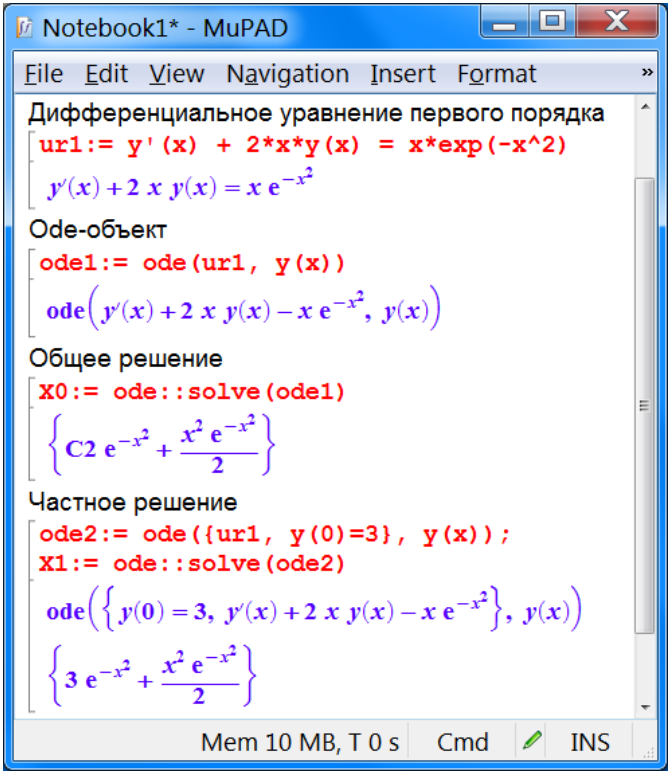


Рис. 8.1. Использование функции `ode::solve` для решения обыкновенных дифференциальных уравнений

Если функция `ode::solve` МОЖЕТ решить дифференциальное уравнение, она возвращает множество `{}`, каждый элемент которого является решением соответствующего дифференциального уравнения, указанного при вызове этой функции с помощью `<ode-объекта>`. Примеры использования функции `ode::solve` для нахождения общего и частного решения обыкновенного дифференциального уравнения первого порядка приведены на рис. 8.1.

Если функция `ode::solve` НЕ МОЖЕТ решить указанное дифференциальное уравнение, то соответствующая команда дублируется в поле

вывода, как при отложенных вычислениях. Например, при попытке выполнить команды

```
odeUr := ode(y''(x) = x*sqrt(y(x)), y(x)):
ode::solve(odeUr)
```

в поле вывода появится: solve(ode($y''(x) - x\sqrt{y(x)}$, $y(x)$)).

С помощью последовательности раскрывающихся списков `General Math` → `Solve` → `ODE`, расположенной на командной панели `Command Bar`, в текст программы можно вставить шаблон `solve(ode(#, #(#)))` для поиска решений обыкновенного дифференциального уравнения. Поля, отмеченные символом #, необходимо заполнить в соответствии с описанными выше правилами.

8.1.2. Фундаментальная система решений линейного однородного обыкновенного дифференциального уравнения

Кроме общего или частного решения для линейного однородного дифференциального уравнения можно также записать фундаментальную систему решений, которая представляет собой набор линейно независимых функций, любые линейные комбинации которых являются решениями этого дифференциального уравнения. Общее решение линейного однородного дифференциального уравнения представляет собой такую линейную комбинацию $C1 \cdot f_1(x) + C2 \cdot f_2(x) + \ldots + CN \cdot f_N(x)$ линейно независимых функций $f_1(x)$, $f_2(x)$, …, $f_N(x)$, где в качестве коэффициентов выступают константы интегрирования C1, C2, … CN. Поэтому ***фундаментальная система решений линейного однородного дифференциального уравнения*** – это множество $\{f_1(x), f_2(x), \ldots, f_N(x)\}$ линейно независимых функций при константах интегрирования в записи общего решения этого уравнения. И наоборот, общее решение – это линейная комбинация фундаментальных решений с неизвестными (незаданными) коэффициентами, а частное решение – это линейная комбинация фундаментальных решений с фиксированными коэффициентами.

Если для линейного однородного дифференциального уравнения необходимо найти фундаментальную систему решений, выраженную через рациональные, полиномиальные или экспоненциальные вещественные функции, в MuPAD можно воспользоваться функциями

`ode::rationalSolutions`, `ode::polynomialSolutions` или `ode::exponentialSolutions` соответственно. Для получения решения линейного однородного дифференциального уравнения эти функции вызывают следующим образом:

```
<Результат> := ode::rationalSolutions(<уравнение>,
                                <искомая функция>, <опция>)
<Результат> := ode::polynomialSolutions(<уравнение>,
                                <искомая функция>, <опция>)
<Результат> := ode::exponentialSolutions(<уравнение>,
                                <искомая функция>, <опция>)
```

Здесь в качестве параметра `<уравнение>` необходимо записать левую часть приведенного к виду $c_n(x)y^{(n)} + \ldots + c_2(x)y'' + c_1(x)y' + c_0(x)y = 0$ линейного однородного обыкновенного дифференциального уравнения, в которой все производные искомой функции записаны через оператор `diff` (например, `diff(y(x), x)`, а не `y'(x)`). Левая часть уравнения может быть записана явно или через идентификатор присвоенной ранее переменной (но без использования ode-объекта). Чтобы уравнение, производные в котором записаны с помощью оператора `'`, преобразовать в уравнение, производные в котором записаны с помощью оператора `diff`, можно воспользоваться функцией `rewrite(<уравнение>, diff)`, указав в качестве целевой функции идентификатор `diff` (см. разд. 4.3).

При вызове любой из функций `ode::rationalSolutions`, `ode::polynomialSolutions` или `ode::exponentialSolutions` после параметра `<уравнение>` необходимо указать, по отношению к какой функции следует решать данное `<уравнение>`. `<Искомая функция>` должна быть записана вместе с аргументом. Например: `y(x)`.

Для получения фундаментальной системы экспоненциальных решений дифференциального уравнения $y''(x) - y(x) = 0$ можно, например, написать команду `ode::exponentialSolutions(diff(y(x), x$2) - y(x), y(x))`.

Если любая из функций `ode::rationalSolutions`, `ode::polynomialSolutions` или `ode::exponentialSolutions` вызывается без `<опции>`, то она возвращает фундаментальную систему решений указанного `<уравнения>` в виде множества `{}`. При вызове каждой из этих функций можно использовать только одну `<опцию>`: `Generic`. Если функция вызвана с этой `<опцией>`, то она возвращает общее решение

дифференциального <уравнения> в виде выражения. Пример получения фундаментальной системы экспоненциальных решений линейного однородного дифференциального уравнения, а также его общего решения приведен на рис. 8.2.

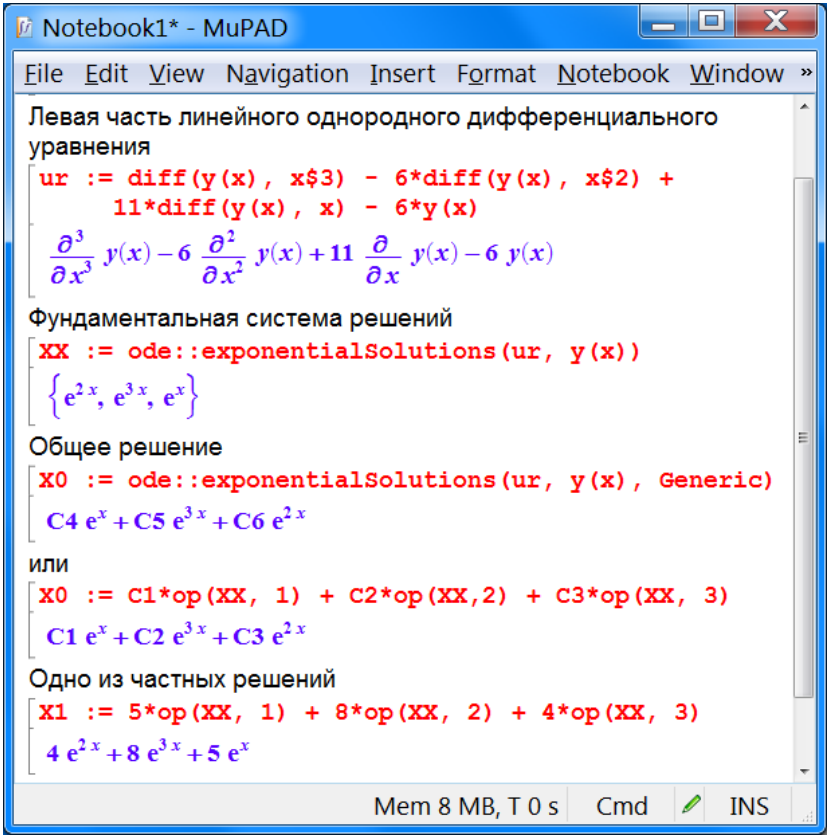


Рис. 8.2. Фундаментальная система решений линейного однородного обыкновенного дифференциального уравнения

Если в действительности фундаментальная система решений <уравнения> содержит еще какие-нибудь функции, кроме экспоненциальных (или если в <уравнение> не является линейным однородным), то

с помощью функции `ode::exponentialSolutions` решение такого дифференциального уравнения найти не удастся – появится сообщение об ошибке. Аналогично функции `ode::rationalSolutions` и `ode::polynomialSolutions` позволяют получить фундаментальную систему решений линейного однородного дифференциального уравнения только в том случае, если она не содержит никаких других функций, кроме рациональных или полиномиальных соответственно.

8.1.3. Решение обыкновенного дифференциального уравнения в виде разложения в ряд

При необходимости решение обыкновенного дифференциального уравнения, полученное с помощью функции `ode::solve`, можно разложить в ряд с помощью функции `series` (см. разд. 6.5). Например, после выполнения команд `ur := ode({y''(x) + 25*y(x), y(0) = 5, y'(0) = 20}, y(x)); X := ode::solve(ur)` получится решение дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ в виде $\{5\cos(5x) + 4\sin(5x)\}$. Разложить это решение в ряд вблизи точки $x = 0$ можно с помощью команды `series(op(X, 1), x=0)`, в результате выполнения которой получится: $5 + 20x - 125x^2/2 - 250x^3/3 + 3125x^4/24 + 625x^5/6 + O(x^6)$.

С помощью функции `ode::series` можно получить решение обыкновенного дифференциального уравнения сразу в виде разложения в ряд. Для этого функцию `ode::series` вызывают следующим образом:

```
<Результат> := ode::series(<уравнение>,
            <искомая функция>, <аргумент> = <значение>,
                                          <порядок>)
```

В качестве параметра `<уравнение>` необходимо записать левую часть приведенного к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$ обыкновенного дифференциального уравнения и его начальные условия (если есть). Левая часть уравнения может быть записана явно или через идентификатор присвоенной ранее переменной (но без использования ode-объекта). Левая часть уравнения и его начальные условия перечисляются через запятую в фигурных скобках `{}`. Если начальные условия отсутствуют, то фигурные скобки можно опустить. После параметра `<уравнение>` необходимо указать, по отношению к какой функции следует решать

данное <уравнение>. <Искомая функция> должна быть записана вместе с аргументом. Например: y(x). Производные <искомой функции> в <уравнении> можно записывать как через оператор diff, так и через оператор '. Далее с помощью параметра <аргумент> = <значение> следует указать, вблизи какой точки необходимо выполнить разложение в ряд, а с помощью параметра <порядок> задать, до какого порядка необходимо выполнить разложение. Эти параметры (<аргумент> = <значение> и <порядок>) имеют такой же смысл как при вызове функции series (см. разд. 6.5). Если параметр <порядок> не задан, то ряд будет содержать члены, порядок которых не превышает значение переменной среды ORDER (ORDER = 6 по умолчанию). Если разложение в ряд решения дифференциального уравнения необходимо выполнить вблизи нуля, то параметр <значение> и знак = перед ним можно опустить. Например, последовательность команд ur := {y''(x) + 25*y(x), y(0) = 5, y'(0) = 20}; ode::series(ur, y(x), x = 0) позволяет получить решение дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ сразу в виде ряда $5 + 20x - 125x^2/2 - 250x^3/3 + 3125x^4/24 + 625x^5/6 + O(x^6)$.

Из приведенных примеров видно, что результат выполнения функции ode::series и последовательного применения функций ode::solve и series для частного решения дифференциального уравнения одинаков. Однако для общего решения дифференциального уравнения результаты выполнения функции ode::series и последовательного применения функций ode::solve и series отличаются. Функция ode:: series выражает константы интегрирования через значения искомой функции и ее производных в точке, вблизи которой выполняется разложение в ряд. Например, в результате выполнения последовательности команд ur := ode(y''(x) + 25*y(x), y(x)); X := ode::solve(ur); series(op(X, 1), x = 0) получится общее решение дифференциального уравнения $y'' + 25y = 0$ в виде: $C1(1 - 25x^2/2 + 625x^4/24) + C2(5x - 125x^3/6 + 625x^5/24) + O(x^6)$, а в результате выполнения последовательности команд ur := {y''(x) + 25*y(x)}; ode::series(ur, y(x), x = 0) то же решение будет записано следующим образом: $\{y(0) + xy'(0) - 25x^2y(0)/2 - 25x^3y'(0)/6 + 625x^4y(0)/24 + 125x^5y'(0)/24 + O(x^6)\}$.

Если разложение в ряд выполняется вблизи особых точек решения дифференциального уравнения, функции ode::series не всегда удается

выразить константы интегрирования через значения искомой функции и ее производных в точке, вблизи которой выполняется разложение в ряд. В этом случае полученное функцией `ode::series` разложение также может содержать константы интегрирования.

8.1.4. Проверка достоверности решений обыкновенного дифференциального уравнения

Чтобы убедиться, что полученная функция действительно является решением некоторого дифференциального уравнения, необходимо подставить полученную функцию в это уравнение. Если при этом после выполнения всех операций дифференцирования и арифметических преобразований дифференциальное уравнение обращается в тождество при любых значениях аргумента функции, то данная функция считается решением этого дифференциального уравнения.

Таким образом, если решение дифференциального уравнения получено в виде множества `{}` (например, с помощью функции `ode::solve`), то сначала необходимо извлечь это решение из множества, например, с помощью функции `op`. В частности, для извлечения первого решения из множества, обозначенного ранее переменной `res`, можно написать команду `op(res, 1)`. Извлеченное решение будет представлять собой выражение. Если в проверяемом дифференциальном уравнении для записи производных использовался оператор `D` (или `'`), то он выполняет только дифференцирование функций, а не выражений. Чтобы в уравнение можно было подставить выражение и выполнить дифференцирование, необходимо, чтобы в таком уравнении для записи производных использовался оператор `diff`. Как уже отмечалось ранее, дифференциальное уравнение (или его левую часть), записанное через оператор `D` (или `'`), можно с помощью функции `rewrite` переписать через оператор `diff` (см. разд. 4.3). Для этого функцию `rewrite` необходимо вызвать следующим образом:

`<Результат> :=` **`rewrite`**`(<уравнение>, diff)`

В качестве параметра `<уравнение>` здесь обычно указывают идентификатор переменной, которой ранее было присвоено дифференциальное уравнение или его левая часть. В результате решаемое уравнение или его левая часть будет преобразована в уравнение или выражение, содержащее искомую функцию (вместе с аргументом) и ее производные, записанные

через отношение дифференциалов, например: $\frac{\partial\, y(x)}{\partial x}$, $\frac{\partial^2\, y(x)}{\partial x^2}$ и так далее вместо $y'(x)$, $y''(x)$ и т.д.

Например, если переменной `ur := y''(x) + 25*y(x) = 0` было присвоено уравнение $y'' + 25y = 0$, то после выполнения команды `rewrite(ur, diff)` в поле вывода получится уравнение:

$$\frac{\partial^2\, y(x)}{\partial x^2} + 25y(x) = 0\,.$$

Далее с помощью функции `subs` можно подставить в преобразованное уравнение (выражение) вместо искомой функции извлеченное из множества решений выражение:

```
<Результат> := subs(<уравнение>,
                    <искомая функция> = <решение>)
```

Здесь `<искомую функцию>` необходимо записывать вместе с аргументом. Например, `y(x)`. В качестве `<решения>` следует указать выражение либо явным образом, либо с помощью идентификатора переменной, которой оно ранее обозначено. Например, если дифференциальное уравнение $y'' + 25y = 0$ обозначено переменной `ur := diff(y(x), x, x) + 25*y(x) = 0`, то, чтобы вместо искомой функции `y(x)` подставить в это уравнение $\sin 5x$, можно написать команду `subs(ur, y(x) = sin(5*x))`.

В результате выполнения такой подстановки получится уравнение (или его левая часть), в котором все операции дифференцирования отложены. Далее для выполнения отложенных вычислений можно использовать функцию `eval`. Кроме того, чтобы убедиться, что после подстановки искомой функции в уравнение получается тождество, можно использовать функцию `is` (как при проверке корректности решений алгебраических или трансцендентных уравнений в разд. 5.1).

Таким образом, для проверки корректности полученного решения дифференциального уравнения необходимо реализовать в программе следующую последовательность действий:

1) если решения хранятся в множестве, извлечь одно выражение из множества;

2) записать в решаемом дифференциальном уравнении производные через отношение дифференциалов;

3) подставить в преобразованное уравнение вместо искомой функции извлеченное из множества выражение;

4) выполнить отложенные вычисления и проверку истинности полученного в результате подстановки логического выражения.

Например, для решения дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ и проверки полученного решения можно написать программу:

```
ur := y''(x) + 25*y(x) = 0;
odeur := ode({ur, y(0) = 5, y'(0) = 20}, y(x));
X := ode::solve(odeur);
ur2 := rewrite(ur, diff);
proverka := subs(ur2, y(x) = op(X, 1));
eval(proverka);
is(proverka)
```

Для линейных обыкновенных дифференциальных уравнений вместо последовательности функций `subs` и `eval` можно использовать функцию `ode::evalOde`, которая подставляет указанное выражение вместо неизвестной функции в левую часть линейного дифференциального уравнения, выполняет дифференцирование и упрощает полученное выражение. Для этого функцию `ode::evalOde` вызывают следующим образом:

<Результат> := **ode::evalOde**(<уравнение>,
<искомая функция> = <решение>)

В отличие от функции `subs` функция `ode::evalOde` может подставлять выражение вместо искомой функции только в левую часть линейного дифференциального уравнения, приведенного к виду $F(y^{(n)}, \ldots, y'', y', y, x) = 0$. Например, с помощью команды

```
ode::evalOde(diff(y(x), x $ 2) - 25*y(x),
                    y(x) = 3*exp(-5*x) + 2*exp(5*x))
```

можно в левую часть уравнения $y'' - 25y = 0$ вместо искомой функции $y(x)$ подставить выражение $3e^{-5x} + 2e^{5x}$. Так же как при использовании функции `subs` производные в <уравнении> должны быть записаны через отношение дифференциалов с помощью оператора `diff`.

Если в качестве параметра <уравнение> при вызове функции `ode::evalOde` указана не левая часть, а дифференциальное уравнение полностью, или дифференциальное уравнение, левая часть которого указана в качестве параметра <уравнение>, является нелинейным, появится сообщение об ошибке.

Так как функция `ode::evalOde` возвращает выражение, то далее с помощью функции `is` следует проверять, равно ли это выражение нулю.

Например, для решения линейного дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ и проверки полученного решения при написании программы вместо функций `subs` и `eval` можно использовать функцию `ode::evalOde` следующим образом:

```
ur := y''(x) + 25*y(x);
odeur := ode({ur, y(0) = 5, y'(0) = 20}, y(x));
X := ode::solve(odeur);
ur2 := rewrite(ur, diff);
proverka := ode::evalOde(ur2, y(x) = op(X, 1));
is(proverka=0)
```

Здесь вместо исходного дифференциального уравнения записана только его левая часть.

8.1.5. Графическое представление решения обыкновенного дифференциального уравнения

Если частное решение дифференциального уравнения получено в аналитическом виде, то можно построить график искомой функции.

Однако даже если уравнение имеет единственное решение, функция `ode::solve` возвращает множество, каждый элемент которого представляет собой выражение, зависящее от аргумента искомой функции, и является решением дифференциального уравнения. Чтобы извлечь из множества отдельное решение, можно воспользоваться функцией `op`. Если полученное выражение не содержит других не заданных ранее символьных переменных, то его можно использовать для создания графического объекта типа «двумерный график» с помощью функции `plot::Function2d` (см. разд. 3.1), который в дальнейшем можно визуализировать в поле вывода с помощью функции `plot`.

Таким образом, чтобы построить график функции, являющейся решением обыкновенного дифференциального уравнения, необходимо запрограммировать следующую последовательность действий:

1) если решения хранятся в множестве, извлечь одно решение из множества;

2) создать графический объект типа «двумерный график» с помощью функции `plot::Function2d`;

3) визуализировать графический объект с помощью функции `plot`.

Например, для графического представления решения дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ на интервале $x \in [0, 5]$ можно написать последовательность команд:

```
ur := y''(x) + 25*y(x);
odeur := ode({ur, y(0) = 5, y'(0) = 20}, y(x));
X := ode::solve(odeur);
Y1 := op(X, 1);
gr1 := plot::Function2d(Y1, x = 0..5);
plot(gr1)
```

Кроме графика искомой функции можно построить графики ее производных. Если искомую функцию и ее первую производную необходимо изобразить на одном графике, то программа должна реализовывать следующую последовательность действий:

1) если решения хранятся в множестве, извлечь одно решение из множества;

2) вычислить производную от выражения, извлеченного из множества, по переменной, являющейся аргументом искомой функции;

3) с помощью функции `plot::Function2d` создать два графических объекта типа «двумерный график» для искомой функции и для ее производной;

4) визуализировать оба графических объекта с помощью одной функции `plot`.

Например, для графического представления функции, являющейся решением дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$, и производной этой функции на интервале $x \in [0, 5]$ можно написать последовательность команд:

```
ur := y''(x) + 25*y(x);
odeur := ode({ur, y(0) = 5, y'(0) = 20}, y(x));
X := ode::solve(odeur);
Y1 := op(X, 1);    Y2 := diff(Y1, x);
gr1 := plot::Function2d(Y1, x=0..5):
gr2 := plot::Function2d(Y2, x=0..5, Color=RGB::Red):
plot(gr1, gr2)
```

Результат выполнения этого программного кода приведен на рис. 8.3.

При решении некоторых задач анализ решения дифференциального уравнения удобно проводить, если построить зависимость значений первой производной искомой функции от значений самой искомой функции

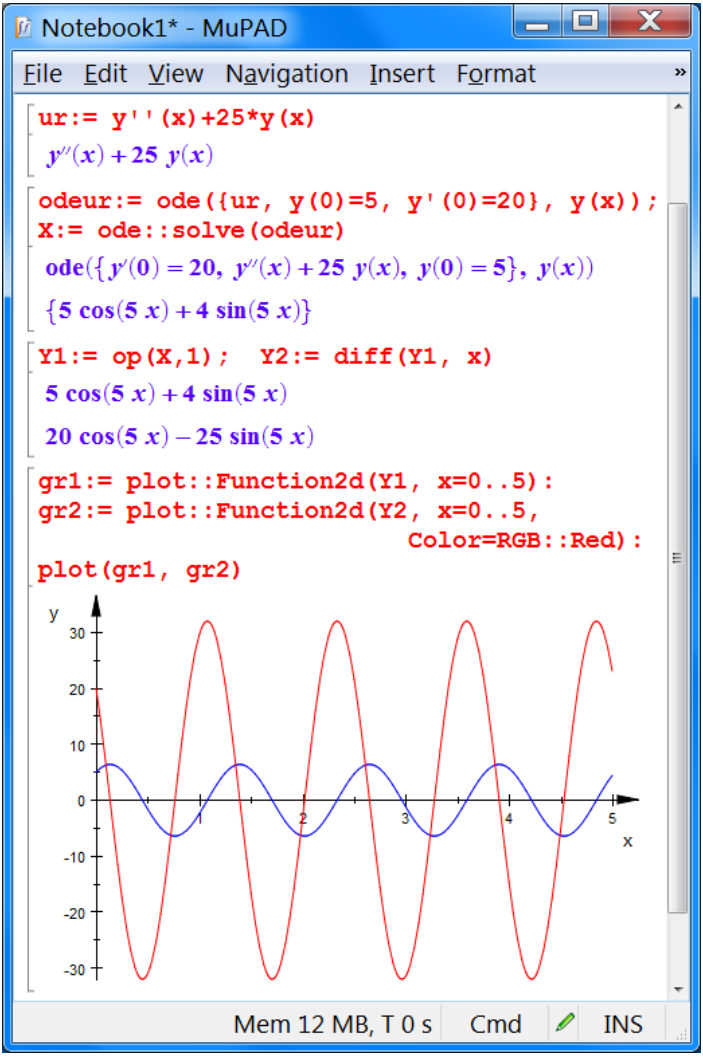


Рис. 8.3. График искомой функции, являющейся решением обыкновенного дифференциального уравнения, и производной этой функции

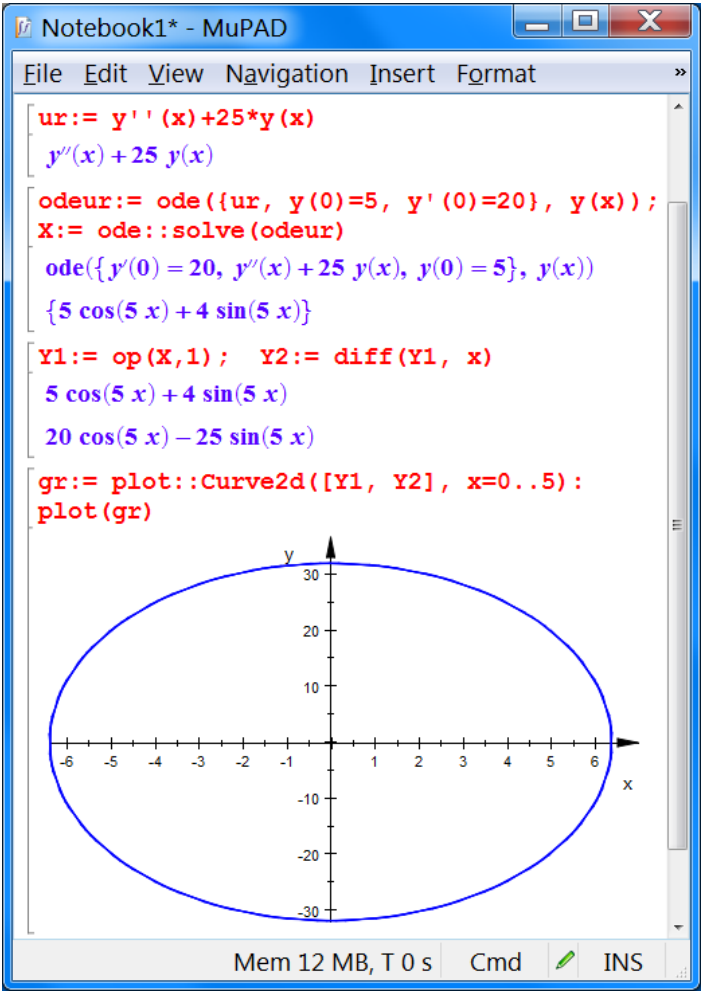


Рис. 8.4. Фазовая кривая, описывающая решение обыкновенного дифференциального уравнения

(например, зависимость скорости от координаты тела). Такой график $y'(y)$ называют ***фазовой кривой*** (фазовой траекторией), или графиком функции в фазовой плоскости ($y(x)$, $y'(x)$). Чтобы построить такую кривую, необходимо по оси абсцисс откладывать значения искомой функции, а по оси ординат – значения производной этой функции, вычисленные при одних и тех же значениях аргумента. Если известны два выражения, описывающие зависимость от одного и того же аргумента искомой функции и ее производной, то графический объект, соответствующий фазовой кривой для такого решения, можно создать с помощью функции `plot::Curve2d` (см. разд. 3.2).

Тогда для построения графика фазовой кривой, описывающей решение дифференциального уравнения, программа должна реализовывать следующую последовательность действий:

1) если решения хранятся в множестве, извлечь одно решение из множества;

2) вычислить производную от выражения, извлеченного из множества, по переменной, являющейся аргументом искомой функции;

3) с помощью функции `plot::Curve2d` для извлеченного выражения и для его производной создать графический объект типа «кривая»;

4) визуализировать графический объект с помощью функции `plot`.

Например, для построения графика фазовой кривой $y'(y)$, описывающей решение дифференциального уравнения $y'' + 25y = 0$ при начальных условиях $y(0) = 5$, $y'(0) = 20$ на интервале $x \in [0, 5]$ можно написать последовательность команд:

```
ur := y''(x) + 25*y(x);
odeur := ode({ur, y(0) = 5, y'(0) = 20}, y(x));
X := ode::solve(odeur);
Y1 := op(X, 1); Y2 := diff(Y1, x);
gr := plot::Curve2d([Y1, Y2], x = 0..5): plot(gr)
```

Результат выполнения этого программного кода приведен на рис. 8.4.

8.2. Аналитическое решение систем обыкновенных дифференциальных уравнений

Под ***системой обыкновенных дифференциальных уравнений*** понимают задачу об одновременном решении нескольких обыкновенных дифференциальных уравнений, каждое из которых связывает несколько функций, зависящих от одного и того же аргумента, с их производными.

Решением системы обыкновенных дифференциальных уравнений является множество функций, зависящих от одной (одной и той же) переменной. Под начальными условиями системы обыкновенных дифференциальных уравнений понимают известные значения искомых функций и их производных при фиксированном значении аргумента.

Для аналитического решения системы обыкновенных дифференциальных уравнений в MuPAD используется та же функция `ode::solve`, как и для решения одного дифференциального уравнения. При этом ode-объект, для которого вызывается функция `ode::solve`, должен содержать все уравнения системы. Для создания ode-объекта системы обыкновенных дифференциальных уравнений используется функция `ode`, при вызове которой в качестве параметра `<уравнение>` необходимо перечислить в фигурных скобках через запятую все уравнения системы и при необходимости указать для них начальные условия:

```
<Ode-объект> := ode({<уравнение1>, <уравнение2>, …
          <начальные условия>}, {<искомая функция1>,
                                  <искомая функция2>, …})
```

В качестве второго параметра при вызове функции `ode` для создания ode-объекта системы дифференциальных уравнений необходимо указать в фигурных скобках множество искомых функций. Количество искомых функций должно совпадать с количеством решаемых уравнений. Каждую искомую функцию необходимо указывать в виде ее идентификатора и аргумента этой функции (в круглых скобках после идентификатора функции). Например: `{x(t), y(t)}` или `{y_1(x), y_2(x)}`. Все искомые функции должны зависеть от одного и того же аргумента. При создании ode-объекта разрешается также перечислять уравнения (вместе с начальными условиями) и искомые функции не в фигурных, а в квадратных скобках.

Производные от искомых функций в уравнениях системы (как и для одного дифференциального уравнения) можно записывать, используя как оператор `D` (или `'`), так и оператор `diff`.

Например, чтобы для дальнейшего решения системы дифференциальных уравнений

$$\begin{cases} y_1' = -4y_2 + 6\cos x; \\ y_2' = y_1 \end{cases} \quad \text{(без начальных условий)}$$

создать ode-объект, можно выполнить следующую команду:

```
ode({y_1'(x) = -4*y_2(x) + 6*cos(x), y_2'(x) = y_1(x)},
                                        {y_1(x), y_2(x)})
```

А чтобы для решения системы дифференциальных уравнений

$$\begin{cases} x' = 2x + y + t - 1; \\ y' = -x + 4y - 5t + 1; \end{cases} \text{ при начальных условиях } x(0) = 5,\ y(0) = 2$$

создать ode-объект, можно выполнить следующую команду:

```
ode({x'(t) = 2*x(t) + y(t) + t - 1, y'(t) =
 -x(t) + 4*y(t) -5*t + 1, x(0) = 5, y(0) = 2}, {x(t), y(t)})
```

Если каждое из дифференциальных уравнений, входящих в систему, можно преобразовать к виду $F_i(y_1^{(n)}, \ldots, y_1'', y_1', y_1, y_2^{(n)}, \ldots, y_2'', y_2', y_2, \ldots, x) = 0$, то при создании ode-объекта такой системы можно указывать только левую часть каждого из дифференциальных уравнений. Например, для создания ode-объекта системы дифференциальных уравнений

$$\begin{cases} y_1' = -4y_2 + 6\cos x; \\ y_2' = y_1; \end{cases} \Rightarrow \begin{cases} y_1' + 4y_2 - 6\cos x = 0; \\ y_2' - y_1 = 0 \end{cases}$$

можно написать команду:

```
ode({y_1'(x) + 4*y_2(x) - 6*cos(x), y_2'(x) - y_1(x)},
                                        {y_1(x), y_2(x)})
```

После создания ode-объекта следует вызвать функцию `ode::solve` (или `solve`) для этого объекта в соответствии с правилами, описанными ранее в разд. 8.1.1. Например, для решения системы дифференциальных уравнений

$$\begin{cases} x'(t) = 2x(t) + y(t); \\ y'(t) = 5t; \end{cases}$$

можно выполнить следующие команды:

```
odesys := ode({x'(t) = 2*x(t) + y(t), y'(t) = 5*t},
                        {x(t), y(t)}); ode:: solve(odesys)
```

Если ode-объект содержит систему дифференциальных уравнений, то функция `ode::solve` возвращает множество `{}`, каждый элемент которого представляет собой список `[]` искомых функций. Если система имеет единственное решение, то множество содержит только один список. Список содержит такое количество элементов, сколько искомых функций было указано при создании соответствующего ode-объекта. Например, при решении системы дифференциальных уравнений

$$\begin{cases} x'(t) = 2x(t) + y(t); \\ y'(t) = 5t; \end{cases}$$ при начальных условиях $x(0) = 8$, $y(0) = 1$

в результате выполнения команд

```
odesys := ode({x'(t) = 2*x(t) + y(t), y'(t) = 5*t,
                        x(0) = 8, y(0) = 1}, {x(t), y(t)});
res := ode::solve(odesys); expand(res)
```

можно получить следующий результат (единственное решение):

$$\left\{\left[x(t) = \frac{73e^{2t}}{8} - \frac{5t}{4} - \frac{5t^2}{4} - \frac{9}{8},\ y(t) = \frac{5t^2}{2} + 1\right]\right\}.$$

Чтобы подставить полученный результат в исходную систему (выполнить проверку), необходимо из полученного функцией `ode::solve` множества извлечь один из списков. Для этого удобно воспользоваться функцией `op`. Например, для извлечения первого решения (списка) из множества, обозначенного ранее переменной `res`, можно воспользоваться командой `op(res, 1)`. Этот список целиком можно подставить в исходную систему дифференциальных уравнений с помощью функции `subs`. Нет необходимости извлекать из решения каждую искомую функцию отдельно, так как согласно правилам использования функции `subs`, если ранее подстановка вида <искомая функция> = <выражение> обозначена какой-либо переменной, то ее идентификатор можно использовать при вызове функции `subs` вместо явной записи соответствующей подстановки (см. разд. 4.1). Аналогично, если с помощью функции `subs` необходимо выполнить несколько подстановок, то составленный из них список `[]` также можно обозначить переменной, а ее идентификатор использовать при вызове функции `subs` вместо явного вида всех подстановок.

Перед подстановкой полученного решения необходимо убедиться, что в исходной системе уравнений производные записаны через отношение дифференциалов с помощью оператора `diff`, а не `D` или `'`. Переписать исходную систему через отношение дифференциалов можно с помощью функции `rewrite({<уравнение1>, <уравнение2>,…}, diff)`. Эту функцию можно применять не только к отдельным уравнениям (или выражениям), но и к спискам и к множествам. Тогда преобразуется каждый элемент списка (множества). К последовательностям применить функцию `rewrite` нельзя.

Notebook1* - MuPAD

File Edit View Navigation Insert Format

Последовательность исходных уравнений

```
sysur:= x'(t) = 2*x(t) +   y(t),
        y'(t) = 8*x(t) + 4*y(t)
```

$$x'(t) = 2\,x(t) + y(t),\ y'(t) = 8\,x(t) + 4\,y(t)$$

Решение при заданных начальных условиях

```
odesys:= ode({sysur, x(0)=6, y(0)=12},
                          {x(t), y(t)}):
res:= ode::solve(odesys):
res:= expand(res)
```

$$\left\{\left[y(t) = 16\,e^{6\,t} - 4,\ x(t) = 4\,e^{6\,t} + 2\right]\right\}$$

```
R:= op(res,1)
```

$$\left[y(t) = 16\,e^{6\,t} - 4,\ x(t) = 4\,e^{6\,t} + 2\right]$$

Преобразование исходной системы

```
sysur:= rewrite([sysur], diff)
```

$$\left[\frac{\partial}{\partial t}\,x(t) = 2\,x(t) + y(t),\ \frac{\partial}{\partial t}\,y(t) = 8\,x(t) + 4\,y(t)\right]$$

Проверка

```
proverka:= subs(sysur, R)
```

$$\left[\frac{\partial}{\partial t}\left(4\,e^{6\,t} + 2\right) = 24\,e^{6\,t},\ \frac{\partial}{\partial t}\left(16\,e^{6\,t} - 4\right) = 96\,e^{6\,t}\right]$$

```
proverka:= eval(proverka)
```

$$\left[24\,e^{6\,t} = 24\,e^{6\,t},\ 96\,e^{6\,t} = 96\,e^{6\,t}\right]$$

```
simplify(proverka)
```

[TRUE, TRUE]

Mem 12 MB, T 0 s Cmd INS

Рис. 8.5. Проверка решения системы обыкновенных дифференциальных уравнений

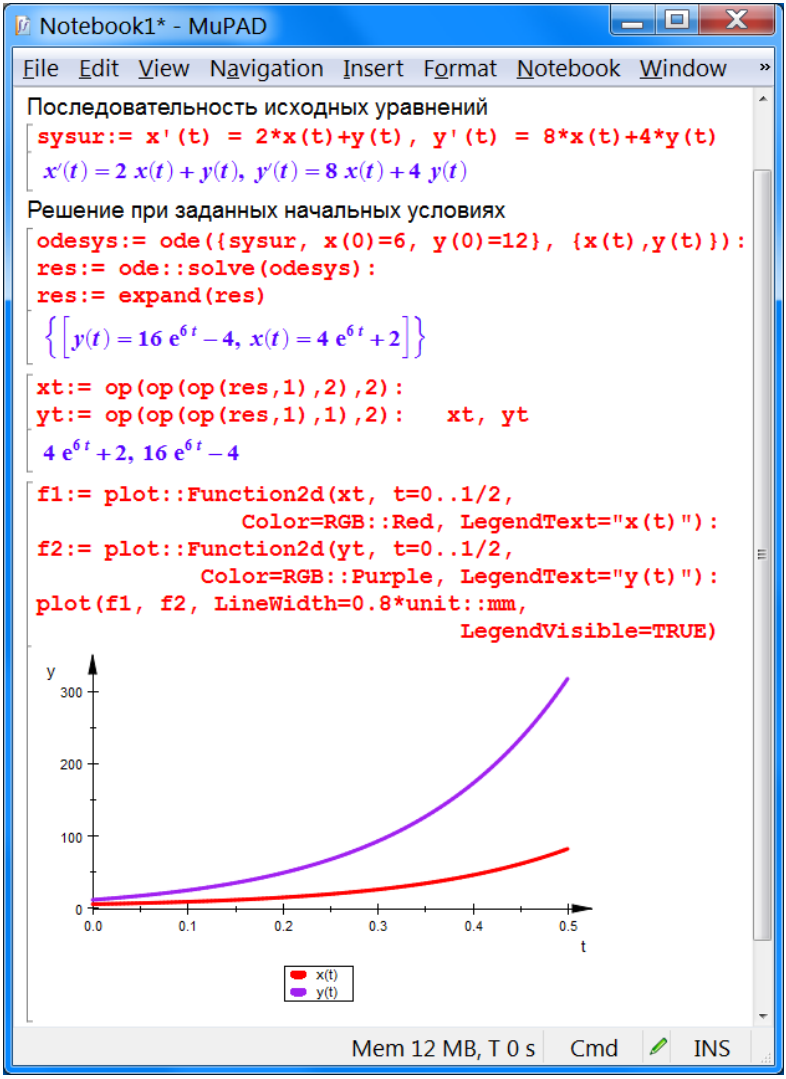


Рис. 8.6. График решения системы обыкновенных дифференциальных уравнений

Следует помнить, что после подстановки в систему (с помощью функции `subs`) функций, предположительно являющихся ее решением, операции дифференцирования остаются невыполненными (отложенными). Для принудительного их выполнения можно использовать функцию `eval`. В результате этого получится множество (или список) логических выражений. Если в результате применения функции `is` (или упрощения `simplify`) каждого из этих логических выражений получится `TRUE`, то найденные функции действительно являются решением исходной системы дифференциальных уравнений. Пример проверки решения системы обыкновенных дифференциальных уравнений приведен на рис. 8.5.

Для построения графиков искомых функций, полученных в результате решения системы обыкновенных дифференциальных уравнений, необходимо не только извлечь из множества список с решениями, но и из каждого элемента списка извлечь (с помощью функции `op`) выражение, описывающее зависимость искомой функции от ее аргумента. Например, если множество, полученное функцией `ode::solve`, присвоено переменной `res` и содержит несколько решений системы (несколько списков), то с помощью команды `op(op(op(res, 1), 1), 2)` можно из первого решения извлечь выражение, описывающее первую искомую функцию. Аналогично извлекают все выражения, описывающие остальные искомые функции для одного и того же решения. Для каждого выражения необходимо с помощью функции `plot::Function2d` создать соответствующий графический объект и все объекты изобразить на одном графике с помощью функции `plot`. Пример построения графиков функций, являющихся решением системы обыкновенных дифференциальных уравнений, приведен на рис. 8.6.

Если система состоит из двух дифференциальных уравнений, то каждое решение состоит из двух функций, зависящих от одного и того же аргумента. Для пары таких функций можно построить параметрическую кривую (траекторию). Для этого следует сначала извлечь из одного и того же решения два выражения, описывающих каждую из искомых функций, а затем, используя эти выражения, с помощью функции `plot::Curve2d` создать графический объект типа «кривая» и изобразить его на графике с помощью функции `plot`. Например, если переменным `xt`, `yt` присвоены зависящие от переменной `t` выражения, которые с помощью команд `xt := op(op(op(res, 1), 1), 2);` `yt := op(op(op(res, 1), 2), 2)` извлечены из решения системы обыкно-

венных дифференциальных уравнений, обозначенного переменной `res`, то для построения параметрической кривой этого решения можно написать следующие команды:

```
cr := plot::Curve2d([xt, yt], t=0..5); plot(cr)
```

8.3. Численное решение обыкновенных дифференциальных уравнений и их систем

Численно находят только частные решения обыкновенных дифференциальных уравнений и их систем. То есть, чтобы применить численные методы, каждое из решаемых уравнений должно иметь полный набор начальных условий для искомых функций и всех их производных, кроме производных наивысшего для данной системы уравнений порядка.

8.3.1. Методы численного решения обыкновенных дифференциальных уравнений

В вычислительной математике разработано большое количество методов решения обыкновенных дифференциальных уравнений первого порядка, имеющих вид:

$$\frac{dy(x)}{dx} = F(x, y(x)) \text{ при начальных условиях } y(x_0) = y_0 .$$

Чтобы найти значение $y(x_1)$ в точке $x_1 = x_0 + h$, необходимо преобразовать исходное уравнение и проинтегрировать его по x от x_0 до $x_0 + h$ следующим образом:

$$\frac{d\,y(x)}{dx} = F(x, y(x)) \;\Rightarrow\; d\,y(x) = F(x, y(x))dx \;\Rightarrow$$

$$\int_{y(x_0)}^{y(x_0+h)} d\,y(x) = \int_{x_0}^{x_0+h} F(x, y(x))dx \;\Rightarrow\; y(x_0 + h) - y(x_0) = \int_{x_0}^{x_0+h} F(x, y(x))dx \;\Rightarrow$$

$$y(x_0 + h) = y_0 + \int_{x_0}^{x_0+h} F(x, y(x))dx .$$

Таким образом, чтобы при известном значении $y_0 = y(x_0)$ найти значение $y(x_0 + h)$, необходимо вычислить интеграл, подынтегральная функция которого зависит от неизвестных пока значений функции $y(x)$ на интервале

$[x_0, x_0 + h]$. Так как аналитический вид искомой функции $y(x)$ неизвестен, первообразную для подынтегральной функции найти невозможно, поэтому этот интеграл вычисляют численно с помощью квадратурных формул. Для этого сначала с помощью замены переменных $x = x_0 + \alpha h$ ($\Rightarrow dx = h \cdot d\alpha$) удобно свести искомый интеграл к интегралу на интервале [0, 1]:

$$y(x_0 + h) = y_0 + h\int_0^1 F(x_0 + \alpha h, y(x_0 + \alpha h))d\alpha .$$

Далее для вычисления интеграла используют квадратурную формулу (см. разд. 6.62):

$$\int_0^1 F(x_0 + \alpha h, y(x_0 + \alpha h))d\alpha \approx \sum_{k=1}^{n} c_k F(x_0 + \alpha_k h, y(x_0 + \alpha_k h)) ,$$

где $\alpha_k \in [0, 1]$ – узлы, а c_k – весовые коэффициенты квадратурной формулы.

Хотя благодаря использованию квадратурной формулы удалось обойтись без вычисления первообразной, но в формуле для приближенного вычисления интеграла по-прежнему используются неизвестные пока значения искомой функции $y(x_0 + \alpha_k h)$ на интервале $\alpha_k \in [0, 1]$ (значения $y(x_0 + \alpha_k h)$ совпадают со значениями $y(x_k)$ при $x_k \in [x_0, x_0 + h]$). Эти значения можно определить по аналогии с полученным ранее выражением для $y(x_0 + h)$ следующим образом:

$$y(x_0 + \alpha_k h) = y_0 + h\int_0^{\alpha_k} F(x_0 + \alpha h, y(x_0 + \alpha h))d\alpha \approx$$

$$\approx y_0 + h\sum_{i=1}^{k-1} \beta_{ki} F(x_0 + \alpha_i h, y(x_0 + \alpha_i h)) .$$

Чтобы каждое искомое значение функции $y(x_0 + \alpha_k h)$ выражалось только через известные значения, здесь для вычисления каждого интеграла используются разные квадратурные формулы, содержащие $(k - 1)$ узел на интервале $[0, \alpha_k]$ с коэффициентами β_{ki}. При этом для вычисления интеграла на интервале $[0, \alpha_{k+1}]$ к узлам, выбранным на интервале $[0, \alpha_k]$, добавляют один узел, а суммирование выполняют с другими весовыми коэффициентами. Это позволяет использовать вычисленные ранее значения $y(x_0 + \alpha_1 h)$, $y(x_0 + \alpha_2 h)$, …, $y(x_0 + \alpha_k h)$ при последующем вычислении $y(x_0 + \alpha_{k+1} h)$. Тогда можно последовательно вычислить $y(x_0 + \alpha_k h)$ следующим образом:

$$y(x_0+\alpha_1 h)\approx y_0\,,$$
$$y(x_0+\alpha_2 h)\approx y_0+\beta_{21}hF(x_0+\alpha_1 h, y(x_0+\alpha_1 h))\,,$$
$$y(x_0+\alpha_3 h)\approx y_0+\beta_{31}hF(x_0+\alpha_1 h, y(x_0+\alpha_1 h))+$$
$$+\beta_{32}hF(x_0+\alpha_2 h, y(x_0+\alpha_2 h))\,,$$
$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$
$$y(x_0+\alpha_n h)\approx y_0+\sum_{i=1}^{n-1}\beta_{ni}hF(x_0+\alpha_i h, y(x_0+\alpha_i h))\,.$$

Для значений функции $F(x_0 + \alpha_i h,\ y(x_0 + \alpha_i h))$ принято использовать обозначения K_i. Тогда последовательность вычислений значений $y(x_0 + \alpha_k h)$ на интервале $\alpha_k \in [0, 1]$ можно переписать в виде:

$$y(x_0+\alpha_1 h)\approx y_0\,,$$
$$y(x_0+\alpha_2 h)\approx y_0+\beta_{21}hK_1\,,$$
$$y(x_0+\alpha_3 h)\approx y_0+\beta_{31}hK_1+\beta_{32}hK_2\,,$$
$$\dots\dots\dots\dots\dots\dots\dots\dots$$
$$y(x_0+\alpha_n h)\approx y_0+\sum_{i=1}^{n-1}\beta_{ni}hK_i\,.$$

Таким образом, численное решение обыкновенного дифференциального уравнения первого порядка вида $\frac{d\,y(x)}{dx}=F(x,y(x))$ при начальном условии $y(x_0)=y_0$ выполняется следующим образом:

$$K_1=F(x_0+\alpha_1 h, y(x_0+\alpha_1 h))\approx F(x_0+\alpha_1 h, y_0)\,,$$
$$K_2=F(x_0+\alpha_2 h, y(x_0+\alpha_2 h))\approx F(x_0+\alpha_2 h, y_0+\beta_{21}hK_1)\,,$$
$$K_3=F(x_0+\alpha_3 h, y(x_0+\alpha_3 h)\approx F(x_0+\alpha_3 h, y_0+\beta_{31}hK_1+\beta_{32}hK_2)\,,$$
$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$
$$K_n=F(x_0+\alpha_n h, y(x_0+\alpha_n h))\approx F(x_0+\alpha_n h, y_0+\sum_{i=1}^{n-1}\beta_{ni}hK_i)\,.$$

$$y(x_0+h)\approx y_0+h\sum_{k=1}^{n}c_k K_k\,.$$

Такая последовательность вычислений называется ***явным методом Рунге–Кутты***. Для вычисления значения $y(x_0 + h)$ по этому методу необходимо задать: 1) α_k – расположение n узлов квадратурной формулы

на интервале [0,1]; 2) β_{ki} – коэффициенты квадратурной формулы для вычисления промежуточных интегралов K_i; 3) c_k – коэффициенты квадратурной формулы для вычисления искомого значения функции $y(x_0 + h)$. Разным наборам коэффициентов α_k, β_{ki}, c_k соответствуют разные методы Рунге–Кутты. Компактная запись коэффициентов α_k, β_{ki}, c_k в виде таблицы

$$
\begin{array}{c|ccccc}
\alpha_1 & & & & & \\
\alpha_2 & \beta_{21} & & & & \\
\alpha_3 & \beta_{31} & \beta_{32} & & & \\
\dots & \dots & \dots & \dots & & \\
\alpha_n & \beta_{n1} & \beta_{n2} & \dots & \beta_{n\,n-1} & \\
\hline
 & c_1 & c_2 & \dots & c_{n-1} & c_n
\end{array}
$$

была предложена Бутчером и носит название ***схемы Бутчера***.

Обычно методы Рунге–Кутты строятся на использовании определенной квадратурной формулы интегрирования, которая задает определенное расположение узлов α_k и значения весовых коэффициентов c_k. Коэффициенты β_{ki} подбираются так, чтобы метод численного решения дифференциального уравнения давал точное решение для искомых функций, являющихся полиномами не выше некоторого порядка. Этот порядок характеризует точность конкретного метода Рунге–Кутты, называется ***порядком метода*** и указывается в справочниках наряду с коэффициентами α_k, β_{ki}, c_k. Некоторые частные случаи методов Рунге–Кутты были получены другими учеными раньше, чем были опубликованы методы Рунге–Кутты, поэтому они имеют самостоятельные названия. Например, метод Эйлера является методом первого порядка, базируется на вычислении интеграла методом левых прямоугольников, включает в себя следующую последовательность вычислений и описывается соответствующей ей схемой Бутчера:

$$
\begin{aligned}
&K_1 = F(x_0, y_0), \\
&y(x_0 + h) \approx y_0 + hK_1.
\end{aligned}
\qquad
\begin{array}{c|c}
0 & \\
\hline
 & 1
\end{array}
$$

В методе Эйлера используется только один ($n = 1$) узел $\alpha_1 = 0$ с весовым коэффициентом $c_1 = 1$ (коэффициенты β_{ki} отсутствуют). Классический метод Рунге–Кутты является методом четвертого порядка, включает в себя следующую последовательность вычислений и описывается схемой Бутчера:

$$K_1 = F(x_0, y_0),$$

$$K_2 = F(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}K_1),$$

$$K_3 = F(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}K_2),$$

$$K_4 = F(x_0 + h, y_0 + hK_3),$$

$$y(x_0 + h) \approx y_0 + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4).$$

$$\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Метод Бутчера шестого порядка описывается схемой, представленной на рис. 8.7.

Для получения решения обыкновенного дифференциального уравнения с заданной точностью метод Рунге–Кутты некоторого порядка дополняют методом другого порядка с теми же коэффициентами α_k, β_{ki}, но другими коэффициентами c_k. Такие дополненные методы Рунге–Кутты называются ***вложенными*** и описываются схемой Бутчера следующего вида:

$$\begin{array}{c|ccccc}
\alpha_1 & & & & & \\
\alpha_2 & \beta_{21} & & & & \\
\alpha_3 & \beta_{31} & \beta_{32} & & & \\
\dots & \dots & \dots & \dots & & \\
\alpha_n & \beta_{n1} & \beta_{n2} & \dots & \beta_{n\,n-1} & \\
\hline
 & c_1 & c_2 & \dots & c_{n-1} & c_n \\
 & \tilde{c}_1 & \tilde{c}_2 & \dots & \tilde{c}_{n-1} & \tilde{c}_n
\end{array}$$

При использовании вложенных методов Рунге–Кутты каждое значение $y(x_0 + h)$ вычисляют дважды: с коэффициентами α_k, β_{ki}, c_k и с коэффициентами α_k, β_{ki}, $\tilde{c}_k$. Если разница полученных при этом значений $y(x_0 + h)$ превышает заданную точность, шаг интегрирования h уменьшается и вычисления выполняются заново. В настоящее время разработано большое количество вложенных методов Рунге–Кутты различных порядков. Названия этих методов связаны с фамилиями разработавших их ученых.

MuPAD позволяет решать обыкновенные дифференциальные уравнения первого порядка следующими методами: методом Эйлера первого

порядка, явным методом Рунге–Кутты 4-го порядка, явным методом Бутчера 6-го порядка, вложенными методами Рунге–Кутты–Фельберга 3-го, 4-го, 5-го, 7-го, 8-го порядков, вложенными методами Дорманда–Принса 4–8-го порядков, вложенными методами Кэша–Карпа 4-го, 5-го порядков, а также неявными методами Гаусса, использующими для численного интегрирования квадратурную формулу Гаусса.

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
1/2 & 1/2 & & & & & & \\
2/3 & 2/9 & 4/9 & & & & & \\
1/3 & 7/36 & 2/9 & -1/12 & & & & \\
5/6 & -35/144 & -55/36 & 35/48 & 15/8 & & & \\
1/6 & -1/360 & -11/36 & -1/8 & 1/2 & 1/10 & & \\
1 & -41/260 & 22/13 & 43/156 & -118/39 & 32/195 & 80/39 & \\
\hline
 & 13/200 & 0 & 11/40 & 11/40 & 4/25 & 4/25 & 13/200
\end{array}
$$

Рис. 8.7. Коэффициенты метода Бутчера 6-го порядка для численного решения обыкновенного дифференциального уравнения первого порядка

С помощью функции `numeric::butcher` можно получить схемы Бутчера для этих методов. Для этого функцию `numeric::butcher` вызывают следующим образом:

`<Результат> :=` **`numeric::butcher`**`(<метод>, <точность>)`

В качестве параметра `<метод>` можно использовать один из следующих стандартных идентификаторов: для метода Эйлера – `EULER1`; для классического метода Рунге–Кутты 4-го порядка – `RK4`; для метода Бутчера 6-го порядка – `BUTCHER6`; для методов Рунге–Кутты–Фельберга – `RKF43` (3-го порядка), `RKF34` (4-го порядка), `RKF54a`, `RKF54b` (4-го порядка), `RKF45a`, `RKF45b` (5-го порядка), `RKF87` (7-го порядка), `RKF78` (8-го порядка); для методов Дорманда–Принса – `DOPRI54` (4-го порядка), `DOPRI45` (5-го порядка), `DOPRI65` (5-го порядка), `DOPRI56` (6-го порядка), `DOPRI87` (7-го порядка), `DOPRI78` (8-го порядка); для методов Кэша–Карпа – `CK54` (4-го порядка), `CK45` (5-го порядка); для метода Гаусса *s*-го порядка – `GAUSS(s)`. Для всех методов, кроме метода Гаусса, коэффициенты, возвращаемые функцией `numeric::butcher`, записываются в рациональном формате, а для метода Гаусса – в десятичном формате. Параметр `<точность>` можно использовать только для метода

Гаусса. Он указывает количество точных значащих цифр, которые следует получить при вычислении коэффициентов этого метода.

Для указанного <метода> функция `numeric::butcher` возвращает список [n, α, β, c, $\tilde{c}$, <порядок1>, <порядок2>]. В этом списке: первый элемент – количество n используемых методом узловых точек; второй элемент – вектор размерностью n, состоящий из значений узловых точек α_k; третий элемент – матрица размером $n \times n$, состоящая из коэффициентов β_{ki}; четвертый и пятый элементы – векторы размерностью n, содержащие коэффициенты c_k и $\tilde{c}$, основной и вложенной квадратурной формулы (если метод не является вложенным, то функция `numeric::butcher` возвращает два одинаковых вектора $\tilde{c}$, $= c_k$); два последних элемента <порядок1> и <порядок2> – это порядок основного и вложенного методов (если указанный метод не является вложенным, то функция `numeric::butcher` возвращает два одинаковых числа, равных порядку метода, <порядок2> = <порядок1>).

Например, в результате выполнения команды `numeric::butcher(RKF43)` в поле вывода появится следующий список:

$$\left[5, \begin{pmatrix} 0 & \frac{1}{4} & \frac{4}{9} & \frac{6}{7} & 1 \end{pmatrix},\right.$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 0 & 0 \\ 4/81 & 32/81 & 0 & 0 & 0 \\ 57/98 & -432/343 & 1053/686 & 0 & 0 \\ 1/6 & 0 & 27/52 & 49/156 & 0 \end{pmatrix},$$

$$\left.\begin{pmatrix} \frac{43}{288} & 0 & \frac{243}{416} & \frac{343}{1872} & \frac{1}{12} \end{pmatrix}, \begin{pmatrix} \frac{1}{6} & 0 & \frac{27}{52} & \frac{49}{156} & 0 \end{pmatrix}, 4, 3\right],$$

который соответствует вложенному методу Рунге–Кутты–Фельберга третьего порядка со схемой Бутчера:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
1/4 & 1/4 & & & & \\
4/9 & 4/81 & 32/81 & & & \\
6/7 & 57/98 & -432/343 & 1053/686 & & \\
1 & 1/6 & 0 & 27/52 & 49/156 & \\
\hline
 & 43/288 & 0 & 243/416 & 343/1872 & 1/12 \\
 & 1/6 & 0 & 27/52 & 49/156 & 0
\end{array}
$$

В этой схеме основной метод имеет порядок 3, а вложенный (используемый для контроля точности вычислений) – порядок 4.

Используя полученные коэффициенты можно самостоятельно запрограммировать соответствующий метод численного решения дифференциального уравнения.

Следует отметить, что каждый из методов численного решения обыкновенных дифференциальных уравнений позволяет получить только одно значение $y_1 = y(x_1)$ искомой функции $y(x)$ в точке $x_1 = x_0 + h$, расположенной на расстоянии h от начальной точки. Далее, считая вычисленное значение y_1 начальными условиями в точке x_1, повторно применяя тот же самый метод, можно получить следующее значение y_2 искомой функции $y(x)$ в точке $x_2 = x_1 + h$ и т.д. Для каждого вычисленного значения y_1, y_2, … шаг h подбирается с учетом заданной точности. Такое последовательное применение численного метода позволяет получить искомую функцию, но не в аналитическом виде, а в виде последовательности точек, в которых вычислены значения искомой функции на заданном интервале. Поэтому при численном решении дифференциальных уравнений для аргумента искомой функции необходимо указывать интервал, на котором требуется вычислять искомые значения.

8.3.2. Применение методов численного решения обыкновенных дифференциальных уравнений

Для численного решения обыкновенных дифференциальных уравнений в MuPAD можно использовать одну из функций: `numeric::odesolve` или `numeric::odesolve2`. Так же как и при аналитическом решении дифференциальных уравнений, для использования этих функций необходимо исходное дифференциальное уравнение (или систему дифференциальных уравнений) вместе с начальными условиями

представить в виде специального векторного объекта. Для корректного преобразования исходного дифференциального уравнения (или системы дифференциальных уравнений) в векторный объект специального вида рекомендуется использовать функцию `numeric::ode2vectorfield` (или `numeric::odeToVectorField`, что то же самое), которую вызывают следующим образом:

```
<Векторный объект> := numeric::ode2vectorfield(
                    {<уравнение>, <начальные условия>},
                                  [<искомая функция>])
<Векторный объект> := numeric::odeToVectorField(
                    {<уравнение>, <начальные условия>},
                                  [<искомая функция>])
```

Для вызова функции `numeric::ode2vectorfield` (или `numeric::odeToVectorField`) необходимо задать два параметра. Дифференциальное `<уравнение>` (или систему уравнений) и его `<начальные условия>` следует указывать в виде множества (в фигурных скобках) или списка (в квадратных скобках) через запятую – первый параметр. Вместо множества (списка) дифференциальных уравнений можно также использовать идентификатор переменной, которой это множество (список) были ранее обозначены. В качестве второго параметра необходимо указывать список (в квадратных скобках) искомых функций и всех их производных, порядок которых ниже порядка соответствующего дифференциального уравнения.

Например, чтобы создать векторный объект для дифференциального уравнения $y'' + 25y = 0$ и его начальных условий $y(0) = 5$, $y'(0) = 20$, можно написать команду `numeric::ode2vectorfield({y''(x) + 25*y(x) = 0, y(0) = 5, y'(0) = 20}, [y(x), y'(x)])`.

Функция `numeric::ode2vectorfield` преобразует все исходные дифференциальные уравнения $y^{(n)} = F(y^{(n-1)}, \ldots, y'', y', y, x)$, порядок которых выше единицы, с помощью замены переменных $y^{(k)} = u_k$ в систему дифференциальных уравнений первого порядка: $u_0' = u_1$, $u_1' = u_2$, …, $u_{n-2}' = u_{n-1}$, $u_{n-1}' = F(u_{n-1}, ..., u_2, u_1, u_0, x)$. Если из искомых функций такой системы составить вектор $Y = (u_0\ u_1\ u_2\ \ldots\ u_{n-1})$, то исходное дифференциальное уравнение (систему уравнений) можно представить в матричном виде как векторное дифференциальное уравнение первого порядка: $Y' = f(x,Y)$. Каждый элемент вектора Y зависит от x, поэтому, чтобы

подчеркнуть эту зависимость, вектор Y называют ***искомой вектор-функцией***. Правая часть $f(x,Y)$ матричного дифференциального уравнения тоже является вектором, каждый элемент которого зависит от x. Поэтому $f(x,Y)$ называют ***вектор-функцией исходного дифференциального уравнения***. Функция `numeric::ode2vectorfield` должна преобразовать исходное дифференциальное уравнение (систему уравнений) в матричное дифференциальное уравнение первого порядка и сформировать начальные условия для него в виде вектора значений искомой вектор-функции Y в некоторой (одной и той же) начальной точке. Поэтому начальные условия для исходных (нематричных) дифференциальных уравнений также должны быть заданы в одной и той же начальной точке.

Функция `numeric::ode2vectorfield` возвращает `<векторный объект>` в виде последовательности, состоящей из трех элементов: 1) вектор-функции $f(x, Y)$ исходного дифференциального уравнения – правой части представленного в матричном виде $Y' = f(x,Y)$ дифференциального уравнения; 2) начальной точки; 3) вектора значений искомой вектор-функций в начальной точке. В поле вывода вектор-функция $f(x, Y)$ исходного дифференциального уравнения отображается в виде функции двух аргументов proc …(t,Y) … end, начальная точка – в виде числового скаляра, а вектор значений искомой вектор-функции в начальной точке – в виде списка. Следует отметить, что в создаваемом функцией `numeric::ode2vectorfield` `<векторном объекте>` аргумент искомых функций всегда обозначается через t.

Например, в результате выполнения команд `ur := {y''(x) + 25*y(x) = 0, y(0) = 5, y'(0) = 20}; UR := numeric::ode2vectorfield(ur, [y(x), y'(x)])` можно получить векторный объект для дифференциального уравнения $y'' + 25y = 0$ и его начальных условий $y(0) = 5$, $y'(0) = 20$ в виде:

proc(t,Y) … end, 0, [5, 20]

Здесь proc(t,Y) … end – краткое обозначение безымянной вектор-функции $f(t, Y)$, с помощью которой задается правая часть матричного дифференциального уравнения $Y' = f(t, Y)$; 0 – точка ($t = 0$), в которой заданы начальные условия; [5, 20] – вектор (список) начальных значений.

Для дальнейшего использования полученный векторный объект удобно преобразовать в список следующим образом:

```
[<Вектор-функция>, <начальная точка>,
                <начальный вектор>] := [<векторный объект>]
```

Если вместо элементов результирующего списка указать разные символьные переменные, то после такого присвоения отпадает необходимость извлекать элементы из векторного объекта с помощью функции `op`. Кроме того, в результате такого присвоения безымянной вектор-функции proc(t,Y) … end будет присвоено имя (идентификатор) переменной `<вектор-функция>`, расположенной в результирующем списке на месте первого элемента. Например, выполнив вслед за последовательностью команд `ur := {y''(x) + 25*y(x) = 0, y(0) = 5, y'(0) = 20}; UR := numeric::ode2vectorfield(ur, [y(x), y'(x)])` команду `[f, t0, Y0] := [UR]`, можно вместо единого векторного объекта `UR` получить три переменные `f`, `t0`, `Y0` именно в том формате, в котором необходимо указывать информацию о дифференциальном уравнении функциям `numeric::odesolve` или `numeric::odesolve2`. Аналогичный результат получится, если элементы векторного объекта `UR` извлечь с помощью функции `op` следующим образом: `f := op(UR, 1); t0 := op(UR, 2); Y0 := op(UR, 3)`.

В MuPAD для численного решения обыкновенных дифференциальных уравнений можно использовать одну из двух функций `numeric::odesolve` или `numeric::odesolve2`, вызов которых осуществляется по похожим правилам:

```
<Результат> := numeric::odesolve(<вектор-функция>,
                <начальная точка> .. <конечная точка>,
                <начальный вектор>, <метод>, <опции>)
<Результат> := numeric::odesolve2(<вектор-функция>,
                <начальная точка>, <начальный вектор>,
                                   <метод>, <опции>)
```

В качестве параметров `<вектор-функция>`, `<начальная точка>`, `<начальный вектор>` необходимо указать элементы полученного с помощью функции `numeric::ode2vectorfield` векторного объекта.

Например, если для некоторого дифференциального уравнения ранее в программе переменными `f`, `t0`, `Y0` обозначены полученные с помощью функции `numeric::ode2vectorfield` вектор-функция, начальная точка и начальный вектор соответственно, то для численного решения этого уравнения можно написать одну из следующих команд: `numeric::odesolve(f, t0..t0+10, Y0)` или `numeric::odesolve2(f, t0, Y0)`.

Отличие функций numeric::odesolve и numeric::odesolve2 состоит в том, что функция numeric::odesolve возвращает одно значение искомой вектор-функции (т.е. по одному значению каждой из искомых функций и их производных) в некоторой фиксированной точке, отличной от начальной, а функция numeric::odesolve2 возвращает искомую вектор-функцию $Y(t)$ в виде функциональной зависимости от ее аргумента t. Поэтому для вызова функции numeric::odesolve нужно обязательно указывать <конечную точку> – числовое значение аргумента искомой вектор-функции, при котором необходимо рассчитать численные значения элементов этой вектор-функции. В приведенном примере при вызове функции numeric::odesolve в качестве параметра <конечная точка> указано t0 + 10. Для вызова функции numeric::odesolve2 конечную точку указывать не нужно. Поэтому при ее вызове вместо параметров <вектор-функция>, <начальная точка>, <начальный вектор> можно указать <векторный объект> целиком в виде, полученном непосредственно функцией numeric::ode2vectorfield, не разделяя его на элементы:

```
<Результат> := numeric::odesolve2(
                    <векторный объект>, <метод>, <опции>)
```

Например, если предварительно присвоена переменная ur := {y''(x) + 25*y(x) = 0, y(0) = 5, y'(0) = 20} и создан векторный объект UR := numeric::ode2vectorfield(ur, [y(x), y'(x)]), то последовательность команд [f, t0, Y0] := [UR]; numeric::odesolve2(f, t0, Y0) даст такой же результат, как команда numeric::odesolve2(UR).

Кроме параметров <вектор-функция>, <начальная точка>, <начальный вектор> (а также параметра <конечная точка> для функции numeric::odesolve) при вызове функций numeric::odesolve или numeric::odesolve2 можно выбрать метод численного решения обыкновенного дифференциального уравнения. Для этого следует указать параметр <метод> с помощью одного из следующих стандартных идентификаторов: EULER1, RK4, BUTCHER6, RKF43, xRKF43, RKF34, xRKF34, RKF54a, xRKF54a, RKF54b, xRKF54b, RKF45a, xRKF45a, RKF45b, xRKF45b, RKF87, xRKF87, RKF78, xRKF78, DOPRI54, xDOPRI54, DOPRI45, xDOPRI45, DOPRI65, xDOPRI65, DOPRI56, xDOPRI56, DOPRI87, xDOPRI87, DOPRI78,

xDOPRI78, CK54, xCK54, CK45, xCK45, GAUSS(s). Эти идентификаторы, используемые в качестве параметра <метод> при вызове функций numeric::odesolve и numeric::odesolve2, соответствуют тем же методам, что и при вызове функции numeric::butcher (см. разд. 8.3.1). Дополнительно идентификаторами, которые начинаются с x (например: xRKF43, xDOPRI78 и т.д.), обозначены методы, которые совпадают с методами, идентификаторы которых не содержат x (например: RKF43, DOPRI78 и т.д.), но при вычислении промежуточных интегралов дополнительно учитывают значения $y(x_0 - h)$ искомой функции, полученные на предыдущем шаге решения дифференциального уравнения. Как правило, этот прием повышает точность метода на один порядок. Если параметр <метод> не указан явным образом, то обе функции numeric::odesolve и numeric::odesolve2 используют вложенный метод Дорманда–Принса 8-го порядка, в котором для вычисления интеграла используется квадратурная формула с 13 узлами, а вложенный метод имеет 7-й порядок (DOPRI78).

Тогда, чтобы численно решить дифференциальное уравнение $y'' = y$ (с начальными условиями $y(0) = -1$, $y'(0) = 2$), используя классический метод Рунге–Кутты 4-го порядка, после создания векторного объекта (ur := {y''(x) = y(x), y(0) = -1, y'(0) = 2}; [f, t0, Y0] := [numeric::ode2vectorfield(ur, [y(x), y'(x)])]) можно написать одну из следующих команд: Y := numeric::odesolve(f, t0..t0+5, Y0, RK4) или Z := numeric::odesolve2(f, t0, Y0, RK4).

При вызове каждой из функций numeric::odesolve и numeric::odesolve2 можно использовать следующие <опции>: RelativeError = δ или AbsoluteError = ε – позволяет задавать относительную (RelativeError) или абсолютную (AbsoluteError) погрешность, которая учитывается при выборе шага интегрирования (если разница значений, полученных основным и вложенным методами, больше указанной погрешности, то шаг интегрирования уменьшается), при использовании каждой из этих <опций> вместо ε или δ необходимо указать положительное вещественное число, по умолчанию значения ε и δ задаются через переменную окружения DIGITS следующим образом: RelativeError = $10^{-\mathrm{DIGITS}}$, AbsoluteError = $10^{-10\cdot\mathrm{DIGITS}}$; опция MaxStepsize = $h_{\max}$ – позволяет указать максимальное значение, которое

может использоваться при выборе шага интегрирования (т.е. при любой погрешности вычислений шаг интегрирования не будет превосходить h_{max}), при использовании этой <опции> вместо h_{max} следует написать положительное вещественное число; опция `Stepsize` = h – позволяет выполнять интегрирование с постоянным шагом (если эта опция задана, автоматический выбор шага отключается), при использовании этой <опции> вместо h также следует указать положительное вещественное число.

Кроме перечисленных выше <опций> функцию `numeric::odesolve` можно вызвать также с <опциями>: `Alldata` = n – позволяет вывести список, в который включено каждое n-е вычисленное функцией `numeric::odesolve` значение элементов вектора Y (если $n = 1$, выводятся все значения, если $n \leq 0$, выводятся только начальное и конечное значения); `Symbolic` – позволяет вывести в аналитическом виде формулы всех вычислений, которые необходимы для реализации одного шага выбранного метода численного решения обыкновенного дифференциального уравнения.

Кроме <опций> `RelativeError`, `AbsoluteError`, `Stepsize`, `MaxStepsize` функцию `numeric::odesolve2` (в отличие от функции `numeric::odesolve`) можно вызвать только с <опцией> `RememberLast`. Эта опция предотвращает запоминание всех результатов, полученных функцией `numeric::odesolve2` при предыдущих вызовах. При использовании этой <опции> MuPAD запоминает только результаты, полученные при самом последнем вызове. Если функция вызывается часто (сотни тысяч раз), использование этой опции может оказать существенное влияние на продолжительность вычислений.

Функция `numeric::odesolve` возвращает значение искомого вектора Y в заданной <конечной точке> в виде списка. При решении одного дифференциального уравнения первого порядка список содержит один элемент – значение искомой функции для аргумента, равного <конечной точке>: $Y = [y]$. При решении одного дифференциального уравнения второго порядка список содержит два элемента – значения искомой функции и ее первой производной для аргумента, равного <конечной точке>: $Y = [y, y']$. При решении системы, состоящей из двух дифференциальных уравнений первого порядка, список также содержит два элемента – значения обеих искомых функций для их аргумента, равного <конечной точке>: $Y = [y, z]$ и т.д. Соответствие между элементами списка,

возвращаемого функцией `numeric::odesolve`, искомыми функциями и их производными устанавливается на основании списка `[<искомая функция>]`, указанного при создании векторного объекта. Например, если при создании векторного объекта с помощью команд `ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}; UR := numeric::ode2vectorfield(ur, [y(x), y'(x), y''(x)])` в качестве списка искомых функций указан `[y(x), y'(x), y''(x)]`, то после выполнения команд `[f, t0, Y0] := [UR]: DIGITS := 3: Y := numeric::odesolve(f, t0 .. PI/sqrt(3), Y0)` в поле вывода появится список [16.6, 15.4, 17.1], первый элемент которого – это $y(x) = 16.6$ (значение искомой функции), второй элемент – $y'(x) = 15.4$, третий элемент – $y''(x) = 17.1$ при $x = \pi/\sqrt{3}$. Если при создании векторного объекта элементы искомого вектора указать в другом порядке, например `[y''(x), y'(x), y(x)]`, то в результате выполнения команд `UR := numeric::ode2vectorfield(ur, [y''(x), y'(x), y(x)]): [f, t0, Y0] := [UR]: Y := numeric::odesolve(f, t0..PI/sqrt(3), Y0)` получится список [17.1, 15.4, 16.6]. Возвращаемый функцией `numeric::odesolve` список можно присвоить списку, состоящему из такого же количества символьных переменных. Тогда каждой символьной переменной будет присвоено соответствующее значение. Например, в результате выполнения следующих команд:

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1};
UR := numeric::ode2vectorfield(ur,
                        [y(x), y'(x), y''(x)]):
[f, t0, Y0] := [UR]: DIGITS := 3:
Y := numeric::odesolve(f, t0..PI/sqrt(3), Y0);
[y0, y1, y2] := Y;
y0, y1, y2
```

переменной `y0` будет присвоено значение 16.6, переменной `y1` – значение 15.4, а переменной `y2` – значение 17.1. Тот же результат можно получить, если вместо двух команд `Y := numeric::odesolve(f, t0 .. PI/sqrt(3), Y0); [y0, y1, y2] := Y` написать одну `[y0, y1, y2] := numeric::odesolve(f, t0..PI/sqrt(3), Y0)`. Такой прием позволяет извлекать значения из списка без использования функции `op`.

В отличие от функции `numeric::odesolve`, которая возвращает решение дифференциального уравнения (системы уравнений) в виде списка числовых значений, функция `numeric::odesolve2` возвращает решение в виде неаналитической зависимости (функции) искомого вектора *Y* от его аргумента *t*. Эта функция в командном поле отображается в виде: proc Y(t) … end. Вместо имени функции (вместо Y) указывается та переменная, которой был присвоен результат вызова функции `numeric::odesolve2`. То есть после выполнения команды `Y := numeric::odesolve2(f, t0, Y0)` создается функция *Y*(*t*), и в поле вывода появится proc Y(t) … end, а после выполнения команды `Z := numeric::odesolve2(f, t0, Y0)` создается функция *Z*(*t*), и в поле вывода появится proc Z(t) … end. В действительности функция `numeric::odesolve2` не решает дифференциальное уравнение, а только организует вызов функции `numeric::odesolve2` для произвольного конечного значения аргумента. То есть выполнение команды `Z := numeric::odesolve2(f, t0, Y0)` – это объявление функции `Z := t -> numeric::odesolve(f, t0..t, Y0)`. Численное решение уравнения начинается только тогда, когда объявленная функция *Z*(*t*) получает конкретное числовое значение аргумента *t*. Функция *Z*(*t*) для фиксированного числового значения аргумента *t* возвращает элементы искомого вектора в виде списка значений искомых функций и их производных в порядке, указанном в списке `[<искомая функция>]` при создании векторного объекта (аналогично функции `numeric::odesolve`). Задавая разные значения аргумента, можно получить значения искомых функций и их производных в разных конечных точках. Если в качестве аргумента функции *Z*(*t*) использовать не заданную ранее символьную переменную, то вычисления будут отложены до тех пор, пока этой символьной переменной не будет присвоено числовое значение. Например, в результате выполнения команд

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}:
UR := numeric::ode2vectorfield(ur,
                        [y(x), y'(x), y''(x)]):
Z := numeric::odesolve2(UR);
DIGITS := 3: Z(PI/sqrt(3)); Z(x)
```

в поле вывода можно получить список значений искомой функции и ее производных в точке $x = \pi/\sqrt{3}$, а также отложенное вычисление значения функции `Z`:

[16.6, 15.4, 17.1]

$Z(x)$

Используя функцию `op`, можно выделить из вектора, возвращаемого функцией $Z(t)$, полученной в результате решения дифференциального уравнения, функциональную зависимость для одной из искомых функций или одной из их производных – скалярную функцию. Например:

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}:
UR := numeric::ode2vectorfield(ur,
                        [y(x), y'(x), y''(x)]):
Z := numeric::odesolve2(UR);
y0 := x -> op(Z(x), 1);        y1 := x -> op(Z(x), 2)
```

В этом примере объявлены две неаналитические скалярные функции: `y0(x)` соответствует решению (искомой функции $y(x)$) исходного дифференциального уравнения $y''' = y$, а `y1(x)` соответствует первой производной $y'(x)$ от искомой функции. В поле вывода, соответствующем объявлению каждой из неаналитических скалярных функций, также отобразятся отложенные вычисления вида `x` → `op(Z(x), 1)`, практически дублирующие введенную команду. Задавая этим функциям конкретные числовые значения аргумента, можно вычислить значения искомой функции и ее производной в разных точках. Например, в результате выполнения следующих команд можно получить значения искомой функции и ее первой производной в точках $x = \pi/\sqrt{3}$ и $x = 2\pi/\sqrt{3}$:

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}:
UR := numeric::ode2vectorfield(ur,
                        [y(x), y'(x), y''(x)]):
Z := numeric::odesolve2(UR):
y0 := x -> op(Z(x), 1):  y1 := x -> op(Z(x), 2):
DIGITS := 3:
y0(PI/sqrt(3)), y0(2*PI/sqrt(3));
y1(PI/sqrt(3)), y1(2*PI/sqrt(3))
```

В поле вывода получатся два значения искомой функции (16.6, 99.9) и два значения ее первой производной (15.4, 100.0) в указанных точках.

Для того же дифференциального уравнения при тех же начальных условиях те же неаналитические зависимости $y(x)$ и $y'(x)$ можно получить, используя вместо функции `numeric::odesolve2` функцию `numeric::odesolve`. Для этого следует написать следующую последовательность команд:

Notebook1* - MuPAD

File Edit View Navigation Insert Format Notebook

```
DIGITS:= 3:
```

Дифференциальное уравнение

```
ur:= {y'''(x)=y(x), y(0)=5, y'(0)=2, y''(0)=1};
UR:= numeric::ode2vectorfield(ur,
                    [y(x), y'(x), y''(x)])
```

$\{y'''(x) = y(x),\ y'(0) = 2,\ y''(0) = 1,\ y(0) = 5\}$

proc(t, Y) ... end, 0, [5, 2, 1]

Численное решение в векторной форме

```
Z:= numeric::odesolve2(UR)
```

proc Z(t) ... end

Неаналитические функции, описывающие численное решение дифференциального уравнения (в скалярной форме): y0 - это y(x), y1 - это y'(x), y2 - это y''(x)

```
y0:= x -> op(Z(x), 1);
y1:= x -> op(Z(x), 2);
y2:= x -> op(Z(x), 3)
```

$x \to \mathrm{op}(Z(x), 1)$

$x \to \mathrm{op}(Z(x), 2)$

$x \to \mathrm{op}(Z(x), 3)$

Значения y(x), y'(x) и y''(x) в точках x=pi/sqrt(3) и x=2pi/sqrt(3)

```
y0(PI/sqrt(3)), y0(2*PI/sqrt(3))
```

16.6, 99.9

```
y1(PI/sqrt(3)), y1(2*PI/sqrt(3))
```

15.4, 100.0

```
y2(PI/sqrt(3)), y2(2*PI/sqrt(3))
```

17.1, 101.0

Mem 11 MB, T 0 s Cmd INS

Рис. 8.8. Численное решение обыкновенного дифференциального уравнения

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}:
[f, x0, Y0] := [numeric::ode2vectorfield(ur,
                        [y(x), y'(x), y''(x)])]:
Z := x -> numeric::odesolve(f, x0..x, Y0):
y0 := x -> op(Z(x), 1):  y1 := x -> op(Z(x), 2):
```

Объявленные таким образом неаналитические скалярные функции `y0(x)`, `y1(x)` и т.д. позволяют вычислять значения искомой функции и ее производных, но не могут использоваться для создания графического объекта с помощью функции `plot::Function2d` при построении графиков этих функций (см. следующий раздел). Пример численного решения обыкновенного дифференциального уравнения третьего порядка приведен на рис. 8.8.

В MuPAD обращаться к элементам списка можно так же, как к элементам вектора, указав в квадратных скобках соответствующий номер элемента. Например, если переменной `Y` присвоен список [16.6, 15.4, 17.1], то выбрать второй элемент списка можно либо с помощью команды `op(Y, 2)`, либо с помощью команды `Y[2]`. В обоих случаях получится одинаковый результат: 15.4. Поэтому во всех приведенных выше примерах функцию `op` можно заменить на соответствующий номер элемента списка, указанный в квадратных скобках. Например, для объявления скалярной функции `y0(x)` можно вместо команды `y0 := x -> op(Z(x), 1)` написать команду `y0 := x -> Z(x)[1]`.

При вызове любой несимвольной скалярной функции с конкретным значением аргумента каждая из них вычисляет в указанной точке все элементы вектора, а затем извлекает нужный. Учитывая, что MuPAD помнит результаты сделанных ранее вычислений пока не будет выполнена очистка памяти, практически выполнять вычисление всех элементов вектора будет только одна из объявленных неаналитических скалярных функций, вызванная первой. Остальные неаналитические скалярные функции, вызванные с тем же аргументом, находят в памяти MuPAD вычисленный ранее вектор и извлекают из него соответствующий элемент (повторное вычисление элементов вектора для одного и того же аргумента не выполняется).

8.3.3. Графическое представление численного решения обыкновенного дифференциального уравнения

Как было показано ранее, в результате численного решения обыкновенного дифференциального уравнения (системы уравнений) с помощью функции `numeric::odesolve2` (или `numeric::odesolve`) можно получить неаналитическую вектор-функцию, которая позволяет рассчитать числовые значения искомой функции (функций) и ее (их) производных, соответствующие некоторому фиксированному значению аргумента. Несмотря на то, что из этой вектор-функции можно создать соответствующее количество скалярных функций, каждая из которых описывает только одну зависимость (искомую функцию или одну из ее производных), эти скалярные функции также являются неаналитическими. Поэтому использовать такие функции для создания графического объекта типа «двумерный график» с помощью функции `plot::Function2d` нельзя. Но так как полученные при решении дифференциального уравнения (системы уравнений) неаналитические функции позволяют вычислять их значения в отдельных точках, можно для такого решения создать графический объект типа «набор точек» с помощью функции `plot::PointList2d`. Для этого нужно сначала создать набор точек в виде списка, каждый элемент которого содержит *x*- и *y*-координаты одной точки в виде `[x, y]` (см. разд. 3.3). Для создания такого списка можно использовать оператор `$`, в качестве параметра которого следует использовать аргумент функции, являющейся решением дифференциального уравнения (системы уравнений). Например, если ранее была объявлена неаналитическая скалярная функция `y0 := x -> op(Z(x), 1)`, где `Z(x)` – вектор-функция полного решения дифференциального уравнения, то для создания набора точек можно написать команду: `XY := [[x, y0(x)] $ x=0..6 step 0.25]`. Здесь `x` (аргумент функции, являющейся решением дифференциального уравнения) выбран изменяющимся на интервале [0, 6] с шагом 0,25. Полученный таким образом набор точек `XY` можно использовать для создания графического объекта с помощью функции `plot::PointList2d` следующим образом: `plot::PointList2d(XY)`. Пример графического представления решения дифференциального уравнения в виде набора точек приведен на рис. 8.9.

Аналогичным образом можно построить графики производных искомой функции. При большом количестве точек и малом их размере такие графики могут выглядеть как графики непрерывных функций. Представление решения дифференциального уравнения в виде дискретного набора точек особенно эффективно при графическом сравнении численного и аналитического решениий. Для этого необходимо на одном графике изобразить аналитическое решение (см. разд. 8.1.5) и дискретный набор точек, соответствующий численному решению одного и того же дифференциального уравнения. Такое же сравнение можно выполнить и в фазовой плоскости. В этом случае в качестве координат точек следует использовать значения искомой функции и значения ее производной. Например, если `Z` – вектор-функция решения дифференциального уравнения, `y0 := x -> op(Z(x), 1)` – искомая функция, `y1 := x -> op(Z(x), 2)` – производная искомой функции, то `YY1 := [[y0(x), y1(x)] $ x=0.. 6 step 0.25]` – набор точек в фазовой плоскости. Так как результаты выполненных ранее вычислений хранятся в памяти MuPAD, пока не будет выполнена очистка памяти с помощью команды `reset()`, для каждого значения аргумента элементы вектор-функции `Z(x)` вычисляются только один раз, например только при вызове функции `y0(x)`. А при вызове функции `y1(x)` для того же аргумента MuPAD находит вычисленное ранее значение вектор-функции `Z(x)` и извлекает из него второй элемент. Поэтому тот же набор точек в фазовой плоскости можно получить, например, следующим образом: `y01 := x -> [op(Z(x), 1), op(Z(x), 2)]; YY1 := [y01(x) $ x=0..6 step 0.25]`.

Далее для полученного одним из перечисленных выше способов набора точек следует создать графический объект (`plot::PointList2d(YY1)`) и визуализировать его с помощью функции `plot`.

Кроме того, для графического представления численного решения обыкновенных дифференциальных уравнений в MuPAD существует функция `plot::Ode2d`, которая создает соответствующий двумерный графический объект, для визуализации которого (как и для визуализации других двумерных графических объектов) необходимо использовать функцию `plot`. Для создания двумерного графического объекта функцию `plot::Ode2d` вызывают следующим образом:

```
<Результат> := plot::Ode2d(<вектор-функция>,
                [<список точек>], <начальный вектор>,
                   [<кривая1>], [<кривая2>], …,
          <метод>, <опции решения>, <графические опции>)
```

Функция `plot::Ode2d` вызывает функцию `numeric::odesolve` для решения дифференциального уравнения в нескольких точках и на основе этих данных создает двумерный графический объект. Часть параметров функции `plot::Ode2d` используется для управления численным решением дифференциального уравнения (системы уравнений), а остальные – для создания графического объекта. Поэтому в качестве первой группы параметров для этой функции необходимо указать не решение, а исходное дифференциальное уравнение (систему уравнений) в векторной форме. То есть в качестве параметров `<вектор-функция>`, `<начальный вектор>` необходимо указать элементы полученного с помощью функции `numeric::ode2vectorfield` векторного объекта для обыкновенного дифференциального уравнения (системы уравнений), решение которого необходимо изобразить графически, а именно `<вектор-функция>` – это правая часть дифференциального уравнения (системы уравнений), приведенного к матричному виду $Y' = f(t,Y)$; `<начальный вектор>` – вектор, состоящий из известных значений искомых функций и их производных в начальной точке (см. разд. 8.3.2). При использовании функции `numeric::ode2vectorfield` для подготовки этих параметров следует помнить, что искомая вектор-функция, состоящая из значений функции и ее производных, всегда обозначается переменной `Y`, а аргумент искомой вектор-функции – переменной `t`. Если параметры `<вектор-функция>`, `<начальный вектор>` формируются без использования функции `numeric::ode2vectorfield`, то искомый вектор и аргумент искомой функции могут быть обозначены другими переменными. Эти обозначения следует учитывать при формировании остальных параметров функции `plot::Ode2d`.

Для построения графического объекта функция `plot::Ode2d` должна вычислить элементы искомой вектор-функции (искомые функции и их производные) для нескольких значений аргумента. Поэтому при вызове функции `plot::Ode2d` в качестве параметра `<список точек>` необходимо указать в квадратных скобках список значений аргументов, для каждого из которых функции `plot::Ode2d` предстоит найти решение дифференциального уравнения. Решение дифференциального уравнения

Notebook1* - MuPAD

File Edit View Navigation Insert Format Notebook

```
DIGITS:= 3:
```

Дифференциальное уравнение

```
ur:= {y'''(x)=y(x), y(0)=5, y'(0)=2, y''(0)=1};
UR:= numeric::ode2vectorfield(ur,
                    [y(x), y'(x), y''(x)])
```

$\{y'''(x) = y(x),\ y'(0) = 2,\ y''(0) = 1,\ y(0) = 5\}$

proc(t, Y) ... end, 0, [5, 2, 1]

Численное решение в векторной форме

```
Z:= numeric::odesolve2(UR)
```

proc Z(t) ... end

y0 - искомая функция y(x) в скалярной форме

```
y0:= x -> op(Z(x), 1)
```

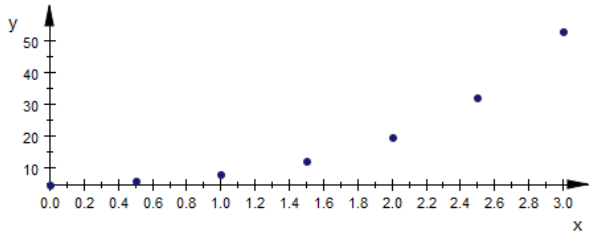
$x \to \mathrm{op}(Z(x), 1)$

Набор точек графика искомой функции

```
XY:= [[x,y0(x)] $ x=0..3 step 0.5]
```

[[0.0, 5.0], [0.5, 6.23], [1.0, 8.43], [1.5, 12.5], [2.0, 19.8], [2.5, 32.2], [3.0, 53.2]]

```
p1:= plot::PointList2d(XY):
plot(p1, Height=50*unit::mm)
```



Mem 11 MB, T 0 s Cmd INS

Рис. 8.9. Графическое представление численного решения обыкновенного дифференциального уравнения в виде дискретного набора точек

в предыдущей точке используется как начальное значение при поиске решения в последующей точке. Первой в списке должна быть указана <начальная точка>, в которой заданы начальные условия (<начальный вектор>) для решаемого дифференциального уравнения. <Список точек> можно сформировать различными способами:

1) перечислив все значения через запятую: [t0, t1, t2, …];

2) используя оператор $ следующим образом:
[t $ t = <начальная точка> .. <конечная точка> step <шаг>]
или [$ <начальная точка> .. <конечная точка> step <шаг>];

3) создав список автоматически следующим образом:
[Automatic, <начальная точка>, <конечная точка>, <шаг>].

Например, если для решаемого дифференциального уравнения начальные условия заданы в точке $t = 2$ и необходимо построить график решения на интервале [2, 6] с шагом 0,5, то <список точек> можно сформировать следующим образом: [2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6], или [t $ t = 2..6 step 0.5], или [Automatic, 2, 6, 0.5].

При вызове функции plot::Ode2d обязательными являются только три параметра: <вектор-функция>, <список точек> и <начальный вектор>. Для построения графика решения дифференциального уравнения $4y' = y$ (с начальным условием $y(2) = 3$) на интервале [2, 6], например, можно выполнить следующие команды: 1) записать уравнение вместе с начальными условиями ur := {4*y'(x) = y(x), y(2) = 3}; 2) создать для него векторный объект [f, t0, Y0] := [numeric::ode2vectorfield(ur, [y(x)])]; 3) используя векторный объект, создать графический объект gr := plot::Ode2d(f, [t $ t=2..6 step 0.5], Y0); 4) визуализировать этот графический объект plot(gr).

Параметр <метод> можно задать так же, как при вызове функции numeric::odesolve, с помощью одного из стандартных идентификаторов, перечисленных в разд. 8.3.2. Так же как при вызове функции numeric::odesolve, если параметр <метод> не указан, то по умолчанию для решения дифференциального уравнения используется метод Дорманда–Принса 8-го порядка (DOPRI78).

В качестве <опций решения> можно использовать следующие <опции> функции numeric::odesolve: RelativeError = δ, Abso-

luteError = ε или Stepsize = *h* (см. описание <опций> функции numeric::odesolve в разд. 8.3.2).

Таблица 8.2

Основные графические опции функции plot::Ode2d

Название	Описание	Значение по умолчанию
AntiAliased	Сглаживать линии и точки?	TRUE
Colors	Список цветов	[RGB::Blue, …, RGB::Black]
Frames	Количество кадров в анимации	50
LegendText	Текст для легенды	""
LegendEntry	Добавлять описание этого объекта в легенду	FALSE
LineWidth	Толщина линий	0.35
LineStyle	Стиль линий (Solid – сплошная, Dashed – штрихами, Dotted – пунктиром)	Solid
LinesVisible	Видимые линии?	TRUE
PointSize	Размер точек	1.5
PointStyle	Стиль точек (○ Circles, ● FilledCircles, + Crosses, × XCrosses, ◊ Diamonds, ♦ FilledDiamonds, □ Squares, ■ FilledSquares, ✳ Stars)	FilledCircles
PointsVisible	Видимые точки?	TRUE
TimeRange	Промежуток времени для анимации	0.0 .. 10.0
Visible	Видимый?	TRUE

<Графические опции>, которые можно использовать при вызове функции plot::Ode2d, приведены в табл. 8.2. Каждую из этих опций следует задавать в виде: <название> = <значение>. Опция Colors по умолчанию представляет собой следующий список цветов: Colors = [RGB::Blue, RGB::Red, RGB::Green, RGB::MuPADGold, RGB::Orange, RGB::Cyan, RGB::Magenta, RGB::LimeGreen, RGB::CadmiumYellowLight, RGB::AlizarinCrimson, RGB:: Aqua, RGB::Lavender, RGB::SeaGreen, RGB::AureolineYellow, RGB::Banana, RGB::Beige, RGB::YellowGreen, RGB:: Wheat,

RGB::IndianRed, RGB::Black]. Значения по умолчанию остальных <графических опций> приведены в табл. 8.2.

Параметры <метод>, <опции решения>, <графические опции> указывают только в том случае, если необходимо изменить какие-то настройки, заданные по умолчанию.

Например, с помощью опции PointsVisible = FALSE можно не выделять на графике точки, в которых вычислены значения искомых функций и их производных. В этом случае для создания графического объекта можно написать команду plot::Ode2d(f, [t $ t=0..6], Y0, PointsVisible = FALSE).

С помощью параметров [<кривая1>], [<кривая2>], … можно управлять количеством двумерных кривых, которые будут включены в создаваемый с помощью функции plot::Ode2d графический объект. Если ни один из параметров [<кривая1>], [<кривая2>], … не указан, то в графический объект по умолчанию включается столько кривых, сколько элементов содержит искомая вектор-функция: искомая функция (функции) и все ее (их) производные, значения которых вычисляются в процессе решения дифференциального уравнения (системы уравнений). При этом по оси ординат откладываются значения искомой функции (функций) и ее (их) производных в каждой из точек, указанных в <списке точек>, а по оси абсцисс – аргумент искомой функции (функций). По умолчанию точки каждой кривой соединяются сплайнами.

Чтобы графический объект содержал не все, а только некоторые кривые выборочно, необходимо использовать параметры [<кривая1>], [<кривая2>], … каждый из которых следует задавать следующим образом:

[<Функция>, Style = <стиль>, Color = <цвет>]

Если при вызове функции plot::Ode2d параметры [<кривая1>], [<кривая2>], … отсутствуют, цвет каждой кривой, включенной в создаваемый графический объект, последовательно выбирается из списка, заданного с помощью <графической опции> Colors. Если необходимо, чтобы описываемая с помощью параметра [<кривая>] также использовала очередной цвет из списка Colors, то параметр Color = <цвет> при описании кривой можно опустить. Если необходимо, чтобы описываемая с помощью параметра [<кривая>] имела цвет, отличный от очередного цвета из списка Colors, то соответствующее значение цвета следует задать при описании кривой в виде Color = <цвет>, где

<цвет> – это либо стандартный идентификатор цвета (см. табл. 3.2), либо значение в RGB-формате, записанное в виде списка, состоящего из трех десятичных чисел от 0 до 1 каждое. Например, `Color = RGB::Red` или `Color = [0.2, 0.1, 0.4]`.

Параметр `Style =` <стиль> позволяет задать стиль линии описываемой кривой. В качестве значения этого параметра можно использовать следующие идентификаторы: `Points` – изображаются только точки, соответствующие вычисленным при решении дифференциального уравнения значениям; `Lines` – точки, соответствующие вычисленным при решении дифференциального уравнения значениям, не изображаются, но соединяются отрезками прямой; `Splines` – точки не изображаются, но соединяются сплайнами; `[Splines, Points]` – точки изображаются и соединяются сплайнами; `[Lines, Points]` – точки изображаются и соединяются отрезками прямой. По умолчанию используется стиль `Style = [Splines, Points]`. Если нет необходимости использовать другой стиль кривой, то параметр `Style =` <стиль> можно опустить (не указывать).

Разные кривые могут иметь не только разные цвета, но и разные стили.

При описании `[`<кривой>`]` единственным обязательным параметром является <функция>, с помощью которой выбирают определенную скалярную функциональную зависимость для добавления в графический объект. Если для создания векторного объекта исходного дифференциального уравнения (системы уравнений) использовалась функция `numeric::ode2vectorfield`, то по умолчанию для обозначения искомой вектор-функции и ее аргумента используются переменные `Y` и `t` соответственно, тогда параметр <функция> при описании `[`<кривых>`]` следует задавать в виде:

`(t, Y)` **->** `[`<x-координата>`,` <y-координата>`]`

Вместо <x-координаты> и <y-координаты> необходимо записать выражения, в которых можно использовать переменную `t` и элементы вектора `Y`. Например, если параметр <функция> задан в виде `(t, Y) -> [t, op(Y, 1)]`, то графический объект будет содержать кривую, описывающую зависимость первого элемента вектор-функции `Y` от ее аргумента `t`. Чаще всего в качестве первого элемента вектор-функции `Y` указывают саму искомую функцию, тогда использование параметра `[(t, Y) -> [t, op(Y, 1)]]` при вызове функции `plot::Ode2d` приведет к построению графика искомой функции. Если в качестве

<y-координаты> записать другой элемент искомой вектор-функции Y (например, [(t, Y) -> [t, op(Y, 2)]]), то можно получить либо график производной искомой функции (для дифференциального уравнения выше первого порядка), либо график другой искомой функции (для системы дифференциальных уравнений).

Например, если для дифференциального уравнения $y''' = y$ при начальных условиях $y(0) = 5$, $y'(0) = 2$, $y''(0) = 1$ с помощью функции numeric::ode2vectorfield создан векторный объект, сохраненный в виде списка [f, t0, Y0] следующим образом:

```
ur := {y'''(x) = y(x), y(0) = 5, y'(0) = 2, y''(0) = 1}:
[f, t0, Y0] := [numeric::ode2vectorfield(ur,
                          [y(x), y'(x), y''(x)])]:
```

то, чтобы создать графический объект, включающий графики $y(x)$, $y'(x)$ и $y''(x)$ на интервале $x \in [0, 1.5]$, достаточно написать команду без использования параметров [<кривая1>], [<кривая2>] и т.д. (рис. 8.10):

```
plot::Ode2d(f, [t $ t=0..1.5 step 0.1], Y0)
```

Чтобы для того же дифференциального уравнения создать графический объект, включающий только один график $y(x)$ на том же интервале, необходимо при вызове функции plot::Ode2d указать одну кривую в виде [(t, Y) -> [t, op(Y, 1)]], так как при создании векторного объекта дифференциального уравнения искомая функция $y(x)$ указана в качестве первого элемента в списке искомых функций. То есть функцию plot::Ode2d можно вызвать следующим образом:

```
plot::Ode2d(f, [t $ t = 0..1.5 step 0.1], Y0,
                              [(t, Y) -> [t, op(Y, 1)]])
```

Аналогично, чтобы для того же дифференциального уравнения создать графический объект, включающий графики $y(x)$ и $y'(x)$ на том же интервале, необходимо при вызове функции plot::Ode2d указать две кривые в виде [(t, Y) -> [t, op(Y, 1)]] и [(t, Y) -> [t, op(Y, 2)]], так как при создании векторного объекта дифференциального уравнения искомая функция $y(x)$ указана в качестве первого элемента, а $y'(x)$ – в качестве второго элемента в списке искомых функций. То есть функцию plot::Ode2d можно вызвать следующим образом (рис. 8.11):

```
plot::Ode2d(f, [t $ t = 0..1.5 step 0.1], Y0,
   [(t, Y) -> [t, op(Y, 1)]], [(t, Y) -> [t, op(Y, 1)]])
```

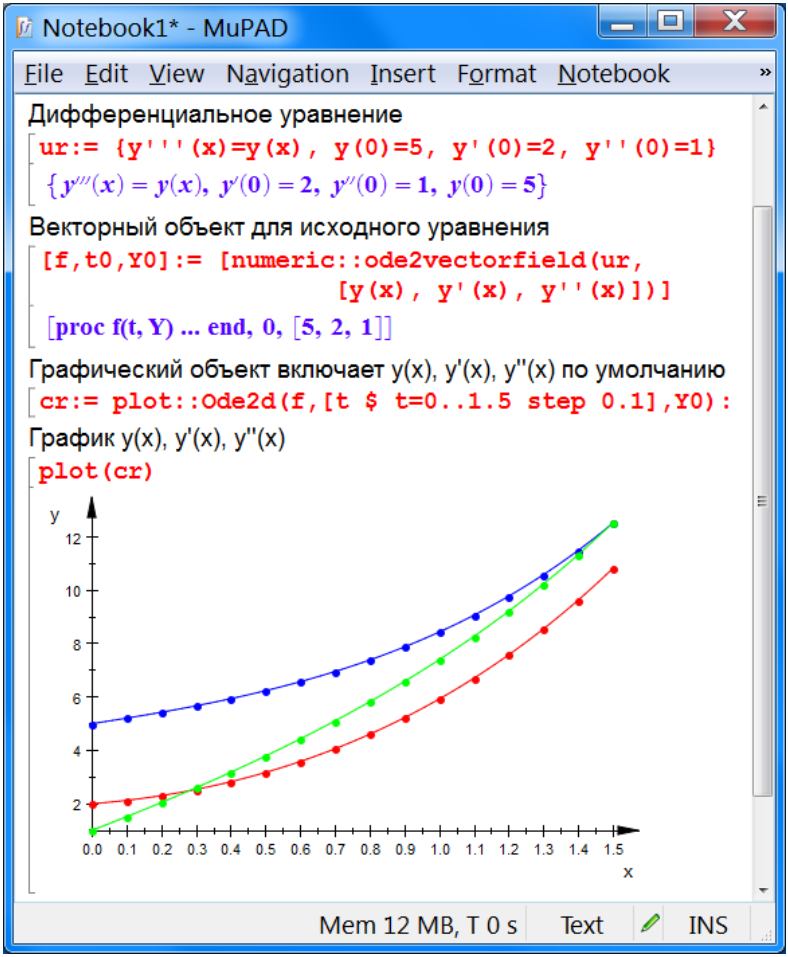


Рис. 8.10. Построение графиков $y(x)$, $y'(x)$, $y''(x)$ для численного решения обыкновенного дифференциального уравнения третьего порядка с использование функции `plot::Ode2d`

При описании `[<кривых>]` элементы искомой вектор-функции `Y` можно использовать не только в качестве `<y-координаты>`, но и в качестве `<x-координаты>`. Тогда можно построить, например, график

решения в фазовой плоскости ($y(x)$, $y'(x)$). Для этого при вызове функции `plot::Ode2d` необходимо указать кривую в виде `[(t, Y) -> [op(Y, 1), op(Y, 2)]]` ($y(x)$ – первый элемент, а $y'(x)$ – второй элемент в списке искомых функций). То есть для того же дифференциального уравнения функцию `plot::Ode2d` можно вызвать следующим образом (рис. 8.12):

```
plot::Ode2d(f, [t $ t = 0..1.5 step 0.1], Y0,
                        [(t, Y) -> [op(Y, 1), op(Y, 2)]])
```

Так как для обращения к элементам списка вместо функции `op` можно использовать номер элемента, заключенный в квадратные скобки, то для получения того же результата при описании параметрической кривой вместо `[(t, Y) -> [op(Y, 1), op(Y, 2)]]` можно написать `[(t, Y) –> [Y[1], Y[2]]]`, а при описании первой кривой в предыдущем примере вместо `[(t, Y) -> [t, op(Y, 1)]]` можно, соответственно, написать `[(t, Y) -> [t, Y[1]]]` и т.д.

В приведенных примерах цвет и стиль кривых, метод и погрешность решения дифференциального уравнения заданы по умолчанию и при необходимости могут быть изменены.

Описывать `[<кривые>]` можно также, используя идентификаторы объявленных ранее функций, записанных по тем же правилам.

В процессе численного решения дифференциального уравнения функция `plot::Ode2d` для каждой кривой создает дискретный набор точек. Количество точек в каждом наборе совпадает с длиной `<списка точек>`, указанного при вызове функции `plot::Ode2d`. В соответствии со стилем каждой кривой при добавлении к графическому объекту эти точки либо соединяются прямыми, либо соединяются сплайнами, либо остаются не соединенными.

Аналогично другим функциям MuPAD, предназначенным для создания графических объектов, функция `plot::Ode2d` позволяет создавать и анимированные графики. Для этого в одном из параметров функции `plot::Ode2d` необходимо использовать дополнительную переменную и указать границы ее изменения (см. разд. 3.6).

Следует помнить, что для визуализации графического объекта, полученного с помощью функции `plot::Ode2d`, необходимо использовать функцию `plot`.

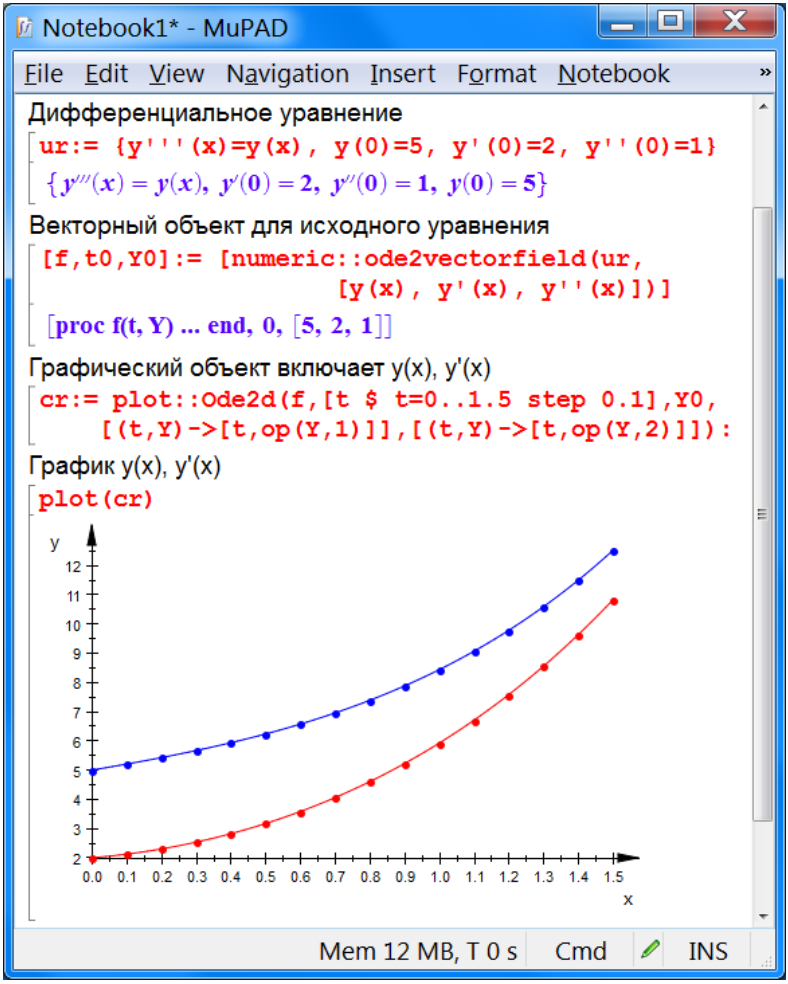


Рис. 8.11. Построение графиков $y(x)$, $y'(x)$ для численного решения обыкновенного дифференциального уравнения третьего порядка с использованием функции `plot::Ode2d`

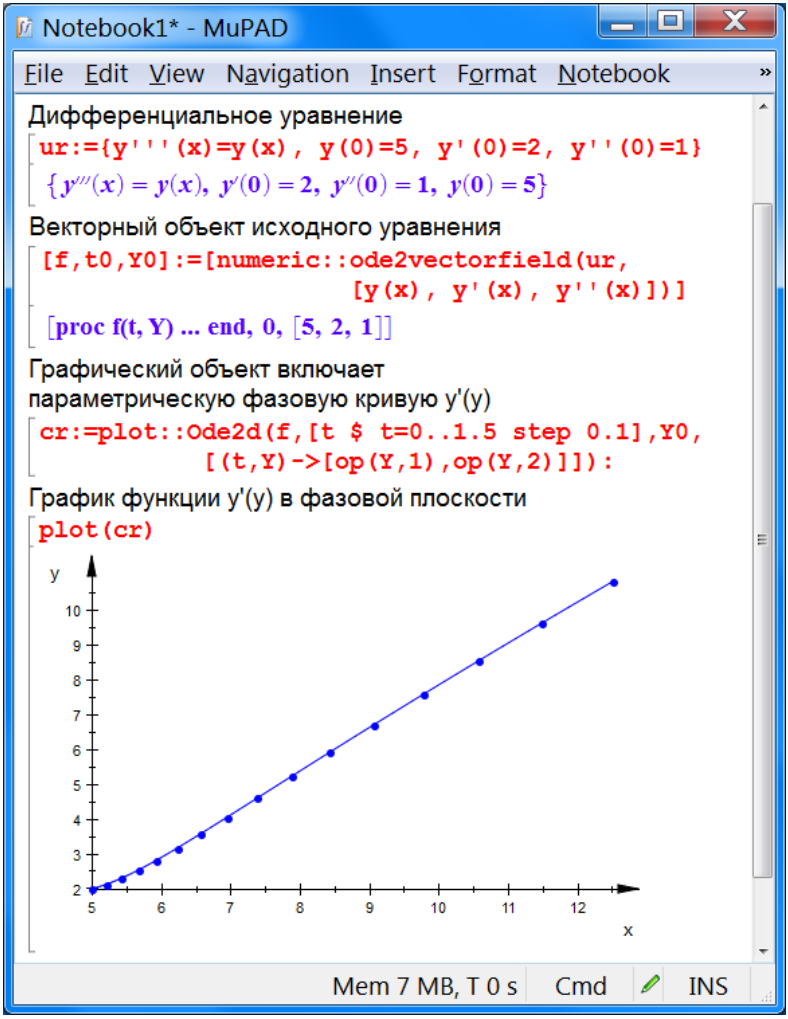


Рис. 8.12. Построение графика численного решения обыкновенного дифференциального уравнения третьего порядка в фазовой плоскости с использованием функции `plot::Ode2d`

Примеры построения графиков численного решения обыкновенного дифференциального уравнения с помощью функции `plot::Ode2d` приведены на рис. 8.10–8.12.

Для создания трехмерного графического объекта, изображающего численное решение обыкновенного дифференциального уравнения (системы уравнений), в MuPAD существует функция `plot::Ode3d`, вызов которой осуществляется аналогично вызову функции `plot::Ode2d`. Детально с правилами использования функции `plot::Ode3d` можно ознакомиться в справочной системе MuPAD.

Глава 9. ИНТЕРПОЛЯЦИЯ И ПОЛИНОМИАЛЬНОЕ ПРЕДСТАВЛЕНИЕ ФУНКЦИЙ

При решении обыкновенных дифференциальных уравнений поиск аналитического вида неизвестной функции одной переменной осуществляется на основе известной аналитической зависимости между искомой функцией, ее производными и аргументом. Другой подход к решению задачи о нахождении аналитического вида неизвестной функции одной переменной используется, если для конечного дискретного набора значений аргумента x_1, x_2, x_3, …, x_n искомой функции известны только ее числовые значения y_1, y_2, y_3, …, y_n (а иногда и числовые значения ее производных). При такой постановке задачи точки x_1, x_2, x_3, …, x_n, в которых известны числовые значения искомой функции, называются ***узлами***. Если известные значения y_1, y_2, y_3, …, y_n искомой функции в узлах являются точными (без погрешности), то считается, что необходимо найти такую функцию, значения которой в тех же узлах будут точно совпадать с этими известными значениями. Если полученную в результате таких поисков функцию планируют использовать внутри интервала, на котором расположены узлы, то задачу о нахождении такой функции принято называть ***интерполяцией***. Если искомую функцию планируют использовать вне интервала, на котором расположены узлы, то задачу о нахождении такой функции принято называть ***экстраполяцией***. Если известные значения искомой функции в узлах обладают погрешностью, то искомая функция в узлах может отличаться от них, но должна проходить вблизи этих значений. Задачу о нахождении такой функции называют ***аппроксимацией***. Чаще всего экстраполяцию рассматривают как расширение области применения интерполяции или аппроксимации. Для решения задач аппроксимации в MatLab существует отдельное приложение: Curve Fitting (см. разд. 11.4.9). Поэтому MuPAD позволяет решать только задачи интерполяции.

9.1. Интерполяция Лагранжа

Полученную в результате решения задачи интерполяции функцию принято называть интерполяционной. Интерполяционную функцию обычно

ищут в виде линейной комбинации полиномов того или иного вида. Если для нахождения искомой аналитической функции используют линейную комбинацию многочленов Лагранжа

$$L_{n-1}(x,k)=\frac{(x-x_1)}{(x_k-x_1)}\cdot\ldots\cdot\frac{(x-x_{k-1})}{(x_k-x_{k-1})}\cdot\frac{(x-x_{k+1})}{(x_k-x_{k+1})}\cdot\ldots\cdot\frac{(x-x_n)}{(x_k-x_n)}=\prod_{\substack{i=1\\ i\neq k}}^{n}\frac{x-x_i}{x_k-x_i},$$

то интерполяционная функция имеет вид

$$F(x)=\sum_{k=1}^{n}y_kL_{n-1}(x,k)\text{, где } n \text{ – количество узлов.}$$

Эта интерполяционная функция является полином $(n-1)$-й степени. Метод построения такой интерполяционной функции называется интерполяцией Лагранжа. Для применения этого метода достаточно, чтобы было известно не меньше двух значений искомой функции в отличных друг от друга точках (узлах). Никаких других ограничений ни на вид функции, ни на расположение узлов этот метод не накладывает.

В MuPAD для построения интерполяционной функции (полинома) методом Лагранжа можно использовать функцию `interpolate`, которую вызывают следующим образом:

```
<Результат> := interpolate(<список узлов>,
                           <список значений>, <аргумент>)
```

В качестве параметра `<список узлов>` необходимо через запятую в квадратных скобках перечислить абсциссы точек, через которые должна проходить искомая функция. В качестве параметра `<список значений>` следует также через запятую и в квадратных скобках указать ординаты этих точек. Оба списка должны содержать одинаковое количество элементов. В `<списке узлов>` не допускается наличие одинаковых значений. Ординаты каждого узла должны располагаться в `<списке значений>` на том же месте, на котором располагается его абсцисса в `<списке узлов>`. В качестве параметров `<список узлов>` и `<список значений>` можно также использовать идентификаторы переменных, которым ранее были присвоены соответствующие списки. Элементы обоих списков могут являться числовыми значениями (или арифметическими выражениями, результатом вычисления которых являются числовые значения), а также не заданными ранее символьными переменными или выражениями.

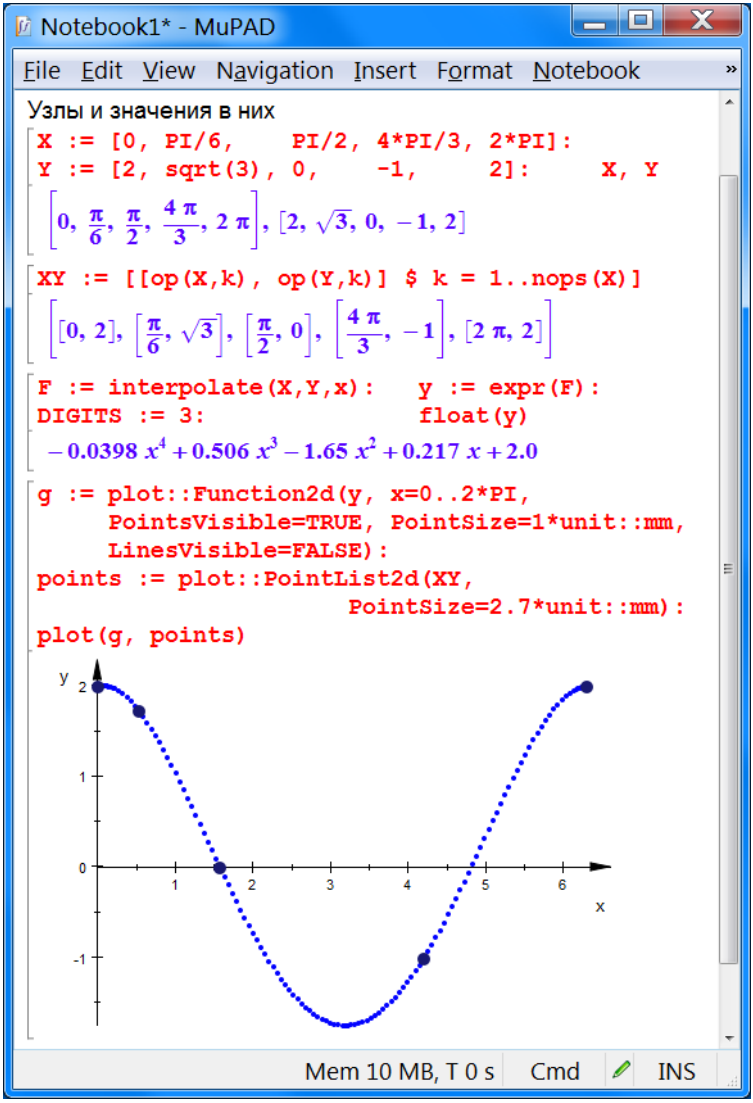


Рис. 9.1. Интерполяция Лагранжа

В качестве параметра <аргумент> необходимо указать идентификатор символьной переменной, которая будет являться аргументом искомой интерполяционной функции.

В качестве <результата> функция `interpolate` возвращает полином poly, степень которого на единицу меньше длины <списка узлов>. В поле вывода этот полином отображается в виде:

poly(<выражение>, [<переменная>]),

где <выражение> – арифметическое выражение полиномиального типа относительно указанной <переменной>.

Несмотря на то, что полином poly записывается в поле вывода с помощью <выражения>, в действительности он представляет собой объект, состоящий из арифметического <выражения>, идентификатора <переменной> и типа этого <выражения>. Например, если с помощью команды `p := interpolate([1, 2, 3], [4, 9, 6], x)` был получен полином `p`, который в поле вывода отображается в виде: poly($-4x^2 + 17x - 9$, [x]), то с помощью команды `op(p)` можно получить все элементы этого полинома poly в виде такой последовательности: $-4x^2 + 17x - 9$, [x], Expr.

Если для дальнейшего использования полученный полином необходимо преобразовать в арифметическое выражение, это можно осуществить с помощью функции `expr`. Например, результатом выполнения команд `p := interpolate([1, 2, 3], [4, 9, 6], x): expr(p)` будет арифметическое выражение $-4x^2 + 17x - 9$, с которым в дальнейшем можно выполнять различные манипуляции (вычисления, построение графиков и т.д.). Пример программы для построения графика интерполяционной функции Лагранжа приведен на рис. 9.1. На этом рисунке точками увеличенного размера отмечены узлы интерполяции.

При большом количестве узлов построение интерполяционной зависимости с помощью функции `interpolate` может занять много времени, так как степень интерполяционного полинома пропорциональна количеству узлов, а коэффициенты при различных степенях этого полинома вычисляются аналитически.

9.2. Кубические сплайны

В результате использования функции `interpolate` строится одна непрерывная интерполяционная функция, которая проходит через все

известные узлы. Чтобы по известным значениям в узлах построить гладкую кусочно-непрерывную функцию, используют кубические сплайны. ***Кубические сплайны*** – это численный метод построения интерполяционной функции, при котором между каждыми двумя соседними узлами строится полином третьей степени так, чтобы в каждом узле полиномы, построенные справа и слева от него, имели одинаковую производную. Благодаря этому искомая интерполяционная функция является гладкой не только между узлами, но и в узлах. В большинстве современных программных средств кубические сплайны используются по умолчанию для построения графика функции между заданными точками.

Для получения интерполяционной функции в виде кубического сплайна в MuPAD можно использовать функцию `numeric::cubicSpline`, которую вызывают одним из следующих способов:

<Имя> := **numeric::cubicSpline**([<список узлов>, <список значений>], <опции>)

<Имя> := **numeric::cubicSpline**(<точка1>, <точка2>, …, <опции>)

Параметры <список узлов> и <список значений> имеют тот же смысл, как и при использовании функции `interpolate`. При вызове функции `numeric::cubicSpline` параметры <список узлов> и <список значений> можно перечислить в квадратных скобках через запятую: [<список узлов>, <список значений>]; либо квадратные скобки можно опустить и записать эти параметры через запятую: <список узлов>, <список значений>. <Список узлов> должен быть отсортирован по возрастанию.

Кроме того, узлы и значения в них можно сгруппировать как координаты точек, через которые должна проходить искомая функция: <точка1>, <точка2>, … Каждую точку следует задавать в виде списка *x*- и *y*-координат: [<узел>, <значение>], а сами точки перечислять через запятую. Например, если в узлах 0, π/6, π/4, π/3, π/2 известны следующие значения: 2, $\sqrt{3}$, $\sqrt{2}$, 1, 0, то функцию `numeric::cubicSpline` без опций можно вызвать одним из следующих способов: `numeric::cubicSpline([[0, PI/6, PI/4, PI/3, PI/2], [2, sqrt(3), sqrt(2), 1, 0]]); numeric::cubicSpline([0, PI/6, PI/4, PI/3, PI/2], [2, sqrt(3), sqrt(2), 1, 0]); numeric::cubicSpline([0, 2], [PI/6, sqrt(3)], [PI/4, sqrt(2)], [PI/3, 1], [PI/2, 0])`. Однако при построении интер-

поляционной функции по двум точкам по умолчанию считается, что узлы и значения в них сгруппированы по точкам (последним из перечисленных способов).

Функцию `numeric::cubicSpline` можно вызывать с одной из следующих <опций>: `Symbolic` – использует рациональный формат для числовых коэффициентов, полученных при построении сплайна (интерполяционной функции); `NoWarning` – подавляет предупреждающие сообщения; <граничные условия> – позволяет с помощью стандартных идентификаторов (`NotAKnot`, `Natural`, `Periodic`, `Complete = [a, b]`) выбирать значения производных в граничных узлах. Для построения полинома третьей степени по двум узлам необходимо в этих узлах знать не только значения искомой функции, но и значения ее первой производной. Для вычисления первых производных в каждом из узлов (кроме граничных) используются значения в соседних узлах. Значения первой производной искомой функции в граничных узлах можно задать различными способами: `NotAKnot` (используется по умолчанию) – так, чтобы третья производная интерполяционной функции на граничном и на примыкающем к нему интервале была одинаковой; `Natural` – так, чтобы вторая производная в граничных узлах обращалась в ноль; `Periodic` – так, чтобы значения первой и второй производных в граничных узлах совпадали (рекомендуется для интерполяции периодических функций, когда значения искомой функции в граничных узлах тоже совпадают); `Complete = [a, b]` – позволяет задать конкретные значения первой производной в граничных узлах (здесь `a` – значение первой производной в меньшем из граничных узлов, `b` – в большем из граничных узлов).

Функция `numeric::cubicSpline` возвращает неаналитическую функцию, аргументом которой по умолчанию является символьная переменная `z`. В поле вывода эта функция отображается с помощью идентификатора переменной <имя>, которой она присвоена при вызове функции `numeric::cubicSpline`. Например, в результате выполнения команды `S := numeric::cubicSpline([0, PI/6, PI/4, PI/3, PI/2], [2, sqrt(3), sqrt(2), 1, 0])` в поле вывода можно получить: proc S(z) … end. Если результат вызова функции `numeric::cubicSpline` будет присвоен другой переменной, то и сгенерированная неаналитическая функция будет иметь другое <имя>.

Notebook1* - MuPAD

File Edit View Navigation Insert Format Notebook

Узлы и значения в них

```
X := [0, PI/6,    PI/2, 4*PI/3, 2*PI]:
Y := [2, sqrt(3), 0,    -1,     2]:     X, Y
```

$$\left[0, \frac{\pi}{6}, \frac{\pi}{2}, \frac{4\pi}{3}, 2\pi\right], [2, \sqrt{3}, 0, -1, 2]$$

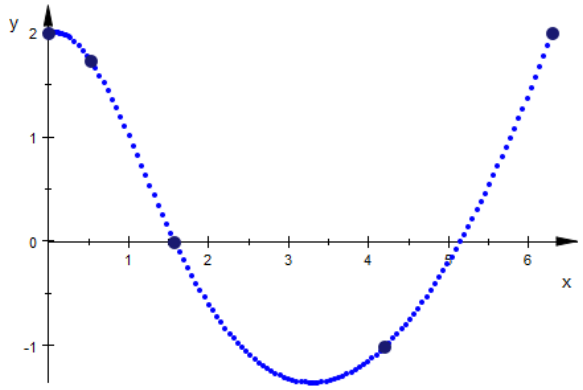
```
XY := [[op(X,k), op(Y,k)] $ k = 1..nops(X)]
```

$$\left[[0, 2], \left[\frac{\pi}{6}, \sqrt{3}\right], \left[\frac{\pi}{2}, 0\right], \left[\frac{4\pi}{3}, -1\right], [2\pi, 2]\right]$$

```
DIGITS := 3:  S := numeric::cubicSpline(X,Y)
```

proc S(z) ... end

```
SP := plot::Function2d(S(x), x=0..2*PI,
      PointsVisible=TRUE, PointSize=1*unit::mm,
      LinesVisible=FALSE):
points := plot::PointList2d(XY,
                        PointSize=2.7*unit::mm):
plot(SP, points)
```



Mem 11 MB, T 0 s Cmd INS

Рис. 9.2. Интерполяция кубическими сплайнами

В свою очередь, сгенерированную неаналитическую функцию <имя>(z) можно вызвать следующим образом:
<имя>(z, <номер интервала>, [<порядок производной>])

Единственным обязательным параметром при этом является z – аргумент сгенерированной функции <имя>. В качестве аргумента z функции <имя> можно использовать не только числовые значения из интервала, на котором заданы узлы, но и не заданную ранее символьную переменную. В этом случае функция <имя> возвращает не значение, а символьное выражение. При этом если для создания функции <имя> использовалась функция numeric::cubicSpline с опцией Symbolic, все числовые коэффициенты в этом выражении будут записаны в рациональном виде. Если для создания функции <имя> использовалась функция numeric::cubicSpline без опции Symbolic, все числовые коэффициенты в этом выражении будут записаны в десятичном формате.

Следует помнить, что с помощью неаналитической функции <имя> можно получить аналитический вид сплайна только между двумя конкретными соседними узлами. Поэтому вызвать функцию <имя> для символьного значения аргумента z можно только одновременно с параметром <номер интервала>. При этом параметр <номер интервала> может принимать только целочисленные значения: 0 – интервал между первым и вторым узлом, 1 – между вторым и третьим, 2 – между третьим и четвертым и т.д. Отрицательным значениям параметра <номер интервала> соответствует интервал, расположенный левее первого узла. На этом интервале функция numeric::cubicSpline строит такую же интерполяционную функцию, как на интервале между первым и вторым узлами. Параметру <номер интервала>, значение которого равно или превышает количество узлов, соответствует интервал, расположенный правее последнего узла. На этом интервале функция numeric::cubicSpline строит такую же интерполяционную функцию, как на интервале между последним и предпоследним узлами.

Указывая параметр [<порядок производной>], можно получить значение в фиксированной точке или аналитический вид на фиксированном интервале не только интерполяционной функции, но и ее производной указанного порядка. Так как между каждыми двумя соседними узлами строится кубический полином, то отличными от нуля могут быть только производные не выше третьего порядка включительно. Об этом следует помнить, задавая значение параметра [<порядок производной>] –

целое положительное число, записанное в квадратных скобках. Например, `[1]`. При этом параметры `z` и `<номер интервала>` имеют тот же смысл, как и для самой интерполяционной функции.

Например, если в результате выполнения последовательности команд `DIGITS := 3: S := numeric::cubicSpline([0, PI/6, PI/4, PI/3, PI/2], [2, sqrt(3), sqrt(2), 1, 0])` создана неаналитическая функция `S(x)` (которая в поле вывода отображается в виде proc S(z) … end), то с помощью следующих команд можно получить: `S(0.8)` – значение интерполяционной сплайн-функции в точке 0,8 (в поле вывода: 1.39); `S(x, 1)` – аналитический вид сплайн-функции на интервале [π/6, π/4] (в поле вывода: $((0.167x - 0.938)(x - 0.524) - 1.0)(x - 0.524) + 1.73$); `S(x, 3)` – аналитический вид сплайн-функции на интервале [π/3, π/2] (в поле вывода: $(x - 1.05)((0.288\,x - 0.794)(x - 1.05) - 1.73) + 1.0$); `S(x, 1, [2])` – аналитический вид второй производной сплайн-функции на интервале [π/6, π/4] (в поле вывода: $1.0x - 2.23$) и т.д.

Полученную с помощью функции `numeric::cubicSpline` неаналитическую функцию можно использовать для создания двумерного графического объекта типа «двумерный график» (`plot::Function2d`). Поэтому, чтобы построить график интерполяционной сплайн-функции, достаточно при создании такого графического объекта сгенерированную функцию `<имя>` вызвать с одним единственным параметром – символьным значением аргумента `z`. Например: `S(x)`. А чтобы построить график первой производной интерполяционной сплайн-функции, при создании графического объекта типа «двумерный график» у функции `<имя>` необходимо указать два параметра: символьное значение аргумента `z` и `[1]` – порядок производной. Например: `S(x, [1])`. При создании соответствующего графического объекта можно также указать интервал, на котором необходимо построить график для каждой из этих функций. Например, для создания графического объекта, соответствующего графику сплайн-функции на интервале [0, 2π], можно написать команду `plot::Function2d(S(x), x = 0..2*PI)`. Пример программы для построения графика кубического сплайна приведен на рис. 9.2. На этом рисунке точками увеличенного размера отмечены узлы интерполяции.

Даже при большом количестве узлов построение интерполяционной зависимости с помощью функции `numeric::cubicSpline` (в отличие от функции `interpolate`) не занимает много времени, так как для этого используются полиномы не выше третьей степени.

Для кусочно-непрерывной интерполяции функций двух переменных в MuPAD существует функция `numeric::cubicSpline2d`, с особенностями вызова которой можно ознакомиться в справочной системе MuPAD.

9.3. Представление аналитической функции полиномами в форме Бернштейна

Интерполяция используется не только для нахождения аналитического вида неизвестной функции по ее значениям в нескольких точках, но и при поиске полиномиального представления известной аналитической функции. В некоторых случаях, несмотря на то что аналитический вид функции известен, он может оказаться достаточно сложным для последующего использования. Чтобы облегчить дальнейшее использование таких функций, на заданном интервале их можно приближенно заменить интерполяционным полиномом. Для этого достаточно на этом интервале выбрать несколько узлов, вычислить в них значения исходной функции и после этого построить интерполяционный полином. Если исходная функция разлагается в сходящийся ряд Тейлора в любой точке заданного интервала, то с увеличением количества узлов последовательность соответствующих интерполяционных полиномов будет сходиться к истинной функции. Для построения таких интерполяционных полиномов в аналитическом виде можно воспользоваться интерполяцией Лагранжа (`interpolate`). В некоторых случаях в качестве полиномиального представления известной аналитической функции можно использовать полином, полученный путем отбрасывания остаточного члена разложения этой функции в ряд Тейлора (см. разд. 6.5). Однако следует помнить, что, в отличие от интерполяционного полинома Лагранжа, который совпадает с исходной функцией во всех узлах, полином, полученный путем отбрасывания остаточного члена ряда Тейлора, совпадает с исходной функцией только вблизи одной точки. Поэтому эти два способа получения полиномиального представления функции имеют, как правило, разные области применения.

Кроме того, начиная с MatLab 2015 в MuPAD включена функция `bernstein`, которая позволяет получить полиномиальное представление известной аналитической функции в форме Бернштейна. Функция `bernstein` тоже генерирует интерполяционный полином в аналитическом виде, но, в отличие от функции `interpolate`, которая требует узлы и

значения в них в качестве обязательных параметров, функция `bernstein` сама выбирает нужное количество узлов, вычисляет значения исходной функции в этих узлах и строит по ним полином определенного вида. Если полиномиальное представление известной функции $f(t)$ необходимо построить на интервале [0, 1], то узлы можно расположить на этом интервале равномерно с шагом $1/n$. Тогда на интервале [0, 1] получится $(n + 1)$ узел: 0, $1/n$, $2/n$, $3/n$, …, k/n, …, 1. В этих узлах функция $f(t)$ принимает следующие значения: $f(0)$, $f(1/n)$, $f(2/n)$, $f(3/n)$, …, $f(k/n)$, …, $f(1)$. Если для построения полиномиального представления по этим узлам используют линейную комбинацию многочленов Бернштейна

$$b_n(t,k) = \binom{k}{n} t^k (1-t)^{n-k} ,$$

то искомое представление можно записать в виде:

$$B_n(t) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) b_n(t,k),$$

где $f(k/n)$ – значения исходной функции в узлах, $\binom{k}{n}$ – биномиальный коэффициент. Такое представление исходной функции называется ***представлением в виде полинома, записанного в форме Бернштейна***. Следует отметить, что полиномиальное представление Бернштейна не является интерполяционным, так как функция $B_n(t)$ совпадает с исходной функцией $f(t)$ не во всех, а только в некоторых узлах. Однако доказано, что $\lim\limits_{n\to\infty} B_n(t) = f(t)$.

Чтобы построить в MuPAD такое представление исходной функции на интервале [0, 1], функцию `bernstein` вызывают следующим образом:

```
<Результат> := bernstein(<функция>, <аргумент>,
                         <степень>, <переменная>)
```

В качестве параметра `<функция>` необходимо использовать математическое выражение, идентификатор объявленной ранее математической функции или ее вызов с символьными аргументами. `<Функция>` должна содержать хотя бы одну не заданную ранее символьную переменную. С помощью параметра `<аргумент>` следует указать, относительно какой из символьных переменных (если их несколько), использованных при записи `<функции>`, необходимо строить полиномиальное представление

(выбирать узлы и вычислять значения <функции> в них). Если при записи <функции> использована только одна символьная переменная, то параметр <аргумент> можно опустить. В качестве параметра <степень> следует использовать целое положительное число n, которое будет задавать степень многочленов Бернштейна $b_n(t, k)$, используемых в разложении. Для получения такого разложения на интервале [0, 1] автоматически будут выбраны узлы с шагом $1/n$, количество которых превосходит указанную <степень> на единицу. В качестве параметра <переменная> необходимо указать идентификатор символьной переменной, через которую следует записать полученный полином. Функция `bernstein` получает полиномиальное представление в аналитическом виде и возвращает полиномиальное выражение, скомпонованное по степеням указанной <переменной>.

Например, если необходимо получить представление в виде полинома 5-й степени, записанного в форме Бернштейна, для функции e^{-25x} на интервале [0, 1], это можно сделать одним из следующих способов: 1) `bernstein(exp(-25*x), 5, t);` 2) `bernstein(exp(-25*x), x, 5, t);` 3) `f := x -> exp(-25*x): bernstein(f, 5, t).` В результате выполнения каждой из приведенных последовательностей команд в поле вывода появится следующий полином:

$$-(t-1)^5 + 5e^{-5}t(t-1)^4 - 10e^{-10}t^2(t-1)^3 + 10e^{-15}t^3(t-1)^2 - \\ -5e^{-20}t^4(t-1) + e^{-25}t^5 .$$

Подставляя в полученное выражение узлы 0, 1/5, 2/5, 3/5, 4/5, 1, нетрудно убедиться, что построенный полином совпадает с исходной функцией e^{-25x} только в узлах 0 и 1.

Так как функция `bernstein` выбирает узлы и строит полиномиальное представление только на интервале [0, 1], чтобы получить аналогичное представление на интервале $[a, b]$ конечной длины, не совпадающем с интервалом [0, 1], необходимо в исходной функции $g(x)$ сделать такую замену переменных $x = Kt + L$, чтобы значениям $x \in [a, b]$ соответствовали значения $t \in [0, 1]$, т.е. $K = b - a$, $L = a$. В результате такой замены переменных получится функция $f(t)$, значения которой на интервале [0, 1] совпадают со значениями исходной функции $g(x)$ на интервале $[a, b]$:

$$g(x) = g(a + (b-a)\cdot t) = f(t) \text{ при } x = a + (b-a)\cdot t .$$

Далее для вспомогательной функции $f(t)$ на интервале [0, 1] можно получить полиномиальное представление в форме Бернштейна с помощью

функции `bernstein`, а в полученном полиномиальном выражении выполнить обратную замену переменных $t = \dfrac{x-a}{b-a}$. В результате получится полиномиальное представление для исходной функции $g(x)$ на интервале $[a, b]$. Таким образом, чтобы построить полиномиальное представление в форме Бернштейна математической функции одной переменной на интервале $[a, b]$ необходимо выполнить следующую последовательность действий:

1) записать аналитическое выражение исходной функции через ее аргумент в символьном виде (или объявить исходную функцию);

2) сделать замену переменных в исходной функции так, чтобы значения исходной функции на интервале $[a, b]$ совпадали со значениями преобразованного выражения на интервале [0, 1] (это можно осуществить, например, с помощью функции `subs`);

3) для полученного выражения построить полиномиальное представление в форме Бернштейна (с помощью функции `bernstein`);

4) выполнить обратную замену переменных в полученном полиномиальном выражении;

5) при необходимости построить график полученного полинома на интервале $[a, b]$.

Например, чтобы получить полиномиальное представление (полиномами третьей степени) в форме Бернштейна функции $e^{-x/10}\cos x$ на интервале [1, 5] можно выполнить следующие команды:

```
gx := exp(-x/10)*cos(x);
a := 1: b := 5:
gt := subs(gx, x = a+(b-a)*t);
pt := bernstein(gt, 3, t);
px := subs(pt, t = (x-a)/(b-a));
```

Если для сравнения полученного представления с исходной функцией необходимо построить их графики, то приведенную выше последовательность команд следует дополнить:

```
f1 := plot::Function2d(gx, x=a..b, Color=RGB::Blue):
XY := [[a+k*(b-a)/3, subs(gx, x = a+k*(b-a)/3)]
                                        $ k=0..3];
points := plot::PointList2d(XY):
f2 := plot::Function2d(px, x=a..b, Color=RGB::Red):
plot(f1, points, f2)
```

В результате выполнения этих команд на графике кривой синего цвета будет изображена исходная кривая, кривой красного цвета – полиномиальное представление, а точками – использованные для построения этого представления узлы.

Глава 10. ЭЛЕМЕНТЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

MuPAD содержит не только широкий спектр стандартных функций, но и такие средства программирования, как операторы, индексированные переменные, генераторы случайных чисел, нестандартные функции и т.д., позволяющие в каждой программе организовать такую последовательность символьных вычислений и численных расчетов, которая наилучшим образом соответствует конкретной решаемой задаче. Рассмотрим эти возможности MuPAD подробнее.

10.1. Генераторы случайных чисел

Несмотря на то, что генерация случайных чисел может происходить в различных диапазонах значений и в соответствии с разными функциями распределения, в действительности значения этих чисел выбираются из встроенной в соответствующий язык программирования базовой последовательности. Поэтому генерируемые программой числа называются псевдослучайными. MuPAD содержит большое количество стандартных функций, реализующих генерацию псевдослучайных чисел, подчиняющихся различным статистическим распределениям. Каждая функция, предназначенная для генерации случайных чисел, по-разному преобразует числа, извлеченные ею из базовой последовательности. Благодаря этому с помощью разных функций можно получить разные последовательности (наборы) случайных чисел в разных числовых диапазонах.

Случайные числа считаются равномерно распределенными на интервале $[a, b]$, если при многократной генерации числа из разных частей этого интервала появляются с одинаковой вероятностью. Для генерации вещественных чисел, равномерно распределенных на интервале $[0, 1)$, в MuPAD можно использовать функцию `frandom`. Эту функцию вызывают следующим образом:

```
<Результат> := frandom()
<Имя> := frandom(<опция>)
```

Если функция `frandom` вызывается без `<опции>`, то она возвращает из интервала $[0, 1)$ одно вещественное число в десятичном формате. Например, после выполнения команды `x := frandom()` переменная `x`

будет содержать одно число из интервала [0, 1), возможно 0.2703581656. В результате повторного выполнения команды `x := frandom()`, переменная `x` будет содержать другое число из этого же интервала. Однако после каждого перезапуска MuPAD будет генерироваться одно и то же число. Последовательность генерируемых функцией `frandom()` чисел после каждого перезапуска MuPAD также будет одинаковой. Это связано с тем, что в момент запуска (или после очистки памяти с помощью функции `reset`) MuPAD устанавливает начальную позицию для выбора элементов из базовой последовательности, имитирующей появление случайных чисел. Эта начальная позиция одинакова для всех функций, генерирующих случайные числа. Каждая функция, в том числе и `frandom`, начинает выбор элементов из базовой последовательности с этой позиции. Чтобы изменить позицию, с которой далее начнут выбираться значения из базовой последовательности, для функции `frandom` используют `<опцию>`. В качестве `<опции>` можно использовать целое положительное число, которое меньше 10^{10}. Указанное число – позиция, начиная с которой будут выбираться числа из базовой последовательности. Если в качестве `<опции>` задано фиксированное число, то при повторном выполнении программы будет генерироваться одна и та же последовательность случайных чисел из интервала [0, 1). Но эта последовательность будет отличаться от последовательности, создаваемой функцией `frandom` без `<опции>`. Если при каждом выполнении программы необходимо получать разные последовательности случайных чисел (использовать разную начальную позицию), то в качестве `<опции>` используют идентификатор `CurrentTime`. Например: `r := frandom(CurrentTime)`.

Если функция `frandom` вызывается с `<опцией>`, то она создает новую функцию-генератор. Переменная `<имя>` становится идентификатором этой функции, который используется для ее вызова следующим образом:
`<Результат> := <имя>()`

В этом случае переменной `<результат>` также будет присвоено одно вещественное число в десятичном формате из интервала [0, 1). Например: `r := frandom(CurrentTime): x := r()`.

Многократный вызов созданной таким образом функции `<имя>()` позволяет получить последовательность равномерно распределенных на интервале [0, 1) случайных чисел. Эта последовательность будет содержать разные значения при разных запусках программы, которая ее генерирует,

а также будет отличаться от последовательности, создаваемой функцией `frandom()`.

Количество значащих цифр в сгенерированных функцией `frandom` вещественных числах фиксируется с помощью переменной окружения `DIGITS` (по умолчанию `DIGITS = 10`).

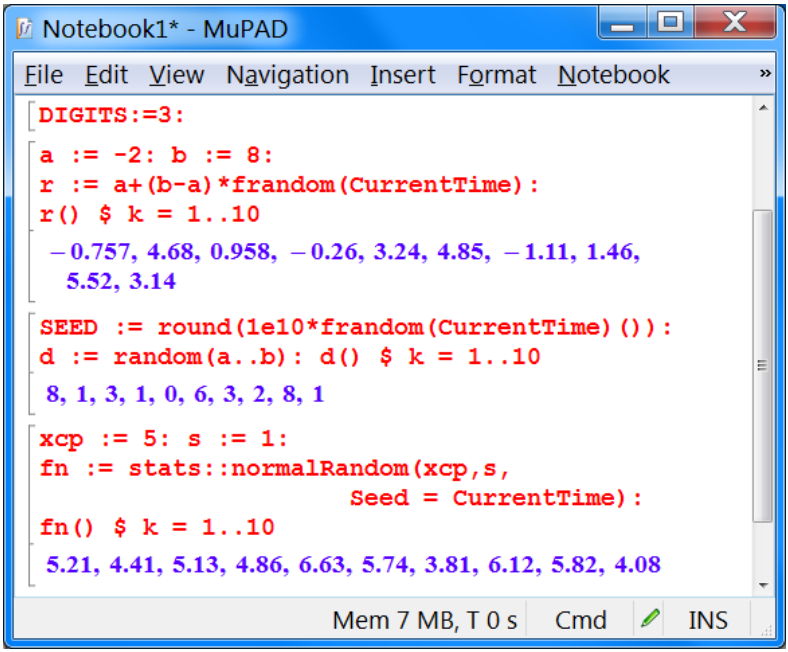


Рис. 10.1. Генерация последовательностей случайных чисел

Для генерации случайных вещественных чисел, равномерно распределенных на конечном интервале, отличающемся от интервала [0, 1), можно использовать функцию `stats::uniformRandom` или выполнить линейное преобразование сгенерированных функцией `frandom` значений:

```
<Результат> := <начало>+(<конец>-<начало>)*frandom()
```

или

```
<Имя> := <начало>+(<конец>-<начало>)*frandom(<опция>);
<Результат> := <имя>()
```

В качестве <начала> и <конца> интервала можно использовать числовые значения или арифметические выражения, результатами вычисления которых являются числовые значения. В этом случае переменной <результат> будет присвоено случайное вещественное число из интервала [<начало>, <конец>). Например, в результате выполнения команды `x := 2 + (6-2)*frandom()` можно получить случайное вещественное число из интервала [2, 6); а с помощью команды `x := a + (b–a)*frandom()` – случайное вещественное число из интервала [*a, b*), если переменным `a` и `b` ранее присвоены числовые значения. Соответственно, для получения случайного вещественного числа из интервала [*a, b*) с помощью функции `frandom` с <опцией> `CurrentTime` необходимо написать две команды, например:

```
r_a_b := a + (b-a)*frandom(CurrentTime): x := r_a_b()
```

Для генерации целых чисел, равномерно распределенных на заданном интервале, в MuPAD можно использовать функцию `random`. Эту функцию вызывают следующим образом:

```
<Результат> := random()
<Имя> := random(<начало>..<конец>)
```

Если функция `random` вызывается без параметров, то она возвращает одно целое число из интервала [0, 10^{12}]. Например, после выполнения команды `x := random()` переменная `x` будет содержать одно целое положительное число, возможно 427419669081.

Если в качестве параметра функции `random` указать интервал <начало>..<конец>, то создается новая функция <имя>, с помощью которой можно генерировать целые числа, подчиняющиеся равномерному распределению в заданном интервале. Для этого сгенерированную ранее функцию <имя> вызывают следующим образом:

```
<Результат> := <имя>()
```

Например, в результате выполнения следующих команд: `d1 := random(2..6): x := d1()`, – переменной `x` будет присвоено одно из чисел: 2, 3, 4, 5 или 6. Если вместо интервала <начало>..<конец> указать одно значение, то вновь созданная функция <имя> будет генерировать целые неотрицательные числа меньше указанного значения. Например, в результате выполнения следующих команд: `d2 := random(4): x := d2()`, – переменной `x` будет присвоено одно из чисел: 0, 1, 2 или 3.

У функции `random` отсутствуют опции, позволяющие устанавливать начальную позицию при выборе значений из базовой последовательности,

имитирующей появление случайных чисел. Однако эту позицию можно задать с помощью переменной окружения `SEED`, которая может принимать целые положительные значения меньше 10^{10}. Чтобы повлиять на выбор начальной позиции в базовой последовательности, необходимо присвоить нужное значение переменной `SEED` перед вызовом функции `random`. Например:

```
SEED := 341: d3 := random(-4..6): x := d3()
```

Чтобы значение переменной `SEED` зависело от текущего времени, ее значение задают с помощью функции `frandom` следующим образом:

```
SEED := round(1e10*frandom(CurrentTime)())
```

Функция `stats::normalRandom` позволяет создавать функции, генерирующие вещественные случайные числа, подчиняющиеся нормальному распределению (распределению Гаусса). При многократной генерации подчиняющиеся этому распределению числа чаще всего появляются вблизи числа, которое называется математическим ожиданием. Кроме математического ожидания нормальное распределение характеризуется также среднеквадратичным отклонением – это положительное число, равное половине длины интервала, в который попадает большинство (67%) сгенерированных случайных чисел, если их количество стремится к бесконечности. Такой интервал расположен симметрично относительно математического ожидания. Чтобы создать функцию, генерирующую вещественные случайные числа, подчиняющиеся нормальному распределению, функцию `stats::normalRandom` вызывают следующим образом:

```
<Имя> := stats::normalRandom(<м. ожидание>,
                             <ср.кв. отклонение>, <опция>)
```

В качестве `<м. ожидания>` необходимо задать числовое значение, вблизи которого генерируемые значения должны появляться чаще всего. При большом количестве сгенерированных чисел среднее арифметическое значение этих чисел будет равно `<м. ожиданию>`. В качестве `<ср.кв. отклонения>` также необходимо задать положительное числовое значение, которое определяет среднее отклонение сгенерированных чисел от `<м. ожидания>`. Для нормального распределения вероятность того, что сгенерированное число будет отличаться от `<м. ожидания>` больше, чем на три `<ср.кв. отклонения>`, меньше 5%. В качестве параметров `<м. ожидание>` и `<ср.кв. отклонение>` можно также использовать

арифметические выражения, если результатом их вычисления являются соответствующие числовые значения.

Чтобы для нормального распределения задать начальную позицию в базовой последовательности случайных чисел, отличной от значения заданного по умолчанию, при вызове функции `stats::normalRandom` используют <опцию>. Ее можно задать в виде: `Seed = <число>` либо `Seed = CurrentTime`. В качестве параметра <число> можно использовать только целое положительное число. <Опция> для функции `stats::normalRandom` имеет тот же смысл, что и <опция> для функции `frandom`, а значение переменной окружения `SEED` не оказывает влияния на функцию `stats::normalRandom`.

Чтобы сгенерировать одно случайное число, подчиняющееся нормальному распределению, используют созданную ранее с помощью `stats::normalRandom` функцию <имя> следующим образом:

```
<Результат> := <имя>()
```

Например, для получения случайного числа, подчиняющегося нормальному распределению с математическим ожиданием 5 и среднеквадратичным отклонением 2, необходимо выполнить следующие команды:

```
g := stats::normalRandom(5, 2, Seed = CurrentTime):
x := g()
```

Количество значащих цифр в сгенерированных функцией <имя> (созданной с помощью функции `stats::normalRandom`) вещественных числах также фиксируется с помощью переменной окружения `DIGITS` (по умолчанию `DIGITS = 10`).

MuPAD содержит также функции, позволяющие генерировать случайные числа, подчиняющиеся другим статистическим распределениям. Эти функции используются аналогично перечисленным выше функциям.

Каждая из рассмотренных выше функций генерирует одно случайное число. Чтобы сгенерировать несколько случайных чисел, подчиняющихся одному и тому же распределению, вызов соответствующей функции необходимо повторить соответствующее количество раз. Для этого можно использовать оператор `$`, который записывают после вызова функции, генерирующей одно случайное число:

```
<Результат> := <функция>() $
                          <переменная> = 1..<количество>
```

В качестве параметра <функция>() можно использовать вызов одной из функций frandom(), random() или нестандартной функции <имя>(), созданной с помощью одной из описанных выше стандартных функций. Так как <функция>(), генерирующая случайные числа, не имеет параметров (аргументов), то указанная после оператора $ <переменная> никак не связана с используемым распределением (ее можно интерпретировать как номер сгенерированного числа), но в данном случае она является обязательным параметром для этого оператора. С помощью параметра <количество> можно задать, сколько случайных чисел будет сгенерировано. При необходимости начальное и конечное значения <переменной> можно задать не как 1..<количество>, а иначе, но в соответствии с правилами использования оператора $. После выполнения команды переменной <результат> будет присвоена последовательность случайных чисел, которые в поле вывода отображаются через запятую.

Далее приведены примеры получения (без присвоения) последовательности случайных чисел, которые при повторном выполнении программы будут одинаковыми:

1) 10 вещественных случайных чисел, подчиняющихся равномерному распределению в интервале [0, 1):

```
frandom() $ k = 1..10
```

2) 10 целых случайных чисел, подчиняющихся равномерному распределению в интервале [0, 10^{12}]:

```
random() $ k = 1..10
```

3) 10 вещественных случайных чисел, подчиняющихся нормальному распределению с математическим ожиданием 0 и среднеквадратичным отклонением 1:

```
f := stats::normalRandom(0, 1): f() $ k = 1..10
```

Далее приведены примеры получения (без присвоения) последовательности случайных чисел, которые при повторном выполнении программы будут разными (рис. 10.1):

1) 10 вещественных случайных чисел, подчиняющихся равномерному распределению в интервале [a, b) (если переменным a и b ранее присвоены разные числовые значения):

```
r := a+(b-a)*frandom(CurrentTime): r() $ k = 1..10
```

2) 10 целых случайных чисел, подчиняющихся равномерному распределению в интервале [a, b] (если переменным a и b ранее присвоены разные целочисленные значения):

```
SEED := round(1e10*frandom(CurrentTime)()):
d := random(a..b): d() $ k = 1..10
```

3) 10 вещественных случайных чисел, подчиняющихся нормальному распределению с математическим ожиданием `xcp` и среднеквадратичным отклонением `s` (если переменным `xcp` и `s` ранее присвоены положительные числовые значения):

```
fn := stats::normalRandom(xcp, s, Seed = CurrentTime):
fn() $ k = 1..10
```

Следует напомнить, что оператор `$` можно использовать также и для получения последовательности значений других функций. Например, если `f(x)` – ранее объявленная стандартная или нестандартная функция, зависящая от одного аргумента, можно получить последовательность ее значений для `x` от `a` до `b` с шагом `dx` следующим образом: `f(x) $ x = a..b step dx` (если переменным `a`, `b`, `dx` ранее присвоены значения, соответствующие типу аргумента функции `f`). В этом случае параметр `<переменная>` должен совпадать с аргументом функции (см. разд. 4.1).

10.2. Ввод информации с клавиатуры в диалоговом режиме

Несмотря на то, что в MuPAD и код программы, и результаты ее выполнения хранятся в одном и том же файле, а все исходные данные доступны для изменения в любой момент времени, существует возможность организовать ввод информации в диалоговом режиме. Для этого используется оператор `input`. Если необходимо ввести с клавиатуры значение одной переменной, то оператор `input` вызывают следующим образом:

```
<Переменная> := input("<приглашение>")
```

В качестве параметра `<приглашение>` в двойных кавычках можно указать текст, побуждающий пользователя вводить определенные данные. Дойдя до оператора `input`, программа останавливается, появляется дополнительное диалоговое окно «`Enter Input`», содержащее поле ввода и две кнопки: `Ok` и `Cancel`. Над полем ввода отображается текст `<приглашения>`. Если параметр `<приглашение>` опустить (не указывать), то над полем ввода по умолчанию появится текст «Please enter expression:». При этом, даже если `<приглашение>` отсутствует, круглые

скобки после оператора `input` писать необходимо. Например: `input()`. Окончание ввода данных определяется по нажатию кнопки `Ok` в диалоговом окне «`Enter Input`». Введенная информация (число, выражение, последовательность и т.д.) присваивается указанной слева от оператора `input` (перед знаком `:=`) `<переменной>`, в поле вывода появляется введенное значение. Например, в процессе выполнения следующего оператора `x := input("Введите значение переменной x:")` появится окно, изображенное на рис. 10.2, а после ввода выражения и нажатия на кнопку `Ok` под командным полем этого оператора появится поле вывода, в котором будет написано: $\sqrt{2}+\pi$.

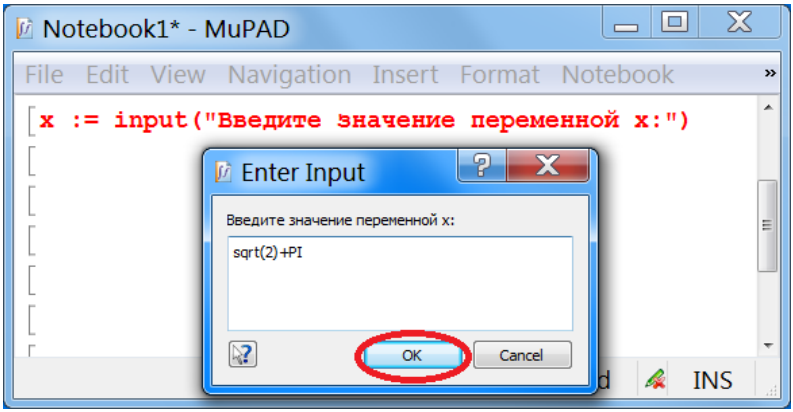


Рис. 10.2. Ввод значения с клавиатуры

Если необходимо ввести с клавиатуры несколько переменных, то оператор `input` можно записать следующим образом:

```
input("<приглашение1>", <переменная1>,
                        "<приглашение2>", <переменная2>,…)
```

В качестве параметров `<переменная1>`, `<переменная2>`… необходимо указать идентификаторы переменных, значения которых будут вводиться пользователем. При выполнении этого оператора диалоговое окно «`Enter Input`» будет появляться для ввода каждой переменной отдельно. Поэтому для разных переменных следует написать разные приглашения: `<приглашение1>`, `<приглашение2>`, …, заключив каждое в кавычки. После выполнения такого оператора `input` в поле вывода будет отображаться только значение последней введенной переменной.

При вводе значений в окне «Enter Input» необходимо соблюдать правила записи переменных различных типов: списки необходимо записывать в квадратных скобках через запятую, множества – в фигурных скобках через запятую, строки – в двойных кавычках и т.д. Например, при выполнении оператора myset := input("Введите множество:") в окне «Enter Input» можно ввести: $\{3, 5*x, -6, x-2\}$.

Если важно, чтобы введенная пользователем информация при любых условиях интерпретировалась программой как текст (строка), то для ввода такой переменной можно использовать оператор textinput, правила использования которого аналогичны правилам использования оператора input:

<Переменная> := **textinput**("<приглашение>")

В отличие от оператора input в процессе выполнения оператора textinput введенный в появившемся диалоговом окне «Enter Text» текст после нажатия кнопки Ok автоматически заключается в кавычки. Поэтому писать их в окне «Enter Text» не нужно. Например, для получения одного и того же результата при выполнении оператора name := input("Введите свое имя:") в окне «Enter Input» текст "Петя" нужно вводить в кавычках, а при выполнении оператора name := textinput("Введите свое имя:") в окне «Enter Text» – без кавычек.

10.3. Вывод информации

Вывод результатов вычислений (результатов выполнения команд) осуществляется в поле вывода по умолчанию. Оператор print позволяет отображать информацию, полученную программой, в удобном для программиста виде, не ограничиваясь правилами, принятыми по умолчанию. Для вывода значений переменных (информации, содержащейся в них) оператор print используют следующим образом:

print(<опции>, <переменная1>, <переменная2>, …)

В качестве параметров <переменная1>, <переменная2>… можно использовать числовые, символьные и строковые константы, идентификаторы переменных, выражения, списки, множества, матрицы и прочие объекты MuPAD. В результате выполнения этого оператора, вызванного без опций, в поле вывода будут записаны через запятую все перечисленные объекты, вместо заданных ранее переменных будут выведены их значения. Тот же результат получится, если в командном поле перечислить те же объекты в том же порядке через запятую. Например, и

команда `print("Ответ:", a, b, "x = ", x)`, и команда `"Ответ:", a, b, "x = ", x` позволяют получить один и тот же результат: "Ответ:", a, b, "x = ", $\sqrt{2}+\pi$. Но, в отличие от перечисления в командном поле, вывод с помощью оператора `print` нельзя запретить с помощью двоеточия. Например, после выполнения команды `"Ответ:", a, b, "x = ", x:` поле вывода не появится, а после выполнения оператора `print("Ответ:", a, b, "x = ", x):` в поле вывода отображается тот же результат, как в предыдущем примере: "Ответ:", a, b, "x = ", $\sqrt{2}+\pi$. Это свойство оператора `print` оказывается существенным при его использовании совместно с операторами циклов и операторами условных переходов (см. разд. 10.6, 10.7).

Кроме того, благодаря использованию `<опций>` можно существенно изменить способ отображения выводимой информации. Оператор `print` можно использовать с одной из следующих `<опций>`:

`Unquoted` – строковые значения отображаются без кавычек; выполняется форматирование текста с помощью управляющих символов (`\n` – новая строка; `\t` – табуляция; `\b` – разрыв строки);

`NoNL` – аналогична опции `Unquoted`, но без добавления новой строки в конце выведенной информации;

`KeepOrder` – предотвращает изменение порядка слагаемых в выражениях (по умолчанию положительные слагаемые выводятся раньше отрицательных);

`Plain` – вывод результатов вычислений в текстовом виде (как в командном поле, например: `(x - 2)/(x + 5) + sqrt(2)`);

`Typeset` – вывод результатов вычислений в графическом виде; одновременно с `Plain` не используется, одновременно с `Unquoted` или `NoNL` игнорируется.

Так, чтобы при выводе значения нескольких переменных не разделялись запятыми, можно выполнить следующие операции: значение каждой переменной преобразовать в текст (строку); полученные строки объединить в одну и вывести с опцией `Unquoted` (или `NoNL`). Для преобразования выражения в текст (строку) можно использовать функцию `expr2text`, в качестве единственного параметра которой необходимо указать `<исходное выражение>` – константу, переменную, математическое выражение, функцию с ее аргументами, список, последовательность, матрицу и т.д.:

<Строка> := **expr2text**(<исходное выражение>)

Результатом вызова этой функции является <строка> – текстовый образ <исходного выражения>. Например, в результате выполнения команды `expr2text(a + b)` получится строка "a + b". При использовании функции `expr2text` следует помнить, что перед преобразованием в строку <исходное выражение> записывается в текстовом виде. Например, в результате выполнения команд `y := 5/x: expr2text(y)` получится строка "5/x".

Объединить две строки (<строка1> и <строка2>) в одну (<строка3>) можно с помощью функции . (точка) следующим образом:
<Строка3> := <строка1>.<строка2>

Например, в результате выполнения команды `"x = "."a + b"` получится строка "x = a + b".

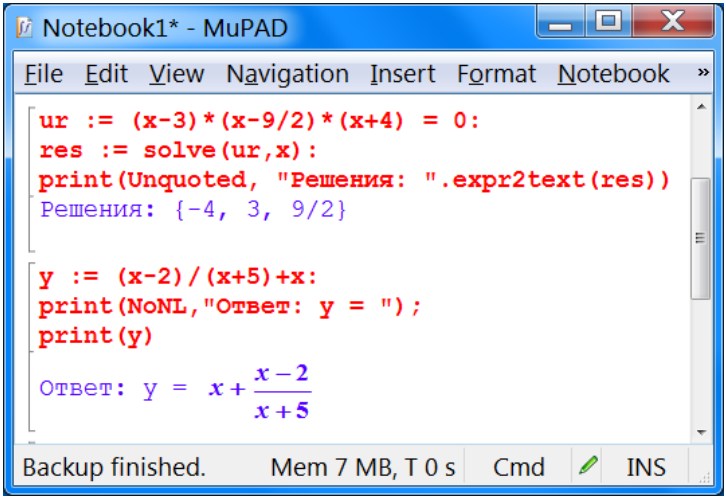


Рис. 10.3. Вывод результатов вычислений с поясняющим текстом

Используя эти функции и оператор `print` с опцией `Unquoted`, можно организовать вывод результатов вычислений с поясняющим текстом (рис. 10.3). Например, если в командном поле записана последовательность операторов:

```
ur := (x - 3)*(x - 9/2)*(x + 4) = 0:
res := solve(ur, x):
```

```
print(Unquoted, "Решения: ".expr2text(res))
```
то в поле вывода появится текст:
```
Решения: {-4, 3, 9/2}
```

Такой способ вывода удобен, если выводимая переменная содержит числовые значения в десятичном формате. Если выводимая переменная содержит дроби, квадратные корни, суммы, пределы, интегралы и т.д., то текстовое представление таких переменных часто является неудобным. В этом случае текстовое пояснение перед значением переменной можно вывести с помощью одного оператора `print` с опцией `NoNL`, а саму переменную – с помощью другого оператора `print` без опций (рис. 10.3). Например:
```
y := (x - 2)/(x + 5) + x:
print(NoNL, "Ответ: y = ");
print(y)
```

Тогда в поле вывода получится:

Ответ: y = $x+\dfrac{x-2}{x+5}$

Использование функции `round` позволяет выводить полученные числовые значения, представленные в десятичном формате, с различной точностью. Например, в результате выполнения команд `z := PI/3 + sqrt(2): print(z = round(z, 2), round(z, 5), float(z))` можно вывести значение переменной `z` в десятичном формате с двумя (`round(z, 2)`), с пятью (`round(z, 5)`) десятичными знаками после запятой, а также с количеством знаков, заданным по умолчанию (`float(z)`). В результате в поле вывода получится: $\dfrac{\pi}{3}+\sqrt{2}=2.46$, 2.46141, 2.461411114.

10.4. Использование внешних файлов для сохранения и загрузки информации

Под файлом понимают поименованную область внешней (дисковой) памяти компьютера. Внешние файлы, расположенные на жестком диске компьютера, могут использоваться как для хранения исходных данных, так и для хранения результатов работы программы. Рассмотрим возможности MuPAD по использованию файлов на примере текстовых (форматированных) файлов, которые могут содержать не только буквы русского

и латинского алфавитов, целые, вещественные и комплексные числа, но и любые символы псевдографики.

10.4.1. Обращение к файлу

Перед началом взаимодействия с конкретным внешним файлом необходимо ***открыть файл для чтения или записи*** информации. Для этого используют функцию fopen, которую вызывают следующим образом:

```
<Идентификатор файла> := fopen("<имя файла>",
                                <цель>, <тип>, <опция>)
```

В качестве параметра <имя файла> необходимо указать название файла, которое должно включать расширение и путь к файлу (если файл находится в той же папке, что и использующая его программа, путь к файлу можно не указывать). Если в качестве параметра <имя файла> используется идентификатор TempFile, то функция fopen откроет временный файл в системной папке Temp. Название этого файла можно узнать с помощью функции fname, которая будет описана позже. С помощью параметра <цель> необходимо указать, с какой целью открывается файл, используя один из следующих идентификаторов: Read – для чтения, Write – для записи (создается новый или заменяется существующий файл), Append – для добавления информации к существующему файлу.

С помощью параметра <тип> можно указать, в каком формате будет читаться или записываться информация в этот файл. Если информацию планируется читать или сохранять в двоичном формате, в качестве параметра <тип> функции fopen используют идентификаторы Bin или Raw; если файл текстовый, – то идентификатор Text. Если параметр <тип> не задан, по умолчанию для записи (Write) используется Bin, а при чтении (Read) и добавлении (Append) информации – формат, в котором хранятся данные в открываемом файле. Поэтому, если необходимо сохранить данные в текстовый файл, параметр <тип> следует обязательно указать, а если данные читают из текстового файла, то этот параметр можно опустить.

С помощью <опции> можно принудительно выбрать кодировку. Значение <опции> по умолчанию соответствует кодировке, используемой операционной системой. При необходимости кодировку задают в виде Encoding = <кодировка>, где в качестве значения параметра

<кодировка> используют одну из следующих строковых констант: "Big5", "ISO-8859-1", "windows-932", "EUC-JP", "ISO-8859-2", "windows-936", "GBK", "ISO-8859-3", "windows-949", "KSC_5601", "ISO-8859-4", "windows-950", "Macintosh", "ISO-8859-9", "windows-1250", "Shift_JIS", "ISO-8859-13", "windows-1251", "US-ASCII", "ISO-8859-15", "windows-1252", "UTF-8", "windows-1253", "windows-1254", "windows-1257".

Функция fopen возвращает <идентификатор файла> – числовое значение, которое рекомендуется присвоить отдельной переменной, наличие которой упрощает обращение к обозначенному ею файлу в дальнейшем.

Таким образом, для корректного открытия файла рекомендуется использовать функцию fopen следующим образом. Например, чтобы открыть файл example1.dat на чтение, можно написать команду

```
FL := fopen("example1.dat", Read);
```

Чтобы открыть текстовый файл variant3.txt для записи – команду

```
F := fopen("variant3.txt", Write, Text).
```

Переменные FL и F используются в дальнейшем в программе для обращения к файлам example1.dat и variant3.txt соответственно, если эти файлы расположены в той же папке, что и программа, в которой написаны приведенные выше команды.

Если необходимо задать расположение файла на жестком диске, используют полное имя файла, содержащее последовательно названия диска, каталогов, в которых расположен файл, и имя файла с расширением. Например:

```
FL := fopen("c:\\examples\\example1.dat", Read):
 F := fopen("d:\\work\\data\\variant3.txt",
                                        Write, Text):
```

Обратите внимание! При работе в операционной системе Windows в качестве разделителей папок (каталогов) вместо \ в MuPAD следует писать \\.

Если файл открыт для чтения или записи (Read, Write), указатель файла устанавливается на его начало. Если файл открыт для добавления (Append) информации, указатель файла устанавливается на его конец.

При использовании функции fopen имя файла можно указывать как непосредственно на месте ее первого параметра, так и с помощью переменной, содержащей соответствующую информацию в строковом формате.

Примеры использования дополнительной переменной для хранения имени файла:

```
fileName := "tema.txt":  FL := fopen(fileName, Read):
Sfile := "d:\\work\\var5.txt":
                         F := fopen(Sfile, Write, Text):
```

Когда необходимо использовать несколько файлов из одной папки, путь к папке бывает удобно представить в виде отдельной переменной и в дальнейшем объединять его с именами отдельных файлов с помощью функции . (точка). Например:

```
Path := "d:\\work\\":
fDat := fopen(Path."variant4.txt", Read):
fRes := fopen(Path."result4.txt", Write, Text):
```

Определить, в какой папке (полный путь) находится данная программа, можно с помощью функции NOTEBOOKPATH, которую вызывают следующим образом:

<Переменная> := **NOTEBOOKPATH**

Функция NOTEBOOKPATH возвращает полный путь к текущему файлу в строковом формате.

Тогда путь к открываемому файлу можно формировать с учетом этой папки. Например, если внутри папки, в которой хранится данная программа, существует папка DATA, в которой лежит файл variant4.txt, то открыть его для чтения можно с помощью следующих команд:

```
Path := NOTEBOOKPATH:
fDat := fopen(Path."DATA\\variant4.txt", Read):
```

Если в качестве параметра <имя файла> при вызове функции fopen используется идентификатор TempFile, то узнать название и расположение используемого временного файла можно с помощью функции fname, которую вызывают следующим образом:

<Имя файла> := **fname**(<идентификатор файла>)

Функция fname возвращает в строковом формате <имя файла>, ранее обозначенного в программе <идентификатором файла>, и содержит полный путь к файлу.

После окончания работы (чтения, записи информации) с любым открытым ранее файлом его необходимо ***закрыть***. Для этого используют функцию fclose следующим образом:

fclose(<идентификатор файла>)

При обращении к этой функции следует указать только один параметр – идентификатор открытого ранее файла. Если программа обращалась к нескольким внешним файлам, необходимо закрыть каждый из них. Например:

```
fclose(fDat): fclose(F): fclose(fRes):
```

Своевременное закрытие файла с помощью функции `fclose` позволяет обеспечить сохранность информации в файле.

10.4.2. Чтение данных из файла

В MuPAD ***ввод (чтение) данных из файла*** осуществляется с помощью функций `ftextinput` и `finput`.

Функция `ftextinput` читает информацию, хранящуюся в текущей строке файла, и возвращает ее в программу в строковом виде. Возможны два варианта вызова функции `ftextinput`:

```
<Переменная> := ftextinput(<идентификатор файла>,
                                                <опция>)
ftextinput(<идентификатор файла>, <переменная1>,
                          <переменная2>, …, <опция>)
```

В первом случае функция `ftextinput` читает одну строку из файла, обозначенного ранее в программе переменной, указанной в качестве параметра `<идентификатор файла>`, начиная с позиции, в которой расположен указатель файла, переводит указатель файла на следующую строку, преобразует полученную информацию в строковый формат и присваивает `<переменной>`, идентификатор которой указан слева от функции `ftextinput` перед символом `:=`.

Во втором случае функция `ftextinput` читает из файла, обозначенного ранее в программе переменной, указанной в качестве параметра `<идентификатор файла>`, одну строку за другой, начиная с позиции, в которой расположен указатель файла. Каждая строка последовательно присваивается одной из перечисленных при вызове функции `ftextinput` переменных (`<переменная1>`, `<переменная2>`, …). Функция `ftextinput` считывает из файла столько строк, сколько переменных указано при ее вызове, и соответствующим образом изменяет положение указателя файла.

Например, если текстовый файл открыт на чтение и обозначен ранее в программе переменной `fDat`, то с помощью команды `titul1 := ftextinput(fDat)` можно прочитать одну строку из этого файла; после

выполнения этой команды переменная titul1 будет содержать прочитанную из файла информацию в строковом формате.

Функция ftextinput имеет только одну <опцию> Encoding = <кодировка>, с помощью которой можно выбрать кодировку. Доступные значения параметра <кодировка> приведены в пояснениях к функции fopen (см. разд. 10.4.1).

Чтобы прочитать из файла данные других типов, кроме строкового, используют функцию finput, для вызова которой также существует два способа:

```
<переменная> := finput(<идентификатор файла>,
                                        <опция>)
finput(<идентификатор файла>, <опция>,
                  <переменная1>, <переменная2>, …)
```

В первом случае функция finput читает одно значение из файла, обозначенного ранее в программе переменной, указанной в качестве параметра <идентификатор файла>, и присваивает это значение <переменной>, идентификатор которой указан слева от функции finput перед символом :=.

Во втором случае функция finput читает из файла, обозначенного ранее в программе переменной, указанной в качестве параметра <идентификатор файла>, одно значение за другим, начиная с текущего положения указателя файла. Каждое значение последовательно присваивается одной из перечисленных при вызове функции finput переменных (<переменная1>, <переменная2>, …). Функция finput считывает из файла столько значений, сколько переменных указано при ее вызове.

Например, с помощью команды finput(fDat, x1, x2) можно прочитать значения двух переменных x1 и x2 из файла, который был ранее открыт на чтение и обозначен идентификатором fDat.

Функция finput также имеет только одну <опцию>, с помощью которой можно выбрать кодировку. Правила использования и допустимые значения этой <опции> совпадают с правилами использования и значениями <опции> функции fopen.

Выбор функций ftextinput или finput, а также их последовательность и параметры определяются структурой данных в файле. Например, пусть текстовый файл d:\work\dat1.txt содержит следующие данные:

```
[4.78, 5.34, 8.91, 10.14, 17.77, 13.24, 11.15, 9.67]
```

Тогда с помощью последовательности команд:

```
F := fopen("d:\\work\\dat1.txt", Read):
     S := finput(F):
fclose(F):
S
```

можно в программу загрузить (прочитать) из файла список `S`, содержащий эти данные. В результате выполнения данного программного кода в поле вывода появится значение переменной `S` в виде: [4.78, 5.34, 8.91, 10.14, 17.77, 13.24, 11.15, 9.67].

Если текстовый файл `d:\work\dat2.txt` содержит следующую информацию:

```
Исходные данные:
5.24; 3/2
```

тогда с помощью последовательности команд:

```
F := fopen("d:\\work\\dat2.txt", Read):
     Z := ftextinput(F):
     finput(F, x1, x2):
fclose(F):
Z, x1, x2
```

можно в программу загрузить (прочитать) из файла строковое значение переменной `Z` и числовые значения переменным `x1` и `x2`. В результате выполнения этого программного кода в поле вывода появится: "Исходные данные:", 5.24, $\frac{3}{2}$.

10.4.3. Запись данных в файл

В MuPAD ***вывод (запись) данных в файл*** осуществляется с помощью функции `fprint`, которую вызывают следующим образом:

fprint(<опции>, <формат>, <идентификатор файла>,
<переменная1>, <переменная2>, …)

<Идентификатор файла>, полученный в результате выполнения функции `fopen`, указывает, в какой файл необходимо записывать информацию (см. разд. 10.4.1).

В качестве параметров <переменная1>, <переменная2>, … могут быть указаны идентификаторы одной или нескольких заданных ранее в программе переменных и (или) одно или несколько выражений. Если переменных (выражений) несколько, они перечисляются через запятую.

Кроме того, вместо любого из параметров <переменная1>, <переменная2>, … можно указать числовое значение или текст (набор символов в кавычках).

Параметр <формат> можно указать, используя один из двух стандартных идентификаторов: Bin – для записи информации в двоичном формате; Text – для записи информации в текстовом формате. Параметр <формат> не является обязательным.

Функцию fprint можно вызвать со следующими <опциями>: Unquoted или NoNL – позволяет сохранять строковые переменные без кавычек и с учетом управляющих символов (\n – новая строка; \t – табуляция; \b – разрыв строки); Encoding = <кодировка> – позволяет выбрать нужную кодировку (доступные значения параметра <кодировка> приведены в пояснениях к функции fopen). Использование <опций> тоже не является обязательным.

После выполнения функции fprint в поле вывода ничего не отображается.

Количество, структура и последовательность операторов fprint в программе может быть различной в зависимости от того, какую информацию и как необходимо разместить во внешнем файле.

Например, если в текстовом файле d:\work\res1.txt необходимо сохранить список, обозначенный в программе переменной S, то можно использовать следующий набор команд:

```
S := [4.78,  5.34,  8.91, 10.14, 17.77, 13.24,
     11.15,  9.67,  5.08,  3.11,  7.32, 10.01];
F := fopen("d:\\work\\res1.txt", Write, Text):
    fprint(F, S);
fclose(F):
```

Тогда после выполнения этого программного кода файл d:\work\res1.txt будет выглядеть следующим образом:

```
[4.78, 5.34, 8.91, 10.14, 17.77, 13.24, 11.15, 9.67,
5.08, 3.11, 7.32, 10.01]:
```

Если в текстовом файле d:\work\res2.txt необходимо сохранить значения переменных x, y, z (вещественные координаты) и nx, ny, nz (целые числа), заданные в программе, сопроводив их текстовыми пояснениями, то для этого можно использовать следующий набор команд:

```
x := 4.78: y := 5.34: z := 8.91:
nx := 20: ny := 55: nz := 17:
```

```
FL := fopen("d:\\work\\res2.txt", Write, Text):
   fprint(Unquoted, FL, "Координаты x, y, z");
   fprint(FL, x, y, z);
   fprint(NoNL, FL, " nx = ", nx);
   fprint(Unquoted, FL, " ny = ", ny);
   fprint(Unquoted, FL, " nz = ", nz);
fclose(FL):
```

Тогда после выполнения этого программного кода файл `d:\work\res2.txt` будет выглядеть следующим образом:

```
Координаты x, y, z
4.78:5.34:8.91:
 nx = 20 ny = 55
 nz = 17
```

Обратите внимание! Когда в файл выводятся подряд значения нескольких числовых переменных, они разделяются двоеточием (:).

10.4.4. Сохранение в файле и загрузка из файла переменных MuPAD

В процессе выполнения расчетов бывает важно сохранить не только вычисленные значения, но и обозначения соответствующих им переменных. С помощью функции `write` можно сохранить идентификаторы переменных и присвоенные им символьные выражения в формате, используемом MuPAD. Для этого функцию `write` вызывают следующим образом:

```
write(<формат>, <опция>, <идентификатор файла>,
                    <переменная1>, <переменная2>, …)
```

Параметр `<идентификатор файла>`, полученный ранее в программе в результате выполнения функции `fopen`, указывает, в какой файл необходимо записывать информацию (см. разд. 10.4.1).

В качестве параметров `<переменная1>`, `<переменная2>`, … могут быть указаны идентификаторы одной или нескольких заданных ранее в программе переменных. Если переменных несколько, они перечисляются через запятую.

Необязательный параметр `<формат>` можно указать, используя один из двух стандартных идентификаторов: `Bin` – для записи информации в двоичном формате; `Text` – для записи информации в текстовом формате.

test_write* - MuPAD

File Edit View Navigation Insert Format Notebook

```
f := x -> 10/((x+1)^2+1)+20/((x-3)^2+1)+x/5-10
```

$$x \to \frac{10}{(x+1)^2+1} + \frac{20}{(x-3)^2+1} + \frac{x}{5} - 10$$

```
y1:= f(-3): y2:= f(2): y3:= f(5): Y:= f(z+1):
y1, y2, y3, Y
```

$$-\frac{1491}{185}, \frac{7}{5}, -\frac{175}{37}, \frac{z}{5} + \frac{20}{(z-2)^2+1} + \frac{10}{(z+2)^2+1} - \frac{49}{5}$$

```
FL := fopen("d:\\work\\vars.txt", Write, Text):
    write(FL, f, y1, y2, y3, Y);
fclose(FL)
```

Mem 4 MB, T 0 s Cmd INS

vars.txt — Блокнот

Файл Правка Формат Вид Справка

```
f := x -> 10/((x + 1)^2 + 1) + 20/((x - 3)^2 + 1) + x/5 - 10:
y1 := -1491/185:y2 := 7/5:y3 := -175/37:Y := hold(_plus)(hold
(_divide)(hold(z), 5), hold(_divide)(20, hold(_plus)(hold
(_power)(hold(_plus)(hold(z), -2), 2), 1)), hold(_divide)(10,
hold(_plus)(hold(_power)(hold(_plus)(hold(z), 2), 2), 1)), -
49/5):
```

test_fread* - MuPAD

File Edit View Navigation Insert Format Notebook

```
FL := fopen("d:\\work\\vars.txt", Read):
    fread(FL):
fclose(FL)
```

```
f;
y1, y2, y3, Y
```

$$x \to \frac{10}{(x+1)^2+1} + \frac{20}{(x-3)^2+1} + \frac{x}{5} - 10$$

$$-\frac{1491}{185}, \frac{7}{5}, -\frac{175}{37}, \frac{z}{5} + \frac{20}{(z-2)^2+1} + \frac{10}{(z+2)^2+1} - \frac{49}{5}$$

Mem 4 MB, T 0 s Cmd INS

Рис. 10.4. Сохранение в файле и загрузка из файла переменных

Если для записи используется текстовый формат, то информация записывается в следующем виде:

<переменная1> := hold(<значение1>): и т.д.

Функцию write можно вызвать только с одной <опцией>: Encoding = <кодировка>, которая позволяет выбрать нужную кодировку (доступные значения параметра <кодировка> приведены в пояснениях к функции fopen).

После выполнения функции write в поле вывода ничего не отображается.

На рис. 10.4 приведен пример использования функции write для сохранения в файле переменных и их значений, а также содержание созданного при этом файла.

Сохраненные в файле команды можно читать из этой же или другой программы с помощью функции fread, которую вызывают следующим образом:

fread(<идентификатор файла>, <опции>)

Параметр <идентификатор файла>, полученный ранее в программе в результате выполнения функции fopen, указывает, из какого файла необходимо читать информацию (см. разд. 10.4.1).

Функция fread загружает из файла все идентификаторы и соответствующие им значения переменных. Если для получения какого-либо из загруженных значений необходимо выполнить какие-то вычисления, они сразу же выполняются.

Функцию fread можно вызвать со следующими <опциями>: Plain – позволяет игнорировать заданные в текущем файле ограничения свойств для переменных, загружаемых из файла; Quiet – подавляет вывод результатов выполнения вычислений загруженных переменных (кроме оператора print); Encoding = <кодировка> – позволяет выбрать нужную кодировку (доступные значения параметра <кодировка> приведены в пояснениях к функции fopen).

После выполнения функции fread, вызванной без опции Quiet, в поле вывода отображаются значения последней загруженной их файла переменной.

На рис. 10.4 приведен пример использования функции fread для загрузки из файла сохраненных в нем переменных и их значений.

10.4.5. Сохранение в текстовом файле и загрузка из него команд MuPAD

В создаваемых в MuPAD mn-файлах команды хранятся в текстовом формате, а результаты их выполнения – в графическом. Если необходимо сохранить только команды, соответствующие какой-либо программе или ее части, можно воспользоваться редактором m-файлов (Editor), который можно вызвать из MuPAD с помощью пункта `New Editor` из раздела `File` главного меню или с помощью одноименной кнопки на стандартной панели инструментов (см. разд. 1.3). В результате появится стандартное окно редактора m-файлов MatLab. Команды MuPAD рекомендуется хранить в файлах с расширением mu. Если с помощью редактора m-файлов сохранить или открыть файл с расширением mu, то панель кнопок `Run`, которая обычно в редакторе m-файлов располагается на вкладке `Editor` панели инструментов и предназначена для запуска программ на выполнение, исчезнет. Тогда окно редактора m-файлов примет вид, представленный на рис. 10.5 (для MatLab версии 2012 или выше).

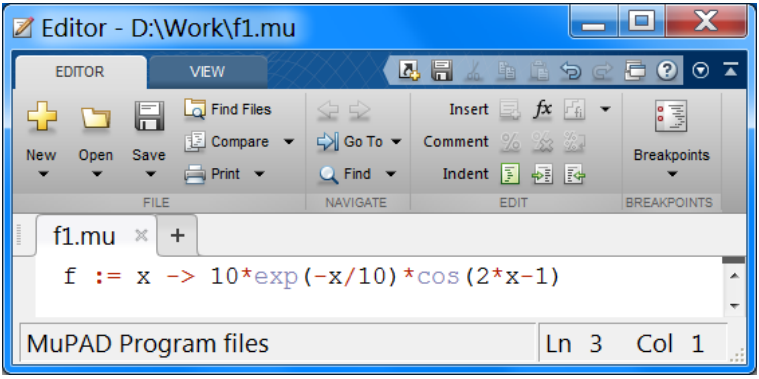


Рис. 10.5. Редактор m-файлов с открытым mu-файлом

Текст команд в mu-файл можно как записывать непосредственно, так и копировать из открытого в MuPAD mn-файла. Если необходимо скопировать команды, расположенные в разных командных полях, перед копированием рекомендуется удалить все соответствующие им поля вывода, например, с помощью пункта `Delete All Outputs` из раздела `Edit`

главного меню MuPAD. Если планируется, что редактируемый mu-файл будет содержать более одной команды, следует помнить, что в последующем все записанные в этот файл команды будут вызываться в одном и том же командном поле, поэтому их необходимо отделить друг от друга одним из символов `:` или `;` (см. разд. 2.6).

Информация в mu-файлах хранится (так же как и в m-файлах) в текстовом формате.

Сохраненные в mu-файле команды можно загрузить mn-файл с помощью функции `read`, которую вызывают следующим образом:

read("<имя файла>", <опции>)

В качестве параметра <имя файла> необходимо, так же как при использовании функции `fopen` (см. разд. 10.4.1), указать название файла, которое должно включать расширение и путь к файлу (если файл находится в той же папке, что и использующая его программа, путь к файлу можно не указывать). Функция `read` открывает указанный файл на чтение, вызывает функцию `fread` для чтения команд из файла, закрывает его и выполняет загруженные команды. Поэтому <опции> функции `read` совпадают с <опциями> функции `fread` (см. разд. 10.4.4). Тот же результат можно получить, вызвав вместо функции `read` последовательно функции `fopen`, `fread` и `fclose` с соответствующими параметрами.

Создание mu-файлов позволяет сохранять и быстро воспроизводить часто повторяющиеся последовательности команд в разных программах.

10.5. Индексированные переменные

MuPAD позволяет любую переменную сделать индексированной (см. разд. 2.2). При этом разные варианты написания индексов приводят к созданию переменных, обладающих различными свойствами. Одни индексированные переменные (так же как и неиндексированные) могут быть ассоциированы только с одним единственным объектом (числовым значением, выражением, уравнением, функцией, графическим объектом и т.д.), другие – с несколькими объектами. В первом случае индекс – это один из символов, используемый для записи идентификатора переменной, который отличается от других символов способом его отображения в поле вывода. Для изменения способа отображения отдельных символов в идентификаторах используются специальные указатели (см. разд. 2.2), одним

из которых является символ _. Например: `f_1`, `f_2` и т.д. Автоматизировать изменение таких индексов в программе нельзя.

Если одной переменной поставлено в соответствие несколько объектов, то индексы – это самостоятельные переменные или константы, с помощью которых осуществляется доступ к отдельным объектам в рамках данной переменной. К таким индексированным переменным относятся строки, последовательности, списки, множества, массивы, таблицы. Индексы у таких переменных указывают в квадратных скобках. Например: `s[k]`, `A[1, 3]`, `L[2][1]` и т.д.

Индексированные переменные, содержащие несколько объектов, в зависимости от их типа создаются и используются по-разному.

10.5.1. Строки, последовательности, списки и множества

В MuPAD ***строкой*** считается совокупность символов, заключенных в кавычки. Например: `"frkfc epsn"`. Переменная, которой присвоена строка, считается строковой. Символы в строковых переменных хранятся последовательно друг за другом и называются элементами строки. Все элементы строки имеют одинаковую длину, равную единице. Строковые переменные считаются индексированными, потому что к каждому элементу строки можно обратиться, указав его порядковый номер в квадратных скобках после идентификатора соответствующей строковой переменной. Например, если в переменной `t` хранится строка `"rewuip"`, то с помощью команды `t[4]` можно получить `"u"`. Информация о номере элемента в строке не хранится, а при обращении к элементу с заданным номером пересчитываются элементы, начиная с первого.

Последовательности и ***списки*** – совокупности объектов (элементов) разной длины, которые располагаются один за другим и отделяются дуг от друга специальными разделителями. Последовательности, как правило, содержат однотипные элементы, а списки могут содержать элементы, относящиеся к разным типам. И список, и последовательность могут содержать повторяющиеся элементы.

В MuPAD последовательность образуется путем перечисления ее элементов через запятую, элементы списка также перечисляются через запятую, но в квадратных скобках. Например,

последовательность чисел: 3, 8, 0, –2, 4, 12, 3, 5

список: $[x - 1, x^2, y + x, 5x, -2y, 30]$

Одним из наиболее распространенных способов автоматического создания последовательности выражений (чисел) является использование оператора $ (см. разд. 4.2, 6.3, 6.4.1), который возвращает <последовательность> результатов подстановки <значений> <переменной> в заданное <выражение>:

<Последовательность> := <выражение> **$**
<переменная> <значения>

Как было показано ранее (см. разд. 4.2, 6.3, 6.4.1), применяя оператор $, после идентификатора <переменной> набор <значений> можно записать двумя способами: 1) <переменная> = <начало>..<конец> step <шаг> – указав <шаг> и диапазон изменения <переменной> (например, в виде: k=1..5 step 1/2); 2) <переменная> in <множество> – указав <множество>, значения из которого необходимо подставлять в <выражение> вместо <переменной> (например, в виде x in {3, 7, 9}).

Тогда, используя оператор $, с помощью команды k*x+1 $ k = –1..3 можно, например, получить последовательность $1 - x$, 1, $x + 1$, $2x + 1$, $3x + 1$.

Для автоматического создания списков чаще всего используют преобразование <последовательности> в список с помощью квадратных скобок:

<Список> := **[**<последовательность>**]**

Например, если переменной p обозначена последовательность p := 1-x, 1, x+1, 2*x+1, 3*x+1, то с помощью команды [p] можно получить список $[1 - x, 1, x + 1, 2x + 1, 3x + 1]$, состоящий из тех же элементов.

Объекты в списках и последовательностях не пронумерованы. Вместе с каждым объектом (элементом) хранится ссылка на следующий объект (элемент). Поэтому пустых (незаданных) элементов в последовательностях и списках не бывает, но бывают пустые списки [], которые не содержат ни одного элемента. Тем не менее для обращения к элементу списка (или последовательности) используется его порядковый номер. Этот порядковый номер и называется индексом элемента последовательности (списка) и указывается в квадратных скобках после идентификатора переменной. Например, s[1], p[3]. При обращении к элементу последовательности (списка) с заданным индексом (номером) последовательно перебираются все элементы, начиная с первого, и подсчитывается

количество разделителей между ними. В результате выбирается тот элемент, который расположен перед разделителем с заданным номером. Например, для списка `SP := [3, x-2, 5+y, 4, x*y]` команда `SP[3]` возвращает элемент: $5 + y$. Как было показано ранее (см. разд. 4.5), этот же результат можно получить, написав команду `op(SP, 3)`. Если элемент списка или последовательности, в свою очередь, состоит из нескольких объектов, например является списком или выражением, то для обращения к подэлементу этого элемента после идентификатора списка (последовательности) необходимо указать сначала индекс элемента, а затем – индекс подэлемента, в квадратных скобках каждый. Например, для списка `SX := [x, 4, 6, [3*x, 2*x, 7], 9]` с помощью команды `SX[4][2]` можно получить выражение $2x$. Тот же результат можно получить с помощью функции `op` следующим образом: `op(op(SX, 4), 2)`.

Множество – это совокупность неповторяющихся однотипных объектов (элементов). Принцип нумерации элементов множества такой же, как и элементов списка (номер элемента определяется количеством предшествующих ему разделителей). В MuPAD множество можно задать, перечислив его элементы в фигурных скобках через запятую. Повторяющиеся элементы автоматически удаляются из множества. При отображении в поле вывода элементы множества сортируются по возрастанию. При этом их расположение во множестве не изменяется. Поэтому при обращении к элементу множества через указание его индекса в квадратных скобках и через функцию `op` можно получить разные результаты. Например, после выполнения команды `S := {3, 9, 1, 4}` в поле вывода это множество примет вид: {1, 3, 4, 9}. Тогда с помощью команды `S[2]` получится число 3, а с помощью команды `op(S, 2)` – число 9. Чтобы избежать подобных недоразумений, перед обращением к элементам множества рекомендуется упорядочить (отсортировать) элементы множества, например с помощью функции `sort` (см. разд. 10.5.3).

Для автоматического создания множеств часто также используют преобразование `<последовательности>`, но с помощью фигурных скобок:
`<Множество> := {<последовательность>}`

При этом если исходная последовательность содержала несколько одинаковых элементов, то в полученном из нее множестве этот элемент будет встречаться только один раз. Например, если переменной `p`

обозначена последовательность `p := 1-x, x, x+1, x, 2*x+1, x, 3*x+1`, то с помощью команды `{p}` можно получить множество $\{x, 1 - x, 2x + 1, 3x + 1, x + 1\}$.

10.5.2. Массивы и таблицы

Массивы – это совокупность определенным образом пронумерованных объектов (элементов). Нумерация элементов в массиве осуществляется только с помощью индексов, принимающих целочисленные значения. Кроме значения элемента в массиве также хранится и его индекс (индексы). Поэтому массив может содержать незаполненные элементы. Для нумерации элементов в массиве могут использоваться один, два и более индексов. Если для нумерации элементов используется один индекс, массив называется одномерным, если два индекса – двумерным и т.д. Тогда для обращения к элементу массива необходимо в квадратных скобках указать один, два или более индексов соответственно. В этом смысле последовательности, списки и множества можно считать одномерными индексированными переменными.

Чтобы создать массив, необходимо использовать функцию `array` следующим образом:

```
<Переменная> := array(<min индекс1>..<max индекс1>,
                      <min индекс2>..<max индекс2>,
                      <min индекс3>..<max индекс3>, …,
                                  <список значений>)
```

С помощью параметров `<min индекс1>..<max индекс1>`, `<min индекс2>..<max индекс2>`, `<min индекс3>..<max индекс3>`, … указывают минимальное и максимальное значения каждого индекса (целочисленные значения). В результате создаваемый массив будет содержать элементы с индексами, каждый из которых принимает только целые значения в указанных границах (через единицу). Если создается одномерный массив, то необходимо указать граничные значения `<min индекс1>..<max индекс1>` только для одного индекса. Если создается двумерный массив, то необходимо указать граничные значения `<min индекс1>..<max индекс1>`, `<min индекс2>..<max индекс2>` для двух индексов и т.д. Индексы могут принимать как положительные, так и отрицательные (целочисленные) значения.

С помощью параметра `<список значений>` можно задать значения элементов создаваемого массива. Если параметр `<список значений>`

отсутствует, создается пустой массив. В поле вывода вместо каждого из элементов такого массива будет написано NIL. Если в процессе создания массива будут заданы значения всех элементов массива, то параметр <список значений> можно указать в виде списка, количество элементов и подэлементов которого должно совпадать с количеством элементов создаваемого массива. Например: array(3..5, [x, 3, x-2]) – одномерный массив; array(-1..1, 0..2, [1, 2, 3, 4, 5, 6, 7, 8, 9]) – двумерный массив; array(2..3, 3..4, 4..5, [1, 2, 3, 4, 5, 6, 7, 8]) – трехмерный массив и т.д. Если размерность массива больше единицы (для нумерации индексов используется два или более индексов), то каждый элемент <списка значений>, в свою очередь, тоже может являться списком. Тогда количество элементов во внешнем списке должно совпадать с количеством возможных значений первого индекса, а количество элементов внутреннего списка – с количеством элементов последнего индекса. Например: array(-1..1, 0..2, [[1, 2, 3], [4, 5, 6], [7, 8, 9]]) – двумерный массив; array(2..3, 3..4, 4..5, [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]) – трехмерный массив и т.д.

Если при создании массива заданы все его элементы и их расположение (с помощью вложенных списков), диапазоны индексов можно не указывать. Тогда нумерация каждого индекса по умолчанию начинается с единицы. Например, индекс одномерного массива array([x, 3, x-2]) и каждый из индексов двумерного массива array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) будут принимать значения от 1 до 3.

Кроме явного перечисления элементов, списки, из которых формируется массив, могут быть созданы автоматически, любыми средствами, доступными для создания списков, например с помощью оператора $: array(3..5, [x^k $ k=-1..1]).

Если при создании массива необходимо задать только некоторые элементы, то параметр <список значений> можно задать в виде последовательности следующего вида:

(<индексы элемента1>) = <значение элемента1>,
(<индексы элемента2>) = <значение элемента2>, …

Индексы задаваемых элементов необходимо указывать в круглых скобках через запятую. Если для нумерации элементов используется только один индекс (одномерный массив), то круглые скобки можно опустить. Количество индексов должно соответствовать размерности массива.

В этом случае все элементы, кроме перечисленных, останутся незаданными. Например: `array(1..8, 2 = x, 5 = x-2)` – одномерный массив; `array(1..5, 1..4, (1, 2) = 3, (5, 4) = 6)` – двумерный массив; `array(1..3, 2..4, 3..5, (1, 2, 3) = 4, (3, 3, 3) = 8)` – трехмерный массив и т.д.

Функция `array` позволяет создать массив произвольной конфигурации (размерности и типа элементов). В MuPAD также существует несколько функций, которые позволяют создавать массивы специального вида. Например, в главе 7 рассмотрено создание и использование матриц – двумерных массивов, нумерация элементов которых всегда начинается с единицы. Для создания вещественных массивов, значения элементов которых хранятся как вещественные числа в десятичном формате, можно использовать функцию `hfarray`, синтаксис которой аналогичен синтаксису функции `array`.

В поле вывода одномерные массивы выводятся как вектор-строка, двумерные – как матрица, а трехмерные и выше – поэлементно с указанием значений индексов.

Для обращения к элементу массива необходимо после идентификатора переменной, которой обозначен массив, указать список соответствующих данному элементу значений индексов (в квадратных скобках через запятую). Например, если переменной `A` обозначен одномерный массив, то для обращения к одному его элементу необходимо указать значение одного индекса: `A[3]`, `A[6]`, `A[k]`. Если переменной `B` обозначен двумерный массив, то для обращения к одному его элементу необходимо указать значения двух индексов: `B[3, 2]`, `B[1, 4]`, `B[k, p]` и т.д. (`k`, `p` – переменные, которым ранее были присвоены целочисленные значения). Значения индексов должны соответствовать указанным при объявлении массива диапазонам.

В MuPAD под ***таблицей*** понимают индексированную совокупность объектов, в качестве индексов которых можно использовать незаданные ранее символьные переменные. Таблицы создаются поэлементно. Чтобы создать элемент таблицы, необходимо присвоить ему значение. Для обращения к элементу таблицы (в том числе и для присвоения значения) также необходимо после идентификатора переменной указать соответствующий список индексов в квадратных скобках. Например: `T[a, b]`, `U[m] := x + 9` (`a`, `b`, `m` – заданные или незаданные ранее символьные или числовые переменные). Таблицы не имеют жесткой структуры и не требуют предварительного объявления. Поэтому любая не объявленная ранее

индексированная переменная, индекс (индексы) которой указан в квадратных скобках (даже если индексом является целое число), по умолчанию считается таблицей. Если в командном поле указать только идентификатор таблицы (без индексов), то в поле вывода все заданные ранее элементы будут представлены в виде таблицы, состоящей из двух столбцов, в одном из которых перечислены все индексы, а в другом – соответствующие им значения элементов таблицы.

10.5.3. Преобразование индексированных переменных

Определить <количество элементов> (длину) строки можно с помощью функции `length`, а <количество элементов> последовательности, списка, множества, массива – с помощью функции `nops`, в качестве параметра которых необходимо указать <имя> – идентификатор соответствующей индексированной переменной:

<Количество элементов> := **length**(<имя>)

<Количество элементов> := **nops**(<имя>)

Например, если задан список `s := [x, 3, x-2]`, то с помощью команды `nops(s)` можно получить длину этого списка, равную трем. Если задан двумерный массив `Z := array(1..5, 1..4, (1, 2) = 3, (5, 4) = 6)`, то с помощью команды `nops(Z)` можно получить количество элементов этого массива, равное 20.

Преобразовать список, множество или массив в последовательность можно с помощью оператора `$`. Для этого вместо <выражения> перед оператором `$` необходимо указать элемент списка, множества, вектора, матрицы или массива соответственно с символьными переменными вместо индексов, а после оператора `$` – идентификатор и границы изменения индексов этих элементов:

<Последовательность> := <имя>[**k1**] **$**
k1 = <начало1>..<конец1>

<Последовательность> := <имя>[**k1, k2**] **$**
k1 = <начало1>..<конец1> **$ k2** = <начало2>..<конец2>

Здесь <имя> – идентификатор индексированной переменной; <начало1> и <конец1> – минимальное и максимальное значения первого индекса (для списка, множества, вектора <начало1> равно 1, а <конец1> – количеству элементов соответствующей переменной), <начало2> и <конец2> – минимальное и максимальное значения

второго индекса. Например, массив `X := array(3..5,[x, y, z])` с помощью команды `X[k] $ k=3..5` можно преобразовать в последовательность *x*, *y*, *z*.

Как было описано выше, последовательность можно преобразовать в индексированную переменную любого другого вида. Поэтому преобразования между другими типами индексированных переменных осуществляют через преобразование в последовательность (как промежуточный этап).

Для объединения двух последовательностей в одну достаточно их записать через запятую. В остальных случаях, чтобы объединить две одинаковые по типу индексированные переменные (<переменная1> и <переменная2>) по горизонтали, используют оператор **.** (точка):

<Результат> := <переменная1>.<переменная2>

Если объединяют двумерные массивы, то количество строк (количество возможных значений первого индекса) <переменной1> и <переменной2> должно быть одинаковым.

Например, если объявлены две последовательности `x1 := 1, 2, 3` и `x2 := a, b, c`, то с помощью команды `x1, x2` можно получить последовательность 1, 2, 3, *a*, *b*, *c*. Если объявлены два списка `y1 := [1, 2, 3]` и `y2 := [a, b, c]`, то с помощью команды `y1.y2` можно получить список [1, 2, 3, *a*, *b*, *c*] и т.д.

Чтобы извлечь (скопировать) один элемент строки, последовательности, списка, множества, массива, можно после его идентификатора <имя> в квадратных скобках указать числовое значение каждого его индекса или использовать функцию `op`:

<Результат> := <имя>**[**<число>**]**

<Результат> := <имя>**[**<число1>, <число2>**]**

<Результат> := op(<имя>, <число>)

При этом исходная строка, последовательность, список, множество или массив не изменится.

Например, если объявлен список `p := [1, 2, 3, a, b, c]`, то в результате выполнения команды `p[5]` получится *b*. Выполнив команду `op(p, 5)`, можно получить тот же результат.

Чтобы извлечь (скопировать) часть строки, последовательности, списка, множества, массива, можно после его идентификатора <имя>

в квадратных скобках вместо одного значения каждого индекса указать диапазон значений или аналогичным образом использовать функцию op:

```
<Результат> := <имя>[<начало>..<конец>]
<Результат> := <имя>[<начало1>..<конец1>,
                                  <начало2>..<конец2>]
<Результат> := op(<имя>, <начало>..<конец>)
```

При этом исходная строка, последовательность, список, множество или массив не изменится. Если для извлечения нескольких элементов используется функция op, следует помнить, что в результате будет получаться последовательность независимо от того, из какой индексированной переменной извлекаются элементы.

Например, если объявлен список `p := [1, 2, 3, a, b, c]`, то в результате выполнения команды `p[3..5]` получится список [3, *a*, *b*], а в результате выполнения команды `op(p, 3..5)` получится последовательность 3, *a*, *b*.

Чтобы изменить значение одного элемента строки, последовательности, списка, множества, массива, матрицы или вектора достаточно присвоить ему <новое значение> следующим образом:

```
<Имя>[<число>] := <новое значение>
<Имя>[<число1>,<число2>] := <новое значение>
```

Для этого перед оператором присвоения := необходимо указать идентификатор <имя> соответствующей индексированной переменной и в квадратных скобках числовые значения каждого индекса изменяемого элемента. В результате исходная индексированная переменная изменится, но количество элементов в ней останется прежним. Например, если объявлен список `p := [1, 2, 3, a, b, c]`, в результате выполнения команды `p[2] := 10-x` исходный список p изменится и примет вид [1, 10 – *x*, 3, *a*, *b*, *c*].

Несколько расположенных подряд элементов строки, последовательности, списка или множества можно заменить одним значением:

```
<Имя списка>[<начало>..<конец>] := [<новое значение>]
<Имя множества>[<начало>..<конец>] :=
                                  {<новое значение>}
<Имя строки>[<начало>..<конец>] := "<новое значение>"
```

При этом количество элементов соответствующей индексированной переменной уменьшается. Применение этой операции к элементам

массива, матрицы, вектора не допускается. Например, если объявлен список `p := [1, 2, 3, a, b, c]`, в результате выполнения команды `p[3..5]` получится новый список [3, *a*, *b*], а исходный список не изменится; в результате выполнения команды `p[3..5] := [10]` исходный список `p` изменится и примет вид [1, 2, 10, *c*].

Если присвоить значение элементу с индексом, номер которого превышает существующее количество элементов, то можно увеличить количество элементов строки, списка, множества или таблицы. Например, если объявлен список `z := [a, b, c]`, то после выполнения команды `z[4..4] := [g]` получится список [*a*, *b*, *c*, *g*]. Увеличение размеров массива, матрицы или вектора таким способом не допускается. Индексы элементов массива, матрицы или вектора всегда должны соответствовать диапазону, указанному при его объявлении с помощью функции `array`, `matrix` или `hfarray`.

С помощью команды `delete` можно удалять элементы индексированных переменных. При этом длина строки, последовательности, списка или множества уменьшается, а размер массива не изменяется. Если удалить элемент массива, объявленного с помощью `array`, то этот элемент окажется незаданным (`NIL`). Для матриц (`matrix`) удаление элементов с помощью команды `delete` не допускается. Например, если объявлен список `p := [1, 2, 3, a, b, c]`, то после выполнения команды `delete p[2]` он примет вид: [1, 3, *a*, *b*, *c*]. А если объявлен массив `P := array([1, 2, 3, a, b, c])`, то после выполнения команды `delete P[2]` он примет вид: (1, NIL, 3, *a*, *b*, *c*).

С помощью функций `min` и `max` можно узнать наименьшее и наибольшее значения из элементов индексированных переменных. Для этого функции `min` и `max` вызывают следующим образом:

`<Результат> :=` **`min`**`(<значения>)`

`<Результат> :=` **`max`**`(<значения>)`

В качестве параметра `<значения>` можно использовать последовательность, список, множество или `hfarray`-массив, записанные явным образом или через идентификаторы переменных, которыми они были ранее обозначены. `<Результатом>` поиска наибольшего или наименьшего значения будет являться одно числовое значение или символьное выражение, если для всех анализируемых `<значений>` определены операции сравнения. Сравнение выполняется с учетом свойств символьных переменных, используемых для записи анализируемых `<значений>`.

Если MuPAD не может выбрать в качестве наибольшего (наименьшего) единственное из указанных значений, то в поле вывода отображается запись поставленной задачи через функции `max` или `min` соответственно, как при отложенных вычислениях. Например, в результате выполнения команды `min([3, 9, 4, 1, 8])` получится число 1; в результате выполнения команды `max([3, 9, 4, 1, 8])` – число 9; а в результате выполнения команды `min(x - 3, x + 9, x - 4, x - 1, x + 8) assuming(x > 0)` после упрощения `simplify(%)` получится выражение $x - 4$. Если хотя бы одно из анализируемых <значений> является комплексным (с ненулевой мнимой частью), то поиск наименьшего и наибольшего значения не выполняется, выводится сообщение об ошибке.

Для некоторых индексированных переменных существуют дополнительные функции для выполнения отдельных операций. Например, с помощью функции `append` можно добавить элементы к списку, с помощью функции `sort` упорядочить элементы списка по возрастанию, а с помощью функции `revert` расположить элементы списка в обратном порядке:

```
<Результат> := append(<список>, <новый элемент1>,
                                    <новый элемент2> ...)
<Результат> := sort(<список>)
<Результат> := revert(<список>)
```

Если с помощью функции `sort` упорядочить не список, а множество, то получится список.

С помощью функции `union` можно объединить два множества (<множество1> и <множество2>), а с помощью функции `card` определить размер множества:

```
<Результат> := <множество1> union <множество1>
<Результат> := card(<множество>)
```

Примеры: в результате выполнения команды `append([4, 5, 6], 1, 10)` получится список [4, 5, 6, 1, 10]; в результате выполнения команды `sort([4, 5, 6, 1, 10])` получится список [1, 4, 5, 6, 10]; в результате выполнения команды `{4, 5, 6} union {1, 10}` получится множество {1, 4, 5, 6, 10}.

Сумму и произведение элементов множества можно вычислить с помощью описанных ранее функций `sum` и `product` (см. разд. 6.1) соответственно:

```
<Результат> := sum(x, x in <множество>)
```

`<Результат> := product(x, x in <множество>)`

Например, в результате выполнения команды `sum(x, x in {x^2, 2*x, 5})` получится сумма элементов множества: $x^2 + 2x + 5$.

Однако следует отметить, что функции `sum` и `product` (см. разд. 6.1) нельзя применять для поэлементного суммирования или перемножения других индексированных объектов (кроме множества). Функции `sum` и `product` предназначены для вычисления суммы и произведения различного рода прогрессий (арифметических, геометрических и т.д.). Поэтому они анализируют характер этих прогрессий, вычисляют соответствующие им параметры, первый и последний члены и по ним находят сумму или произведение. Элементы индексированных переменных (кроме множества), как правило, не обладают такими свойствами, их необходимо складывать или перемножать поэлементно. Для этого можно преобразовать индексированную переменную в последовательность, а далее применить один из следующих операторов:

`<Результат> := `+`(<последовательность>)`

`<Результат> := `*`(<последовательность>)`

или использовать оператор цикла (см. разд. 10.7).

Например, в результате выполнения команды ``+`(3, 7, 4, 6, 8)` получится сумма элементов последовательности: 28; а если объявлен массив `A := array([x+1, x+2, x+3, x+4])`, то в результате выполнения команды `` `+`(A[k] $ k=1..4) `` получится сумма элементов массива: $4x + 10$.

Эти и другие операции с индексированными переменными в MuPAD можно организовать также с помощью операторов циклов и операторов условного перехода.

10.6. Операторы условного перехода

Операторы условного перехода позволяют предусмотреть несколько вариантов выполнения символьных вычислений в программе. В MuPAD существует два оператора условного перехода: `if` и `case`.

10.6.1. Оператор условного перехода IF

Несмотря на то, что оператор условного перехода `if` (*с англ.* если) считается оператором двойственного выбора, он имеет следующую структуру:

```
if <условие1> then <операторы1>
   elif <условие2> then <операторы2>
   elif <условие3> then <операторы3>
   ...
   else <операторы>
end_if
```

`<Условие1>`, `<условие2>`, `<условие3>` … – это выражения логического типа. Каждое из этих выражений может быть записано в виде одной или нескольких операций отношения, а также в виде одной или нескольких логических переменных. Условия, записанные в виде одной операции отношения или одной логической переменной, называют ***простыми***. Условия, записанные в виде нескольких операций отношения и (или) нескольких логических переменных, называют ***составными***. Операции отношения или (и) логические переменные, входящие в составное условие, должны быть связаны логическими союзами И (`and`), ИЛИ (`or`), НЕ (`not`) и т.д. Например, операции отношения, связанные логическим союзом И, записывают в следующем порядке: `x > −3 and x < 2`. Правила записи составных условий с использованием остальных логических союзов аналогичны приведенным выше правилами использования союза И. Результатом вычисления логического выражения (условия) в MuPAD является либо `TRUE` (истина), либо `FALSE` (ложь).

Если `<условие1>` верно (`TRUE`), то выполняются `<операторы1>`, расположенные после ближайшего к `<условию1>` служебного слова `then` вплоть до ближайшего служебного слова `elif` или `else` (если `elif` отсутствует). Если `<условие1>` неверно (`FALSE`), то выполняется проверка `<условия2>`. Если `<условие2>` верно (`TRUE`), то выполняются `<операторы2>`, расположенные после ближайшего к `<условию2>` служебного слова `then` вплоть до следующего служебного слова `elif` или `else` (если далее `elif` отсутствует). Если `<условие2>` неверно (`FALSE`), то выполняется проверка `<условия3>`; и так далее вплоть до последнего условия. Если последнее из записанных в операторе `if`

условий неверно (FALSE), то выполняются <операторы>, расположенные между служебными словами else (*с англ.* иначе) и end_if. Затем выполнение программы продолжается дальше.

Вместо служебного слова end_if можно использовать end.

Если содержанием задачи не предусмотрена проверка дополнительных условий (<условие2>, <условие3> …), то соответствующий блок elif … then … можно опустить. Аналогично, если содержанием задачи не предусмотрены <операторы>, которые необходимо выполнить, когда каждое из условий <условие1>, <условие2>, <условие3> … неверно, то блок else в операторе if можно опустить. Если ни одно из условий <условие1>, <условие2>, <условие3> … не верно, а блок else отсутствует, то оператор if пропускается и программа приступает к выполнению операторов или команд, расположенных после этого оператора if.

Таким образом, в MuPAD допускаются различные формы записи оператора if. Далее приведены наиболее часто используемые формы записи этого оператора:

if <условие1> then <операторы1> end_if	if <условие1> then <операторы1> else <операторы> end_if

if <условие1> then
  <операторы1>
  elif <условие2> then
  <операторы2>
end_if
	if <условие1> then
	  <операторы1>
	  elif <условие2> then
	  <операторы2>
	  else <операторы>
	end_if

Например, если значения вещественных переменных a и b вводятся с клавиатуры с помощью оператора input, то, используя оператор if, можно проверить, какое значение (положительное или отрицательное) получится в результате вычисления выражения $a + \sin b$. Для этого можно написать следующую последовательность команд:

```
a := input("Введите вещественное число:");
b := input("Введите другое вещественное число:");
if   a + sin(b)>0 then
     print(NoNL, "a+sin(b) ПОЛОЖИТЕЛЬНОЕ число")
```

```
elif a + sin(b)<0 then
     print(NoNL, "a+sin(b) ОТРИЦАТЕЛЬНОЕ число")
else print(NoNL, "a+sin(b) равно НУЛЮ")
end_if
```

В результате, если в качестве a и b ввести числа 3 и 5 соответственно, то в поле вывода появится текст "a+sin(b) ПОЛОЖИТЕЛЬНОЕ число"; если ввести числа –3 и 5, то в поле вывода появится текст "a+sin(b) ОТРИЦАТЕЛЬНОЕ число"; если ввести числа –1 и PI/2, то в поле вывода появится текст "a+sin(b) равно НУЛЮ".

В любом из исполняемых блоков (<операторы>; <операторы1>; <операторы2>; <операторы3>) в операторе if может быть заключено любое количество команд и (или) других операторов. В случае истинности условия выполняется вся группа команд (операторов), расположенных в соответствующем блоке. Среди операторов в любом из исполняемых блоков также может встречаться оператор if с другим условием. Такие операторы называются вложенными. Вложенный оператор должен целиком находиться внутри одного и того же исполняемого блока другого оператора, включая служебное слово end_if и соблюдение всех синтаксических правил.

Внутри каждого исполняемого блока в операторе if команды и операторы необходимо отделять друг от друга символом ; (точка с запятой). После последнего в блоке оператора, расположенного перед началом следующего блока или служебным словом end_if, точку с запятой ставить не обязательно.

Если в конце оператора if (после end_if) поставить двоеточие (:), то вывод результатов всех вычислений, которые записаны в любом из блоков этого оператора, подавляется. Если в конце оператора if (после end_if) двоеточие отсутствует, то в поле вывода отображается результат последнего вычисления, выполненного внутри оператора if.

Обратите внимание! Оператор print, расположенный в любом из блоков оператора if, выполняется независимо от наличия двоеточия после end_if. Например, после выполнения следующих команд:

```
if a<3 then  x := a + 3; y := a^2 - 1; z := x^2 - 3;
else         x := a - 3; y := a^2 + 1; z := x^2 + 3;
end_if
```

в поле вывода появится только значение переменной z. А если после end_if поставить двоеточие:

```
if a<3 then  x := a + 3; y := a^2 - 1; z := x^2 - 3;
else         x := a - 3; y := a^2 + 1; z := x^2 + 3;
end_if:
```

в поле вывода никаких значений не появится. Чтобы в поле вывода отобразить значения всех вычисленных переменных, можно, например, после оператора `if` (после `end_if`) добавить еще одну команду:

```
if a<3 then x := a + 3; y := a^2 - 1; z := x^2 - 3;
else        x := a - 3; y := a^2 + 1; z := x^2 + 3;
end_if:
x, y, z
```

или использовать оператор `print`.

10.6.2. Оператор выбора CASE

Оператор `case` (*с англ.* случай) позволяет для разных значений одной и той же переменной предусмотреть выполнение разных операторов или команд:

```
case <ключ>
  of <значение1> do    <операторы1>; break
  of <значение2> do    <операторы2>; break
  ...
  otherwise    <операторы>
end_case
```

`<Ключ>` – идентификатор переменной или выражение, по значениям которого осуществляется выбор, какие операторы необходимо выполнять в программе далее.

`<Значение1>`, `<значение2>` … – возможные значения `<ключа>`, каждому из которых ставится в соответствие определенная последовательность команд `<операторы1>`, `<операторы2>` … соответственно. В качестве значений ключа можно использовать как числовые константы, так и символьные переменные, выражения, последовательности, списки, множества, строки, матрицы и т.д.

В случае если `<ключ>` принимает значение, не указанное ни в одном из блоков `of`, выполняются `<операторы>`, расположенные после служебного слова `otherwise` (*с англ.* другие случаи). При необходимости блок `otherwise` можно опустить. Если `<ключ>` принимает значение, не указанное ни в одном из блоков `of`, а блок `otherwise` отсутствует,

оператор case пропускается и программа приступает к выполнению операторов или команд, расположенных после этого оператора case.

Вместо служебного слова end_case можно использовать end.

Если <ключ> принимает <значениеК>, то выполняются не только все <операторыК>, расположенные после соответствующего блока of <значениеК> do, но и операторы (команды), расположенные во всех последующих блоках, включая блок otherwise. Если необходимо, чтобы операторы (команды), расположенные в разных блоках of, выполнялись независимо, то перед переходом к следующему блоку of или otherwise можно использовать команду break. Если команда break расположена внутри оператора case (или какого-либо другого оператора), то ее выполнение прерывает работу данного оператора case, а программа переходит к выполнению написанных после оператора case команд или операторов. Очевидно, что в последнем блоке оператора case писать команду break не имеет смысла. Если необходимо, чтобы для нескольких <значений> <ключа> выполнялись одни и те же операторы, то блоки of для этих <значений> внутри оператора case следует располагать подряд, а исполняемые <операторы> вместе с командой break указывать только в последнем из этих блоков.

Далее для сравнения приведены два варианта использования оператора case, позволяющие получить одинаковые результаты:

Вариант 1	Вариант 2

`case x`
`  of 1 do y := 1; break`
`  of 3 do y := 9; break`
`  of 2 do y := 1/2; break`
`  of 4 do y := 1/4; break`
`  of 5 do y := 25; break`
`  of 6 do y := 1/6`
`end_case`
	`case x`
	`  of 1 do`
	`  of 3 do`
	`  of 5 do y := x^2; break`
	`  of 2 do`
	`  of 4 do`
	`  of 6 do y := 1/x`
	`end_case`

В программе из приведенного выше варианта 1 каждый из блоков of оператора case выполняется независимо, а в программе из варианта 2 команда y := x^2 выполняется, если переменная x принимает одно из трех значений (1, 3 или 5), а команда y := 1/x – если x принимает значение 2, 4 или 6. При других значениях переменной x оператор case в обоих вариантах программы никаких команд не выполняет.

Внутри каждого блока `of` оператора `case` команды (операторы) необходимо отделять друг от друга символом `;` (точка с запятой). После последнего в блоке оператора, расположенного перед началом следующего блока или служебным словом `end_case`, точку с запятой ставить не обязательно.

Если в конце оператора `case` (после `end_case`) поставить двоеточие (`:`), то вывод результатов всех вычислений, которые записаны в любом из блоков этого оператора, подавляется. Если в конце оператора `case` (после `end_case`) двоеточие отсутствует, то в поле вывода отображается результат последнего вычисления, выполненного внутри оператора `case`.

Обратите внимание! Оператор `print`, расположенный в любом из блоков оператора `case`, выполняется независимо от наличия двоеточия после `end_case`.

10.7. Операторы цикла

В отличие от MatLab в MuPAD существует три оператора цикла, которые позволяют различным образом организовывать повторение каких-либо команд или операторов.

10.7.1. Оператор FOR-цикла

Если некоторую последовательность команд или операторов необходимо повторять так, что количество повторных выполнений этих операторов известно (или может быть вычислено) до начала выполнения этих операторов и не зависит от результатов, получаемых повторяемыми операторами, используют оператор цикла `for` (*с англ.* для) одним из приведенных ниже способов:

```
for <параметр цикла> from <начало> to <конец>
                                      step <шаг> do
   <операторы>
end_for
for <параметр цикла> from <начало> downto <конец>
                                      step <шаг> do
   <операторы>
end_for
for <параметр цикла> in <список> do <операторы>
end_for
```

for <параметр цикла> **in** <множество> **do** <операторы> **end_for**

<Параметр цикла> (счетчик) – переменная, значения которой оператор цикла выбирает автоматически либо из указанного диапазона, либо из списка или множества.

Если значения <параметра цикла> выбираются из диапазона, необходимо задать границы диапазона (<начало> и <конец>), а также <шаг> изменения <параметра цикла>. <Шаг> изменения <параметра цикла> может быть как целым, так и дробным, но всегда положительным. Если параметр <шаг> не указан, то он по умолчанию считается равным единице. В этом случае служебное слово step (*с англ.* шаг) писать не нужно. Если необходимо, чтобы в процессе выполнения цикла <параметр цикла> увеличивался от начального значения до конечного, то между параметрами <начало> и <конец> используют служебное слово to. При этом <начало> не должно быть больше <конца>. Если необходимо, чтобы в процессе выполнения цикла <параметр цикла> уменьшался от начального значения до конечного, то между параметрами <начало> и <конец> используют служебное слово downto. При этом <начало> не должно быть меньше <конца>. Если для изменения <параметра цикла> задан диапазон, то цикл повторяется столько раз, сколько значений можно вычислить в данном диапазоне с указанным <шагом>, считая первое значение <параметра цикла> равным <началу>.

Если в процессе выполнения цикла значения <параметра цикла> выбираются из множества (или списка), то это <множество> (<список>) или соответствующий ему идентификатор необходимо указать после служебного слова in. В этом случае цикл повторяется столько раз, сколько элементов содержит множество (или список).

Для каждого значения <параметра цикла> (счетчика) выполняются <операторы>, расположенные между служебными словами do и end_for.

Вместо служебного слова end_for можно использовать end.

Если хотя бы один из параметров <начало>, <конец>, <шаг>, <множество> или <список> указаны с помощью переменных, то их значения должны быть заданы до начала цикла (до служебного слова for).

Ни <параметр цикла>, ни <шаг> его изменения, ни границы диапазона, а также <множество> (или <список>) значений <параметра цикла> внутри цикла изменять нельзя.

Если в цикле необходимо вычислять разные значения одного и того же выражения, то оператор цикла `for` аналогичен оператору `$`. В отличие от оператора `$`, в цикле `for` можно повторять любое количество команд и операторов. Поэтому цикл `for`, в отличие от оператора `$`, в процессе его выполнения не создает из вычисленных значений последовательность. Чтобы проиллюстрировать различия между оператором `$` и циклом `for`, приведем следующие фрагменты программ, позволяющие получить одинаковые списки, состоящие из значений функции `f(x)`:

Пример 1	Пример 2

`f := x -> x+sin(x*PI)^2;`
`S := []:`
`for x from 0 to 1`
`  step 1/6 do`
`  S := S.[f(x)]`
`end_for`
	`f := x -> x+sin(x*PI)^2;`
	`S := [f(x) $ x=0..1`
	`  step 1/6]`

Внутри цикла `for` операторы необходимо отделять друг от друга символом `;` (точка с запятой). После последнего оператора, расположенного перед `end_for`, точку с запятой ставить не обязательно.

Если в конце оператора `for` (после `end_for`) поставить двоеточие (`:`), то вывод результатов всех вычислений, которые записаны в цикле, подавляется. Если в конце оператора `for` (после `end_for`) двоеточие отсутствует, то в поле вывода отображается результат последнего вычисления, выполненного внутри цикла.

Обратите внимание! Оператор `print`, расположенный внутри цикла `for`, выполняется независимо от наличия двоеточия после `end_for`.

Использование оператора `break` внутри цикла `for` приводит к досрочному окончанию цикла; выполнение программы продолжается дальше. Использование оператора `next` внутри цикла `for` позволяет досрочно приступить к выполнению этого цикла со следующим значением <параметра цикла>.

Цикл `for` позволяет организовать поэлементные операции с индексированными переменными. Для этого в качестве параметра цикла выбирают индекс соответствующей переменной (массива, списка, множества и т.д.). Например, следующий набор оператором позволяет вычислить сумму элементов массива:

```
Z := array([-19*PI/4, 7*PI/3, 6*PI/5]);
s := 0:
for k from 1 to nops(Z) do     s := s + Z[k]
end_for
```

Следующий набор операторов позволяет заполнить матрицу целыми случайными числами из интервала [–3, 5]:

```
B := matrix(2, 4):
SEED := round(1e10*frandom(CurrentTime)()):
gen := random(-3..5):
for k from 1 to 2 do
    for p from 1 to 4 do
        B[k, p] := gen()
    end_for
end_for:
B
```

А следующий набор операторов позволяет проверить, является ли каждый элемент списка корнем уравнения:

```
ur := (5*x-6*PI)*(3*x-7*PI)*(4*x+19*PI) = 0;
res := [-19*PI/4, 7*PI/3, 6*PI/5];
for xk in res do
   proverka := subs(ur, x = xk);
   if is(proverka) then
      print(NoNL, "Корень уравнения: "); print(xk)
   end_if
end_for
```

10.7.2. Оператор цикла WHILE

Если некоторую последовательность команд или операторов необходимо повторять так, что количество повторных выполнений этих операторов неизвестно (и не может быть вычислено) до начала выполнения этих операторов и зависит от результатов, получаемых до цикла и внутри него, используют оператор цикла `while` (*с англ.* до тех пор, пока) следующим образом:

```
while <условие> do
      <операторы>
end_while
```

<Условие> – логическое выражение. Блок <операторов>, расположенный между служебными словами do (*с англ.* делать) и end_while, будет выполняться только в том случае, если <условие> принимает значение «истина» (TRUE). Когда <условие> перестает выполняться (FALSE), цикл заканчивается.

Чтобы организовать цикл while конечной длины, необходимо: 1) программный код, предшествующий циклу, организовать так, чтобы при первой проверке <условия> оно имело значение «истина»; 2) блок операторов, выполняемых в цикле, организовать так, чтобы при повторной проверке <условия> хотя бы одна из используемых в логическом выражении переменных имела другое значение (по сравнению с предыдущей проверкой). То есть значение хотя бы одной переменной из используемых для записи <условия> должно изменяться внутри цикла, а перед циклом должны быть заданы все переменные, используемые для записи <условия>. Тогда спустя какое-то количество повторений условие перестанет выполняться и цикл закончится.

Вместо служебного слова end_while можно использовать end.

Внутри цикла while операторы (команды) необходимо отделять друг от друга символом ; (точка с запятой). После последнего оператора (команды), расположенного перед end_while, точку с запятой ставить не обязательно.

Если в конце оператора while (после end_while) поставить двоеточие (:), то вывод результатов всех вычислений, которые записаны в цикле, подавляется. Если в конце оператора while (после end_while) двоеточие отсутствует, то в поле вывода отображается результат последнего вычисления, выполненного внутри цикла.

Обратите внимание! Оператор print, расположенный внутри цикла while, выполняется независимо от наличия двоеточия после end_while.

Использование оператора break внутри цикла while приводит к досрочному окончанию цикла; выполнение программы продолжается дальше. Использование оператора next внутри цикла while позволяет досрочно приступить к следующей проверке <условия>.

Например, продолжительность следующего цикла зависит от того, какие значения будет вводить пользователь:

```
x := 0:
while x < 100 do
    a := input(" a = ");
    print(NoNL, " a = "); print(NoNL, a);
    x := x + abs(a);
    print(NoNL, " X = "); print(x)
end_while
```

10.7.3. Оператор цикла REPEAT

Если некоторую последовательность команд или операторов необходимо повторять так, что количество повторных выполнений этих операторов неизвестно (и не может быть вычислено) до начала выполнения этих операторов и зависит только от результатов, получаемых внутри цикла, используют оператор цикла `repeat` (*с англ.* повторять) следующим образом:

repeat
    <операторы>
until <условие> **end_repeat**

<Условие> – логическое выражение. Блок <операторов>, расположенный между служебными словами `repeat` и `until` (*с англ.* до тех пор, пока не), будет сначала выполняться один раз, а повторное выполнение этих <операторов> осуществляется только в том случае, если <условие> принимает значение «ложь» (`FALSE`). Когда <условие> начнет выполняться (`TRUE`), цикл заканчивается.

Чтобы организовать цикл `repeat` конечной длины, необходимо блок <операторов>, выполняемых в цикле, и операторы, расположенные перед ним, организовать так, чтобы при первой проверке <условия> оно имело значение «ложь» (`FALSE`), а при повторной проверке <условия> хотя бы одна из используемых в логическом выражении переменных имела другое значение (по сравнению с предыдущей проверкой). То есть значение хотя бы одной переменной из используемых для записи <условия> должно изменяться внутри цикла. Тогда спустя какое-то количество повторений <условие> примет значение `TRUE` и цикл закончится. Так как проверка <условия> происходит после выполнения <операторов>, расположенных внутри цикла, переменные, использованные для записи <условия>, перед циклом могут быть не заданы (если они задаются в цикле).

Вместо служебного слова `end_repeat` можно использовать `end`.

Внутри цикла `repeat` операторы (команды) необходимо отделять друг от друга символом `;` (точка с запятой). После последнего оператора (команды), расположенного перед `until`, точку с запятой ставить не обязательно.

Если в конце оператора `repeat` (после `end_repeat`) поставить двоеточие (`:`), то вывод результатов всех вычислений, которые записаны в цикле, подавляется. Если в конце оператора `repeat` (после `end_repeat`) двоеточие отсутствует, то в поле вывода отображается результат последнего вычисления, выполненного внутри цикла.

Обратите внимание! Оператор `print`, расположенный внутри цикла `repeat`, выполняется независимо от наличия двоеточия после `end_repeat`.

Использование оператора `break` внутри цикла `repeat` приводит к досрочному окончанию цикла; выполнение программы продолжается дальше. Использование оператора `next` внутри цикла `repeat` позволяет досрочно приступить к повторному выполнению `<операторов>` без проверки `<условия>`.

Например, продолжительность следующего цикла также зависит от того, какие значения будет вводить пользователь:

```
x := 0:
repeat
    a := input(" a = ");
    print(NoNL, " a = "); print(NoNL, a);
    x := x + abs(a);
    print(NoNL, " X = "); print(x)
until x >= 100 end_repeat
```

10.8. Нестандартные функции

В программировании **функция** – часть программы, имеющая собственное имя, предназначенная для решения определенной задачи. К нестандартным функциям относятся функции, создаваемые программистом в процессе выполнения конкретных символьных вычислений и численных расчетов и не входящие в комплект поставки MuPAD. Нестандартные функции в MuPAD можно создавать двумя способами. Если последовательность операторов (команд), которые выделяют в отдельную функцию, можно свести к выполнению одной команды (например, к вычислению одного выражения), такая функция считается простой. Для объявления

простой функции используется оператор ->, применение которого подробно рассмотрено в разд. 2.7.2.

В этом разделе рассматривается другой способ объявления нестандартных функций в MuPAD – с помощью оператора proc. Для объявления функции оператор proc используют следующим образом:

```
<Имя> := proc(<аргумент1> = <значение1> : <тип1>,
              <аргумент2> = <значение2> : <тип2>, …) :
                                        <тип результата>
  option <опции>;
  local  <локальные переменные>;
  save   <глобальные переменные>;
begin
   <операторы>;
   <выражение>
end_proc
```

Параметр <имя> функции – уникальный идентификатор, который в дальнейшем будет использоваться в программе для обращения к функции.

После зарезервированного слова proc в круглых скобках через запятую необходимо указать список аргументов объявляемой функции: <аргумент1>, <аргумент2> … – идентификаторы независимых символьных переменных, с помощью которых программа сообщает функции исходные данные для выполняемых функцией вычислений. Например: … := proc(x) begin … end_proc – функция одного аргумента; … := proc(x, y) begin … end_proc – функция двух аргументов и т.д. Несмотря на то, что объявление функции происходит в том же файле, в котором расположена вызывающая ее программа, заданные ранее значения переменных, идентификаторы которых совпадают с аргументами объявляемой функции, не подставляются в выражения и операторы, записанные внутри, пока не будет осуществлен вызов этой функции.

Каждому аргументу объявляемой функции можно поставить в соответствие значение, которое будет по умолчанию присваиваться этому аргументу, если при вызове функции он не будет указан явно. Значение по умолчанию указывают после знака = вслед за соответствующим идентификатором аргумента следующим образом: <аргумент1> = <значение1> и т.д. Если для какого-либо аргумента указано значение по умолчанию, то при вызове функции этот аргумент можно опустить. Если значение по умолчанию не указано, то такой аргумент обязательно указывают при вызове функции. При объявлении функции сначала следует

записывать идентификаторы обязательных аргументов (у которых отсутствуют значения по умолчанию), а затем – идентификаторы необязательных аргументов. Например: … := proc(x, y, k1=0, k2=-1) begin … end_proc – функция четырех аргументов, для двух из которых заданы значения по умолчанию.

Если для аргумента объявляемой функции (одного или нескольких) указать тип, то при попытке подставить на место данного аргумента значение или переменную другого типа (отличного от указанного) будет появляться сообщение об ошибке. Тип аргумента указывают после знака : вслед за значением по умолчанию или вслед за идентификатором аргумента, если значение по умолчанию отсутствует (в виде <аргумент1> : <тип1> или <аргумент1> = <значение1> : <тип1>).

Если необходимо, чтобы объявляемая функция возвращала в программу только результат заданного типа, этот тип следует указать после символа : вслед за списком аргументов, заключенным в круглые скобки: … := proc(…) : <тип результата>. Если в результате выполнения относящихся к объявляемой функции операторов (команд) получится значение или выражение, тип которого отличается от указанного, появится сообщение об ошибке.

В качестве типов аргументов и возвращаемого результата в MuPAD можно использовать следующие идентификаторы: Type::Real, DOM_FLOAT – вещественное число; Type::Imaginary – мнимое число; Type::Complex – комплексное число; Type::Integer, DOM_INT – целое число; Type::Rational, DOM_RAT – рациональное число; Type::Positive – положительное число; Type::Negative – отрицательное число; Type::NonNegative – неотрицательное число; Type::Zero – ноль; Type::NonZero – ненулевое значение; Type::Even – четное целое число; Type::Odd – нечетное целое число; Type::PosInt – положительное целое число; Type::NegInt – отрицательное целое число; DOM_EXPR – выражение; DOM_BOOL – логическое значение; DOM_IDENT – идентификатор; DOM_INTERVAL – интервал; DOM_LIST – список; DOM_POLY – полином; DOM_SET – множество; DOM_TABLE – таблица; DOM_ARRAY – массив; Dom::Matrix() – матрица; DOM_PROC, DOM_FUNC_ENV – функция и т.д. Например: … := proc(n:Type::Even) begin … end_proc – функция, в качестве аргумента которой можно использовать только четные целые числа; … :=

proc(x, n):DOM_LIST begin … end_proc – функция, которая возвращает список.

При необходимости текущий тип любой переменной или выражения можно определить с помощью функции type (см. разд. 2.3), а проверить, совпадает ли тип <переменной> с указанным <типом>, можно с помощью функции testtype(<переменная>, <тип>), которая возвращает TRUE или FALSE соответственно.

Значения аргументов по умолчанию, а также тип аргументов и возвращаемого функцией результата не являются обязательными при объявлении функции (при необходимости их можно опустить).

Вместо служебного слова end_proc можно использовать end.

Между служебными словами begin и end_proc располагаются <операторы>, команды, <выражения>, которые необходимо выполнить при вызове функции для получения возвращаемого результата. Если в функции между служебными словами begin и end_proc отсутствует оператор return, то функция возвращает значение последнего вычисленного <выражения>. <Выражение>, значение которого должно быть возвращено функцией, можно не присваивать какой-либо переменной. Если же значение, которое должна возвращать функция, присвоено какой-либо переменной, то чтобы передать его программе как возвращаемый результат, необходимо использовать оператор return в любом месте функции (между begin и end_proc) следующим образом:

return(<переменные>)

С помощью оператора return (*с англ.* вернуть) можно вернуть значения одной или нескольких <переменных>. Для этого идентификаторы возвращаемых <переменных> необходимо перечислить через запятую. После выполнения оператора return дальнейшее выполнение функции прерывается (расположенные после оператора return команды, <операторы> или <выражения> не выполняются).

Например: функция … := proc(x) begin x^2 end_proc возвращает значение возведенного в квадрат аргумента; функция … := proc(x, y) begin sqrt(x^2 + y^2) end_proc возвращает расстояние от начала координат до точки с координатами (*x*, *y*); функция … := proc(x, y) begin if x>y then return(x) end_if; y end_proc возвращает большее из чисел *x* или *y*; функция … := proc(t) local x, y, z; begin x := cos(5*t); y := sin(3*t + 1/10); z := t/5;

`[x, y, z] end_proc` возвращает координаты тела в момент времени *t* в виде списка.

Указанные при объявлении функции аргументы перечисляются через запятую, поэтому образуют последовательность. Внутри функции к ее аргументам можно обратиться не только с помощью их идентификаторов, но и с помощью функции `args`, которую вызывают следующим образом:

```
<Последовательность аргументов> := args()
<Значение аргументаk> := args(k)
<Количество аргументов> := args(0)
```

Если функцию `args` вызывают без параметров: `args()`, то она возвращает последовательность всех указанных при вызове функции аргументов (количество аргументов функции, указанных при ее объявлении и вызове, может не совпадать). Если вместо аргумента функции `args` указать ноль: `args(0)`, то она возвращает количество аргументов текущей функции, указанных при ее вызове. Если в качестве аргумента функции `args` указать целое положительное число или переменную, которой присвоено целое положительное число, то функция `args` возвращает значение аргумента текущей функции, расположенного на указанном месте в последовательности аргументов при вызове функции. Использование функции `args` для обращения к аргументам объявляемой функции вместо их идентификаторов полезно, когда при вызове функции можно указывать не все аргументы и когда необходимо выполнить одинаковые преобразования для каждого аргумента. Тогда с помощью оператора `$` или оператора `for` можно организовать цикл по всем аргументам объявляемой функции. Например, следующая функция создает список из ее аргументов:

```
…:= proc(x1, x2 = 0, x3 = 0, x4 = 0, x5 = 0)
begin [args(k) $ k= 1..args(0)]
end_proc
```

Если при вызове такой функции указать только два аргумента, список будет состоять из двух элементов. Следующая функция создает полиномиальное выражение вида $((((x + a_1) + a_2)^2 + a_3)^3 + a_4)^4$:

```
…:= proc(x, a1, a2 = 0, a3 = 0, a4 = 0)
begin for k from 2 to args(0) do
      x := (x + args(k))^(k - 1) end_for
end_proc
```

Если внутри объявляемой функции (между begin и end_proc) использовать идентификатор <имя>, которому присвоена объявляемая функция, то выполняется ее рекурсивный вызов. Чтобы внутри функции использовать ее <имя> без вызова, используют функцию procname следующим образом:

<Результат> := **procname**(<аргументы>)

<Имя> функции со списком указанных при вызове аргументов чаще всего используется для идентификации ситуации, когда вычисление данной функции невозможно. В этом случае инициируют возвращение имени функции с указанными при вызове аргументами следующим образом:

return(procname(arg()))

Например, следующая функция для четных целых значений аргумента возвращает половину аргумента, а в остальных случаях – имя функции с указанным аргументом (возвращает введенную команду невыполненной):

…:= proc(n) begin if testtype(n, Type::Even) then n/2 else return(procname(n)) end_if end_proc.

Для получения возвращаемого результата внутри функции могут потребоваться дополнительные переменные. Если до объявления функции в программе были заданы значения некоторых переменных, то присвоение этих переменных внутри функции приведет к изменению значений этих переменных не только внутри функции, но и в программе. Если необходимо, чтобы присвоение вспомогательных переменных внутри функции не влияло на значения одноименных переменных в программе, такие переменные внутри функции следует объявить локальными. Для этого перед служебным словом begin необходимо расположить раздел local, в котором после служебного слова local (*с англ.* локальный) перечислить через запятую идентификаторы всех переменных, которые должны быть локальными для объявляемой функции. В этом случае на время выполнения функции создаются новые переменные (даже если их идентификаторы совпадают с идентификаторами, использованными в программе ранее). После выполнения функции <локальные переменные> автоматически уничтожаются. Например, следующая функция вычисляет среднее арифметическое значение ее аргументов:

```
…:= proc(x1, x2 = 0, x3 = 0, x4 = 0, x5 = 0)
local s, k;
begin s := 0;
      for k from 1 to args(0) do
```

```
      s := s + args(k) end_for; s/args(0)
end_proc
```

Однако некоторые вычисления требуют использования (и изменения в процессе выполнения функции) <глобальных переменных> MuPAD, например переменных окружения. Если перечислить такие переменные в разделе save (*с англ.* сохранить), который также пишется перед служебным словом begin, то перед началом выполнения объявляемой функции создается резервная копия текущих значений этих переменных, а после выполнения всех <операторов>, команд и <выражений>, записанных между служебными словами begin и end_proc, текущие значения указанных переменных заменяются сохраненными ранее значениями. Например, внутри следующей функции аргумент используется как вещественное число в десятичном формате с тремя значащими цифрами, а возвращается в программу в десятичном формате с десятью значащими цифрами (если значение DIGITS=10 по умолчанию не было изменено ранее в программе): … := proc(x) save DIGITS; begin DIGITS := 3; y := float(x); print(y); y end_proc. Таким образом, оба раздела (local и save) обеспечивают независимость между вычислениями внутри функции и в программе, но используют для этого разные методы.

Кроме разделов local и save функция может содержать еще и раздел option, в котором указывают <опции> с помощью одного (или нескольких) следующих идентификаторов: escape – позволяет организовать обращение не только к самой объявляемой функции, но и к ее внутренним конструкциям (например, если внутри функции объявлена другая функция); hold – позволяет объявить отложенными все записанные между begin и end_proc <операторы>, команды и <выражения>; noDebug – запрещает выполнение отладки (debug) текста объявляемой функции; noExpose – позволяет скрыть все записанные между begin и end_proc <операторы>, команды и <выражения>; noFlatten – запрещает преобразование последовательности, передаваемой в функцию в качестве аргументов при ее вызове; remember – позволяет сохранить все значения, полученные функцией при предыдущих вызовах, и использовать эти значения без повторного выполнения вычислений при вызове функции с одинаковыми параметрами. Если в разделе options необходимо указать несколько опций, их перечисляют через запятую.

С помощью функции expose можно вывести текст любой объявленной ранее функции (стандартной или нестандартной), если она написана без опции noExpose. Для этого функцию expose вызывают следующим образом:

expose (<имя функции>)

Например, если ранее объявлена функция f := x -> x^2, то далее с помощью команды expose(f) в поле вывода появится текст: x -> x^2, если объявлена функция f := proc(x) begin x^2 end_proc, то далее с помощью команды expose(f) в поле вывода появится текст: proc(x) name f; begin x^2 end_proc, а с помощью команды expose(sin) – длинный текст, начало и конец которого приведены далее: proc(x) name sin; local f, y; option noDebug; begin if args(0) = 0 then ... return(procname(x)) end_if end_proc.

Обращение (вызов) к функции, объявленной с помощью оператора proc, выполняется так же, как и обращение (вызов) к функции, объявленной с помощью оператора ->:

<Результат> := <имя функции>(<значение аргумента1>,
<значение аргумента2>, …)

Количество указанных при вызове функции значений аргументов должно соответствовать количеству аргументов, указанных при объявлении этой функции: аргументы, для которых заданы значения по умолчанию, можно не указывать, остальные аргументы необходимо обязательно указать явно.

Вызывать нестандартные функции, объявленные с помощью оператора proc, можно как с присвоением возвращаемого функцией результата некоторой переменной, так и без присвоения (аналогично функциям, объявленным с помощью оператора ->).

Например, независимо от того, с помощью какого оператора объявлена функция (radius := (x, y) -> sqrt(x^2 + y^2) или radius := proc(x, y) begin sqrt(x^2 + y^2) end_proc) для ее вызова можно написать следующие команды: radius(3, 5), radius(x0, y0) или r1 := 3 + radius(x1 - x0, y1 - y0)+radius(1, 3). Функцию koord := proc(t) local x, y, z; begin x := cos(5*t); y := sin(3*t + 1/10); z := t/5; [x, y, z] end_proc можно вызвать, например, так:

[x1, y1, z1] := koord(t1); [x2, y2, z2] := koord(t2)

Если внутри функции возвращаемый результат записан с использованием переменных, ранее заданных в программе и не являющихся локальными для данной функции, то значения таких переменных подставляются в текст функции непосредственно перед ее вызовом. Например, если объявлена функция `g := proc(x) begin x + a end_proc`, то в результате выполнения команд `a := 5: g(x)` (перед вызовом функции задано значение переменной `a`) в командном поле появится выражение $x + 5$; если далее выполнить команды: `a := 7: g(x)`, то в командном поле появится выражение $x + 7$.

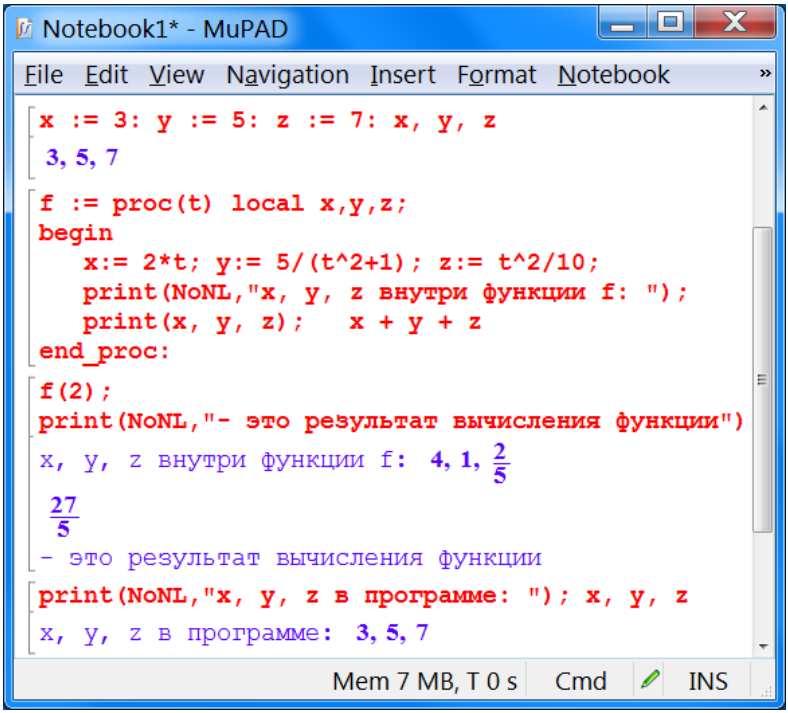


Рис. 10.6. Использование локальных переменных при объявлении функции

При написании функций следует помнить, что если переменная не объявлена локальной (`local`) и не сохранена в разделе `save`, то присвоение ей значения внутри функции приводит к изменению значения этой

переменной в программе. Например, если перед объявлением функции задано значение переменной `q := 10`, а затем объявлена функция `ff := proc(x) begin q := q + x; x^2 end_proc`, то в результате выполнения команды: `ff(3), q` в поле вывода появится: 9, 13. Если далее выполнить команду: `ff(4), q`, то в поле вывода появится: 16, 17. На рис. 10.6 приведен пример, демонстрирующий сохранность значений одноименных переменных, используемых в программе и в функции, если внутри функции одни объявлены локальными (`local`). Если в этом примере при объявлении функции вместо `local x, y, z;` написать `save x, y, z;`, то вместо создания локальных переменных будут создаваться резервные копии текущих значений глобальных переменных, а результаты выполнения программы при этом не изменятся.

10.9. Контроль продолжительности вычислений

Во многих случаях время, в течение которого программа выполняет определенные вычисления, считается одним из существенных критериев, в соответствии с которыми выбирают тот или иной алгоритм (метод) программирования. В MuPAD функция `time` позволяет измерить время, в течение которого выполнялись некоторые операторы (команды). Для этого функцию `time` вызывают следующим образом:

`<Переменная> := **time**(<оператор1>, <оператор2>, …)`

В качестве каждого из параметров `<оператор1>`, `<оператор2>`, … можно использовать любую команду, оператор или функцию MuPAD. Перед выполнением `<оператора1>` функция `time` запрашивает системное время CPU, после выполнения последнего из перечисленных в качестве параметров операторов функция `time` вновь запрашивает системное время CPU и возвращает разницу полученных значений времени в миллисекундах. При этом результаты выполнения `<оператора1>`, `<оператора2>`, … в поле вывода не отображаются. Например, в результате выполнения команды

```
time((a := factor(expand(int(exp(x)*sin(x)^3, x)))))
```

в поле вывода появится число (например, 1762.8113), а не значение интеграла $\frac{1}{5}(3\cos^3 x - \sin x\cos^2 x + 4\sin x - 6\cos x)e^x$.

Если функцию `time` вызывают без параметров, то она один раз запрашивает системное время CPU компьютера. Это позволяет измерять время

выполнения операторов, записанных в разных командных полях. Для этого функцию `time()` вызывают дважды (перед и после выполнения операторов), а затем вычисляют разницу значений времени. При этом результаты выполнения операторов, расположенных между двумя вызовами функции `time()`, могут отображаться в поле вывода. Например, после выполнения команд `t0 := time(): a := factor(expand(int(exp(x)*sin(x)^3, x))); t1 := time(): t1 - t0` в поле вывода появится не только число (например, 1809.6116), соответствующее времени вычисления интеграла, но и значение интеграла $\frac{1}{5}(3\cos^3 x - \sin x \cos^2 x + 4\sin x - 6\cos x)e^x$.

При измерении времени следует учитывать, что MuPAD хранит результаты всех вычислений, выполненных после последней очистки памяти с помощью функции `reset()`. При повторной попытке осуществить вычисления, выполненные ранее, вычисления не производятся, а выводятся полученные ранее результаты. Поэтому, если измерять время повторного получения вычисленных ранее результатов, то соответствующее время окажется значительно меньше по сравнению со временем, потраченным на получение этого результата впервые после очистки памяти. Для двух приведенных выше примеров на том же компьютере получится соответственно 15.6001 и 46.8003 мс.

Следует помнить, что вывод результатов в графическом виде также занимает определенное время. Поэтому если во втором из приведенных ранее примеров выполнять вычисления без вывода значения интеграла, то такие вычисления займут меньше времени. Это особенно заметно, если сравнивать время повторных вычислений. Так, для второго из приведенных ранее примеров при повторных вычислениях с выводом вычисленного интеграла и без него можно получить следующие значения измерения времени: 46.8003 и 15.6001 мс соответственно.

Результат измерения времени зависит от загруженности процессора. Поэтому при повторном выполнении той же программы результаты измерения времени могут отличаться.

Глава 11. СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ В М-ФАЙЛАХ

MatLab позволяет осуществлять символьные вычисления не только в приложении MuPAD, но и в командном окне (`Command Window`), в котором запускаются обычные программы, предназначенные для числовых расчетов. Для выполнения символьных вычислений в командном окне (`Command Window`) в MatLab существует пакет Symbolic Math Toolbox. Если в лицензию MatLab входят символьные вычисления, функции, входящие в этот пакет, доступны в командном окне (а следовательно, и в m-файлах, выполняемых в командном окне) без каких-либо специальных подключений. Приложение MuPAD и пакет Symbolic Math Toolbox специализируются на решении практически одних и тех же задач путем выполнения символьных вычислений, но при использовании каждого из них необходимо учитывать их синтаксические особенности. Многие функции MuPAD и Symbolic Math Toolbox, предназначенные для решения аналогичных задач, обозначены сходными (иногда одинаковыми) идентификаторами, но при этом могут иметь разные параметры и возвращать результаты вычислений по-разному. Так как пакет Symbolic Math Toolbox используется в m-файлах и командном окне MatLab, для него справедливы все синтаксические правила и операторы языка программирования MatLab. Приложение MuPAD использует другие синтаксические конструкции и операторы. Так, оператор присвоения в MatLab обозначается символом =, а в MuPAD – символом `:=`; для подавления автоматического вывода полученных результатов в MatLab используется символ `;`, а в MuPAD – символ `:` (табл. 11.1); строки в MatLab записываются в апострофах `' '`, а в MuPAD – в кавычках `" "`. Например, чтобы переменной `y` присвоить значение 57 необходимо в MuPAD написать `y := 57`, а в MatLab – написать `y = 57`. Неисполняемая часть программы (комментарии) в MuPAD записывается в полях специального типа (в текстовых полях), а в MatLab – в любом месте текста программы после символа `%` и до конца строки. По умолчанию результаты вычислений в MatLab выводятся в текстовом виде, а в MuPAD – в графическом (см. рис. 1.1, 1.2) и т.д. Синтаксис элементарных математических операций (табл. 11.2) и некоторых операций отношения (табл. 11.3) совпадает,

а логические союзы (табл. 11.4) и стандартные константы (табл. 11.5) записываются в MuPAD и в MatLab по-разному.

Таблица 11.1

Основные различия синтаксиса MuPAD и MatLab

Очистка памяти	
MuPAD	`reset()`
MatLab	`clear all`
Присвоение	
MuPAD	`<Переменная> := <выражение>`
MatLab	`<Переменная> = <выражение>`
Подавление автоматического вывода результата	
MuPAD	`:`
MatLab	`;`
Вывод в графическом виде	
MuPAD	автоматически по умолчанию или `print(<выражение>)`
MatLab	`pretty(<выражение>)`
Вывод в текстовом виде	
MuPAD	`print(Plain, <выражение>)`
MatLab	автоматически по умолчанию или `disp(<выражение>)`
Строковое значение	
MuPAD	`"<текст>"`
MatLab	`'<текст>'`

Таблица 11.2

Арифметические операции в MuPAD и MatLab

	Математика	**MuPAD**	**MatLab**
Сложение	$a + b$	`a + b`	`a + b`
Вычитание	$a - b$	`a - b`	`a - b`
Умножение	ab	`a*b`	`a*b`
Деление	$\frac{a}{b}$	`a/b`	`a/b`
Возведение в степень	a^b	`a^b`	`a^b`

Окончание табл. 11.2

	Математика	MuPAD	MatLab
Корень квадратный	$\sqrt{x}$	sqrt(x)	sqrt(x)
Модуль числа	$\|x\|$	abs(x)	abs(x)
Остаток от деления *a* на *b*		a mod b	mod(a, b) rem(a, b)
Деления *a* на *b* нацело		a div b	

Таблица 11.3

Операции отношения в MuPAD и MatLab

	Математика	MuPAD	MatLab
Равно	$a = b$	a = b	a == b
Неравно	$a \neq b$	a <> b	a ~= b
Приближенно равно	$a \approx b$	a ~= b	
Больше	$a > b$	a > b	a > b
Меньше	$a < b$	a < b	a < b
Больше или равно	$a \geq b$	a >= b	a >= b
Меньше или равно	$a \leq b$	a <= b	a <= b

Таблица 11.4

Логические операции в MuPAD и MatLab

	Математика	MuPAD	MatLab
И	$A \bigcap B$	A and B	A & B или and(A, B)
Или	$A \bigcup B$	A or B	A | B или or(A, B)
Не	$/A$	not A	~A или not(A)

Таблица 11.5

Стандартные константы и переменные окружения в MuPAD и MatLab

	Математика	MuPAD	MatLab
Встроенное число π (3,141592653589793…)	π	PI	pi
Мнимая единица ($\sqrt{-1}$)	i	I	i

Окончание табл. 11.5

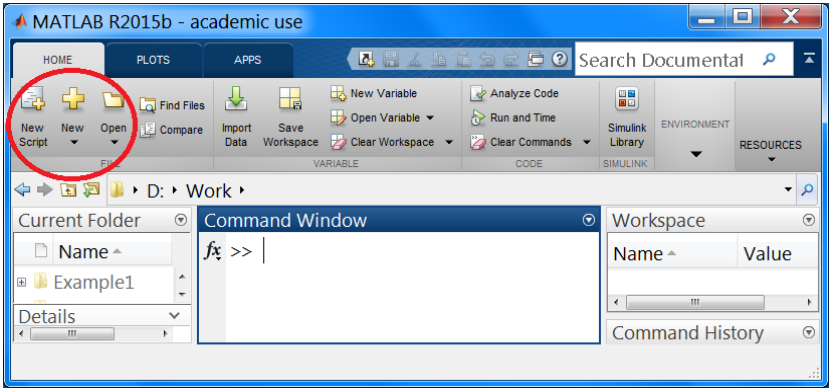
	Математика	MuPAD	MatLab
Число *e* (2,718281828459046…)	*e*	E	exp(1)
Бесконечность	∞	infinity	Inf
Истина		TRUE	1
Ложь		FALSE	0
Количество значащих цифр (для вещественных чисел, записанных в десятичном формате) – переменная окружения		DIGITS = 10	digits(32)

Между MuPAD и MatLab существуют различия не только в синтаксисе языка, но и в структуре хранения данных. В MuPAD используются такие индексированные переменные, как последовательности, списки, множества, массивы, матрицы. Практически всем им в MatLab соответствуют матрицы (двумерные и одномерные массивы, нумерация элементов которых начинается с единицы).

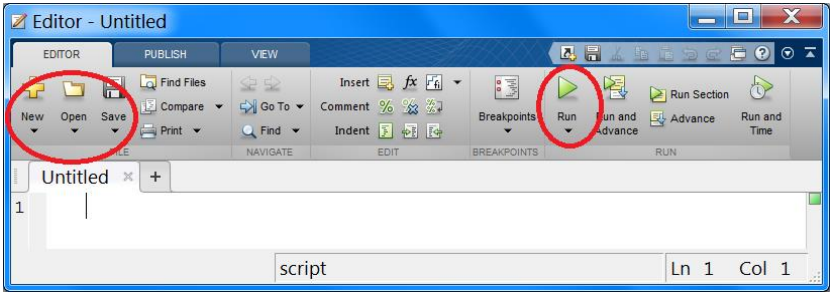
Для создания массива (матрицы, вектора) в MatLab как при числовых, так и при символьных вычислениях достаточно перечислить его элементы в квадратных скобках через запятую и (или) точку с запятой. Элементы, относящиеся к одной строке матрицы (вектора), перечисляются через запятую, а строки отделяются друг от друга точкой с запятой. Например, в результате выполнения команды `V = [1, 3, 9, 6]` получится вектор-строка (одномерный массив), а в результате выполнения команды `A = [2, 4, 6; 9, 7, 5]` – матрица размером 2 × 3 (двумерный массив). При обращении к элементам массива в MatLab индексы указывают в круглых скобках (а не в квадратных, как в MuPAD). Например, `V(2)`, `V(4)` или `A(3, 1)` и т.д.

Так как в MatLab все переменные интерпретируются как матрицы (даже те, в которых хранится только одно значение), все арифметические операции в MatLab выполняются как соответствующие операции с матрицами (по правилам линейной алгебры). Для поэлементного выполнения арифметических операций с массивами в MatLab введены дополнительные операции: `.+` (поэлементное сложение), `.-` (поэлементное вычитание), `.*` (поэлементное умножение), `./` (поэлементное деление), `.^` (поэлементное возведение в степень). Например, если задан массив `V = [1, 3, 9, 6]`, то при попытке выполнить команду `V^2` появится

сообщение об ошибке, а в результате выполнения команды `V.^2` получится новый массив с элементами `[1, 9, 81, 36]`.



а



б

Рис. 11.1. Основное окно среды MatLab (*а*) и окно редактора m-файлов (*б*)

Для создания и редактирования программ в комплект поставки MatLab входит редактор m-файлов, для запуска которого достаточно создать новый (script или function) или открыть существующий файл, написанный на языке программирования MatLab и сохраненный с расширением m. Для создания нового m-файла можно воспользоваться кнопкой `New Script` на панели инструментов основного окна MatLab (или `New` → `Function` в окне редактора m-файлов). Чтобы открыть существующий m-файл, можно воспользоваться кнопкой `Open` на той же панели инструментов (рис. 11.1, *а*) основного окна

MatLab. Редактор m-файлов, окно которого изображено на рис. 11.1, *б*, позволяет не только вносить изменения в текст программы, но и управлять ее сохранением (`Save`) и выполнением (`Run`). Выполнение программы, сохраненной в файле с расширением m, происходит в командном окне (`Command Window`), которое расположено в основном окне MatLab. В командном окне также можно писать и выполнять отдельные команды, но в этом случае последовательность написанных в этом окне команд в файле не сохраняется. Язык программирования MatLab (при наличии соответствующей лицензии) позволяет организовывать как числовые (несимвольные), так и символьные вычисления. Поэтому как программы для выполнения числовых расчетов, так и программы для выполнения символьных вычислений в MatLab рекомендуется писать и редактировать в редакторе m-файлов.

11.1. Символьные переменные и выражения

В MuPAD все не присвоенные ранее переменные считаются символьными (по умолчанию). В MatLab по умолчанию все переменные считаются несимвольными. В процессе выполнения программного кода в командном окне MatLab контролирует, чтобы всем несимвольным переменным на момент использования их в качестве исходных параметров функций, выражений соответствовало хотя бы одно значение (числовое или строковое). Тип таких переменных определяется типом хранящейся в них информации. В отличие от несимвольных, для выполнения операций с символьными переменными им не обязательно присваивать какое-либо значение (числовое, строковое, символьное). Чтобы MatLab не требовал обязательного присвоения конкретных значений символьным переменным, перед использованием их необходимо объявить как переменные специального типа – символьные переменные. После этого все операции с переменными, объявленными символьными, будут выполняться в программе так же, как с числовыми.

Для объявления символьных переменных в MatLab используют оператор `syms` в любом месте программы следующим образом:

`syms <переменная1> <переменная2> … <свойство>`

После служебного слова `syms` необходимо перечислить отделенные друг от друга пробелами идентификаторы `<переменных>`, которые далее в программе следует считать символьными. Если объявляемые переменные должны обладать одинаковыми свойствами, то эти свойства

можно указать после списка объявляемых переменных. Если <свойство> явным образом не указано, то объявленные переменные по умолчанию считаются символьными со свойствами комплексных чисел. В качестве <свойств> можно использовать следующие стандартные идентификаторы: real, positive, integer, rational. Свойства символьных переменных учитываются при выполнении аналитических преобразований выражений, уравнений, неравенств, записанных с их использованием.

Для объявления символьных переменных, обладающих разными свойствами, следует использовать несколько операторов syms последовательно. Например, с помощью команд syms x y z; syms t positive можно в командном окне (или m-файле) объявить четыре символьных переменных, одна из которых (t) будет считаться положительно определенной, а остальные – комплексными.

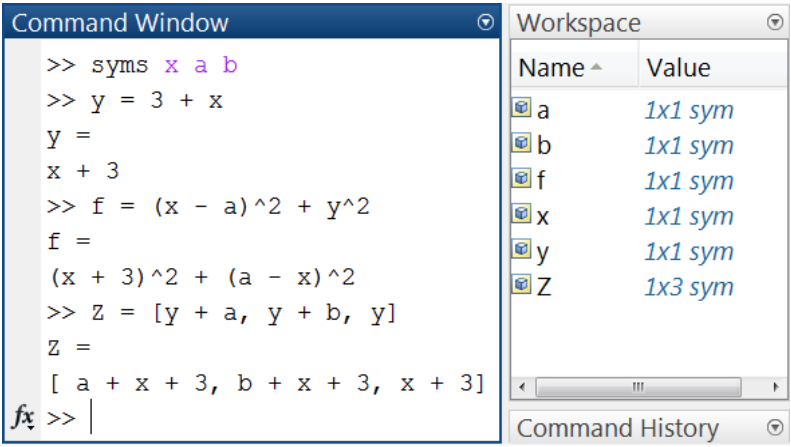


Рис. 11.2. Объявление символьных переменных в MatLab

Если переменной присваивается выражение (уравнение, неравенство), записанное с помощью одной или нескольких символьных переменных, такая переменная автоматически становится символьной. Таким образом, нет необходимости все символьные переменные объявлять с помощью оператора syms. Достаточно объявить только те переменные, которым ранее никакие значения не присваивались. Например, после выполнения команд syms a; f = a^3 – a – 4 в программе будут созданы две символьные

переменные a и f, хотя с помощью оператора syms объявлена только одна из них. Если при создании массива (матрицы) в записи хотя бы одного его элемента используется объявленная ранее символьная переменная, все элементы массива автоматически преобразуются в символьные. Например, в результате выполнения команд syms x; W = [1/2, 2 - x, x + 1, 4] будет объявлена символьная переменная x и создан одномерный массив-строка W, состоящий из символьных выражений: 1/2, 2 - x, x + 1, 4. На рис. 11.2 приведены другие примеры объявления символьных переменных в командном окне MatLab (в строках, отмеченных символом >>, записаны команды; объявленные переменные отображаются в окне Workspace, а их значения – в командном окне с помощью вывода по умолчанию в строках, не отмеченных символом >>).

Для принудительного (неавтоматического) преобразования числовых или строковых значений в символьные выражения можно использовать функцию sym, которую вызывают следующим образом:

<Переменная> = sym(<выражение>, <свойство>)

В качестве <выражения> может использоваться математическое выражение, уравнение или неравенство, записанное в строковом или числовом виде, явным образом или через идентификатор переменной, которой оно ранее было присвоено. Если при записи <выражения> в числовом виде использована ранее заданная несимвольная переменная, ее значение автоматически ПОДСТАВЛЯЕТСЯ в преобразуемое <выражение>. Если при записи <выражения> в строковом виде использована ранее заданная (несимвольная или символьная) переменная, то ее значение в преобразуемое <выражение> НЕ ПОДСТАВЛЯЕТСЯ. В качестве параметра <выражение> можно также использовать массив (матрицу), элементами которой являются числовые или строковые выражения, тогда в результате выполнения функции sym будет создан массив того же размера, состоящий из соответствующих символьных выражений. С помощью параметра <свойство> можно указать, каким свойством будет обладать создаваемый функцией sym символьный объект. В качестве параметра <свойство> можно использовать следующие стандартные идентификаторы: real, positive, integer, rational, clear.

Если созданный функцией sym символьный объект (выражение) присваивается какой-то <переменной>, то эта <переменная> автоматически становится символьной. Если символьный объект создан в результате преобразования строкового представления математической

формулы, то символьной объявляется только <переменная>, которой присвоен символьный объект, а входящие в формулу незаданные ранее переменные остаются необъявленными. Все входящие в формулу числовые значения преобразуются в символьные. После создания символьное выражение автоматически упрощается. Например, в результате выполнения команды `q = sym(1/2 + 1/3)` получится символьная переменная `q`, в которой хранится число `5/6` в символьном виде; в результате выполнения команды `y = sym('x^2 - 3')` будет объявлена только символьная переменная `y`; в результате выполнения команды `V = sym([1/2, 2 -sym('x'), sym('x') + 1, 4])` получится одномерный массив-строка `V`, элементами которого являются символьные выражения: `1/2`, `2 - x`, `x + 1`, `4`, при этом переменная `x` объявлена не будет (тот же результат получится в результате выполнения команды `V = sym('[1/2, 2 - x, x + 1, 4]')`). Примеры преобразования числовых и строковых выражений в символьные переменные в командном окне MatLab приведены на рис. 11.3 (в строках, отмеченных символом >>, записаны команды; объявленные переменные отображаются в окне Workspace, а их значения – в командном окне с помощью вывода по умолчанию в строках, не отмеченных символом >>).

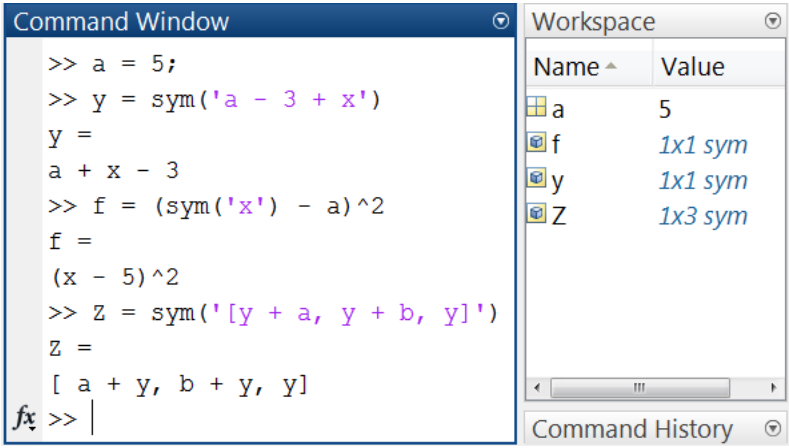


Рис. 11.3. Преобразование числовых и строковых выражений в символьные

Обратное преобразование символьного выражения в строку можно осуществить с помощью функции char, которую вызывают следующим образом:

<Переменная> = char(<символьное выражение>)

Например, в результате выполнения команд syms x; y = 3 - x^2 + x; char(y) получится строка 'x - x^2 + 3'.

Таким образом, в зависимости от способов дальнейшего использования символьные выражения, уравнения, неравенства или их системы, записанные с помощью числовых значений и буквенных переменных, в MatLab можно создать двумя способами: 1) объявить все буквенные переменные, через которые предстоит записать соответствующее выражение (уравнение, неравенство), символьными, а затем, используя эти символьные переменные, записать выражение (уравнение, неравенство), которое, в свою очередь, присвоить новой переменной; 2) математическое выражение (уравнение, неравенство) записать в строковом виде, а затем преобразовать полученную строку в символьное выражение (уравнение, неравенство), которое, в свою очередь, присвоить новой переменной. При использовании второго способа следует помнить, что символьной будет объявлена только одна переменная, которой обозначено символьное выражение, а входящие в него буквенные обозначения символьными переменными объявлены не будут. Поэтому если они нужны для выполнения дальнейших вычислений, их следует объявить символьными отдельно.

Возможность создавать кусочно-заданные выражения (аналогичные piecewise в MuPAD) в MatLab отсутствует.

С помощью функции symvar в MatLab можно определить, какие независимые символьные переменные входят в символьное выражение. Для этого функцию symvar вызывают следующим образом:

<Результат> = symvar(<исходное выражение>)

В качестве параметра <исходное выражение> символьное выражение можно записать либо явным образом, либо через идентификатор переменной, которой это выражение было ранее присвоено (обозначено). В качестве <результата> функция symvar возвращает одномерный массив символьных переменных, которые встречаются в указанном выражении. Например, если ранее в программе были объявлены символьные переменные syms x a b y t, то в результате выполнения команд z = a + 5; p = 4; symvar(z - p + x) получится массив [a, x].

Чтобы определить тип созданной ранее переменной, в MatLab используется команда class, а чтобы удалить созданную ранее переменную –

команда clear. Эти команды используются одинаково как для символьных, так и для несимвольных переменных:

class(<переменная>)

clear <переменная1> <переменная2> …

При использовании команды class можно указать идентификатор только одной <переменной>, тип которой необходимо определить. Если указанная <переменная> содержит одно символьное выражение, то в результате выполнения команды class получится 'sym'. Если указанная <переменная> содержит одно числовое значение, то в результате выполнения команды class получится 'double' и т.д.

При использовании команды clear можно перечислить через пробел идентификаторы нескольких <переменных>, которые необходимо удалить из памяти.

Обратите внимание, что в отличие от MatLab в MuPAD для определения типа и удаления созданных ранее переменных используются другие команды: type и delete соответственно (см. разд. 2.3).

Для преобразования в символьном выражении рациональных числовых значений в десятичный формат в MatLab используется функция vpa (в MuPAD ей соответствует функция float), которую вызывают следующим образом:

<Результат> = vpa(<выражение>)

<Результат> = vpa(<выражение>, <количество знаков>)

В качестве параметра <выражение> необходимо указать символьное выражение (уравнение, неравенство), содержащее числовые коэффициенты. Параметр <выражение> можно записать либо явным образом, либо с помощью идентификатора переменной, которой соответствующее выражение (уравнение, неравенство) было ранее присвоено. В качестве <результата> функция vpa возвращает то же <выражение>, в котором все числовые коэффициенты представлены в десятичном формате. Параметр <количество знаков> позволяет указать, сколько значащих цифр необходимо использовать при переходе к десятичному формату. Если параметр <количество знаков> не указан, по умолчанию количество значащих цифр числа, записанного в десятичном формате, определяется переменной окружения digits. Например, если ранее в программе было создано символьное выражение y = sym('x + 4*sqrt(3)'), то в результате выполнения команды vpa(y, 3)

Таблица 11.6

Стандартные функции преобразования типов в MuPAD и MatLab

	MuPAD	**MatLab**
Округление вещественного числа до ближайшего целого	`round(x)`	`round(x)`
Округление вещественного числа до n десятичных знаков после запятой	`round(x, n)`	`vpa(x, n)`
Дробная часть вещественного числа	`frac(x)`	`frac(x)`
Преобразование числа в десятичный формат	`float(x)` `coerce(x, Dom::Float)`	`vpa(x)` `double(x)`
Преобразование числа в рациональный формат	`rationalize(x, ApproximateFloats)` `coerce(x, Dom::Rational)`	`rats(x, strlen)`

Преобразование числа в целое	`coerce(x, Dom::Integer)`	`int8(x)`
		`int16(x)`
		`int32(x)`
		`int64(x)`

Вещественная часть комплексного числа	`real(x)`	`real(x)`
Мнимая часть комплексного числа	`imag(x)`	`imag(x)`
Округление в сторону нуля	`fix(x)`	`fix(x)`
Преобразование целого числа в его строковое представление	`int2text(x)`	`int2str(x)`
Преобразование текста в целое число	`text2int(x)`	`str2int(x)`
Преобразование текста в выражение (функцию)	`text2expr(x)`	`str2func(x)`
Преобразование выражения (функции) в текст	`expr2text(x)`	`char(x)`
Преобразование объекта в выражение	`expr(x)`	

получится выражение: `x + 6.93`, а в результате выполнения команды `vpa(y)` получится:
`x + 6.928203230275509174109785366023 5.`

Сопоставление других стандартных функций преобразования типов MuPAD и MatLab приведено в табл. 11.6.

Для проверки истинности логических выражений при выполнении символьных вычислений в MatLab можно использовать одну из следующих функций:

```
isAlways(<логическое выражение>)
logical(<логическое выражение>)
isequal(<выражение1>, <выражение2>)
```

Параметр `<логическое выражение>` должен содержать одну или несколько операций отношения. Каждая из функций `isAlways`, `logical`, `isequal` возвращает 1, если `<логическое выражение>` истинно или `<выражение1>` эквивалентно `<выражению2>` (при использовании функции `isequal`). В противном случае каждая из этих функций возвращает 0. Например, чтобы сравнить два символьных выражения `x + 5` и `y - 3`, можно написать одну из следующих команд: `isAlways(x + 5 == y - 3)`, `logical(x + 5 == y - 3)` или `isequal(x + 5, y - 3)`. Если ранее в программе были выполнены команды `syms t; x = t^2 - 2; y = t^2 + 4`, то в результате любой их приведенных выше проверок получится 0, так как выражения `x + 5` и `y - 3` не равны.

Этим функциям в MuPAD соответствует функция `is(<логическое выражение>)`.

11.2. Символьные функции

Традиционные (несимвольные) функции языка программирования MatLab имеют дело с числовыми или строковыми входными параметрами (аргументами) и возвращают также числовые или строковые значения (числа, массивы). Символьные функции, как правило, имеют символьные аргументы (входные параметры) и возвращают результат в символьном виде.

Пакет Symbolic Math Toolbox позволяет создавать нестандартные символьные функции несколькими способами. Если соответствие между аргументами и результирующим значением функции можно записать в виде одного математического выражения, то такая функция (как и в MuPAD) считается простой. Для создания простой символьной функции

в MatLab достаточно у переменной, которой присваивается соответствующее символьное выражение, указать символьный аргумент:

```
<Имя>(<аргумент>) = <символьное выражение>
<Имя>(<аргумент1>, <аргумент2>) =
                                <символьное выражение>
```

Другой способ создания (объявления) простой символьной функции аналогичен способу, используемому в MuPAD (см. разд. 2.7.2):

```
<Имя> = @ (<аргумент>) <символьное выражение>
<Имя> = @ (<аргумент1>, <аргумент2>)
                                <символьное выражение>
```

Кроме того, преобразовать `<символьное выражение>` в простую функцию можно с помощью стандартной функции `symfun`, которую в этом случае вызывают следующим образом:

```
<Имя> = symfun(<символьное выражение>, <аргумент>)
<Имя> = symfun(<символьное выражение>,
                              <аргумент1>, <аргумент2>)
```

Во всех перечисленных выше случаях в качестве каждого из `<аргументов>` следует использовать идентификатор объявленной ранее символьной переменной. Переменная `<имя>` далее будет считаться идентификатором функции. `<Символьное выражение>` должно быть записано с использование всех переменных, которыми обозначены аргументы объявляемой функции.

Например, если ранее в программе была объявлена символьная переменная `x` (например, `syms x`), то чтобы функцию $f(x) = x^2$ объявить в программе как символьную, можно использовать одну из следующих команд: `f(x) = x^2`, `f = @(x) x^2` или `f = symfun(x^2, x)`. Другие примеры объявления символьных функций в командном окне MatLab приведены на рис. 11.4 (в строках, отмеченных символом >>, записаны команды; объявленные переменные и функции отображаются в окне Workspace, а их значения – в командном окне с помощью вывода по умолчанию).

Чтобы преобразовать в символьную объявленную ранее несимвольную функцию, в большинстве случаев достаточно вызвать эту функцию с символьным аргументом следующим образом:

```
<Имя>(<символьный аргумент>) =
          <несимвольная функция>(<символьный аргумент>)
```

```
<Имя>(<символьный аргумент1>,
      <символьный аргумент2>) =
        <несимвольная функция>(<символьный аргумент1>,
                                <символьный аргумент2>)
```

Исходная `<несимвольная функция>` может быть объявлена в соответствии с правилами MatLab либо в отдельном файле (в виде `function … = <несимвольная функция>(<аргумент>, …) … end`), либо в том же m-файле, но немного ранее в виде `<несимвольная функция> = @ (<аргумент>, …) <выражение>`.

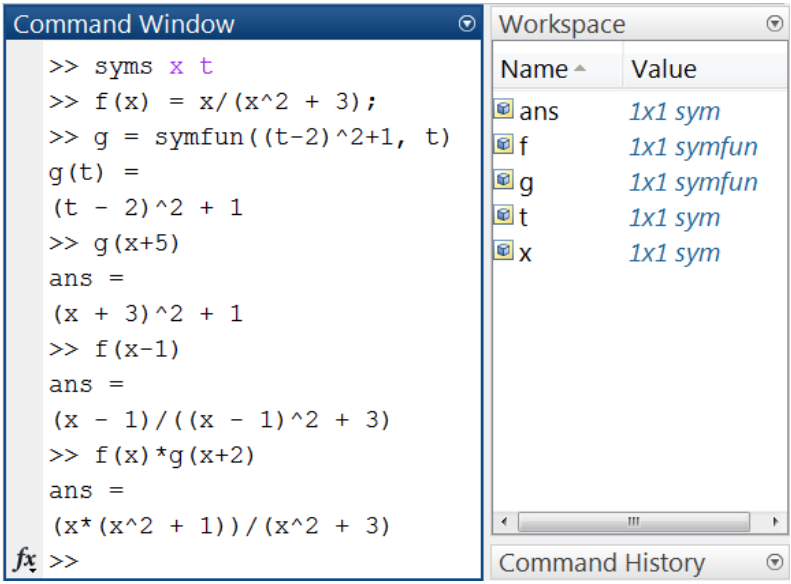


Рис. 11.4. Объявление и вызов символьных функций

Например, если в отдельном m-файле записана несимвольная функция `function y = g(x) y = x^2; end`, то с помощью команд `syms z; f(z) = g(z)` в другом m-файле из той же папки можно создать аналогичную ей символьную функцию `f(z)`.

Некоторые задачи (например, дифференциальные уравнения) записываются через символьные функции неизвестного вида. Чтобы объявить

такие функции, можно воспользоваться оператором `syms` следующим образом:

```
syms <имя>(<аргумент>)
syms <имя>(<аргумент1>, <аргумент2>)
```

Например: `syms x(t) y(t)`. При необходимости можно указать свойства (см. разд. 11.1) объявляемых таким образом функций.

Для вызова объявленной ранее символьной функции необходимо указать ее идентификатор (имя) и (в круглых скобках через запятую) список обязательных аргументов. В качестве аргументов символьных функций могут использоваться числовые значения, символьные переменные или выражения (числовые или символьные). Например, если ранее в программе была объявлена символьная функция `f(x) = 2*sin(x)`, то для ее вызова можно написать: `f(pi/3)`, `f(t^2 + t)` и т.д. Если `t` объявлена в программе как символьная переменная, в результате такого вызова функции получится `3^(1/2)` и `2*sin(t^2 + t)` соответственно.

Если при вызове символьной функции в качестве ее аргумента используется массив, то в результате автоматически формируется массив того же размера, элементами которого являются значения вызываемой функции с аргументами, соответствующими элементам исходного массива. Например, если ранее была объявлена символьная переменная `x` и символьная функция `f(x) = x^2 - 3`, то в результате выполнения команд `A = [2, 3, x, x + 1, x^2]; f(A)` получится массив `[1, 6, x^2 - 3, (x + 1)^2 - 3, x^4 - 3]`.

Если при вычислении значений символьной функции ее аргумент необходимо автоматически изменять в некоторых пределах от начального значения до конечного с постоянным шагом, то для этого в MatLab можно использовать оператор двоеточие (`:`) следующим образом (аналогично оператору `$` в MuPAD):

```
<Массив> = <начальное значение>:<шаг>:
                                <конечное значение>;
<Результат> = <функция>(<массив>)
```

или

```
<Результат> = <функция>(
      <начальное значение>:<шаг>:<конечное значение>)
```

В качестве параметра `<функция>` необходимо использовать идентификатор (имя) объявленной ранее символьной функции. В обоих случаях сначала с помощью оператора `:` создается одномерный массив значений (от `<начального значения>` до `<конечного>` с заданным `<шагом>`),

затем каждый элемент массива подставляется в <функцию> в качестве аргумента, а из вычисленных значений формируется другой массив той же длины. При этом массив значений, подставляемых в функцию, можно создать как с присвоением его отдельной переменной, так и без присвоения. Например, если ранее в программе была объявлена символьная функция `f(x) = x^2`, то с помощью команды `f(x - 3:2:x + 5)` можно получить массив символьных выражений: `[(x - 3)^2, (x - 1)^2, (x + 1)^2, (x + 3)^2, (x + 5)^2]`.

Чтобы в MatLab для объявленной ранее простой символьной функции получить информацию, от каких аргументов она зависит и какой формулой выражается, можно воспользоваться функциями `argnames` и `formula` соответственно:

<Результат> = `argnames(`<функция>`)`

<Результат> = `formula(`<функция>`)`

В качестве параметра <функция> необходимо использовать идентификатор (имя) объявленной ранее символьной функции. Функция `formula` возвращает символьное выражение, а функция `argnames` – массив символьных переменных, соответствующих аргументам указанной <функции>.

Например, если ранее была объявлена символьная функция `g(x, y) = sin(x) + cos(y)`, то в результате выполнения команды `argnames(g)` получится массив `[x, y]`, а в результате выполнения команды `formula(g)` получится выражение `sin(x) + cos(y)`.

При необходимости нестандартную символьную функцию MatLab можно преобразовать в нестандартную несимвольную с помощью стандартной функции `matlabFunction` следующим образом:

<Имя> = `matlabFunction(`<символьная функция>`, `<опции>`)`

В качестве обязательного параметра <символьная функция> необходимо указать либо символьное выражение, либо идентификатор объявленной ранее символьной функции (без указания аргументов). Если выполняется преобразование символьной функции, то аргументы вновь созданной функции будут совпадать с аргументами исходной функции. Если выполняется преобразование символьного выражения, то сначала в него подставляются все присвоенные ранее переменные, а затем в качестве аргументов вновь созданной функции выбираются входящие в полученное выражение незаданные ранее символьные переменные. Вместо параметра <символьная функция> также можно указать через запятую

идентификаторы нескольких функций (или несколько выражений). В этом случае из всех перечисленных функций (выражений) будет сформирована одна несимвольная функция MatLab.

Функцию `matlabFunction` можно вызвать со следующими <опциями>: `'File'` – позволяет задать название файла, в котором будет храниться сгенерированная функция; `'Optimize'` (возможные значения: `true` по умолчанию или `false`) – позволяет указать, необходимо ли выполнять оптимизацию программного кода созданной функции; `'Sparse'` (возможное значения: `false` по умолчанию или `true`) – позволяет указать способ преобразования символьной матрицы в числовую; `'Vars'` – позволяет указать, в какой последовательности необходимо расположить аргументы создаваемой функции (символьные переменные перечисляются через запятую в квадратных скобках); `'Outputs'` – позволяет указать, какие переменные необходимо использовать в качестве результирующих (выходных) параметров формируемой функции (идентификаторы переменных перечисляются в апострофах через запятую в фигурных скобках). Для каждой <опции> необходимо указать через запятую ее название и значение. Например, чтобы сохранить вновь созданную функцию в файле с названием Func.m, можно выполнить следующие команды: `syms x; f(x) = x^3 - sin(x); matlabFunction(f, 'File', 'Func')`. Название файла записывается в апострофах `' '` без расширения. В результате выполнения приведенных выше команд в текущей папке будет создан файл Func.m, который можно открыть в редакторе m-файлов:

```
function f = Func(x)
%FUNC
%   F = FUNC(X)
%   This function was generated by the Symbolic
%   Math Toolbox version 5.10.  05-May-2016 17:29:08
t2 = x.^2;
f = -sin(x) + t2.*x;
```

Если функция `matlabFunction` вызывается с <опцией> `'File'`, это следует делать без присвоения (переменную <имя> и символ = необходимо опустить). В этом случае в дальнейшем идентификатором функции будет считаться название файла.

Если функция `matlabFunction` вызывается без <опции> `'File'`, несимвольная функция создается в памяти MatLab и отображается в командном окне как @-функция. В этом случае в дальнейшем

идентификатором функции считается переменная <имя>, которую необходимо указать перед оператором присвоения при вызове функции matlabFunction. Например, после выполнения команд syms x; f(x) = x^3 - sin(x); Func = matlabFunction(f) в командном окне появится: Func = @(x) -sin(x) + x.^3.

Вызов несимвольной функции, созданной с помощью функции matlabFunction, осуществляется по обычным правилам (например, Func(3)).

11.3. Управление из MatLab выполнением программ, написанных в MuPAD

MatLab позволяет не только запускать приложение MuPAD и выполнять в нем символьные вычисления, но и встраивать результаты, полученные в MuPAD, в программы MatLab. Для этого с помощью специальных команд нужно открыть (загрузить в MuPAD) созданный ранее файл MuPAD, передать в него определенную информацию из командного окна MatLab, выполнить символьные вычисления в MuPAD, вернуть результат в MatLab и закрыть файл MuPAD. Соответствующая последовательность команд может быть записана в m-файле или непосредственно введена в командном окне MatLab.

Чтобы в процессе выполнения m-файла открыть существующий файл MuPAD (с расширением mn или mu), можно воспользоваться одной из следующих функций:

```
<Идентификатор файла> = mupad('<mn-файл>');
<Идентификатор файла> = openmn('<mn-файл>');
<Идентификатор файла> = openmu('<mu-файл>');
```

В качестве параметров <mn-файл> или <mu-файл> необходимо указать название файла, которое должно включать расширение и путь к файлу (если файл находится в той же папке, что и использующая его программа, путь к файлу можно не указывать). Название файла следует записывать в апострофах ' ', а папки, в которых расположен файл, отделять друг от друга одним символом \.

В результате выполнения функции mupad запускается приложение MuPAD, в котором откроется указанный файл. Если при вызове функции mupad параметр '<mn-файл>' опустить (не указывать), то в запущенном приложении создается новый mn-файл. Функции openmn и openmu следует использовать, если приложение MuPAD уже запущено. Если

перед вызовом функции openmn или openmu приложение MuPAD не было запущено, сначала оно запускается, а затем в него автоматически загружается указанный файл.

Каждая из функций mupad, openmn, openmu возвращает <идентификатор файла>, с помощью которого в дальнейшем можно организовать управление соответствующим файлом. Поэтому результат вызова каждой из этих функций рекомендуется присвоить уникальной в пределах данной программы переменной и использовать ее далее для корректного обращения к файлу.

Например: My_mn = mupad('Test1.mn');
Fmn1 = openmn('D:\Work\examples\example1.mn');

Чтобы закрыть открытый ранее файл, необходимо использовать функцию close следующим образом:

close(<идентификатор файла>);
close([<идентификатор файла1>,
<идентификатор файла2>, …]);

В качестве <идентификатора файла> следует указать идентификатор переменной, которой ранее был обозначен соответствующий открытый файл. Если необходимо закрыть несколько файлов, то из их идентификаторов необходимо сформировать одномерный массив-строку и этот массив использовать в качестве параметра функции close. Тогда будут закрыты все перечисленные в массиве файлы.

Например: close(My_mn); close(Fmn1);

Начиная с MatLab 2014 массив идентификаторов всех открытых в данный момент файлов MuPAD можно получить с помощью функции allMuPADNotebooks, которая вызывается без параметров:

<Массив идентификаторов> = allMuPADNotebooks;

В качестве параметра <массив идентификаторов> следует указать идентификатор переменной, которой будет присвоен сформированный функцией allMuPADNotebooks массив. К элементам этого массива можно обращаться обычным образом: указав после его названия в круглых скобках номер элемента. Например: MNfiles = allMuPADNotebooks; close(MNfiles(1));

Начиная с MatLab 2014 с помощью функции evaluateMuPADNotebook можно инициировать выполнение вычислений в одном или нескольких открытых ранее файлах MuPAD следующим образом:

evaluateMuPADNotebook(<идентификатор файла>, <опция>);

```
evaluateMuPADNotebook([<идентификатор файла1>,
                    <идентификатор файла2>, …], <опция>);
```

В качестве <идентификатора файла> следует указать идентификатор переменной, которой ранее был обозначен соответствующий открытый в MuPAD файл. Если необходимо выполнить вычисления, сохраненные в разных файлах, то из идентификаторов этих файлов необходимо сформировать одномерный массив-строку и этот массив использовать в качестве первого параметра функции evaluateMuPADNotebook. Тогда будут выполнены вычисления во всех перечисленных в массиве файлах последовательно.

Функцию evaluateMuPADNotebook можно вызвать только с одной <опцией>: IgnoreErrors, которая может принимать значения true или false. В MatLab, чтобы задать <опцию>, необходимо последовательно через запятую указать сначала ее название, а затем значение. Название <опции> следует указывать в апострофах ' '. Например: evaluateMuPADNotebook(Fnm1, 'IgnoreErrors', true) или evaluateMuPADNotebook(Fnm1, 'IgnoreErrors', false). Если <опция> IgnoreErrors имеет значение true, то возникающие при выполнении вычислений ошибки игнорируются, а вычисления продолжаются дальше. Если <опция> IgnoreErrors не указана или имеет значение false (по умолчанию), а при выполнении вычислений возникает ошибка, то вычисления прерываются.

Функция evaluateMuPADNotebook для каждого файла возвращает 1 (или 0), если вычисления в нем были выполнены без ошибок (с ошибками).

Чтобы значение переменной, присвоенной в файле MuPAD, передать в MatLab, можно использовать функцию getVar следующим образом:

```
<Переменная MatLab> = getVar(<идентификатор файла>,
                                    '<переменная MuPAD>')
```

В качестве <идентификатора файла> следует указать идентификатор переменной, которой ранее был обозначен соответствующий открытый в MuPAD файл. В качестве параметра <переменная MuPAD> необходимо указать идентификатор переменной, значение которой (присвоенное в файле MuPAD) необходимо передать в MatLab. Идентификатор <переменной MuPAD> записывается в апострофах ' '. Возвращаемое функцией getVar значение рекомендуется присвоить <переменной MatLab>. Это обеспечит корректное использование полученных

результатов в дальнейшем. Например, если в открытом ранее mn-файле, обозначенном идентификатором `Fmn1`, переменной `res` присваивается символьное выражение, то чтобы это выражение стало доступным в MatLab, можно написать команду `z = getVar(Fmn1, 'res')`.

Если необходимо, наоборот, передать информацию из MatLab в MuPAD, можно использовать функцию `setVar` следующим образом:

```
setVar(<идентификатор файла>, '<переменная MuPAD>',
                                <выражение MatLab>)
```

Аналогично предыдущей функции в качестве `<идентификатора файла>` следует указать идентификатор переменной, которой ранее был обозначен соответствующий открытый в MuPAD файл. В качестве параметра `<переменная MuPAD>` необходимо указать идентификатор переменной, которой в файле MuPAD будет присвоена переданная из MatLab информация. Идентификатор `<переменной MuPAD>` записывается в апострофах `' '`. В качестве параметра `<выражение MatLab>` можно указать символьное или числовое выражение, записанное по правилам MatLab, или идентификатор переменной, которой оно ранее обозначено. В результате в указанном файле `<переменной MuPAD>` будет присвоено `<выражение MatLab>`. Например, если открытый ранее mn-файл обозначен идентификатором `Fmn1`, то с помощью команд `syms x; y = exp(-x); setVar(Fmn1, 'y', y)` можно из MatLab передать в этот файл выражение e^{-x}, присвоив его там переменной `y`. Следует помнить, что при вызове функции `setVar` нельзя использовать функции MuPAD в качестве параметра `<выражение MatLab>`.

Обратите внимание! Если значение `<переменной MuPAD>` передано в файл MuPAD с помощью функции `setVar`, а после этого планируется выполнить вычисления, записанные в этом файле, то использование в mn-файле функции `reset()` может привести к удалению переданной из MatLab информации.

На рис. 11.5 приведен пример выполнения символьных вычислений в MatLab с использованием программы, написанной в MuPAD: текст вызываемого mn-файла в редакторе MuPAD; текст программы в редакторе m-файлов; результаты выполнения программы в командном окне MatLab.

Чтобы из MatLab выполнить только одну функцию или команду MuPAD, а полученный результат сразу же вернуть в MatLab, можно

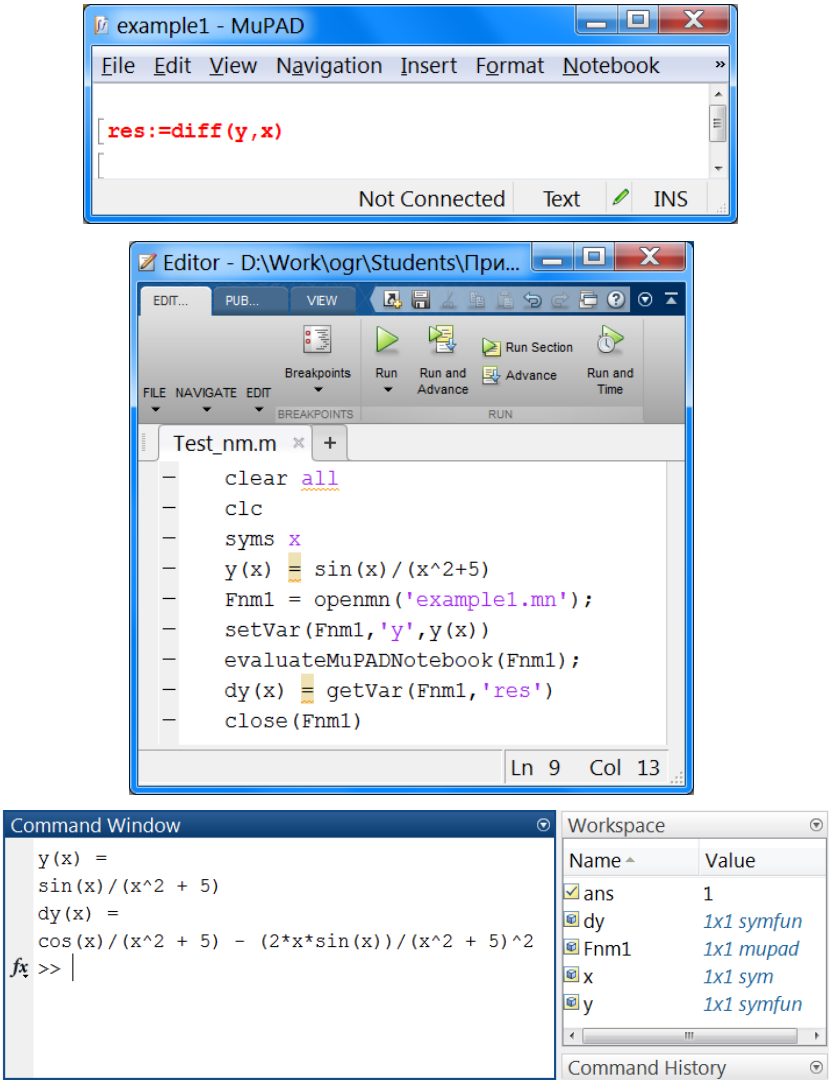


Рис. 11.5. Использование программы, написанной в MuPAD, для выполнения символьных вычислений в MatLab

использовать функции feval и evalin соответственно. Эти функции вызывают следующим образом:

```
<Результат> = feval(symengine, '<функция MuPAD>',
                                <параметр1>, <параметр2>, …)
<Результат> = evalin(symengine, '<команда MuPAD>')
```

При вызове функции feval необходимо как отдельные параметры указать название <функции MuPAD> (в апострофах ' ') и все необходимые для ее вызова параметры. Например, в результате выполнения команд syms x; y = exp(-x) + sin(x + 1); z = feval(symengine, 'op', y, 2) получится символьное выражение sin(x + 1), которое будет присвоено переменной z.

При вызове функции evalin и название функции MuPAD, и ее параметры следует записывать в виде одного параметра <команда MuPAD> (в апострофах ' ') с соблюдением синтаксиса, принятого в MuPAD. Например, в результате выполнения команд syms x; y = exp(-x) + sin(x + 1); z = evalin(symengine, ['op(' char(y) ', 2)']) также получится символьное выражение sin(x+1), которое будет присвоено переменной z. В примере использована функция char, которая преобразует символьное выражение в строковое, а с помощью [] три строки объединены в одну. Функция evalin полезна, особенно в тех случаях, когда в MatLab не удается найти функцию, аналогичную функции MuPAD.

При выполнении команд и функций MuPAD через feval и evalin окно приложения MuPAD на экране не отображается.

Чтобы в MatLab загрузить и выполнить mu-файл (без отображения окна приложения MuPAD), можно воспользоваться функцией read, которая включена в Symbolic Math Toolbox начиная с 2011 г. Для этого функцию read вызывают следующим образом:

```
read(symengine, '<mu-файл>')
```

В качестве параметра <mu-файл> необходимо указать название файла, которое должно включать расширение и путь к файлу (если файл находится в той же папке, что и загружающая его программа, путь к файлу можно не указывать). Название файла следует записывать в апострофах ' ', а папки, в которых расположен файл, отделять друг от друга одним символом \. Следует отметить, что mu-файл можно создать с помощью редактора m-файлов, не вызывая MuPAD.

Если загруженный с помощью функции `read` mu-файл содержит одну или несколько нестандартных функций, то они становятся доступными для использования в MatLab через функции `feval` или `evalin`. Например, если в файле Proizvod.mu записана функция `func := (y, x) -> diff(op(y, nops(y)), x, x)`, которая вычисляет вторую производную по `x` от последнего операнда выражения `y`, тогда в результате выполнения команд `syms x; y = exp(-x) + sin(3*x + 1); read(symengine, 'Proizvod.mu'); z = feval(symengine, 'func', y, x)` получится символьное выражение `-9*sin(3*x + 1)`, которое будет присвоено переменной `z`.

Если необходимо очистить память приложения MuPAD (удалить все переменные и результаты вычислений), то из MatLab можно воспользоваться функцией `reset` следующим образом:

```
reset(symengine)
```

Кроме того, следует помнить, что при выполнении символьных вычислений в MuPAD из m-файлов или командного окна MatLab команда

```
clear all
```

удаляет не только переменные и результаты вычислений MatLab, но и переменные и результаты вычислений MuPAD.

В MatLab 2016 появилась дополнительная функция

```
convertMuPADNotebook(<файл MuPAD>,
                     <файл MatLabLiveScript>)
```

которая позволяет конвертировать программу, написанную в `<файле MuPAD>`, в `<файл MatLabLiveScript>` с расширением mlx. С особенностями использования этой функции можно познакомиться в справочной системе MatLab в разделе Symbolic Math Toolbox.

Таким образом, благодаря перечисленным выше функциям MatLab позволяет так писать программы (m-файлы), чтобы использовать в них символьные вычисления, запрограммированные в MuPAD.

11.4. Стандартные функции MatLab для выполнения символьных вычислений

Пакет Symbolic Math Toolbox не только позволяет внедрять в программы MatLab вычисления, выполненные в MuPAD, но и предоставляет возможность выполнять символьные вычисления в m-файлах без привлечения MuPAD. Для этого в пакет Symbolic Math Toolbox включены стандартные функции для выполнения аналитических расчетов, которые

доступны, если в состав приобретенной лицензии MatLab входят символьные вычисления. Следует отметить, что функции для выполнения численных расчетов включены в основное ядро MatLab и доступны без приобретения дополнительной лицензии. В m-файлах MatLab (при наличии соответствующей лицензии) одинаково доступны функции как для выполнения аналитических, так и для выполнения численных расчетов (при этом не требуется ни подключения библиотек, ни использования специальных команд). Поэтому далее перечислены не только функции, входящие в пакет Symbolic Math Toolbox, но и функции, включенные в ядро MatLab.

Обратите внимание! Далее при описании некоторых функций пакета Symbolic Math Toolbox указан год, начиная с которого данная функция включена в пакет Symbolic Math Toolbox или в ядро MatLab. Например, `sign(x)` 2013 означает, что функция `sign` доступна в MatLab версии 2013 и выше.

11.4.1. Элементарные и специальные математические функции

И в MuPAD, и в MatLab значения многих элементарных и специальных математических функций можно вычислять как численно, так и аналитически. Как видно из табл. 11.7, 11.8, синтаксис этих функций максимально приближен к принятым в математике обозначениям, но в MuPAD и в MatLab для некоторых из них используются разные идентификаторы. Например, функция для вычисления арксинуса в MuPAD записывается как `arcsin(x)`, а в MatLab – как `asin(x)`; функция для вычисления факториала в MuPAD записывается как `n!`, а в MatLab – как `factorial(n)`.

Если в качестве аргумента элементарной или специальной математической функции используется символьное выражение (или число, преобразованное в символьный формат), вычисление этой функции осуществляется аналитически, а результат записывается в рациональном виде. Например, в результате выполнения команд `syms x positive; sqrt(x^3/sym(5))` получится символьное выражение `5^(1/2)*x^(3/2)/5`. Если в качестве аргумента функции используется числовое значение (не преобразованное в символьный формат), вычисление этой функции осуществляется численно, а результат записывается в десятичном формате, установленном по умолчанию. Например, в результате

выполнения команд `x = 2; sqrt(x^3/5)` получится числовое значение `1.2649`.

Таблица 11.7

Элементарные математические функции в MuPAD и MatLab

	Математика	**MuPAD**	**MatLab**
Косинус	$\cos x$	`cos(x)`	`cos(x)`
Синус	$\sin x$	`sin(x)`	`sin(x)`
Тангенс	$\mathrm{tg}\, x$	`tan(x)`	`tan(x)`
Арксинус	$\sin^{-1} x$	`arcsin(x)`	`asin(x)`
Арккосинус	$\cos^{-1} x$	`arccos(x)`	`acos(x)`
Арктангенс	$\mathrm{tg}^{-1} x$	`arctan(x)`	`atan(x)`
Косинус гиперболический	$\mathrm{ch}\, x$	`cosh(x)`	`cosh(x)`
Синус гиперболический	$\mathrm{sh}\, x$	`sinh(x)`	`sinh(x)`
Тангенс гиперболический	$\mathrm{tgh}\, x$	`tanh(x)`	`tanh(x)`
Арксинус гиперболический	$\mathrm{sh}^{-1} x$	`arcsinh(x)`	`asinh(x)`
Арккосинус гиперболический	$\mathrm{ch}^{-1} x$	`arccosh(x)`	`acosh(x)`
Арктангенс гиперболический	$\mathrm{tgh}^{-1} x$	`arctanh(x)`	`atanh(x)`
Экспонента	e^x	`exp(x)`	`exp(x)`
Натуральный логарифм	$\ln x$	`log(x)` или `ln(x)`	`log(x)`
Десятичный логарифм	$\lg x$	`log10(x)`	`log10(x)`
Логарифм по основанию 2	$\log_2 x$	`log2(x)`	`log2(x)`
Комплексное сопряжение	x^*	`conjugate(x)`	`conj(x)`
Знак числа		`sign(x)`	`sign(x)` 2013

Таблица 11.8

Специальные математические функции в MuPAD и MatLab

Факториал $n!$	
MuPAD	`n!`
MatLab	`factorial(n)`
Дельта-функция Дирака δ `(x)`	
MuPAD	`dirac(x)`
MatLab	`dirac(x)`
Дельта символ Кронекера $\delta_{n,m}$	
MuPAD	`kroneckerDelta(n, m)`
MatLab	`kroneckerDelta(n, m)`
Гамма-функция $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$	
MuPAD	`gamma(x)`
MatLab	`gamma(x)`
Бета-функция $B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$	
MuPAD	`beta(x, y)`
MatLab	`beta(x, y)`
Биномиальные коэффициенты $\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$	
MuPAD	`binomial(n, k)`
MatLab	`nchoosek(n, k)`
Дигамма-функция $\psi(x) = \frac{d}{dx}\ln\Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}$	
MuPAD	`psi(x)`
MatLab	`psi(x)`
Дилогарифм $Li_2(x) = -\int_0^x \frac{\ln(1-t)}{t} dt$	
MuPAD	`dilog(x)`
MatLab	`dilog(x)`

Модифицированная функция Бесселя первого рода $$I_n(x)=\frac{(\pi/2)^n}{\sqrt{\pi}\Gamma(n+1/2)}\int_0^{\pi} e^{x\cos t}\sin^{2n} t\,dt$$	
MuPAD	`besselI(n, x)`
MatLab	`besseli(n, x)`
Функция Бесселя первого рода $$J_n(x)=\frac{(\pi/2)^n}{\sqrt{\pi}\Gamma(n+1/2)}\int_0^{\pi}\cos(x\cos t)\sin^{2n} t\,dt$$	
MuPAD	`besselJ(n, x)`
MatLab	`besselj(n, x)`
Модифицированная функция Бесселя второго рода $$K_n(x)=\frac{\pi}{2}\frac{I_{-n}(x)-I_n(x)}{\sin n\pi}$$	
MuPAD	`besselK(n, x)`
MatLab	`besselk(n, x)`
Функция Бесселя второго рода $Y_n(x)=\dfrac{J_n(x)\cos n\pi - J_{-n}(x)}{\sin n\pi}$	
MuPAD	`besselY(n, x)`
MatLab	`bessely(n, x)`
Интегральная показательная функция $Ei(x)=\int_{-\infty}^{x}\frac{e^t}{t}dt$	
MuPAD	`Ei(x)`
MatLab	`ei(x)`
Интегральный логарифм $li(x)=\int_0^x\frac{dt}{\ln t}$	
MuPAD	`Li(x)`
MatLab	`logint(X)`
Интегральный косинус $Ci(x)=-\int_x^{\infty}\frac{\cos t}{t}dt$	
MuPAD	`Ci(x)`
MatLab	`cosint(x)`

Интегральный синус $Si(x)=\int_0^x \frac{\sin t}{t}dt$	
MuPAD	`Si(x)`
MatLab	`sinint(x)`
Функция ошибок $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$	
MuPAD	`erf(x)`
MatLab	`erf(x)`
Дополнительная функция ошибок $erfc(x)=1-erf(x)$	
MuPAD	`erfc(x)`
MatLab	`erfc(x)`
Функция Доусона $e^{-x^2}\int_0^x e^{t^2}dt$	
MuPAD	`dawson(x)`
MatLab	`dawson(x)`
С-интеграл Френеля $C(x)=\int_0^x \cos\left(\frac{\pi}{2}t^2\right)dt$	
MuPAD	`fresnelC(x)`
MatLab	`fresnelc(x)`
S-интеграл Френеля $S(x)=\int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$	
MuPAD	`fresnelS(x)`
MatLab	`fresnels(x)`
Полный эллиптический интеграл первого рода $K(x)=\int_0^{\pi/2}\frac{d\theta}{\sqrt{1-x^2\sin^2\theta}}$	
MuPAD	`ellipticK(x)`
MatLab	`ellipticK(x)`

Эллиптический интеграл первого рода (неполный) $$F(\varphi, x) = \int_0^{\varphi} \frac{d\theta}{\sqrt{1 - x^2 \sin^2 \theta}}$$	
MuPAD	`ellipticF(φ, x)`
MatLab	`ellipticF(φ, x)`
Эллиптический интеграл второго рода $E(\varphi, x) = \int_0^{\varphi} \sqrt{1 - x^2 \sin^2 \theta}\, d\theta$	
MuPAD	`ellipticE(φ, x)`
MatLab	`ellipticE(φ, x)`
Эллиптический интеграл третьего рода $$\Pi(c; \varphi, x) = \int_0^{\varphi} \frac{d\theta}{(1 + c^2 \sin^2 \theta)\sqrt{1 - x^2 \sin^2 \theta}}$$	
MuPAD	`ellipticPi(c, φ, x)`
MatLab	`ellipticPi(c, φ, x)`
Многочлен Лагерра степени n $L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} \left(e^{-x} x^n\right)$	
MuPAD	`laguerreL(n, x)`
MatLab	`laguerreL(n, x)`
Многочлен Эрмита степени n $H_n(x) = \sum_{k=0}^{n/2} (-1)^k \frac{n!}{k!(n-2k)!} (2x)^{n-2k}$	
MuPAD	`orthpoly::hermite(n, x)`
MatLab	`hermiteH(n, x)`
Многочлен Лежандра степени n $P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left(x^2 - 1\right)^n$	
MuPAD	`orthpoly::legendre(n, x)`
MatLab	`legendreP(n, x)`
Многочлен Чебышева первого рода степени n $$T_n(x) = \sum_{k=0}^{n/2} \binom{n}{2k} (x^2 - 1)^k x^{n-2k}$$	
MuPAD	`orthpoly::chebyshev1(n, x)`

Окончание табл. 11.8

MatLab	`chebyshevT(n, x)`
Многочлен Чебышева второго рода степени n $$U_n(x) = \sum_{k=0}^{n/2} \binom{n+1}{2k+1} (x^2-1)^k x^{n-2k}$$	
MuPAD	`orthpoly::chebyshev2(n, x)`
MatLab	`chebyshevU(n, x)`

11.4.2. Графики символьных функций

В табл. 11.9 сопоставлены функции MuPAD и Symbolic Math Toolbox, предназначенные для построения двумерных графиков. Функции MuPAD, перечисленные в табл. 11.9, детально описаны в предыдущих главах пособия. Остановимся подробнее на функциях MatLab (в том числе из пакета Symbolic Math Toolbox), предназначенных для построения двумерных графиков.

Для построения двумерного графика по числовым данным в MatLab используется функция `plot`, которую вызывают следующим образом:

`<объект> = plot(<массив X>, <массив Y>, <опции>)`

Обязательные параметры `<массив X>` и `<массив Y>` должны содержать одинаковое количество *x*- и *y*-координат точек (в двух разных массивах соответственно), через которые будет проходить график. Эти массивы должны содержать только числовые (несимвольные) значения. С помощью `<опций>` можно управлять цветом и стилем линии, а также стилем изображения точек графика (подробно см. в справочной системе MatLab).

Массивы `<массив X>` и `<массив Y>` с координатами точек графика можно получить как с помощью несимвольной, так и с помощью символьной функции. Для простых функций это можно реализовать (чаще всего) одним и тем же способом. Например, для объявленной ранее функции `f(x)`:

```
dx = (b - a)/(n - 1); X = a:dx:b; Y = f(X)
```

или с использованием оператора цикла (см. разд. 11.5):

```
dx = (b - a)/(n - 1); X = a:dx:b;
for k = 1:n; Y(k) = f(X(k)); end
```

Тогда в массиве `X` получится n значений, расположенных в интервале `[a, b]` на одинаковом расстоянии `dx` друг от друга, а в массиве `Y` – соответствующие им значения функции `f(x)` одной переменной. Если

переменные a и b не являются символьными, а содержат числовые значения, то в качестве f(x) может использоваться как несимвольная, так и символьная функция – в обоих случаях массив Y будет содержать числовые значения. Если функция f(x) является символьной, то в некоторых случаях она может возвращать числовые значения в рациональном виде. Чтобы представить такие числовые значения в десятичном формате, можно воспользоваться функцией vpa. Тогда программный код для подготовки данных для построения графика объявленной ранее функции f(x) примет вид, например:

```
dx = (b - a)/(n - 1); X = a:dx:b;
for k = 1:n; Y(k) = f(X(k)); end; Y = vpa(Y)
```

Далее для построения графика достаточно написать: plot(X, Y).

Таблица 11.9

Функции для построения двумерных графиков в MuPAD и MatLab

График символьной функции одной переменной	
MuPAD	<объект> := plot::Function2d(f(x), <опции>); plot(<объект>)
MatLab	<объект> = ezplot(f(x), <опции>) <объект> = fplot(f(x), <опции>) 2016
График символьной параметрически заданной функции	
MuPAD	<объект> := plot::Curve2d([x(t), y(t)]); plot(<объект>)
MatLab	<объект> = ezplot(x(t), y(t), <опции>) <объект> = fplot(x(t), y(t), <опции>) 2016
Изображение набора точек	
MuPAD	<объект> := plot::PointList2d([<точка1>, <точка2>, …], <опции>)
MatLab	<объект> = scatter((<массив X>, <массив Y>, <опции>)

Кроме того, для построения графиков символьных функций пакет Symbolic Math Toolbox содержит дополнительные функции, перечисленные в табл. 11.9.

В 2016 г. функция ezplot была заменена функцией fplot, имеющей аналогичное назначение и для построения двумерного графика функции одной переменной, и для построения графика параметрически заданной

функции. Правила вызова функций `ezplot` и `fplot` приведены в табл. 11.9. Так же, как при вызове аналогичных функций MuPAD (`plot::Function2d` и `plot::Curve2d`), при вызове функции `ezplot` (или `fplot`) в качестве параметра `f(x)` можно указывать символьную функцию или символьное выражение, содержащее одну незаданную ранее символьную переменную. При вызове функции `fplot` вместо параметра `f(x)` также можно указать массив символьных функций и (или) выражений (в квадратных скобках через запятую). В результате на одном графике будет построено несколько кривых. По умолчанию для каждой функции `f(x)` на заданном интервале автоматически вычисляются координаты нескольких точек: функция `ezplot` вычисляет координаты точек на интервале [–2π, 2π], а функция `fplot` – на интервале [–5, 5]. Интервал и количество точек для построения графика, а также цвет и стиль линии, стиль точек можно изменять с помощью <опций>. Но у функции `ezplot` с помощью <опции> можно изменить только интервал. А функция `fplot` имеет значительно больше <опций>. С их помощью можно указать интервал (в виде [<начало> <конец>]), стиль линии (в виде специального набора символов), `'MeshDensity'` – количество точек, `'ShowPoles'` (`'on'` или `'off'`) – показывать ли точки, `'Color'` – цвет линии, `'LineWidth'` – толщину линии, `'Marker'` – стиль точек, `'MarkerSize'` – размер точек и т.д. Например, чтобы построить график функции, заданной выражением $50/(x^2 + 2)$ на интервале [–3, 5] можно использовать следующие команды:

```
syms x; ezplot(50/(x^2 + 2), [-3 5]) или
syms x; fplot(50/(x^2 + 2), [-3 5]).
```

В соответствии с правилами MatLab, чтобы при вызове функции `fplot` изменить значение одной из <опций> `'MeshDensity'`, `'ShowPoles'`, `'Color'`, `'Marker'` и т.д., необходимо подряд через запятую указать ее название (в апострофах) и новое значение. Например, в команде `fplot(f(x), 'Color', 'r')` параметры `'Color', 'r'` задают красный цвет линии, соединяющей точки графика (`f(x)` объявлена ранее как символьная функция).

Функция `ezplot` (или `fplot`) возвращает идентификатор созданного ею графического объекта, соответствующего этому графику. Если этот идентификатор присвоить какой-либо переменной <объект>, то с ее помощью можно получать и изменять различные свойства графика.

Например, если выполнить команды `syms x; h = ezplot(50/(x^2 + 2))`, появится графическое окно с изображением соответствующей кривой. Тогда с помощью команды `get(h)` можно вывести в командное окно список названий и значений свойств данного графического `<объекта>`, а с помощью команды `set` можно задать новое значение одному или нескольким свойствам этого объекта. Например, после выполнения команды `set(h, 'Color', 'r')` цвет графика в приведенном выше примере изменится на красный.

С помощью команды `hold` в MatLab можно разрешить (`on`) или запретить (`off`, по умолчанию) отображение следующей кривой на том же графике. Например, с помощью команд `syms x; ezplot(sin(x)); hold on; h = ezplot(cos(x)); set(h, 'Color', 'r'); hold off` можно построить график двух функций (одна синего цвета по умолчанию, а другая – красного).

Для выполнения других настроек графического окна, в котором строится график, можно использовать графические функции MatLab: `title` – задает титульную надпись; `xlabel`, `ylabel` – задают подписи осей координат; `xlim`, `ylim` – задают границы осей координат; `legend` – задает условные обозначения; `grid on` (`off`) – разрешает (запрещает) изображение координатной сетки; `hold on` (`off`) – разрешает (запрещает) совмещение на одном графике различных графических объектов и т.д. Детальнее с правилами использования этих функций можно ознакомиться в справочной системе MatLab. При этом необходимо помнить, что эти функции следует записывать в программе только после вызова функции `ezplot` (`fplot`, `plot` и т.д.), так как они вносят изменения в созданный ранее графический объект.

Для изображения дискретного набора точек в MatLab можно использовать функцию `scatter` (см. табл. 11.9), правила вызова которой аналогичны правилам вызова функции `plot` в MatLab. Однако в опциях этих функций имеются различия, которые подробно описаны в справочной системе MatLab.

Например, чтобы на одном графике изобразить символьную функцию $f(x) = \dfrac{15}{(x-3)^2 + 4}$ и дискретный набор, состоящий из ее значений в семи точках, на интервале [–3, 5], в m-файле можно написать программу:

```
clc          % очистка командного окна
clear all    % очистка памяти
```

```
syms x     % объявление символьной переменной
% объявление символьной функции
f(x) = 15/((x-3)^2+4)
% a, b - границы интервала;
% N - количество точек в дискретном наборе
a = -3; b = 5; N = 7;
% вычисление координат точек для дискретного набора
dx = (b-a)/(N-1);  xx = a:dx:b;
for k=1:N;      yy(k) = f(xx(k)); end
% построение графика символьной функции
fp = ezplot(f(x), [a b]);
% для MatLab 2016: fp = fplot(f(x), [a b]);
set(fp, 'Color','r') % задать цвет графика
% разрешить построение изображения на том же графике
hold on
% изображение дискретного набора точек
sc = scatter(xx,yy);
% задать цвет и способ изображения точек
set(sc, 'MarkerEdgeColor','b', 'Marker','o')
% запретить построение изображения на том же графике
hold off
ylabel('f(x)') % подписать вертикальную ось
grid on        % нанести сетку
```

В этом примере, начиная с версии MatLab 2016, вместо функции `ezplot` следует использовать функцию `fplot`.

11.4.3. Аналитические преобразования математических выражений, уравнений, неравенств

В табл. 11.10 сопоставлены функции MuPAD и Symbolic Math Toolbox, предназначенные для выполнения аналитических преобразований символьных выражений, уравнений, неравенств. Перечисленные в табл. 11.10 функции MuPAD подробно описаны в предыдущих главах пособия. Сравним функции Symbolic Math Toolbox с соответствующими функциями MuPAD. Многие из этих функций в MuPAD и Symbolic Math Toolbox имеют одинаковое название. При этом они могут существенно отличаться не только набором опций (необязательных параметров), но и

правилами записи обязательных параметров (см. табл. 11.10). Например, если в символьном выражении $(x - 2)^2 + (x + 4)^3 - x$ переменную x необходимо заменить на символьное выражение $y - 1$, то и в MuPAD, и в Symbolic Math Toolbox можно использовать функцию `subs`, но в MuPAD исходная переменная `x` и подставляемое выражение `y-1` отделяются друг от друга символом =, а в MatLab они записываются через запятую. В MuPAD: `z := (x - 2)^2 + (x + 4)^3 - x; subs(z, x = y - 1);` в Symbolic Math Toolbox: `syms x y; z = (x - 2)^2 + (x + 4)^3 - x, subs(z, x, y - 1).` В обоих случаях получится выражение: $(y - 3)^2 - y + (y + 3)^3 + 1$, но в командном окне MatLab оно (как и все результаты вычислений) будет записано не в графическом, а в текстовом виде: `(y – 3)^2 – y + (y + 3)^3 + 1`.

Таблица 11.10

Функции преобразования символьных выражений, уравнений, неравенств в MuPAD и MatLab

Подстановка значений	
MuPAD	`evalAt(<исходное выражение>, <старое1> = <новое1>, <старое2> = <новое2>, …)` `subs(<исходное выражение>, <старое1> = <новое1>, <старое2> = <новое2>, …, <опция>)`
MatLab	`subs(<исходное выражение>, <старое>, <новое>)` `subs(<исходное выражение>, [<старое1>, <старое2>, …], [<новое1>, <новое2>, …])`
Подстановка выражений	
MuPAD	`subsex(<исходное выражение>, <старое1> = <новое1>, <старое2> = <новое2>, …, <опция>)`
MatLab	
Упрощение выражений	
MuPAD	`simplify(<исходное выражение>, <опции>)`
MatLab	`simplify(<исходное выражение>, <опции>)`
Приведение дробей к нормальному виду	
MuPAD	`normal(<исходное выражение>, <опции>)` `simplifyFraction(<исходное выражение>, <опции>)`
MatLab	`simplifyFraction(<исходное выражение>, <опции>)`

Продолжение табл. 11.10

Разложение на простые дроби	
MuPAD	partfrac(<исходное выражение>, <аргумент>, <опции>)
MatLab	partfrac(<исходное выражение>, <аргумент>, <опции>)
Упрощение радикалов	
MuPAD	radsimp(<исходное выражение>)
MatLab	
Запись выражения через другие элементарные функции	
MuPAD	rewrite(<исходное выражение>, <целевая функция>)
MatLab	rewrite(<исходное выражение>, <целевая функция>)
Раскрытие скобок	
MuPAD	expand(<исходное выражение>, <ограничение>, … <опции>)
MatLab	expand(<исходное выражение>, <опции>)
Деление полиномов	
MuPAD	factorout(<исходное выражение>, <делитель>)
MatLab	quorem(<исходное выражение>, <делитель>)
Вынесение общих множителей	
MuPAD	factor(<исходное выражение>, <опции>)
MatLab	factor(<исходное выражение>, <опции>)
Компоновка операндов	
MuPAD	collect(<исходное выражение>, [<аргумент1>, <аргумент2>, …], <функция>)
MatLab	collect(<исходное выражение>, [<аргумент1>, <аргумент2>, …])
Запись через функции составных аргументов	
MuPAD	combine(<исходное выражение>, <цель>, <опция>)
MatLab	combine(<исходное выражение>, <цель>, <опции>)
Определение количества операндов (элементов)	
MuPAD	nops(<исходное выражение>)
MatLab	

Продолжение табл. 11.10

Извлечение одного или нескольких операндов	
MuPAD	op(<исходное выражение>, <номер>)
MatLab	
Извлечение всех операндов	
MuPAD	op(<исходное выражение>)
MatLab	children(<исходное выражение>) 2012
Извлечение левой (правой) части уравнения, неравенства	
MuPAD	<левая часть> := lhs(<исходное выражение>) <правая часть> := rhs(<исходное выражение>)
MatLab	
Извлечение числителя (знаменателя) дроби	
MuPAD	<числитель> := numer(<исходное выражение>) <знаменатель> := denom(<исходное выражение>)
MatLab	[<числитель>, <знаменатель>]= numden(<исходное выражение>)
Отложенные вычисления	
MuPAD	hold(…)
MatLab	
Выполнение отложенных вычислений	
MuPAD	eval(…)
MatLab	
Объявление свойств переменных	
MuPAD	assume(<свойство>) <команда> assuming(<свойство>)
MatLab	assume(<свойство>) assume(<переменная>, <свойство>) <переменная> = sym('<переменная>', <свойство>); syms <переменная> <свойство> Свойства: 'integer', 'rational', 'real', 'positive' или неравенство
Удаление свойств переменных	
MuPAD	unassume(<переменная>) unassume([<переменная1>, <переменная2>])
MatLab	assume(<переменная>, 'clear') assume([<переменная1> <переменная2>], 'clear')

Окончание табл. 11.10

Получение информации о свойствах переменной	
MuPAD	`getprop(<переменная>)`
MatLab	`assumptions(<переменная>)`

Некоторые функции, предназначенные для выполнения одних и тех же действий, в MuPAD и в Symbolic Math Toolbox имеют разные названия, но возвращают одинаковый результат. Например, функция для получения текущих свойств переменной в MuPAD называется `getprop`, а в Symbolic Math Toolbox – `assumptions`; для деления полинома на выражение в MuPAD используется функция `factorout`, а в Symbolic Math Toolbox – `quorem`.

Некоторые функции MuPAD и Symbolic Math Toolbox имеют одинаковые название и назначение, но возвращаемый ими результат может отличаться. Например, функция `factor` позволяет разложить арифметическое выражение на простые множители. Но в MatLab 2016 эта функция возвращает массив, состоящий из простых множителей, а в MuPAD – выражение в виде произведения этих множителей, в котором не раскрыты скобки. Чтобы в MatLab 2016 получить такое выражение, необходимо после функции `factor` использовать еще функцию `prod` для перемножения элементов полученного массива. Например, в MuPAD в результате выполнения команд `y1 := -x^3 + 5*x^2 + 2*x - 24; y2 := factor(y1)` получится выражение $-(x-3)(x-4)(x+2)$; чтобы в MatLab 2016 получить такой же результат, следует выполнить команды: `syms x; y1 = -x^3 + 5*x^2 + 2*x - 24, q = factor(y1), y2 = prod(q)`; а вот в предыдущих версиях MatLab достаточно написать: `syms x; y1 = -x^3 + 5*x^2 + 2*x - 24, y2 = factor(y1).`

Некоторые из перечисленных в табл. 11.10 функций MuPAD не имеют аналогов в Symbolic Math Toolbox. Например, в пакете Symbolic Math Toolbox отсутствует функция для определения количества операндов в выражении, а также функция для извлечения операнда с указанным номером. Это объясняется тем, что функция `children` позволяет получить массив, состоящий из операндов; количество операндов будет равно длине этого массива (которую можно определить с помощью функции `length`), а выбрать один операнд можно, обратившись к элементу массива с заданным номером. Например, в результате выполнения команд

`syms x; y = 2*x^3 + 4*x^2 - 3*x + 5, Q = children(y), n = length(Q), Q(2)` получится, что выражение содержит 4 операнда (`n = 4`) и выбран второй операнд: $4x^2$.

Если свойства символьных переменных в MatLab задаются через указание их принадлежности типовым множествам, то в функции `assume` переменная и множество записываются через запятую. Для обозначения множеств используются следующие строковые константы: `'integer'`, `'rational'`, `'real'`, `'positive'` (например: `assume(x, 'real')`). Если для указания свойств переменной используется неравенство, то параметр `<переменная>` можно опустить, а записать только `<свойство>` (например: `assume(x > 2)`). Кроме того, как описано в разд. 11.1, в MatLab свойства символьным переменным можно задавать при их объявлении (например: `syms x 'rational'`), но только в виде принадлежности к определенным числовым множествам.

Некоторые из функций, перечисленных в табл. 11.10, имеют `<опции>`. Опции функций MuPAD перечислены в предыдущих главах пособия. Большинство функций Symbolic Math Toolbox, предназначенных для выполнения аналитических преобразований символьных выражений, уравнений, неравенств, опций не имеют. Поэтому описание опций этих функций в данном пособии не приводится. Подробнее с правилами использования функций Symbolic Math Toolbox, перечисленных в табл. 11.10 и имеющих опции, можно познакомиться самостоятельно в справочной системе MatLab (в разделе Symbolic Math Toolbox).

Например, чтобы в выражении $(2x^2 - 1)(4x^3 + 3x)$ заменить x на $\cos x$, а затем упростить каждый из сомножителей по отдельности, можно в m-файле написать программу:

```
clc         % очистка командного окна
clear all   % очистка памяти
syms x      % объявление символьной переменной
v = (2*x^2-1)*(4*x^3-3*x) % исходное выражение
U = children(v)  % массив из сомножителей
N = length(U)    % количество сомножителей
% замена x на cos(x) в каждом сомножителе
Y = subs(U, x, cos(x))
Y = simplify(Y) % упрощение каждого сомножителя
% перемножение сомножителей:
result = Y(1)*Y(2)*Y(3) % окончательный результат
```

В результате выполнения этой программы в командном окне MatLab получится выражение: `cos(2*x)*cos(3*x)`.

11.4.4. Решение алгебраических и трансцендентных уравнений (неравенств) и их систем

В табл. 11.11 сопоставлены функции MuPAD и MatLab, предназначенные для решения алгебраических и трансцендентных уравнений (неравенств) и их систем. Из этой таблицы видно, что MuPAD содержит значительно больше разнообразных функций для аналитического и численного решения различных уравнений, неравенств и их систем.

Перечисленные в табл. 11.11 функции MuPAD детально описаны в предыдущих главах пособия. Остановимся подробнее на некоторых функциях MatLab из пакета Symbolic Math Toolbox из приведенных в табл. 11.11.

Функция `solve` из пакета Symbolic Math Toolbox аналогична функции `solve` в MuPAD и по названию, и по правилам вызова, но они существенно отличаются способами записи опций. В MatLab (а значит, и для функций из пакета Symbolic Math Toolbox) необходимо указывать через запятую название опции (в апострофах) и ее значение. При этом большинство опций функции `solve` в обоих пакетах имеют сходные названия. Опции функции `solve` из пакета Symbolic Math Toolbox: `'IgnoreAnalyticConstraints'` (`false` по умолчанию или `true`); `'IgnoreProperties'` (`false` по умолчанию или `true`); `'MaxDegree'` (`2` по умолчанию или положительное целое число меньше 5); `'PrincipalValue'` (`false` по умолчанию или `true`); `'Real'` (`false` по умолчанию или `true`); `'ReturnConditions'` (`false` по умолчанию или `true`). Например, в результате выполнения команд `syms x; solve(x^3 – 4 == 0, x, 'MaxDegree', 3)` для уравнения $x^3 - 4 = 0$ можно получить три корня, которые в командном окне будут иметь вид: `2^(2/3)`, `2^(2/3)*((3^(1/2)*i)/2 - 1/2)`, `–2^(2/3)*((3^(1/2)*i)/2 + 1/2)`.

Функции численного решения уравнений и их систем относятся к основному ядру MatLab (а не к пакету Symbolic Math Toolbox), поэтому доступны даже тогда, когда в лицензию MatLab не включены символьные вычисления. В 2012 г. функция `fzero` была заменена функцией

vpasolve. При этом существенно расширились ее возможности. Но функцию vpasolve можно вызвать только с опцией 'random' (false по умолчанию или true).

Таблица 11.11

Функции для решения уравнений, неравенств и их систем в MuPAD и MatLab

Поиск корней многочлена	
MuPAD	RootOf(<многочлен>, <неизвестная>)
MatLab	root(<многочлен>, <неизвестная>, <номер корня>) 2015
Аналитическое решение алгебраических или трансцендентных уравнений и их систем	
MuPAD	isolate(<уравнение>, <неизвестная>) 2015 polylib::realroots(<полином>) linsolve(<уравнения>, <неизвестные>, <опции>) solve(<уравнения>, <неизвестные>, <опции>)
MatLab	solve(<уравнения>, <неизвестные>, <опция>)
Численное решение алгебраических и трансцендентных уравнений и их систем	
MuPAD	numeric::polyroots(<полином>, <опции>) numeric::polyrootbound(<полином>) numeric::realroots(<функция>, <неизвестная> = <начало> .. <конец>, <точность>, <опция>) numeric::realroot(<функция>, <неизвестная> = <начало> .. <конец>, <опция>) numeric::fsolve(<уравнения>, [<неизвестная> = <начало> .. <конец>, …], <опции>) numeric::linsolve(<уравнения>, <неизвестные>, <опции>) numeric::polysysroots(<уравнения>, <неизвестные>, <опции>) numeric::solve(<уравнения>, [<неизвестная> = <начало> .. <конец>, …], <опции>)
MatLab	fzero(<функция>, <начало>, <опции>) vpasolve(<уравнения>, <неизвестные>, [<начало> <конец>], <опции>) 2012

И при численном, и при аналитическом решении системы уравнений в MatLab 2016 их следует записывать в виде одномерного массива, элементами которого являются отдельные уравнения. В этом случае <неизвестные> также необходимо записать в виде одномерного массива. В предыдущих версиях уравнения и неизвестные записывались через запятую. Например, в MatLab версии раньше 2016 в результате выполнения команд `syms x y a; s1 = a*x + 6*y == 4, s2 = 4*x - 2*a*y == 2, [x, y] = solve(s1, s2, x, y)` в командном окне получится следующее аналитическое решение: `x = (2*(2*a + 3))/(a^2 + 12), y = -(a - 8)/(a^2 + 12)`. А в результате выполнения команд `syms x y; s1 = 2*x - y == 0, s2 = (x - 3)^2 - 6*sin(3*x/4) == 4, digits(4); [x, y] = vpasolve(s1, s2, x, y)` получится численное решение: `x = 0.5057`, `y = 1.011`.

В MatLab, как и в MuPAD, при необходимости можно указать интервал (интервалы), в котором необходимо искать решения уравнения (системы уравнений) или задать предполагаемое значение искомого корня. Например, в результате выполнения команд `syms x; digits(4); vpasolve(sin(x) == 1/2, x, [2,3])` получится `2.618`.

А чтобы и аналитически, и численно (с точностью до пяти значащих цифр) найти все корни уравнения $(x + 4)(x^2 - 5)(e^x - 45) = 0$, можно в m-файле написать программу:

```
clc          % очистка командного окна
clear all    % очистка памяти
syms x       % объявление символьной переменной
% исходное уравнение
ur = (x+4)*(x^2-5)*(exp(x)-45) == 0
X1 = solve(ur,x)     % аналитическое решение
% численное решение
digits(5)    % с точностью 5 значащих цифр
X2(1) = vpasolve(ur,x,[-5 -3]);
X2(2) = vpasolve(ur,x,[-3 -2]);
X2(3) = vpasolve(ur,x,[2 3]);
X2(4) = vpasolve(ur,x,[3 10]);
X2 % массив корней, полученных численно
```

В результате выполнения этой программы получатся два массива: массив-столбец `X1`, содержащий аналитические решения `-4, 5^(1/2), -5^(1/2), log(45)`; массив-строка `X2`, содержащий численные решения `-4.0, -2.2361, 2.2361, 3.8067`.

11.4.5. Решение задач математического анализа

В табл. 11.12 приведено сопоставление функций MuPAD и MatLab, предназначенных для решения задач математического анализа (вычисления сумм, произведений, пределов, интегрирования, дифференцирования, разложения в ряд, поиска особых точек функции и т.д.).

Функции численного решения задач математического анализа относятся к основному ядру MatLab (а не к пакету Symbolic Math Toolbox), поэтому доступны даже тогда, когда в лицензию MatLab не включены символьные вычисления.

Таблица 11.12

Функции для решения задач математического анализа в MuPAD и MatLab

Вычисление сумм и произведений символьных выражений	

MuPAD	`sum(<выражение>, <индекс> <значения>)`
	`product(<выражение>, <индекс> <значения>)`
	`numeric::sum(<выражение>, <индекс> <значения>)`
	`numeric::product(<выражение>, <индекс> <значения>)`

MatLab	`symsum(<выражение>, <индекс>, <min индекс>, <max индекс>)`
	`symprod(<выражение>, <индекс>, <min индекс>, <max индекс>)` 2011
	`sum(subs(<выражение>, <индекс>, <min индекс>:<max индекс>))`
	`prod(subs(<выражение>, <индекс>, <min индекс>:<max индекс>))`

Дифференцирование	

MuPAD	`diff(<выражение>, <список переменных>)`
	`D([<список номеров>], <функция>)`
	`<функция>'`

MatLab	`diff(<выражение>, <переменная>, <порядок>)`
Вычисление пределов	
MuPAD	`limit(<функция>, <аргумент> = <значение>, <опции>)`

MatLab	limit((<функция>, <аргумент>, <значение>, <опции>) Опции: 'left','right'
Поиск особых точек	
MuPAD	discont(<функция>, <аргумент> = <начало> .. <конец>, <опции>) poles(<функция>, <аргумент> = <начало> .. <конец>, <опции>)
MatLab	poles((<функция>, <аргумент>, <начало>, <конец>)
Разложение функции в ряд	
MuPAD	taylor(<функция>, <аргумент> = <значение>, <опции>) asympt(<функция>, <аргумент> = <значение>, <опции>) series(<функция>, <аргумент> = <значение>, <опции>)
MatLab	taylor(<функция>, <аргумент>, <значение>, <опции>) series(<функция>, <аргумент>, <значение>, <опции>) 2015 Значение опции 'Order' по умолчанию равно 6
Аналитическое вычисление интегралов	
MuPAD	int(<функция>, <аргумент> = <начало> .. <конец>, <опции>) intlib::byparts(<интеграл>, <часть u'>) intlib::changevar(<интеграл>, <замена>, <новый аргумент>)
MatLab	int(<функция>, <аргумент>, <начало>, <конец>, <опции>)
Численное интегрирование	
MuPAD	numeric::ncdata(<количество узлов>) numeric::gtdata(<количество узлов>) numeric::gldata(<количество узлов>, <точность>) numeric::quadrature(<функция>, <аргумент> = <начало> .. <конец>, <опции>)

Окончание табл. 11.12

	numeric::int(<функция>, <аргумент> = <начало> .. <конец>, <опции>)
MatLab	integral(<функция>, <начало>, <конец>, <опции>) 2012 методом Гаусса-Кронрода: quadgk(<функция>, <начало>, <конец>, <опции>) метод трапеций по точкам: trapz(<массив аргументов>, <массив значений>)

Перечисленные в табл. 11.12 функции MuPAD детально описаны в предыдущих главах пособия. Остановимся подробнее на некоторых функциях MatLab (в том числе из пакета Symbolic Math Toolbox) из приведенных в табл. 11.12.

Для функций MatLab, чтобы задать опцию, необходимо указывать ее название (в апострофах ' ') и (через запятую) значение этой опции. Только функция limit является исключением из этого правила. У этой функции необходимо указывать только название опций: 'left' или 'right'.

Функцию taylor можно вызвать с опциями: 'ExpansionPoint' (0 по умолчанию или выражение (числовое или символьное)); 'Order' (6 по умолчанию или целое положительное число); 'OrderMode' ('absolute' по умолчанию или 'relative'). Функцию series можно вызвать с опциями: 'ExpansionPoint' (0 по умолчанию или выражение (числовое или символьное)); 'Order' (6 по умолчанию или целое положительное число); 'Direction' ('complexPlane' по умолчанию или 'left', 'right', 'realAxis'). Функцию int можно вызвать с опциями: 'IgnoreAnalyticConstraints' (false по умолчанию или true); 'IgnoreSpecialCases' (false по умолчанию или true); 'PrincipalValue' (false по умолчанию или true). Функцию integral можно вызвать с опциями: 'AbsTol' (10^{-10} по умолчанию или неотрицательное вещественное число); 'RelTol' (10^{-6} по умолчанию или неотрицательное вещественное число); 'ArrayValued' (false по умолчанию или true, 0, 1); 'Waypoints' (массив точек, где функция имеет разрыв). Кроме функции integral, предназначенной для численного вычисления однократных интегралов, в MatLab существуют функции integral2 и integral3 для вычисления двукратных и трехкратных интегралов. Функцию quadgk можно вызвать с опциями: 'AbsTol' (10^{-10} по

умолчанию или неотрицательное вещественное число); `'RelTol'` (10^{-6} по умолчанию или неотрицательное вещественное число); `'MaxIntervalCount'` (`650` по умолчанию или максимальное количество разбиений интервала); `'Waypoints'` (массив точек, где функция имеет разрыв).

Например, чтобы для функции $1/(5-x)^3$ сначала вычислить неопределенный интеграл, а затем и аналитически, и численно рассчитать определенные интегралы на интервалах $[0, \pi/2]$, $[2\pi, \infty)$ и $[0, 2\pi]$, можно в m-файле написать программу:

```
clc            % очистка командного окна
clear all      % очистка памяти
syms x b eps   % объявление символьных переменных
f = @(x) 1./(5-x).^3 % объявление функции
I0 = int(f(x), x)    % неопределенный интеграл
% определенные интегралы
%       АНАЛИТИЧЕСКИЕ значения
I1 = int(f(x), x, 0, pi/2)    % на [0, pi/2]
vpa(I1)
I2 = int(f(x), x, 2*pi, Inf)  % на [2pi, ∞)
vpa(I2)
% на [0, 2pi] (2 способа):
% способ 1
I01 = subs(I0, x, 5-eps) - subs(I0, x, 0)
I02 = subs(I0, x, 2*pi) - subs(I0, x, 5+eps)
I3 = limit(I01 + I02, eps, 0)
% способ 2
I3 = int(f(x), x, 0, 2*pi, 'PrincipalValue', true)
vpa(I3)
%       ЧИСЛЕННЫЕ значения
I1num = integral(f, 0, pi/2)   % на [0, pi/2]
I2num = integral(f, 2*pi, Inf) % на [2pi, ∞)
% на [0, 2pi]
I3num = integral(f, 0, 4.99) + integral(f, 5.01, 2*pi)
```

Здесь при объявлении функции `f(x)` вместо `/` и `^` использованы операции поэлементного деления (`./`) и поэлементного возведения в степень (`.^`). Без этого в MatLab не удается корректно организовать численное вычисление интегралов. Для вывода посчитанных значений использован вывод по умолчанию.

В результате выполнения приведенной выше программы получатся следующие значения: неопределенный интеграл равен `1/(2*(x - 5)^2)`; определенный интеграл на интервале [0, π/2] равен `2/(pi - 10)^2 - 1/50` (аналитические вычисления) и `0.0225` (численные расчеты); несобственный интеграл на интервале [2π, ∞) равен `-1/(2*(2*pi - 5)^2)` и `-0.3037` (численные расчеты); несобственный расходящийся интеграл в смысле главного значения на интервале [0, 2π] равен `1/(2*(2*pi - 5)^2) - 1/50` (аналитические вычисления двумя способами) и `0.2837` (численные расчеты). Числовые значения аналитически посчитанных определенных интегралов (`vpa(I1)`, `vpa(I2)`, `vpa(I3)`) совпадают с их значениями, полученными численно (`I1num`, `I2num`, `I3num`).

11.4.6. Решение задач линейной алгебры

В табл. 11.13 приведено сопоставление функций MuPAD и MatLab, предназначенных для решения задач линейной алгебры (создание, преобразование, сложение, умножение, транспонирование матриц, вычисление определителя, обратной матрицы, скалярного и векторного произведения, собственных векторов и собственных значений и т.д.).

Таблица 11.13

Функции для решения задач линейной алгебры в MuPAD и MatLab

Создание матрицы (вектора)	

MuPAD	`A := matrix(<количество строк>, <количество столбцов>, <значения>, <опция>)`
	`A := matrix(<количество строк>, <количество столбцов>, (i, j) ->a.i.j)`
	`A := matrix(<количество строк>, <количество столбцов>, (i, j) ->a[i, j])`

MatLab	`A = [<символьные значения>]`
	`A = sym('a', [<количество строк> <количество столбцов>])`
	`A = sym(<числовая матрица>)`
	`A = sym('a%d%d', [<количество строк> <количество столбцов>])`

Создание единичной матрицы	
MuPAD	`matrix::identity(<количество строк>)` только квадратная
MatLab	`eye(<количество строк>, <количество столбцов>)`
Создание матрицы из нулей	
MuPAD	`matrix(<количество строк>, <количество столбцов>)`
MatLab	`zeros(<количество строк>, <количество столбцов>)`
Создание матрицы, состоящей из случайных чисел	

MuPAD	`linalg::randomMatrix(<количество строк>,`
	`<количество столбцов>, <тип>,`
	`<начало интервала>..<конец интервала>,<опция>)`

MatLab	`sym(<начало интервала> + (<конец интервала> -`
	`<начало интервала>) * rand(<количество строк>,`
	`<количество столбцов>))`
	`sym(randi([<начало интервала>,`
	`<конец интервала>], <количество строк>,`
	`<количество столбцов>))`
	`sym(<м. ожидание> + <ср.кв. отклонение> *`
	`randn(<количество строк>,<количество столбцов>))`

Умножение столбца (или строки) на число	

MuPAD	`linalg::multCol(<матрица>, <список столбцов>,`
	`<число>)`
	`linalg::multRow(<матрица>, <список строк>, <число>)`

MatLab	`<матрица>(:, <столбец>) =`
	`<матрица>(:,<столбец>)* <число>`
	`<матрица>(<строка>, :) =`
	`<матрица>(<строка>, :)* <число>`

Извлечение подматрицы	

MuPAD	`linalg::submatrix( <матрица>, <список строк>,`
	`<список столбцов>)`
	`<матрица>[<строка1> .. <строка2>,`
	`<столбец1> .. <столбец2>]`

MatLab	`<матрица>(<строка1>:<строка2>,` `<столбец1>:<столбец2>)`

Извлечение столбца (или строки)	
MuPAD	`linalg::col(<матрица>, <список столбцов>)` `linalg::row(<матрица>, <список строк>)`
MatLab	`<матрица>(:, <столбец1>:<столбец2>)` `<матрица>(<строка1>:<строка2>, :)`
Удаление столбца (или строки)	
MuPAD	`linalg::delCol(<матрица>, <список столбцов>)` `linalg::delRow(<матрица>, <список строк>)`
MatLab	`<матрица>(:, <столбец1>:<столбец2>) = []` `<матрица>(<строка1>:<строка2>, :)=[]`
Перестановка двух столбцов (строк)	
MuPAD	`linalg::swapCol(<матрица>, <столбец1>, <столбец2>, <список строк>)` `linalg::swapRow(<матрица>, <строка1>, <строка2>, <список столбцов>)`
MatLab	
Замена подматрицы другой матрицей	
MuPAD	`linalg::substitute(<матрица1>, <матрица2>, <позиция m>, <позиция n>)`
MatLab	
Линейная комбинация столбцов (строк)	
MuPAD	`linalg::addCol(<матрица>, <столбец1>, <столбец2>, <число1>, <число2>)` `linalg::addRow(<матрица>, <строка1>, <строка2>, <число1>, <число2>)`
MatLab	
Замена столбца (строки) вектором	
MuPAD	`linalg::setCol(<матрица>, <позиция>, <вектор-столбец>)` `linalg::setRow(<матрица>, <позиция>, <вектор-строка>)`
MatLab	`<матрица>(:, <столбец>) = <вектор-столбец>` `<матрица>(<строка>, :) = <вектор-строка>`

Продолжение табл. 11.13

Объединение матриц по вертикали	
MuPAD	`linalg::stackMatrix(<матрица1>, <матрица2>, …)`
MatLab	`cat(1, <матрица1>, <матрица2>, …)` или `vertcat(<матрица1>, <матрица2>, …)` или `[<матрица1>; <матрица2>; …]`
Объединение матриц по горизонтали	
MuPAD	`linalg::concatMatrix(<матрица1>, <матрица2>,…)` `<матрица1>.<матрица2>`
MatLab	`cat(2, <матрица1>, <матрица2>, …)` или `horzcat(<матрица1>, <матрица2>, …)` или `[<матрица1> <матрица2> …]`
Определение размеров матрицы (вектора)	

MuPAD	`linalg::matdim(<матрица>)`
	`linalg::nrows(<матрица>)`
	`linalg::ncols(<матрица>)`
	`linalg::vecdim(<вектор>)`

MatLab	`size (<матрица>)`
	`size (<матрица>, 1)`
	`size (<матрица>, 2)`
	`length (<вектор>)`

Количество ненулевых элементов в матрице	
MuPAD	`linalg::nonZeros(<матрица>)`
MatLab	`nnz(<матрица>)` 2014
Является ли матрица эрмитовой, унитарной или положительно определенной	

MuPAD	`linalg::isHermitian( <матрица>)`
	`linalg::isUnitary( <матрица>)`
	`linalg::isPosDef(<матрица>)`

MatLab	`ishermitian(<матрица>)`
Сложение, вычитание, умножение матриц	

MuPAD	`<матрица1> + <матрица2>`
	`<матрица1> – <матрица2>`
	`<число> * <матрица>`
	`<матрица1> * <матрица2>`

MatLab	`<матрица1> + <матрица2>`
	`<матрица1> – <матрица2>`
	`<число> * <матрица>`
	`<матрица1> * <матрица2>`

Продолжение табл. 11.13

След матрицы	
MuPAD	`linalg::tr(<матрица>)`
MatLab	`trace(<матрица>)`
Ранг матрицы	
MuPAD	`linalg::rank(<матрица>)` `numeric::rank(<матрица>, <точность>, <опция>)`
MatLab	`rank(<матрица>)`
Комплексное сопряжение, транспонирование и эрмитово сопряжение матрицы	

MuPAD	`conjugate(<матрица>)`
	`transpose(<матрица>)`
	`htranspose(<матрица>)`

MatLab	`conj(<матрица>)`
	`<матрица>.'` или `transpose(<матрица>)`
	`<матрица>'` или `ctranspose(<матрица>)`

Определитель матрицы	
MuPAD	`det(<матрица>, <опции>)` `numeric::det(<матрица>, <опции>)`
MatLab	`det(<матрица>)`
Союзная матрица	
MuPAD	`linalg::adjoint(<матрица>)`
MatLab	`adjoint(<матрица>)`
Обратная матрица	
MuPAD	`inverse(<матрица>, <опция>)` `numeric::inverse(<матрица>, <опции>)`
MatLab	`inv(<матрица>)`
Приведение матрицы к треугольному виду методом Гаусса	
MuPAD	`linalg::gaussElim(<матрица>, <опция>)`
MatLab	
Приведение матрицы к диагональному виду	
MuPAD	`linalg::gaussJordan(<матрица>, <опция>)`
MatLab	`rref(<матрица>)`
LU-разложение матрицы	
MuPAD	`linalg::factorLU(<матрица>)` `numeric::factorLU( <матрица>, <опции>)`
MatLab	`lu(<матрица>)`

QR-разложение матрицы	
MuPAD	`linalg::factorQR(<матрица>)` `numeric::factorQR(<матрица>, <опции>)`
MatLab	`qr(<матрица>)`
Приведение системы линейных уравнений к матричному виду	
MuPAD	`linalg::expr2Matrix(<система уравнений>, <неизвестные>, <опция>)`
MatLab	`equationsToMatrix(<система уравнений>, <неизвестные>)`
Решение системы линейных и нелинейных уравнений в матричном виде	
MuPAD	`linalg::matlinsolve(<матрица A>, <матрица B>, <опции>)` `numeric::matlinsolve(<матрица A>, <матрица B>, <опции>)`
MatLab	`linsolve(<матрица A>, <матрица B>)`
Составление характеристической матрицы и характеристического полинома	
MuPAD	`linalg::charmat(<матрица>, <переменная>)` `linalg::charpoly(<матрица>, <переменная>)`
MatLab	`charpoly(<матрица>, <переменная>)`
Нахождение собственных значений и собственных векторов матрицы	

MuPAD	`linalg::eigenvalues(<матрица>, <опция>)`
	`numeric::eigenvalues(<матрица>, <опции>)`
	`linalg::eigenvectors(<матрица>)`
	`numeric::eigenvectors(<матрица>, <опции>)`

MatLab	`eig(<матрица>)` `eig(vpa(<матрица>))`
Скалярное и векторное произведение векторов	
MuPAD	`linalg::scalarProduct(<вектор A>, <вектор B>, <опция>)` `linalg::crossProduct(<вектор A>, <вектор B>)`
MatLab	`dot(<вектор A>, <вектор B>)` `cross(<вектор A>, <вектор B>)`
Угол между векторами	
MuPAD	`linalg::angle(<вектор A>, <вектор B>)`
MatLab	`subspace(<вектор A>, <вектор B>)`

Окончание табл. 11.13

Ортогонализация векторов	
MuPAD	`linalg::orthog(<набор векторов>, <опция>)`
MatLab	`orth(<матрица векторов>, <опции>)`
Нахождение базиса векторов	
MuPAD	`linalg::basis(<набор векторов>)`
MatLab	
Градиент, дивергенция, ротор, лапласиан векторного (скалярного) поля	

MuPAD	`gradient(<скалярная функция>, <координаты>, <опция>)`
	`divergence(<векторная функция>, <координаты>, <опция>)`
	`curl(<векторная функция>, <координаты>, <опция>)`
	`laplacian(<скалярная функция>, <координаты>, <опция>)`

MatLab	`gradient(<скалярная функция>, [<координаты>])` 2011 `divergence(<векторная функция>, <координаты>])` 2012 `curl(<векторная функция>, [<координаты>])` 2012 `laplacian(<скалярная функция>, [<координаты>])` 2012

Функции численного решения задач линейной алгебры относятся к основному ядру MatLab (а не к пакету Symbolic Math Toolbox), поэтому доступны даже тогда, когда в лицензию MatLab не включены символьные вычисления.

Перечисленные в табл. 11.13 функции MuPAD детально описаны в предыдущих главах пособия. Остановимся подробнее на некоторых функциях MatLab (в том числе из пакета Symbolic Math Toolbox) из приведенных в табл. 11.13.

Из перечисленных в табл. 11.13 функций MatLab с опциями можно вызывать только функцию `orth`: `'real'` и (или) `'skipnormalization'`. Значения этих опций указывать не нужно. Остальные функции опций не имеют.

На рис. 11.6 приведен пример создания символьной матрицы размером 3×5, состоящей из случайных целых чисел из интервала [–4, 6] (используется преобразование числовой матрицы в символьную с помощью

функции `sym`), и приведения ее к диагональному виду методом Гаусса–Жордана с помощью функции `rref` в командном окне MatLab (в строках, отмеченных символом >>, записаны команды; переменные отображаются в окне Workspace, а их значения – в командном окне с помощью вывода по умолчанию). Те же результаты можно получить, если эти же команды записать в m-файле.

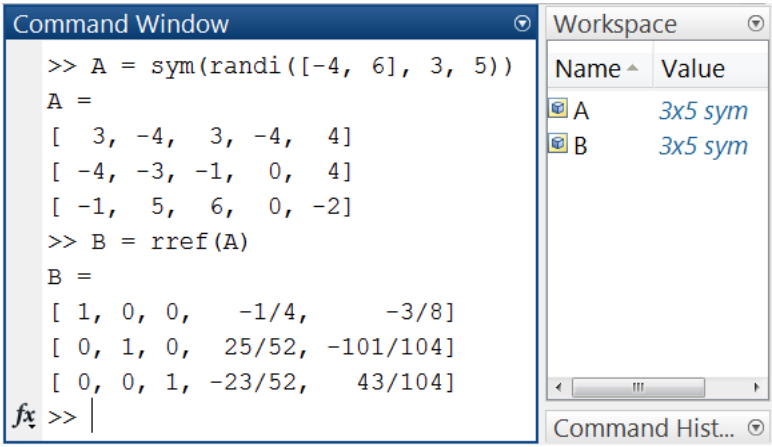


Рис. 11.6. Создание в MatLab символьной матрицы, состоящей из случайных целых чисел, и приведение ее к диагональному виду

Чтобы определить, при каких вещественных значениях переменной x скалярное произведение векторов

$$\vec{V}_1 = (x-2 \quad x+3 \quad 5) \text{ и } \vec{V}_2 = (1 \quad 3-x \quad 2-x)$$

больше нуля, можно, например, в m-файле написать программу:

```
clc          % очистка командного окна
clear all    % очистка памяти
syms x 'real'      % объявление символьной переменной
V1 = [x-2 x+3 5]   % первый вектор
V2 = [1 3-x 2-x]   % второй вектор
C = dot(V1, V2)    % скалярное произведение векторов
C = simplify(C)
xx = solve(C > 0, x) % решение неравенства
```

Для вывода значений переменных здесь использован вывод по умолчанию. В результате выполнения этой программы получится интервал

$[-\sqrt{21}-2,\ \sqrt{21}-2]$, который в командном окне будет отображаться в виде: `Dom::Interval(-21^(1/2) - 2, 21^(1/2) - 2)`.

11.4.7. Решение обыкновенных дифференциальных уравнений и их систем

В табл. 11.14 приведено сопоставление функций MuPAD и MatLab, предназначенных для решения и графического изображения решений обыкновенных дифференциальных уравнений и их систем.

Функции численного решения обыкновенных дифференциальных уравнений относятся к основному ядру MatLab (а не к пакету Symbolic Math Toolbox), поэтому доступны даже тогда, когда в лицензию MatLab не включены символьные вычисления.

Перечисленные в табл. 11.14 функции MuPAD детально описаны в предыдущих главах пособия. Остановимся подробнее на некоторых функциях MatLab (в том числе из пакета Symbolic Math Toolbox) из приведенных в табл. 11.14.

Таблица 11.14

Функции для решения и графического изображения решений обыкновенных дифференциальных уравнений и их систем в MuPAD и MatLab

Проверка дифференциального уравнения на однородность	
MuPAD	`ode::isLODE(<левая часть уравнения>, <искомая функция>, <опция>)`
MatLab	
Приведение дифференциального уравнения к нормальному виду	
MuPAD	`ode::normalize(<левая часть уравнения>, <искомая функция>, <аргумент функции>, <порядок>)`
MatLab	
Определение порядка дифференциального уравнения	
MuPAD	`ode::getOrder(<левая часть уравнения>, <искомая функция>)`
MatLab	

Аналитическое решение обыкновенного дифференциального уравнения или системы дифференциальных уравнений	

MuPAD	`ode({<уравнения>, <начальные условия>}, <искомая функция>)`
	`ode::solve(<ode-объект>, <опции>)`
	`ode::series(<уравнение>, <искомая функция>, <аргумент> = <значение>, <порядок>)`
	Подставить решение в исходное уравнение
	`ode::evalOde(<уравнение>, <искомая функция> = <решение>)`

MatLab	`dsolve(<уравнения>, <начальные условия>, <опции>)`
Численное решение обыкновенного дифференциального уравнения или системы обыкновенных дифференциальных уравнений	

MuPAD	`numeric::butcher(<метод>, <точность>)`
	
	Преобразование в систему уравнений первого порядка
	`<вектор-функция> :=`
	`numeric::ode2vectorfield({<уравнения>, <начальные условия>}, [<искомые функции>])`
	`<вектор-функция> :=`
	`numeric::odeToVectorField({<уравнение>, <начальные условия>}, [<искомая функция>])`
	Решение
	`numeric::odesolve2(<вектор-функция>, <начальная точка>, <начальный вектор>, <метод>, <опции>)`
	Решение в конечной точке (аргумента)
	`numeric::odesolve(<вектор-функция>, <начальная точка> .. <конечная точка>, <начальный вектор>, <метод>, <опции>)`

MatLab	Преобразование в систему уравнений первого порядка `odeToVectorField(<уравнения>, <начальные условия>)` 2012 `<ode-функция> = odeFunction([<уравнение1>; <уравнение2>;…], [<искомая функция1>; <искомая функция2>;…], <опции>)` 2015

Окончание табл. 11.14

	Решение ode23(<ode-функция>, [<начальная точка> <конечная точка>], <начальные условия>, options) – соответствует явному методу Рунге–Кутты порядка 2(3) ode45(<ode-функция>, [<начальная точка> <конечная точка>], <начальные условия>, options) – соответствует методу DOPRI45 порядка 4(5) (см. разд. 8.3.1) ode113(<ode-функция>, [<начальная точка> <конечная точка>], <начальные условия>, options) – многошаговый метод переменного порядка (от 1 до 12) Задать опции для ode23, ode45 или ode113 options = odeset(<опции>) Подставить в решение значение аргумента deval(<решение>, <конечная точка>)
Графическое изображение численного решения обыкновенного дифференциального уравнения (или системы уравнений)	
MuPAD	plot::Ode2d(<вектор-функция>, [<список точек>], <начальный вектор>, [<кривая1>], [<кривая2>], …, <метод>, <опции решения>, <графические опции>)
MatLab	Задать опцию для ode23, ode45 или ode113, чтобы решение выводилось в виде графика options = odeset('OutputFcn', @odeplot)

В MatLab, чтобы задать <опцию>, необходимо указывать ее название (в апострофах ' ') и (через запятую) значение этой опции.

Функцию dsolve можно вызвать с опциями: 'MaxDegree' (2 по умолчанию или целое положительное число меньше 5); 'IgnoreAnalyticConstraints' (true по умолчанию или false). Функцию odeFunction можно вызвать с опциями: 'File' ('' по умолчанию или название m-файла в строковом виде); 'Optimize' (true по умолчанию или false); 'Sparse' (false по умолчанию или true). Функция odeFunction создает <ode-функцию> для вызова одной из функций ode23, ode45 или ode113. Функции ode23, ode45, ode113 можно вызвать с опциями,

которые удобно задать с помощью функции `odeset`: `'RelTol'` (10^{-3} по умолчанию или положительное вещественное число); `'AbsTol'` (10^{-6} по умолчанию или положительное вещественное число); `'NormControl'` (`'off'` по умолчанию или `'on'`); `'NonNegative'` (`[]` по умолчанию или целое число, массив целых чисел); `'OutputFcn'` (`@odeplot` по умолчанию или `@odephas2`, `@odephas3`, `@odeprint`, нестандартная @-функция); `'OutputSel'` (вектор); `'Refine'` (1 для всех кроме `ode45` (для `ode45` 4) по умолчанию или целое положительное число); `'Stats'` (`'off'` по умолчанию или `'on'`); `'InitialStep'` (положительное вещественное число); `'MaxStep'` (положительное вещественное число); `'Events'` (@-функция)

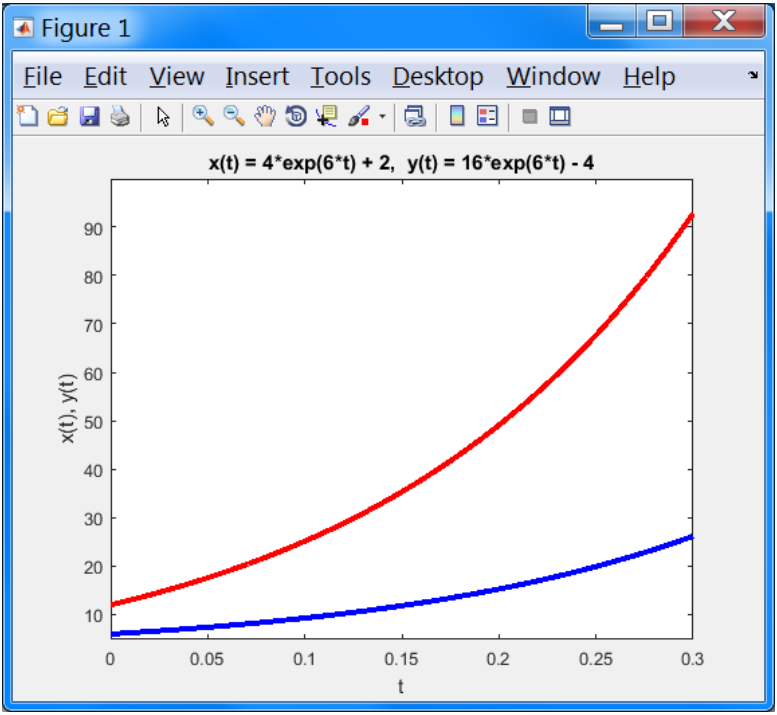


Рис. 11.7. График аналитического решения системы обыкновенных дифференциальных уравнений

'Jacobian' (матрица); 'JPattern' (матрица); 'Vectorized' ('off' по умолчанию или 'on'); 'Mass' (матрица); 'MStateDependence' ('weak' по умолчанию или 'none', 'strong'); 'MvPattern' (матрица); 'MassSingular' ('maybe' по умолчанию или 'yes', 'no'); 'MaxOrder' (5 по умолчанию или целое положительное число меньше 6). Если список текущих опций для вызова функций ode23, ode45, ode113 присвоен какой-либо переменной, то с помощью функции odeget можно получить значения некоторых из них. Для этого в качестве первого параметра функции odeget нужно указать переменную, которой присвоен список опций, а в качестве последующих параметров – названия опций, значения которых нужно вывести. Если в качестве опции 'OutputFcn' для вызова функций ode23, ode45, ode113 используется @odeplot (по умолчанию), то решение выводится в виде графика, на котором последовательно отмечается каждая вычисленная точка. Следует помнить, что с помощью опций функций ode23, ode45, ode113 нельзя изменять настройки (цвет, стиль и т.д.) графиков, заданные по умолчанию.

При символьном решении дифференциальных уравнений следует помнить, что искомые функции необходимо заранее объявить как символьные вместе и их аргументом. Например: syms y(x).

Например, чтобы аналитически решить систему дифференциальных уравнений $x'(t) = 2x(t) + y(t)$, $y'(t) = 8x(t) + 4y(t)$ при начальных условиях $x(0) = 6$, $y(0) = 12$, можно написать следующую последовательность команд (в m-файле, например):

```
syms x(t) y(t)
% уравнения
ur1 = diff(x, t) == 2*x + y
ur2 = diff(y, t) == 8*x + 4*y
% аналитическое решение
[xt, yt]=dsolve(ur1, ur2, x(0) == 6, y(0) == 12)
% график решения
fx = ezplot(xt, [0, 0.3]);
set(fx, 'Color', 'blue', 'LineWidth', 3);
hold on
fy = ezplot(yt, [0, 0.3]);
set(fy, 'Color', 'red', 'LineWidth', 3);
hold off
```

```
title(['x(t) = ' char(xt) ',  y(t) = ' char(yt)])
ylabel('x(t), y(t)')
```

В результате переменным `xt` и `yt` будут присвоены следующие аналитические выражения: `xt = 4*exp(6*t) + 2` и `yt = 16*exp(6*t) – 4`, а в графическом окне появится график, изображенный на рис. 11.7. В MatLab 2016 и выше вместо функции `ezplot` следует использовать функцию `fplot` (см. разд. 11.4.2).

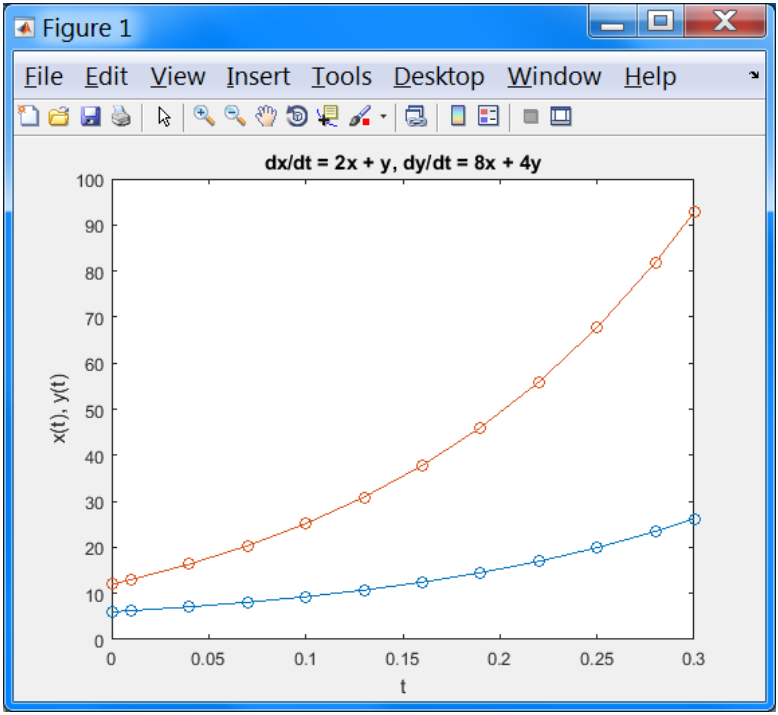


Рис. 11.8. График численного решения системы обыкновенных дифференциальных уравнений

Для численного решения дифференциальных уравнений (так же, как в MuPAD) их нужно записать специальным образом – в виде векторной функции. Например, для численного решения той же системы дифференциальных уравнений при тех же начальных условиях ($x'(t) = 2x(t) + y(t)$,

$y'(t) = 8x(t) + 4y(t)$, $x(0) = 6$, $y(0) = 12$) можно написать следующую последовательность команд:

```
% векторная функция двух переменных, содержащая
% правые части каждого из дифференциальных уравнений
ode_func = @(t, Y) [2*Y(1) + Y(2); 8*Y(1) + 4*Y(2)]
% Далее при вызове функции ode23 указаны:
% векторная функция ode_func;
% интервал [0, 0.3] по времени;
% начальные условия в векторном виде [6; 12]
ode23(ode_func, [0 0.3], [6; 12])
title('dx/dt = 2x + y, dy/dt = 8x + 4y')
ylabel('x(t), y(t)'); xlabel('t');
xlim([0 0.3]); ylim([0 100])
```

В результате выполнения этого программного кода появится графическое окно, в котором будут изображены графики двух функций (рис. 11.8). Точки, в которых выполнялись вычисления, по умолчанию отмечаются окружностями. Для представления дифференциальных уравнений в векторном виде можно предварительно воспользоваться функцией `odeToVectorField` (не использована в этом примере). При составлении векторной ode-функции следует помнить, что для последующего численного решения она не должна содержать символьных переменных. До 2015 г. векторную ode-функцию можно было запрограммировать только вручную (как в приведенном выше примере). Начиная с MatLab 2015, появилась возможность автоматизировать процесс составления такой функции благодаря появлению функции `odeFunction`.

Влияние опций описанных выше функций на результаты аналитического и численного решения обыкновенных дифференциальных уравнений в MatLab можно изучить самостоятельно.

11.4.8. Интерполяция и полиномиальное представление аналитических функций

В табл. 11.15 приведено сопоставление функций MuPAD и MatLab, предназначенных для построения интерполяции и представления функции полиномами в форме Бернштейна.

Функции численного построения интерполяции относятся к основному ядру MatLab (а не к пакету Symbolic Math Toolbox), поэтому доступны

даже тогда, когда в лицензию MatLab не включены символьные вычисления.

Таблица 11.15

Функции для построения интерполяции и полиномиального представления функции в MuPAD и MatLab

Интерполяция Лагранжа	
MuPAD	`interpolate(<список узлов>, <список значений>, <аргумент>)`
MatLab	
Интерполяция кубическими сплайнами	
MuPAD	`numeric::cubicSpline([<список узлов>, <список значений>], <опции>)` `numeric::cubicSpline(<точка1>, <точка2>, …, <опции>)`

MatLab	`interp1(<x-координаты узлов>, <y-координаты узлов>, <x-координаты между узлами>, <метод>, <опция>)`
	Методы: `'nearest'`, `'next'`, `'previous'`, `'linear'`, `'spline'`, `'pchip'`, `'cubic'`
	
	Интерполяция кубическими сплайнами
	`spline(<x-координаты узлов>, <y-координаты узлов>, <x-координаты между узлами>)`
	`interp1(<x-координаты узлов>, <y-координаты узлов>, <x-координаты между узлами>, 'spline')`
	
	Интерполяция кубическими полиномами Эрмита
	`pchip(<x-координаты узлов>, <y-координаты узлов>, <x-координаты между узлами>)`
	`interp1(<x-координаты узлов>, <y-координаты узлов>, <x-координаты между узлами>, 'pchip')`

Полиномиальное представление в форме Бернштейна	
MuPAD	`bernstein(<функция>, <аргумент>, <степень>, <переменная>)` 2015
MatLab	`bernstein(<функция>, <аргумент>, <степень>, <переменная>)` 2013

Перечисленные в табл. 11.15 функции MuPAD детально описаны в предыдущих главах пособия. Остановимся подробнее на некоторых функциях MatLab (в том числе из пакета Symbolic Math Toolbox) из приведенных в табл. 11.15.

Функции `bernstein` в MuPAD и в Symbolic Math Toolbox вызываются с одинаковыми параметрами и по одинаковым правилам, которые описаны в предыдущих главах пособия.

Для функций `interp1`, `spline`, `pchip` (основное ядро MatLab) параметры `<x-координаты узлов>`, `<y-координаты узлов>`, `<x-координаты между узлами>` следует задавать в виде одномерных массивов-строк или массивов-столбцов (или их идентификаторов), содержащих соответствующие координаты узлов и точек между ними, в которых необходимо вычислить значения интерполяционной функции. Массивы `<x-координаты узлов>`, `<y-координаты узлов>` должны содержать одинаковое количество элементов. Считается, что *x*- и *y*-координаты одного и того же узла в этих массивах имеют одинаковые индексы (номера).

При задании `<опций>` и `<методов>` функции `interp1` необходимо указывать только их значение.

В качестве значений `<методов>` функции `interp1` можно использовать следующие строковые константы, которые соответствуют разным типам интерполяционных полиномов: `'linear'` – линейные (по умолчанию); `'nearest'`; `'next'`; `'previous'`; `'spline'` – кубические сплайны; `'pchip'` – кубические полиномы Эрмита; `'cubic'`.

Функцию `interp1` можно вызвать с `<опцией>`, для которой допустимы только три значения: `'pp'` (создает кусочно-непрерывный полином, значения которого в конкретной точке могут быть посчитаны с помощью функции `ppval`); `'extrap'` или числовое значение (для построения экстраполяции). При использовании `<опции>` со значением `'pp'` указывать *x*-координаты между узлами не нужно. Аналогичные кусочно-непрерывные полиномы можно получить с помощью функций `spline` и `pchip`, если при их вызове опустить (не указывать) *x*-координаты между узлами. Если *x*-координаты между узлами указаны, каждая из функций `interp1`, `spline` или `pchip` возвращает массив значений интерполяционной функции в этих точках.

Например, имея массив узлов `xU` и массив значений искомой функции в этих узлах `yU`, можно составить интерполяционную функцию, используя

метод кубических сплайнов, следующим образом (*x*-координаты *между* узлами не заданы):

```
% узлы
xU = [2 2.5 3 3.5 4.1 4.6 5]
% значения в узлах
yU = [-5.8 -6.3 -1.55 3.1 3.6 0.74 -1.6]
% интерполяционный полином
pol = interp1(xU, yU, 'spline', 'pp')
% или pol = spline(xU, yU)
% интерполяционная символьная функция
syms x
F = @(x) ppval(pol, x)
% график символьной функции на интервале [2,5]
gr = ezplot(F, [xU(1), xU(7)]);
set(gr, 'LineWidth', 3);
% изображение узлов на том же графике
hold on; uz = plot(xU, yU, 'o');
set(uz, 'LineWidth', 2); hold off
```

Функцию `F(x)` можно объявить и как символьную, и как несимвольную (без команды `syms x`) и далее использовать для вычисления значения в произвольной точке между узлами. Но для построения графика несимвольной функции `F(x)` вместо функции `ezplot` следует использовать функцию `plot`.

Кроме функции `interp1` в MatLab существует также функция `interp2`, которая позволяет построить интерполяцию по узлам, расположенным не на прямой, а в плоскости (интерполяция функции, зависящей от двух аргументов). С особенностями использования этой функции можно ознакомиться в справочной системе MatLab.

11.4.9. Аппроксимация

Задачи поиска аналитического вида функции по известным числовым значениям $y_1, y_2, y_3, \ldots, y_n$ этой функции в некоторых точка (узлах) $x_1, x_2, x_3, \ldots, x_n$ решает не только интерполяция, но и аппроксимация. В отличие от интерполяции, где значения $y_1, y_2, y_3, \ldots, y_n$ искомой функции в узлах считаются полностью достоверными, при построении аналитической функции методами аппроксимации предполагается, что известные значения $y_1, y_2, y_3, \ldots, y_n$ искомой функции в узлах могут содержать некоторую

погрешность. Поэтому искомая функция может проходить не через узлы, а вблизи них. Полученная в результате применения методов аппроксимации искомая аналитическая функция называется аппроксимирующей. Одним из широко распространенных методов аппроксимации является метод наименьших квадратов, в котором ищут такую аналитическую функцию, для которой средний квадрат расстояний между искомой функцией и известными значениями в узлах минимален. Обычно искомую аппроксимирующую функцию записывают в виде линейной комбинации нескольких элементарных (или специальных) функций с неизвестными числовыми коэффициентами. В этом случае задача нахождения аналитической функции сводится к вычислению таких числовых коэффициентов, при которых средний квадрат расстояний между искомой функцией и известными значениями в узлах минимален. Поэтому методы аппроксимации относят не к аналитическим, а к численным методам. В MuPAD отсутствуют функции, реализующие эти методы.

В MatLab аппроксимирующую функцию можно построить с помощью функции `fit`, которую вызывают следующим образом:

```
<Имя функции> = fit(<x-координаты узлов>,
          <y-координаты узлов>, <тип функции>, <опции>)
[<Имя функции>, <достоверность>] =
        fit(<x-координаты узлов>, <y-координаты узлов>,
                                  <тип функции>, <опции>)
```

Функция `fit` входит в пакет Curve Fitting Toolbox, который доступен в MatLab без предварительного подключения.

При вызове функции `fit` обязательными являются параметры `<x-координаты узлов>`, `<y-координаты узлов>`, `<тип функции>`. Параметры `<x-координаты узлов>`, `<y-координаты узлов>` можно задавать только в виде одномерных массивов-столбцов (или их идентификаторов), содержащих соответствующие координаты узлов. Массивы `<x-координаты узлов>`, `<y-координаты узлов>` должны содержать одинаковое количество элементов. Считается, что *x*- и *y*-координаты одного и того же узла в этих массивах имеют одинаковые индексы (номера).

С помощью параметра `<тип функции>` необходимо выбрать, из комбинации каких элементарных (или специальных) функций будет строиться искомая аппроксимирующая функция. Параметр `<тип функции>` можно задать с помощью следующих стандартных идентификаторов:

• для построения полиномиальных зависимостей: `'poly1'` – искомая функция будет строиться в виде $p_1x + p_2$, `'poly2'` – в виде $p_1x^2 + p_2x + p_3$, `'poly3'` – в виде $p_1x^3 + p_2x^2 + ... + p_4$ и так до `'poly9'` включительно, для которого искомая функция строится в виде $p_1x^9 + p_2x^8 + ... + p_{10}$;

• для построения степенных зависимостей: `'power1'` – искомая функция будет строиться в виде ax^b, `'power2'` – в виде $ax^b + c$;

• для построения экспоненциальных зависимостей: `'exp1'` – искомая функция будет строиться в виде ae^{bx}, `'exp2'` – в виде $ae^{bx} + ce^{dx}$;

• для построения рациональных зависимостей: `'rat02'` – искомая функция будет строиться в виде $p_1/(x^2 + q_1x + q_2)$, `'rat21'` – в виде $(p_1x^2 + p_2x + p_3)/(x + q_1)$, `'rat55'` – в виде $(p_1x^5 + ... + p_6)/(x^5 + ... + q_5)$;

• для построения тригонометрических зависимостей: `'sin1'` – искомая функция будет строиться в виде $a_1\sin(b_1x + c_1)$, `'sin2'` – в виде $a_1\sin(b_1x + c_1) + a_2\sin(b_2x + c_2)$, `'sin3'` – в виде $a_1\sin(b_1x + c_1) + ... + a_3\sin(b_3x + c_3)$ и так до `'sin8'` включительно, для которого искомая функция строится в виде $a_1\sin(b_1x + c_1) + ... + a_8\sin(b_8x + c_8)$;

• для построения зависимостей в виде конечного ряда Фурье: `'fourier1'` – искомая функция будет строиться в виде $a_0 + a_1\cos(\omega x) + b_1\sin(\omega x)$, `'fourier2'` – в виде $a_0 + a_1\cos(\omega x) + b_1\sin(\omega x) + a_2\cos(2\omega x) + b_2\sin(2\omega x)$, `'fourier3'` – в виде $a_0 + a_1\cos(\omega x) + b_1\sin(\omega x) + ... + a_3\cos(3\omega x) + b_3\sin(3\omega x)$ и так до `'fourier8'` включительно, для которого искомая функция строится в виде $a_0 + a_1\cos(\omega x) + b_1\sin(\omega x) + ... + a_8\cos(8\omega x) + b_8\sin(8\omega x)$, где $\omega = 2\pi/(x_{max} - x_{min})$;

• для построения зависимостей в виде линейной комбинации функции Гаусса: `'gauss1'` – искомая функция будет строиться в виде $a_1\exp(-((x - b_1)/c_1)^2)$, `'gauss2'` – в виде $a_1\exp(-((x - b_1)/c_1)^2) + a_2\exp(-((x - b_2)/c_2)^2)$, `'gauss3'` – в виде $a_1\exp(-((x - b_1)/c_1)^2) + ... + a_3\exp(-((x - b_3)/c_3)^2)$ и так далее до `'gauss8'` включительно, для которого искомая функция строится в виде $a_1\exp(-((x - b_1)/c_1)^2) + ... + a_8\exp(-((x - b_8)/c_8)^2)$;

• для построения интерполяционных зависимостей: `'cubicspline'`, `'smoothingspline'`, `'linearinterp'`, `'nearestinterp'`, `'cubicinterp'`, `'pchipinterp'`.

Кроме того, параметр <тип функции> можно записать в виде строкового представления (в апострофах ' ') искомой аналитической функции, накпример в виде 'a*(x-b)^n'. Все используемые в таком выражении переменные, кроме x, будут считаться коэффициентами, подлежащими определению.

При вызове функции fit можно использовать большое количество <опций>. Как и для других функций MatLab, чтобы задать <опцию> функции fit, необходимо указать название этой опции (в апострофах ' ') и (через запятую) ее значение. Далее кратко перечислены <опции> функции fit, их возможные значения и значения по умолчанию: 'Normalize' ('off' по умолчанию или 'on'); 'Exclude' (массив или выражение); 'Weights' ([] по умолчанию или массив); 'problem' (массив); 'SmoothingParam' (число в интервале от 0 до 1); 'Span' (0,25 по умолчанию или число в интервале от 0 до 1); 'Robust' ('off' по умолчанию, 'LAR', 'Bisquare'); 'Lower' ([]по умолчанию или массив); 'Upper' ([] по умолчанию или массив); 'StartPoint' ([] по умолчанию или массив); 'Algorithm' ('Levenberg-Marquardt' по умолчанию или 'Trust-Region'); 'DiffMaxChange' (0,1 по умолчанию); 'DiffMinChange' (10^{-8} по умолчанию); 'Display' ('notify' по умолчанию, 'final', 'iter', 'off'); 'MaxFunEvals' (600 по умолчанию); 'MaxIter' (400 по умолчанию); 'TolFun' (10^{-6} по умолчанию); 'TolX' (10^{-6} по умолчанию). Подробнее с особенностями использования <опций> функции fit можно ознакомиться в справочной системе MatLab.

Функция fit всегда возвращает искомую аппроксимирующую функцию в виде объекта, состоящего из аналитического выражения, записанного в общем виде через неизвестные коэффициенты, и значений этих коэффициентов. Созданный функцией fit объект присваивается переменной <имя функции>, указанной при вызове функции fit в качестве результирующего параметра перед оператором присвоения: <имя функции> = fit(...). С помощью идентификатора этой переменной далее в программе осуществляется обращение к данному объекту. Например, если указать объект <имя функции> в качестве параметра функции plot, можно построить график полученной аппроксимирующей функции на том же интервале, на котором расположены узлы: plot(<имя функции>). Следует отметить, что объект <имя функции>, созданный функцией

fit, можно использовать ТОЛЬКО в качестве ПЕРВОГО параметра функции plot. Чтобы вычислить значение аппроксимирующей функции, полученной с помощью функции fit, при определенном значении ее аргумента, следует после идентификатора объекта <имя функции> в круглых скобках указать значение аргумента: <имя функции>(<значение аргумента>).

Кроме объекта, описывающего искомую функцию, функция fit может возвращать дополнительную информацию, характеризующую достоверность полученных результатов. Чтобы получить эту информацию, для присвоения возвращаемого функцией fit результата следует использовать не одну, а две переменные, которые записывают веред оператором присвоения (=) в квадратных скобках через запятую: [<имя функции>, <достоверность>] = fit(…). Тогда первой из указанных переменных будет присвоен созданный функцией fit объект, а второй – структура, содержащая 5 числовых характеристик оценки достоверности построенной аппроксимации: sse – сумма квадратов отклонений искомой функции от значений в узлах; rsquare – коэффициент определенности; dfe – степень свободы; adjrsquare – степень свободы с учетом коэффициента определенности; rmse – стандартное среднеквадратичное отклонение искомой функции от значений в узлах.

Например, чтобы, используя данные, расположенные в одномерных массивах-столбцах xU и yU, построить линейную функцию методом наименьших квадратов и изобразить ее на графике одновременно с узлами, можно в m-файле написать программу:

```
clc          % очистка командного окна
clear all    % очистка памяти
% исходные данные:
% узлы
xU = [10; 12; 14; 15; 16; 18; 19; 22];
% значения в узлах
yU = [20; 23; 30; 31; 34; 34; 38; 43];
% аппроксимация полиномом первой степени
f = fit(xU, yU, 'poly1')
% график аппроксимирующей функции и узлов
plot(f,'-r', xU, yU, 'ob')
% вычисление значений аппроксимирующей функции
% в дополнительных точках с координатами x1
x1 = [13; 20; 21];
```

```
y1(1) = f(x1(1)); y1(2) = f(x1(2));
y1(3) = f(x1(3));                    % или y1 = f(x1);
% изображение дополнительных точек на том же графике
hold on; plot(x1,y1,'sk'); hold off
% настройка дополнительных параметров графика:
% сетки, легенды и заголовка
grid on
legend('узлы','аппроксимация', ...
       'Location','NorthWest')
title(['Линейная аппроксимация '...
       'по методу наименьших квадратов'])
```

Необходимо отметить, что последовательность символов `...` используется в MatLab для переноса части текста команды на другую строку.

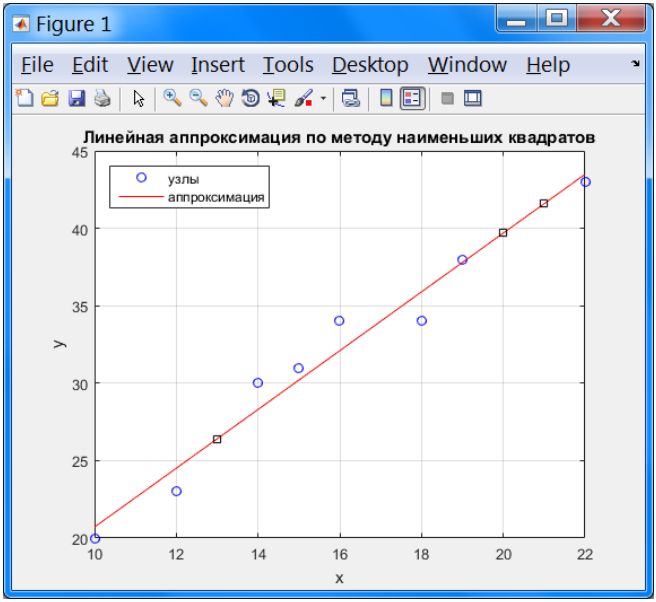


Рис. 11.9. Пример графического представления результатов аппроксимации

В результате выполнения этой программы в командном окне созданный функцией `fit` объект `f` будет отображаться следующим образом:

```
f =
     Linear model Poly1:
     f(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =       1.898  (1.537, 2.259)
       p2 =        1.73  (-4.105, 7.565)
```

Кроме того, программой будет создано графическое окно, изображенное на рис. 11.9. На этом графике синими окружностями изображены узлы, красной линией – аппроксимирующая функция, а черными квадратами – значения аппроксимирующей функции в дополнительных точках.

Из созданного функцией `fit` объекта (обозначенного переменной `<имя функции>`) можно извлечь аналитическое выражение, описывающее искомую функцию, ее аргумент, а также идентификаторы и значения входящих в функцию числовых коэффициентов. Для этого можно воспользоваться функциями `argnames`, `formula`, `coeffnames`, `coeffvalues` следующим образом:

```
<Результат> = argnames(<имя функции>)
<Результат> = formula(<имя функции>)
<Результат> = coeffnames(<имя функции>)
<Результат> = coeffvalues(<имя функции>)
```

Функция `argnames` возвращает строковый массив-столбец, элементами которого являются идентификаторы аргумента аппроксимирующей функции и числовых коэффициентов. Функция `formula` возвращает аналитическое выражение в строковом виде. Функция `coeffnames` возвращает строковый массив-столбец, элементами которого являются только идентификаторы числовых коэффициентов. А функция `coeffvalues` возвращает числовой массив-строку, состоящий из значений числовых коэффициентов. Полученное с помощью функции `formula` аналитическое выражение аппроксимирующей функции можно преобразовать в символьное выражение с помощью функции `sym`. Далее с этим выражением можно выполнять все доступные для символьных выражений операции. Например, можно подставить в него значения коэффициентов, продифференцировать, проинтегрировать и т.д.

Таким образом, несмотря на то, что функция `fit` выполняет не аналитические, а численные расчеты, она позволяет получить результат в виде символьной функции, зависящей от одного аргумента.

Например, чтобы, используя данные, расположенные в одномерных массивах-столбцах `xU` и `yU`, построить методом наименьших квадратов

линейную *символьную* функцию и изобразить ее на графике одновременно с узлами, можно в m-файле написать программу:

```
clc          % очистка командного окна
clear all    % очистка памяти
% исходные данные:
% узлы
xU = [10; 12; 14; 15; 16; 18; 19; 22];
% значения в узлах
yU = [20; 23; 30; 31; 34; 34; 38; 43];
% аппроксимация полиномом первой степени
f = fit(xU, yU, 'poly1')
% выделение из объекта f
Vstr = formula(f);      % формулы полинома
Zstr = coeffnames(f);   % названий коэффициентов
P = coeffvalues(f);     % значений коэффициентов
syms x        % объявление символьной переменной
% преобразование формулы в символьное выражение
Vsym = sym(Vstr);
% подстановка в выражение первого коэффициента
Vsym = subs(Vsym, sym(Zstr(1,1)), P(1));
% подстановка в выражение второго коэффициента
Vsym = subs(Vsym, sym(Zstr(2,1)), P(2));
% создание символьной функции
F(x) = Vsym
% график символьной функции на интервале [10,22]
ezplot(F(x), [xU(1), xU(8)])
% изображение узлов на том же графике
hold on; plot(xU, yU, 'or'); hold off
% настройка дополнительных параметров графика:
% сетки, заголовка, названий осей
grid on; title(Vstr); xlabel('x'); ylabel('y')
```

В MatLab 2016 и последующих версиях вместо функции `ezplot` следует использовать функцию `fplot`. В результате выполнения приведенной выше программы в командном окне искомая функция будет представлена двумя способами – как объект `f`, созданный функцией `fit`, и как символьная функция `F(x)` с подставленными в нее значениями (в рациональном виде) коэффициентов:

```
f =
     Linear model Poly1:
```

```
      f(x) = p1*x + p2
      Coefficients (with 95% confidence bounds):
        p1 =       1.898  (1.537, 2.259)
        p2 =        1.73  (-4.105, 7.565)
F(x) =
(801*x)/422 + 365/211
```

Кроме того, в отдельном окне будет построен график символьной функции с нанесенными на него узлами.

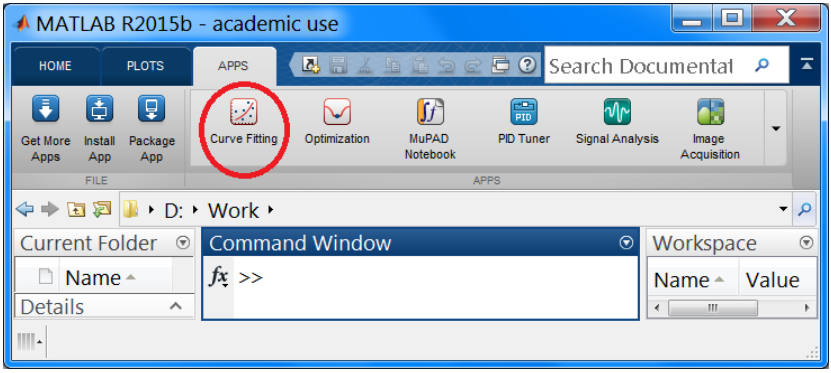


Рис. 11.10. Вызов приложения Curve Fitting

Как правило, заранее неизвестно, аналитическая функция какого типа наилучшим образом будет описывать имеющийся набор данных (узлы и значения в них). Поэтому в процессе поиска аппроксимирующей функции для одного и того же набора данных многократно вызывают функцию `fit`, используя разные значения параметра `<тип функции>` и `<опций>`, сравнивают полученные результаты. Для повышения эффективности такого поиска в MatLab существует специализированное приложение Curve Fitting, которое позволяет не только в диалоговом режиме изменять параметры вызова функции `fit`, но и автоматически совмещать на графике исходные данные с полученной аппроксимирующей функцией.

Прежде чем запустить приложение Curve Fitting, необходимо создать два массива, содержащих координаты узлов и значений искомой функции в них. Эти данные можно загрузить из файла, либо присвоить двум разным переменным, перечислив соответствующие значения через точку с запятой в квадратных скобках после оператора присвоения в командном окне или

m-файле. Если массивы заданы в m-файле, перед запуском приложения Curve Fitting этот m-файл необходимо выполнить так, чтобы в окне Workspace присутствовали идентификаторы данных массивов.

Чтобы запустить приложение Curve Fitting, необходимо на панели инструментов основного окна MatLab перейти на вкладку `Apps` (см. рис. 1.3), на которой расположена раскрывающаяся панель кнопок. На этой панели следует выбрать кнопку `Curve Fitting` (рис. 11.10). В результате появится окно приложения Curve Fitting (рис. 11.11).

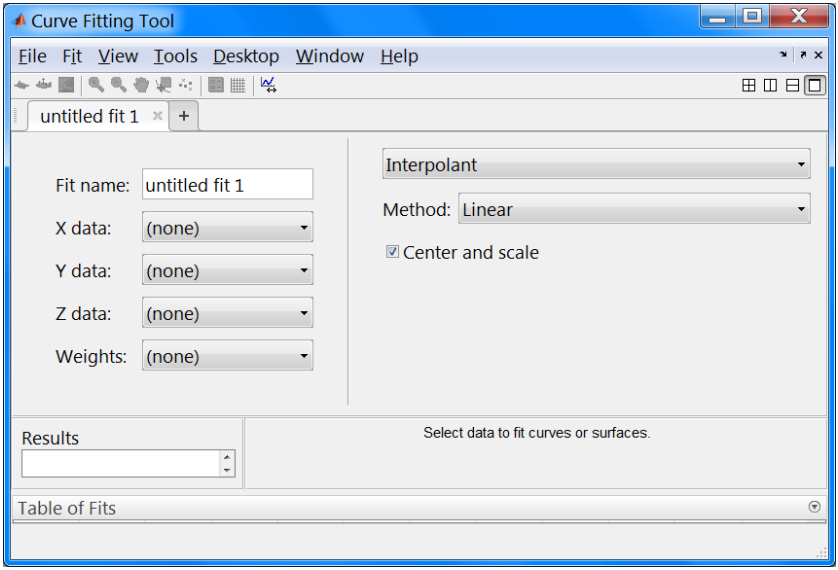


Рис. 11.11. Окно приложения Curve Fitting

После этого необходимо загрузить созданные ранее массивы исходных данных в приложение. В раскрывающихся списках `X data`, `Y data` и т.д. автоматически отображаются все одномерные числовые массивы, загруженные в данный момент в память MatLab. Из списка `X data` необходимо выбрать массив, содержащий узлы, а из списка `Y data` – массив, содержащий значения искомой функции в узлах.

В средней части окна расположено несколько списков, позволяющих выбрать тип искомой аппроксимирующей функции. По умолчанию в первом из этих списков всегда выбрано Interpolant (интерполяционная). После

загрузки в поля `X data` и `Y data` конкретных числовых массивов (например, `xU` и `yU` соответственно) тип аппроксимирующей функции автоматически изменяется на Polynomial (полиномиальную). В поле `Degree` можно выбрать степень полинома (1 по умолчанию). При любом изменении значений полей в нижней части окна автоматически перестраивается

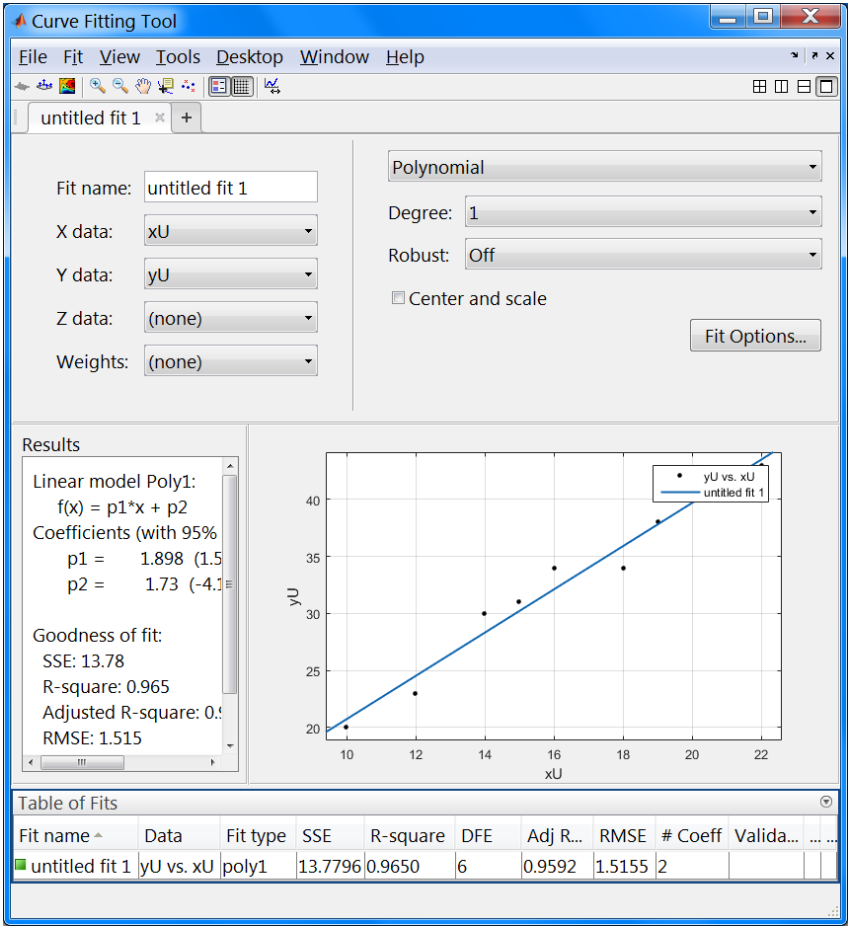


Рис. 11.12. Аппроксимация полиномом первой степени в приложении Curve Fitting

график, а слева от него выводится информация об объекте, создаваемом функцией `fit`: аналитический вид аппроксимирующей функции, значения входящих в нее коэффициентов, а также sse, r-square, adjrsquare, rmse – вспомогательная информация о достоверности полученных результатов. На графике изображается аппроксимирующая кривая и узлы.

Например, если (как и в предыдущем примере) в качестве исходных данных использовать массивы `xU = [10; 12; 14; 15; 16; 18; 19; 22]` и `yU = [20; 23; 30; 31; 34; 34; 38; 43]`, то результат аппроксимации полиномом первой степени в приложении Curve Fitting будет выглядеть, как показано на рис. 11.12.

Приложение Curve Fitting предоставляет широкий спектр интерактивных инструментов для настройки параметров аппроксимирующей функции (в том числе путем выбора необходимых опций), а также разнообразные способы сохранения полученных результатов. Например, выбрав пункт `Generate Code` в разделе `File` главного меню приложения Curve Fitting, можно сгенерировать программный код, воспроизводящий полученные результаты (в виде нестандартной функции MatLab).

Функция `fit`, а следовательно, и приложение Curve Fitting позволяют строить аппроксимирующие функции, зависящие как от одного, так и от двух аргументов. С особенностями использования функции `fit` и приложения Curve Fitting для построения аппроксимирующих функций, зависящих от двух переменных, можно ознакомиться в справочной системе MatLab.

11.5. Элементы структурного программирования

В табл. 11.16 приведено сопоставление операторов и функций MuPAD и MatLab, используемых для автоматизации вычислений и отладки программ. Все перечисленные в табл. 11.16 операторы и функции MatLab доступны, независимо от наличия лицензии на символьные вычисления.

Перечисленные в табл. 11.16 функции и операторы MuPAD детально описаны в предыдущих главах пособия. С особенностями использования перечисленных в табл. 11.16 функций и оператором MatLab можно познакомиться в справочной системе MatLab или в другой учебной литературе по программированию на языке MatLab. Применение этих функций и операторов в символьных вычислениях осуществляется так же, как при

выполнении несимвольных вычислений. Однако следует помнить, что функции MatLab `rand`, `randi`, `randn` для генерации случайных чисел возвращают одно или несколько несимвольных значений. Для использования в символьных вычислениях может потребоваться преобразовать полученные с помощью этих функций значения в символьные с помощью функции `sym`.

Таблица 11.16

Функции и операторы для структурного программирования в MuPAD и MatLab

Случайное вещественное число, подчиняющееся равномерному распределению на интервале [0, 1]	
MuPAD	`frandom(<опция>)`
MatLab	`rand`
Случайное целое число, подчиняющееся равномерному распределению на заданном интервале	
MuPAD	`random(<начало>..<конец>)`
MatLab	`randi([<начало>, <конец>])`
Случайное вещественное число, подчиняющееся нормальному распределению с заданным математическим ожиданием и среднеквадратичным отклонением	
MuPAD	`stats::normalRandom(<м. ожидание>, <ср.кв. отклонение>, <опция>)`
MatLab	`<м. ожидание> + <ср.кв. отклонение> * randn`
Оператор присвоения	
MuPAD	`<Переменная> := <выражение>`
MatLab	`<Переменная> = <выражение>;`
Ввод информации с клавиатуры	
MuPAD	`<Переменная> := input("<приглашение>")` `<Переменная> := textinput("<приглашение>")`
MatLab	`<Переменная> = input('<приглашение>');`
Вывод информации	
MuPAD	`print(<опции>, <переменная1>, <переменная2>,…)`
MatLab	`disp (<переменная или текст>)` `disp ([<переменная> <переменная> …])`

Продолжение табл. 11.16

Открытие и закрытие файла	

MuPAD	`<Идентификатор файла> := fopen ("<имя файла>",`
	`<цель>, <тип>, <опция>)`
	`fclose (<идентификатор файла>)`

MatLab	`<Идентификатор файла> = fopen ('<имя файла>',`
	`<цель>);`
	`fclose (<идентификатор файла>);`

Чтение данных из файла	

MuPAD	`<Переменная> := ftextinput (<идентификатор файла>,`
	`<опция>)`
	`ftextinput (<идентификатор файла>,`
	`<переменная1>, <переменная2>, …, <опция>)`
	`<переменная> := finput (<идентификатор файла>,`
	`<опция>)`
	`finput (<идентификатор файла>, <опция>,`
	`<переменная1>, <переменная2>, …)`

MatLab	`<переменная> = fgetl (<идентификатор файла>);`
	`<переменная> = scanf (<идентификатор файла>,`
	`<формат>, <размер>);`

Запись данных в файл	
MuPAD	`fprint (<опции>, <формат>, <идентификатор файла>,` `<переменная1>, <переменная2>, …)`
MatLab	`fprintf (<идентификатор файла>, <формат>,` `<переменная>)`
Сохранение переменных в файле	
MuPAD	`write (<формат>, <опция>, <идентификатор файла>,` `<переменная1>, <переменная2>, …)`
MatLab	`save ('<имя файла>', <переменная1>,` `<переменная2>, …, <опции>)`
Загрузка переменных и команд из файла	
MuPAD	`fread (<идентификатор файла>, <опции>)` `read ("<имя файла>", <опции>)`
MatLab	`load ('<имя файла>', <опции>, <переменная1>,` `<переменная2>, …)`
Создание массива	
MuPAD	`<Переменная> := array (<min индекс1> ..` `<max индекс1>, <min индекс2> .. <max индекс2>,`

	<min индекс3> .. <max индекс3>, …, <список значений>)
MatLab	<Переменная> = [<список1 значений>; <список2 значений>;…] Преобразование в массив другой размерности reshape(<исходный массив>, <max индекс1>, <max индекс2>, …)
Создание одномерного массива, состоящего из *n* элементов, значения которых равномерно расположены на интервале [*a*, *b*]	
MuPAD	array(1..n, [$ x=a..b step (b-a)/(n-1)])
MatLab	linspace(a, b, n) a:(b-a)/(n-1):b
Сумма и произведение элементов последовательности, одномерного массива	
MuPAD	`+`(<последовательность>) `*`(<последовательность>)
MatLab	sum(<одномерный массив>) prod(<одномерный массив>)
Нахождение минимального и максимального значения в списке, одномерном массиве	
MuPAD	min(<список>) min(<одномерный массив>) max(<список>) max(<одномерный массив>)
MatLab	min(<одномерный массив>) max(<одномерный массив>)
Сортировка элементов списка, одномерного массива по возрастанию	
MuPAD	sort(<список>)
MatLab	sort(<одномерный массив>)
Оператор условного перехода if	
MuPAD	if <условие1> then <операторы1> elif <условие2> then <операторы2> elif <условие3> then <операторы3> . . . else <операторы> end_if

MatLab	`if <условие1>;  <операторы1>`
	`elseif <условие2>;  <операторы2>`
	`elseif <условие3>;  <операторы3>`
	`. . .`
	`else; <операторы>`
	`end`

Оператор множественного выбора	

MuPAD	`case <ключ>`
	`of <значение1> do  <операторы1>; break`
	`of <значение2> do  <операторы2>; break`
	`. . .`
	`of <значениеN> do  <операторыN>; break`
	`otherwise  <операторы>`
	`end_case`

MatLab	`switch <ключ>`
	`case <значение1>;  <операторы1>`
	`case <значение2>;  <операторы2>`
	`. . .`
	`case <значениеN>;  <операторыN>`
	`otherwise;  <операторы>`
	`end`

Оператор счетного цикла	

MuPAD	`for <параметр цикла> from <начало>`
	`to <конец> step <шаг> do <операторы>`
	`end_for`
	`for <параметр цикла> from <начало>`
	`downto <конец> step <шаг> do <операторы>`
	`end_for`
	`for <параметр цикла> in <список> do`
	`<операторы>`
	`end_for`
	`for <параметр цикла> in <множество> do`
	`<операторы>`
	`end_for`

MatLab	`for <параметр цикла> = <начало>:<шаг>:<конец>`
	`<операторы>`
	`end`

Окончание табл. 11.16

Операторы условных циклов	

MuPAD	`while <условие> do  <операторы>`
	`end_while`
	`repeat  <операторы>`
	`until <условие> end_repeat`

MatLab	`while <условие>;  <операторы>` `end`
Объявление простой нестандартной функции	
MuPAD	`<Имя> := (<аргументы>) -> <выражение>`

MatLab	`<Имя> = @(<аргументы>) <выражение>;`
	`<Имя>(<аргументы>) = <выражение>;`
	– только для символьной функции

Объявление нестандартной функции	

MuPAD	`<Имя> := proc(<аргумент1> = <значение1> :`
	`<тип1>, <аргумент2> = <значение2> : <тип2>, …) :`
	`<тип результата>`
	` option <опции>;`
	` local <локальные переменные>;`
	` save <глобальные переменные>;`
	`begin`
	`<операторы>;`
	`<выражение>`
	`end_proc`

MatLab	`function [<результаты>] = <имя>(<аргументы>)`
	`<операторы>`
	`<результаты> = <выражение>;`
	`end`

Количество аргументов, указанных при вызове функции	
MuPAD	`<Количество аргументов> := args(0)`
MatLab	`<Количество аргументов> := nargin`
Вывод текста объявленной ранее функции	
MuPAD	`expose(<имя функции>)`
MatLab	`type <имя функции>`

Рассмотрим пример использования операторов цикла и условного перехода при выполнении символьных вычислений для нахождения экстремумов функции (точек, в которых ее производная обращается в ноль).

Чтобы вычислить значения функции в каждой точке, где производная этой функции обращается в ноль, можно воспользоваться циклом `for`, а чтобы проверить знак второй производной в этих точках – условным оператором `if` (если вторая производная отрицательна, то в этой точке функция имеет максимум):

```
syms x
f(x) = x^4/4 - 122/45*x^3 + 77/30*x^2 + 98/5*x -50;
f1(x) = diff(f, x); f2(x) = diff(f, x, 2);
X0 = solve(f1(x)==0, x, 'MaxDegree', 3);
disp(['Функция f(x) = ' char(f(x))]), disp('имеет:')
for k = 1:3
   if vpa(f2(X0(k))) < 0; s = 'Максимум';
   else                   s = 'Минимум ';
   end
   Y0(k) = f(X0(k));
   disp([s ' при x = ' char(X0(k)) ...
              ' y = ' char(Y0(k))])
end
```

Необходимо отметить, что последовательность символов `...` используется в MatLab для переноса части текста команды на другую строку.

```
Command Window
Функция f(x) = (98*x)/5 + (77*x^2)/30 - (122*x^3)/45 + x^4/4 - 50
имеет:
Минимум  при x = -6/5 y = -40388/625
Максимум при x = 7/3 y = -84191/4860
Минимум  при x = 7 y = -4201/36
fx >>
```

Рис. 11.13. Пример результатов использования символьных вычислений для нахождения экстремумов функции

Если этот программный код записан в m-файле, то результат его выполнения в командном окне будет иметь вид, изображенный на рис. 11.13.

Точки, в которых функция имеет максимум или минимум, далее можно отметить на графике. Например:

```
% график функции
gr = ezplot(f, [-4, 9]);
set(gr, 'LineWidth', 2); title('Экстремумы функции');
```

```
% изображение экстремумов
hold on; uz = plot(X0, Y0,'o');
set(uz, 'LineWidth', 2); hold off
```

В результате появится графическое окно, в котором на графике функции маленькими окружностями будут отмечены точки, соответствующие экстремумам функции. В MatLab 2016 и выше вместо функции `ezplot` следует использовать функцию `fplot`.

Как в MuPAD, так и в MatLab существуют стандартные функции, позволяющие запрашивать и получать текущее время CPU, а также измерять промежуток времени, которое затратила программа на выполнение указанных операторов. Эти функции перечислены в табл. 11.17. Перечисленные в табл. 11.17 функции MuPAD детально описаны в предыдущих главах пособия. Приведенные в табл. 11.17 функции MatLab включены в основное ядро MatLab и доступны независимо от наличия лицензии на символьные вычисления. С особенностями использования этих функций можно ознакомиться в справочной системе MatLab.

Таблица 11.17

Функции для определения времени выполнения операторов в MuPAD и MatLab

Определение времени выполнения операторов	
MuPAD	`time()` текущее время CPU `time(<оператор1>, <оператор2>, …)`

MatLab	`cputime` текущее время CPU
	`tic`
	`  <оператор1>, <оператор2>, …`
	`toc`
	`timeit(<функция>)`

Сочетание стандартных операторов программирования, функций MatLab с символьными вычислениями позволяет существенно расширить возможности программной обработки данных и область их применения.

Глава 12. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ НА КОМПЬЮТЕРЕ

Перед выполнением заданий ознакомьтесь со следующими ***рекомендациями***:

1. Задания из тем 1–8 выполняйте в приложении MuPAD (mn-файлах). Задания из темы 9 – в m-файлах MatLab.

2. В приложении MuPAD всегда начинайте новую программу с удаления из памяти результатов всех предыдущих вычислений с помощью команды `reset()` в первом командном поле.

3. В программе придерживайтесь тех же обозначений, которые использованы в задании.

4. Задания, относящиеся к разным темам, выполняйте в разных файлах.

5. В каждом mn-файле в первом текстовом поле укажите свою фамилию, инициалы, номер группы и дату выполнения задания. Перед каждым командным полем, с которого начинается выполнение очередного задания, добавьте текстовое поле и укажите в нем номер задания.

6. В приложении MuPAD выполняйте все записанные в mn-файле вычисления с помощью команды `Evaluate All` из раздела `Notebook` главного меню. Проверяйте правильность выполненных программой вычислений.

7. Перед каждым выполнением вычислений сохраняйте файл с программой.

8. После завершения всех вычислений сохраните окончательный вариант программы, а затем, если вычисления выполнялись в mn-файле, экспортируйте его в pdf-формат.

Тема 1. Выражения, функции. Графики функций

Задание 1.1. Запишите выражения:

$\frac{3}{5}+\frac{5}{9}+\frac{7}{12}$; $\sqrt{24}$; $\sin\left(\frac{5\pi}{3}\right)$; $\arccos(-1)$; $|-3|$. Выполните вычисления.

Задание 1.2. Запишите выражение: $3x + x^2 - 4x - 5x^2 + x - 8 + 6x$. Приведите подобные.

Задание 1.3. Присвойте переменной z числовое значение $6+\frac{1}{2}-\frac{3}{7}$, а переменной s символьное выражение $\frac{x}{2}+\sin(y-1)$.

Задание 1.4. С учетом предыдущих обозначений присвойте переменной k_1 следующее выражение: $\sqrt{7z-z^2-5}$.

Задание 1.5. С учетом предыдущих обозначений присвойте переменной k_2 следующее выражение: $\frac{2}{5}z+4s-zs$.

Задание 1.6. Объявите функцию $f_1(x) = x^2 - 10$. Вычислите значения функции $f_1(x)$ при $x = 1$; $x = -3{,}2$; $x = a - 2$; $x = z$. Присвойте переменной y_1 значение $f_1(x - 6)$.

Задание 1.7. Объявите функцию $f_2(x, y) = \frac{x}{2}+\sin^2(y-1)+x\sin(y-1)$. Вычислите значения функции $f_2(x, y)$ при $x = 2$, $y = 1$; $f_2(3, 4)$; $f_2(-1, \pi)$; $f_2(u - 2, \pi/2 + 1)$; $f_2(4y, -x)$.

Задание 1.8. Объявите функцию $f_3(x) = (x - Q)^2 + 10$. Вычислите значения функции: $f_3(x)$, $f_3(0)$, $f_3(5)$, $f_3(-5)$, $f_3(a - 2)$. Присвойте переменной Q значение 5 и вновь вычислите те же значения функции: $f_3(x)$, $f_3(0)$, $f_3(5)$, $f_3(-5)$, $f_3(a - 2)$. Присвойте переменной Q значение –10 и вновь вычислите те же значения функции: $f_3(x)$, $f_3(0)$, $f_3(5)$, $f_3(-5)$, $f_3(a - 2)$.

Задание 1.9. С учетом предыдущих обозначений объявите функцию $g(x) = f_1(x - 3) + 15\cos(2x + 4)$. Постройте график функции $g(x)$.

Задание 1.10. Постройте график функции $g(x - 6)$ на интервале $[-10, 3]$.

Задание 1.11. Постройте график кусочно-непрерывной функции $h(x)$:

$$h(x)=\begin{cases}(2x-4)^2-119, & x<-4;\\ (x+9)^2, & -4\le x<4;\\ 209-10x, & x\ge 4.\end{cases}$$

Подберите интервал, на котором все ветви функции $h(x)$ имеют одинаковую длину.

Задание 1.12. На подобранном в задании 1.11 интервале постройте график двух объявленных ранее функций $f_1(x)$ и $h(x)$. Для каждой кривой задайте цвет и толщину линии (разные). На графике изобразите координатную сетку.

Задание 1.13. На подобранном в задании 1.11 интервале постройте график двух объявленных ранее функций $f_1(x)$, $h(x)$ и выражения $\frac{600}{2+(x-2)^2}+\frac{500}{1+(x+5)^2}$. Для каждой кривой задайте цвет и толщину линии (разные). На графике изобразите координатную сетку. Подпишите оси координат иначе, чем это делается по умолчанию.

Тема 2. Аналитические преобразования выражений, уравнений, неравенств

Задание 2.1. Запишите выражение $\sin^2\alpha+\sqrt{3}\cos\alpha\sin\beta$. Подставьте в это выражение сначала значения $\alpha=\frac{\pi}{6}$ и $\beta=\pi$ с помощью функции `evalAt`; а затем $\alpha=\pi$ и $\beta=\frac{\pi}{4}$ с помощью функции `subs`. Если какой-либо из полученных результатов можно упростить, сделайте это с помощью функции `simplify`.

Задание 2.2. Запишите выражение $a+(a-2)^2+\frac{5}{a+2}-\frac{10}{a^2+2}$. Получите новые выражения, с помощью функции `subs`, выполнив в исходном выражении по очереди следующие замены переменных: $a=b$; $a=c+2$; $a=c-2$. Выполните следующие вычисления: из исходного вычтите последнее полученное выражение. Задайте $a=3$, $c=5$. Вновь выведите результат вычисления и проанализируйте его.

Задание 2.3. Запишите выражение $(x+3)+(x+3)^2-\dfrac{x+3}{(x+3)^2+6}+3$. С помощью функции `subs` в этом выражении по очереди выполните следующие замены: $x = t - 3$; $x + 3 = t$. В исходном выражении с помощью функции `collect` соберите члены при разных степенях выражения $(x + 3)$. В полученном выражении вновь сделайте замену: $x + 3 = t$. Проанализируйте результаты и определите, какую из функций подстановки необходимо использовать, чтобы после замены $x + 3 = t$ выражение не содержало переменной x.

Задание 2.4. Запишите выражение $u\sin\left(q\cdot(u+\sqrt{3}\sin q)\right)$. Подставьте в это выражение последовательно все значения переменной u, расположенные на интервале [–1, 4] с шагом 1/2. В полученной последовательности выражений определите количество элементов. В эту последовательность подставьте числовое значение $q=\dfrac{\pi}{3}$ двумя способами – с помощью функций `evalAt` и `subs`. При необходимости упростите полученные результаты. Рекомендации: так как функцию `simplify` нельзя применять к последовательностям, перед упрощением преобразуйте последовательность в список `[]`, а после упрощения списка вновь преобразуйте его в последовательность (использовав функцию `op`).

Задание 2.5. Запишите выражение $r^2-rs^2-rs+r+s$. Подставьте в это выражение следующие значения: $r = x - 5$, $s = x + 6$. Раскройте скобки в полученном выражении.

Задание 2.6. Запишите выражение $x^5-4x^4-20x^3+72x^2+51x-180$. Разложите его на множители. Используя функцию `op`, из полученного выражения извлеките все множители. Результат должен являться последовательностью. Вновь разложите исходное выражение на множители, но так, чтобы в полученном выражении множитель x^2-3 был представлен в виде $(x-\sqrt{3})(x+\sqrt{3})$. В полученном выражении раскройте скобки. Вычтите последнее полученное выражение из исходного.

Задание 2.7. Запишите уравнение $2x^3-37x=5x^2-60$. Извлеките правую часть уравнения. Используя арифметические операции, перенесите ее в левую сторону (приведите уравнение к виду $f(x) = 0$). Разложите уравнение на множители. Получите новые уравнения, извлекая и приравнивая к нулю каждый сомножитель. Выполнив необходимые преобразования

(умножение, вычитание констант и т.д., функциональные преобразования), решите каждое из полученных уравнений.

Задание 2.8. Запишите выражение $3x+\frac{313}{x+8}-42$. Приведите это выражение к общему знаменателю. Извлеките из полученной дроби числитель и составьте неравенство: извлеченный числитель дроби больше $103-5x$. Выполнив необходимые преобразования (умножение, вычитание констант и т.д., функциональные преобразования), решите полученное неравенство.

Задание 2.9. Запишите выражение $\frac{16x^4+45x^3+205x^2+585x-27}{16x^3+45x^2+144x+405}$. Разложите выражение на простые дроби. Извлеките из полученного выражения числитель первой дроби и знаменатель второй. Определите, при каких значениях переменной x квадрат числителя первой дроби равен знаменателю второй дроби. Для этого с помощью извлеченных выражений составьте соответствующее уравнение и решите его, выполнив необходимые преобразования (умножение, вычитание констант и т.д., функциональные преобразования).

Задание 2.10. Запишите выражение $y^2\cos 2x - 8\sin(x+y) + x^2\sin 3y + a\sin(x+y) + b\sin(x+y) + 5a\cos 2x + by\sin 3y$. Выпишите из этого выражения все независимые переменные. Скомпонуйте члены при одинаковых тригонометрических функциях. Обозначьте полученное выражение как `r1`. В выражении `r1` раскройте скобки и тригонометрические тождества. В полученном выражении вновь соберите (скомпонуйте) члены при одинаковых тригонометрических функциях. С помощью функции `combine` выполните тригонометрические преобразования. Вновь скомпонуйте члены при одинаковых тригонометрических функциях. Проанализируйте, удалось ли вернуть выражение к виду `r1`.

Задание 2.11. Объявите функцию:

$$f(x)=\cos x+\cos^2 x-\sin x+\sin x\cos x.$$

Запишите новое выражение, подставив в функцию вместо аргумента следующее выражение: $\frac{5\pi}{2}-\frac{\pi}{3}$, без преобразования аргумента (отложенные вычисления). Выполните все вычисления в полученном выражении.

Запишите новое выражение, подставив в функцию вместо аргумента следующее выражение: $2\pi+\frac{\pi}{3}$, без вычисления значений функции

(отложенные вычисления). Выполните все вычисления в полученном выражении.

Задание 2.12. Запишите выражения: $\sin\left(\frac{\pi n}{2}\right)$, $\cos\left(\frac{\pi n}{2}\right)$, $\sin\left(\frac{\pi n}{4}\right)$, $\cos\left(\frac{\pi n}{4}\right)$, $\sin\left(\frac{\pi n}{6}\right)$, $\cos\left(\frac{\pi n}{6}\right)$. Вычислите их значения, если n принимает: а) целые значения; б) четные значения; в) значения, кратные четырем; г) значения, кратные шести. Для этого упростите эти выражения с учетом каждого из перечисленных свойств переменной n по очереди.

Задание 2.13. Очистите (с помощью команды `delete`) переменные a, b, x, y, n, ω. Запишите следующие выражения:

$$\left(\sqrt{x}\right)^2;\ \sqrt{a}\cdot\sqrt{a};\ \frac{\sqrt{a}}{\sqrt{a}};\ \sqrt{x^2};\ \sqrt{a}\sqrt{\frac{1}{a}};\ \frac{\sqrt{y-3}\sqrt{y+3}}{\sqrt{y^2-9}};\ \sqrt{\frac{a}{b-3}}\sqrt{\frac{b+5}{a}};$$

$$\ln a+\ln\frac{1}{a};\ 2\ln\sqrt{x}-\frac{1}{2}\ln x^2;\ 2\ln\sqrt{x}+\frac{1}{2}\ln x^2;\ \ln\frac{a}{b-3}-\ln\frac{a}{b+5};$$

$$\ln(y-3)+\ln(y+3)-2\ln(y^2-9);\ \sin\pi n;\ \sin 2\pi n;\ \cos\pi n;\ \cos 2\pi n;$$

$$\arcsin(\sin\omega);\ \arccos(\cos\omega).$$

С помощью функции `simplify` упростите их.

Объявите все переменные (a, b, x, y, n, ω) вещественными, использовав функцию `assume`. Снова упростите выражения.

Объявите все переменные положительными вещественными. Снова упростите выражения.

Объявите все переменные отрицательными вещественными. Снова упростите выражения.

Объявите все переменные целыми. Снова упростите выражения.

Очистите свойства переменных a, b, x, y, n, ω.

Проанализируйте полученные результаты. Какие минимальные ограничения свойств переменных нужно задать, чтобы упростить каждое из этих выражений? Реализуйте эти действия, использовав функцию `assuming`.

Тема 3. Решение алгебраических и трансцендентных уравнений (неравенств) и их систем

Задание 3.1. Запишите уравнение $5x + \pi = \sqrt{7}$. С помощью функции `isolate` найдите решение уравнения. С помощью функций `subs` и `is` выполните проверку, является ли полученное значение решением этого уравнения. Аналогичным образом решите следующие уравнения:

$5e^{\frac{x}{4}} = 43$; $\sin x = 1$; $\sin y - \cos y = 1$.

Задание 3.2. Запишите уравнение $8x^3 - 46x^2 - 67x + 330 = 0$. Разложите уравнение на множители. Мысленно определите непересекающиеся интервалы, на каждом из которых уравнение имеет по одному корню. С учетом этих интервалов с помощью функции `isolate` последовательно найдите все корни уравнения. Для каждого корня с помощью функций `subs` и `is` выполните проверку, является ли данное значение решением уравнения.

Задание 3.3. Запишите уравнение $9x^3 - 15x^2 + 3\pi x - 5\pi = 5x^2 - 3x^3$. Перенесите все операнды в одну сторону. Разложите уравнение на множители. С помощью функции `solve` найдите (аналитически) ВСЕ корни уравнения в рациональном формате. При необходимости используйте опции. Получите числовые значения всех корней в десятичном формате. Для каждого корня, записанного в рациональном формате, с помощью функций `subs` и `is` выполните проверку, является ли полученное значение решением этого уравнения. Найдите все вещественные корни уравнения в рациональном формате. Аналогичным образоми решите следующее уравнение: $3x^5 - 102x^3 + 67x = 16x^4 - 22x^2 + 198$.

Задание 3.4. Запишите неравенство: $24x > 5$. Перенесите все операнды в одну сторону. Разложите неравенство на множители. С помощью функции `solve` найдите решение неравенства. Определите, какие опции необходимо при этом использовать. Из каждого интервала, полученного в результате решения, выберите по одному значению. Для каждого из этих значений с помощью функций `subs` и `is` выполните проверку, является ли данное значение решением неравенства. Аналогичным образом найдите решения следующих неравенств: $64y^2 < 121$; $x^2 - 2x > 24$; $x^2 - 5x + 3 < 0$; $x^3 + 11x > 6x^2 + 6$.

Задание 3.5. Запишите функцию (полином): $f(x) = 42x^5 - 869x^4 + 4739x^3 + 5405x^2 - 77891x + 60258$. Запишите уравнение $f(x) = 0$. Численно с точностью до двух значащих цифр определите (с помощью функции `numeric::polyrootbound`) интервал, на котором расположены все корни полинома. Постройте график функции на этом интервале. Численно (с помощью функции `numeric::polyroots`) найдите все корни уравнения (полинома) с точностью до 11 значащих цифр. Для каждого из полученных значений вычислите значение полинома. Для каждого ли из полученных корней полином обращается в ноль? Почему?

Аналогичным образом найдите все корни полиномов Чебышева $T_2(x)$, $T_4(x)$, полиномов Лежандра $P_2(x)$, $P_5(x)$.

Задание 3.6. Запишите функцию $g(x) = x^3 + 1 - e^x$. Запишите уравнение $g(x) = 0$. Численно с точностью до двух значащих цифр определите (с помощью функции `numeric::realroots`) все интервалы, на каждом из которых расположено по одному вещественному корню уравнения. Постройте график функции, подобрав такой интервал, который содержит все вещественные корни уравнения. Используя полученные ранее интервалы, последовательно (с помощью функции `numeric::realroot`) найдите все вещественные корни уравнения с точностью до 11 значащих цифр. Для каждого из полученных значений вычислите значение функции. Для каждого ли из полученных корней функция обращается в ноль? Почему?

Задание 3.7. Запишите функцию $h(x) = \ln(x+5) - 2\cos\frac{x}{2}$. Запишите уравнение $h(x) = 0$. Численно с точностью до двух значащих цифр определите (с помощью функции `numeric::realroots`) все интервалы, на каждом из которых расположено по одному вещественному корню уравнения. Постройте график функции, подобрав такой интервал, который содержит все вещественные корни уравнения. Используя полученные ранее интервалы, с помощью функции `numeric::fsolve` без опций найдите второй корень уравнения с точностью до 5 значащих цифр. Далее с помощью функции `numeric::fsolve` найдите все корни уравнения с точностью до 11 значащих цифр, при необходимости используйте опции.

Задание 3.8. Используя функцию `numeric::solve`, численно с точностью до 5 значащих цифр найдите все вещественные решения следующих уравнений:

$\sin x = \ln 12x$; $\frac{x^2}{20} - 10x = 15\cos(x+15)$; $\frac{5\sin x}{x} - \frac{x^2}{200} = 0$;

$5^x - 6x = 7$; $x^2 - 2 = \ln(x+5)$; $x^5 - 4x^3 + 3x^2 + 7x - 1 = 0$.

Убедитесь в полноте полученных решений с помощью графиков соответствующих функций.

Задание 3.9. Аналитически решите следующие системы линейных уравнений (a не является неизвестным):

$$\begin{cases} 3x + 2y = 3; \\ x - y = -4; \end{cases} \quad \begin{cases} x + y + z = 1; \\ 3x + y = 3; \\ x - 2y - z = 0; \end{cases} \quad \begin{cases} 4ax + 3y = 12; \\ 5x - 7ay = 35; \end{cases} \quad \begin{cases} x + ay + z = 1; \\ 3x + y = 3a; \\ x - 2y - az = 0. \end{cases}$$

Выполните проверку: каждое из полученных решений подставьте в исходную систему. Определите, при каких *вещественных* значениях параметра a все решения двух последних систем являются положительными. Для этого составьте и решите соответствующие неравенства.

Задание 3.10. Численно с точностью до 4 значащих цифр решите систему линейных уравнений:

$$\begin{cases} 17x_1 + 10x_2 = 31; \\ 33x_1 - 77x_2 = 19; \end{cases} \quad \begin{cases} 4x_1 - 12x_2 + 23x_3 = 15; \\ 2x_1 - 4x_2 + 3x_3 = 2; \\ 5x_1 + 8x_2 - 7x_3 = 8; \end{cases} \quad \begin{cases} 3x_1 + 14x_2 + 6x_3 - 15x_4 = 2; \\ 8x_1 - 6x_2 + 5x_3 - 12x_4 = -4; \\ 6x_1 - 8x_2 - 4x_3 + 8x_4 = 7; \\ 17x_1 - 8x_2 + 14x_3 - 7x_4 = 13. \end{cases}$$

Выполните проверку: каждое из полученных решений подставьте в исходную систему.

Задание 3.11. Численно с точностью до 4 значащих цифр решите систему полиномиальных уравнений:

$$\begin{cases} x^3 + y^3 = 1; \\ 5x + 5y = 6; \end{cases} \quad \begin{cases} x^2 + 2y^2 = 15; \\ x^2 - 2xy = -5; \end{cases} \quad \begin{cases} x + z - xy + x^2 = 6; \\ 3x + y = 3; \\ x - 2y - z = 4; \end{cases} \quad \begin{cases} uvx^2 = 8; \\ vx^2w = 24; \\ x^2wu = 12; \\ u + v + w = x + 4. \end{cases}$$

Выполните проверку: каждое из полученных решений подставьте в исходную систему.

Задание 3.12. Численно с точностью до 4 значащих цифр решите систему трансцендентных (неполиномиальных) уравнений:

$$\begin{cases} 5x - 5\cos(y-1) = 4; \\ y - \cos x = 2; \end{cases} \quad \begin{cases} x^2 + y^2 = 5; \\ 1 - 2e^{-xy} = y; \end{cases} \quad \begin{cases} \sqrt{x+y} + \sqrt[4]{x-y} = 8; \\ \sqrt[4]{x^3 + x^2y - xy^2 - y^3} = 12; \end{cases}$$

$$\begin{cases} y+2x=4; \\ \sqrt{\dfrac{3x}{x+y}}+\sqrt{\dfrac{x+y}{9x}}=2; \end{cases} \quad \begin{cases} \sin^2 x+\sin^2 y=\dfrac{7}{4}; \\ x+y=\dfrac{5\pi}{6}; \end{cases} \quad \begin{cases} x+y=\sin \pi x; \\ x-y=\cos \pi y; \end{cases}$$

$$\begin{cases} (x+y)^{(x-y)}=(x-y)^{(x+y)}; \\ \log_2 x-\log_2 y=1. \end{cases}$$

Обратите внимание, что для получения решения может потребоваться конкретизировать интервал поиска для каждой из искомых переменных или использовать некоторые опции. Если система имеет несколько решений, получите два из них.

Выполните проверку: каждое из полученных решений подставьте в исходную систему.

Тема 4.1. Решение задач математического анализа

Задание 4.1. Запишите сумму $\sum_{n=1}^{10} n^3$, используя отложенные вычисления. Вычислите эту сумму аналитически. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Эти же действия выполните для следующих сумм:

$$\sum_{n=1}^{\infty}\frac{2n+3}{(n+1)(n+2)(n+3)};\ \sum_{k=1}^{n}k^6;\ \sum_{k=0}^{n}x^k \text{ при } x\neq 1;\ \sum_{n=0}^{10}\sin(n\pi+1);$$

$$\sum_{n=1}^{9}\sin(n\pi+x);\ \sum_{n=1}^{\infty}\frac{1}{n^6};\ \sum_{n=1}^{\infty}\frac{1}{(4n^2-1)^3};\ \sum_{n=1}^{\infty}\frac{1}{n(9n^2-1)};\ \sum_{n=1}^{\infty}\frac{1}{2^n n^2};$$

$$\sum_{n=0}^{\infty}\frac{1}{(2n)!};\ \sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!};\ \sum_{n=0}^{\infty}\frac{(x\ln a)^n}{n!};\ \sum_{k=1}^{\infty}kx^k;\ \sum_{k=1}^{n}\cos kx;\ \sum_{k=1}^{\infty}\frac{x^k}{k};\ \sum_{k=1}^{\infty}\frac{1}{kx^k};$$

$$\sum_{k=1}^{\infty}\left(\frac{1}{k+x}-\frac{1}{k+y}\right);\ \sum_{k=1}^{\infty}\frac{1}{(2k-1)^2+x^2};\ \sum_{k=1}^{n-1}\frac{1}{k};\ \sum_{k=1}^{\infty}\left(\frac{1}{k+x}-\frac{1}{k+1}\right);$$

$$\sum_{k=1}^{\infty}\frac{1}{k(k+x)};\ \sum_{k=1}^{\infty}\frac{\sin kx}{k^3} \text{ при } x\in[0,\ 2\pi];\ \sum_{k=0}^{\infty}\frac{(x/k)^{2k}}{k!^2};$$

$$\sum_{k=1}^{\infty}\frac{(k+1)^3}{k!}\,;\ \sum_{k=1}^{n}\frac{x^k}{k(k+1)}\,.$$

Задание 4.2. Запишите произведение $\prod_{k=1}^{4}k^2$, используя отложенные вычисления. Вычислите это произведение аналитически. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Эти же действия выполните для следующих произведений:

$$\prod_{k=1}^{n}\left(\frac{1}{k}+1\right);\ \prod_{k=1}^{n}k\,;\ \prod_{k=1}^{3}(x-1)(y+2)\,;\ \prod_{k=1}^{3}\left(\frac{x}{y}+2^{-k}\right);\ \prod_{k=1}^{6}\left(\frac{2k-x}{ak}+x^k\right);$$

$$\prod_{k=-2}^{4}(x^k+1)\,;\ \prod_{k=1}^{8}bc_k y^k\,;\ \prod_{k=2}^{\infty}\left(1-\frac{1}{k^2}\right);\ \prod_{k=1}^{\infty}\left(1-\frac{1}{4k^2}\right);\ \prod_{k=1}^{\infty}\left(1-\frac{1}{(2k+1)^2}\right)$$

;

$$\prod_{k=1}^{\infty}\frac{e^{1/k}}{1+1/k}\,;\ \prod_{k=1}^{\infty}\frac{(1+1/k)^x}{1+x/k}\ \text{ при } x>1;\ \prod_{k=1}^{n}\left(1+\frac{1}{k^2}\right);\ \pi x\cdot\prod_{n=1}^{\infty}\left(1-\frac{x^2}{n^2}\right).$$

Задание 4.3. Используя отложенные вычисления, запишите следующие выражения:

$$\sum_{k=1}^{3}\left((k+1)\prod_{n=1}^{2}ny\right);\ \sum_{k=1}^{4}\left(\frac{(1+x)^{k+1}}{(2k-1)!}\prod_{p=1}^{k}\frac{y}{p+1}\right);\ \sum_{k=1}^{\infty}\left(x^k\sum_{p=1}^{k}\frac{1}{p}\right);$$

$$\sum_{n=1}^{4}\left(2^{n-1}\prod_{k=0}^{n-1}\sin\left(\pi y+\frac{\pi k}{n}\right)\right);\ \sum_{n=1}^{\infty}\left(\frac{1}{n^4}\sum_{k=0}^{\infty}\frac{(-1)^k(nx)^{2k}}{(2k)!}\right).$$

Аналитически вычислите значения этих выражений при $x = 3/4$ и (или) $y = 5/3$. Результат представьте в десятичном виде.

Задание 4.4. Запишите предел $\lim\limits_{x\to\infty}e^{-x}$, используя отложенные вычисления. Вычислите этот предел. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Эти же действия выполните для следующих пределов:

$$\lim_{x\to 0}\frac{\sin 3x}{x}\,;\ \lim_{x\to\infty}\left(1+\frac{2}{x}\right)^x;\ \lim_{x\to 0}\frac{\cos x-1}{x^2}\,;\ \lim_{x\to 0}\frac{e^{ax}-e^{bx}}{x}\,;\ \lim_{x\to\infty}\frac{\pi-2\operatorname{arctg}x}{e^{b/x}-1}\,;$$

$$\lim_{x\to 0}\sin^x ax\,;\ \lim_{x\to 0}\frac{x}{\operatorname{tg} ax}\,;\ \lim_{x\to \pi/2}\operatorname{tg}^{2\cos x} x\,;\ \lim_{x\to 1/a}\left(\frac{1}{ax-1}-\frac{1}{\ln ax}\right);$$

$$\lim_{x\to a}\left(\frac{x}{a}\right)^{\ln(a-x)}\,;\ \lim_{x\to 0}\frac{\sqrt{x^2+p^2}-p}{\sqrt{x^2+q^2}-q}\,;\ \lim_{x\to 1}\operatorname{tg}\left(\frac{\pi n x}{2}\right)\ln(x-1)\,.$$

При необходимости укажите ограничения на свойства символьных переменных так, чтобы все неопределенности были раскрыты.

Задание 4.5. Запишите выражение $\lim\limits_{n\to\infty}\left(\frac{1}{n^2}\sum\limits_{k=1}^{n}k\right)$, используя отложенные вычисления. Вычислите значение этого выражения. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Эти же действия выполните для следующих выражений:

$$\lim_{n\to\infty}\left(\frac{1}{n^2}\sum_{k=1}^{n}k^2\right);\ \lim_{n\to\infty}\left(-\ln n+\sum_{k=1}^{n}\frac{1}{k}\right);\ \lim_{x\to 2}\left(\sum_{k=0}^{\infty}\frac{x^k(k+1)}{k!}\right);\ \lim_{x\to\frac{\pi}{2}}\left(\sum_{k=1}^{\infty}\frac{\sin(kx)}{k^2}\right);$$

$$\lim_{x\to 0}\left(\frac{1}{x}\sum_{k=0}^{\infty}\frac{(-1)^k\left((ax)^{2k+1}+(bx)^{2k+1}\right)}{(2k+1)!}\right);\ \lim_{x\to 0}\left(\prod_{k=1}^{\infty}\left(1-\frac{4}{3}\sin^2\frac{x}{3^k}\right)\right).$$

Задание 4.6. Проверьте, равны ли пределы слева и справа для функции $f(x)=2^{\frac{x}{9-x^2}}$ в точках $x=-3$ и $x=3$. Аналогичные действия выполните для следующих функций:

$f(x)=\dfrac{\sin bx}{x}$ в точке $x=0$; $f(x)=\dfrac{|x-4|}{x-4}$ в точке $x=4$;

$f(x)=\operatorname{tg}\dfrac{1}{x}$ в точках $x=\pm\dfrac{2}{\pi}$; $f(x)=\dfrac{x-c}{x(x^2-4)}$ в точках $x=0,\pm 4$.

Задание 4.7. Запишите производную $\dfrac{d^3}{dx^3}\operatorname{tg} x$, используя отложенные вычисления. Вычислите эту производную, упростите результат. Эти же действия выполните для следующих производных:

$$\frac{d^4}{dx^4}\sqrt{1+x^3}\,;\ \frac{\partial}{\partial x}\left(\frac{\cos xy}{y}\right);\ \frac{\partial}{\partial y}\left(\frac{\cos xy}{y}\right);\ \frac{\partial^2}{\partial x\partial y}\left(\frac{\cos xy}{y}\right);\ \frac{\partial^2}{\partial y\partial x}\left(\frac{\cos xy}{y}\right).$$

Задание 4.8. Объявите функцию $f(x)=\dfrac{\operatorname{tg}x}{x}$. Вычислите все производные этой функции до второго порядка включительно так, чтобы результатами являлись функции. Эти же действия выполните для следующих функций: $f(x,y)=x^2y^2$; $f(x,y)=\dfrac{x^a y^b}{x^3-e^{-y}}$ (a и b – константы).

Задание 4.9. Вычислите производную $\dfrac{d}{dx}\sin 2x$ в точке $x = 0$ (т.е. вычислите производную и подставьте в полученный результат указанное значение переменной). Если в результате вычислений получится рациональное число, представьте его в десятичном формате. Аналогичные действия выполните для следующих выражений:

$$\left.\frac{d^2}{dx^2}e^{-ax}\right|_{x=0};\quad \left.\frac{1}{4!}\frac{d^4}{dx^4}(x^2-1)^4\right|_{x=1/2};\quad \left.\frac{e^x}{n!}\frac{d^5}{dx^5}e^{-x}x^n\right|_{\substack{x=3/5\\ n=5}};\quad \left.\frac{\partial^3}{\partial y^2\partial x}x^n e^{-y}\right|_{\substack{x=a\\ y=2\ln a}}.$$

Задание 4.10. Проверьте, сколько раз нужно применить правило Лопиталя (заменить числитель и знаменатель дроби их производными), чтобы вычислить придел $\lim\limits_{x\to 0}\dfrac{e^{\operatorname{tg}x}-e^x}{\operatorname{tg}2x-2x}$.

Для этого: запишите дробь $\dfrac{e^{\operatorname{tg}x}-e^x}{\operatorname{tg}2x-2x}$; из числителя и знаменателя дроби составьте список; убедитесь, что при подстановке $x = 0$ все элементы списка обращаются в ноль; продифференцируйте полученный список по dx; в полученный после дифференцирования список вновь подставьте $x = 0$. Продолжайте дифференцирование списка, пока хотя бы один из его элементов при подстановке $x = 0$ не перестанет обращаться в ноль. Вычислите отношение элементов последнего из полученных списков при $x = 0$. Сравните полученный результат со значением искомого предела, записав и вычислив его с помощью функции `limit`.

Задание 4.11. Объявите функцию $f(x)=15e^{-x^2/3}\cos 3x-2$. Найдите ее экстремумы, отметьте их на графике. Определите, какие из них соответствуют точкам максимума, а какие – точкам минимума функции. Подберите такой интервал вблизи начала координат, чтобы функция имела на нем 10 экстремумов.

Задание 4.12. Объявите функцию $f(x)=15e^{-x^2/3}\cos(2x+1)-2x$. Найдите ее точки перегиба, отметьте их на графике. Подберите такой интервал вблизи начала координат, чтобы функция имела на нем 10 точек перегиба.

Задание 4.13. Объявите функцию $f(x)=\dfrac{1}{x^2-x-6}-\dfrac{x}{1-e^{-x^2+5x+6}}$. Найдите особые точки функции, отметьте их на оси абсцисс графика функции. Интервал для построения графика подберите так, чтобы особые точки функции были хорошо видны. Определите характер (устранимая, неустранимая, полюс) каждой из найденных особых точек. Для этого вычислите соответствующие пределы (в каждой из особых точек, а также на $\pm\infty$).

Задание 4.14. Объявите функцию $f(x)=25e^{-x^2/4}\sin(x+2)-\dfrac{2x}{5}$. Запишите две функции, касательные к функции $f(x)$ в точках $x=-2$ и $x=2$ соответственно. Выведите на экран уравнения обеих касательных. Постройте график функции и графики касательных. Отметьте на графике функции точки, в которых касательные касаются функции $f(x)$.

Задание 4.15. Объявите функцию $f(x)=\sqrt{\dfrac{x^3}{x-2}}$. Найдите все ее наклонные асимптоты. Изобразите их на графике функции. Интервал для построения графика подберите так, чтобы на нем четко были видны и области, где функция отличается от асимптот, и области, где функция стремится к асимптотам.

Тема 4.2. Решение задач математического анализа (продолжение)

Задание 4.16. С помощью функции `series` разложите математическую функцию $f(x)=\dfrac{1}{x^2(a-x)^{5/3}}$ в ряд: 1) в окрестности нуля; 2) в окрестности точки $x=a$; 3) в окрестности точки $x=2a$, с точностью до членов 4-го порядка. Проанализируйте полученный результат: является ли полученный ряд рядом Тейлора, Маклорена, Лорана, Пюизе.

Задание 4.17. Разложите функцию $f(x) = e^{x-a} + e^{a-x}$ в ряд Тейлора с точностью до 7-го порядка: 1) в окрестности нуля; 2) в окрестности точки $x = a$. Для каждого из этих рядов вычислите отношение четвертого члена ряда к третьему. Определите, для каких вещественных значений x это отношение по модулю меньше единицы. Для этого составьте и решите (с помощью функции `solve`) соответствующее неравенство.

Задание 4.18. Объявите функцию $f(x) = 3\cos^2 2x - 2x^4 \cos 2x - 1$. По графику определите, на каком интервале функция $f(x)$ обращается в ноль. Получите корни уравнения $f(x) = 0$, расположенные вблизи нуля. Для этого разложите $\cos 2x$ в ряд Маклорена с точностью до четвертого порядка $O(x^4)$. Преобразуйте полученный ряд в выражение и подставьте в исходную функцию и в уравнение вместо $\cos 2x$. Решите полученное после преобразования уравнение. Результат представьте в десятичном виде. На интервале, на котором расположены найденные корни, постройте график двух функций: исходной и преобразованной. Отметьте на оси абсцисс этого графика полученные решения. Сравните полученные решения с истинными корнями исходного уравнения.

Задание 4.19. Запишите неопределенный интеграл $\int \sin x\,dx$, используя отложенные вычисления. Вычислите этот интеграл. Эти же действия выполните для следующих интегралов:

$$\int e^{-x} \sin x\,dx;\ \int \sqrt{1-x^2}\,dx;\ \int \frac{x\,dx}{1+x^3};\ \int \operatorname{ch}^3 x\,dx;$$

$$\int x^n \ln x\,dx \text{ при } n \neq -1;\ \int \frac{dx}{a^2 + b^2 \sin^2 x};\ \int \frac{\cos x\,dx}{\sqrt{x}};\ \int \frac{e^{ax}\,dx}{x^3}.$$

Задание 4.20. Запишите определенный интеграл $\int_0^1 \operatorname{tg} x\,dx$, используя отложенные вычисления. Вычислите этот интеграл. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте его в десятичном формате. Эти же действия выполните для следующих интегралов:

$$\int_0^1 \sqrt{\frac{1+x}{1-x}}\,dx;\ \int_0^{2\pi} (1-\cos x)^5 \cos 5x\,dx;\ \int_1^2 x(x^3 - ax)\,dx;\ \int_{\sqrt{a}}^{\sqrt{b}} \frac{\ln x}{x^3}\,dx;\ \int_{-\infty}^{\infty} \frac{dx}{x^2+1};$$

$$\int_0^{\infty} e^{-3x}\cos x\,dx\,;\ \int_0^{\infty}\left(\frac{1}{1+x}-e^{-x}\right)\frac{dx}{x}\,;\ \int_0^{a}\frac{e^{-qx}dx}{\sqrt{qx}}\ \text{ при } q>0;\ \int_0^{b} e^{-x^2}dx\,.$$

Задание 4.21. Определите, какими свойствами (ограничениями на область определения) должны обладать константы a, b, c, q, p, ω и u, чтобы интегралы

$$\int_1^5 \ln\left(1-\frac{x}{a}\right)dx\,;\ \int_0^1 \sin(bx)\sqrt{1-x^2}\,dx\,;\ \int_{-5}^{\infty} e^{cx}dx\,;\ \int_0^{\infty}\frac{e^{-qx}dx}{e^{-x}+1}\,;\ \int_{-\pi}^{\pi}\frac{dx}{p^2-x^2}\,;$$

$$\int_0^{\infty} e^{\omega x}\cos(\omega x+2)\,dx\,;\ \int_0^{\infty}\frac{\sin ux}{x}\,dx$$

вычислялись аналитически. Назначьте эти свойства (ограничения) константам a, b, c, q, p, ω и u, вычислите интегралы.

Задание 4.22. Убедитесь, что следующие определенные интегралы нельзя вычислить аналитически в общем смысле (по формуле Ньютона–Лейбница). Используя опции функции `int`, вычислите эти интегралы в смысле главного значения по критерию Коши. Для каждого интеграла постройте график подынтегральной функции на интервале интегрирования.

$$\int_{\pi/4}^{\pi}\operatorname{tg} x\,dx\,;\ \int_1^4\frac{dx}{(x-2)^3}\,;\ \int_{-2}^{4}\frac{x\,dx}{(x-3)(x^3-1)}\,;\ \int_{1/2}^{3/2}\frac{dx}{\ln x}\,;\ \int_{-1}^{4}\frac{dx}{(2x+1)\ln(2x+1)}\,;$$

$$\int_{-1}^{1}\frac{dx}{e^x-2}\,;\ \int_{-3}^{4}\frac{xdx}{1-e^{-x^2+4}}\,;\ \int_{-\infty}^{\infty}\sin x\,dx\,;\ \int_{-\infty}^{\infty}\frac{e^{-x/3}}{1-e^{-x}}\,dx\,;\ \int_0^{\infty}\frac{e^{-2x}}{3-x}\,dx\,;$$

$$\int_{-\infty}^{\infty}\frac{(x+5)\,dx}{x^2+3\sqrt{2}x+9}\,;\ \int_{-c}^{2c}\frac{dx}{(x-c)^5}\,.$$

Задание 4.23. Следующие интегралы вычислите аналитически и численно, используя точность и метод интегрирования, заданные по умолчанию:

$$\int_0^{2/3}\frac{x\,dx}{x^3-1}\,;\ \int_{-1}^{6}x(x-2)(x-5)\cos 7\pi x\,dx\,;\ \int_0^5 x^{7/3}e^{-x}dx\,;\ \int_0^2 e^{-x^2}dx\,;$$

$$\int_{-3/4}^{17/3}\cos\left(\frac{\pi}{2}x^2\right)dx\,;\ \int_0^{24\pi/9}\sqrt{1-\frac{2}{3}\sin^2 x}\,dx\,;\ \int_0^{25}\frac{3^x-4^x}{2^x+5^x}\,dx\,.$$

Если в результате аналитических вычислений получится рациональное число или константа, для которой в математике введено специальное

обозначение, представьте результат в десятичном формате. Для каждого из этих интегралов сравните значения, вычисленные численно и аналитически.

Задание 4.24. Каждый из следующих интегралов вычислите с помощью квадратурной формулы Ньютона–Котеса, используя 20, 10, 5 и 3 узла, с точностью до четырех значащих цифр:

$$\int_{\pi/2}^{6\pi} \frac{\sin x\,dx}{x}; \quad \int_{0}^{22/3} \sin\left(\frac{\pi}{2}x^2\right)dx; \quad \int_{-1}^{6} x(x-2)(x-5)\cos^2 7\pi x\,dx.$$

Для каждого интеграла подберите минимально необходимое количество итераций, чтобы можно было осуществить численное интегрирование с заданной точностью для каждого указанного количества узлов. Вычислите значения этих интегралов аналитически. Сравните полученные результаты.

Задание 4.25. Использовав отложенные вычисления, запишите, а затем вычислите аналитически следующие интегралы:

$$\iint (x-y)^2 \sin y\,dxdy; \quad \int_{-\pi n}^{\pi n}\int_{a}^{b} (x-y)^2 \sin y\,dxdy; \quad \iint y^6 e^{-x^2}\,dxdy;$$

$$\int_0^1 dy \int_y^1 y^6 e^{-x^2}\,dx; \quad \iint y \sin x^2\,dxdy; \quad \int_0^3 dy \int_{y^2}^9 y \sin x^2\,dx;$$

$$\int_0^{\pi/2}\int_0^{\pi/2} \frac{\sin\alpha \sin y\,dxdy}{\sqrt{1-\sin^2\alpha \sin^2 x \sin^2 y}}; \quad \iiint (x+y^2+z^3)dx\,dy\,dz;$$

$$\int_0^1\int_{-1}^2\int_1^3 (x+y^2+z^3)dx\,dy\,dz; \quad \int_1^2\int_0^2\int_0^5 (ax-b\sin y+z^3)dx\,dy\,dz;$$

$$\int_0^{2\pi}\int_0^{\pi}\int_0^2 e^{-2r} r^6 \sin\theta \cos^2\varphi(1+\sin^2\theta)dr\,d\theta\,d\varphi.$$

Задание 4.26. Запишите выражение $\sum_{k=1}^{n} \frac{x^k}{k!}$. Убедитесь, что это выражение невозможно упростить. Для этого выражения вычислите первую, вторую и третью производные, а также неопределенный интеграл по x.

Получите новое выражение, умножив исходное выражение на x и упростите (если это возможно). Для нового выражения вычислите первую и

вторую производные и неопределенный интеграл по x. Сравните результаты этих вычислений с результатами аналогичных вычислений для выражения $\sum_{k=1}^{n} \frac{x^{k+1}}{k!}$. Для этого удобно вычислить и упростить разность между аналогичными результатами, полученными для обоих выражений.

Вычислите первую и вторую производные, а также неопределенный интеграл по x для выражения $\sum_{k=1}^{n} \frac{x^k}{k}$. Объясните полученные результаты.

Задание 4.27. Объявите функцию $f(x) = 15e^{-x^2/5}\cos\pi x + 13$. Вычислите площадь плоской фигуры, ограниченной на интервале [1, 4] графиком этой функции и осью абсцисс. Результат вычислений представьте в десятичном формате. Постройте график функции и заштрихуйте на нем эту фигуру.

Задание 4.28. Объявите две функции:

$$f_1(x) = \frac{55}{2+(x-2)^2} + \frac{40}{1+(x+4)^2} - 7\frac{5}{6} \text{ и } f_2(x) = (x+2)^2 - 15.$$

Вычислите площадь плоской фигуры, ограниченной графиками этих функций на интервале [–6, 3]. Результат вычислений представьте в десятичном формате. Постройте график двух функций и заштрихуйте на нем эту фигуру.

Задание 4.29. Для плоской фигуры из задания 4.28 рассчитайте координаты центра инерции. Результаты вычислений представьте в десятичном формате. На графике, изображающем обе функции и заштрихованную фигуру, точкой отметьте центр инерции.

Тема 5. Решение задач линейной алгебры

Задание 5.1. Задайте следующие матрицы и векторы:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 3 & 5 & 9 \\ 2 & 6 & 7 \\ 1 & 4 & 8 \end{pmatrix}, \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 8 & 0 \end{pmatrix}, \begin{pmatrix} 9 & 3 & 0 & 0 \\ 5 & 9 & 3 & 0 \\ 0 & 5 & 9 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$\begin{pmatrix} x & 0 & 0 \\ 0 & x-1 & 0 \\ 0 & 0 & x+1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & y^2 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} x^2 y & yx \\ \sin x & \dfrac{1}{x} \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ 10 \end{pmatrix}, \begin{pmatrix} x^2 \\ (x-1)^2 \\ y^3 \\ (y+1)^3 \end{pmatrix},$$

$$\begin{pmatrix} 10 & 1 & -xy & 4 \end{pmatrix}, \begin{pmatrix} x_1 & 1 & 0 & x_2 & x_3 & -1 \end{pmatrix}.$$

Для каждой матрицы (вектора) подберите оптимальный способ создания.

Задание 5.2. Задайте: матрицу размером 5 × 8 с элементами $x^i y^j$ и матрицу размером 3 × 7 с элементами a_{ij} (i – номер строки, j – номер столбца); векторы-строки длиной (размерностью) 5, 7, 9, элементы которых имеют вид: $j \cdot x^j$ (j – номер столбца).

Задание 5.3. Создайте последовательность целых чисел от 1 до 30. Из этой последовательности создайте матрицы следующих размеров: 6 × 5, 5 × 6, 3 × 10, 10 × 3, 2 × 15, 15 × 2, 1 × 30, 30 × 1, 4 × 7.

Задание 5.4. Создайте матрицу A размером 3 × 6, состоящую из случайных целых чисел в интервале от –2 до 5. Преобразуйте полученную матрицу в матрицу с элементами произвольного типа. Присоедините (по вертикали) снизу к этой матрице единичную матрицу размером 6 × 6. Замените пятую строку получившейся матрицы линейной комбинацией первой и пятой строк, умноженных на коэффициенты 2 и 1/2 соответственно. Извлеките (скопируйте) третий столбец, создав из него вектор B. Умножьте полученный вектор на 10 и удалите из него вторую и девятую строки. В полученной ранее матрице A удалите 7-ю и 8-ю строки. К матрице A присоедините (по горизонтали) справа вектор B. В матрице A столбец № 5 замените столбцом из единиц. Определите размер (количество строк и столбцов) преобразованной матрицы A и приведите ее к треугольному виду методом Гаусса.

Задание 5.5. Для следующих матриц

а) $A = \begin{pmatrix} 1 & 3 \\ 0 & 2 \\ 2 & -5 \end{pmatrix}$, $B = \begin{pmatrix} -2 & 4 \\ 1 & 3 \end{pmatrix}$, $C = \begin{pmatrix} 2 & 3 & -2 \\ 3 & -2 & 5 \end{pmatrix}$;

б) $A = \begin{pmatrix} 0 & y & -3 \\ z & 1 & 0 \\ -2 & 0 & x \end{pmatrix}$, $B = \begin{pmatrix} x & y & z \\ x-1 & y-2 & z-3 \\ xy & y^2 & zy \end{pmatrix}$, $C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{pmatrix}$

выполните те из перечисленных операций, которые разрешены правилами линейной алгебры: $A + B$, $2A^{\mathrm{T}} + C$, AB, BA, AC, A^2, $A^{\mathrm{T}}B$, $B^{-1}C$.

Задание 5.6. Аналитически вычислите определители следующих матриц: а) матрицы размером 2×2 с элементами a_{ij}; б) матрицы размером 3×3 с элементами $x^i - \sin ij$; в) матрицы размером 4×4 с элементами $\dfrac{3^i}{4+j} + \cos\dfrac{i}{j}$ (i – номер строки, j – номер столбца). Результат представьте в десятичном формате с точностью до 5 значащих цифр.

Задание 5.7. Для следующих матриц аналитически вычислите определитель, союзную матрицу, обратную матрицу. Кроме того, к каждой матрице присоедините по горизонтали справа единичную матрицу того же размера и приведите полученную матрицу к треугольному виду методом Гаусса и к диагональному виду методом Гаусса–Жордана.

$$\begin{pmatrix} -1 & -1 & 0 & 3 \\ 3 & -2 & 2 & 0 \\ 6 & 1 & 0 & -3 \\ 6 & 0 & -3 & 0 \end{pmatrix}, \begin{pmatrix} 3 & -3 & -1 & -3 & 4 \\ -5 & 5 & -5 & 2 & -4 \\ 4 & 1 & 0 & 1 & 5 \\ -4 & 4 & -2 & -1 & 3 \\ 2 & -5 & 1 & -1 & -4 \end{pmatrix}, \begin{pmatrix} 4 & x+4 & 4 \\ x+2 & 0 & 2 \\ -2 & x-1 & -2 \end{pmatrix}.$$

Задание 5.8. Аналитически вычислите по пять любых миноров $(n-1)$-го и $(n-2)$-го порядка для следующих матриц (n – количество строк и столбцов матрицы):

$$\begin{pmatrix} 2 & -1 & 1 & 0 \\ 0 & 1 & 2 & -1 \\ 3 & 1 & 2 & 3 \\ 3 & 1 & 6 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & \sin z \\ 1 & \sin x & \cos y \\ \sin x & 0 & \cos z \end{pmatrix}.$$

Выведите все вспомогательные матрицы.

Задание 5.9. Аналитически решите следующие системы линейных уравнений в матричном виде:

$$\begin{cases} x_1 + x_2 + x_3 = 3, \\ 2x_1 - 3x_2 + x_3 = 0, \\ 4x_1 + x_2 - 5x_3 = 0, \end{cases} \quad \begin{cases} 2x_1 + 3x_2 - 3x_3 + 4x_4 = -14, \\ 2x_1 + x_2 - x_3 + 2x_4 = -14, \\ 8x_1 + 3x_2 + 2x_3 + 2x_4 = -1, \\ 8x_1 + 5x_2 + x_3 + 5x_4 = -7, \end{cases}$$

$$\begin{cases} x_1 + x_2 - 4x_3 + 5x_4 = 3, \\ x_1 + 2x_2 - 3x_3 + 4x_4 = 4, \\ x_1 + 3x_2 - 2x_3 + 4x_4 = 5, \\ x_1 - x_2 - 6x_3 + 7x_4 = 1, \end{cases} \quad \begin{cases} x_1 - 2x_2 + x_3 - 4x_4 + x_5 = 0, \\ 2x_1 - 4x_2 + x_3 - 5x_4 + 2x_5 = 0, \\ 3x_1 - 2x_2 - x_3 - 4x_4 + 3x_5 = 0. \end{cases}$$

Если система имеет неединственное решение, получите два частных решения. Для каждого полученного решения выполните проверку: убедитесь, что полученные значения действительно являются решениями соответствующей системы линейных уравнений.

Задание 5.10. Аналитически решите матричные уравнения:

$$\begin{pmatrix} -3 & 0 & 1 & 0 \\ 0 & -1 & 1 & -1 \\ 2 & -3 & 3 & 3 \\ 3 & -1 & 0 & 0 \end{pmatrix} \cdot X = \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ -1 & 1 \\ 2 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & a-2 & -1 \\ -2 & 3 & -2 \\ 0 & 2a+2 & -1 \end{pmatrix} \cdot X = \begin{pmatrix} 2 & 2 & 2 \\ -2 & 1 & 1 \\ -2 & -2 & 2 \end{pmatrix}.$$

Убедитесь, что полученные матрицы действительно являются решениями соответствующих уравнений.

Задание 5.11. Для следующих матриц составьте характеристическую матрицу, характеристическое уравнение (полином), аналитически найдите собственные значения и собственные векторы. При необходимости представьте результат в десятичном формате. Если аналитическое решение получить не удалось, то найдите собственные значения и собственные векторы численно. Убедитесь, что каждый из полученных векторов и значений является собственным для соответствующей матрицы.

$$\begin{pmatrix} -3 & -1 \\ 5 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 3 & 0 \\ 0 & -1 & 0 \\ 3 & 8 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 2 \\ -1 & 0 & 2 \end{pmatrix}, \begin{pmatrix} 4 & 0 & 4 & -2 \\ 0 & 4 & 4 & 2 \\ -2 & 4 & -2 & 2 \\ -2 & 2 & 0 & -2 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 & -3 \\ -3 & 2 & 3 & 0 \\ -4 & 2 & 1 & -1 \\ -1 & 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} -2 & -2 & -2 & -2 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 & -2 \\ 0 & 0 & 0 & 0 & -2 \\ 4 & 4 & 0 & 2 & -2 \end{pmatrix}, \begin{pmatrix} x & y \\ 2x & 3y \end{pmatrix}.$$

Задание 5.12. Найдите базис в пространстве, в котором заданы следующие векторы:

$$\begin{pmatrix} 1 & -1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & -2 & 0 \end{pmatrix}, \begin{pmatrix} -3 & 3 & 0 \end{pmatrix}, \begin{pmatrix} 8 & 4 & 1 \end{pmatrix}, \begin{pmatrix} -6 & -6 & -1 \end{pmatrix}.$$

Вычислите модуль каждого вектора. Определите, какие из этих векторов являются ортогональными. Из этих векторов выберите: 1) два вектора так, чтобы их векторное произведение было отлично от нуля; 2) три вектора так, чтобы их смешанное произведение было равно нулю. При каких значениях x и y вектор $\begin{pmatrix} x & y-1 & 2 \end{pmatrix}$ параллелен вектору $\begin{pmatrix} 8 & 4 & 1 \end{pmatrix}$, а при каких – перпендикулярен вектору $\begin{pmatrix} -6 & -6 & -1 \end{pmatrix}$.

Задание 5.13. В декартовой системе координат найдите градиент и лапласиан следующих функций:

$$f_1(x,y) = 10 - \frac{x^2}{2} - y^2; \quad f_2(x,y) = |x|\, e^{-x^2-y^2}; \quad f_3(x,y,z) = 3x^2 + y\sqrt{z};$$

$$f_4(x,y,z) = \sin x + yz^2; \quad f_5(x,y,z) = x^2 + e^y \cos z.$$

Для каждого из полученных градиентов вычислите (и представьте в десятичном формате с точностью до шести значащих цифр) его модуль в точке с координатами $x = 2$, $y = 1$, $z = -2$.

Задание 5.14. В декартовой системе координат рассчитайте дивергенцию и ротор следующих векторных функций:

$$\vec{F}_1(x,y,z) = \begin{pmatrix} x^2 & y^2 & z^2 \end{pmatrix}; \quad \vec{F}_2(x,y,z) = \begin{pmatrix} x^2yz & -xy^3 & y\sin z \end{pmatrix};$$

$$\vec{F}_3(x,y,z) = \begin{pmatrix} xy & y-z & xe^{-yz} \end{pmatrix}.$$

Если векторное поле, описываемое соответствующей векторной функцией, не является потенциальным, определите (и представьте в десятичном формате с точностью до четырех значащих цифр) угол между вектором $\begin{pmatrix} 3 & 1 & -1 \end{pmatrix}$ и направлением ротора этого поля в точке с координатами $x = 2$, $y = -1$, $z = -2$.

Тема 6. Решение обыкновенных дифференциальных уравнений и их систем

Задание 6.1. Аналитически найдите общее решение для каждого из следующих обыкновенных дифференциальных уравнений. Определите, для получения решения или упрощения каких из этих уравнений имеет смысл использовать опцию `IgnoreAnalyticConstraints`. Выполните проверку, действительно ли полученное в процессе решения выражение является решением соответствующего дифференциального уравнения.

$$2x \cdot y(x)\frac{d\,y(x)}{dx} = y^2(x) - 4x^2; \qquad x\frac{d\,y(x)}{dx} - y(x) = x \cdot \mathrm{tg}\frac{y(x)}{x};$$

$$x \cdot y(x) + y^2(x) = 2(x^2 + x \cdot y(x))\frac{dy(x)}{dx}; \quad (1+e^{ax})y^2(x)\frac{dy(x)}{dx} = e^{ax};$$

$$\frac{d^2y(x)}{dx^2} + a\frac{dy(x)}{dx} = e^x; \qquad \frac{d^3y(x)}{dx^3} + a\frac{d^2y(x)}{dx^2} = bx.$$

Задание 6.2. Аналитически найдите частное решение для каждого из следующих обыкновенных дифференциальных уравнений. При необходимости используйте опцию `IgnoreAnalyticConstraints` для получения решения и (или) его упрощения. Выполните проверку, действительно ли полученное в процессе решения выражение является решением соответствующего дифференциального уравнения.

$x^2 - y^2(x) + 2xy(x)\dfrac{dy(x)}{dx} = 0$ при начальном условии $y(1) = 2$;

$t\dfrac{dx(t)}{dt} - x(t) = 2t \cdot \mathrm{tg}\dfrac{x(t)}{t}$ при начальном условии $x(4) = \pi$;

$\dfrac{dy(t)}{dt} - y^2(t) + 2y(t) = 0$ при начальном условии $y(0) = -1/4$;

$\dfrac{y(t)}{t}\dfrac{dy(t)}{dt} = 2^{y^2(t)-t^2}$ при начальном условии $y(1) = 1$;

$\dfrac{d^2y(t)}{dt^2} - 4\dfrac{dy(t)}{dt} + 3y(t) = 0$ при начальных условиях $y(0) = 0$, $y'(0) = 1$.

Задание 6.3. Проверьте, является ли каждое из следующих обыкновенных дифференциальных уравнений и однородным, и линейным одновременно. Если уравнение является и линейным, и однородным, получите для него фундаментальную систему решений, а затем (без использования `ode::solve`) общее решение и одно частное (отличное от тривиального) решение, задав константы интегрирования по своему усмотрению. Выполните проверку, действительно ли частное решение удовлетворяет соответствующему дифференциальному уравнению.

$y''(x) - 3y'(x) + 2y(x) = 0$, $\quad y''(x) + 6y'(x) + 9y(x) = 0$,

$y''(x) + 5y'(x) = 0$, $\quad y(x)y''(x) + (y'(x))^2 = 0$,

$x^2y''(x) - 3xy'(x) + 3y(x) = 0$, $\quad y''(x) + 25y(x) = 0$,

$y''(x) - 3y'(x) - x^2 + x = 0$, $\quad y'''(x) - 6y''(x) + 11y'(x) - 6y(x) = 0$,

$y^{(4)}(x) + 4y''(x) + 4y(x) = 0$, $\quad y^{(5)}(x) + 3y^{(4)}(x) + 3y'''(x) + y''(x) = 0$.

Если уравнение не является линейным и (или) однородным, с помощью функции `ode::solve` получите для него общее решение и убедитесь, что оно не может быть записано в виде фундаментальной системы.

Задание 6.4. Аналитически найдите решение для каждого из следующих обыкновенных дифференциальных уравнений. При необходимости используйте опцию `IgnoreAnalyticConstraints` для получения решения и (или) его упрощения. Постройте график решения каждого из этих уравнений на интервале $t \in [0, 3]$ и отметьте точку, в которой задано начальное условие.

$x'(t) = 4t^3$ при начальном условии $x(0) = 1$;

$t \cdot y'(t) - y(t) = t \cdot \mathrm{tg}\dfrac{y(t)}{t}$ при начальном условии $y(1) = 1$.

Задание 6.5. Аналитически решите дифференциальное уравнение $y''(x) + 4y(x) = 0$ при начальных условиях $y(0) = 0, y'(0) = 1$. Постройте два графика. На первом изобразите две кривые: $y(x)$ и $y'(x)$. На втором – фазовую кривую $y'(y)$ в плоскости $(y(x), y'(x))$. Для построения кривых используйте разные цвета. На каждой из кривых отметьте точку того же цвета, в которой заданы начальные условия. С помощью легенды подпишите кривые.

Задание 6.6. Аналитически найдите общее решение для каждой из следующих систем обыкновенных дифференциальных уравнений. Для упрощения полученных решений раскройте в них скобки. Выполните проверку, действительно ли полученные в процессе решения выражения является решением соответствующей системы дифференциальных уравнений.

$$\begin{cases} x'(t) = 2x(t) + y(t); \\ y'(t) = 5t; \end{cases} \quad \begin{cases} x'(t) = a^2 y(t); \\ y'(t) = b^2 x(t); \end{cases} \quad \begin{cases} x'(t) = a^2 y(t) + abc \cdot t; \\ y''(t) = c^2 x'(t) \end{cases}$$

(a, b и c – незаданные, не зависящие от t символьные константы).

Задание 6.7. Аналитически найдите частное решение для каждой из следующих систем обыкновенных дифференциальных уравнений. Выполните проверку, действительно ли полученные в процессе решения выражения является решением соответствующей системы дифференциальных уравнений.

$\begin{cases} x'(t) = 3 - 2y(t); \\ y'(t) = 2x(t) - 2t; \end{cases}$ при начальных условиях $x(0) = 1, y(0) = 0$;

$$\begin{cases} x'(t) = x(t) - y(t) + t; \\ y'(t) = 2x(t) + y(t) - t^2; \end{cases}$$ при начальных условиях $x(0) = 0$, $y(0) = 0$;

$$\begin{cases} y''(x) = 2z(x); \\ z''(x) + 2y(x) = 0; \end{cases}$$ при начальных условиях $\begin{matrix} y(0) = 1, y'(0) = 0, \\ z(0) = 2, z'(0) = 0. \end{matrix}$

Задание 6.8. Аналитически найдите решение для каждой из следующих систем обыкновенных дифференциальных уравнений. Для каждого решения постройте по три графика. На первом изобразите искомые функции (например, $x(t)$ и $y(t)$ для первой системы). На втором – фазовые кривые для всех искомых функций (например, $x'(x)$ и $y'(y)$ для первой системы). На третьем – все зависимости одной искомой функции от другой (например, $y(x)$ для первой системы). Для построения кривых используйте разные цвета. На каждой их кривых отметьте точку того же цвета, в которой заданы начальные условия. С помощью легенды подпишите кривые. Интервал для аргумента искомых функций подберите так, чтобы вид каждой кривой был наиболее выразительным.

$$\begin{cases} x'(t) = 5y(t); \\ y'(t) = 3x(t) + 2t; \end{cases}$$ при начальных условиях $x(0) = 0$, $y(0) = 1$;

$$\begin{cases} y'(x) + 5y(x) + z(x) = e^x; \\ z'(x) = y(x) - 3z(x) + e^{2x}; \end{cases}$$ при начальных условиях $y(0) = 1$, $z(0) = 2$;

$$\begin{cases} x'(t) = x(t) - y(t); \\ y'(t) = x(t) + z(t); \\ z'(t) = y(t) - x(t); \end{cases}$$ при начальных условиях $\begin{matrix} x(0) = 1, \\ y(0) = 2, \\ z(0) = -1. \end{matrix}$

Задание 6.9. *Это задание рекомендуется выполнять в отдельном* mn-*файле.*

Аналитически решите дифференциальное уравнение $y''(x) - \frac{1}{15}y'(x) = 50\cos x - \frac{10}{3}\sin x$ при начальных условиях $y(0) = 50$, $y'(0) = -1$. На интервале [0, 40] постройте график $y(x)$ полученного решения.

Сравните полученное аналитическое решение с численным решением того же дифференциального уравнения. Для этого найдите численное решение этого уравнения (вычислите $y(x)$) двумя методами: 1) методом Рунге–Кутты четвертого порядка, используя постоянный (фиксированный) шаг интегрирования, равный 1/4 (задается с помощью опций), в каждой точке интервала $x \in [0, 40]$ с шагом 1/2; 2) методом Рунге–Кутты четвертого

порядка, но с автоматическим подбором шага интегрирования (по умолчанию) в каждой точке интервала $x \in [0, 40]$ с шагом 1/2–1/100. При необходимости при выполнении численных расчетов можно воспользоваться опцией `RememberLast`. Так как MuPAD помнит результаты всех выполненных ранее вычислений, не используйте для обозначения решений, полученных разными методами, одинаковые переменные. Перед применением второго метода очистите использованные по умолчанию переменные.

Оба численных решения изобразите на графике аналитического решения точками двух разных цветов, не соединяя их между собой. С помощью легенды подпишите полученные кривые.

После этого для решения, полученного каждым из численных методов, определите, насколько численные решения в точках $x = 6$, $x = 10$ и $x = 20$ отличаются от аналитического решения (вычислите разницу).

Задание 6.10. Для каждого из следующих дифференциальных уравнений на одном графике постройте по три кривые, соответствующие численным решениям одного и того же дифференциального уравнения при разных начальных условиях. Для построения графиков выберите интервал [0, 10]. Для построения кривых используйте разные цвета. Каждая кривая должна содержать 20–21 точку, в каждой из которых численно рассчитывается значение искомой функции. На каждой кривой точкой того же цвета отметьте значение искомой функции, соответствующее значению ее аргумента t равного 5. С помощью легенды укажите на графике начальные условия каждого решения.

$$x'(t) = \cos^2 x(t) + e^{-t}$$

при начальных условиях $x(0) = 1$, $x(0) = 2$, $x(0) = 3$;

$$y'(t) = y^2(t)e^{-y(t)} - \frac{1}{45}t^3$$

при начальных условиях $y(0) = 1$, $y(0) = 2$, $y(0) = 3$.

Задание 6.11. Для численного решения дифференциального уравнения $x''(t) = \frac{1}{2}\sin x(t)$ задайте начальные условия в точке $t = 0$ и на интервале [0, 15] постройте следующие графики: 1) $x(t)$ и $x'(t)$; 2) фазовую кривую $x'(x)$ – график решения в плоскости $(x(t), x'(t))$; 3) фазовые кривые $x'(x)$, соответствующие трем различным начальным условиям. Для построения графиков выберите интервал так, чтобы вид каждой кривой был наиболее выразительным. Для построения кривых используйте разные цвета. С помощью легенды подпишите кривые. Подпишите оси координат.

Точки, в которых вычислены значения искомой функции и ее производной, соединяйте *прямыми*. Изменяя количество этих точек, добейтесь, чтобы полученные кривые были гладкими.

Задание 6.12. Для каждой из следующих систем дифференциальных уравнений постройте по три графика, изображающих ее численное решение: 1) графики зависимости каждой из искомых функций от ее аргумента; 2) график зависимости одной искомой функции от другой; 3) три графика зависимости одной искомой функции от другой при разных начальных условиях. Для построения кривых используйте разные цвета. На каждой из кривых изобразите точку соответствующего цвета, в которой заданы начальные условия. Для построения графиков выберите интервал так, чтобы вид каждой кривой был наиболее выразительным. Подпишите оси координат. С помощью легенды подпишите кривые. Точки, в которых вычисляются значения искомых функций, соединяйте *прямыми*. Изменяя количество этих точек, добейтесь, чтобы полученные кривые были гладкими.

$$\begin{cases} x'(t) = p(t); \\ p'(t) = \sin x(t); \end{cases}$$ при начальных условиях $x(0) = 1$, $p(0) = 0$;

$$\begin{cases} x'(t) = 2y(t); \\ y'(t) = \sin x(t) - 1{,}1x'(t); \end{cases}$$ при начальных условиях $x(0) = 0$, $y(0) = 1$.

Задание 6.13. Для численного решения системы обыкновенных дифференциальных уравнений:

$$\begin{cases} x'(t) = y(t) - z(t)\sin t; \\ y'(t) = z(t) - x(t)\cos t; \\ z'(t) = x^2(t) - y^2(t); \end{cases}$$ при начальных условиях $\begin{matrix} x(0) = 1, \\ y(0) = 0, \\ z(0) = 2 \end{matrix}$

на интервале [–2, 4] постройте следующие графики: 1) $x(t)$, $y(t)$, $z(t)$; 2) $y(x)$, $z(x)$, $z(y)$. Для построения кривых используйте разные цвета. На каждой из кривых отметьте точку соответствующего цвета, в которой заданы начальные условия. С помощью легенды подпишите кривые.

Тема 7. Интерполяция и полиномиальное представление функций

Задание 7.1. Задайте два списка: список узлов: –11/4, –1/2, 7/4, 4 и список значений неизвестной функции в этих узлах: $(11\sqrt{2} - 3)/4$, 5/2, $(15 - 7\sqrt{2})/4$, 6. По этим данным постройте интерполяцию Лагранжа.

Определите степень полученного полинома. Сравните полученный интерполяционный полином с функцией $f(x) = x + 2 + 2x\sin(x\pi)$. Для этого на интервале, на котором расположены узлы, постройте одновременно графики функции $f(x)$ и полинома, отметьте узловые точки; вычислите и сравните значения функции $f(x)$ и полинома в узлах. Кривые на графиках изображайте точками, не соединяя их между собой.

Задание 7.2. Задайте пару (два) списков:

список узлов: –3, –5/2, –2, –3/2, –1, –1/2, 0, 1/2, 1, 3/2, 2, 5/2, 3, 7/2, 4, 9/2, 5, 11/2, 6, 13/2, 7;

список значений неизвестной функции в этих узлах:

$14,\ (20\sqrt{2}+1)/4,\ -3,\ -(20\sqrt{2}+23)/4,\ -18,\ -(20\sqrt{2}+39)/4,\ -11,$

$(20\sqrt{2}-47)/4,\ -2,\ (20\sqrt{2}-47)/4,\ -11,\ -(20\sqrt{2}+39)/4,\ -18,$

$-(20\sqrt{2}+23)/4,\ -3,\ (20\sqrt{2}+1)/4,\ 14,\ (20\sqrt{2}+33)/4,\ 13,$

$-(20\sqrt{2}-73)/4,\ 14.$

Из этой пары списков создайте новую пару (два) списков, первый из которых содержит 11 узлов (включая первый и последний), второй – соответствующие этим узлам значения неизвестной функции. Далее аналогичным образом составьте еще четыре пары списков, содержащих 6, 5, 3, 2 узла и соответствующих им значений неизвестной функции. Обратите внимание, первый и последний элементы исходной пары списков должны входить *во все* созданные пары списков. Для каждой пары списков, включая исходную, постройте интерполяцию Лагранжа и определите степень полученного полинома. Сравните полученные интерполяционные полиномы с функцией $f(x) = (x-1)^2 - 12 + 10\sin\left(\frac{x\pi}{2}\right)$. Для этого на интервале, на котором расположены узлы, постройте одновременно графики функции $f(x)$ и всех полиномов; отметьте узловые точки. Интерполяционные кривые на графиках изображайте точками, не соединяя их между собой, а функцию $f(x)$ – сплошной линией. С помощью легенды подпишите кривые. Для какого количества узлов интерполяционный полином меньше всего отличается от функции $f(x)$?

Задание 7.3. Задайте два списка – список узлов: 7/4, 5/2, 13/4, 4, 19/4, 11/2, список значений неизвестной функции в этих узлах: $5\sqrt{2}e^{-7/12}$, 0, $-5\sqrt{2}e^{-13/12}$, $10e^{-4/3}$, $-5\sqrt{2}e^{-19/12}$, 0. По этим данным постройте интерполяцию кубическими сплайнами. Получите аналитический вид сплайна

и его первой производной на интервале между вторым и третьим узлами. Сравните полученную интерполяционную кривую с функцией $f(x) = 10e^{-x/3}\cos((x-2)\pi)$. Для этого на интервале, на котором расположены узлы, постройте одновременно графики функции $f(x)$ и интерполяционную кривую, отметьте узловые точки; вычислите и сравните значения функции $f(x)$ и интерполяционной кривой в узлах. Кривые на графиках изображайте точками, не соединяя их между собой.

Задание 7.4. Задайте пару (два) списков:

список узлов: –6, –11/2, –5, –9/2, –4, –7/2, –3, –5/2, –2, –3/2, –1, –1/2, 0, 1/2, 1, 3/2, 2, 5/2, 3, 7/2, 4, 9/2, 5, 11/2, 6;

список значений неизвестной функции в этих узлах: $-25e^{-49}-16$, $-25e^{-169/4}-41/4$, $-25e^{-36}-5$, $-25e^{-121/4}-1/4$, $4-25e^{-25}$, $31/4-25e^{-81/4}$, $11-25e^{-16}$, $55/4-25e^{-49/4}$, $16-25e^{-9}$, $71/4-25e^{-25/4}$, $19-25e^{-4}$, $79/4-25e^{-9/4}$, $20-25e^{-1}$, $79/4-25e^{-1/4}$, –6, $71/4-25e^{-1/4}$, $16-25e^{-1}$, $55/4-25e^{-9/4}$, $11-25e^{-4}$, $31/4-25e^{-25/4}$, $4-25e^{-9}$, $-25e^{-49/4}-1/4$, $-25e^{-16}-5$, $-25e^{-81/4}-41/4$, $-25e^{-25}-16$.

Из этой пары списков создайте новую пару (два) списков, первый из которых содержит 5 узлов, второй – соответствующие этим узлам значения неизвестной функции. Далее аналогичным образом составьте еще три пары списков, содержащих 10, 15, 20 узлов и соответствующих им значений неизвестной функции. Проанализируйте составленные списки. Если в каком-нибудь из полученных списков узлов отсутствует первый и (или) последний узел, добавьте его (их) в этот список, а также соответствующее ему (им) значение функции – в парный список значений. Для каждой пары списков, включая исходную, постройте кубический сплайн. Сравните полученные сплайны с функцией $f(x) = -x^2 + 20 - 25e^{-(x-1)^2}$. Для этого на интервале, на котором расположены узлы, постройте одновременно графики функции $f(x)$ и всех сплайнов; отметьте узловые точки. Сплайны на графике изображайте точками, не соединяя их между собой, функцию $f(x)$ – сплошной линией. С помощью легенды подпишите кривые. Для какого количества узлов соответствующий кубический сплайн меньше всего отличается от функции $f(x)$?

Задание 7.5. Объявите функцию $g(x) = \dfrac{30(x+1)}{(x+1)^2 + 3/2} - 2x$. Для этой функции получите представление в виде полинома 3-й степени в форме Бернштейна на интервале [–6, 5]. Сравните полученное полиномиальное

представление с исходной функцией. Для этого постройте эти функции на одном графике на указанном интервале. Полиномиальное представление на графике изображайте точками, не соединяя их между собой, а функцию $g(x)$ – сплошной линией. Рассчитайте и отметьте на том же графике узловые точки, по которым построено полученное представление. Для той же функции $g(x)$ получите представление Бернштейна в виде полиномов 5-й, 10-й, 20-й степеней. Изобразите все полученные представления на одном графике и сравните их с исходной функцией $g(x)$. С помощью легенды подпишите кривые. Полиномиальное представление какой степени меньше всего отличается от исходной функции?

Тема 8. Программирование в MuPAD

Задание 8.1. С помощью матрицы A, составленной из коэффициентов и свободных членов системы линейных уравнений, и матрицы-столбца X, содержащей идентификаторы неизвестных, запишите систему линейных уравнений в явном виде. Каждое уравнение выведите в отдельной строке. Решите полученную систему уравнений двумя способами, используя матричную и алгебраическую формы записи этой системы.

$$A = \begin{pmatrix} 1 & 1 & -4 & 5 & 3 \\ 1 & 2 & -3 & 4 & 4 \\ 1 & 3 & -2 & 4 & 5 \\ 1 & -1 & -6 & 7 & 1 \end{pmatrix} \text{ и } X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ z \end{pmatrix}.$$

Задание 8.2. Напишите функцию, которая вычисляет минор $(n-1)$-го порядка квадратной матрицы размером $n \times n$ и выводит вспомогательную матрицу. С помощью этой функции вычислите ВСЕ миноры $(n-1)$-го порядка следующих матриц:

$$\begin{pmatrix} x & y \\ 2x & 3y \end{pmatrix}, \begin{pmatrix} 1 & 3 & 0 \\ 0 & -1 & 0 \\ 3 & 8 & 1 \end{pmatrix}, \begin{pmatrix} 2 & 3 & -3 & 4 \\ 2 & 1 & -1 & 2 \\ 8 & 3 & 2 & 2 \\ 8 & 5 & 1 & 5 \end{pmatrix}.$$

Задание 8.3. Для следующего набора векторов составьте базис:

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ -1 \\ 4 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \\ 14 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \\ 6 \end{pmatrix}.$$

Используя циклы, вычислите векторное произведение каждого вектора базиса с каждым вектором исходного набора и углы между ними в градусах. Для каждой пары перемножаемых векторов выведите эти векторы, угол между ними (в десятичном формате с точностью до одного десятичного знака после запятой) и результирующий вектор.

Задание 8.4. Используя циклы, вычислите все скалярные произведения следующих векторов друг с другом и углы между ними в градусах (по одному разу):

$$\begin{pmatrix}1\\3\\2\\3\end{pmatrix},\begin{pmatrix}1\\4\\-1\\3\end{pmatrix},\begin{pmatrix}2\\5\\-1\\4\end{pmatrix},\begin{pmatrix}5\\1\\2\\14\end{pmatrix},\begin{pmatrix}2\\-1\\1\\0\end{pmatrix},\begin{pmatrix}0\\-1\\2\\-1\end{pmatrix},\begin{pmatrix}3\\-1\\2\\3\end{pmatrix},\begin{pmatrix}3\\1\\6\\1\end{pmatrix}.$$

Для каждой пары перемножаемых векторов выведите эти векторы, угол между ними (в десятичном формате с точностью до одного десятичного знака после запятой) и их скалярное произведение.

Задание 8.5. Напишите функцию, которая проверяет, является ли переменная `x` корнем уравнения `ur`. С помощью этой функции проверьте, какие из элементов списка

$$\left[-\frac{11}{3},3,9,-\frac{\sqrt{21}}{7}i,\frac{\sqrt{21}}{7}i,6+2i,-\frac{4\pi}{3},\frac{\pi}{4},\frac{\pi}{3},\frac{7\pi}{6}\right]$$

являются корнями уравнения $(84x^4 - 448x^3 - 2736x^2 - 192x - 1188)\sin^2 x = 63x^4 - 336x^3 - 2052x^2 - 144x - 891$. По результатам проверки сформируйте два списка: в один поместите значения, являющиеся корнями данного уравнения, а в другой – не являющиеся.

Задание 8.6. Определите, какие независимые символьные переменные содержатся в выражении

$$x^4+(a-y)\cos x-e^{-a(x+y)}+e^{b+x}\sin^2(y+a)-\sqrt{a^2+b^2}\ .$$

Используя циклы, вычислите для этого выражения все частные производные первого и второго порядков по одному разу. Упростите полученные выражения.

Задание 8.7. Заполните одномерный массив случайными вещественными числами, подчиняющимися равномерному распределению на интервале $[a, b]$. Длину массива и границы интервала введите с клавиатуры. Вычислите среднее арифметическое значение элементов полученного массива. Сохраните полученное значение в текстовом файле. Повторите

заполнение массива и вычисление среднего значения 7 раз так, чтобы в файл были записаны все вычисленные средние значения.

Задание 8.8. Напишите функцию, которая для некоторого выражения выделяет все входящие в него независимые переменные, находит разложение этого выражения в ряд вблизи нуля по первой независимой переменной, а затем, суммируя первый, третий и пятый члены ряда, составляет новое выражение. Протестируйте полученную функцию, подставив в нее в качестве аргумента выражение e^{-x}. Сохраните полученную функцию в текстовом файле. Создайте новый mn-файл, загрузите в него функцию из текстового файла и примените ее к следующим выражениям: $\sin x$, $\cos(x + a)$, $\cos y + \cos x$, $\ln(x + 1)\cdot\cos x$.

Задание 8.9. Из текстового файла tochki.txt прочитайте координаты точек и постройте по ним график. Сравните полученный график с графиком функции $f(x) = 10e^{-x/10}\cos(x-1)$ на интервале [1, 25].

Задание 8.10. Измерьте время, затраченное программой на решение системы линейных уравнений численно и аналитически.

$$\begin{cases} x_1 + x_2 - 4x_3 + 5x_4 = 3, \\ x_1 + 2x_2 - 3x_3 + 4x_4 = 4, \\ x_1 + 3x_2 - 2x_3 + 4x_4 = 5, \\ x_1 - x_2 - 6x_3 + 7x_4 = 1. \end{cases}$$

Задание 8.11. Аналитически найдите общее решение дифференциального уравнения $y''(x) + y(x) = 4\cos x$. Получите множество, состоящее и четырех частных решений этого уравнения, задавая в качестве констант интегрирования целые числа из интервала [–2, 5] так, чтобы сумма констант интегрирования в каждом решении была нечетной. Для каждого полученного частного решения определите, каким начальным условиям в точке $x = \pi/4$ оно соответствует (т.е. вычислите значение искомой функции $y(x)$ и ее производной $y'(x)$ в этой точке).

Тема 9. Символьные вычисления в MatLab

Задания из этой темы выполняйте в m-файлах, используя встроенный редактор MatLab. Каждую программу начинайте с очистки памяти и командного окна с помощью команд `clear all` и `clc`.

Задание 9.1. Запишите символьное выражение: $3x + x^2 - 4x - 5x^2 + x - 8 + 6x$. Приведите подобные.

Задание 9.2. Объявите символьную функцию $f_1(x) = x^2 - 10$. Вычислите значения функции $f_1(x)$ при $x = 1$; $x = -3{,}2$; $x = a - 2$; $x = z$. Присвойте переменной y_1 значение $f_1(x - 6)$. Постройте график функции $f_1(x - 4)$ на интервале [–8, 4].

Задание 9.3. Запишите символьное выражение $x^5 - 4x^4 - 20x^3 + 72x^2 + 51x - 180$. Разложите его на множители, а затем в полученном выражении раскройте скобки.

Задание 9.4. Запишите уравнение $3x^5 - 16x^4 - 102x^3 + 22x^2 + 67x - 198 = 0$. Разложите уравнение на множители. Аналитически найдите все корни уравнения. Получите числовые значения всех корней в десятичном формате. Для каждого корня выполните проверку, является ли данное значение решением уравнения. Найдите все вещественные корни уравнения.

Задание 9.5. Численно с точностью до 5 значащих цифр найдите все вещественные решения следующих уравнений:

$$\sin x = \ln 12x; \quad \frac{x^2}{20} - 10x = 15\cos(x+15); \quad \frac{5\sin x}{x} - \frac{x^2}{200} = 0;$$

$$5^x - 6x = 7; \qquad x^2 - 2 = \ln(x+5); \qquad x^5 - 4x^3 + 3x^2 + 7x - 1 = 0.$$

Убедитесь в полноте полученных решений с помощью графиков соответствующих функций.

Задание 9.6. Запишите символьное выражение $\lim\limits_{n\to\infty}\left(\frac{1}{n^2}\sum\limits_{k=1}^{n} k\right)$ и вычислите его значение. Если в результате вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Эти же действия выполните для следующих выражений:

$$\lim_{n\to\infty}\left(\frac{1}{n^2}\sum_{k=1}^{n} k^2\right); \ \lim_{n\to\infty}\left(-\ln n + \sum_{k=1}^{n}\frac{1}{k}\right); \ \lim_{x\to 2}\left(\sum_{k=0}^{\infty}\frac{x^k(k+1)}{k!}\right); \ \lim_{x\to\frac{\pi}{2}}\left(\sum_{k=1}^{\infty}\frac{\sin(kx)}{k^2}\right);$$

$$\lim_{x\to 0}\left(\frac{1}{x}\sum_{k=0}^{\infty}\frac{(-1)^k\left((ax)^{2k+1}+(bx)^{2k+1}\right)}{(2k+1)!}\right); \ \lim_{x\to 0}\left(\prod_{k=1}^{\infty}\left(1-\frac{4}{3}\sin^2\frac{x}{3^k}\right)\right).$$

Задание 9.7. Определите, какие независимые символьные переменные содержатся в выражении

$$x^4 + (a-y)\cos x - e^{-a(x+y)} + e^{b+x}\sin^2(y+a) - \sqrt{a^2+b^2}\ .$$

Используя циклы, вычислите для этого выражения все частные производные первого и второго порядков по одному разу. Упростите полученные выражения.

Задание 9.8. Объявите функцию $f(x)=15e^{-x^2/3}\cos(2x+1)-2x$. Найдите ее точки перегиба, отметите их на графике. Подберите такой интервал вблизи начала координат, чтобы функция имела на нем 10 точек перегиба.

Задание 9.9. Следующие интегралы вычислите аналитически и численно:

$$\int_0^{2/3}\frac{x\,dx}{x^3-1};\ \int_{-1}^{6}x(x-2)(x-5)\cos 7\pi x\,dx;\ \int_0^5 x^{7/3}e^{-x}dx;\ \int_0^2 e^{-x^2}dx;$$

$$\int_{-3/4}^{17/3}\cos\left(\frac{\pi}{2}x^2\right)dx;\ \int_0^{24\pi/9}\sqrt{1-\frac{2}{3}\sin^2 x}\,dx;\ \int_0^{25}\frac{3^x-4^x}{2^x+5^x}dx.$$

Если в результате аналитических вычислений получится рациональное число или константа, для которой в математике введено специальное обозначение, представьте результат в десятичном формате. Для каждого из этих интегралов сравните значения, полученные численно и аналитически.

Задание 9.10. Объявите две функции:

$$f_1(x)=\frac{55}{2+(x-2)^2}+\frac{40}{1+(x+4)^2}-7\frac{5}{6}\ \text{ и }\ f_2(x)=(x+2)^2-15.$$

Вычислите площадь плоской фигуры, ограниченной графиками этих функций на интервале [–6, 3]. Постройте графики двух функций на этом интервале. Результат вычислений представьте в десятичном формате.

Задание 9.11. Вычислите ВСЕ миноры (n – 1)-го и (n – 2)-го порядка для следующих матриц (n – количество строк и столбцов матрицы):

$$\begin{pmatrix}2&-1&1&0\\0&1&2&-1\\3&1&2&3\\3&1&6&1\end{pmatrix},\ \begin{pmatrix}0&1&\sin z\\1&\sin x&\cos y\\\sin x&0&\cos z\end{pmatrix}.$$

Для этого составьте вспомогательную функцию и используйте операторы цикла. Выведите все вспомогательные матрицы.

Задание 9.12. Решите следующие системы линейных уравнений в матричном виде:

$$\begin{cases}x_1+x_2+x_3=3,\\2x_1-3x_2+x_3=0,\\4x_1+x_2-5x_3=0,\end{cases}\qquad\begin{cases}2x_1+3x_2-3x_3+4x_4=-14,\\2x_1+x_2-x_3+2x_4=-14,\\8x_1+3x_2+2x_3+2x_4=-1,\\8x_1+5x_2+x_3+5x_4=-7,\end{cases}$$

$$\begin{cases} x_1 - 2x_2 + x_3 - 4x_4 + x_5 = 0, \\ 2x_1 - 4x_2 + x_3 - 5x_4 + 2x_5 = 0, \\ 3x_1 - 2x_2 - x_3 - 4x_4 + 3x_5 = 0. \end{cases}$$

Если система имеет неединственное решение, получите два частных решения. Для каждого полученного решения выполните проверку: убедитесь, что полученные значения действительно являются решениями соответствующей системы линейных уравнений.

Задание 9.13. В декартовой системе координат найдите градиент и лапласиан следующих функций. Для каждого из полученных градиентов вычислите и представьте в десятичном формате с точностью до шести значащих цифр его модуль в точке с координатами $x = 2$, $y = 1$, $z = -2$.

$$f_1(x,y) = 10 - \frac{x^2}{2} - y^2;\ f_2(x,y) = |x|e^{-x^2-y^2};\ f_3(x,y,z) = 3x^2 + y\sqrt{z};$$

$$f_4(x,y,z) = \sin x + yz^2;\ f_5(x,y,z) = x^2 + e^y \cos z.$$

Задание 9.14. Аналитически найдите общее решение дифференциального уравнения $y''(x) + y(x) = 4\cos x$. Получите массив, состоящий из четырех частных решений этого уравнения, задавая в качестве констант интегрирования целые числа из интервала [–2, 5] так, чтобы сумма констант интегрирования в каждом решении была нечетной. Для каждого полученного частного решения определите, каким начальным условиям в точке $x = \pi/4$ оно соответствует (т.е. вычислите значение искомой функции $y(x)$ и ее производной $y'(x)$ в этой точке).

Задание 9.15. Аналитически найдите частное решение для каждой из следующих систем обыкновенных дифференциальных уравнений. Выполните проверку, действительно ли полученные в процессе решения выражения является решением соответствующей системы дифференциальных уравнений.

$$\begin{cases} x'(t) = x(t) - y(t) + t; \\ y'(t) = 2x(t) + y(t) - t^2; \end{cases}$$ при начальных условиях $x(0) = 0$, $y(0) = 0$;

$$\begin{cases} y''(x) = 2z(x); \\ z''(x) + 2y(x) = 0; \end{cases}$$ при начальных условиях $\begin{matrix} y(0) = 1, y'(0) = 0, \\ z(0) = 2, z'(0) = 0. \end{matrix}$

Задание 9.16. Аналитически найдите решение для каждой из следующих систем обыкновенных дифференциальных уравнений. Для каждого

решения постройте по три графика. На первом изобразите искомые функции (например, $x(t)$ и $y(t)$ для первой системы). На втором – фазовые кривые для всех искомых функций (например, $x'(x)$ и $y'(y)$ для первой системы). На третьем – все зависимости одной искомой функции от другой (например, $y(x)$ для первой системы). Для построения кривых используйте разные цвета. На каждой из кривых отметьте точку того же цвета, в которой заданы начальные условия. С помощью легенды подпишите кривые. Интервал для аргумента искомых функций подберите так, чтобы вид каждой кривой был наиболее выразительным.

$$\begin{cases} y'(x) + 5y(x) + z(x) = e^x; \\ z'(x) = y(x) - 3z(x) + e^{2x}; \end{cases}$$ при начальных условиях $y(0) = 1$, $z(0) = 2$;

$$\begin{cases} x'(t) = x(t) - y(t); \\ y'(t) = x(t) + z(t); \\ z'(t) = y(t) - x(t); \end{cases}$$ при начальных условиях $x(0) = 1$, $y(0) = 2$, $z(0) = -1$.

Задание 9.17. Для следующего дифференциального уравнения на одном графике постройте по три кривые, соответствующие численным решениям этого дифференциального уравнения при разных начальных условиях. Для построения графиков выберите интервал [0, 10]. Для построения кривых используйте разные цвета. На каждой из кривых отметьте точку того же цвета, в которой заданы начальные условия. С помощью легенды укажите на графике начальные условия каждого решения.

$$x'(t) = \cos^2 x(t) + e^{-t}$$

при начальных условиях $x(0) = 1$, $x(0) = 2$, $x(0) = 3$.

Задание 9.18. Задайте два массива – массив узлов: 7/4, 5/2, 13/4, 4, 19/4, 11/2; массив значений неизвестной функции в этих узлах: $5\sqrt{2}e^{-7/12}$, 0, $-5\sqrt{2}e^{-13/12}$, $10e^{-4/3}$, $-5\sqrt{2}e^{-19/12}$, 0. По этим данным постройте интерполяцию кубическими сплайнами. Сравните полученную интерполяционную кривую с функцией $f(x) = 10e^{-x/3}\cos((x-2)\pi)$. Для этого на интервале, на котором расположены узлы, постройте одновременно графики функции $f(x)$ и интерполяционную кривую, отметьте узловые точки; вычислите и сравните значения функции $f(x)$ и интерполяционной кривой в узлах. Кривые на графиках изображайте точками, не соединяя их между собой.

Задание 9.19. Задайте два массива-столбца – массив узлов: 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 5.5, 6.0, 7.0, 7.5, 8.0, 9.0, 9.5, 10; массив значений неизвестной функции в этих узлах: –0.6, 0.3, –1.0, –2.5, –1.3, 4.3, 2.4, –1.6, –2.2, 0.3, 0.5, –1.3, 1, 3.6. Используя метод наименьших квадратов, подберите аппроксимирующую функцию так, чтобы стандартное среднеквадратичное отклонение этой функции от исходных данных в узлах было минимальным. Вычислите значения полученной аппроксимирующей функции в точках с координатами: 4,5; 6,5; 8,5. Полученную функцию преобразуйте в символьную и вычислите ее производную. На интервале, на котором расположены узлы, постройте одновременно графики аппроксимирующей функции и ее производной, отметьте узловые точки. Кривые на графиках изображайте точками, не соединяя их между собой.

Задание 9.20. В MuPAD напишите программу, в которой для выражения $4x^3y - 3x^2y^2 + 4xy^3$ вычисляются все первые производные (по dx и dy) и неопределенные однократные интегралы. Каждое вычисление оформите в виде функции, вызвав эти функции, вычислите значения производных и интегралов при $x = 2$ и $y = 3$. Кроме того, вычислите значение исходного выражения при этих значениях переменных x и y. Создайте mu-файл, в который скопируйте все объявленные функции. Создайте новый mn-файл, в котором выполните те же вычисления для выражения $(x - y)^2 \sin y$, загрузив функции из созданного ранее mu-файла. В редакторе m-файлов напишите программу, которая, используя функции, сохраненные в mu-файле, выполняет те же вычисления для символьных выражений $4x^3y - 3x^2y^2 + 4xy^3$ и $(x - y)^2 \sin y$. Результаты вычислений выведите в командное окно и сравните с результатами, полученными в MuPAD.

АЛФАВИТНЫЙ УКАЗАТЕЛЬ ФУНКЦИЙ И ОПЕРАТОРОВ MUPAD

$ 91, 149, 154, 365, 370
:= 30
-> 43
append 374
args 391
array 367
assume 110
assumeAlso 111
assuming 113
asympt 169
bernstein 335
card 374, 375
case 379
coerce 204, 242
collect 98
combine 100
conjugate 217
curl 256
D 150
delete 25
denom 105
det 220
diff 148
DIGITS 24, 126
discont 158
divergence 256
eval 88, 108
evalAt 86
expand 96
expose 394
expr 171, 172
expr2text 350
factor 97
factorout 96
fclose 354
finput 356
float 78
fname 354
fopen 352
for 381
fprint 357
frandom 339
fread 361
freeze 108
ftextinput 355
getprop 112
gradient 256
hold 107, 143, 148, 153, 177, 178, 179
htranspose 217
I 24
if 376
ifactor 98
indets 102
infinity 24
input 346
int 173
interpolate 326

intlib::byparts.....178
intlib::changevar.....179
inverse.....222
is.....124, 215, 279
isolate.....115
laplacian.....256
length.....370
lhs.....105
limit.....145
linalg::addCol.....209
linalg::addRow.....209
linalg::adjoint.....222
linalg::angle.....252
linalg::basis.....254
linalg::charmat.....241
linalg::charpoly.....241
linalg::col.....207
linalg::concatMatrix.....212
linalg::crossProduct.....250
linalg::delCol.....207
linalg::delRow.....208
linalg::det.....220
linalg::eigenvalues.....243
linalg::eigenvectors.....244
linalg::expr2Matrix.....234
linalg::factorLU.....228
linalg::factorQR.....230
linalg::gaussElim.....224
linalg::gaussJordan.....227
linalg::hilbert.....201
linalg::isHermitian.....215
linalg::isPosDef.....215
linalg::isUnitary.....215
linalg::matdim.....213
linalg::matlinsolve.....235
linalg::multCol.....208
linalg::multRow.....208
linalg::ncols.....214
linalg::nonZeros.....214
linalg::nrows.....214
linalg::orthog.....253
linalg::pascal.....201
linalg::randomMatrix.....202
linalg::rank.....219
linalg::row.....207
linalg::scalarProduct..249
linalg::setCol.....211
linalg::setRow.....211
linalg::stackMatrix.....212
linalg::submatrix.....206
linalg::substitute.....212
linalg::swapCol.....209
linalg::swapRow.....209
linalg::toeplitz.....200
linalg::tr.....218
linalg::transpose.....217
linalg::vandermonde.....201
linalg::vecdim.....214
linsolve.....136
lmonomial.....171
matrix.....192
matrix::identity.....200
max.....373
min.....373
nops.....102, 370
normal.....82
NOTEBOOKPATH.....354
nthmonomial.....171
numer.....105
numeric::butcher.....297
numeric::cubicSpline.....329
numeric::det.....220

numeric::eigenvalues.....243
numeric::eigenvectors ..244
numeric::factorLU.....228
numeric::factorQR.....230
numeric::fsolve.....131, 139
numeric::gldata.....183
numeric::gtdata.....182
numeric::int.....185
numeric::inverse 222
numeric::linsolve.....136
numeric::matlinsolve.....236
numeric::ncdata.....181
numeric::ode2vectorfield 300
numeric::odesolve.....302
numeric::odesolve2 302
numeric::odeToVectorField 300
numeric::polyrootbound 127
numeric::polyroots 126
numeric::polysysroots ..138
numeric::product 145
numeric::quadrature 184
numeric::rank 219
numeric::realroot.....130
numeric::realroots 128
numeric::solve.....133, 140
numeric::sum.....145
O.....165
ode 268, 286
ode::evalOde.....280
ode::exponentialSolutions 274
ode::getOrder 266
ode::isLODE 263
ode::mkODE 262
ode::normalize 265
ode::polynomialSolutions 274
ode::rationalSolutions 274
ode::series 276
ode::solve.....269, 287
ode::vectorize 264
op.....103
ORDER.....24, 166
partfrac 82
PI.....24
piecewise 46
plot 50
plot::Curve2d.....62
plot::Function2d 56
plot::Hatch 68
plot::Line2d.....64
plot::Ode2d 313
plot::PointList2d.....64
plot::Polygon2d.....64
poles.....159
polylib::realroots.....125
print.....348
proc 48, 388
procname 392
product.....141
radsimp.....84
random 342
read.....363
repeat 386
reset.....19
return 390
revert 374
rewrite.....93, 278
RGB::random 56
rhs 105

RootOf 118
series 170
simplify 79, 115
Simplify 79, 81, 115
simplifyFraction 82
simplifyRadical 84
solve 117, 139, 269
sort 374
stats::normalRandom 343
subs 87, 123, 279
subsex 90
sum 141
taylor 167
textinput 348
time 396
transpose 217
type 25
unassume 111
unfreeze 109
union 374
val 86, 107
while 385
write 359

ЛИТЕРАТУРА

1. *MuPAD*: MatLab Documentation [Электронный ресурс]. URL: http://www.mathworks.com/help /symbolic/mupad-1.html (дата обращения: 23.05.2016)
2. *Symbolic Math Toolbox*: MatLab Documentation [Электронный ресурс]. URL: http://www.mathworks.com/help/symbolic/index.html (дата обращения: 23.05.2016)
3. *Дьяконов В. П.* MATLAB : полный самоучитель. М. : ДМК Пресс, 2014. 768 с.
4. *Амос Г.* MATLAB : теория и практика. М. : ДМК Пресс, 2016. 416 с.
5. *Мещеряков В.В.* Задачи по математике с MATLAB&SIMULINK. М. : ДИАЛОГ-МИФИ, 2007. 528 с.
6. *Ануфриев И.Е., Смирнов А.Б., Смирнова Е.Н.* MATLAB 7. СПб. : БХВ-Петербург, 2005. 1104 с.
7. *Половко А.М., Бутусов П.Н.* MatLab для студента. СПб. : БХВ-Петербург, 2005. 320 с.
8. *Корюкина Е.В.* Моделирование физических и биологических процессов в системе MAPLE 11 : учеб.-метод. комплекс. Томск : Ин-т дистанционного образования Том. гос. ун-та, 2008 [Электронный ресурс]. URL: http://ido.tsu.ru/cd-dvd/0/1801/ Доступ из сети ТГУ (дата обращения: 23.05.2016)
9. *Ревинская О.Г.* Основы программирования в MatLab : учебное пособие. СПб. : БХВ-Петербург, 2016. 208 с.
10. *Ревинская О.Г.* Введение в практикум по вычислительной математике. Интерполяция и аппроксимация. Решение нелинейных уравнений и их систем. Численное интегрирование и дифференцирование: учебное пособие. Томск : Изд-во НТЛ, 2012. 236 с.
11. *Авдюшев В.А.* Численные методы интегрирования обыкновенных дифференциальных уравнений (ОДУ) : учеб.-метод. комплекс. Томск : Ин-т дистанционного образования Том. гос. ун-та, 2009 [Электронный ресурс]. URL: http://ido.tsu.ru/cd-dvd/0/2488/ Доступ из сети ТГУ (дата обращения: 23.05.2016)

СОДЕРЖАНИЕ

Предисловие 3

Глава 1. Приложение MuPad – пакет расширения MatLab для выполнения символьных вычислений 4

1.1. Приложение MuPAD 5
1.2. Запуск приложения MuPAD 5
1.3. Интерфейс приложения MuPAD 7
1.4. Создание, редактирование, сохранение mn–файла 12
1.5. Выполнение вычислений 15
1.6. Встроенная справочная система 16

Глава 2. Основные элементы языка и средства программирования в MuPAD 18

2.1. Структура программы 18
2.2. Идентификаторы 19
2.3. Константы и переменные 23
2.4. Математические и логические операции 27
2.5. Выражения, уравнения и неравенства 28
2.6. Присвоение значений переменным 30
2.7. Функции 33
2.7.1. Стандартные функции 35
2.7.2. Нестандартные функции 42

Глава 3. Графические возможности MuPAD 50

3.1. График функции одной переменной 56
3.2. График параметрически заданной функции 61
3.3. Графическое представление дискретного множества точек 64
3.4. График нескольких функций одной переменной 68
3.5. График функции двух переменных 72
3.6. Анимация 74

Глава 4. Аналитические преобразования математических выражений, уравнений, неравенств 77

4.1. Упрощение выражений, уравнений, неравенств 79
4.2. Подстановка 84
4.3. Запись выражения, уравнения, неравенства через другие элементарные функции 93
4.4. Раскрытие скобок и вынесение общих множителей 95
4.5. Преобразование отдельных составляющих частей выражений, уравнений, неравенств 101
4.6. Отложенные вычисления 106
4.7. Ограничение свойств переменных 109

Глава 5. Решение алгебраических и трасцендентных уравнений (неравенств) и их систем 114
5.1. Аналитическое решение уравнений и неравенств 114
5.2. Численное решение уравнений 126
5.3. Решение систем уравнений, неравенств 136

Глава 6. Решение задач математического анализа 141
6.1. Вычисление сумм и произведений 141
6.2. Вычисление пределов 145
6.3. Дифференцирование 148
6.4. Исследование функции 153
6.4.1. Поиск экстремумов функции 153
6.4.2. Поиск точек перегиба функции 157
6.4.3. Поиск особых точек функции 158
6.4.4. Построение касательных и асимптот 161
6.5. Разложение функции в ряд 164
6.6. Интегрирование 173
6.6.1. Аналитическое вычисление интегралов 173
6.6.2. Численное интегрирование 181
6.6.3. Площадь фигуры под кривой 187
6.6.4. Площадь плоской фигуры, заключенной между двумя кривыми 188
6.6.5. Положение центра инерции плоской фигуры 189

Глава 7. Решение задач линейной алгебры 191
7.1. Матрицы и векторы (создание) 191
7.1.1. Создание матрицы произвольного вида 192
7.1.2. Создание матриц стандартного вида 198
7.2. Изменение матрицы (вектора) 204
7.3. Свойства матрицы (вектора) 213
7.4. Операции с матрицами (векторами) 216
7.5. Применение матриц (векторов) 232
7.5.1. Вычисление миноров 232
7.5.2. Решение систем линейных уравнений. Решение матричных уравнений 233
7.5.3. Собственные векторы и собственные значения матрицы 240
7.5.4. Векторы и векторные поля 248

Глава 8. Решение обыкновенных дифференциальных уравнений и их систем 260
8.1. Аналитическое решение обыкновенного дифференциального уравнения 267
8.1.1. Общее и частное решение обыкновенного дифференциального уравнения 267
8.1.2. Фундаментальная система решений линейного однородного обыкновенного дифференциального уравнения 273

8.1.3. Решение обыкновенного дифференциального уравнения в виде разложения в ряд 276
8.1.4. Проверка достоверности решений обыкновенного дифференциального уравнения....278
8.1.5. Графическое представление решения обыкновенного дифференциального уравнения....281
8.2. Аналитическое решение систем обыкновенных дифференциальных уравнений....285
8.3. Численное решение обыкновенных дифференциальных уравнений и их систем....292
8.3.1. Методы численного решения обыкновенных дифференциальных уравнений 292
8.3.2. Применение методов численного решения обыкновенных дифференциальных уравнений 299
8.3.3. Графическое представление численного решения обыкновенного дифференциального уравнения....311

Глава 9. Интерполяция и полиномиальное представление функций 325
9.1. Интерполяция Лагранжа 325
9.2. Кубические сплайны....328
9.3. Представление аналитической функции полиномами в форме Бернштейна 334

Глава 10. Элементы структурного программирования....339
10.1. Генераторы случайных чисел 339
10.2. Ввод информации с клавиатуры в диалоговом режиме 346
10.3. Вывод информации....348
10.4. Использование внешних файлов для сохранения и загрузки информации 351
10.4.1. Обращение к файлу 352
10.4.2. Чтение данных из файла 355
10.4.3. Запись данных в файл 357
10.4.4. Сохранение в файле и загрузка из файла переменных MuPAD 359
10.4.5. Сохранение в текстовом файле и загрузка из него команд MuPAD....362
10.5. Индексированные переменные....363
10.5.1. Строки, последовательности, списки и множества 364
10.5.2. Массивы и таблицы....367
10.5.3. Преобразование индексированных переменных....370
10.6. Операторы условного перехода....375
10.6.1. Оператор условного перехода IF....376
10.6.2. Оператор выбора CASE 379
10.7. Операторы цикла 381

10.7.1. Оператор FOR-цикла 381
10.7.2. Оператор цикла WHILE 384
10.7.3. Оператор цикла REPEAT 386
10.8. Нестандартные функции 387
10.9. Контроль продолжительности вычислений 396

Глава 11. Символьные вычисления в m-файлах 398
11.1. Символьные переменные и выражения 403
11.2. Символьные функции 410
11.3. Управление из MatLab выполнением программ, написанных в MuPAD 416
11.4. Стандартные функции MatLab для выполнения символьных вычислений 422
11.4.1. Элементарные и специальные математические функции 423
11.4.2. Графики символьных функций 429
11.4.3. Аналитические преобразования математических выражений, уравнений, неравенств 433
11.4.4. Решение алгебраических и трансцендентных уравнений (неравенств) и их систем 439
11.4.5. Решение задач математического анализа 442
11.4.6. Решение задач линейной алгебры 446
11.4.7. Решение обыкновенных дифференциальных уравнений и их систем 454
11.4.8. Интерполяция и полиномиальное представление аналитических функций 460
11.4.9. Аппроксимация 463
11.5. Элементы структурного программирования 474

Глава 12. Задания для самостоятельного выполнения на компьютере 482
Тема 1. Выражения, функции. Графики функций 483
Тема 2. Аналитические преобразования выражений, уравнений, неравенств 484
Тема 3. Решение алгебраических и трансцендентных уравнений (неравенств) и их систем 488
Тема 4.1. Решение задач математического анализа 491
Тема 4.2. Решение задач математического анализа (продолжение) 495
Тема 5. Решение задач линейной алгебры 499
Тема 6. Решение обыкновенных дифференциальных уравнений и их систем 503
Тема 7. Интерполяция и полиномиальное представление функций 508
Тема 8. Программирование в MuPAD 511
Тема 9. Символьные вычисления в MatLab 513

Алфавитный указатель функций и операторов MuPAD 519

Литература 523

Учебное издание

Ревинская Ольга Геннадьевна

СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ
В MatLab

Учебное пособие

Редактор Е.Г. Шумская
Оригинал-макет Е.Г. Шумской
Дизайн обложки Л.Д. Кривцовой

Подписано к печати 03.10.2018 г. Формат $60\times84^{1}/_{16}$.
Бумага для офисной техники. Гарнитура Times.
Усл. печ. л. 30,81.
Тираж 50 экз. Заказ № 2958.

Отпечатано на оборудовании
Издательского Дома
Томского государственного университета
634050, г. Томск, пр. Ленина, 36
Тел. 8+(382-2)–52-98-49
Сайт: http://publish.tsu.ru
E-mail: rio.tsu@mail.ru

ISBN 978-5-94621-745-3

9 785946 217453