ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Физический факультет

А.В. Борисов
А.А. Воронцов

Введение в MATLAB и его применение для конструирования физических моделей

Учебно-методическое пособие

Томск
2012

УДК 004.67 MatLab:517.925(075.8)
ББК 22.3
Б 82

РАССМОТРЕНО И УТВЕРЖДЕНО методической комиссией физического факультета

Протокол № 4 от 09.04.2012 г.

Председатель комиссии  В.М. Вымятнин

Учебно-методическое пособие предназначено для изучения MATLAB-среды решения научных и инженерных задач.

В пособии рассмотрены основные особенности интерфейса и языка программирования MATLAB. Описаны простейшие встроенные функции для численного решения научных и инженерных задач. Рассмотрены возможности MATLAB для проектирования пользовательских графических интерфейсов.

Для студентов младших курсов физико-математических и технических специальностей университетов, а также для преподавателей по этим направлениям.

Авторы: Борисов А.В., доцент каф. общей и экспериментальной физики ТГУ
Воронцов А.А., ст.препод. каф. общей и экспериментальной физики ТГУ

Рецензент: Горчаков Л.В. – профессор каф. общей и экспериментальной физики ТГУ

© Томский государственный университет, 2012

Содержание

1. Методические указания **4**

2. Введение **5**

3. Введение в MATLAB **8**
3.1. Краткая история и структура MATLAB 8
3.2. Интерфейс MATLAB 9
3.3. Типы данных в MATLAB 18
3.4. Работа в редакторе .m файлов 27
3.5. Задания для самостоятельной работы 33

4. Визуализация данных **34**
4.1. Построение графиков от функции одной переменной 34
4.2. Задания для самостоятельной работы 38

5. Вычисления в MATLAB **40**
5.1. Векторные и матричные операции 40
5.2. Комплексные числа 44
5.3. Решение уравнений и систем уравнений 45
5.4. Интерполяция 48
5.5. Численное дифференцирование и интегрирование 49
5.6. Решение обыкновенных дифференциальных уравнений 53
5.7. Введение в символьные вычисления 57
5.8. Задания для самостоятельной работы 65

6. Создание графического интерфейса средствами MATLAB **67**
6.1. Иерархия классов MATLAB 67
6.2. Среда создания графического интерфейса GUIDE 74
6.3. Задания для самостоятельной работы 88

7. Заключение **89**

8. Литература **90**

1. Методические указания

Учебно-методический комплекс (УМК) "Введение в MATLAB и его применение для конструирования физических моделей" разработан в рамках проекта "Национальный исследовательский университет".

"Учебно-методический комплекс адресован студентам физико-математических специальностей, начинающим изучение программирования.

Целью курса "Введение в MATLAB и его применение для конструирования физических моделей" является получение базовых знаний об интегрированной системе MATLAB, её вычислительных и графических функциях, возможностях по проектированию графического интерфейса. Эти знания необходимы будущим специалистам для проведения эффективных расчётов и создания математических моделей физических явлений. **Задачами курса** являются:

- изучение и настройка интерфейса MATLAB;
- изучение типов данных и базовых структур программирования;
- изучение средств визуализации данных;
- изучение классических численных методов на примере встроенных функций;
- изучение инструментов проектирования графического интерфейса.

Требования к уровню освоения курса сводятся к следующему. Студент должен знать:

- ключевые слова и константы MATLAB;
- встроенные математические функции MATLAB и операторы языка программирования;
- встроенные функции для визуализации данных;
- встроенные функции для создания и преобразованния массивов;
- встроенные функции, реализующие классические численные методы;
- иерархию классов MATLAB.

Студент должен уметь:

- изменять настройки графического интерфейса MATLAB;
- строить и оформлять средствами MATLAB графики различных функций;
- программировать сценарии и функции;
- проектировать пользовательский графический интерфейс.

Для самоконтроля успешности освоения курса в конце каждой главы имеются задания для самостоятельной работы. Для наглядности в тексте курса приведено множество скриншотов с работающей MATLAB 2010b.

2. Введение

Данный учебно-методический комплекс адресован пользователям, не имеющим опыта работы в MATLAB. Навыки программирования для освоения курса также не требуются, но будут полезны для понимания ценности и эффективности изучаемого программного продукта. У читателей предполагается знание физики и высшей математики в объеме одного-двух первых семестров университетского курса. Знания по физике необходимы для понимания примеров и для создания самостоятельных моделей физических явлений. Знания в математических дисциплинах предполагают умение считать производные, интегралы, решать дифференциальные уравнения (обыкновенные) и умение работать с векторами и матрицами – арифметические операции, обращение, поиск собственных векторов матриц, решение системы линейных уравнений.

Использование MATLAB может свести непосредственно программирование к минимуму. Значительная часть всех вычислительных алгоритмов, используемых в данный момент, реализована в MATLAB в виде встроенных функций. От пользователя зачастую требуется только знание формата вызова соответствующих алгоритмов. Математическая же подготовка может пригодиться для понимания, какой именно алгоритм позволит построить решение задачи наиболее эффективно и по точности, и по времени поиска решения. Многие операции в MATLAB не только не требуют программирования, но и допускают работу только в режиме использования графического интерфейса.

Всякий раз, когда это возможно, при работе в MATLAB следует выбирать для решения задачи алгоритмы, уже реализованные в системе. Как показывает практика, крайне сложно написать пользовательскую функцию, обеспечивающую такую же скорость и точность вычислений, как встроенные в MATLAB функции. Основная особенность MATLAB, пронизывающая все вычислительные алгоритмы, состоит в использовании в качестве базовой структуры массива. После традиционных языков программирования может показаться странным, что даже простое число в MATLAB и то представляется как массив 1x1. Определить вектор в MATLAB предельно просто с использованием оператора двоеточия. А такая классическая структура программирования, как цикл, в MATLAB может быть заменена векторными вычислениями, и это обеспечит максимальную скорость работы.

Данный УМК не является справочником по функциям MATLAB. Одно их перечисление с указанием даже формального синтаксиса вызова займёт несколько тысяч страниц. Достаточно сказать, что только по одному из расширений MATLAB – Symbolic Math Toolbox фирменное руководство,

доступное на сайте производителя, занимает почти 500 страниц, и даже это в разы превосходит объем данного курса. Авторы УМК предполагают, что, получив некоторый минимум информации о MATLAB, пользователь далее может самостоятельно найти в обширной документации более подробную информацию. Так, например, при обсуждении методов решения обыкновенных дифференциальных уравнений в MATLAB был приведен пример использования только одного из них – наиболее универсального. Подробное описание самих численных методов не входит в задачи данного УМК, и за более подробной информацией можно обратиться к одному из источников, указанных в списке литературы. Подробное описание всех алгоритмов MATLAB для решения задачи данного класса также не входит в нашу задачу – для этого следует обратиться к фирменной документации.

УМК состоит из 4 глав. Первая глава посвящена введению в MATLAB. Использовать MATLAB возможно в режиме очень мощного калькулятора. В этом случае не требуется писать программные коды, вся работа заключается в наборе простых команд в главном окне MATLAB и просмотре результатов их выполнения. Для такой работы нужно знать интерфейс MATLAB, уметь настроить его и использовать встроенные средства для ввода/вывода, импорта/экспорта данных. При работе с MATLAB важно уметь:

- использовать историю команд для доступа к недавно введенным командам;
- вести дневник сессий MATLAB, в который записываются не только вводимые команды, но и вся выводимая в главное окно информация;
- использовать специальное окно для просмотра информации о всех переменных в памяти; MATLAB, для создания новых и удаления старых переменных, для быстрого открытия/сохранения, импорта/экспорта данных, для быстрого построения графиков функций;
- использовать редактор массивов для удобного редактирования данных;
- использовать редактор исходных кодов MATLAB для создания простейших программ;
- использовать специальное окно для обзора текущего каталога, для создания/удаления файлов;
- настраивать внешний вид всех окон и их удобное расположение;
- работать с разными типами данных в MATLAB, изменять тип данных, изменять представление данных на экране.

Все эти важные мелочи помогают сделать работу в MATLAB достаточно комфортной даже для новичка.

Во второй главе рассматриваются средства MATLAB по визуализации данных. Как уже было указано, многие операции в MATLAB могут быть

выполнены без программирования, лишь средствами графического интерфейса. Оформление графиков функций одной и двух переменных, специальные типы графиков и анимации могут быть подготовлены в специальном графическом окне. Вне его необходимо только создать данные, подлежащие визуализации.

Третья глава посвящена классическим численным методам – решению уравнений. в том числе нелинейных, решению систем линейных уравнений, интерполяции, дифференцированию, интегрированию, численным методам решения обыкновенных дифференциальных уравнений. Кроме классического подхода к решению таких задач, кратко обсуждается возможность символьного решения подобных задач. Краткость расмотрения обусловлена тем, что эти возможности собраны в специальном расширении MATLAB – Symbolic Math Toolbox, которое лицензируется отдельно от базовой версии MATLAB.

Четвертая глава посвящена средствам проектирования графического интерфейса пользователя. Конструирование интерфейса в MATLAB происходит по технологии drag-and-drop с использованием специального конструктора. Допускается создание графического интерфейса целиком из исходных кодов, но для этого требуется опыт. Численные методы решения физических задач, будучи помещенными внутрь графического интерфейса, создают наглядную модель явления. Создание подобной модели на примере изучения движения тела, брошенного под углом к горизонту, рассматривается также в этой главе.

В конце каждой главы находятся задания для самостоятельной работы, требующие серьезной работы. Для выполнения упражнений требуется использование MATLAB, достаточное время и некоторые знания, прямо не указанные в тексте главы. Выполнение заданий может вызвать некоторые затруднения, преодолевая которые читатель будет получать неоценимый опыт.

За границами УМК остаётся обширный материал о применении более чем 90 расширений MATLAB, о создании исполняемых приложений, о совмещении MATLAB с другими технологиями. Не будут обсуждаться также возможности параллельного программирования в MATLAB. Все эти вопросы (как и многие другие) вряд ли будут полезны начинающим пользователям.

3. Введение в MATLAB

3.1. Краткая история и структура MATLAB

Слово "computer" само по себе означает "вычислитель" и исторически первая и наиболее существенная роль компьютеров заключается в численном решении научно-технических задач. Одним из первых языков программирования, специально созданных для научных вычислений в 1957 году, стал FORTRAN. С самого начала программы на этом языке демонстрировали высокую скорость работы и достаточную простоту написания. В конце 1970-х для реализации доступа к библиотекам LINPACK и EISPACK на языке Фортран был создан пакет программ, получивший позднее название MATLAB. В 1984 году основная часть программного кода была переписана на языке C и основана компания MathWorks.

MATLAB стал первым и до сих пор является наиболее популярным средством решения научно-технических задач, в основе которого лежат векторные вычисления. Использование векторов, как базисных конструкций программирования, не только повышает скорость работы программ, но и сокращает время их разработки. В современном виде MATLAB представляет собой не просто язык программирования высокого уровня, а интегрированную среду для решения любых научно-технических задач. Важной особенностью MATLAB является модульность, благодаря чему возможности вычислительного ядра стали быстро дополняться специализированными наборами программ — расширений (Toolboxes). В последней на данный момент версии MATLAB таких расширений более 90. Среди них присутствуют, как классические — для решения задач интерполяции, статистики, математической физики, так и расширения для обработки сигналов, изображений, работы с базами данных, решения финансовых задач. Некоторая часть расширений имеет собственный графический интерфейс.

В качестве наиболее значительного дополнения к MATLAB следует рассматривать SIMULINK, позволяющий моделировать сложные динамические системы без программирования из заранее предоставленных блоков. При этом имеется возможность создавать пользовательские блоки как с помощью встроенных средств с графическим интерфейсом, так и с нуля из кодов MATLAB. Третьим важным составляющим среды разработки MATLAB является Stateflow — инструмент для событийно-управляемого проектирования.

В MATLAB имеется возможность создавать собственный графический интерфейс пользователя, исполняемые приложения, не требующие для работы установленного MATLAB. Имеются также интерфейсы ко многим технологиям (COM, DDE, .NET) и языкам программирования (C, C++, Fortran, Java, Ada). Есть возможность использовать MATLAB совместно

с офисными программами и специализированными CAD/CAM/CAE программами.

Кроме поставляемых самой компанией MathWorks тулбоксов, имеется около 500 расширений MATLAB, предоставляемых сторонними разработчиками. По официальной информации, количество проданных лицензий превышает 4 миллиона, количество книг, посвященных использованию MATLAB в различных предметных областях, превышает 1000. Сотни университетов по всему миру используют MATLAB в учебных курсах и в исследованиях. Наиболее актуальная информация о последних версиях и всех продуктах компании MathWorks доступна на сайте http://www.MathWorks.com

3.2. Интерфейс MATLAB

Для любой интегрированной среды разработки дружественный интерфейс, с богатыми инструментальными возможностями и тонкой настройкой является важной составляющей успеха. Не стоит при этом забывать, что любые изыски и удобства графического интерфейса неизбежно отражаются на системных требованиях программы и размерах занимаемой ею оперативной памяти. Для математических программ критично быстродействие и объем оперативной памяти. В большинстве современных математических пакетов ещё с первых версий сохранился консольный интерфейс. Командная строка обеспечивает минимум возможностей, не допускает пользовательскую настройку, но при этом занимает в памяти минимум места. Чтобы запустить MATLAB в консольном режиме, достаточно в командной строке вашей операционной системы набрать команду matlab (предполагается, что в переменной path хранится путь к исполняемому файлу MATLAB). Кроме этого можно выбрать свойства ярлыка запуска MATLAB и указать опцию -nojvm. Эту же опцию можно указывать и непосредственно в консоли. В течение последних нескольких лет графический интерфейс MATLAB использует виртуальную машину Java (Java Virtual Mashine).

Включение этой опции подавляет использование виртуальной машины Java и значительно сокращает объём занятой MATLAB оперативной памяти. Работать с MATLAB в консольном режиме следует в случае недостатка оперативной памяти, для целей освоения базовых возможностей MATLAB следует использовать графический интерфейс, устанавливаемый по умолчанию. Запуск MATLAB в консольном режиме сделает невозможными работу большинства графических окон и установку некоторых важных параметров работы. Также возможно использовать MATLAB в следующем режиме: в качестве редактора исходных кодов использовать стороннюю

программу, а из командной строки вызывать только соответствующие m-функции.

По умолчанию MATLAB запускается в варианте "четыре окна":

1. Command Window;
2. Current Folder;
3. Command History;
4. Workspace.

В окне Command Window вводятся команды языка MATLAB и выводится результат их выполнения. Для подавления вывода результатов вычислений в конце каждого оператора MATLAB следует использовать точку с запятой. Запрет вывода результатов играет не только эстетическую роль. Для достаточно больших массивов время вывода на экран может превышать время его вычисления (как ни странно). В качестве эксперимента можно использовать любую из нескольких десятков процедур MATLAB для генерации матриц. Даже если не создана специальная переменная для сохранения результата, MATLAB создаёт её самостоятельно с именем ans. По этому же имени можно обращаться к результату последнего вычисления MATLAB. Пользоваться этим следует лишь в исключительных ситуациях или в режиме использования MATLAB в качестве калькулятора для простейших вычислений. В любых программах каждый раз, когда пользователь не объявил переменную для сохранения результата, MATLAB будет использовать системное имя ans. При каждом новом вычислении без объявления результата вычисления значение переменной ans переписывается. При использовании этого имени в пользовательских программах может быть ситуация, когда в качестве значения этой переменной используется значение, уже существующее в памяти MATLAB.

Для набора нескольких строк программного кода в командном окне используется сочетание клавиш Shift + Enter . Нажатие клавиши Enter приводит к выполнение той части программного кода, которая была набрана, начиная с символа >> в командном окне. Следует обратить внимание на сравнительно новую функцию командного окна MATLAB. Слева от знака начала записи команд >> расположена кнопка с надписью fx для быстрого доступа к справочной системе MATLAB. К тому же эффекту приведёт выбор пунтка Function Browser меню Help главной панели инструментов.

Для очистки командного окна служит команда clc, её можно использовать и прямо в консоли MATLAB, и в любых программах.

При выводе числовых данных по умолчанию используется режим, в котором числа отображаются и в консоли и в окне Workspace в десятичном формате с 4 знаками после запятой. Формат вывода числовых данных определяется для консоли командой format. Если она указана без аргумен-

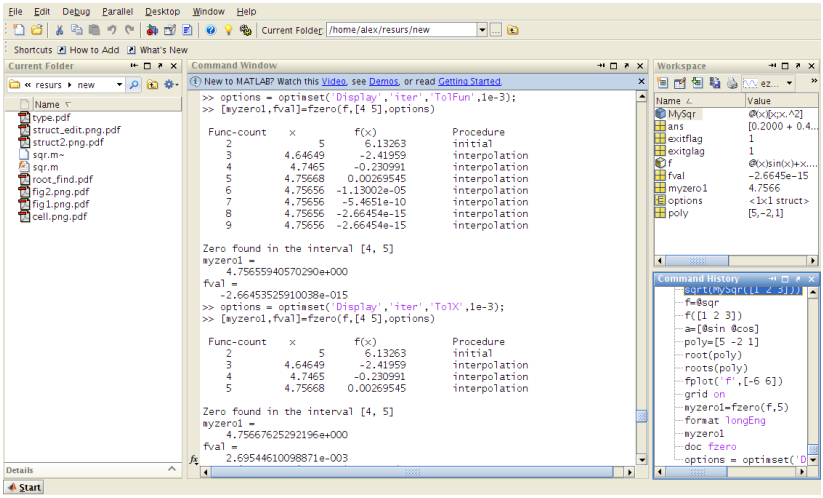


Рис. 1. Главное окно MATLAB с графическим интерфейсом по умолчанию. В центре – консоль Command Window, слева – обзор текущего каталога (Current Folder), справа сверху – рабочее пространство Workspace, справа внизу – история команд Command History

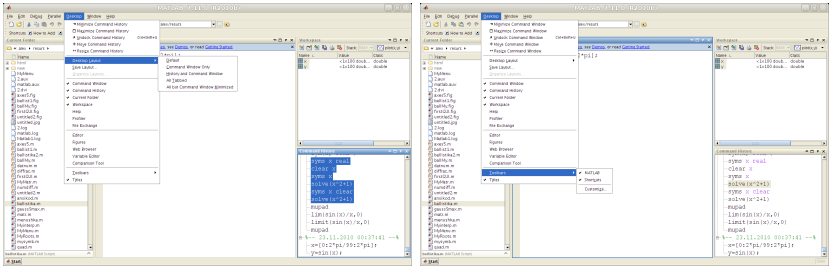


Рис. 2. Главный пункт меню, управляющий внешним видом интерфейса MATLAB. Слева – стандартные варианты графического интерфейса. Справа – стандартные панели инструментов MATLAB с возможностью создания пользовательских панелей

та, то устанавливается режим вывода по умолчанию. В окне Variable Editor формат отображения устанавливается в меню View–Numerical Array Format.

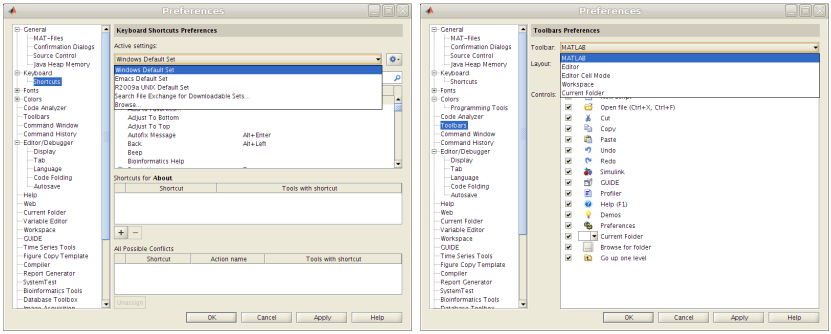


Рис. 3. Вкладки для настройки MATLAB. Слева – настройка сочетания клавиш. Обратите внимание, что есть несколько заранее предустановленных схем. Справа – настройка панелей инструментов

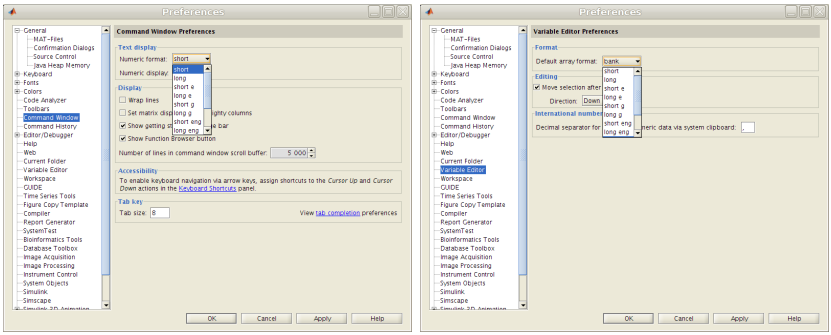


Рис. 4. Установка формата вывода чисел для главного окна и рабочего пространства – слева, для редактора переменных – справа. В первом случае дополнительно можно установить включение/выключение лишних строк в выводе данных на консоль

Таблица 1. Форматы вывода числовых данных

Аргумент	Значение	Пример вывода числа π
short или пустая строка	Значение по умолчанию – масштабируемый десятичный формат с 4 знаками после запятой	3.1416

продолжение на следующей странице

Аргумент	Значение	Пример вывода числа π
long	Масштабируемый десятичный формат с 14 знаками после запятой	3.141592653589793
short e	Экспоненциальный формат с 4 знаками после запятой и двумя разрядами для показателя степени	3.1416e+00
long e	Экспоненциальный формат с 15 знаками после запятой и двумя разрядами для показателя степени	3.141592653589793e+00
long g	Десятичный формат с 14 знаками после запятой	3.14159265358979
short g	Десятичный формат с 4 знаками после запятой	3.1416
longEng	Инженерный формат с 14 знаками после запятой и тремя знаками для показателя степени	3.14159265358979e+000
shortEng	Инженерный формат с 4 знаками после запятой и тремя знаками для показателя степени	3.1416e+000
hex	Шестнадцатиричный формат	400921fb54442d18
bank	Десятичный формат с двумя знаками после запятой	3.14
rat	Формат рациональных чисел	355/113
+	Положительные, отрицательные или нулевые числа отображаются соответственно знаками +, – и пробел, мнимая часть комплексного числа игнорируется	+
compact	Подавление вывода пустых строк результата вычислений в консоли	
loose	Включение вывода пустых строк результата вычислений в консоли	

Обратите внимание на различие в форматах short и short g. При использовании форматов short и long MATLAB самостоятельно "масштабирует" вывод чисел, приводя их к общему множителю. При этом пользователь

Пример 1. Различия в выводе чисел в массивах

```
>> x =[1 2 3 1234567890]
x =
   1.0e+09 *
    0.0000    0.0000    0.0000    1.2346
>> format short g
>> x
x =
            1            2            3   1.2346e+09
```

может неверно истолковать значения переменных.

Результат команды format является глобальным – новые установки вывода чисел будут применяться или до конца сеанса работы MATLAB, или до нового вызова format. Главная особенность этой команды заключается в том, что её применение не зависит от фактического типа данных переменной. Как уже упоминалось, все числовые переменные в MATLAB имеют тип double, если пользователь явно не указал другое. Параметры вывода чисел можно изменить глобально в графическом интерфейсе MATLAB. Для этого нужно выбрать в главном меню File–Preferences–Command Window и установить значения Numeric Format и Numeric Display

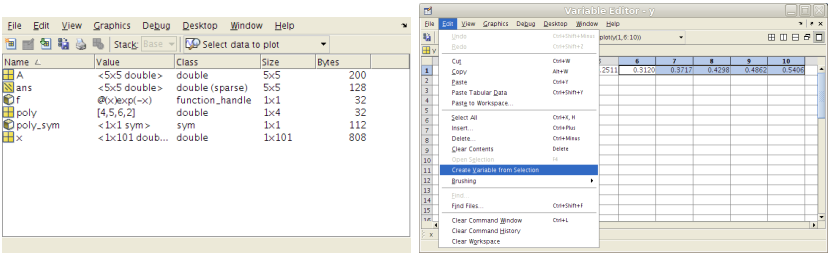


Рис. 5. Слева – окно рабочего пространства с переменными различного типа. Набор столбцов в окне не совпадает с установленным по умолчанию. Справа – окно редактора переменных. Показан один из пунктов меню, позволяющий вырезать, вставить данные и создать новую переменную из части уже имеющихся

В окне Workspace отображаются все данные, используемые (вычисляемые) в среде MATLAB. Как было отмечено, MATLAB самостоятельно присваивает результату последнего вычисления имя ans, и по этому имени переменную можно увидеть в окне рабочей области. По умолчанию в окне отображаются 4 столбца:

1. name – имя переменной;
2. value – значение переменной;
3. min – минимальное значение;

4. max – максимальное значение.

При работе с MATLAB всегда следует помнить, что базовым объектом для вычислений в MATLAB является матрица – двумерный массив. В этом легко убедиться, если в настройках Workspace выбрать дополнительные столбцы. Для этого выделите окно рабочей области, в главном меню MATLAB выберите View–Choose Columns и в появившемся списке выберите Size. После этого произведите любое сложение двух натуральных чисел и в качестве ответа в рабочей области в свойствах переменной ans увидите в поле Size значение $1x1$. Для удобной работы в дальнейшем полезно настроить окно Workspace для вывода информации о типе данных переменной и занимаемой ею памяти. Для этого снова нужно выбрать View–Choose Columns и в списке отметить Class и Bytes. Обратите после этого внимание на то, что результат вычисления суммы двух натуральных чисел отображается в виде переменной типа double с соответствующим размером в 8 байт.

В MATLAB есть еще одна особенность – для вычисления не обязательно явно указывать тип переменной. При этом всегда следует помнить, что если пользователь не указал явно тип переменной, то интерпретатор MATLAB самостоятельно будет рассматривать её как значение типа double (для числовых данных). Для нечисловых данных в MATLAB действует примерно такой же механизм. Например, если в командном окне набрать 'hello', то в рабочей области можно увидеть, что переменная ans имеет тип char с размером в 10 байт. Можно сразу же отметить еще одну особенность работы со строками в MATLAB – любая строка сама считается одномерным массивом, в результате значительная часть операторов MATLAB применима не только к числовым векторам и матрицам, но и к строкам.

Для очистки рабочего пространства можно использовать специальную команду clear. Если её использовать без аргументов, то в результате рабочая область очистится целиком. Если задать аргументы, например clear x, то удалена будет только переменная x. Аргументов у clear может быть несколько, все переменные для удаления перечисляются через пробел. Часто сеанс работы MATLAB длится долгое время или за короткое время происходит слишком частое создание и удаление переменных и область памяти, занимаемая MATLAB, фрагментируется. Для дефрагментации рабочего пространства используется функция pack. Следует избегать частого применения этой функции, и лучше в случае больших объемов информации упорядочивать их с самого начала. Каждый вызов этой функции записывает на жёсткий диск содержимое всех переменных рабочего пространства, очищает его и загружает вновь из временного файла.

Подробную информацию о переменных можно получить с помощью ко-

Пример 2. Выполнение команды whos

```
>> x=1:2:20;
f=sin(x);
whos
  Name        Size                Bytes  Class     Attributes
  f           1x10                   80  double
  x           1x10                   80  double
```

манды whos. Если её использовать без аргумента, то в результате в консоли MATLAB появится информация о всех переменных рабочего пространства в алфавитном порядке.

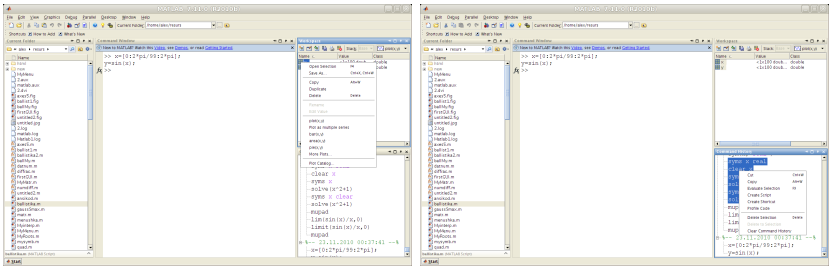


Рис. 6. Слева – контекстное меню рабочего пространства позволяет удалять, добавлять, создавать новые переменные и строить разного рода графики. Справа – контекстное меню истории команд, позволяющее выполнить, сохранить, удалить часть выделенных команд

Для простейших вычислений будут также интересны другие возможности окна Workspace. Например, если определить в командом окне вектор значений аргумента тригонометрической функции синус, а потом вычислить сам синус от этого вектора, то, не зная команды MATLAB по построению графиков, можно всё-таки построить график функции $\sin(x)$, используя только возможности графического интерфейса окна Workspace.

Наберите программный код, представленный в Примере 3, после этого в рабочей области появятся две переменные с именами f и x, достаточно выделить обе переменные, кликнуть правой кнопкой мыши и в появившемся контекстном меню выбрать пункт plot(x,f). В результате в отдельном окне будет построен график функции $\sin(x)$ на интервале $[0 : 2\pi]$. К тому же результату приведет выполнение оператора, находящегося в закомментированной строчке примера 3. При этом первым нужно выделить вектор x, при выделении первым вектора f именно этот вектор будет пер-

Пример 3. Построение графика функции $\sin(x)$

```
x=0:0.1;2*pi;
f=sin(x);
%plot(x,f)
```

вым аргументом процедуры plot, и в контекстном меню появится plot(f,x). Если выделить только один из векторов, то будет построен график вектора и в качестве аргумента будут значения номеров элементов вектора. Подобные манипуляции можно проделать и с многомерными функциями, а также для построения более сложных графиков. Элементы панели инструментов окна Workspace позволяют сохранять и открывать данные в различных Форматах. Например, графические файлы можно экспортировать в рабочее пространство MATLAB в виде матрицы. Любые переменные можно редактировать в специальном окне Variable Editor, которое открывается автоматически при двойном клике левой кнопкой мыши на любой переменной.

Окно Command History содержит историю команд, вводимых в окне Command Window. До тех пор пока история команд не пуста, внутри окна Command Window доступен повторный выбор и выполнение любой их них с использованием клавиш курсора на клавиатуре. Кроме того, в самом окне Command History можно выделить одну или несколько команд и в контекстном меню выбрать их повторное выполнение, удаление из истории или даже сохранение в виде файла исходного кода MATLAB. Можно использовать клавишу TAB для автодополнения вводимой в консоли команды. А если начать ввод команды с некоторого символа, то с помощью клавиш курсора можно будет перемещаться по командам, начинающимся именно с этой буквы. В этом окне сохраняются только команды, вводимые в консоли MATLAB, а весь вывод результатов вычислений не сохраняется. Иногда оказывается удобным сохранять сеанс работы с командным окном полностью. Команда diary filename создаёт специальный файл, в который будет записываться всё содержимое консоли. Команда diary on включает запись, а diary off выключает.

Наконец, окно Current Folder отображает содержимое текущего каталога. В операционных системах *nix в качестве такого каталога по умолчанию считается домашняя директория пользователя. В Windows это папка Мои документы. Возможности и панели инструментов и контекстного меню данного окна позволяют создавать, переименовывать, удалять любые файлы в текущем каталоге, в том числе и файлы исходных кодов MATLAB. Также можно создать архив, включающий файлы текущего каталога. Особого внимания заслуживает пункт Add to Path, который позволяет текущий каталог добавить к списку каталогов, где MATLAB производит поиск программ. Можно для начала выбрать на главной панели инструментов меню File и далее Set Path, в результате в специальном окне будет выведен список всех каталогов, включенных в пути поиска программ. Если введенная или в командном окне или в тексте программы последовательность символов не найдётся в качестве имени функции или сценария ни в одном из этих

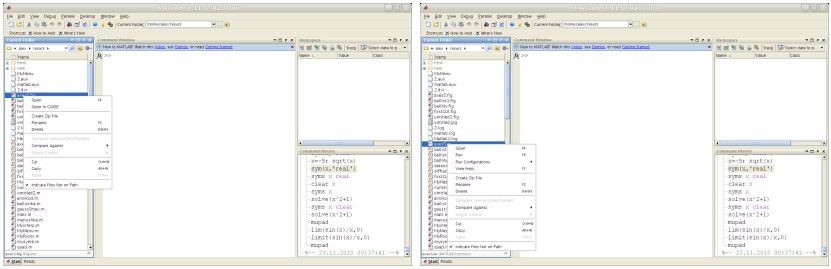


Рис. 7. Окно обзора текущего каталога. Обратите внимание на изменение контекстного меню в зависимости от типа файлов. В MATLAB имеется несколько типов файлов – файлы исходных кодов с расширением .m, файлы графического интерфейса с расширением .fig – чаще всего будут встречаться на страницах курса

каталогов, то MATLAB выдаст предупреждение об ошибке – ??? Undefined function or method

3.3. Типы данных в MATLAB

Иерархия типов данных MATLAB представлена на рис. 8. Любой тип данных с самого начала считается массивом с размерностью n. Даже просто число в памяти MATLAB будет храниться как матрица 1x1, строка будет храниться как вектор. Тот же принцип относится и к более сложным типам данных – структурам и ячейкам.

Для любых переменных в программных кодах MATLAB не обязательно указывать тип переменной, независимо от того, числовая она или текстовая. По умолчанию все числовые значения в MATLAB считаются имеющими тип double – числа с плавающей запятой двойной точности. Преобразование числовых данных к новому типу осуществляется одной из функций, приведенных в табл. 2.

Таблица 2. Функции преобразования числовых типов в MATLAB

Значение	Пояснение	Диапазон значений
int8	Формат однобайтовых целых чисел	от -128 до 127
uint8	Формат однобайтовых целых чисел без знака	от 0 до 255
int16	Формат двухбайтовых целых чисел	от -32768 до 32767

продолжение на следующей странице

Значение	Пояснение	Диапазон значений
uint16	Формат двухбайтовых целых чисел без знака	от 0 до 65535
int32	Формат четырехбайтовых целых чисел	от -2147483648 до 2147483647
uint32	Формат четырехбайтовых целых чисел без знака	от 0 до 4294967295
int64	Формат восьмибайтовых целых чисел	от -2^{63} до $2^{63}-1$
uint64	Формат восьмибайтовых целых чисел без знака	от 0 до $2^{64}-1$
single	Формат четырехбайтовых вещественных чисел с плавающей запятой одинарной точности	от $1.175 \cdot 10^{-38}$ до $3.403 \cdot 10^{38}$
double	Формат восьмибайтовых вещественных чисел с плавающей запятой двойной точности	от $2.225 \cdot 10^{-308}$ до $1.798 \cdot 10^{308}$

Максимальные и минимальные значения числовых данных можно выяснить с помощью функций intmax и intmin для целочисленных типов и realmax и realmin для вещественных чисел. Используя функции преобразования типов, нужно иметь в виду некоторые особенности. Одни из них определяются просто выходом операндов арифметических операций за границы. Другие специфичны для сочетания целочисленных и вещественных данных в MATLAB. Так, например, если один операнд относится к целым числам, а второй к вещественным, то MATLAB будет производить вычисления с двойной точностью, но результат будет преобразован к типу целых чисел. Если операндов более чем два, то результаты, возвращаемые MATLAB, могут быть неверными. Наконец, в любой арифметической операции не допускаются операнды различных целочисленных типов (строки 22–25 пример 4).

Целочисленная арифметика может быть очень удобна при решении некоторых задач. Например, при обработке растровых изображений каждый пиксель кодируется набором целых чисел, определяющих три цвета в RGB палитре. Целые числа всегда экономят память и часто позволяют добиться большей скорости вычислений. Однако, учитывая приведенные выше особенности, следует в традиционных вычислениях крайне осторожно пользоваться целочисленными типами данных.

Два вещественных типа данных MATLAB – double и single – могут ис-

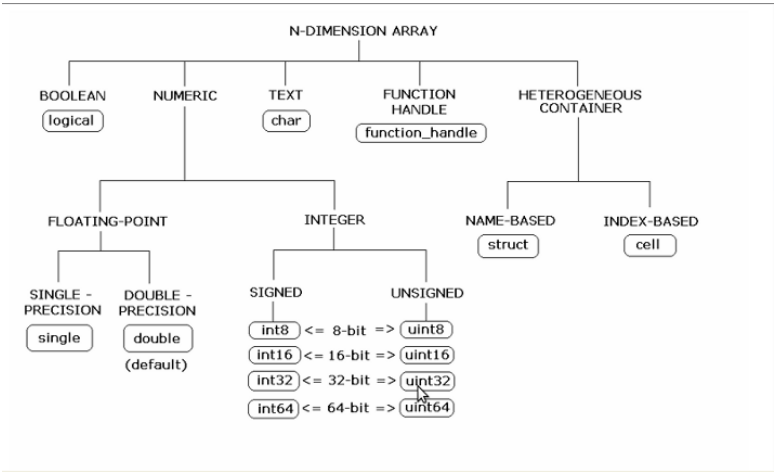


Рис. 8. Типы данных MATLAB

пользоваться совместно во всех арифметических операциях. Все матричные и векторные функции MATLAB одинаково применимы к обоим типам данных. Всегда нужно помнить, что тип single не только экономит память, но и обеспечивает меньшую точность вычислений – 8 знаков после запятой в отличие от 15 знаков после запятой для типа double. В случае выхода за границу вещественного типа данных результатом будет Inf – машинная бесконечность. При совмещении обоих типов данных в арифметических операциях результат операции будет иметь тип single.

Символьный, или строковый, тип данных в MATLAB называется char. Любая строковая переменная определяется выражением, стоящим внутри символов одинарных кавычек. В приведенном выше примере аргументом функции realmax была именно строка. Каждый символ строки хранится в кодировке ASCII Unicode и занимает два байта. Допускаются строковые переменные, включающие символы кириллицы. Следует обратить внимание на различие в задании числовых векторов в MATLAB и строк. Например, конструкция x=[1 2] приведет к созданию вектора из двух чисел. В то же время синтаксически аналогичная конструкция со строковыми операндами приводит к конкатенации (объединению) строк. Для создания строковых массивов следует явно использовать функцию char и операнды указывать через запятую. Вывод строкового массива в консоли производится, как может показаться, по столбцам (хотя любой вектор в MATLAB определяется как вектор строка). Здесь проявляется ключевая особенность MATLAB – по умолчанию любой объект уже считается массивом. И любая строка в

Пример 4. Особенности целочисленной арифметики

```
>> b=50*int8(35)
b =
  127
>> whos b
  Name        Size                Bytes  Class     Attributes

  b           1x1                     1  int8
>> a=0.8*int8([1 2 3])
a =
    1    2    2
>> whos a
  Name        Size                Bytes  Class     Attributes

  a           1x3                     3  int8
>> int8(10)/int8(100)*20.5
ans =
    0
>> 20.5*int8(10)/int8(100)
ans =
    1
>> int8(10)+int16(2)
??? Error using ==> plus
Integers can only be combined with integers of the same class, or scalar
doubles.
```

Пример 5. Вещественная арифметика в MATLAB

```
>> a=8*single(5)
a =
   40
>> b=8+single(5)
b =
   13
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     4  single
  b           1x1                     4  single
>> cos(b)
ans =
   0.9074
>> realmax('single')
ans =
  3.4028e+38
>> 2*ans
ans =
   Inf
>>
```

MATLAB представляет собой вектор, поэтому команда char('ab','abc') будет определять двумерный массив, причём функция char для выравнивания длины векторов самостоятельно к первому вектору добавила символ пробела. В этом можно убедиться, принудительно обратившись к элементу массива командой c(1,3), и в результате получить пустую строку. Вызов функции char с числовым аргументом возвращает символ, соответствующий заданному числу в ASCII таблице.

Логический тип данных – logical присваивается переменным в результате выполнения различных логических операций и специальных функций. Любая переменная логического типа занимает один байт и принимает значение либо ложь (логический 0) либо истина (логическая единица). В MATLAB существует множество функций для проверки типа переменной, результатом которых будет массив с размерностью, равной размерности аргумента, или число. Для просмотра документации по функциям провер-

Пример 6. Особенности определения строковых типов в MATLAB

```
>> b=['ab' 'abcd']
b =
ababcd
>> c=char('ab','abc')
c =
ab
abc
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x2                   4  char
  b           1x6                  12  char
  c           2x3                  12  char
>> c(1,3)
ans =
>> char(67)
ans =
C
```

ки типов данных можно набрать в командном окне doc is* , для проверки на принадлежность к одному из базовых типов данных можно использовать функцию isa(name,'class'). Пример вызова подобных функций приведен в примере 7.

Пример 7. Создание данных логического типа

```
>> a=[1 2 3 4 5]
a =
     1     2     3     4     5
>> b=isprime(a) %проверка на простые числа
b =
     0     1     1     0     1
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x5                  40  double
  b           1x5                   5  logical
>> c=[2 2 3 1 5]
c =
     2     2     3     1     5
>> a= =c
ans =
     0     1     1     0     1
>> a~=c
ans =
     1     0     0     1     0
```

Для создания числовых массивов с заданным типом данных можно использовать другой метод. При создании массива можно явно указать тип переменных – среди аргументов функции по созданию массивов нужно указать строку с именем числового типа данных. Следует осторожно использовать эту возможность, особенно для целочисленных типов данных. Значительная часть матричных функций MATLAB ограниченно поддерживает целочисленные аргументы.

При использовании числовых типов данных в MATLAB существует еще одна особенность. Дело в том, что во многих приложениях встречаются матрицы специфического содержания – значительная часть этих матриц заполнена нулями. В математике такие матрицы принято называть разреженными – sparse matrix. Понятно, что хранить все нулевые значения для таких массивов не эффективно. Специальные алгоритмы сохраняют

Пример 8. Явное указание числовых типов данных при создании массивов

```
1  >> a=rand(5,'single')
2  a =
3      0.7577    0.7060    0.8235    0.4387    0.4898
4      0.7431    0.0318    0.6948    0.3816    0.4456
5      0.3922    0.2769    0.3171    0.7655    0.6463
6      0.6555    0.0462    0.9502    0.7952    0.7094
7      0.1712    0.0971    0.0344    0.1869    0.7547
8  >> b=eye(5,'uint8')
9  b =
10     1     0     0     0     0
11     0     1     0     0     0
12     0     0     1     0     0
13     0     0     0     1     0
14     0     0     0     0     1
15 >> rand(5,'uint8')
16 ??? Error using ==> rand
17 Outputs of class uint8 not supported.
18 >> whos
19   Name        Size              Bytes  Class     Attributes
20
21   a           5x5                 100  single
22   b           5x5                  25  uint8
```

для разреженных матриц только адреса ненулевых элементов и их значения. Существуют специальные функции для создания разреженных матриц. Обычно имена этих функций соответствуют именам для создания обычных матриц, но с префиксом sp-. Например, единичная квадратная матрица размером n создается функцией eye(n), а соответствующая разреженная матрица – функцией speye(n). Преобразование числового массива в разреженный осуществляется функцией sparse, обратное преобразование – функцией full. Прямое преобразование возможно только с данными типа double и, как ни странно, logical. Причём преобразование к разреженному массиву логического массива приводит к странным результатам. Обратите внимание в примере 9 на формирование логического массива c и преобразование его в разреженный. Исходный массив занимает 25 байт, разреженный, будучи по-прежнему логическим, занимает уже 228 байт.

Тип данных struct аналогичен традиционным таблицам и обычно переводится словами "структура" или "запись". Для формирования таких типов данных есть два основных варианта. Первый заключается в использовании функции struct. Аргументами этой функции выступают названия полей структуры и соответствующие значения. Названия полей обязаны быть строковыми переменными, а значения в этих полях могут иметь любой допустимый в MATLAB тип. Второй вариант создания структуры состоит в использовании оператора присваивания. Для обращения к элементам структуры используется команда name.field, возвращающая значение поля field структуры name. Если такая структура существует, то значение будет выведено в консоль. При отсутствии такой структуры она автоматически создаётся (строка 14 пример 10). Обратите внимание, что системная функция whos выводит в консоль только информацию о самой структуре, но не выводит ни поля, ни содержимое этих полей. Для просмотра

Пример 9. Создание разреженных массивов в MATLAB

```
>> a=eye(5)
a =
     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
>> b=sparse(a)
b =
   (1,1)         1
   (2,2)         1
   (3,3)         1
   (4,4)         1
   (5,5)         1
>> whos
  Name        Size                Bytes  Class     Attributes

  a           5x5                   200  double
  b           5x5                   128  double    sparse
>> a=eye(5);
>> b=ones(5);
>> c=ne(a,b)
c =
     0     1     1     1     1
     1     0     1     1     1
     1     1     0     1     1
     1     1     1     0     1
     1     1     1     1     0
>> d=sparse(c)
>> whos
  Name        Size                Bytes  Class     Attributes

  a           5x5                   200  double
  b           5x5                   200  double
  c           5x5                    25  logical
  d           5x5                   228  logical   sparse
```

этой информации следует или обращаться специально к отдельным полям структуры, или использовать окно Variable Editor.

Также обратите внимание, что структура в памяти занимает больше места, чем суммарный размер её полей, – понятно, что в памяти хранятся имена полей, а каждое имя поля – это переменная типа char, и каждый символ в этом имени занимает по два байта. Возможно также создавать и массивы структур (одномерные и двумерные). Для создания таких массивов после имени структуры перед точкой в круглых скобках указывается индекс. Одномерный массив структур будет выглядеть как таблицы, в примере 11 создаётся таблица из 2 строк и 2 столбцов. Каждая структура при этом размещается в строке. Обратите внимание, что массив из двух структур занял в памяти меньше места, чем две структуры по отдельности. Структуры очень часто используются во встроенных и внешних функциях MATLAB. Этот тип данных чрезвычайно удобен для генерирования отчетов и упорядочивания большого количества схожей информации.

Более сложным типом данных MATLAB является тип cell (ячейка) и соответствующие массивы. Массивы ячеек удобны для структурирования разнородной информации. Для доступа к данным, хранящимся в структуре, необходимо знать названия полей структуры, тогда как для доступа к данным массива ячеек нужны только индексы, указывающие на конкретную ячейку. Самый простой вариант создания ячейки состоит в использо-

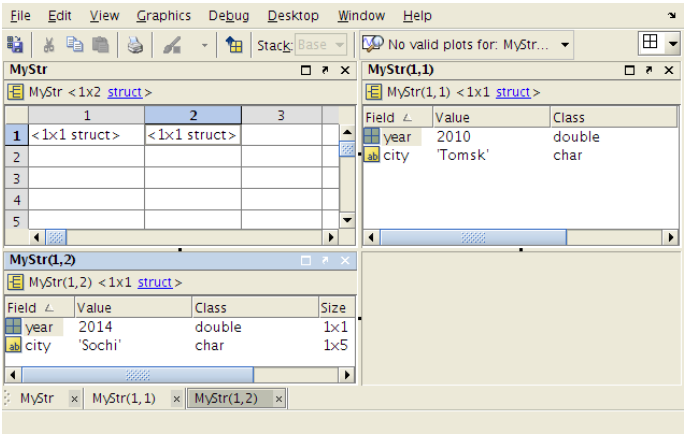


Рис. 9. Использование редактора массивов для просмотра структур

Пример 10. Создание переменных типа struct

```
>> MyStr=struct('year',2010,'city','Tomsk')
MyStr =
    year: 2010
    city: 'Tomsk'
>> MyStr=struct('year',2010,city,'Tomsk')
??? Undefined function or variable 'city'.
>> whos MyStr
  Name        Size              Bytes  Class     Attributes

  MyStr       1x1                 370  struct
>> MyStr.year
ans =
        2010
>> s.surname='ivanov'
s =
    surname: 'ivanov'
```

Пример 11. Создание переменных типа struct

```
>> MyStr=struct('year',2010,'city','Tomsk')
>> MyStr(1)=struct('year',2010,'city','Tomsk')
MyStr =
    year: 2010
    city: 'Tomsk'
>> MyStr(2)=struct('year',2014,'city','Sochi')
MyStr =
1x2 struct array with fields:
    year
    city
>> whos MyStr
  Name        Size              Bytes  Class     Attributes

  MyStr       1x2                 612  struct
```

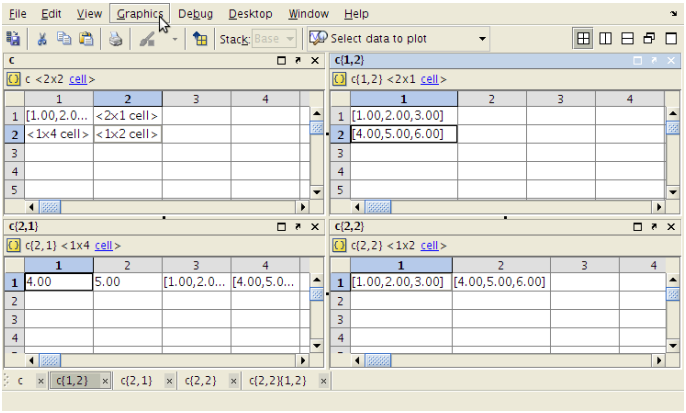


Рис. 10. Использование редактора массивов для просмотра ячеек

Пример 12. Создание переменных типа cell

```
>> a={[1 2 3],[4 5 6]}
a =
    [1x3 double]    [1x3 double]
>> b={[1 2 3];[4 5 6]}
b =
    [1x3 double]
    [1x3 double]
>> c={[1 2 3];[4 5 a]}
>> c={[1 2 3],b;[4 5 a],a}
c =
    [1x3 double]    {2x1 cell}
    {1x4 cell  }    {1x2 cell}
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x2                 272  cell
  b           2x1                 272  cell
  c           2x2                1528  cell
```

вании фигурных скобок. Элементы одной строки отделяются друг от друга запятой, переход на новую строчку производится точкой с запятой. Массивы ячеек в этом отношении аналогичны матрицам, но без ограничений на типы данных, находящихся внутри ячеек. В ячейке может находиться структура или массив ячеек. Для удобного просмотра содержимого массивов следует использовать окно Variable Editor. Тип данных cell удобен для организации массивов данных, когда заранее не известен тип этих данных или известно, что данные разнородны.

Тип данных function handle является указателем на функцию MATLAB. Объявить этот тип данных можно с помощью оператора @ или специальной функции str2func('function name'). Указатели на функции могут быть очень удобны во многих программах MATLAB – особенно синтаксис с оператором @. Это достаточно простой, краткий и мощный способ сформировать в программном коде новую функцию, а затем вызывать её из других функций в

Пример 13. Создание переменных типа function handle

```
>> MySqr=@(x)[x;x.^2]
MySqr =
    @(x)[x;x.^2]
>> MySqr([1 2 3])
ans =
     1     2     3
     1     4     9
>> whos
  Name         Size              Bytes  Class                Attributes

  MySqr        1x1                  32  function_handle
  ans          2x3                  48  double
>> sqrt(MySqr([1 2 3]))
ans =
    1.0000    1.4142    1.7321
    1.0000    2.0000    3.0000
>> a=[@sin @cos]
??? Error using ==> horzcat
Nonscalar arrays of function handles are not allowed; use cell arrays
instead.
```

Пример 14. Создание указателя на существующую функцию sqr.m

```
исходный код функции
function f = sqr(x)
f=x.^2;
создание указателя
>> f=@sqr
f =
    @sqr
>> f([1 2 3])
ans =
     1     4     9
```

качестве аргумента (строка 14, пример 13). В примере 13 показан вариант создания указателя на функцию, возвращающую массив из входного вектора и его квадрата. Если в текущем каталоге содержится функция sqr.m, то указатель на эту функцию можно сформировать так, как показано в примере 14. Обратите внимание, что явное создание массивов указателей не допускается – строка 18 и далее примера 13

3.4. Работа в редакторе .m файлов

Файлы с исходными кодами языка MATLAB имеют расширение .m и могут создаваться в любом текстовом редакторе. Многие из них обладают специальными возможностями по набору программ – подсветка синтаксиса, автодополнение команд, фолдинг отдельных фрагментов кода. При использовании сторонних текстовых редакторов не будет возможности запускать на выполнение MATLAB программу, выполнять её отладку, получать быстрый доступ к разделам справки и др.

Для набора исходных кодов в MATLAB существует специальное приложение с собственным графическим интерфейсом. В прошлых версиях можно было запустить этот редактор независимо от MATLAB. В таком режиме работы у этого редактора нет никаких преимуществ по сравнению со сторонними текстовыми редакторами. В текущей версии встроенный редактор можно запустить только из интерфейса MATLAB. Проще всего для

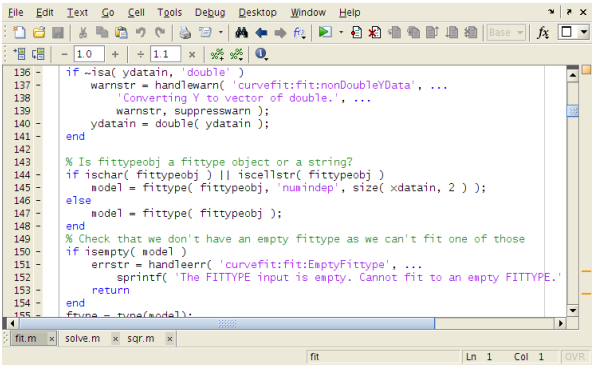


Рис. 11. Окно редактора программ. Открыть можно сразу много файлов. По умолчанию включен режим, когда виден только один, а навигации между файлами осуществляется через нижнюю панель. Расположение всех файлов можно настраивать выпадающим списком в правом верхнем углу.

этого кликнуть на кнопке New Script главной панели инструментов. Среди особенностей редактора следует отметить режим Cell Mode, позволяющий весь исходный код разбить на ячейки (cell), каждая из которых может выполняться или находиться в процессе отладки независимо от другой. На главной панели инструментов редактора также доступна кнопка вызова Function Browser. Внутри кода можно писать комментарии, как однострочные (начинаются с символа %, так и многострочного. В последнем случае комментарий должен начинаться с символов %{ и заканчиваться символами %}. Все символы, стоящие в исходном коде правее указанных символов комментариев, не обрабатываются интерпретатором MATLAB. Особую роль играет символ %% – с него начинается новая ячейка программного кода, в случае если включен режим Cell Mode. Еще одна экстравагантная возможность включить комментарии связана с символом многоточия (три и более точки подряд), который производит перевод на новую строку. Необходимость такого символа связана с ограниченностью видимой части строки. Понятно, что все символы правее многоточия интерпретатором MATLAB также не рассматриваются. Первые строки комментариев в программах MATLAB играют дополнительную важную роль – содержание этих строк, вплоть до первой пустой строки, считается справочной информацией по данному коду и выводится интерпретатором MATLAB командой help.

Для подавления вывода результатов вычислений, кроме символа точки

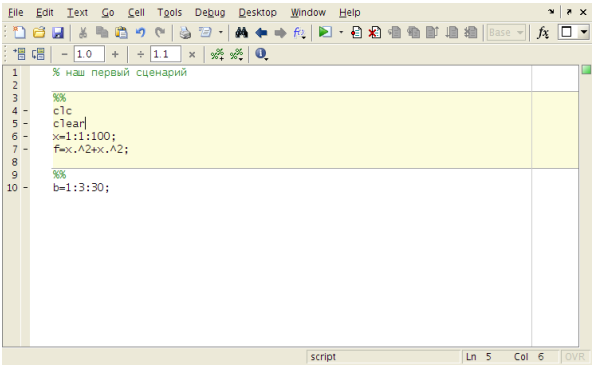


Рис. 12. Окно редактора программ. Включена опция Cell Mode – выполнение кода в каждой ячейке независит от другой. Активная в данный момент ячейка подсвечивается.

Пример 15. Простейший сценарий

```
% наш первый сценарий

clear x f
clc
x=1:1:100;
f=x.^2;
```

с запятой в конце каждого оператора MATLAB, можно использовать команду echo. Команда echo off в исходных кодах подавляет вывод в командное окно MATLAB результатов вычислений, при этом каждый оператор не обязательно заканчивать символом точки с запятой. Для переключения на режим по умолчанию следует использовать echo on .

Все исходные коды на языке MATLAB можно разделить на два типа:

1. Сценарии, или скрипты (script);
2. Функции (function).

Простейший способ написать программу на языке MATLAB состоит в использовании сценариев. Сценарий не имеет входных аргументов. Недостаток сценариев заключается в том, что все переменные внутри программы будут находиться внутри рабочего пространства MATLAB – все переменные являются глобальными. Для организации сложных алгоритмов этот вариант нецелесообразен. Рассмотрим простейший пример сценария. В MATLAB отсутствует встроенная функция для возведения в квадрат, создадим вначале сценарий, возвращающий в рабочее пространство вектор, каждый элемент которого равен квадрату соответствующего элемента исходного вектора. Обратите внимание на полезные мелочи при работе с редактором: если вы забудете поставить точку с запятой для подавления

Пример 16. Определение функции в исходном коде

```
function [out1, out2, ...] = myfun(in1, in2, ...)
```

Пример 17. Простейшая MATLAB функция

```
% наша первая функция
function [y,f] = sqr(x)
y=x;
f=x.^2;
whos
```

вывода выражения, то возникнет предупреждение с подсказкой сочетания клавиш и специальной кнопкой для исправления ситуации.

Если набрать этот код в текстовом редакторе и нажать клавишу F5, то сценарий будет выполнен, и в рабочем пространстве можно будет увидеть две переменные. Если до запуска файл исходного кода не был сохранен, то MATLAB предложит сделать это, а перед каждым следующим запуском файл будет перезаписываться. По умолчанию включена дополнительная опция работы с редактором, которая производит автосохранение файла программы каждые 5 минут, причём автоматически сохраняемый файл имеет расширение .m и по умолчанию является скрытым. В окне Current Folder этот файл виден независимо от настроек операционной системы.

Будьте внимательны – при работе в режиме Cell Mode выполнение каждой ячейки по сочетанию клавиш Ctrl + Enter не приводит к перезаписыванию файла программы. Если созданный сценарий сохранить с именем sqr.m в текущем каталоге, то для его выполнения в дальнейшем можно просто в командном окне набрать команду sqr. Недостаток сценариев очевиден – чтобы выполнить программу для других значений входных данных, нужно вносить изменения в исходный код. Вызывать сценарии из других программ также неэффективно, поскольку все переменные сценария являются глобальными. Значительно большие возможности по написанию программ представляют функции MATLAB. Ключевым отличием является то, что все переменные внутри функции являются локальными (за исключением входных и выходных данных функции, которые должны передаваться в функцию из внешней программы и обратно). Синтаксис определения функции в MATLAB таков: Ключевое имя function указывает, что в файле содержится именно функция, а не сценарий, myfun – имя функции. После имени в круглых скобках стоят входные данные – в качестве входных данных может выступать число, несколько чисел, векторы, матрицы и более сложные конструкции. После объявления функции слева от знака равенства стоят выходные данные. Для создания простейшей собственной функции этого вполне достаточно. Синтаксис вызова этой функции следующий [x,f] = sqr(x). Понятно, что перед вызовом необходимо определить вектор входных значений x, для удобства можно ввести в консоли следующее: x=[1 2 3 4 5]. В результате вызова функции в рабочем пространстве

Пример 18. Результат вызова функции [x,f] = sqr(x)

```
>> [x,f]=sqr(x)
  Name        Size                Bytes  Class     Attributes
  f           1x5                    40  double
  x           1x5                    40  double
  y           1x5                    40  double

x =
     1     2     3     4     5
f =
     1     4     9    16    25
```

появятся две переменных. Обратите внимание, что объявленная внутри функции новая переменная y, будучи локальной, не была передана в рабочее пространство. Команда whos выводит в консоли информацию не о всех переменных рабочего пространства, а только о тех, которые определены внутри функции. Если необходимо сделать какую-то переменную внутри функции глобальной, то её просто следует объявить с ключевым именем global. При этом переменная по-прежнему будет отсутствовать в рабочем пространстве, но будет доступна вне кода данной функции, например в качестве входных данных других функций. Независимо от того, сценарий или функция определены внутри файла с расширением .m при вводе имени этого файла в командном окне, находящаяся внутри него программа выполняется. Важно понимать, какие действия производит интерпретатор MATLAB при вводе команды. Последовательность действий такова:

1. По введенному имени производится поиск переменной в рабочем пространстве MATLAB. Если переменная найдена, то в консоли выводится её значение;
2. Если переменная не найдена, то производится поиск среди встроенных функций MATLAB. Список этих функций можно посмотреть, набрав в консоли команду help elfun. Если встроенная функция найдена, то производится её вычисление.
3. Если не найдена ни переменная в рабочей области, ни встроенная функция, то MATLAB производит поиск файла с таким именем сначала в текущем каталоге, а потом во всех каталогах, включенных в переменную Path MATLAB. Если файл будет найден, то он выполняется.

Обратите внимание, что MATLAB не запрещает назвать файл, содержащий сценарий или функцию именем, совпадающим с именем встроенной функции. Однако при выполнении такого сценария в командное окно будет выведена рекомендация
Warning: Function ... has the same name as a MATLAB builtin.We suggest you rename the function to avoid a potential name conflict.
Исходный код встроенных функций находится в вычислительном ядре MATLAB, и просмотреть этот код нет возможности. Остальные функции

называются внешними (по отношению к ядру), их исходный код можно посмотреть командой edit name, где name – интересующее вас имя функции. Если в качестве аргумента для edit ввести имя несуществующей функции, то MATLAB выведет в специальном окне предупреждение с предложением создать такую функцию. Специальный оператор MATLAB – exist позволяет выяснить, используется имя функции или нет. В качестве аргумента этого оператора следует указать имя интересующей функции. Результатом выполнения exist будет число.

Таблица 3. Результаты выполнения exist

Значение	Пояснение
0	Переменная с таким именем не существует в рабочем пространстве, файл с таким именем не существует ни в текущем каталоге. ни в путях поиска
1	Переменная с таким именем существует в рабочем пространстве
2	Файл с таким именем существует или в текущем каталоге, или в путях поиска
3	MEX-файл с таким именем существует в текущем каталоге или в путях поиска
4	MDL-файл (файл, содержащий модель SIMULINK) с таким именем существует или в текущем каталоге, или в путях поиска
5	Является встроенной функцией
6	P-файл с таким именем существует в текущем каталоге или в путях поиска
7	Является директорией, присутствующей в путях поиска
8	Является классом

При совпадении имён пользовательских файлов с именами встроенных и внешних функций предпочтение отдаётся пользовательским функциям. Причём существует ряд особенностей.

1. Если файл пользовательской функции или сценария расположен в текущем каталоге, то его приоритет будет даже выше, чем у встроенной функции.
2. Если файл пользовательской функции или сценария расположен в одном из каталогов, установленных в пути поиска, то приоритет при совпадении имен будет у встроенной функции, а внешняя функция будет менее приоритетной.

По умолчанию в путях поиска размещаются все каталоги начиная с папки /toolboxes/ корневой директории MATLAB. В этих каталогах содержатся .m файлы с исходными кодами соответствующих расширений MATLAB. При первом запуске MATLAB домашняя директория пользователя добавляется в этот список верхней строчкой. Значит, вначале поиск файлов происходит именно по этому каталогу. При добавлении новых путей поиска следует иметь в виду, что если добавлена директория /home/, то не обязательно добавляются её поддиректории. При добавлении только корневой папки поиск файлов будет происходить только в ней, не затрагивая вложенные папки.

3.5. Задания для самостоятельной работы

1. Настройте интерфейс MATLAB так, чтобы одновременно было видно пять окон – 4 стандартных и окно редактора программ.
2. Запустите MATLAB с опцией -nojvm и обычным образом. Проверьте средствами операционной системы, в каком случае MATLAB занимает меньше места в оперативной памяти? Насколько велико различие? Сравните приближенно скорость вычислений в обоих случаях.
3. Создайте два вектора вещественных чисел и вычислите двумерную функцию $\sin(x^2 + y)$. Постройте график этой функции, используя только средства графического интерфейса MATLAB.
4. Создайте вектор вещественных чисел, сохраните его на диск, закройте MATLAB. При следующем запуске MATLAB откройте файл данных, содержащий тот же вектор. Сделайте это, используя или только командное окно MATLAB, или только возможности графического интерфейса.
5. Введите в командное окно MATLAB команды, которые возвращали бы результаты – NaN, Inf, eps. В чём отличие Inf и NaN? Какие еще константы определены в языке MATLAB?
6. Можно ли встроенными средствами MATLAB вычислить число π с произвольной точностью? Попробуйте вывести это число с точностью 10, 16 и 30 знаков после запятой.
7. Запрограммируйте собственную функцию с именем cos, которая будет вычислять синус угла причём аргумент должен выражаться в радианах.
8. Напишите простой сценарий, создающий вектор из n вещественных чисел. Напишите подобную функцию, где n будет входным параметром (n выбирайте в зависимости от быстродействия вашего компьютера). Сравните скорость вычислений сценария и функции.

4. Визуализация данных

4.1. Построение графиков от функции одной переменной

Для начала рассмотрим построение графиков функций одной переменной в разных системах координат. Чтобы построить график можно сформировать вектор значений независимой переменной и потом вычислить функцию от этого вектора. Например, построим график синусоиды на отрезке $[0,2\pi]$. Весь интервал разобьем на 100 точек, для этого вектор значений независимой переменной следует задать в виде x=[0:2*pi/99:2*pi], а функцию в виде – y=sin(x). Чтобы получить график функции, можно выбрать один из следующих способов:

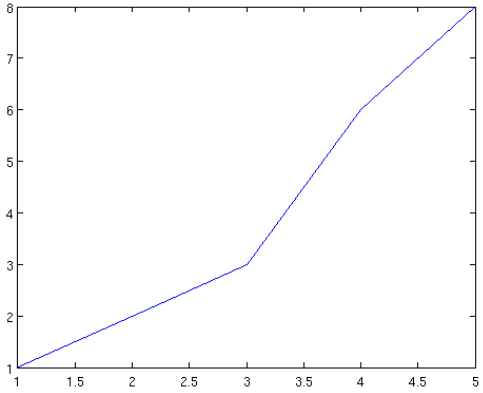


Рис. 13. Результат выполнения команды plot([1 2 3 6 8]) – график вектора, где в качестве абсцисс считаются индексы массива

1. Ввести в командном окне plot(x,y);
2. Выделить в окне Workspace переменные, щелкнуть правой кнопкой мыши и в появившемся меню выбрать ту же команду или после выделения щелкнуть кнопку на главной панели инструментов окна Workspace;
3. Ввести в консоли MATLAB команду plottools или нажать кнопку START в левом нижнем углу главного окна, потом выбрать MATLAB и нажать на Plot Tools.

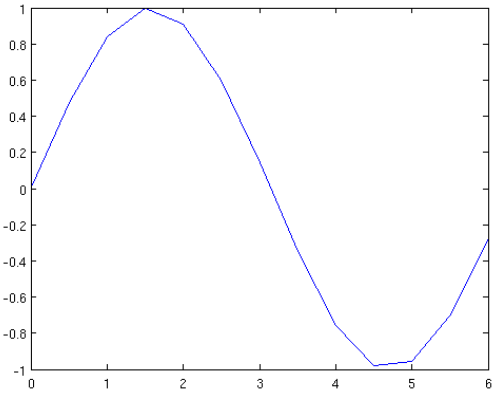


Рис. 14. График синусоиды с относительно небольшим числом точек. На графике видны изломы

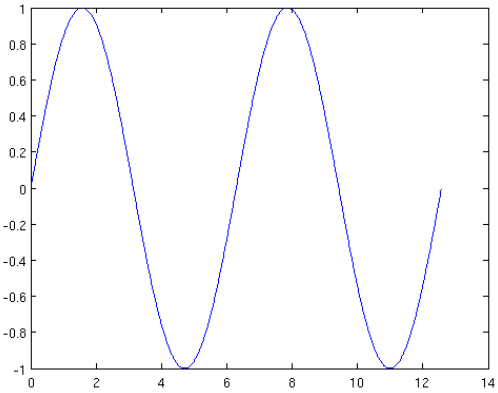


Рис. 15. График синусоиды при большом количестве элементов вектора

Два первых варианта приведут к появлению нового окна, в котором будет отрезок синусоиды. Попробуем похожим образом построить график более сложной функции – $\sin(x)\exp(-x)$. Значения независимой переменной оставим те же. Для новой функции в консоли MATLAB следует ввести `g=sin(x).*exp(x);`. Точка перед знаком умножения – признак поэлементного умножения. Если применить умножение простое (по терминологии

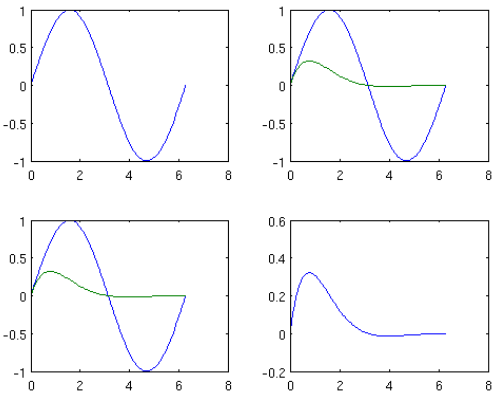


Рис. 16. Несколько графиков в одном окне. На второй и третьей позициях проиллюстрирован результат построения двух графиков на одних осях

MATLAB), то возникнет ошибка (??? Error using ==> mtimes Inner matrix dimensions must agree.). Обычное умножение MATLAB умеет делать с массивами, размеры которых согласуются. Попробуем изобразить оба графика в одном окне. В MATLAB существует два варианта подобного – в одном случае у каждого графика будут свои координатные оси, в другом оба графика будут с одними осями. Второй случай проще всего реализовать командой plot(x,f,x,g), Первый вариант оформляется командой subplot(n,m,pos), которая разбивает графическое окно на подобласти из n строк и m колонок, а pos указывает позицию, в которой следующая команда plot будет строить график. Другую возможность нарисовать несколько линий графиков на одних осях координат предоставляет команда hold on. Смысл функции в том, что она оставляет главное графическое окно и оси координат на месте при прорисовке следующего графика. По умолчанию каждое новое выполнение команды по построению графика приводит к построению нового графического окна. Для оформления графика прямо в тексте программы можно использовать функции title, xlabel, ylabel, legend – для отображения названия, подписи к горизонтальной, вертикальной осям и легенды соответственно. Синтаксис функций в простейшем случае допускает указание только одного аргумента – строки. Другие аргументы позволяют установить шрифт, расположение, заливку и прочее.

Построение графика функции, заданной параметрически, ничем не отличается от приведённого выше примера, только вначале нужно сформировать вектор значений параметра, а потом векторы двух функций. Вы-

Пример 19. Построение нескольких графиков рядом друг с другом

```
x=[0:2*pi/99:2*pi];
f=sin(x);
g=sin(x).*exp(-x);
subplot(2,2,1);plot(x,f)
subplot(2,2,2);plot(x,f,x,g)
subplot(2,2,3);plot(x,f,x,g)
subplot(2,2,4);plot(x,g)
```

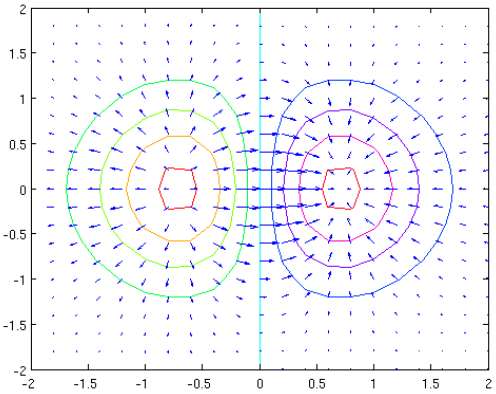


Рис. 17. Визуализация поля градиентов для функции $x \cdot exp(-x^2 - y^2)$. Дополнительно построены линии уровня той же функции

зывать функцию plot следует с обоими функциями в качестве аргументов.

Специальная функция в MATLAB строит графики в полярной системе координат – polar(phi,r), аргументы функции, как можно понять из названий, – угол и расстояние в полярной системе координат.

Интересную возможность построения графика функции реализует fplot. Для применения этой функции не нужно заранее готовить вектор значений аргументов и функции. В качестве первого входного параметра fplot принимает указатель на функцию или строку, содержащую имя функции, а вторым параметром является вектор из двух чисел – начальная и конечная точки интервала изменения абсциссы на графике. Например, для построения графика синусоиды нужно ввести команду – fplot(@sin,[0 2pi]).

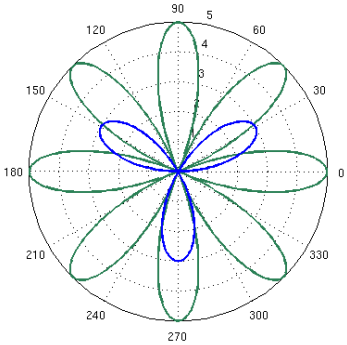


Рис. 18. Графики функций $\rho = 5\cos(4\varphi)$ и $\rho = 3\sin(3\varphi)$ в полярной системе координат

Пример 20. Использование функции polar для построения двух графиков в полярных координатах. Изменения цвета и толщины линий были произведены в редакторе plottools

```
t = 0:.01:2*pi;
x = 5*cos(4*t);
y = 3*sin(3*t);
polar(t,x);
hold on
polar(t,y);
```

4.2. Задания для самостоятельной работы

1. Если тело совершает колебания по двум взаимно перпендикулярным направлениям, то при определенных отношениях частот этих колебаний получаются линии, называемые фигуры Лиссажу. Постройте в одном окне 6 фигур Лиссажу, соответствующих отношениям частот – 1/2, 1/3, 2/3, 2/5, 3/5, 3/4. Можно ли в качестве заголовков к осям разместить выражние ω=... и вместо многоточия соответствующее отношение?
2. Выполните предыдущее задание, но вместо статичных графиков нарисуйте для каждой оси координат анимационную кривую.
3. Можно ли на одних координатных осях разместить несколько трехмерных графиков?
4. Постройте двумерный и трехмерный графики для функции, имеющей разрыв на интервале построения графиков.
5. Постройте график любой трехмерной функции так, чтобы на плоско-

сти XY были нарисованы линии уровня этой функции с подписями значений этой функции.

6. Выполните задания 1,2, 4 и 5 вначале в виде файла сценария, а потом с использованием plottools.
7. Функция fplot строит графики функций одной переменной. Создайте функцию, которая будет делать то же самое для трехмерных графиков.

5. Вычисления в MATLAB

5.1. Векторные и матричные операции

Ключевым объектом во всех вычислительных алгоритмах MATLAB является матрица и частный случай матрицы – вектор. Как уже несколько раз упоминалось, синтаксис определения вектора очень простой – внутри квадратных скобок определяются начальное значение вектора, шаг изменения, конечное значение. При этом шаг изменения указывать не обязательно – по умолчанию он считается равным единице. Шаг изменения может быть даже отрицательным, и тогда начальное значение должно быть меньше конечного. Обратите внимание, что практически во всех языках программирования для определения вектора используется цикл, причём счётчик цикла может принимать только целочисленные значения, и для формирования некоторых векторов приходится писать сложные алгоритмы. В MATLAB формирование любого вектора занимает одну строчку программного кода. Один элемент вектора от другого в случае прямого их указания можно отделять или запятой или пробелом. В качестве элементов вектора могут выступать и комплексные числа, но генерировать комплексные векторы при помощи оператора двоеточия не получится. В приведенном ниже примере обратите внимание на последнее введенное выражение – попытка генерировать комплексный вектор привела к сообщению об ошибке, и в результате был получен вещественный вектор, а мнимая часть комплексного числа была просто проигнорирована. По умолчанию все векторы становят-

Пример 21. Простейшие способы определения векторов в MATLAB

```
>> a=[1 2 3 4 5]
     1     2     3     4     5
>> a=[1:5]
a =
     1     2     3     4     5
>> a=[1:.25:2]
a =
    1.0000    1.2500    1.5000    1.7500    2.0000
>> a=[20:-2:1]
a =
    20    18    16    14    12    10     8     6     4     2
>> a=[2+2i:1:4+2i]
Warning: Colon operands must be real scalars.
a =
     2     3     4
>>
```

ся вектор-строками. Для преобразования их в вектор-столбец необходимо использовать операцию транспонирования. Благодаря тому, что массивы являются основой программирования MATLAB, транспонирование реализовано простым символом – апострофом. Определение его сразу после имени вектора приводит к преобразованию вектор-строки в вектор-столбец и наоборот. Для обращения к отдельному элементу вектора после его имени в

Пример 22. Обращение к элементам вектора

```
>> a=[1 4 9 16 25]
a =
     1     4     9    16    25
>> a'
ans =
     1
     4
     9
    16
    25
>> a(2)
ans =
     4
>> a(2:4)=0
a =
     1     0     0     0    25
```

Пример 23. Создание матриц в MATLAB

```
>> A=[11 22 33; 21 22 23; 31 32 33]
A =
    11    22    33
    21    22    23
    31    32    33
>> A(1:2,2:3)=0
A =
    11     0     0
    21     0     0
    31    32    33
>> A(3,3)
ans =
    33
```

круглых скобках нужно поставить номер элемента. Подобным же образом можно изменять значения отдельных элементов вектора.

Подобным образом в MATLAB генерируются и матрицы. Стоит только обратить внимание на то, что матрицы состоят из векторов. Матрицы задаются по строкам. Одна строка от другой отделяется символом точки с запятой.

В MATLAB существует большое число функций для генерации стандартных матриц.

Таблица 4. Наиболее важные функции для создания матриц стандартного вида

Название	Значение
zeros	Матрица, заполненная нулями
ones	Матрица, все элементы которой равны единице
eye	Единичная матрица
rand	Матрица, заполненная случайными числами

Любую функцию из этой таблицы можно вызвать как с одним аргументом, например zeros(n), так и с двумя – zeros(n,m). В первом случае будет создана квадратная матрица размерности n, во втором случае – прямоугольная матрица из n строк и m столбцов. Часто оказывается полезной функция diag(a), создающая диагональную матрицу, на главной диагонали которой расположены соответствующие элементы ветора a.

Для векторов и матриц определены естественные арифметические операции – сложение, вычитание и умножение на число. Особенностью MATLAB является наличие специальных операторов, реализующих поэлементные арифметические операции, при этом элементы нового массива получаются из соответствующих элементов массивов операндов.

Таблица 5. Арифметическит операторы в MATLAB

Название	Значение
+/-	Сложение/вычитание массивов одинаковой размерности. Один из операндов может быть числом, и тогда происходит сложение/вычитание этого числа с каждым элементом массива второго слагаемого
*	Умножение массивов одинаковой размерности. Один из операндов может быть просто числом, и тогда это число умножается на каждый элемент массива второго множителя
/	Поэлементное деление массива на число. Если оба операнда матрицы, то автоматически считается матрица, обратная к той, что стоит в знаменателе
\	Если A – квадратная матрица, то A\B = inv(A)*B. Если A квадратная матрица размерности n, а B – вектор-столбец из n чисел, то X=A\B равен вектору решения системы линейных уравнений AX=B
.*	Поэлементное умножение массивов одинакового размера
./	Поэлементное деление массивов одинаковых размеров. Операция C=A./B приводит к C(i,j)=A(i,j)/B(i,j)
.\	Поэлементное левое деление массивов одинаковой размерности. Операция C=A.\B приводит к C(i,j)=B(i,j)/A(i,j)
.^	Поэлементное возведение в степень массивов одинаковой размерности. Основанием степени являются элементы первого массива, а показателями степени – элементы второго массива. Операция C=A.^ B приводит к C(i,j)=A(i,j)^ B(i,j)
'	Вычисление комплексно-сопряженного массива
.'	Транспонирование массива

Использование векторов и матриц в MATLAB пронизывает практически

все алгоритмы. Например, для решения уравнений соответствующий многочлен определяется как вектор коэффициентов при степенях переменной. Кроме того, значительное число функций для работы с векторами применимы и для строк. Такая классическая конструкция программирования, как цикл в MATLAB, легко реализуется с помощью оператора двоеточия, который создаёт именно вектор.

Таблица 6. Операции с матрицами и векторами

Формат вызова	Пояснение
length(a)	Вычисляет длину вектора
inv(A)	Обращение квадратной матрицы
rank(A)	Вычисляет ранг матрицы
trace(A)	Вычисляет след матрицы
size(A) size(1,A) size(2,A)	Возвращает размер массива. первый вариант возвращает вектор из двух чисел – первое число строк, второе – число столбцов матрицы. Для векторов синтаксис функции точно такой же. Второй и третий вариант функции возвращает только число, равное количеству строк или столбцов
norm(a)	Вычисляет евклидову норму вектора
dot(a,b)	Вычисляет скалярное произведение векторов
cross(a,b)	Вычисляет векторное произведение векторов. результат функции – вектор, такой же размерности, как векторы-операнды
A(:,j)	Возвращает j-й столбец матрицы A
A(:,j1:j2)	Возвращает столбцы матрицы с j1 по j2
A(:,j1:h:j2)	Возвращает столбцы матрицы с j1 по j2 с шагом h
A(i1:i2,j1:j2)	Возвращает подматрицу с номерами строк с i1 по i2 и номерами столбцов с j1 по j2
tril(A)	Возвращает нижний треугольник матрицы, включая диагональ. При этом все элементы исходной матрицы, стоящие выше главной диагонали, приравниваются нулю.
triu(A)	Возвращает верхний треугольник матрицы включая диагональ. При этом все элементы исходной матрицы, стоящие ниже главной диагонали, приравниваются нулю.
diag(a)	Возвращает матрицу, на главной диагонали которой стоят элементы вектора a

Пример 24. Определение комплексных чисел

```
>> z=2+5*i
z =
   2.0000 + 5.0000i
>> z=2+5*j
z =
   2.0000 + 5.0000i
>> i=2
i =
     2
>> z=2+5*i
z =
    12
>> z=2+5i
z =
   2.0000 + 5.0000i
>> z=complex(2,5)
z =
   2.0000 + 5.0000i
```

diag(A)	возвращает вектор, состоящий из элементов, стоящих на главной диагонали матрицы A

5.2. Комплексные числа

Определение комплексных чисел в MATLAB соответствует их естественному определению – для указания мнимой единицы можно использовать как букву i, так и j. До тех пор, пока в рабочем пространстве не определены переменные с такими именами, конструкция вида a+b*i или a+b*j будет определять комплексное число. При этом можно обратить внимание, что независимо от того, в каком виде будет представлена мнимая единица – i или j э вывод на консоль будет использовать именно символ i. Если существует переменная с именем i или j, то для определения комплексного числа можно использовать трюк – ввести a+bi. Пропуск оператора умножения будет воспринят как признак мнимой части. Для создания комплексных переменных существует также встроенная функция complex(a,b).

Комплексные числа в MATLAB никак специально не объявляются. Если результатом вычисления является комплексное число, то MATLAB самостоятельно определит тип результата как double(complex). Легко догадаться, что как вещественная, так и мнимая часть имеют тип double, и потому комплексное число занимает в памяти 16 байт. Вещественную и мнимую часть комплексного числа возвращают соответственно функции Re и Im. Для быстрого создания векторов из комплексных чисел следует предварительно создать два вектора для действительной и мнимой частей, а потом вектор для мнимой части умножить на мнимую единицу. Сложение результатов вычислений приведёт к комплексному вектору. По аналогии можно создать и комплексные матрицы.

Пример 25. Определение многочлена в MATLAB и переключение между различными вариантами представления

```
>> poly=[5 -2 1]
poly =
     3     1

>> poly=[5 -2 1]
poly =
     5    -2     1

>> poly_sym=poly2sym(poly)
poly_sym =
5*x^2 - 2*x + 1
>> sym2poly(poly_sym)
ans =
     5    -2     1
>> whos
  Name            Size                Bytes  Class                Attributes

  poly            1x3                    24  double
  poly_sym        1x1                   112  sym
```

5.3. Решение уравнений и систем уравнений

Для решения уравнений в MATLAB существует множество функций. Решение одного уравнения можно представить как поиск нулей функции, стоящей в левой части уравнения. В частном случае в левой части уравнения стоит алгебраический многочлен n-ой степени. Как уже упоминалось выше, многочлен в MATLAB представляется вектором коэффициентов при степенях переменной. Рассмотрим пример – определим в MATLAB многочлен $5x^2-2x+1$, для этого в консоли нужно ввести poly=[5 -2 1]. Обратите внимание, что при использовании символа пробела в качестве разделителя элементов вектора знак перед коэффициентом нужно указывать без пробела – иначе MATLAB этот знак будет считать признаком арифметической операции. Коэффициенты при степенях переменных следует вводить в порядке понижения степени. Определение многочлена в виде вектора характерно для MATLAB и обеспечивает краткость программных кодов и эффективность вычислений, однако не обеспечивает наглядности, привычной для естественной записи многочлена. Для переключения между MATLAB формой полинома и естественной записью служат две функции – poly2sym и sym2poly. Ключевое слово sym указывает на объект специфического класса. Класс sym позволяет в MATLAB производит символьные вычисления. Объекты из этого класса чаще всего выводятся на экран в виде, максимально приближенном к естественной математической записи. Функция poly2sym преобразует входной вектор коэффициентов многочлена в привычную запись, а sym2poly делает обратное преобразование. Обратите внимание, что эти функции не только изменяют формат вывода, но и содержимое переменной.

Для поиска корней алгебраического полинома следует пользоваться функцией roots(polynom). Причем в качестве входного многочлена нужно указывать его в виде именно вектора. Поиск корней многочленов, пред-

Пример 26. Поиск корней многочлена

```
>> poly=[5 -2 1]
poly =
     5    -2     1
>> roots(poly)
ans =
   0.2000 + 0.4000i
   0.2000 - 0.4000i
```

Пример 27. Поиск нулей функции

```
>> f=@(x) sin(x)+x.^2.*cos(x)
f =
    @(x)sin(x)+x.^2.*cos(x)
>> fplot(f,[-6 6])
 grid on %включаем линии сетки на графике
>> myzero1=fzero(f,[4 6])
myzero1 =
    4.7566
>> myzero1=fzero(f,5)
myzero1 =
    4.7566
>> format longEng
>> myzero1
myzero1 =
    4.75655940570290e+000
>>
>> myzero1=fzero(f,[3 4])
??? Error using ==> fzero at 283
The function values at the interval endpoints must differ in sign.
>> [myzero1,f]=fzero(f,5)
myzero1 =
    4.75655940570290e+000
f =
   -2.66453525910038e-015
>>
```

ставленных в символьном виде, также возможен, но с использованием дополнительного расширения – Symbolic Math Toolbox. Вызов roots с аргументом, отличным от алгебраического полинома, приводит к ошибке.

Еще одна функция для поиска нулей функций – fzero работает с любой функцией MATLAB в качестве аргумента. В качестве опций указывается или интервал для поиска корней, или начальное условие. Недостатком функции fzero является необходимость знать начальное приближенное значение нуля функции. Классическая рекомендация состоит в том, что до использования fzero следует построить график функции. Следует иметь в виду, что поиск нулей данная функция осуществляет по признаку изменения знака, и, следовательно найти точки, в которых график функции касается оси абсцисс, этой функцией нельзя.

После построения графика функции $\sin(x) + x^2\cos(x)$ можно увидеть, чему равны нули функции приближенно. Синтаксис вызова функции допускает указание или начального приближения, или интервала, на котором ищется значение. Если на указанном интервале не существует нулей функции, то будет выведено соответствующее предупреждение (строка 18, пример 27). Вызов функции fzero с двумя аргументами вычисляет не только значение аргумента, но и значение функции в этой точке. Определение дополнительных опций для функции fzero производится с помощью функции optimset, аргументом которой служит последовательность вида 'name',

Пример 28. Вывод дополнительной информации при поиске нулей функции

```
>> options = optimset('Display','iter','TolX',1e-3);
>> [myzero1,fval]=fzero(f,[4 5],options)
 Func-count     x          f(x)             Procedure
     2          5          6.13263          initial
     3          4.64649    -2.41959         interpolation
     4          4.7465     -0.230991        interpolation
     5          4.75668    0.00269545       interpolation
Zero found in the interval [4, 5]
myzero1 =
    4.75667625292196e+000
fval =
    2.69544610098871e-003
>> options = optimset('Display','iter','TolX',1e-6);
>> [myzero1,fval]=fzero(f,[4 5],options)
 Func-count     x          f(x)             Procedure
     2          5          6.13263          initial
     3          4.64649    -2.41959         interpolation
     4          4.7465     -0.230991        interpolation
     5          4.75668    0.00269545       interpolation
     6          4.75656    -1.13002e-05     interpolation
     7          4.75656    -1.13002e-05     interpolation
Zero found in the interval [4, 5]
myzero1 =
    4.75655891581512e+000
fval =
   -11.3002431273435e-006
```

value – название опции в апострофах и ее значение. Сама функция optimset возвращает структуру. В следующем примере указаны всего две специальные опции – Display, для вывода информации на каждом этапе вычислений и TolX – относительная погрешность вычисления по аргументу. В первом случае для поиска корня задается относительная погрешность одна тысячная, и процесс поиска сходится уже на 4 итерации. Во втором примере погрешность задана равной одной миллионной, и для поиска корня потребовалось 6 шагов.

Для поиска локальных экстремумов функций используются функции fminbnd и fminsearch. Синтаксис этих функций очень похож на fzero. Для получения подробной информации достаточно посмотреть фирменную документацию по указанным функциям поиска локальных экстремумов. Для поиска максимума в MATLAB не предусмотрена специальная функция – предполагается, что максимум любой функции может быть вычислен с помощью минимума соответствующей функции противоположного знака. Решение систем линейных уравнений является одной из основных задач всей вычислительной математики. В MATLAB решить эту задачу можно с помощью специального оператора – левого деления матриц или с помощью функции linsolve. В любом случае аргументами или операндами являются матрица коэффициентов при переменных и вектор правых частей уравнений. Размерности соответствующих массивов должны совпадать. Рассмотрим пример решения системы из 5 уравнений с 5 неизвестными:

$$2x_1 + 4x_2 + 0x_3 + 2x_4 + 4x_5 = 1;$$

$$3x_1 - 2x_2 + 1x_3 + 3x_4 + 1x_5 = 2;$$

$$2x_1 + 1x_2 + 4x_3 + 2x_4 + 3x_5 = 3;$$

Пример 29. Решение системы линейных уравнений

```
>> A=[2 4 0 2 4; 3 -2 1 3 1; 2 1 4 2 3; 1 3 2 -1 1; 2 3 -2 4 1]
A =
     2     4     0     2     4
     3    -2     1     3     1
     2     1     4     2     3
     1     3     2    -1     1
     2     3    -2     4     1
>> B=[1 2 3 4 5]
B =
     1     2     3     4     5
>> A\B'
ans =
    1.0388
    1.2284
    0.9526
    0.7543
   -1.8750
>> linsolve(A,B)
??? Error using ==> linsolve
Matrix dimensions must agree.

>> linsolve(A,B')
ans =
    1.0388
    1.2284
    0.9526
    0.7543
   -1.8750
```

$$1x_1 + 3x_2 + 2x_3 - 1x_4 + 1x_5 = 4;$$

$$2x_1 + 3x_2 - 2x_3 + 4x_4 + 1x_5 = 5/$$

Пример 29 иллюстрирует решение системы линейных уравнений. Матрицу коэффициентов и вектор-столбец правой части не обязательно вводить по отдельности. Можно также использовать Variable Editor для ввода массивов.

5.4. Интерполяция

Как известно, задача интерполяции сводится к построению достаточно простой функции, которая приближенно равна некоторым значениям y в соответствующих точках x. В MATLAB для интерполяции в качестве входных данных строятся два числовых вектора. Самый простейший вариант интерполяции – использование полиномов невысоких степеней. Функция polyfit по заданным значениям входных векторов возвращает вектор алгебраического полинома указанной степени. Для построения графика интерполяционного многочлена, кроме его коэффициентов, нужно еще вычислить значения в точках некоторого интервала. В MATLAB есть функция polyval, которая по заданному многочлену и вектору аргументов вычисляет его значения. Для сравнения исходных данных и результата интерполяции построим на тех же осях координат график исходного массива в виде маркеров без линии. Соответствующий сценарий приведен в примере **??**

Достаточно хорошие результаты интерполяции таблично заданных функций обеспечивают сплайны. Синтаксис вызова функции следующий:

Пример 30. Интерполяция алгебраическими полиномами

```
x=[0.5 0.01 0.17 0.25 0.3 0.36];
y=[0.05 0.1003 0.1717 0.2553 0.3093 0.3764];
t=[-.1:0.02:0.5];
plot(x,y,'k*','markersize',12);
hold on;
plot(t,polyval(polyfit(x,y,5),t));
```

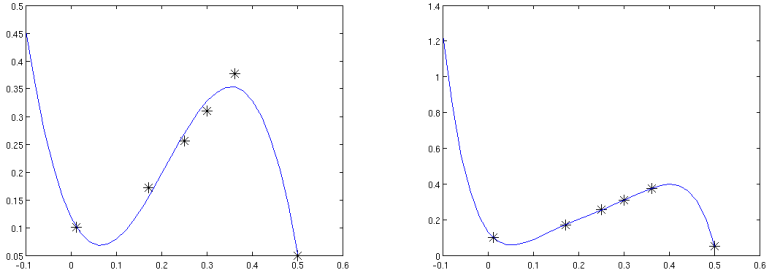


Рис. 19. Интерполяция данных из примера 30 полиномом третьей степени (слева) и полиномом пятой степени (справа)

f=interp1(x, y, a, method). Как, в предыдущих примерах, x и y – абсциссы и ординаты искомой зависимости, a – вектор абсцисс, в котором будут вычислены значения сплайна, возвращенные в векторе f. Method – один из методов интерполяции, может принимать следующие значения:

- 'nearest' – интерполяция кусочно-непрерывными полиномами нулевой степени;
- 'linear' – интерполяция кусочно-непрерывными полиномами первой степени, используется по умолчанию;
- 'spline' – интерполяция кусочно-непрерывными полиномами третьей степени;
- 'cubic' – интерполяция полиномами Эрмита третьей степени.

Использование функций spline или pchip возвращает вектор коэффициентов сплайнов. Для вычисления значений самих сплайнов может использоваться ppval. Пример использования такой интерполяции показан в примере 31.

5.5. Численное дифференцирование и интегрирование

В MATLAB нет функции, выполняющей численное дифференцирование. Исключение составляют функции вычисления производной от алгебраического полинома, заданного в символьной форме – diff и заданного в форме вектора – polyder. Функция polyder возвращает вектор коэффициентов

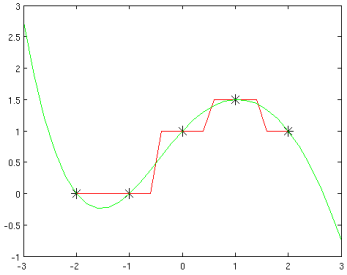


Рис. 20. Интерполяция данных из примера 31 сплайнами. Ступенчатая линия – интерполяция сплайном нулевой степени, плавная – третьей

Пример 31. Интерполяция сплайнами

```
x=[-2 -1 0 1 2];
y=[0 0 1 1.5 1];
t=[-3:0.2:3];
plot(x,y,'k*','markersize',12);
hold on;
Near=interp1(x,y,t,'nearest');
spl=interp1(x,y,t,'spline');
plot(t,Near,'Color','r');
plot(t,spl,'color','g');
```

производной от многочлена. В первой строке примера 32 был фактически определен многочлен $2x^3 + 3x^2 + 5x + 7 = 0$. При аналитическом дифференцировании этого выражения степень многочлена понижается и число коэффициентов уменьшается на единицу (производная от последнего числа в левой части равна нулю).

Если в качестве аргумента функции diff передать вектор, то в результате будет вычислен вектор разностей. Причина, по которой численное дифференцирование в MATLAB реализовано только одной функцией, заключается в том, что вычисление производных основано на интерполяции дифференцируемой функции алгебраическими полиномами не очень высоких степеней. Простейший способ вычисления производной с использо-

Пример 32. Численное дифференцирование и интегрирование многочлена

```
>> a=[2 3 5 7]
a =
     2     3     5     7
>> b=diff(a)
b =
     1     2     2
>> polyder(a)
ans =
     6     6     5
>> poly=poly2sym(a)
poly =
2*x^3 + 3*x^2 + 5*x + 7
>> int(poly)
ans =
x^4/2 + x^3 + (5*x^2)/2 + 7*x
```

Пример 33. Простейшее численное дифференцирование

```
%исходный код сценария
x=0:0.1:5;
y=x.^2;
xd=x(1:length(x)-1);
dxdy=diff(y)./diff(x);
plot(x,y)
hold on %для вывода обоих графиков в в одном графическом окне
grid on %включаем сетку на графике
plot(xd,dxdy)
delta=2*xd-dxdy
```

ванием функции diff заключается в простом вычислении отношения приращения функции к приращению аргумента. Понятно, что в определении производной приращение аргумента обязано быть бесконечно малым, и, используя такое приближение для производной, точных результатов ожидать не стоит. Для получения более точных результатов следует использовать специальные формулы (например, Ньютона или Стирлинга) для численного дифференцирования. В примере 33 построен график квадратичной функции и её производной. Как легко убедиться, выполнив этот сценарий, производная будет линейной функцией. Аналитическое нахождение производной приведет к выражению 2*xd, и разность этого значения с приближенным значением производной dxdy будет равна 0.1

Численное интегрирование – вычисление площади под графиком функции (в одномерном случае), также реализуется при помощи интерполяции функции различными способами. Для эффективного вычисления интеграла необходимо представлять особенности подынтегральной функции. Например, в случае разрывов (устранимых) функции в некоторой точке следует разбить интеграл на два так, чтобы точка разрыва попала на границу. Для обеспечения нужной точности и скорости сходимости следует выбирать алгоритм интегрирования в соответствии с характером роста интегрируемой функции.

Таблица 7. Функции MATLAB для вычисления определенных интегралов

Название	Значение
trapz(y,x)	Вычисляет определенный интеграл методом трапеций. Аргументы: y – подынтегральная функция, x – вектор аргумент. Обеспечивает достаточную точность для линейных функций

quad(y(x), xmin, xmax, eps, trace)	Вычисляет определенный интеграл по формуле Симпсона, обеспечивает достаточную точность для функций, четвертая производная от которых равна нулю. Обязательными аргументами являются имя функции и границы интегрирования. В качестве дополнительного аргумента можно указать точность вычислений – eps, по умолчанию равна 10^{-6}. Специальная опция trace позволяет включить вывод промежуточных значений на каждой итерации, для этого нужно задать любое ненулевое значение
quadl(y(x), xmin, xmax, eps, tarce)	Вычисляет определенный интеграл методом Лобатто. Смысл аргументов, такой же как у quad
quadgk(y(x), xmin, xmax, eps, tarce)	Вычисляет определенный интеграл методом Гаусса-Кронрода. Смысл аргументов такой же как у quad
dblquad(z(x,y), xmin, xmax, ymin, ymax, eps, method)	Вычисляет двойной определенный интеграл над прямоугольной областью. Необходимо указать границы изменения каждой переменной. Дополнительный параметр method может принимать два значения – quad и quadl
quad2d(z(x,y), xmin, xmax, ymin, ymax)	Вычисляет двойной определенный интеграл над плоской областью. Необходимо указать границы изменения каждой переменной
triplequad(z(x,y), xmin, xmax, ymin, ymax, eps, method)	Вычисляет тройной определенный интеграл. Необходимо указать границы изменения каждой переменной. Дополнительный параметр method может принимать два значения – quad и quadl

Вызов функций quad и quadl с двумя выходными парметрами приводит к выводу в командное окно числа обращений к вычислению подынтегральной функции. Вызов quadgk с двумя параметрами приводит к выводу приближенного значения погрешности интегрирования Как видно из примера, функция quadgk обеспечивает вычисление интеграла от заданной функции с погрешностью, не превышающей $21.1636264069171 \cdot 10^{-018}$,

Пример 34. Итегрирование в MATLAB

```
>> f=@(x)exp(-x).*cos(x)
f =
    @(x)exp(-x).*cos(x)
>> format longEng
>> s=quad(f,0,1)
s =
    555.396877498501e-003
>> s=quad(f,0,1,1e-8)
s =
    555.396882649120e-003
>> s=quad(f,0,1,1e-12)
s =
    555.396882653349e-003
>> [s,iter]=quad(f,0,1,1e-15)
s =
    555.396882653350e-003
iter =
    781.000000000000e+000
>> [s2,iter2]=quadl(f,0,1,1e-15)
s2 =
    555.396882653350e-003
iter2 =
    48.0000000000000e+000
>> [s3,err]=quadgk(f,0,1)
s3 =
    555.396882653350e-003
err =
    21.1636264069171e-018
>> [s2,iter2]=quadl(f,0,1,1e-15,2)
      18      0.0000000000      5.00000000e-01      0.5553968827
      23      0.0000000000      4.58758548e-02      0.0875482138
      28      0.0917517095      9.23207464e-02      0.1510834176
      33      0.2763932023      1.11803399e-01      0.1406211476
      38      0.5000000000      1.11803399e-01      0.0995695619
      43      0.7236067977      9.23207464e-02      0.0561174607
      48      0.9082482905      4.58758548e-02      0.0204570810
s2 =
    555.396882653350e-003
iter2 =
    48.0000000000000e+000
>> f=@(x)exp(-x)
f =
    @(x)exp(-x)
>> [s,err]=quadgk(f,2,Inf)
s =
    135.335283236613e-003
err =
    2.22626006030936e-012
>>
```

а функции quad и quadl обеспечивают точность 10^{-015} за 781 и 48 итераций соответственно. В фирменной документации утверждается, что quadl обеспечивает более высокую точность, чем quad; а quadgk обеспечивает высокую точность для периодических функций и, кроме этого, поддерживает бесконечные пределы интегрирования.

5.6. Решение обыкновенных дифференциальных уравнений

Обыкновенные дифференциальные уравнения возникают во многих задачах естественных наук. Чаще всего встречается задача Коши: отыскать некоторую функцию $y(x)$, удовлетворяющую дифференциальному уравнению

$$\frac{d^n y}{dx^n} = f(x, y)$$

и некоторым начальным условиям:

$$y(x_0) = y_0, y^{(1)}(x_0) = y_0^{(1)} \cdots y^{(n-1)}(x_0) = y_0^{(n-1)}$$

. Например, в общей физике наибольший интерес представляют дифференциальные уравнения второго порядка. Схема решения задачи Коши в MATLAB следующая:

1. Приведение уравнения n-го порядка к n уравнениям первого порядка. Это необходимо для задания вектора правых частей дифференциального уравнения. Размерность вектора равна порядку старшей производной;
2. Написание функции, задающей вектор правых частей системы дифференциальных уравнений первого порядка. Следует обратить внимание, что простое использование вектора вызовет ошибку во всех решателях;
3. Вызов решателя с нужными аргументами – вектором начальных условий и функцией правых частей системы уравнений.

Дальнейшая работа с полученными данными возможна обычными средствами MATLAB. Краткий синтаксис вызова решателя следующий: [x,y]=ode(function,x0,y0). Слева от знака равенства указаны выходные данные функции, которые будут вычислены обращением к решателю ode, y0 – вектор начальных условий, x0 – вектор значений независимой переменной. Расширенный синтаксис вызова решателей позволяет установить дополнительные опции, например абсолютную (по умолчанию 10^{-6}) и относительную (по умолчанию 10^{-3})погрешности, шаг интегрирования и др. Удобный способ задания всех таких опций предоставляет структура odeset. Вызов odeset без аргументов приведет к выводу всех полей этой структуры и их значений по умолчанию. Предварительное определение такой структуры командой вида opt=odeset('name',value) позволяет в дальнейшем при вызове решателя в качестве одного из аргументов указать opt.

Рассмотрим в качестве примера решение уравнения движения для тела, брошенного под углом к горизонту с некоторой скоростью. Если пренебрегать сопротивлением воздуха, то решение этой задачи прекрасно знает любой школьник – траекторией движения тела будет парабола. Для решения нам потребуется два дифференциальных уравнения второго порядка – по одному для каждой координаты:

$$\frac{d^2x}{dt^2} = 0;$$

$$\frac{d^2y}{dt^2} = -9.8.$$

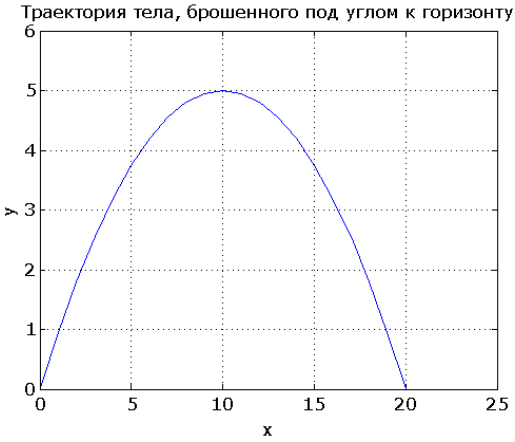


Рис. 21. Траектория тела, брошенного под углом к горизонту. Угол в данном примере равен 45°

В качестве начальных значений выберем начальные координаты? равными нулю, а первые производные этих координат – равными 10. Для формирования вектора правых частей дифференциального уравнения нам потребуется определить две новых функции (в обычном понимании) – по порядку уравнения. Введем обозначения: $y_1 = y$ и $y_2 = y_1'$. В новых функциях вторая производная от координаты будет представлена как первая производная от вспомогательной функции $y1$ и уравнение второго порядка для координаты y будет сведено к системе двух уравнений первого порядка:

$$\frac{dy_1}{dt} = y_2;$$

$$\frac{dy_2}{dt} = -9.8.$$

По своему физическому смыслу y_2 – это скорость тела вдоль вертикальной оси. Вектор начальных условий для системы дифференциальных уравнений тогда будет иметь вид `y0=[0;10]`. Для корректного решения вектор должен быть столбцом, поэтому элементы вектора отделены не пробелом, а точкой с запятой. Первое значение – начальная координата, второе – начальная (вертикальная) скорость. Функцию первый аргумент решателя сформируем с помощью оператора `@` – `yt=@(t,y)[y(2);-10]`. В круглых скобках указаны выходные параметры функции, а вектор-столбец есть вектор коэффициентов правых частей системы дифференциальных уравнений. В качестве решателя используем `ode45`, реализующий метод Рунге-Кутты 4

порядка – [to,yo]=ode45(yt,ts,y0). Выходными значениями [to,yo] являются вектор значений времени to и массив, содержащий координаты и скорости вдоль вертикали. Аналогичным образом преобразуется уравнение для горизонтальной координаты. Поскольку выходные массивы xo и yo содержат два вектор-столбца, то из них предварительно выделены столбцы, соответствующие координатам. В конце построен график зависимости одной координаты от другой – траектория тела.

Пример 35. Простейший сценарий решения задачи Коши для баллистического движения

```
%исходный код сценария
y0=[0;10]
x0=[0;10]
ts=0:0.1:2;
yt=@(t,y)[y(2);-10]
xt=@(t,x)[x(2);0]
[to,yo]=ode45(yt,ts,y0)
[to,xo]=ode45(xt,ts,x0)
x=xo(:,1)
y=yo(:,1)
plot(x,y)
grid on
set(gca,'FontName','Verdana','FontSize',14)
title('Траектория тела, брошенного под углом к горизонту')
xlabel('x')
ylabel('y')
```

В качестве дополнительного примера можно рассмотреть решение уравнения затухающих колебаний: $y''+2\beta y'+\omega^2 y=0$ (пусть $2\beta=5$, а $\omega^2=10$). Снова определяя вспомогательные функции в виде $y_1=y$ и $y_2=y_1'$, получим вектор для правых частей двух дифференциальных уравнений в виде [y(2);-2*y(2)-10*y(1)]. Пример 36 строит зависимость координаты затухающих колебаний от времени.

В MATLAB присутствует несколько решателей (solver) обыкновенных дифференциальных уравнений. Формат вызова практически всех решателей одинаков, аргументы также имеют одинаковый смысл. Для начала численных экспериментов с дифференциальными уравнениями рекомендуют использовать решатель ode45 и, только если этой функцией построить решение не удается, следует выбрать более подходящий для задачи решатель. Также если ode45 обеспечивает слишком высокую точность, а скорость работы, наоборот, низкую, то есть смысл выбрать другой решатель – ode23. Выбор решателя, обеспечивающего приемлемую точность за приемлемое время вычислений, определяется характером поведения искомой функции на интервале интегрирования. Ни один из решателей не является встроенной функцией, и это значит, что можно посмотреть и использовать его код для модификации стандартных решателей под конкретные задачи.

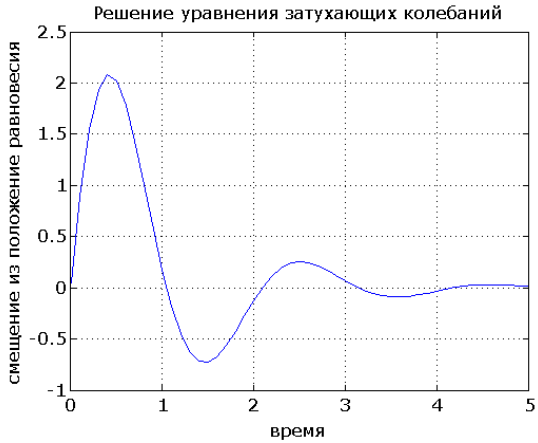


Рис. 22. График решения уравнения затухающих колебаний из примера 36

Пример 36. Простейший сценарий решения задачи Коши для уравнения затухающих колебаний

```
%исходный код сценария
y0=[0;10];
ts=0:0.1:5;
yt=@(t,y)[y(2);-2*y(2)-10*y(1)];
[to,yo]=ode45(yt,ts,y0);
plot(ts,yo(:,1))
grid on
set(gca,'FontName', 'Verdana', 'FontSize',14)
title('Решение уравнения затухающих колебаний')
xlabel('время')
ylabel('смещение из положение равновесия')
```

5.7. Введение в символьные вычисления

Возможности MATLAB по вычислению производных и интегралов велики, и приближенные вычисления это не всё, что можно реализовать. В MATLAB поддерживаются символьные вычисления, и многие математические операции могут быть произведены аналитически. Все такие возможности базируются на функции sym, возвращающей символьную переменную. Большая часть функций для аналитических вычислений собрана в специальном расширении MATLAB – Symbolic Math Toolbox, последние несколько версий которой имеют собственный графический интерфейс – muPad.

Для реализации символьных вычислений в MATLAB имеется отельный тип данных – sym. Для создания простейшего символьного объекта достаточно ввести в консоли sym('x'). Допускается объявлять символьные объекты списком – syms a b c автоматически создаст три переменные символьного типа. Для понимания различия между обычными вычислениями

Пример 37. Особенности арифметических действий над вещественными числами и символьными объектами

```
>> a=4/3
a =
    1.33333333333333e+000
>> b=a-1
b =
    333.333333333333e-003
>> c=b*3
c =
    1.00000000000000e+000
>> e=1-c
e =
    222.044604925031e-018
>> num2hex(e)
ans =
3cb0000000000000
>> num2hex(eps)
ans =
3cb0000000000000
>> a=sym(4/3)
a =
4/3
>> b=a-sym(1)
b =
1/3
>> c=3*b
c =
1
>> e=1-c
e =
0
```

и символьными следует начать с привычных вычислений. Вещественные числа типа double хранятся в памяти компьютера в приближенном виде, из 64 разрядов младшие 52 отводятся под мантиссу. При каждой арифметической операции совершается погрешность, равная младшему разряду мантиссы. Это значение называется машинной точностью, для понимания машинной арифметики удобно использовать шестнадцатиричную систему счисления. Такое представление числа можно в MATLAB получить с помощью функции num2hex. Пример 37 иллюстрирует погрешность машинных вычислений. Обратите внимание, что при вычислении над символьными объектами результат оказался точно равный нулю. Можно посмотреть, как накапливаются погрешности вычислений на примере матриц. Можно рассчитать произведение матрицы Гильберта на обратную к ней матрицу. Как известно, такое произведение обязано быть равным единичной матрице, однако если ввести команду a=hilb(n)*invhilb(n);, то можно увидеть, что в полученной матрице недиагональные элементы не все равны нулю (можно использовать большие числа n – обе функции встроенные и работают очень быстро). Стоит ввести команду sym(hilb(n))*sym(invhilb(n)), и будет видно, что в результате получилась действительно единичная матрица (не вводите n слишком большим – вывод результата на консоль будет долгим, как и сами вычисления).

Функция digits(n) устанавливает глобально точность вычислений для функции vpa, и это позволяет считать не только с точностью до 16 знака после запятой, но и до значительно большего. По умолчанию digits равно

32. Главная особенность в том, что повышенная точность вычислений требует большего времени счета. Функция vpa может и локально внутри себя изменить точность вычислений – vpa(expr,n) будет вычислять с точностью до n знаков после запятой. Вообще, существует три режима вычислений:

- с плавающей запятой машинной точности – представление вещественных чисел соответствует стандарту, ошибка одной арифметической операции не превышает константы eps;
- с плавающей запятой произвольной точности – числа представляются с произвольной точностью;
- точные вычисления – числа представляются как рациональные.

Каждый из вариантов предполагает свой формат отображения и хранения чисел и, соответственно, существуют функции преобразования из одного формата в другой. Для преобразования из машинной точности в рациональные числа просто используется функция sym. Однако у функции есть множество опций.

Таблица 8. Опции функции sym

Название	Значение
'f'	Преобразует вещественное число двойной точности с плавающей запятой в символьное число с плавающей запятой
'e'	Преобразует вещественное число в рациональное и возвращает разность между рациональным представлением числа и представлением с машинной точностью
'r'	Преобразует вещественное число в рациональное
'd'	Преобразует вещественное число в десятичную форму с количеством знаков после запятой, установленных функцией digits
'complex'	При объявлении символьного объекта дополнительно определяет его как комплексное число
'real'	При объявлении символьного объекта дополнительно определяет его как вещественное число
'positive'	При объявлении символьного объекта дополнительно определяет его как положительное число
'integer'	При объявлении символьного объекта дополнительно определяет его как целое число

Дополнительные ограничения на символьные переменные могут быть очень полезны во многих задачах. С объявлением этих опций есть одна особенность. Для очистки памяти MATLAB используется команда clear, при

Пример 38. Варианты объявления символьных переменных

```
>> sym(3.6)
ans =
18/5
>> sym(3.6,'f')
ans =
8106479329266893/2251799813685248
>> sym(3.6,'d')
ans =
3.60000000000000000888178419700125
>> digits(30)
>> sym(3.6,'d')
ans =
3.60000000000000000888178419700125
>> sym(3.6,'e')
ans =
(2*eps)/5 + 18/5
```

этом даже если сами переменные будут удалены из памяти MATLAB, то информация о том, какие ограничения на них наложены, всё равно останется. Для очистки рабочего пространства MATLAB от ограничений следует также использовать syms, но после имени переменной добавить clear. Применение этого приема иллюстрирует пример 39

Пример 39. Особенности снятия ограничений с символьных переменных. Переменная x была определена как вещественная, и потому уравнение $x^2 + 1 = 0$ корней не имеет. После удаления переменной из памяти MATLAB запись об ограничении, на нее наложенном, осталась.

```
>> syms x real
>> clear x
>> syms x
>> solve(x^2+1)
Warning: Explicit solution could not be found.
> In solve at 81
ans =
[ empty sym ]
>> syms x clear
>> solve(x^2+1)
ans =
 -i
  i
```

С символьными данными можно производить привычные для математики операции: аналитическое дифференцирование и интегрирование, решать уравнения, преобразовывать алгебраические выражения. Обратите внимание на то, что символьные объекты в памяти занимают значительно больше места, чем числовые типы данных.

Целая группа команд выполняет преобразования символьных математических выражений. Например:

- collect представляет выражение в виде многочлена, приводя подобные относительно показателей степени неизвестной;
- expand раскрывает все скобки;
- factor представляет многочлен в виде произведения многочленов меньшей степени;
- horner возвращает представление Горнера для многочлена;

Пример 40. Простейшие символьные вычисления

```
>> y=sym('x^n')
y =
x^n
>> x=sym('x')
x =
x
>> dydx=diff(y,x)
dydx =
n*x^(n - 1)
>> intyx=int(y,x)
intyx =
piecewise([n = -1, log(x)], [n <> -1, x^(n + 1)/(n + 1)])
>> limit(sin(x)/x,0)
ans =
1
>> whos
  Name        Size              Bytes  Class     Attributes

  dydx        1x1                 112  sym
  intyx       1x1                 112  sym
  x           1x1                 112  sym
  y           1x1                 112  sym
>> syms a b c
>> solve(a*x^2+b*x+c,x)
ans =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

Пример 41. Простейшие символьные преобразования

```
>> syms x y
>> expand(sin(x+y))
ans =
cos(x)*sin(y) + cos(y)*sin(x)
>> collect((x+3)*(x-2)*(x+5))
ans =
x^3 + 6*x^2 - x - 30
>> horner(ans)
ans =
x*(x*(x + 6) - 1) - 30
>> factor(x^6-1)
ans =
(x - 1)*(x + 1)*(x^2 + x + 1)*(x^2 - x + 1)
>> simplify(sin(x)^2+cos(x)^2)
ans =
1
>> simplify(2*sin(x)*cos(x))
ans =
sin(2*x)
```

- symplify упрощает математические выражения.

Использование преобразований приведено в примере 41.

Начиная с релиза в 2008 году, MATLAB включает дополнительный графический интерфейс в пакете Symbolic Math Toolbox – MuPad . Для его запуска достаточно в консоли набрать mupad. Изначально MuPad существовал как независимое приложение, а пакет символьных вычислений не имел никакого графического интерфейса, причём существовало два варианта. Один базовый обеспечивал около сотни команд по работе с символьными объектами, а второй поддерживал подмножество команд популярной системы компьютерной алгебры Maple (все команды, кроме графических). В релизе 2008a поддержка Maple была удалена, но появился интегрированный графический интерфейс с богатым набором команд. До сих пор существует приложение, обеспечивающее обмен данными между MATLAB и Maple, но распространяется компанией-производителем Maple

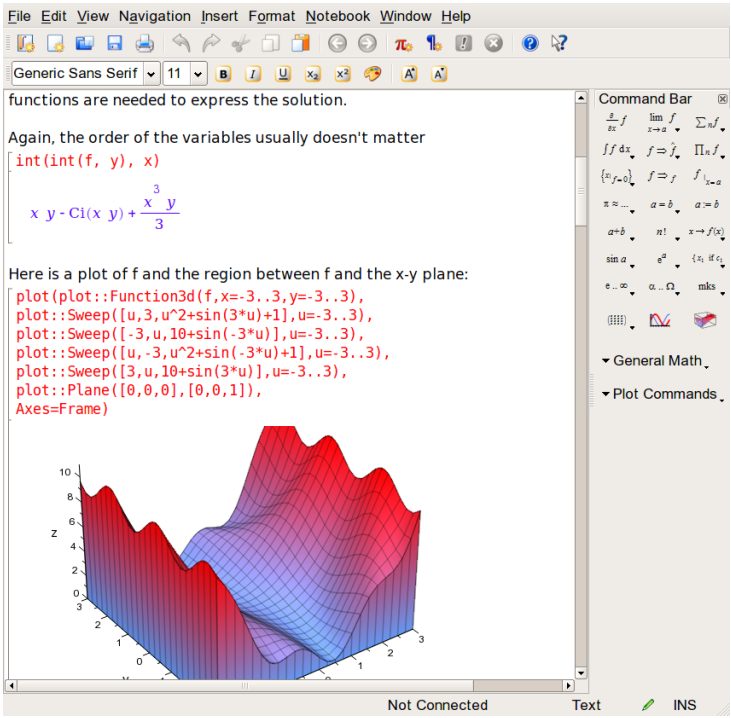


Рис. 23. Главное окно MuPad. Справа – панель инструментов для быстрого добавления команд в рабочую область. Обратите внимание на сочетание текста, программного кода и графики в одном документе

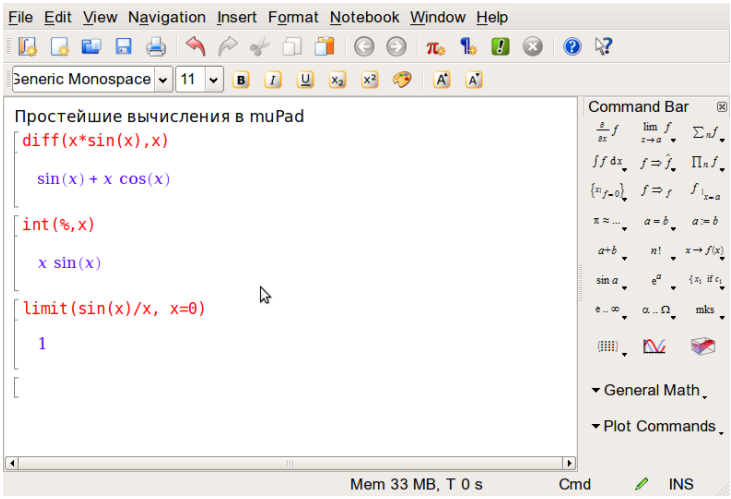


Рис. 24. Примеры простейших вычислений. Переменные не обязательно объявлять даже как символьные

Графический интерфейс для символьных вычислений не просто прихоть. Одна из ключевых особенностей систем компьютерной алгебры – близость оформленных в ней программ к естественной записи подобных вычислений на бумаге и объединение в рамках одного документа программного кода и текста. Вспомните классические программы MATLAB – программный код это обычный текст. Какие-либо возможности оформления этого текста отсутствуют. Попробуйте сделать текст выровненным по ширине или добавить в исходный код список, картинку или таблицу. Далее – результат выполнения программного кода выводится в новые окна отдельно от кода. MuPad позволяет создать документ, в котором живой программный код будет сочетаться с оформлением, достойным любой журнальной статьи. Это открывает широкие возможности для использования MATLAB в тех отраслях, где лидирующие позиции занимают Maple и Mathematica. Возникает новая возможность создания в MATLAB научно-технических публикаций и образовательных ресурсов. Сама по себе среда MuPad достаточно развита, для того чтобы решать практически все задачи непосредственно внутри, не переходя в MATLAB. Всё, о чем было рассказано в этом курсе до этого момента, может быть реализовано средствами только MuPad. Возможно, скорость вычислений будет не так высока, как в MATLAB, но для освоения и для решения учебных задач предельное быстродействие и не нужно.

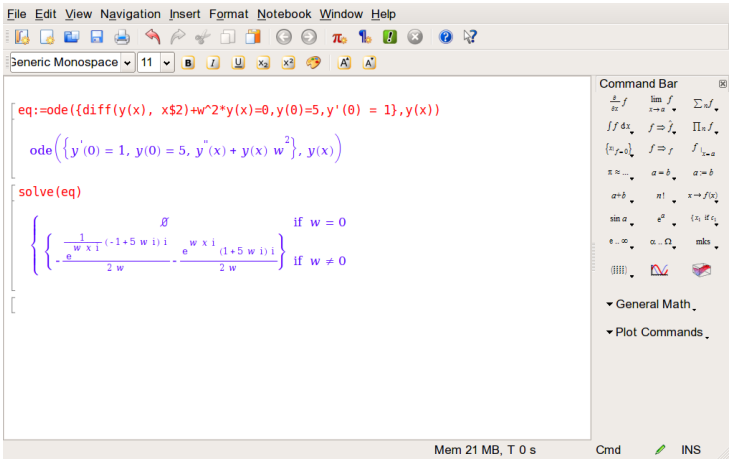


Рис. 25. Пример решения обыкновенного дифференциального уравнения – уравнения свободных колебаний. Обратите внимание, что числовое значение w - частоты не указано и MuPad самостоятельно построил два случая решения

В качестве дополнительных возможностей muPad можно указать следующие:

- удобная справочная система, включающая живые примеры;
- большой набор стандартных панелей элементов, облегчающих почти все вычисления, – например, для интегрирования функции не нужно помнить имя соответствующей команды, достаточно найти кнопку с изображением интеграла и соответствующая команда сама появится в ячейке ввода;
- замечательные графические возможности, в том числе анимации и трехмерные графики;
- экспорт документов в HTML и PDF формат с сохранением всех введенных команд и всех результатов работы.

Обмен данными между MATLAB и MuPad позволит организовывать эффективную работу и обеспечит одинаково хорошие и скорость вычислений и дружественное представление программ.

После запуска MuPad создаст пустой документ, на главной панели инструментов окна muPad можно выбрать один из двух вариантов нового документа:

- notebook – режим документа, в котором красивое визуальное оформление сочетается с программным кодом;
- editor – исходный код – режим создания и редактирования исходного кода, без возможности его красивого оформления.

Ввод команд в окне следует производить в ячейку ввода – справа от символа открывающей квадратной скобки. После ввода команды выполнение её запускается клавишей Enter, для перехода на новую строку следует нажать Shift + Enter. Запуск всего документа на выполнение производится с помощью меню NoteBook-Evaluate All главного меню.

5.8. Задания для самостоятельной работы

1. Напишите функцию, создающую квадратную матрицу, каждый элемент которой удовлетворяет соотношению $a(i,j) = \sqrt{i+j-1}$. Можно ли ту же матрицу создать встроенной в MATLAB функцией?
2. Напишите функцию, которая будет создавать квадратную матрицу, заполненную элементами, которые пользователь ввел с клавиатуры.
3. Для произвольных векторов a и b напишите функцию, создающую комплексный вектор c, для которого действительная часть будет равна соответствующему элементу первого вектора, а мнимая – элементу второго.
4. Напишите функцию, которая смогла бы вычислить определенный интеграл функции, зависящей от параметра. Простейший пример – ax^2, значение параметра a необходимо указывать при вызове пользовательской функции.
5. Модернизируйте программу о движении тела, брошенного под углом к горизонту так, чтобы получилась MATLAB-функция с углом наклона скорости к горизонту в качестве входного аргумента. Сделайте так, чтобы крайняя точка на построенном графике соответствовала дальности полета, а над верхней точкой траектории была бы надпись, содержащая значение высоты подъема и дальности полета.
6. Напишите программу, решающую дифференциальное уравнение вынужденных затухающих колебаний. При построении графика организуйте вывод четырех осей координат, которые бы содержали зависимость от времени координаты и скорости в двух интервалах времени – от нуля до времени релаксации и от времени релаксации дальше.
7. Напишите MATLAB-функцию, которая по заданному аргументу – указателю на функцию, считала бы определённый интеграл от неё на указанном интервале. Границы интервала также должны быть в качестве входных параметров функции. Добавьте возможность указания ошибки вычисления интеграла.

8. Напишите MATLAB-функцию, которая бы для любой функции одного аргумента считала бы определенный интеграл от нее с использованием трех встроенных функция MATLAB. В качестве выходных данных программы должна выступать структура, содержащая, как минимум, следующие поля: значение интеграла, имя решателя, значение погрешности.

6. Создание графического интерфейса средствами MATLAB

6.1. Иерархия классов MATLAB

Для понимания идеологии создания графического интерфейса следует вспомнить работу с обычными графическими объектами MATLAB. Принципиальное значение имеют встроенные функции для доступа к свойствам таких объектов – set и get. Первая функция устанавливает значения свойств, вторая возвращает установленные значения и, что более важно для данной главы, формирует указатели на графические объекты. Важно отметить различие в результатах использования функции set. Например, в программе можно написать – CurFig=figure, и будет создано пустое графическое окно. Инструкция set(CurFig,'Position',[0 0 800 600]) приведет к созданию окна с размером 800 на 600 пикселей. А команда set(figure,'Position',[0 0 800 600]) изменит настройки по умолчанию, и все объекты figure далее будут создаваться с такими размерами.

Уникальность MATLAB в части проектирования графического интерфейса заключается в том, что традиционные для любых сред разработки приложений элементы – кнопки, меню, полосы прокрутки и др. – практически не отличаются по своей внутренней структуре от специфических объектов – осей координат, подписей к графикам. Например, в традиционных средствах разработки создание пользовательского интерфейса реализуется легко, а создание средств научной графики требует подключения дополнительных библиотек или низкоуровневого программирования. В MATLAB доступ ко всем графическим объектам реализован одинаково независимо от их специфики. Базируется такая общность на иерархии классов MATLAB.

MATLAB является объекто-ориентированным языком программирования. В основе этой технологии лежат три понятия – класс, наследование, полиморфизм. Рассмотрим эти понятия на примерах. Когда в MATLAB строится график функции, то создаётся новое графическое окно. Пустое окно можно создать встроенной командой figure. В любом случае создаётся (по терминологии объектно-ориентрированного программирования) объект или экземпляр класса. В иерархии MATLAB содержится определение класса, со всеми свойствами, методами событиями и прочим. Когда был создан экземпляр этого класса, то сам базовый класс при этом не был изменен, и далее на основе базового класса можно создавать новые объекты. Таким образом, в MATLAB в одном графическом окне размещаются сразу несколько экземпляров класса Axes. В иерархии классов у всех за исключением самого верхнего есть родительские классы (parent) и дочер-

ние, или производные (children). Идея наследования состоит в том, что дочерние классы обладают теми же свойствами, что и родительские. Таким образом, сокращаются расходы на программирование новых классов, в том числе и пользовательских. В MATLAB объект figure обладает свойством Position, которое хранит координаты левого нижнего угла, ширину и высоту графического окна и у всех объектов ниже по иерархии имеется свойство Position. Это значительно облегчает повторное использование и сопровождение кода, при этом в новых классах, создаваемых на основе базовых, можно определять и новые свойства. Наконец, идея полиморфизма заключается в так называемой перегрузке функций – функции, унаследованные от базового класса, можно переопределить, и в результате функция одна и та же будет возвращать результат в зависимости от того, к какому аргументу она применена.

Иерархию классов в MATLAB продемонстрировать очень просто. В консоли введите команду figure – откроется пустое окно. Далее введите get(gcf), будет выведен подробный список свойств графического объекта. Найдите в этом списке два – Parent и Children. Убедитесь, что значение первого свойства 0, а второго – пустой массив – []. После этого в консоли введите axes. Очень важно, что создание осей координат не вызовет создание нового объекта figure, а приведет к появлению в уже имеющемся окне (окно, созданное после предыдущей команды, закрывать не нужно) осей координат. После этого введите – get(gca) и в списке снова найдите те же свойства. Убедитесь, что для осей они равны соответственно 1 и []. После этого введите get(gcf), и вы увидите, что свойство Children у объекта figure перестало быть пустым массивом – у объекта figure появился дочерний экземпляр класса axes. Далее обратите внимание, что у axes имеются свойства, отсутствующие у figure – FontSize, FontName и другие, связанные со шрифтами.

Верхним в иерархии графических объектов является Root, – дословно – корневой объект. Указателем на этот объект является число ноль, и поэтому в качестве родителя объекта figure было именно нулевое значение. Важно понять, что Root это собственно окно самой системы MATLAB (конечно, если MATLAB работает в режиме графического интерфейса). Используя свойства этого объекта, можно, программно управлять настройками MATLAB. Например, обеспечить вывод чисел в консоль в формате longEng – set(0,'format','longEng'). Следующими в иерархии являются figure и axes. Производными от axes классами являются:

- image – картинка, возможно отображение файла, в любом допустимом графическом формате или визуализация массивов, созданных в MATLAB;

- line – линия, с помощью этого низкоуровневого объекта создаются графики двумерных функций;
- patch – многоугольник на плоскости или многогранник в пространстве;
- surface – поверхность, с помощью этого объекта строятся графики двумерных функций;
- rectangle – прямоугольник;
- text – текстовые объекты;
- light – освещение графических объектов;
- uicontextmenu – пользовательское контекстное меню;
- hggroup – группы любых объектов, исключая light. С группой объектов можно обращаться, как с единым целым при выделении и установке свойства видимости;
- hgtransform – группа любых объектов, исключая light. Дополнительно к выделению и общему заданию свойства visible доступны сразу для всех объектов перемещения, повороты и масштабирования.

Ввод в консоли MATLAB имени любого из этих объектов обеспечит его создание со всеми параметрами, установленными по умолчанию. Чтобы создавать пользовательские объекты с измененными свойствами, следует использовать функцию set. Для начала нужно создать указатель на новый графический объект, например командой – MyHandle=get(figure). После этого можно по созданному указателю установить любые свойства пользовательского графического окна. Сделать это можно и после того, как окно уже прорисовалось и до этого. Установка значений всегда имеет следующий синтаксис – set('name','value'), name – имя свойства, value – название (если название – переменная типа char, то она должна быть в апострофах). Для создания, например, осей координат не обязательно заранее создавать экземпляр класса figure, его создание происходит автоматически, как объекта, расположенного выше по иерархии. То же относится к объектам, дочерним по отношению к axes.

Для эффективного программирования таких объектов следует кратко рассмотреть некоторые из свойств, получаемых по команде get(name), где name – имя объекта. Часть из этих свойств имеет очевидный смысл и значение, например, Visible или LineWidth, и комментировать эти свойства нет смысла. Другие свойства довольно специфичны или сами по себе или их значения. Первое, что сразу бросается в глаза и чем чаще всего приходится управлять при программном размещении элементов на экране, это свойство Position. Смысл его состоит в указании экранных координат объекта. Для большинства объектов это свойство задаётся в виде вектора из 4 вещественных чисел – [x y w h]. x и y – координаты горизонтальная и

вертикальная левого нижнего края объекта, а w и h – соответственно ширина и высота. Все числа указываются в системе единиц, установленной свойством Units. Существуют 6 вариантов систем единиц в MATLAB

1. normalized – нормализованный, или нормированный, вариант задания координат. При этом независимо от расположения на экране координаты измеряются от левого нижнего края [0 0] объекта родителя до верхнего правого края объекта родителя – точка с координатами [1 1]. Соответственно, все остальные координаты выражаются в виде чисел от 0 до 1. Этот вариант удобен для правильного относительного расположения графических объектов и используется по умолчанию для всех объектов начиная с axes и ниже по иерархии. Чтобы задать, например, axes, расположенным в нижнем правом углу главного графического окна, можно ввести в консоли команду – axes('Position',[0.5 0.2 0.4 0.4]). Обратите внимание, что 0.4 0.4 в векторе – это всё-таки ширина и высота, которая задаётся также в относительных единицах по отношению к объекту-родителю. Поэтому если последнюю пару чисел набрать – 1 1, то axes будет выходить за границы figure – ведь его ширина и высота равны свойствам объекта-родителя.
2. pixels – все размеры определяются в пикселях, причём началом отсчёта считается левый нижний угол монитора, а не вышележащего по иерархии объекта. Способ крайне удобен для определения положения и размера объектов Root и figure. Для Root и вовсе невозможно установить никакую относительную систему единиц. Именно для двух верхних уровней в иерархии классов этот вариант единиц измерения установлен по умолчанию. Для просмотра действия команд достаточно в консоли набрать figure('Position',[0 0 1920 1080]), где 1920 1080 – разрешение монитора (используйте пару чисел, соответствующих вашему монитору). Используя свойство ScreenSize объекта Root, можно программно определить разрешение монитора, используемого в данный момент, и заставить MATLAB рисовать графическое окно figure всегда равным четверти от всего размера монитора.
3. inches – все размеры указываются в дюймах, началом отсчёта считается левый нижний угол. Для примера можно в консоли набрать – figure('Units','inches','Position', [0 0 2 2]). В нижней части экрана при этом появится графическое окно, размером 2 на 2 дюйма. Этот вариант удобен для подготовки графических объектов к печати, где необходимо соблюдать абсолютные размеры. Важно посмотреть эффект команды axes('Units','inches','Position', [0 0 2 2]) на изменение размеров графического окна – размеры осей координат не изменяются, и это основное неудобство этого варианта координат.

4. cantimeters – все размеры указываются в сантиметрах, остальное так же, как и для предыдущего случая.
5. points – все размеры указываются в пунктах, по определению пункт это 1/72 часть дюйма.
6. characters – все размеры измеряются в символах. За ширину символа принимается ширина буквы "x" шрифта, установленного в MATLAB по умолчанию. За высоту – расстояние между базовыми линиями соседних строк текста.

Для доступа к каждому объекту используется или его имя, или указатель, или значение свойства Tag объекта. По умолчанию в случае создания объектов с помощью конструктора GUIDE MATLAB самостоятельно присваивает Tag значение, образуемое конкатенацией имени класса объекта и порядкового номера. При программном создании объектов свойство tag остается пустым. Отслеживание взаимодействия пользователя и объектов можно организовать с помощью свойств Selected и SelectedHighlight, которые указывают, выбран ли объект и отмечен ли этот выбор специальным подсвечиванием. Свойство HitTest управляет режимом выбора объекта. Если установить это значение равным off, то щелчок кнопкой мыши по этому объекту приводит к тому, что свойство CurrentObject объекта figure устанавливается равным пустому массиву. Визульано это означает, что объект нельзя выбрать. По умолчанию это свойство включено. Однако следует иметь в виду, что часто свойство Selected установлено по умолчанию равным off и выбирать объект нет возможности. Возможность изменять размеры объекта устанавливает свойство resize, имеющее два два значения – 'on' и 'off'.

У объекта figure есть ряд особых свойств, которые имеют смысл только для него и отключены, несмотря на наследование у остальных объектов-потомков. Свойство name устанавливает заголовок, отображаемый в той части окна, где находятся системные кнопки: закрыть, свернуть, развернуть. Принцип формирования этого заголовка такой же, как для свойства tag. У вновь создаваемого окна имя будет Figure1. Чтобы изменить эту строку, введите figure('Name','Первое приложение'), в результате в верхней части окна будет строка: Figure1: Первое Приложение. Чтобы убрать Figure1 из отображаемого имени, нужно свойство NumberTitle сделать равным пустому массиву (или любой строке, если вы ее хотите увидеть в заголовке). Свойство MenuBar управляет выводом стандартного главного меню – часто в пользовательских графических окнах эти меню не нужны. Для их отключения следует свойству присвоить значение 'none'. Для обеспечения взаимодействия с пользователем часто приходится выяснять положение курсора мыши. Координаты курсора хранятся в свойстве CurrentPoint в виде вектора с двумя числами, начало координат соответствует левому

нижнему углу. Свойство устанавливается равным нулевому вектору в момент создания объекта figure. и относится к типу read-only – только для чтения. Оно не может быть установлено, а только прочитано.

Для создания пользовательского интерфейса используется также объект uicontrol. Самое важное свойство этого объекта – style. При различных значениях этого свойства объект uicontrol на экране выглядит по-разному. Каждый вариант свойства style в конструкторе графического интерфейса GUIDE реализован как отдельный элемент управления.

Таблица 9. Возможные варианты свойства style

Название	Значение
text	Графический объект со статическим текстом. Чаще всего используется для вывода текстовой информации, в том числе изменямой в процессе выполнения программы
edit	Графический объект с редактируемым текстом. Чаще всего используется для взаимодействия с пользователем, когда нужно чтение введенных с клавиатуры символов
frame	Прямоугольная область для группировки и визуального выделения других графических объектов. В текущей версии MATLAB рекомендуется группировку объектов производить с помощью uipanel или uibuttongroup
lisbox	Графический объект, содержащий список. Используется чаще всего для организации возможности выбора одного из нескольких заранее запрограммированных случаев
popupmenu	Раскрывающийся список – при отображении на экране подобен listbox, но в каждый момент отображает только текущий выбор
pushbutton	Стандартная командная кнопка
radiobutton	Переключатель. Удобен для организации выбора только одного варианта из всех
checkbox	Флаг. Позволяет организовывать множественный выбор из всех вариантов
slider	Стандартная полоса прокрутки. Удобна для выбора значения из заранее установленного диапазона, с заранее установленным шагом
togglebutton	Командная кнопка, визуальное представление которой меняется в зависимости от того, активирована она или нет

В любом случае объект uicontrol имеет такие свойства, как Position, Visible, Selected и многие другие, унаследованные от объектов-родителей.

Но есть и специфические свойства. Свойство Enable устанавливает доступность объекта в данный момент времени и имеет два возможных значения – 'on' и 'off'. Если объект имеет свойство Enabled, равное 'off', то его использование через графический интерфейс в режиме выполнения будет заблокировано. По умолчанию для всех объектов это свойство равно 'on' (конечно, если вообще присутствует). Свойство TooltipString позволяет определить текст, который будет появляться при наведении мышкой на данный объект. Обычно этот текст является пояснением к используемому элементу. Свойство String определяет текст, видимый на объекте. Например, команда uicontrol('Style','pushbutton','String','Кнопка') в консоли MATLAB приведет к созданию графического окна с командной кнопкой в нижней его части. На кнопке будет соответствующая надпись – кнопка. Есть свойства, специфичные для данного стиля объекта. Например, индекс выбранной строки в listbox или шаг ползунка для slider.

Главное свойство всех этих элементов – Callback, определяет, какой программный код будет выполняться при обработке определенных событий, связанных с этим объектом. Рассмотрим пример. В одной из прошлых глав был создан сценарий, строящий траекторию движения тела, брошенного под углом к горизонту. Предположим, соответствующий файл сохранен в одном из каталогов, присутствующих в путях поиска MATLAB с именем ballistika.m. Попробуем создать графическое окно, в котором присутствовала бы кнопка, запускающая наш сценарий и выводящая в текущее графическое окно траекторию. Может показаться, что нужно слишком долго программировать. На самом деле это не так. Как уже много раз упоминалось, не обязательно вручную создавать объект figure. Он будет создан автоматически, как только создается один из объектов, лежащих ниже по иерархии. Кнопка таким объектом и является. Значит, всё что нужно – это создать кнопку и её свойству Callback присвоить значение 'ballistika'. В консоли MATLAB нужно просто ввести uicontrol('Style','pushbutton','String','Кнопка','Callback','ballistika'). Будет создано графическое окно с кнопкой. Нажмите на неё и убедитесь в выполнении выбранного сценария. Конечно, это простейший пример, и мы скоро его немного усложним. Важно, что предметная простота примера вполне соответствует его программной реализации.

Еще один важный объект – uicontextmenu, он позволяет создать пользовательские контекстные меню, доступные при нажатии правой кнопки мыши. Соответствующие меню можно создать для каждого объекта внутри графического окна. Непосредственно элементы меню создаются функцией uimenu. В примере 42 продемонстрирован результат создания контекстного меню, позволяющего изменять цвет линии графика, полученного в результате работы сценария ballistika.m

Пример 42. Создание простейшего контекстного меню (для работы необходимо подготовить массивы x и y)

```
MyMenu=uicontextmenu;
MyLine=plot(x,y,'uicontextmenu',MyMenu);
MyColor1='set(MyLine,''color'',''r'')';
MyColor2='set(MyLine,''color'',''b'')';
MyColor3='set(MyLine,''color'',''g'')';

item1=uimenu(MyMenu,'Label','красный','callback',MyColor1);
item2=uimenu(MyMenu,'Label','голубой','callback',MyColor2);
item3=uimenu(MyMenu,'Label','зеленый','callback',MyColor3);
```

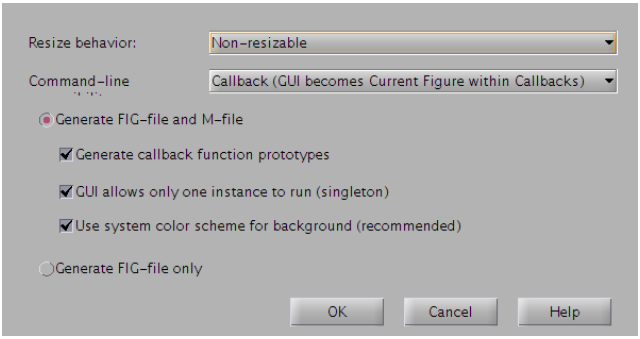


Рис. 26. Окно GUI Options

6.2. Среда создания графического интерфейса GUIDE

В принципе существует две возможности создания графического интерфейса средствами MATLAB. Первый вариант – использование специального инструмента визуального проектирования интерфейса. Чтобы его запустить, нужно или в консоли MATLAB ввести guide, или нажать кнопку START в левом нижнем углу и в меню MATLAB выбрать пункт GUIDE. Второй способ – ручное программирование пользовательского интерфейса "с нуля" предоставляет большие возможности по тонкой настройке интерфейса, но сложен для новичков. Вариант, который будет рассматриваться ниже, сочетает использование и визуального проектирования, и программирования в редакторе .m файлов. Соответственно представлен пользовательский графический интерфейс в режиме по умолчанию двумя файлами – один собственно файл интерфейса с расширением .fig, второй файл программы с расширением .m.

В окне GUIDE выберите TOOLS – GUI Options. В открывшемся окно можно изменять параметры работы приложений, указывать реакцию на изменение размера (resizable) или вовсе запретить пользователю менять размер окна в процессе работы. Вторая группа опций указывает на характер взаимодействия графического интерфейса и командной строки. Опция

Callback означает, что при выводе на экран результатов работы в графические окна ваше приложение будет использовать для вывода текущее графическое окно. Выбор off приведет к тому, что для вывода графической информации будет создаваться новая копия объекта figure. Вариант On разрешает изменять параметры запуска приложения из командной строки MATLAB. Вариант Other позволяет устанавливать параметры взаимодействия в специальном окне Property Inspector или в программном коде. И, наконец, последняя группа опций производит выбор между вариантами сохранения модели – по умолчанию графический интерфейс сохраняется в виде визуального представления и синхронизированной с ним MATLAB-программы. Режим по умолчанию можно изменять, например, можно запретить создание файла с программным кодом и будет сохраняться только визуальное представление в виде графического объекта MATLAB. Файл с расширением fig может быть открыт в MATLAB просто так – как готовый без возможности редактирования, а может быть открыт с помощью GUIDE, и тогда в модель можно будет внести дальнейшие изменения.

Итак, запустите конструктор графического интерфейса. Вам будет предложено на выбор несколько вариантов дальнейших действий. Можно создать новую модель в четырех вариантах:

1. Blank Gui – пустое графическое окно (используется по умолчанию);
2. Gui with Uicontrols – графическое окно с элементами управления;
3. Gui with Axes and Menus – графическое окно с осями координат и меню;
4. Modal Question Dialog – специальный тип графического окна – модальное окно.

Все варианты за исключением первого уже заполнены некоторыми объектами, исходный код всех этих окон может быть открыт и отредактирован. Для начала давайте создадим пустое графическое окно. Давайте решим задачу об изменении сценария ballistika для добавления туда следующих возможностей:

1. Возможности указывать скорость, с которой тело было брошено;
2. Возможности указывать угол наклона вектора скорости к горизонту;
3. Возможности задавать дополнительные опции оформления графика – линии сетки, метки на осях, толщину графика и др.

Перечень интересующих нас возможностей специально подобран так, чтобы задействовать в графическом интерфейсе все базовые элементы управления.

После создания пустой модели откроется главное окно конструктора. В левой его части видна панель инструментов управления. Легко увидеть, что там представлены разные стили объекта uicontrol. Разместим на

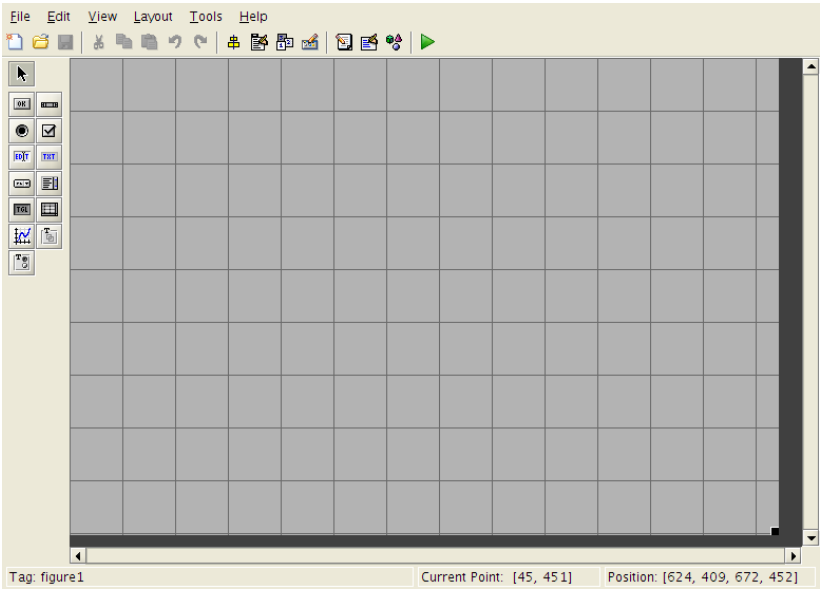


Рис. 27. Главное окно конструктора графического интерфейса GUIDE

экране кнопку pushbutton и два элемента edit. По умолчанию схема именования элементов управления такова, что в поле tag записывается имя стиля объекта uicontrol с порядковым номером. В создаваемом примере мы специально не стали изменять значения tag по умолчанию. Дело в том, что созданный интерфейс – это совокупность двух файлов – визуального и программного. Если пользователь проделает в конструкторе все манипуляции, описанные ниже, то получит внешний вид. Для получения поведения программы такого же, как описан ниже, в таком случае достаточно будет скопировать код из окончательного варианта в тот файл, который GUIDE создаст автоматически. При изменении свойства tag у элементов управления потребовалось бы слишком много лишних ненужных действий для работы с учебным примером. При этом, конечно, мы рекомендуем в реальных проектах указывать более осмысленные значения tag для объектов.

Все свойства элементов управления можно изменять с помощью окна Property Inspector – редактора свойств. Для перехода в это окно достаточно дважды кликнуть мышкой по любому объекту. Если теперь, не закрывая окно редактора свойств, выделить другой объект, то автоматически значения всех полей сменятся на свойства нового выбранного объекта.

Расположение, цвет и имена всех элементов можно задать по своему

Пример 43. Модификация сценария построения траектории тела, брошенного под углом к горизонту. Функция имеет два входных параметра – скорость и угол между скоростью и горизонтом. Два выходных параметра – координаты. Для удобства использованы функции cosd и sind, для которых в качестве аргумента углы задаются в градусах:

```
function [x,y]=ballistika(v0,alpha);
y0=[0;v0*sind(alpha)];
x0=[0;v0*cosd(alpha)];
ts=0:0.1:2;
yt=@(t,y)[y(2);-10];
xt=@(t,x)[x(2);0];
[to,yo]=ode45(yt,ts,y0);
[to,xo]=ode45(xt,ts,x0);
x=xo(:,1);
y=yo(:,1);
```

усмотрению. При расположении объектов на экране удобно пользоваться линиями сетки и линейками, их можно включить/отключить в специальном диалоге. Выделенный объект на этапе дизайна – визуального проектирования интерфейса можно перемещать не только мышкой, но и курсорами клавиатуры. Для упорядочивания пространственного расположения элементов можно использовать встроенное средство по их выравниванию. Перед созданием работающей модели полезно расположить несколько объектов на экране так, как вам нужно. Имейте в виду, что положение любого объекта может быть изменено программно с помощью установки свойства Position.

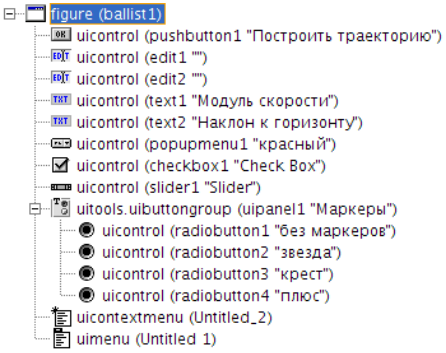


Рис. 28. Окно Object Explorer

Бывают ситуации, когда один объект перекрывает другой и выбор объектов с помощью мыши в графическом окне затруднен. В этом случае помогает специальное окно – object explorer.

По нажатию кнопки должно происходить выполнение функции ballistika. Входными параметрами функции заданы модуль скорости и угол наклона

вектора скорости к горизонту. Выходными параметрами функции являются координаты тела. Для добавления в исходный код программы, связанной с нажатием кнопки, следует вначале определить это событие и добавить тело соответствующей функции. Можно прописать всё это вручную, а можно правой кнопкой мыши щелкнуть на кнопке и в меню выбрать пункт View Callback. В меню должны появиться 5 пунктов:

1. Callback – базовое событие активирования соответствующего объекта;
2. CreateFcn – событие создания объекта;
3. DeleteFcn – событие удаления объекта;
4. PressDownFcn – нажатие кнопки мыши;
5. KeyPressFcn – нажатие клавиши клавиатуры.

В данном случае нам нужно событие Callback. После нажатия кнопкой мыши на соответствующем пункте открывается m-файл, содержащий программный код нашей модели. По умолчанию GUIDE самостоятельно генерирует достаточно комментариев к программе и некоторый дополнительный код, который может быть не обязательным. Первая строчка определяет переменное число входных и выходных параметров функции. Такой синтаксис удобен для обращения к функции в дальнейшем – это позволяет вызвать функцию без аргументов или передать дополнительные опции внешнего вида модели, например размер окна и прочие. Для экономии места из примера 44 удалена большая часть комментариев, оставлены только комментарии, отмечающие начало и конец кода инициализации приложения. Строки с 22 по 27 определяют команды, которые будут выполнены соответственно при открытии и закрытии приложения. Строки с 35 по 51 определяют события создания и активации двух текстовых полей. Если содержимое созданного вами файла отличается от приведенного в примере 44, то это означает, что было проделано больше манипуляций с элементами управления. Часть таких пользовательских действий сохраняется в программном коде. События по активированию текстовых полей пустые – не содержат программного кода, а значит, их можно удалить. Генерируются они по шаблону и в принципе могут потребоваться в дальнейшем.

Теперь нужно заняться самым ответственным – программированием обработчика события нажатия кнопки. В текстовых полях будут во время выполнения программы введены некоторые числа, которые должны стать фактическими аргументами функции ballistika. Свойство в котором хранятся эти значения, называется String и имеет тип char – это важно, потому что нельзя в функцию ballistika в качестве значения скорости и угла передать строковые переменные. Для преобразования к числовому формату используем функцию str2num, а для доступа к свойству объектов edit используем

handles. Важно понять, что структура handle содержит поля с указателями на все графические объекты, дочерние по отношению к figure, и, соответственно, именно эта структура поможет в дальнейшем организовывать доступ ко всем свойствам всех объектов. Функция get(handles.edit2,'string') возвращает строковую переменную и далее str2num преобразует ее к числовому формату. Добавьте в исходный код строки, показанные в примере 44, и запустите программу. Запуск можно делать как из окна редактора исходного кода, так и из окна конструктора. В текстовые поля в режиме выполнения программы введите числа, например 10 и 45, и нажмите кнопку. Убедитесь, что программа работает корректно и строит параболу.

Таков минимальный функционал нашего первого графического интерфейса. Первое, что нужно сделать – обеспечить наглядность модели. Для этого можно в графическом окне разместиться рядом с полями ввода два объекта text — статичный текст с подписями, поясняющими смысл текстовых полей. Далее, следует цветом выделить поля ввода от остальной части, можно задать для них свойство BackGroundColor равным белому цвету. На кнопке лучше разместить слова – Построить траекторию. Для этого нужно в конструкторе отрыть окно Property Inspector и выбрать для кнопки свойство String. Проще всего эти свойства определять именно в графическом интерфейса GUIDE. Можно для этого же прописать свойства в функции function ballMy_OpeningFcn(hObject, eventdata, handles, varargin) – все действия, указанные в теле этой функции, будут выполняться каждый раз при загрузке программы.

Дальнейшая модификация нашей модели будет заключаться в профилактике ошибочных ситуаций. Предположим, текстовые поля при запуске программы пустые (значения этих полей можно на этапе проектирования в GUIDE или обнулить, или заполнить числами). Тогда при запуске модели кнопка, будучи уже активной, вызовет ошибку – ведь ей передадутся в качестве аргументов пустые строки. Подобные ошибки не всегда приводят к появлению в консоли MATLAB сообщений, но уж точно заставляют модель работать не так, как задумывалось заранее. Давайте сделаем кнопку не активной до тех пор, пока в поле ввода не появятся числа и, более того, до тех пор пока после введения чисел в обоих полях пользователь не нажмёт, например, клавишу Enter . Первое решается просто – в функции создания формы разместите команду set(handles.pushbutton1, 'enable','off') – как видно из синтаксиса она делает кнопку неактивной. Запустите модель, и вы увидите, что кнопка так и остается всё время неактивной. Добавим обработку текстовых полей. Для этого нужно работать с функциями function edit1_Callback(hObject, eventdata, handles). В этих функциях мы должны выяснить, равно ли свойство String текстовых полей нулю, и, если в обоих

Пример 44. Исходный код графического окна. Добавлен обработчик нажатия кнопки, по которому выполняется функция ballistika с входными параметрами, считанными из свойств элементов управления edit. Код сгенерирован средой GUIDE, большая часть комментариев для экономии места удалена

```
function varargout = ballMy(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ballMy_OpeningFcn, ...
                   'gui_OutputFcn',  @ballMy_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

function ballMy_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);

function varargout = ballMy_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function pushbutton1_Callback(hObject, eventdata, handles)
v0=str2num(get(handles.edit1,'String'));
alpha=str2num(get(handles.edit2,'string'));
[x,y]=ballistika2(v0,alpha);
plot(x,y);
grid on;
function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Пример 45. Проверка ввода значения в текстовые поля для активации кнопки

```
function edit1_Callback(hObject, eventdata, handles)
if length(get(handles.edit1,'string'))>0 & ...
length(get(handles.edit2,'string'))>0
    set(handles.pushbutton1, 'enable','on');
end
function edit2_Callback(hObject, eventdata, handles)
if length(get(handles.edit1,'string'))>0 & ...
length(get(handles.edit2,'string'))>0
    set(handles.pushbutton1, 'enable','on');
end
```

Пример 46. ASCII коды символов строки. Обратите внимание, что строка обрабатывается как вектор и на выходе получается числовой вектор, где каждый элемент соответствует одному символу строки

```
>> uint8('0123456789')
ans =
   48   49   50   51   52   53   54   55   56   57
```

полях содержится не нулевое значение, то после нажатия Enter на любом из полей кнопка становится активной. Функция length возвращает длину строки, и если было введено любое значение, то эта длина больше нуля. Оператор & – логическое И – обеспечивает проверку условия сразу для обоих текстовых полей.

Далее можно подметить, что и этот фрагмент кода не даёт гарантии безошибочной работы. Ведь в текстовые поля можно ввести любое значения, а для корректной работы функции требуется значение числовое. Для проверки числового значения достаточно вспомнить, что в ASCII таблице числа от 0 до 9 равны кодам с 48 по 57.



Рис. 29. После ввода значений в текстовые поля кнопка становится доступной

Пример 47. Обработчики события Callback текстового поля

```
function edit1_Callback(hObject, eventdata, handles)
if uint8(get(handles.edit1,'string'))>48 &...
uint8(get(handles.edit1,'string'))<57
    set(handles.edit2,'enable','on');
end
function edit2_Callback(hObject, eventdata, handles)
if uint8(get(handles.edit2,'string'))>48 &...
uint8(get(handles.edit2,'string'))<57
    set(handles.pushbutton1, 'enable','on');
end
```

Как уже отмечалось ранее, со строками в MATLAB можно работать, как с традиционными массивами. Организуем для строки проверку, является ли каждый её символ числом. Для начала запрограммируем последовательность ввода значений в текстовые поля. Пусть поле для ввода значения угла наклона будет недоступно до тех пор, пока в поле значения скорости не введено число. В функцию загрузки графического окна добавим команду set(handles.edit2,'enable','off'). До тех пор пока в первое поле не введено корректное значение, второе поле будет недоступным и визуально подсвечено желтым цветом. После срабатывания события Callback на первом поле второе поле становится доступным для ввода туда значения угла наклона. После проверки корректности ввода во второе поле доступной становится и кнопка для построения траектории. Соответствующие изменения в программном коде приведены в примере 47

Можно предусмотреть возможность построения на одних осях двух и более графиков. Как уже упоминали, существует специальная команда hold on, которая запрещает стирать линии текущего графика и новый строит рядом. Разместите в графическом окне элемент checkbox – флажок. Будем считать, что если флажок активирован, то на одном графике строятся две линии, если не активирован, то каждая новая линия затирает все предыдущие. В режиме конструктора сделайте надпись на флажке, например – оставить линию. В режиме редактора нужно добавить соответствующий код. Учитываться состояние флажка должно при обработке события нажатия кнопки. Элемент управления Checkbox имеет свойство Value, которое может принимать два значения – 0 и 1. Первое соответствует неактивированному флажку, второе – активированному. Модификация подфункции нажатия кнопки с учетом включения в модель флага приведена в примере 48.

Добавим в модель несколько возможностей по оформлению графика. Первое, что интересно, формирование заголовка осей координат. В примере со сценарием мы просто определили статичный заголовок. В данном случае можно сформировать его программно, учитывая различные параметры вновь рисуемого графика. Создадим заголовок в виде – "Траектория движения тела, брошенного со скоростью ... под углом ... к горизонту". Причём вместо многоточий поставим фактические значения параметров.

Пример 48. Обработчик события Callback командной кнопки, с учетом состояния checkbox

```
function pushbutton1_Callback(hObject, eventdata, handles)
v0=str2num(get(handles.edit1,'String'));
alpha=str2num(get(handles.edit2,'string'));
[x,y]=ballistika2(v0,alpha);
plot(x,y);
grid on;
if get(handles.checkbox1,'value')==1
    hold on;
else
    hold off;
end
```

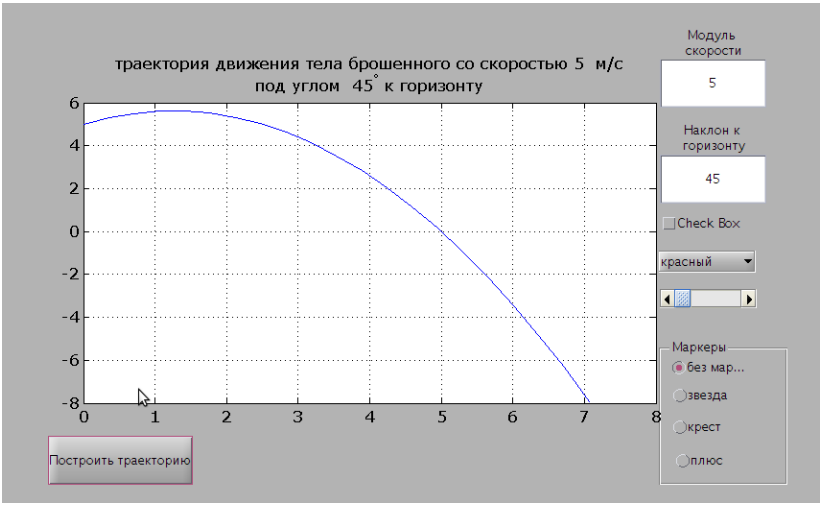


Рис. 30. Результат нажатия кнопки. Заголовок графика формируется программно.

Чтобы сформировать заголовок в две строчки, нам потребуется использовать массив ячеек – он создаётся с помощью фигурных скобок. Каждая из строк может быть создана несколькими способами – или функциями char, strcat или просто операторами – квадратными скобками, которые создают простой вектор из операндов.

Для корректного отображения символов кириллицы нужно установить имя шрифта, который поддерживает такую возможность. Наконец, для отображения символа градусов использована конструкция специального языка разметки – TeX. Символ \ начинает последовательность команд этого языка – нам нужно всего лишь разместить в верхнем индексе символ круга. Соответствующий набор команд представлен в примере 49.

Добавим возможность устанавливать цвет линии графика из заранее определенного списка. Для этого в графическом окне нужно разместить

Пример 49. Добавление заголовка осей координат и названия осей

```
set(gca,'FontName','Verdana','FontSize',14);
title1=['траектория движения тела брошенного со скоростью  ',...
 get(handles.edit1,'string'),'␣␣м/с'];
title2=['под углом ','␣',get(handles.edit2,'string'),...
'^\circ',' к горизонту'];
title({title1;title2});
```

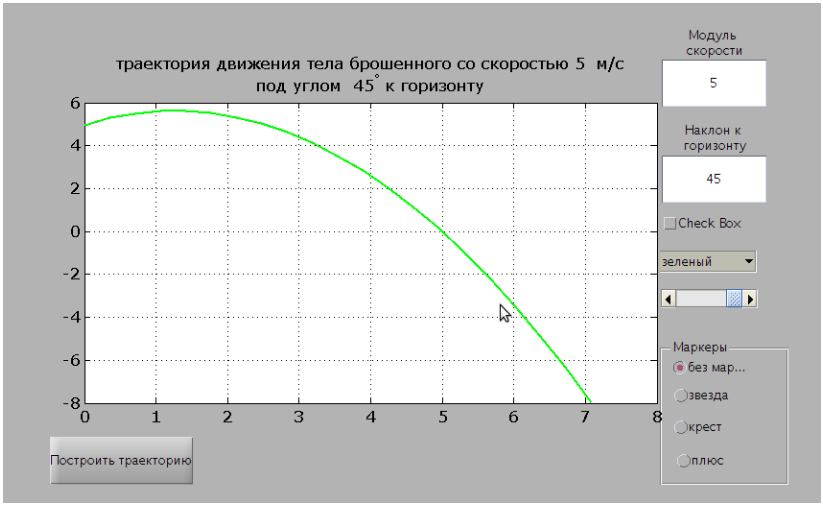


Рис. 31. Изменение цвета графика и толщины линии

Пример 50. Добавление возможности изменения цвета линии графика

```
function popupmenu1_Callback(hObject, eventdata, handles)
nom=get(hObject,'value');

switch nom
case 1
     set(handles.Line,'color','r');
case 2
     set(handles.Line,'color','g');
case 3
     set(handles.Line,'color','b');
end
```

один из двух элементов управления – listbox или pop-up menu. Различие будет заключаться только во внешнем виде – в первом из них все варианты выбора видны сразу же, и это удобно, когда их немного. Однако если их достаточно много, то удобнее пользоваться вторым вариантом – при этом в каждый момент на экране будет видна только одна строчка из всего списка, а при нажатии на элемент будет выпадать весь список. Рассмотрим вариант с размещением pop-up menu. Выберите этот элемент и перейдите в окно Property Inspector, в качестве свойства String установите столбец из трех значений – красный, зеленый, голубой. Выбор того или иного варианта и обработка их должны производиться при срабатывании события callback именно самого элемента pop-up menu, но до построения графика этот элемент должен быть недоступен. Нужно сделать pop-up menu недоступным до срабатывания нажатия кнопки. Установите соответствующее свойство pop-up menu при загрузке окна в 'inactive', а при нажатии кнопки – в 'on'. Обратите внимание: если присвоить свойству enable значение 'off', то выпадающий список будет не просто недоступен, а еще и невидим. Выбор варианта цвета линии организуем с помощью условного оператора case. Выбор будет производиться по значению свойства value выпадающего списка. Основная особенность состоит в использовании объекта Line внутри обработчика. Дело в том, что построение графика функции происходит в другой подфункции всей программы, а потому переменные, используемые там, являются недоступными для данной подфункции. В этом можно убедиться очень просто – измените программный код так, как показано в примере 50 и при попытке выбора цвета линии получите в консоли MATLAB сообщение об ошибке ??? Reference to non-existent field 'Line' – ссылка на несуществующее поле. Речь идёт о том, что структура handles не содержит поля с именем Line. Вспомните о том, что саму эту структуру GUIDE создает самостоятельно и помещает в значения полей имена только тех графических объектов, которые были созданы непосредственно в конструкторе. Для работы с объектом Line (даже если такой действительно существует после построения графика) необходимо вручную добавить соответствующее значение в структуру handles и обеспечить сохранение и доступность этого поля для всех подфункций. Для этого в обработчик события callback

Пример 51. Плавное изменение толщины линии с помощью элемента управления slider

```
function slider1_Callback(hObject, eventdata, handles);
width=get(hObject,'Value');
set(handles.Line,'LineWidth',round(width));
```

кнопки нужно добавить следующие строки:
handles.Line=plot(x,y);
guidata(gcbo,handles);
Команда guidata обеспечивает сохранение созданного поля структуры. После этого по команде handles.Line можно будет обращаться к объекту Line графика и всем его свойствам. После добавления строк запустите модель и убедитесь, что теперь всё работает вполне корректно.

Добавим теперь в модель объект slider. Свойство, которое можно менять этим объектом, по логике должно изменяться в некотором заданном интервале достаточно плавно. Выберем в качестве этого свойства толщину линии графика. У самого объекта slider есть свойство Value, которое может принимать значения от min до max с шагом, установленным в свойстве SliderStep. До запуска модели обязательно установите свойство value – текущее положение бегунка в любое значение между минимальным и максимальным. По умолчанию оно равно 0. В нашем случае выставим минимальное значение 1 (0 в качестве толщины линии нам не подходит), максимальное – 2. Значение value выставите равным минимуму. Для избежания ошибок в обработчике загрузки графического окна напишите – set(handles.slider1,'enable','inactive'), а в обработчике нажатия кнопки – set(handles.slider1,'enable','on'). Соответствующий код показан в примере 51.

И наконец, рассмотрим добавление и программирование радиокнопок. Для удобства их визуального расположения есть специальный элемент управления – button group. Разместите его в окне и внутри него 4 кнопки. Кнопки удобны для выбора только одного из всех вариантов. Для демонстрации их работы нужно выбрать такое свойство, которое меняется в конечном заранее определённом наборе значений с заранее известным шагом. Таким свойством являются, например, маркеры. Создайте для 4 кнопок подписи – без маркеров, звезда, крест, плюс. Для удобства параметр размера маркеров специально задан непропорционально большим. Поскольку мы уже определили структуру для глобального доступа ко всем свойствам графика, то программирование кнопок будет простой задачей. Для начала выставьте всем кнопкам свойства enable в 'off' и в событии нажатия кнопки "Построить траекторию" сделайте их, наоборот, активными. Поскольку все кнопки у нас находятся внутри специального объекта, то для доступа к ним нужно использовать событие объекта контейнера. Для выбора стиля

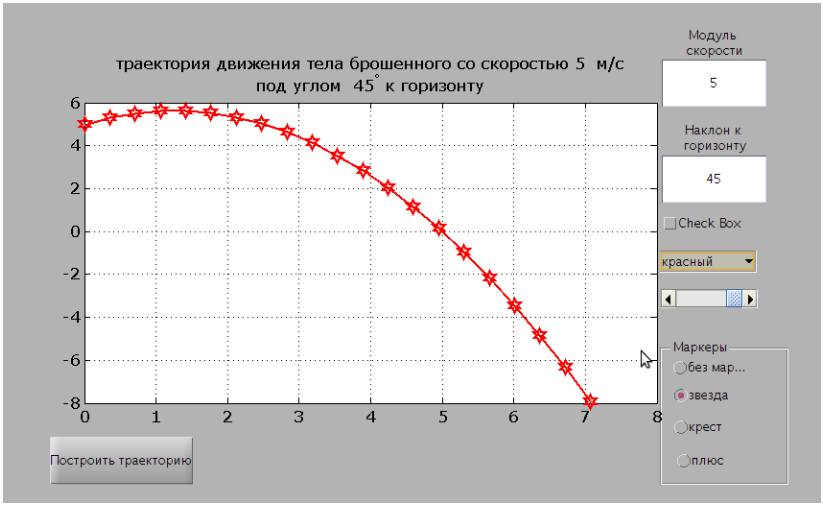


Рис. 32. Добавление маркеров на график

Пример 52. Обработка события выбора одной из группы радиокнопок

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'radiobutton1'
        set(handles.Line,'marker','none');
    case 'radiobutton2'
        set(handles.Line,'marker','h','markersize',12);
    case 'togglebutton1'
        set(handles.Line,'marker','x','markersize',12);
    case 'togglebutton2'
        set(handles.Line,'marker','+','markersize',12);
end
```

маркеров используем условный оператор case, выбор будем производить по найденному значению свойства tag вновь выбранной кнопки. Соответствующий фрагмент кода показан в примере 52.

Дальнейшие усовершенствования интерфейса могут быть связаны, например, с добавлением панели инструментов, которую видно на любом графическом объекте figure по умолчанию. С её помощью можно предусмотреть, например, вращение и увеличение графиков функций. Далее можно запрограммировать меню, в том числе и контекстные. И создание меню и создание панели инструментов максимально дружественно к пользователю. Вызовом соответствующих помощников в GUIDE можно вначале спроектировать их визуально, а потом в редакторе указать код, выполняемый при срабатывании событий.

6.3. Задания для самостоятельной работы

1. Добавьте в созданную в этой главе модель графическое изображение начальной скорости тела в виде линии со стрелкой.
2. Дополните созданную в этой главе модель построением вектора скорости в некоторых вычисляемых программно точках траектории.
3. Используя встроенные в функции MATLAB, по вычисленной траектории найдите высоту подъема тела и дальность его полета. Обеспечьте вывод этих величин с комментариями в графическое окно.
4. Создайте визуально и запрограммируйте панель инструментов и меню, обеспечивающие вашему приложению взаимодействие с операционной системой. Через это меню вы должны уметь сохранять созданную модель, печатать и прочее.
5. Создайте модель, которая будет на любом компьютере строить графическое окно с осями координат, занимающими весь экран.
6. Создайте приложение с графическим интерфейсом, которое позволяло бы построить график любой (в том числе и двумерной функции). Среди элементов интерфейса должно быть поле для ввода имени функции, интервала для абсцисс; элементы управления для оформления графика. Заголовок графика должен содержать имя введеной пользователем функции.

7. Заключение

В предложенном учебно-методическом комплексе рассмотрены базовые возможности интегрированной среды MATLAB. Для получения подробной информации о различных функциях MATLAB, следует обращаться к встроенной справочной системе. Организация справки такова, что позволяет запускать множество примеров, а большую часть программных кодов модернизировать для своих задач. Использовать MATLAB можно в решении любой научно-технической задаче, однако в данном курсе основное внимание было уделено тем возможностям, которые позволяют моделировать физические явления. Значительную часть содержания составляет описание визуализации данных в MATLAB и средств проектирования графического интерфейса. Это обусловлено тем, что наглядное представление физического явления важно как для понимания физики, так и для освоения MATLAB. Именно сочетание графического интерфейса с вычислительной мощью делают MATLAB наиболее эффективным средством в научных исследованиях и образовании.

8. Литература

1. Википедия свободная энциклопедия [Электронный ресурс] / wikipedia.org. Электрон. дан. Б.м., 2009. Режим доступа: http://wikipedia.org/, свободный.
2. Консультационный ценр MATLAB в компании Softline [Электронный ресурс] / matlab.exponenta.ru. Режим доступа: http://matlab.exponenta.ru/, свободный.
3. Форсайт Д.А., Понс Д. Компьютерное зрение. Современный подход. 2004. – 928 с.
4. Компьютерная Графика и Мультимедия Сетевой журнал [Электронный ресурс] / cgm.computergraphics.ru.
Режим доступа: http://cgm.computergraphics.ru, свободный.
5. Дьяконов В.П., MATLAB 7.*/R2006/R2007 Самоучитель. М.: ДМК, 2008. – 768 с.
6. Ануфриев И.Е., Смирнов А.Б., Смирнова Е.Н, MATLAB 7. Санкт-Петербург: БХВ-Петербург, 2005. – 1104 с.
7. Кетков Ю.Л., Кетков А.Ю., Шульц М.М, MATLAB 7: программировние, численные методы. Санкт-Петербург: БХВ-Петербург, 2005. – 752 с.
8. Потемкин В.Г. Вычисления в среде MATLAB М.: Диалог-МИФИ 2004. – 720 с.