МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

О.Г. Ревинская

Основы обработки данных в среде MatLab 2013

Часть 1

Учебное пособие

Томск
Издательский Дом Томского государственного университета
2015

УДК 519.682.2; 044.42
ББК 32.973.26-018.1я73
Р321

Ревинская О.Г.
Р321 Основы обработки данных в среде MatLab 2013. Ч. 1 : учебное пособие. – Томск : Издательский Дом Томского государственного университета, 2015. – 208 с.

Целью пособия является развитие навыков структурного программирования в среде MatLab с использованием графических возможностей этой системы. Пособие содержит краткое описание основных элементов языка программирования, а также ряда встроенных функций. Теоретический материал сопровождается большим количеством примеров использования различных конструкций языка при обработке числовых данных. Пояснения к примерам помогут студентам при самостоятельной подготовке индивидуальных заданий по программированию в среде MatLab.

Материал, изложенный в пособии, может применяться для обучения программированию в среде MatLab, независимо от ее версии. В качестве иллюстраций в пособие включены элементы интерфейса, используемого в MatLab, начиная с версии 2012 и выше.

Для студентов физико-математических и технических специальностей, изучающих дисциплину «Программирование» на младших курсах.

УДК 519.682.2; 044.42
ББК 32.973.26-018.1я73

Рецензенты:
академик Международной академии информатизации, д-р физ.-мат. наук,
зав. кафедрой прикладной математики Томского государственного архитектурно-строительного университета *С.Н. Колупаева*;
д-р техн. наук, профессор кафедры оптимизации систем управления
Национального исследовательского Томского политехнического университета
А.Ф. Тузовский

© О.Г. Ревинская, 2015
© Томский государственный университет, 2015

ПРЕДИСЛОВИЕ

MatLab (сокращение от англ. Matrix Laboratory) – пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. Он работает в большинстве современных операционных систем, включая Linux, Mac OS и Microsoft Windows.

MatLab как язык программирования был разработан Кливом Моулером в конце 1970-х гг. Цель разработки: дать студентам возможность использования программных библиотек Linpack и EISPACK без необходимости изучения Фортрана. Новый язык быстро распространился среди университетов и был с большим интересом встречен учёными, работающими в области прикладной математики. С 1984 г. компанией The MathWorks под руководством Джона Литтла и Клива Моулера развивается коммерческая версия MatLab. Первоначально MatLab предназначался для проектирования систем управления, но быстро завоевал популярность во многих других научных и инженерных областях. Он также широко используется в образовании, в частности для преподавания линейной алгебры и численных методов.

Язык MatLab является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах (двумерных массивах) структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и взаимодействие с программами, написанными на других языках программирования.

Глава 1. СРЕДА РАЗРАБОТКИ ПРОГРАММНЫХ РЕШЕНИЙ MATLAB

Перед тем как начать изучение языка программирования MatLab, необходимо овладеть элементарными навыками работы в интегрированной среде разработки программных решений и использования элементов пакета MatLab. Она объединяет в себе редактор, интерпретатор и отладчик, что позволяет писать, редактировать, выполнять программу и просматривать результаты ее работы, не покидая среды MatLab.

1.1. Запуск среды MatLab

Для запуска интегрированной среды MatLab необходимо активизировать исполняемый файл MatLab.EXE, входящий в комплект поставки. Способов его активизации может быть несколько в зависимости от настройки компьютера.

Если на компьютере установлена ***операционная система Windows***, то для запуска среды MatLab можно:

1. Сделать двойной щелчок левой клавишей мыши по ярлыку `MatLab` на рабочем столе Windows.
2. Нажать кнопку `Пуск` на панели задач и выбрать последовательно среди пунктов всплывающего меню `Все программы` → `MatLab` → `MatLab`.
3. Сделать двойной щелчок левой клавишей мыши на иконке уже существующего файла с программой, написанной на языке MatLab (на иконке файла с расширением `.m`, например `example1.m`).

В результате любого из этих действий после служебных сообщений появится основное окно среды MatLab.

1.2. Структура основного окна среды MatLab 2013

Интерфейс интегрированной среды MatLab, начиная с версии 2012, существенно отличается от интерфейса предыдущих версий, которые по структуре и виду интерфейсных элементов были практически одинако-

выми. Интерфейс версии MatLab 2012 наследуется всеми последующими версиями (2013, 2014, 2015 и т.д.). В пособии для иллюстрации материала использованы элементы интерфейса MatLab 2013, актуального и для последующих версий.

Основное окно среды MatLab 2013 состоит их трех различных по функциональному назначению частей (рис. 1.1): панель инструментов (верхняя часть), командное окно (центральная часть) и окна со вспомогательной информацией (справа и слева от командного окна).

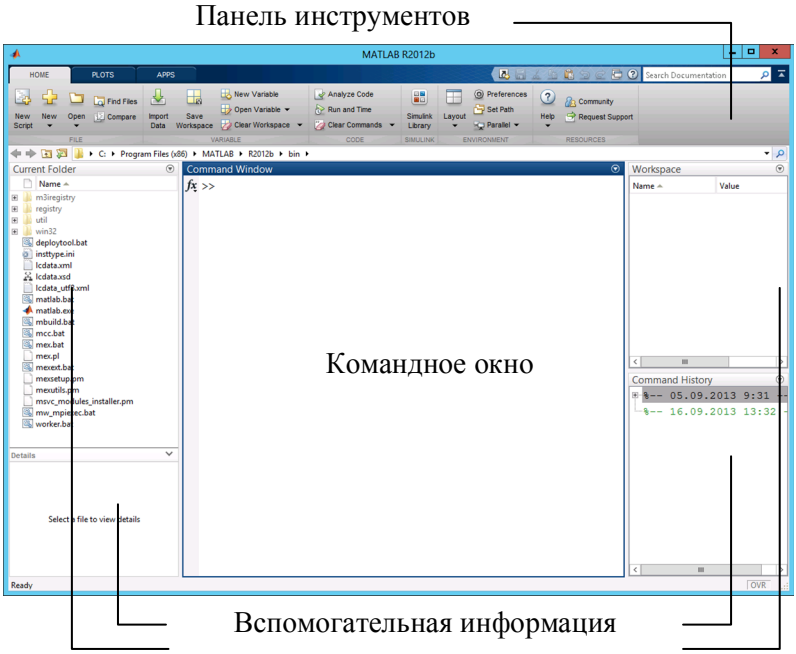


Рис. 1.1. Основное окно среды MatLab 2013

1.2.1. Панель инструментов среды MatLab 2013

Панель инструментов среды MatLab 2013 содержит кнопки для управления средой, тематически сгруппированные на трех вкладках:

Home, Plots, Apps (рис. 1.1). Кнопки на каждой из вкладок расположены на нескольких панелях. На панелях вкладки Home расположены следующие кнопки.

Панель File (содержит основные операции по управлению файлами):

- **New Script**: позволяет создать новый m-файл с управляющей программой (script-файл, сценарий);
- New: позволяет создать новый m-файл (как сценарий, так и функцию);
- **Open**: позволяет открыть сохраненный ранее m-файл;
- Find Files: позволяет искать сохраненные ранее файлы;
- Compare: позволяет сравнить содержимое двух файлов.

Панель Variable (содержит основные операции по управлению переменными в командном окне):

- Import Data: позволяет загрузить данные из файла;
- Save Workspace: позволяет сохранить значения используемых переменных в файле;
- New Variable: позволяет создать новую переменную;
- Open Variable: позволяет редактировать значения используемых переменных;
- **Clear Workspace**: позволяет удалить используемые переменные.

Панель Code (содержит основные операции по управлению программным кодом):

- Analyze Code: позволяет проанализировать код файлов в текущей папке;
- Run and Time: позволяет выполнить и измерить время выполнения программного кода;
- **Clear Commands**: позволяет очистить командное окно.

Панель Simulink (позволяет запускать библиотеку Simulink).

Панель Environment (содержит основные операции по управлению средой MatLab):

- Layout: позволяет настроить расположение и отображение рабочих окон среды;
- Preferences: позволяет выполнять специальные настройки среды;
- Set Path: позволяет изменить папку, которую среда использует для поиска файлов;
- Parallel: позволяет выполнять настройки среды при параллельных вычислениях.

Панель Resorurces (содержит основные операции для получения справочной информации):

- Help;
- Community;
- Request Support.

На панелях вкладки Plots расположены кнопки для построения различных графиков и диаграмм. На панелях вкладки Apps расположены кнопки для использования специальных пакетов и библиотек.

На отдельную панель (рис. 1.1) в виде пиктограмм вынесены инструменты, связанные с редактирование программного кода:

 создать новую команду – Create New;

 сохранить программный код в файл – Save (Ctrl-S);

 вырезать – Cut (Ctrl-X);

 скопировать – Copy (Ctrl-C);

 вставить – Paste (Ctrl-V);

 отменить команду – Undo (Ctrl-Z);

 отменить отмену команды – Restore (Ctrl-Y);

 показать все открытые окна.

При выборе некоторых кнопок на панели инструментов среды MatLab раскрывается дополнительный набор опций, выбор которых может осуществляться либо из списка, либо в режиме диалогового окна.

1.2.2. Командное окно среды MatLab 2013

Командное окно (Command Window) среды MatLab 2013 занимает центральную часть экрана (рис. 1.1) и предназначено для ввода программного кода и вывода результатов.

Программный код вводится в активной строке, отмеченной символом >>. Результаты выводятся в пустой строке после каждого фрагмента кода, не запрещающего отображение результата (см. рис. 1.2).

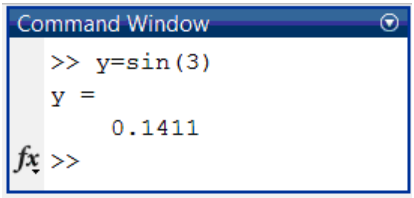


Рис. 1.2. Результаты вычислений в командном окне MatLab

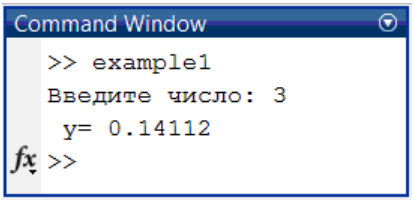


Рис. 1.3. Результаты выполнения в командном окне MatLab записанной в m-файле программы

Командное окно среды также используется для отображения результатов выполнения программ, записанных в m-файлах (сценариях, функциях). В этом случае вывод результатов выполнения записанной в m-файле программы может начинаться с отображения названия файла (рис. 1.3).

Инициировать выполнение программы можно, написав в командном окне название m-файла, в котором она хранится.

1.2.3. Информационные окна

Вспомогательные окна `Current Folder`, `Details`, `Workspace`, `Command History`, расположенные справа и слева от командного окна (рис. 1.1), содержат следующую информацию:

- окно `Current Folder` отображает содержание текущей папки;
- окно `Details` – дополнительную информацию;
- окно `Workspace` – список и текущее значение используемых переменных;
- окно `Command History` – историю выполнения программного кода, сгруппированную по датам.

1.3. Создание m-файла

MatLab позволяет создавать файлы с программным кодом двух типов: m-функции и m-сценарии (управляющие программы).

Программный код функции имеет фиксированную структуру и хранится в виде самостоятельного m-файла. Это позволяет использовать m-функции в других программах, написанных на языке MatLab, без до-

полнительного подключения. Методы написания и использования m-функций описаны в разделе 3.8.

Программный код, не являющийся описанием функции, сохраненный в самостоятельном m-файле, называется сценарием (script) или управляющей программой. Структура script-файла в MatLab не регламентируется.

Кнопка `New Script` на панели инструментов `File` (рис. 1.1) позволяет создать новый файл сценария (управляющей программы). В результате нажатия на кнопку `New Script` в отдельном окне запускается редактор m-файлов (рис. 1.4).

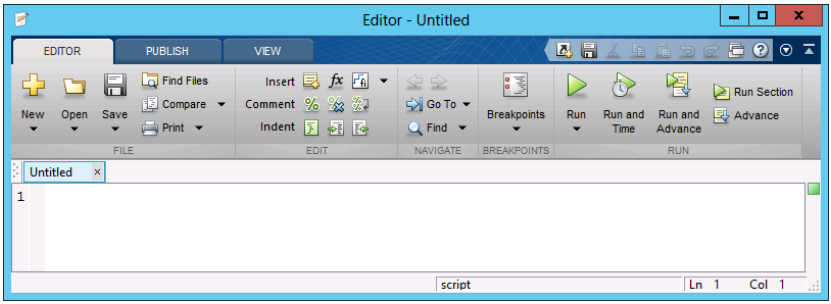


Рис. 1.4. Окно редактора m-файлов

Окно редактора содержит панель инструментов, а также область ввода и редактирования текста программного кода. Между панелью инструментов и текстом программного кода в виде вкладок расположен список открытых в данный момент файлов, который позволяет переключаться между файлами.

1.4. Редактирование программы

Если окно редактора m-файлов пустое, в нем можно набирать новый программный код. Если окно редактора содержит текст программы, его можно редактировать.

Мигающий курсор указывает то место на экране, в котором будет появляться текст.

Движение по тексту программы можно осуществлять с помощью манипулятора мышь, а также кнопок клавиатуры, например, ←, →, ↑, ↓, →

(на одну позицию влево, вправо, на одну строку вниз, вверх), PgUp, PgDn (на один экран вверх, вниз).

Нажатие кнопки Enter на клавиатуре приводит к вставке дополнительной строки в позиции курсора. Строки автоматически нумеруются.

Удалить неправильно набранный символ можно с помощью кнопок на клавиатуре: Del (удаляет символ в позиции курсора) и BackSpace (удаляет символ, расположенный перед курсором).

Эти и другие кнопки позволяют облегчить процесс набора и редактирования программы.

Редактор m-файлов позволяет копировать, переносить и стирать большие фрагменты программы. Выделить необходимый фрагмент можно либо перемещая мышь, удерживая нажатой левую клавишу, либо с помощью комбинации кнопок клавиатуры Shift – ↓ (→,↑,←). Далее можно воспользоваться пиктограммами редактирования (Cut, Copy, Paste) на дополнительной панели инструментов соответствующими комбинациями кнопок клавиатуры (Ctrl-X, Ctrl-C, Ctrl-V) или пунктами контекстного меню.

1.5. Сохранение программы на диске

Панель инструментов File редактора m-файлов содержит кнопку Save (рис. 1.4), позволяющую сохранить файл на рабочем диске компьютера. Раскрывающийся список, расположенный ниже кнопки Save, позволяет также воспользоваться для сохранения файла кнопками Save as..., Save All и т.д.

Операции Save соответствует комбинация кнопок клавиатуры Ctrl-S. Операции Save as... и Save All можно вызвать только с помощью раскрывающегося списка. Операции Save и Save as... применяются только к активному файлу, а операция Save All – ко всем открытым файлам сразу.

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

При повторном сохранении программы результаты применения кнопок Save и Save as... существенно отличаются. Повторное использование кнопки Save для сохранения программы с заданным именем

файла приводит к сохранению новой версии программы со старым именем файла. Использование кнопки `Save as...` для сохранения программы с уже заданным именем файла приведет к появлению еще одного файла, название которого следует ввести в появившемся диалоговом окне.

Кнопка `Save All` используется при работе с несколькими файлами, если есть уверенность, что все открытые файлы необходимо сохранить в том виде, в каком они существуют в настоящий момент, и с прежними названиями файлов.

1.6. Выполнение программы

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

Запуск программы на выполнение из редактора m-файлов осуществляется кнопкой `F5` на клавиатуре или с помощью кнопки `Run` на панели инструментов `Run` (см. рис. 1.4).

Если программа сохранена в папке, отличающейся от папки `Current Folder`, используемой по умолчанию, при первом выполнении программы MatLab предложит (рис. 1.5) или изменить папку, используемую по умолчанию (`Change Folder`), или добавить папку, в которой сохранен файл, к списку папок, используемых средой MatLab (`Add to Path`).

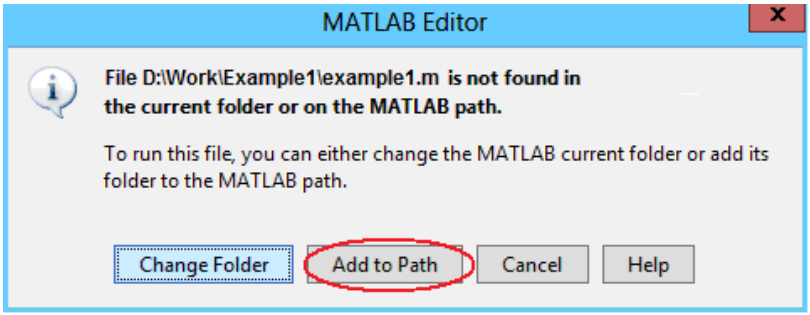


Рис. 1.5. Добавление папки, в которой сохранен файл, к списку папок, используемых средой MatLab по умолчанию

Перед выполнением текст программы автоматически переводится в машинные коды построчно. Этот процесс называется интерпретацией. Во время интерпретации происходит проверка синтаксиса программы. Если интерпретатор обнаружит в программе синтаксическую ошибку, он выдает соответствующее сообщение (Error…) в командном окне, указав номер строки, в которой произошла ошибка. После исправления ошибки необходимо вновь запустить программу на выполнение. Если синтаксических ошибок не обнаружено, строка программы выполняется, а интерпретатор переходит к анализу следующей строки.

Выполнение программы происходит в командном окне среды MatLab.

Если в процессе выполнения программы возникает необходимость прервать ее работу, можно воспользоваться комбинацией кнопок клавиатуры `Ctrl-Break`.

1.7. Просмотр результатов работы программы

Если командное окно среды MatLab не очищалось, результаты работы программы (т.е. выводимую ею информацию, а не сам текст программы) можно посмотреть не только в процессе, но и после выполнения программы в командном окне. Из основного окна среды MatLab в редактор m-файлов и обратно можно вернуться, стандартным образом переключаясь между окнами в операционной системе Windows.

1.8. Встроенная справочная система

В разрешении различных затруднений, возникающих при разработке программ, может помочь встроенная справочная система среды MatLab (`Help`). Справочная система среды MatLab содержит информацию о зарезервированных словах и операторах языка MatLab, директивах интерпретатора, стандартных функциях, ошибках и о самой среде MatLab. Вызов различных разделов справочной системы осуществляется с помощью кнопки `Help` на панели инструментов `Resources` (рис. 1.1) либо с помощью кнопки `F1` на клавиатуре.

Глава 2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ MATLAB

2.1. Структура программы

Структура программы, написанной на языке MatLab, определяется логической последовательностью операций, для выполнения которых пишется программа. Программа выполняется последовательно сверху вниз.

В языке MatLab, как в любом языке программирования, использование любого объекта (переменной, вектора, матрицы и т.д.) в качестве исходных данных требует, чтобы его значение было предварительно задано каким-либо способом.

В процессе выполнения любая программа, как правило, должна осуществлять обработку информации. Всякая обработка информации предполагает три этапа:

1. Ввод данных (информации).
2. Обработка данных (информации).
3. Вывод результатов.

Изменение последовательности этапов или исключение хотя бы одного из них, как правило, делает бессмысленными оставшиеся действия.

Поэтому даже для самых простых задач содержание программного кода (программы) тоже должно состоять из трех частей, реализующих каждый из этих этапов. При решении более сложных задач перечисленные этапы обработки информации могут выполняться несколько раз, но всегда в указанной последовательности – как логически завершенные блоки. Тогда результаты одного блока обработки информации могут выступать в качестве исходных данных для следующего блока.

Этап ввода данных реализуется, в основном, с помощью оператора `input`. Для вывода результатов используют оператор `disp`. Выбор операторов для реализации этапа обработки данных зависит от предполагаемого способа обработки информации. Это могут быть ***операторы присвоения, условных переходов, циклы*** и их комбинации. Отдельные фрагменты программы могут быть оформлены как ***функции***.

2.2. Идентификаторы

Идентификатор – уникальный набор символов, предназначенный для обозначения переменной, константы, команды или функции.

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

Идентификаторы подразделяются на стандартные и нестандартные (идентификаторы, выбираемые программистом). Часть идентификаторов зарезервирована в MatLab для обозначения стандартных команд, операторов и функций. Такие идентификаторы называют стандартными идентификаторами или зарезервированными словами. Использование стандартных идентификаторов не по назначению может вызвать ошибку при интерпретации программного кода.

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

Примеры нестандартных идентификаторов:

```
X           Tau           NAME2
x           tau           name2
X_min       Tau_Q         Name_Student
```

Идентификаторы, написанные строчными и прописными буквами, считаются разными.

2.3. Типы данных

Данные, используемые в программе, могут относиться к различным типам – числовые, символьные и т.д. в зависимости от содержания информации, хранящейся в них. В среде MatLab определено 6 базовых (стандартных) типов данных: `double`, `char`, `sparse`, `uint8`, `cell`, `struct`. Эти типы предназначены для хранения следующей информации:

- **`Double`** – числовые данные (как вещественные, так и целочисленные);
- **`Char`** – символьные данные (строки);
- `Sparse` – разреженные данные;
- `Cell` – ячейки;

❑ `Struct` – записи;

❑ `Uint8`, `Uint16`, `Uint32`, `Uint64` – целые числа без знака (для обработки изображений).

Для преобразования информации одного типа в другой существуют специальные функции.

2.4. Константы и переменные

Переменные – объекты, предназначенные для хранения информации фиксированного типа. Информация, хранящаяся в переменных, доступна для изменения в процессе выполнения программы. Идентификаторы переменных в пределах данной программы должны быть уникальными. В MatLab тип переменных автоматически задается интерпретатором в зависимости от типа информации, хранящейся в них.

В процессе выполнения программы список используемых ею переменных с указанием их типов и текущих значений автоматически отображается в окне `Workspace`. При необходимости этот список можно вывести в командное окно с помощью команды `whos`.

В MatLab переменные автоматически создаются в памяти (выделяется необходимый для хранения информации объем памяти) при первом употреблении переменной в тексте программного кода и не нуждаются в специальном определении. Для удаления переменных из памяти (освобождения памяти) можно воспользоваться командой `clear`.

Пример:

`clear S, r`	удаляет из памяти только переменные `S` и `r`;
`clear all`	удаляет все переменные, использованные ранее в среде MatLab.

Константы – неизменяемая информация фиксированного типа. Константы встречаются в программе либо в виде безымянных конкретных значений (числовых, символьных и т.д.), либо в виде стандартных (например, системных) идентификаторов. Стандартные константы не требуют предварительного задания их значений перед использованием.

Пример безымянных и стандартных констант:

`2.813`	числовая константа;
`'Петров'`	строковая константа;
`i`	мнимая единица ($\sqrt{-1}$);
`pi`	число π (3,141592653589793…).

2.5. Операции

Операции – действия, которые надо выполнить над переменными, константами. Операции в MatLab подразделяются на арифметические, логические, операции отношения и др. Операции с переменными (константами) записывают в строку по правилам, принятым в программировании, используя при необходимости стандартные функции и константы.

Правила записи арифметических операций

	Математика	**MatLab**
Сложение	a+b	a+b
Вычитание	a-b	a-b
Умножение	ab	a*b
Деление	$\frac{a}{b}$	a/b
Возведение в степень	a^b	a^b
Остаток от деления *a* на *b*		mod(a,b)
Остаток от деления *a* на *b*		rem(a,b)

Правила записи стандартных функций

	Математика	**MatLab**
Модуль числа	|x|	abs(x)
Косинус	cos x	cos(x)
Синус	sin x	sin(x)
Тангенс	tg x	tan(x)
Котангенс	ctg x	cot(x)
Арксинус	arcsin x	asin(x)
Арккосинус	arcos x	acos(x)
Арктангенс	arctg x	atan(x)
Экспонента	e^x	exp(x)
Натуральный логарифм	ln x	log(x)
Десятичный логарифм	lg x	log10(x)
Корень квадратный	$\sqrt{x}$	sqrt(x)
Знак числа		sign(x)

Аргументы тригонометрических функций и результаты обратных тригонометрических функций выражаются в радианах.

Правила записи операций отношения

	Математика	**MatLab**
Равно	a=b	a==b
Не равно	a≠b	a~=b
Больше	a>b	a>b
Меньше	a<b	a<b
Больше или равно	a≥b	a>=b
Меньше или равно	a≤b	a<=b
Совпадение строк		ctrcmp('s1','s2')

Правила записи стандартных констант

	Математика	**MatLab**
Встроенное число π	π	pi
Мнимая единица	*i*	i или j
Бесконечность	∞	Inf
Значение не задано		Nan

Правила записи преобразования типов

	MatLab
Округление до ближайшего целого	round(x)
Округление в сторону положительной бесконечности	ceil(x)
Округление в сторону отрицательной бесконечности	floor(x)
Округление в сторону нуля	fix(x)
Преобразование целого числа в его строковое представление	int2str(x)
Преобразование вещественного числа в его строковое представление	num2str(x)
Преобразование строковой информации в числовую	str2num(x)

Особые правила записи некоторых арифметических операций для массивов

	MatLab
Поэлементное сложение	`a.+b`
Поэлементное вычитание	`a.-b`
Поэлементное умножение	`a.*b`
Поэлементное деление	`a./b`
Поэлементное возведение в степень	`a.^b`

2.6. Комментарии

Любой программный код или текст, начинающийся со знака `%`, в MatLab считается комментарием (до конца строки). При выполнении программы интерпретатор пропускает комментарии.

Комментарии очень важны для пояснения текста программы. Информация, расположенная в комментариях, служит более глубокому пониманию текста программы и упорядочиванию ее текста. Комментарии будут наиболее эффективны, если содержат информацию о том, для решения какой задачи создана программа, кто автор программы, какие константы, переменные, команды и функции используются в программе и для какой цели.

Пример:

```
% Программа находит корни квадратного уравнения,
%       если ввести коэффициенты a, b, c
%          автор Свиридов В.К. гр. 531
```

Глава 3. ЭЛЕМЕНТЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Оператор – это условная запись определенного алгоритмического действия (команды).

Отличительной чертой MatLab является особая роль символа `;` (точка с запятой) для операторов, выполнение которых приводит к изменению значения хотя бы одной переменной. Вывод результата действия такого оператора осуществляется в командном окне автоматически сразу после выполнения данного оператора. Для подавления автоматического вывода результата выполнения оператора используется символ `;` (точка с запятой), который ставится в конце оператора (рис. 3.1).

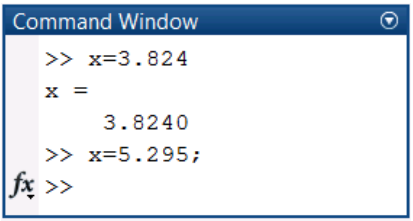


Рис. 3.1. Использование символа `;` для подавления вывода значения переменной в командном окне MatLab

Для очистки командного окна от выведенной ранее информации применяют оператор (команду) `clc`, который обычно используется в начале программного кода управляющего файла (script-файла, сценария).

Также в начале программного кода script-файла рекомендуется освобождать память, удалив все использованные ранее переменные с помощью оператора `clear all`. Это позволит избежать использования значений, полученных программой при ее предыдущем запуске.

3.1. Ввод информации с клавиатуры в диалоговом режиме

```
<переменная> = input ('<приглашение>');
```

Дойдя до оператора `input`, программа останавливается, ожидая окончания ввода данных с клавиатуры. При этом в командном окне появляется текст `<приглашения>`. Окончание ввода данных определяется по нажатию кнопки `Enter` на клавиатуре. Введенная информация запоминается в программе в указанной слева от оператора `input` (перед знаком =) `<переменной>`, и выполнение программы продолжится.

Пример (результат выполнения программного кода приведен на рис. 3.2):

```
x = input (‘ Введите значение переменной x: ’);
```

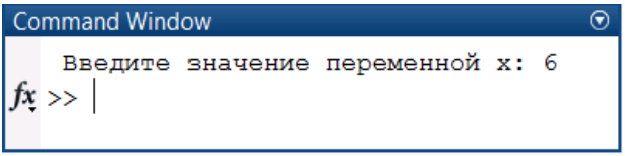


Рис. 3.2. Пример результата выполнения оператора `input`

```
<переменная> = menu (‘<заголовок>’, '<пункт1>',
                '<пункт2>' … '<пунктN>');
```

Оператор `menu` позволяет организовать выбор из нескольких возможных вариантов `<пункт1>`, `<пункт2>` … `<пунктN>`, предлагаемых пользователю в отдельном окне. Программа останавливается, ожидая, какой выбор сделает пользователь. Выбор осуществляется нажатием левой клавиши мыши на одной из кнопок, в виде которых изображаются отдельные пункты меню в данном окне. Указанной слева от оператора `menu` (перед знаком =) `<переменной>` присваивается целочисленное значение от 1 до *N* в зависимости от порядкового номера выбранного пункта меню.

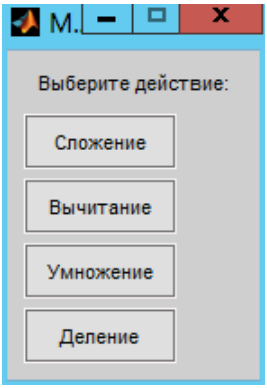


Рис. 3.3. Пример диалогового окна, появляющегося при выполнении оператора `menu`

Пример (результат выполнения программного кода приведен на рис. 3.3):

```
k = menu ('Выберите
   действие:', 'Сложение',
   'Вычитание', 'Умножение',
   'Деление');
```

3.2. Вывод информации

```
disp  (<переменная или текст>)
disp  ([<переменная> <переменная> ...])
```

Этот оператор выводит в командное окно либо значение переменной, либо текст – в скобках можно указать только один аргумент.

Пример:

```
disp (x)
disp('Я учусь программировать')
```

Чтобы вывести значения нескольких переменных, их необходимо предварительно объединить с помощью оператора объединения `[]`. Оператор объединения `[]` можно применять только к однотипным данным. Поэтому если переменные `x1`, `x2`, `x3` одного типа, например числовые, то их значения можно вывести в одной строке с помощью оператора `disp` следующим образом:

```
disp([x1 x2 x3])
```

В результате в командном окне будут выведены три числа в одной строке.

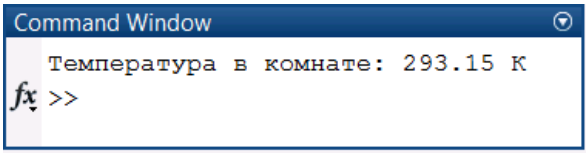


Рис. 3.4. Пример вывода в одной строке данных разного типа

Если в одной строке необходимо вывести данные разных типов, например, число и текст, то необходимо преобразовать эти данные к одному и тому же типу (строковому). После этого их можно объединить и вывести. Для преобразования числовых данных в строковые в MatLab существует две стандартные функции: `int2str(x)` – преобразует целое число `x` в его строковое представление; `num2str(x)` – преобразует вещественное число `x` в его строковое представление. Например, если переменная `T` содержит числовые (вещественные) данные, ее значение вместе с поясняющим текстом можно вывести следующим образом:

```
disp(['Температура в комнате:' num2str(T) ' К'])
```

Результат выполнения данного программного кода приведен на рис. 3.4.

Для вывода значений отдельных переменных можно также использовать ***вывод по умолчанию***. Для этого в тексте программы (или в командном окне) необходимо указать имя переменной без каких-либо операций (в том числе без символа ; в конце):

```
<переменная>
```

Пример (результат выполнения программного кода приведен на рис. 3.5):

```
x   % вывод переменной x по умолчанию
```



Рис. 3.5. Пример отображения в командном окне значения переменной x, полученного с помощь вывода по умолчанию

Вывод значения переменной по умолчанию (в отличие от вывода с помощью оператора `disp`) осуществляется в двух строках: 1) название переменной; 2) значение переменной.

Представление (вид) вещественных чисел в командном окне определяется используемым в MatLab по умолчанию форматом.

Числовые форматы

Название	Пример
Short	-0.3508
(десятичный формат **с фиксированной запятой** и 4 знаками после нее)	
Long	-0.350783227689620
(десятичный формат **с фиксированной запятой** и 15 знаками после нее)	
Hex	bfd6733b7eba621f
(шестнадцатиричный формат)	
Bank	-0.35
(десятичный формат **с фиксированной запятой** и 2 знаками после нее)	
+	-
(записывается только **знак числа)**	
Short E	-3.5078e-01
(экспоненциальный формат **с плавающей запятой** и 4 знаками после нее)	
Long E	-3.507832276896198e-01

Название	Пример
(экспоненциальный формат **с плавающей запятой** и 15 знаками после нее)	
`Short G`	`-0.35078`
(десятичный формат **с фиксированной запятой** и 5 знаками после нее)	
`Long G`	`-0.35078322768962`
(десятичный формат **с фиксированной запятой** и 14 знаками после нее)	
`Short Eng`	`-350.7832e-003`
(экспоненциальный формат **с плавающей запятой** и 4 знаками после нее)	
`Long Eng`	`-350.783227689620e-003`
(экспоненциальный формат **с плавающей запятой** и 12 знаками после нее)	
`Rat`	`17/3`
(запись **в виде рациональной дроби)**	

Формат `Short` (десятичный формат с фиксированной запятой и четырьмя знаками после нее) используется в среде MatLab по умолчанию. Для изменения представления чисел в командном окне используют команду `format`.

Пример (результат выполнения программного кода приведен на рис. 3.6):

```
format bank
disp(sin(3.5))
format short
disp(sin(3.5))
```



Рис. 3.6. Пример использования команды `format` для вывода значения `sin(3.5)`

Использование различных форматов отображения чисел НЕ ПРИВОДИТ к изменению их значений. Последний заданный формат используется далее в среде MatLab по умолчанию.

Действие команды `format` не распространяется на вывод вещественных чисел, преобразованных к строковому типу функцией `num2str`.

Пример сравнения результатов вывода оператором `disp` вещественного числа π и его строкового представления при включении различных числовых форматов (`shot` и `long`) (результат выполнения программного кода приведен на рис. 3.7):

```
format short
disp('включен format short')
disp (pi)
disp (num2str(pi))
format long
disp('включен format long')
disp (pi)
disp (num2str(pi))
```

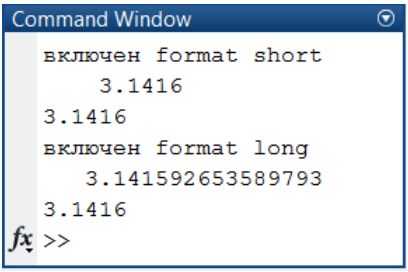


Рис. 3.7. Пример результатов использования функции `num2str` при включении различных числовых форматов

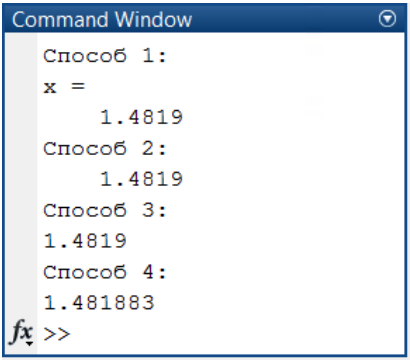


Рис. 3.8. Пример сравнения различных способов вывода значения скалярной вещественной переменной

Чтобы изменить формат представления числа, используемый функцией `num2str` по умолчанию, после преобразуемого числового значения или переменной необходимо указать дополнительную спецификацию:

`num2str(<число>, <спецификация>)`

`<число>` – преобразуемое числовое значение или переменная.

Дополнительная `<спецификация>` может задавать либо количество значащих цифр, которые необходимо отобразить в строковом представлении числа, либо формат представления числа. Количество значащих цифр задается целым числом. Например, `num2str(0.8621733,2)` или

num2str(0.8621733,5). В результате в первом случае после преобразования получится текст '0.86', а во втором – '0.86217'.

Формат преобразования записывается в виде строки, которая начинается со специального символа %. Далее указывают два числа, разделенные точкой: общее количество знаков в числе и количество знаков после запятой, которые должны будут присутствовать в полученном строковом представлении. (Например, 6.2 – для представления числа выделяется всего 6 знаков, из них 2 после запятой.) А в конце необходимо указать буквенный шифр числового формата: f – число с фиксированной запятой; e – число с плавающей запятой (экспоненциальный формат числа). Таким образом, формат для преобразования числа в строку, используемый в качестве дополнительной <спецификации> в функции num2str, может иметь вид '%6.2f', '%8.3e' и т.д.

Примеры использования различных форматов (спецификаций) в функции num2str для преобразования числа 0.008621733203 в строку:

Пример	Результат
num2str(0.008621733203,'%6.2f')	'0.01'
num2str(0.008621733203,'%6.5f')	'0.00862'
num2str(0.008621733203,'%6.7f')	'0.0086217'
num2str(0.008621733203,'%6.2e')	'8.62e-03'
num2str(0.008621733203,'%6.5e')	'8.62173e-03'
num2str(0.008621733203,'%6.7e')	'8.6217332e-03'

Использование дополнительной спецификации у функции num2srt позволяет выводить вещественные числа в нужном пользователю формате, не ограничиваясь форматом, установленным по умолчанию.

Пример.

Задание 3.1. Вещественное значение переменной x выведите четырьмя способами: 1) с помощью вывода по умолчанию; 2) с помощью оператора disp в числовом формате; 3) с помощью оператора disp в строковом формате по умолчанию; 4) с помощью оператора disp в десятичном формате с фиксированной запятой и шестью знаками после нее.

Программа (результат выполнения программного кода приведен на рис. 3.8).

```
disp('Способ 1:') % вывод по умолчанию
x
disp('Способ 2:')
% вывод оператором disp в числовом формате
disp(x)
disp('Способ 3:')
% вывод в строковом виде в формате по умолчанию
disp(num2str(x))
disp('Способ 4:')
% вывод в строковом виде в заданном формате
disp(num2str(x, '%9.6f'))
```

3.3. Оператор присвоения

<переменная> = <выражение>

<переменная> = <выражение>;

<Выражение> – это упорядоченная совокупность переменных, операций и функций. Значения всех используемых в выражении переменных должны быть предварительно заданы. Оператор присвоения предписывает программе выполнить заданную в <выражении> последовательность операций, а полученный результат записать в указанную слева от оператора присвоения (=) <переменную>. Если <переменная>, расположенная слева от оператора присвоения, встречается в программном коде впервые, то ее тип задается интерпретатором автоматически, в соответствии с типом результата вычисленного выражения. Если в выражении какая-либо переменная используется впервые в программном коде (заранее не задана), то возникает сообщение об ошибке.

В зависимости от типа информации, которую необходимо сохранить в расположенной слева от оператора присвоения переменной, <выражение> может содержать арифметические операции, стандартные функции, преобразование типов или логические операции.

Пример:

```
y = sqrt(x) + x    % y – числовая (вещественная) переменная
t = ['Вася ' 'Маша'] % t – символьная переменная (chart)
z = round(x)         % z – числовая (целая) переменная
Nq = num2str((a+b)/n)   % Nq – символьная переменная
```

Если после выражения в операторе присвоения стоит точка с запятой (;), то вычисленное значение в командном окне НЕ ОТОБРАЖАЕТСЯ.

Если точка с запятой (`;`) отсутствует, то после выполнения оператора присвоения в командном окне отображается результат вычислений.

Пример (результат выполнения программного кода приведен на рис. 3.9):

```
x = sin(2.42);
y = -x*x + x + 6
```

Пояснения. В данном случае значение переменной `x`, равное 0,6606, в командном окне не отображается.



Рис. 3.9. Пример результата выполнения оператора присвоения с и без точки с запятой в конце

Если в программном коде `<выражение>` записано без оператора присвоения, то результат вычислений автоматически присваивается специальной переменной `ans` (рис. 3.10). Это позволяет использовать командное окно как программируемый калькулятор.

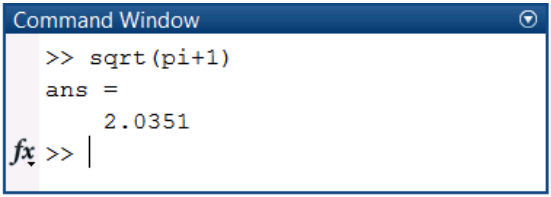


Рис. 3.10. Пример сохранения результата вычислений в специальной переменной `ans`

При обработке данных оператор присвоения = применяется, например, *при вычислении значений простых математических функций*.

Если для получения результата используется одна формула, вычисление можно провести с помощью одного оператора присвоения.

Пример.

Задание 3.2. Выполните вычисление по формуле $z_1 = \sqrt{m^2 + 2L}$.

Программа.

```
z1 = sqrt(m*m+2*L);
```

Пояснения. Учитывая этап ввода исходных данных (свободных параметров) и этап вывода результатов, текст программы, составленной для вычисления значения z_1, примет вид

Программа	*Пояснения*
`clc`	Очистка командного окна
`clear all`	Удаление всех использованных ранее переменных

`disp ('Свободные параметры:')`
`m = input (' m = ');`
`L = input (' L = ');`
	Ввод данных

`z1 = sqrt(m*m+2*L);`	Обработка данных
`disp ([' z1 = ' num2str(z1)])`	Вывод результатов

Если для получения результата нужно провести ряд предварительных вычислений, используют несколько операторов присвоения.

Пример.

Задание 3.3. Выполните вычисления по следующим формулам $y = R_1 R_2 + f_1 - f_2$, где $R_1 = \frac{f_1}{f_2}$, $R_2 = \sqrt{f_1^2 + f_2^2}$.

Программа.

```
R1 = f1/f2;
R2 = sqrt(f1*f1+f2*f2);
y = R1*R2+f1-f2;
```

Пояснения. Так как операторы выполняются последовательно сверху вниз, то в программе сначала следует провести промежуточные вычисления (`R1 = f1/f2; R2 = sqrt(f1*f1+f2*f2);`), а затем с их помощью получать окончательный результат (`y = R1*R2+f1-f2;`). Что-

бы данный программный код выполнялся корректно, необходимо перед ним организовать ввод исходных данных (свободных параметров `f1` и `f2`), а после него – вывод результата аналогично предыдущему примеру.

3.4. Оператор двоеточие (:)

`<начальное значение>:<шаг>:<конечное значение>`

Оператор двоеточие позволяет сформировать упорядоченную последовательность чисел в заданном интервале от `<начального значения>` до `<конечного значения>` с заданным `<шагом>` (рис. 3.11). `<Шаг>` может принимать как положительные, так и отрицательные значения. Если `<шаг>` не задан, то он считается равным единице. Границы и шаг числовой последовательности должны принадлежать к одному и тому же типу данных.

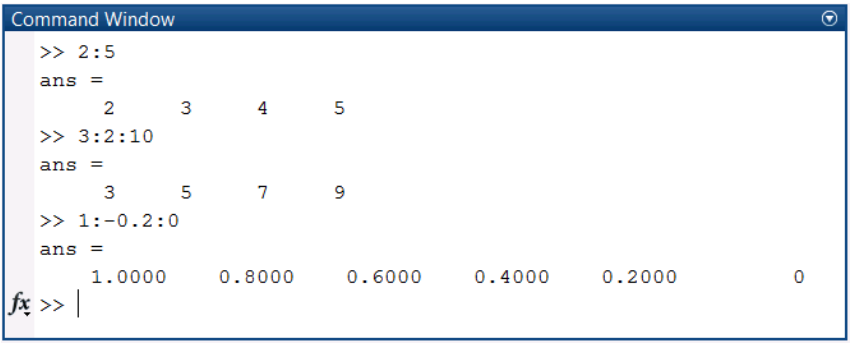


Рис. 3.11. Примеры получения числовых последовательностей с помощью оператора двоеточие в командном окне

3.5. Операторы условного перехода

В MatLab есть два оператора условного перехода: `if` и `switch`.

3.5.1. Оператор условного перехода IF

```
if <условие>
   <операторы 1>
else
   <операторы 2>
end
```

<Условие> – это выражение логического типа. <Условие> может быть записано в виде одной или нескольких операций отношения, а также в виде совокупности логических переменных. <Условия>, записанные в виде одной операции отношения или одной логической переменной, называют ***простыми***. <Условия>, записанные в виде нескольких операций отношения, называют ***составными***. Операции отношения, входящие в составное условие, должны быть заключены в круглые скобки и связаны логическими союзами И (`and, &`), ИЛИ (`or, |`), НЕ (`not, ~`) и т.д. Например, операции отношения, связанные логическим союзом И, записывают в следующем порядке: `(x > −3) & (x < 2)` или, что то же самое, `and(x > −3, x < 2)`. Правила записи составных условий с использованием остальных логических союзов аналогичны приведенным выше правилами использования союза И.

Результатом выполнения логической операции является либо `1` (истина – `True`), либо `0` (ложь – `False`).

Если <условие> верно (`True`), то выполняются операторы, расположенные после условия вплоть до служебного слова `else`. Если условие не выполнено (`False`), то выполняются операторы, расположенные между служебными словами `else` (иначе) и `end`. А затем выполнение программы продолжается дальше.

Если содержанием задачи предусмотрены действия, которые необходимо выполнить, только когда <условие> верно, то блок `else` в операторе `if` можно опустить.

```
if <условие>
   <операторы>
end
```

В этом случае если условие не выполнено (`False`), то программа выполняется дальше, будто условного оператора нет вовсе.

Если в случае неверного (False) выполнения <условия 1> необходимо проверить какое-то дополнительное <условие 2>, то оператор условного перехода записывают следующим образом:

```
if <условие 1>
   <операторы 1>
elseif <условие 2>
       <операторы 2>
    else
       <операторы 3>
end
```

В любом из исполняемых блоков оператора if (<операторы>; <операторы 1>; <операторы 2>; <операторы 3>) может быть заключено любое количество операторов. В зависимости от истинности (ложности) условия выполняется вся группа операторов, расположенных в соответствующем блоке. Среди операторов в любом из исполняемых блоков также может встречаться оператор if с другим условием. Такие операторы называются вложенными. Вложенные операторы должны целиком входить внутрь соответствующего блока (вложенный оператор должен целиком находиться внутри одного и того же исполняемого блока другого оператора, включая служебное слово end и соблюдение всех синтаксических правил). Обычно использование ветвления elseif и вложенных операторов приводит к уменьшению количества проверок.

При обработке данных оператор условного перехода if применяют, например, *при проверке соответствия аргументов функции ее области определения* (знаменатель не должен обращаться в ноль, корни извлекаются только из неотрицательных чисел и т.д.), *которая выполняется перед вычислением значения функции.*

Пример 1 (простое условие).

Задание 3.4. Получите вещественное значение по формуле $f = \sqrt{a+b}$.

Программа, вариант 1.

```
if a+b>=0
   f = sqrt(a+b);        % вычисление без вывода
   disp(['f = ' num2str(f)])      % вывод
else
   disp ('Ошибка: корень из отрицательного числа')
end
```

Программа, вариант 2.

```
if a+b<0
  disp ('Ошибка: корень из отрицательного числа')
else
  f = sqrt(a+b)          % вычисление с выводом
end
```

Пример 2 (составное условие):

Задание 3.5. Выполните вычисление по формуле $f = \sqrt{a-b} + 10\ln(a+3)$.

Программа, вариант 1.

```
if (a-b>=0) & (a+3>0)  % или and(a-b>=0, a+3>0)
   f = sqrt(a-b)+10*log(a+3)
else
   disp ('Ошибка: аргументы не принадлежат обл.опр.')
end
```

Программа, вариант 2

```
if (a-b<0) | (a+3<=0)  % или or(a-b<0, a+3<=0)
   disp ('Ошибка: аргументы не принадлежат обл.опр')
else
   f = sqrt(a-b)+10*log(a+3)
end
```

Оператор условного перехода `if` применяют также *при вычислении кусочно-непрерывных функций*.

Пример 1 (простое условие).

Задание 3.6. Выполните вычисление по формуле $f = \begin{cases} 3, & \text{при } x \ge 1; \\ -7, & \text{при } x<1. \end{cases}$

Программа, вариант 1.

```
if x>=1
   f = 3
else
   f = -7
end
```

Программа, вариант 2.

```
if x<1
   f = -7
else
   f = 3
end
```

Если же кусочно-непрерывная функция имеет более двух частей, следует пользоваться условным оператором `if` с ***составными условиями***, которые содержат два или более простых условий, объединенных логическими союзами И (`and`, `&`), ИЛИ (`or`, `|`), НЕ (`not`, `~`) и т.д.

Пример 2 (составные условия).

Задание 3.7. Выполните вычисление по формуле:

$$f = \begin{cases} -6x & \text{при} \quad x < -3; \\ x & \text{при} \quad -3 \le x < 2; \\ 5{,}1x & \text{при} \quad x \ge 2. \end{cases}$$

Программа, вариант 1. Набор последовательных независимых операторов `If`.

Программа	*Пояснения*

`if (x >= -3) & (x < 2)`
`  f = x`
`end`
	Составное условие.
	Эквивалентная запись условия:
	`and(x >= -3, x < 2)`

`if x < -3`
`  f = -6*x`
`end`
	Простое условие

`if x >= 2`
`  f = 5.1*x`
`end`
	Простое условие

Программа, вариант 2. Использование разветвленного оператора `If`.

Программа	*Пояснения*
`if (x >= -3) & (x < 2)` `  f = x`	Проверка первого (основного) условия
`elseif x < -3` `  f = -6*x`	Проверка дополнительного условия

`else`
`  f = 5.1*x`
`end`
	В остальных случаях

3.5.2. Оператор выбора SWITCH

```
switch <ключ>
case <значение 1>
     <операторы 1>
case <значение 2>
     <операторы 2>
. . . . . .
case <значение N>
     <операторы N>
otherwise
     <операторы>
end
```

<Ключ> – переменная или выражение, значением которого является скаляр или строка. По значениям <ключа> осуществляется выбор.

<Значение 1>, <значение 2>, ... <значение N> – возможные значения <ключа>, каждому из которых ставится в соответствие определенная последовательность действий <операторы 1>, <операторы 2>, ... <операторы N> соответственно.

В противном случае (если <ключ> принимает значение, не указанное ни в одном из блоков case) выполняются операторы, расположенные после служебного слова otherwise (другие случаи). Если <ключ> принимает значение, не указанное ни в одном из блоков case (случай), а блок otherwise отсутствует, оператор case пропускается.

Заголовок каждого из блоков case <значение> записывается в отдельной строке. Если <ключ> принимает <значение K>, то выполняются все операторы, расположенные после заголовка блока case <значение K> и до заголовка следующего блока. К одному блоку case (случаю) может относиться множество значений, которые перечисляются в фигурных скобках через запятую.

Пример:

```
switch x
case 1
     y = 10
case {3,7,11,15}
     y = x - 0.5
otherwise
     y = sin(x)
end
```

Множество значений вида {a, a+d, a+2d, … b} можно создать из числовой последовательности a a+d a+2d … b с помощью преобразования типов num2cell (преобразования числа в ячейку типа cell). Для формирования числовой последовательности используют оператор двоеточие (:). Выражение вида a:d:b позволяет получить числовую последовательность, а выражение num2cell(a:d:b) – множество значений, которое может использоваться, при необходимости, как одно из значений <ключа> в операторе switch.

Пример:

```
switch x
case 1
     y = 10
case num2cell(3:4:15)
     y = x - 0.5
otherwise
     y = sin(x)
end
```

Оператор выбора switch применяют *при вычислении кусочно-непрерывных функций*. В большинстве случаев при использовании оператора switch программа для вычисления кусочно-непрерывной функции становится более наглядной и краткой по сравнению с использованием оператора if. Однако возможности использования оператора switch ограничиваются его применением только для <ключей>, значением которых являются скаляры или строки.

Пример.

Задание 3.8. Для целых значений переменной x выполните вычисление по формуле.

Формула	Программа

$f = \begin{cases} -4x+2 & \text{при } -10 \le x < -3; \\ \lvert x \rvert - 1 & \text{при } -3 \le x < 2; \\ 6 - x & \text{при } 2 \le x < 10. \end{cases}$	switch x
	case num2cell(-10:-4)
	  f = -4*x+2
	case num2cell(-3:1)
	  f = abs(x)-1
	case {2,3,4,5,6,7,8,9}
	  f = 6 - x
	end

Пояснения. В этом примере запись `{2,3,4,5,6,7,8,9}` равносильна `num2cell(2:9)`.

Оператор выбора `switch` незаменим *при организации пользовательского меню* (при необходимости совершать различные действия в зависимости от ситуации). Выбор может осуществлять либо пользователь программы (тот, кто с ней работает), либо сама программа в зависимости от результатов предыдущих действий.

Если *меню разрабатывается для пользователя*, то перед оператором `switch` необходимо с помощью оператора вывода (`disp`) вывести список возможных действий («меню») и предложить пользователю сделать выбор, в результате чего вводится значение `<ключа>` с помощью оператора ввода (`input`). Ввод ключа может быть также организован с помощью оператора `menu`.

Пример.

Задание 3.9. Для значений свободных параметров `a` и `b` выполните одно из трех действий: сложение, вычитание, умножение (по выбору пользователя).

Программа, вариант 1. Выбор действия осуществляется с помощью оператора `input`.

Программа	*Пояснения*

`disp ('Возможные действия:')`
`disp ('1 – сложение')`
`disp ('2 – вычитание')`
`disp ('3 – умножение')`
	Подсказка для пользователя

`n = input ('Выберите: ');`	Вводим значение `<ключа>`

`switch n`
`case 1`
`  y = a+b`
`case 2`
`  y = a-b`
`case 3`
`  y = a*b`
`end`
	Вычисления (результат выбора)

Программа, вариант 2. Выбор действия осуществляется с помощью оператора `menu`.

Программа	*Пояснения*
`n = menu ('Возможные действия:', 'Сложение', 'Вычитание', 'Умножение');`	Вводим значение <ключа> в отдельном окне

`switch n`
`case 1`
`  y = a+b`
`case 2`
`  y = a-b`
`case 3`
`  y = a*b`
`end`
	Вычисления (результат выбора)

Если *выбор осуществляет сама программа по результатам предыдущих вычислений*, то возможный список действий (меню) на экране не отображается, а отображается лишь результат выбора.

Пример.

Задание 3.10. В зависимости от величины остатка от деления числа `b` (количество яблок) на число `a` (количество человек) определите, какое количество яблок останется не поделенным.

Программа	*Пояснения*
`y = rem(b,a);`	Вычисление значения <ключа>

`switch y`
`case 0`
`  disp ('Все яблоки поделены')`
`otherwise`
`  disp(['Не поделенными останутся '`
`  int2str(y) ' яблок(a)'])`
`end`
	Результат выбора

Конструкции операторов `if` и `switch` могут заменять друг друга. Предпочтение следует отдавать тому оператору, который позволит сделать программу более наглядной и легкой для восприятия.

3.6. Операторы цикла

В MatLab существуют два оператора цикла, которые позволяют различным образом организовывать повторение каких-либо действий.

3.6.1. Оператор FOR-цикла

```
for <параметр цикла> = <S1>:<dS>:<S2>
   <операторы>
end
for <параметр цикла> = <S1>:<S2>
   <операторы>
end
```

<Параметр цикла> (счетчик) – переменная, значения которой оператор цикла изменяет автоматически от начального значения <S1> до конечного <S2> с шагом <dS>. Для организации изменения <параметра цикла> используется оператор двоеточия. Следовательно, значения <параметра цикла> представляют собой числовую последовательность. Границы и шаг изменения <параметра цикла> должны принадлежать к одному и тому же типу данных. Шаг изменения <параметра цикла> может быть как положительным, так и отрицательным, как целым, так и дробным. Если шаг изменения параметра цикла явно не указан, он считается равным единице.

Для каждого значения параметра цикла (счетчика) выполняются <операторы>, расположенные после указания границ изменения <параметра цикла> и вплоть до служебного слова end.

При заданных значениях границ и шага изменения параметра цикла <операторы> в цикле выполняются фиксированное количество раз. Если границы и шаг изменения параметра цикла записаны через переменные, то их значения должны быть заданы до начала цикла.

Пример:

```
n = 3;
for k = 1:n
   a = input (' a = ');
   x = 2 + a;
   disp ([' X = ' num2str(x)])
end
```

Параметры for-цикла, его границы и шаг с помощью <операторов>, расположенных внутри цикла, изменять нельзя.

3.6.2. Оператор цикла WHILE

```
while  <условие>
   <операторы>
end
```

<Условие> – логическое выражение. Блок <операторов>, расположенный между <условием> и служебным словом end, будет выполняться только в том случае, если <условие> принимает значение «истина» (True). Когда <условие> перестает выполняться (False), цикл заканчивается.

Чтобы организовать цикл while конечной длины, необходимо: 1) программный код, предшествующий циклу, организовать так, чтобы при первой проверке <условия> оно имело значение «истина»; 2) блок операторов, выполняемых в цикле, организовать так, чтобы при повторной проверке <условия> хотя бы одна из используемых в логическом выражении переменных имела другое значение (по сравнению с предыдущей проверкой). То есть значение хотя бы одной переменной, проверяемой в <условии>, должно изменяться внутри цикла и должно быть задано перед циклом. Тогда спустя какое-то количество повторений условие перестанет выполняться и цикл закончится.

Пример:

```
x = 0;
while x < 100
   a = input (' a = ');
   x = x + abs(a);
   disp ([' X = ' num2str(x)])
end
```

3.6.3. Особенности использования операторов циклов FOR, WHILE

Сравним действия операторов циклов for и while. Конструкция цикла while позволяет организовать цикл, не вводя счетчик. Количество повторений зависит от результатов операций, выполненных в цикле. При организации for-цикла всякие действия по изменению параметра цикла внутри цикла запрещены. Использование for-цикла очень удобно при работе со счетными множествами (например, с массивами). Цикл while следует использовать, когда необходимо проверять условие. Надо пом-

нить, что проверка условия происходит в начале цикла. Таким образом, если при первой проверке логическое условие не выполнено, то цикл `while` не выполнится ни разу.

Выбор того или иного оператора цикла зависит от условий решаемой задачи. Причем в одних ситуациях использование определенного вида оператора является принципиальным, а в других – операторы могут быть взаимозаменяемыми.

Например, если известно, что *действие необходимо совершить n раз*, удобнее всего использовать оператор `for`-цикла. Конструкции с использованием `while`-цикла будут более громоздкими, но столь же эффективными. В `for`-цикле счетчик (параметр цикла) меняется автоматически, в `while`-цикле его нет, поэтому, если это необходимо, об изменении значения счетчика должен позаботиться программист. В `while`-цикле изменение значения счетчика (например, `k=k+1;`) должно располагаться в конце цикла, чтобы «приготовить» значение счетчика `k` для следующего выполнения цикла.

Пример.

Задание 3.11. Вычислите и выведите на экран значения функции $f = \cos 0{,}3\pi k$ для всех значений k от 1 до n (n раз).

Программа, вариант 1. С использованием **`for`-цикла**.

Программа	*Пояснения*
`for k = 1:n`	Начало цикла; начальное и конечное значение `k`, изменение значения `k` на единицу
`  f = cos(0.3*pi*k);` `  disp([' f= ' num2str(f)])`	Действия, которые надо выполнить (вычисление и вывод)
`end`	Конец цикла

Программа, вариант 2. С использованием **`while`-цикла**.

Программа	*Пояснения*
`k = 1;`	Начальное значение `k`
`while k<= n`	Начало цикла; условие продолжения цикла
`  f = cos(0.3*pi*k);` `  disp([' f= ' num2str(f)])`	Действия, которые надо выполнить (вычисление и вывод)
`  k = k+1;`	Изменение значения `k` на единицу
`end`	Конец цикла

Рассмотрим другой пример: *вычисление значений функции f(x) на интервале [a,b] с шагом dx*. Аргумент функции должен изменяться от *a* до *b* с шагом *dx*.

При решении этой задачи с помощью `while`-цикла необходимо организовать изменение значения аргумента функции `x=x+dx;` в цикле. Изменение значения переменной `x` должно происходить в конце цикла, чтобы «приготовить» значение `x` для следующего выполнения цикла.

Задание 3.12. Вычислите и выведите на экран значения функции $f = \cos \pi x$, аргумент которой изменяется на интервале $[a, b]$ с шагом dx.

Программа, вариант 1. С использованием **`while`-цикла**.

Программа	*Пояснения*
`x=a;`	Начальное значение `x`
`while x<=b`	Начало цикла; условие продолжения цикла
`f = cos(pi*x);` `disp([' f= ' num2str(f)])`	Вычисление значения функции для текущего значения аргумента; вывод результата
`x=x+dx;`	Изменение значения `x` на `dx`
`end`	Конец цикла

При организации вычислений с помощью оператора `for`-цикла в качестве параметра цикла удобно использовать аргумент функции. Шаг изменения параметра цикла (`dx`) необходимо указать явным образом. Тогда изменение аргумента функции будет выполняться автоматически.

Программа, вариант 2. С использованием **`for`-цикла**.

Программа	*Пояснения*
`for x=a:dx:b`	Начало цикла; автоматическое изменение переменной `x` от `a` до `b` с шагом `dx`
`f = cos(pi*x);` `disp([' f= ' num2str(f)])`	Вычисление значения функции для текущего значения аргумента; вывод результата
`end`	Конец цикла

Теперь рассмотрим пример, когда использование оператора цикла определенного вида является принципиальным. Наиболее наглядной в этом смысле является следующая задача: *вычислять значения функции до тех пор, пока функция не станет больше определенного, заранее заданного числа (обозначим его через z).*

В данном случае `for`-цикл использовать невозможно, потому что нельзя указать, в каких пределах нужно изменять счетчик `for`-цикла. Для решения этой задачи следует использовать `while`-цикл. Условие в цикле должно соответствовать случаю, когда вычисления следует продолжить, т.е. `f <= z`. Проверка условия происходит в начале цикла. Если значение функции вычисляется внутри цикла, то при проверке условия `f <= z` используется значение `f`, вычисленное в процессе предыдущего выполнения цикла. Перед первой проверкой условия значение функции еще не было получено, поэтому до цикла необходимо задать такое (придуманное) значение `f`, которое обеспечит выполнение условия `f <= z`, например `f = z - 1`.

Задание 3.13. Пока значение функции не станет больше z, вычисляйте и выводите на экран значения функции $f = x + \cos \pi x$ для таких значений аргумента x, которые изменяются начиная с $x = a$ с шагом dx.

Программа. С использованием **`while`-цикла**.

Программа	*Пояснения*
`x = a;`	Начальное значение `x`
`f = z-1;`	Значение `f`, обеспечивающее «вход» в цикл
`while f<=z`	Проверка условия (сравнение текущего значения `f` со значением `z`)
`f = x+cos(pi*x);` `disp([' f= ' num2str(f)])`	Вычисление значения функции для текущего значения аргумента; вывод результата
`x = x+dx;`	Изменение значения `x` на `dx`
`end`	Конец цикла

Задание 3.14. Пока значение функции не станет больше z, вычисляйте и выводите на экран значения функции $f = x + \cos \pi x$ для ее аргументов x, значения которых вводятся с клавиатуры.

Программа. С использованием **while-цикла**.

Программа	*Пояснения*
`f = z-1;`	Значение `f`, обеспечивающее «вход» в цикл
`while f<=z`	Проверка условия (сравнение текущего значения `f` со значением `z`)
`x = input (' x = ');`	Ввод значения `x`
`f = x+cos(pi*x);` `disp([' f= ' num2str(f)])`	Вычисление значения функции для текущего значения аргумента; вывод результата
`end`	Конец цикла

Подробный анализ этих и ряда других примеров приведен в главе 5.

3.6.4. Вычисление сумм и произведений

Одной из наиболее распространенных задач физики и математики является вычисление сумм и произведений, которые с помощью математических символов суммирования **Σ** и перемножения **Π** можно записать в виде

$$S = f_1 + f_2 + f_3 + ... + f_n = \sum_{k=1}^{n} f_k,$$

$$P = f_1 \cdot f_2 \cdot f_3 \cdot ... \cdot f_n = \prod_{k=1}^{n} f_k.$$

Например, вычисление факториала числа n можно записать в виде

$$n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n = \prod_{k=1}^{n} k \ .$$

Вычисление суммы n членов арифметической прогрессии, которая характеризуется первым членом a_1 и разностью d:

$$S_n = a_1 + a_2 + a_3 + ... + a_n = \sum_{k=1}^{n} a_k = \sum_{k=1}^{n} (a_1 + d(k-1)) \ .$$

Для программирования сумм и произведений используется один и тот же подход. Алгоритм (последовательность действий) вычисления суммы выглядит одинаково независимо от формы записи конкретного слагаемого f_k.

Алгоритм вычисления суммы $S = \sum_{k=1}^{n} f_k$

Вычисление суммы осуществляется путем последовательного добавления к текущему значению суммы очередного слагаемого, пока все необходимые слагаемые не будут добавлены. Обозначим через `S` текущее значение суммы (сумму `k - 1` слагаемого).

1. Добавив к текущему значению суммы `k`-е слагаемое, получим сумму из `k` слагаемых. Если полученное значение вновь обозначить через `S`, то его можно использовать для дальнейшего накопления суммы.

   ```
   S = S + fk
   ```

   На языке программирования такая запись читается справа налево и означает: из переменной с именем `S` возьмем значение, добавим к нему значение `fk`, а результат вновь поместим в переменную с именем `S`. Через `fk` обозначена формула, которой выражается `k`-е слагаемое для данной суммы.

   Выражения, с помощью которых новое значение некоторой величины вычисляется через ее же одно или несколько предыдущих значений, называется ***рекуррентным***. Рекуррентное выражение (формула) `S = S +...` является важной частью алгоритма суммирования!
2. Повторяя пункт 1 для всех слагаемых с номерами от `1` до `n`, получим сумму `n` слагаемых.
3. Первое слагаемое, как и все последующие, добавляются к предыдущему значению суммы `S`. Чтобы какие-то другие операции, выполняемые программой ранее, не повлияли на результат, до начала суммирования необходимо задать такое значение `S`, которое исключит попадание в сумму каких-либо посторонних значений:

   ```
   S = 0
   ```

Этот алгоритм можно реализовать с помощью любого типа цикла. Наиболее распространенной и легким для осмысления является органи-

зация суммирования с помощью `for`-цикла, так как в этом цикле изменение номера текущего слагаемого от `1` до `n` происходит автоматически:

```
S = 0;           % начальное значение S
for  k = 1:n     % цикл по номерам слагаемых
     S = S+fk;   % рекуррентная формула
end
```

Эта конструкция имеет наиболее простой для понимания и запоминания вид.

Вычисление произведения $P=\prod_{k=1}^{n} f_k$

Алгоритм вычисления произведения аналогичен алгоритму вычисления суммы. На каждом шаге текущее значение произведения `P` домножается на `k`-й сомножитель `P = P * fk` (рекуррентная формула). При этом начальное значение произведения `P` (до начала цикла) необходимо задать равным единице: `P = 1`. Тогда алгоритм вычисления произведения с использованием `for`-цикла схематично можно записать в виде

```
P = 1;           % начальное значение P
for k = 1:n      % цикл по номерам сомножителей
    P = P * fk;  % рекуррентная формула
end
```

Для вычисления конкретных сумм (произведений) достаточно заменить `fk` конкретным видом слагаемого (сомножителя), стоящего под знаком **Σ** (**П**).

Примеры.

Задание 3.15. Выполните вычисления по формулам.

Сумма	**Произведение**
Формулы	
$S=\sum_{k=1}^{n}\cos 0{,}25\pi k$	$P=\prod_{k=1}^{n}\sin 0{,}35\pi k$
Программы	

`S = 0;`
`for k =1:n`
`  S = S+cos(0.25*pi*k);`
`end`
	`P = 1;`
	`for k = 1:n`
	`  P = P*sin(0.35*pi*k);`
	`end`

В некоторых случаях математическая запись слагаемого (сомножителя) `fk` может быть более сложной, например, содержать промежуточные вычисления. Тогда добавление каждого слагаемого (сомножителя) к сумме (произведению) следует выполнять в два этапа:

а) промежуточные вычисления для `k`-го слагаемого:

`Sk = fk (fk` - формула, зависящая от `k)`

б) добавление `k`-о слагаемого с учетом значений промежуточных вычислений, например,

`S = S+f_Sk (f_Sk` – формула, зависящая от `Sk)`

Тогда правила вычисления суммы и произведения примут следующий вид:

Сумма	**Произведение**	*Пояснения*
`S = 0;`	`P = 1;`	Начальное значение
`for k = 1:n`	`for k = 1:n`	Цикл
`  Sk = fk;`	`  Pk = fk;`	Промежуточные вычисления
`  S =S+f_Sk;`	`  P =P*f_Pk;`	Рекуррентная формула
`end`	`end`	

Если формулы, описывающие промежуточные вычисления, необходимые для расчета очередного слагаемого, зависят от номера слагаемого, то они записываются внутри цикла перед рекуррентной формулой `S = S + …` (или `P = P*…`).

Примеры.

Задание 3.16. Выполните вычисления по формулам.

Сумма	**Произведение**
Формулы	
$S = \sum_{k=1}^{n} \cos x_k$, где $x_k = \frac{\pi}{4k}$	$P = \prod_{k=1}^{n} \sin x_k$, где $x_k = \frac{\pi}{3k}$
Программы	

`S = 0;`
`for k = 1:n`
`  xk = pi/(4*k);`
`  S = S+cos(xk);`
`end`
	`P = 1;`
	`for k = 1:n`
	`  xk = pi/(3*k);`
	`  P = P*sin(xk);`
	`end`

В главе 5 приведены и подробно проанализированы другие примеры вычисления сумм и произведений.

3.7. Массивы

Массив – это совокупность однотипных данных, определенным образом поименованная и пронумерованная. Отдельные значения, хранящиеся в массиве, называются элементами массива. Все элементы в массиве имеют общее название и отличаются нумерацией.

Массивы, для нумерации элементов которых используется один индекс, называются одномерными. Массивы, для нумерации элементов которых используется два индекса, называются двумерными и т.д.

Матрица, состоящая из n строк и m столбцов, – пример ***двумерного массива***. Вектор в n-мерном пространства – пример ***одномерного массив***.

Одномерный массив можно рассматривать как частный случай двумерного, который имеет только одну строку (столбец) – размером 1×m или n×1. ***Скаляр*** также можно интерпретировать как двумерный массив, если считать его размер равным 1×1. В MatLab все переменные по умолчанию считаются массивами (матрицами), размерность которых автоматически определяется количеством и нумерацией помещенных в переменную значений. Поэтому ни одномерные массивы (векторы), ни двумерные массивы (матрицы) в MatLab не требуют какого-либо предварительного описания. Чтобы создать массив определенного размера, необходимо задать (ввести, присвоить, рассчитать) все его элементы.

Индексы элементов массива нумеруются по умолчанию начиная с единицы (1, 2, 3…).

Доступ к конкретному элементу массива (матрицы, вектора и т.д.) осуществляется путем указания значений индексов в круглых скобках после названия переменной.

Пример:

```
A(2) = 34.5;                X(1,3) = 26.1;
B(3) = pi/3;                C(5,2) = pi/5;
```

Наряду с конкретным значением (константой) в качестве индекса может использоваться и переменная. Значение переменной должно быть положительным, целочисленным и задано прежде, чем переменную используют в качестве индекса.

Пример:

```
p = 3; k = p-2;
A(p) = 34.5;                X(1,k) = 26.1;
B(p+2) = pi/(p+1);          C(k+1,p) = pi/5;
```

Если действия, которые надо провести с различными элементами массива, одинаковы для всех элементов массива, процесс обработки массива

можно автоматизировать, использовав оператор цикла для изменения значений индекса элементов массива. Номера элементов массивов являются счетными множествами, поэтому использование оператора `for`-цикла для работы с массивами является удобным и эффективным.

В некоторых случаях допускается обращение целиком к строке (столбцу) массива. Для этого вместо соответствующего индекса используют символ двоеточие (:).

Пример:

```
X(2,:)=10; % все элементы 2-й строки равны 10
C(:,k)=0;  % все элементы k-о столбца равны 0
```

Массивы могут быть использованы в программе на любом этапе обработки информации (ввод, обработка данных и вывод результатов).

3.7.1. Ввод массива с клавиатуры

Если ***вводимые данные являются массивом***, то ввод массива может быть организован как целиком, так и поэлементно. В обоих случаях используется оператор `input`. Для поэлементного ввода массива с помощью оператора `input` надо ввести отдельный элемент и с помощью `for`-цикла указать, для каких значений индексов следует повторять данный оператор.

Пример *поэлементного ввода* одномерного массива (строки) `A` длиной `n`:

```
for k = 1:n
    A(k)=input ('A(k) = ');
end
```

Пример *поэлементного ввода* одномерного массива (столбца) `A` длиной `n`:

```
for k = 1:n
    A(k,1)=input ('A(k) = ');
end
```

Пример *поэлементного ввода* двумерного массива `X` размером `n×m`:

```
for k = 1:n
for p = 1:m
    X(k,p) = input ('X(k,p) = ');
end
end
```

Если слева от оператора `input` указывается не отдельный элемент массива, а массив целиком, пользователь получает возможность ввести сразу несколько значений. При этом необходимо соблюдать определенные правила.

Пример *ввода всех элементов* одномерного массива A:

```
A = input ('Введите несколько значений: ');
```

В этом случае значения следует вводить в квадратных скобках. Если числа при вводе разделены пробелами или запятыми, введенный массив будет являться строкой. Если числа при вводе разделены точками с запятой (;), введенный массив будет являться столбцом.

Узнать количество введенных значений (длину одномерного массива) можно с помощью оператора `length`.

Пример (результат выполнения программного кода приведен на рис. 3.12):

```
A = input ('Введите несколько значений: ');
n = length(A)
```

Пример *ввода всех элементов* двумерного массива X:

```
X = input ('Введите элементы матрицы: ');
```

В этом случае значения следует вводить в квадратных скобках: значения, относящиеся к одной строке, – через пробел, столбцы отделяются точкой с запятой (;).

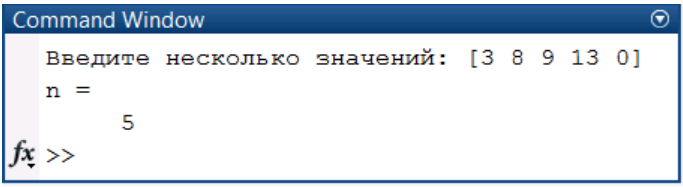


Рис. 3.12. Пример результата использования операторов `input` и `length` для ввода одномерного массива и определения его длины

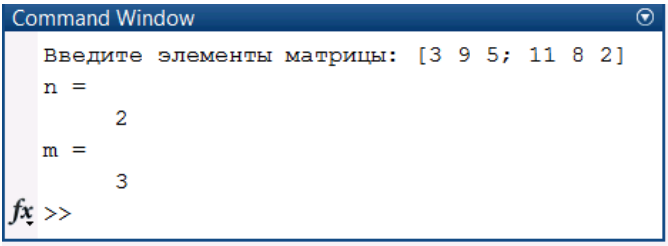


Рис. 3.13. Пример результата использования операторов `input` и `size` для ввода двумерного массива и определения его размеров

Узнать размер введенного массива (матрицы) можно с помощью оператора `size`, который определяет сразу два значения (первое – количество строк, второе – количество столбцов).

Пример (результат выполнения программного кода приведен на рис. 3.13):

```
X = input ('Введите элементы матрицы: ');
[n,m] = size(X)
```

В главе 5 приведены и подробно проанализированы другие примеры организации ввода массивов с клавиатуры.

3.7.2. Вывод значений, хранящихся в массиве

Если ***выводимые данные являются массивом***, но необходимо вывести *значение только какого-то одного элемента*, как и для скалярных переменных, можно использовать оператор `disp` или воспользоваться выводом по умолчанию. В том и в другом случае необходимо указать номера индексов элемента массива, значение которого необходимо вывести (например, `B(3)` или `C(2,4)`). В отличие от скалярных величин при использовании вывода по умолчанию значение отдельного элемента массива отображается в командном окне как значение вспомогательной переменной `ans`.



а *б*

Рис. 3.14. Пример сравнения результатов вывода элемента одномерного массива (*а*) и элемента двумерного массива (*б*) с помощью оператора `disp` и вывода по умолчанию

Примеры вывода значения одного элемента массива с помощью оператора `disp` и по умолчанию (результат выполнения программного кода приведен на рис. 3.14):

а) вывод элемента одномерного массива B с номером 3:

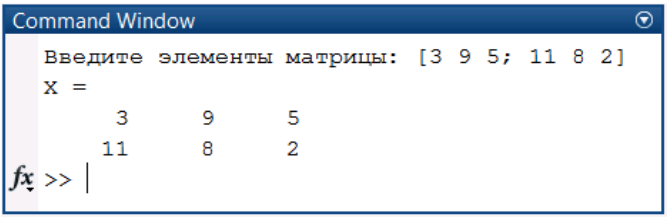
```
disp(['B(3)= ' num2str(B(3))]) % оператором disp
B(3)                  % по умолчанию
```

б) вывод элемента двумерного массива C, расположенного во второй строке и третьем столбце:

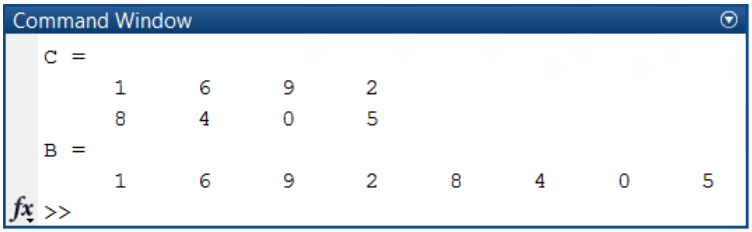
```
disp(['C(2,3)= ' num2str(C(2,3))]) % оператором disp
C(2,3)          % по умолчанию
```

Вывод всех элементов массива можно организовать с помощью циклов. Например, когда необходимо вывод каждого элемента сопровождать поясняющим текстом, содержащим сведения о номерах элементов массива, вывод одномерного массива-строки A длиной n можно оформить следующим образом:

```
for k = 1:n
   disp(['A(' int2str(k) ')= ' num2str(A(k))])
end
```



а



б

Рис. 3.15. Пример результатов вывода по умолчанию:
а – после выполнения оператора input;
б – при использовании имени переменной без каких-либо операций

Однако в большинстве случаев удобнее использовать возможности вывода по умолчанию. Если переменная является массивом, вывод по умолчанию осуществляется в матричном виде (данные, расположенные в одной строке массива, выводятся в одной и той же строке командного окна). Для инициации вывода по умолчанию (как и для скалярных переменных) необходимо либо указать имя переменной, содержащей массив, без каких-либо операций (в том числе без символа `;`), либо НЕ указывать символ точка с запятой (`;`) в конце любого оператора, выполнение которого приводит к присвоению значений одного или нескольких элементов массива. Например, если данные, помещенные в массив, вводились с клавиатуры, а после оператора `input` символ точка с запятой (`;`) не указан, то после ввода данных массив будет выведен на экран.

Примеры вывода по умолчанию массивов целиком (результаты выполнения программного кода приведены на рис. 3.15):

а)
```
X = input ('Введите элементы матрицы: ')
```

б)
```
C              % двумерный массив
B   % одномерный массив (строка)
```

Результат использования оператора `disp` для вывода массивов почти не отличается от результата использования вывода по умолчанию. И в том и в другом случае значения элементов выводятся в командное окно в том же порядке, как они расположены в массиве (по строкам и столбцам). Но в случае использования оператора `disp` название выводимой переменной в командном окне не отображается.

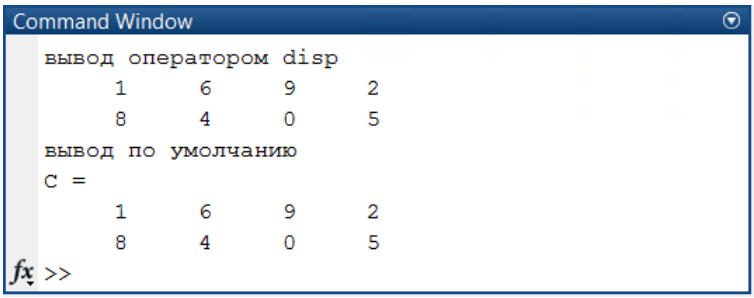


Рис. 3.16. Пример сравнения результатов вывода двумерного массива с помощью оператора `disp` и вывода по умолчанию

Для сравнения результатов вывода массивов с помощью оператора disp и по умолчанию можно использовать следующий программный код, результаты выполнения которого для двумерного массива приведены на рис. 3.16:

```
disp('вывод оператором disp')
disp(C)
disp('вывод по умолчанию')
C
```

Для вывода значений элементов вещественного массива с количеством знаков после запятой, отличных от количества, установленного по умолчанию, удобно использовать функцию num2str с указанием спецификации формата преобразования чисел. В результате применения функции num2str к числовому массиву формируется массив строк, каждая из которых последовательно объединяет полученные по правилам, описанным в разделе 3.2, строковые образы чисел, расположенных в одной строке исходного числового массива. Расстояние между строковыми представлениями (образами) чисел в одной строке определяются спецификацией формата преобразования чисел. Например, при использовании спецификации '%7.2f' на строковое представление каждого числа выделяется по 7 позиций, включая запятую и два знака после нее. Не занятые числом позиции заполняются пробелами. Таким образом, спецификация функции num2str позволяет программисту не только выбирать удобный формат представления чисел, но и регулировать их расположение при выводе числовой информации в командное окно.

Пример.

Задание 3.17. Элементы одномерного массива-строки B выведите четырьмя способами: 1) с помощью вывода по умолчанию; 2) с помощью оператора disp в числовом формате; 3) с помощью оператора disp в строковом формате по умолчанию; 4) с помощью оператора disp в десятичном формате с фиксированной запятой и двумя знаками после нее.

Программа (результат выполнения программного кода приведен на рис. 3.17).

```
disp('Способ 1:') % вывод по умолчанию
B
disp('Способ 2:')
% вывод оператором disp в числовом формате
```

```
disp(B)
disp('Способ 3:')
% вывод в строковом виде в формате по умолчанию
disp(num2str(B))
disp('Способ 4:')
% вывод в строковом виде в заданном формате
disp(num2str(B, '%7.2f'))
```

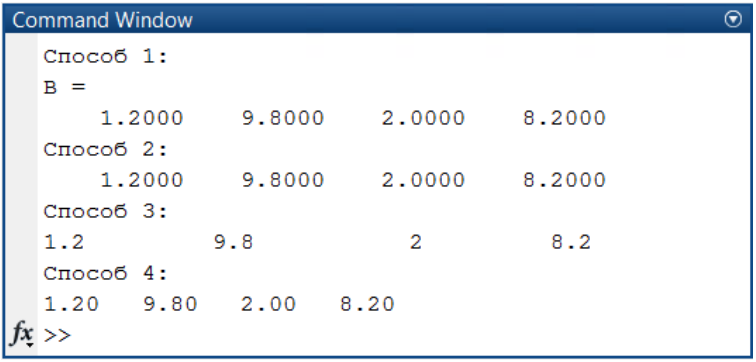


Рис. 3.17. Пример сравнения различных способов вывода одномерного массива-строки

Применение операции транспонирования (преобразование строк в столбцы) позволяет одномерный массив-строку преобразовать в одномерный массив-столбец и обратно. Стандартная функция транспонирования записывается в виде '. Если транспонирование массива нужно только для организации вывода, тогда нет необходимости сохранять результат транспонирования в отдельной переменной, но следует позаботиться о выводе на экран поясняющего текста с помощью оператора `disp`.

Пример вывода одномерного массива строки в виде столбца (результат выполнения программного кода приведен на рис. 3.18):

```
% вывод по умолчанию (для сравнения)
B
disp(' Массив B:') % вывод пояснения
disp(B')    % вывод результата транспонирования
```

Пояснения. Так как для сохранения результата транспонирования никакая переменная не указана, это результат автоматически присваивается вспомогательной переменной `ans`. Однако благодаря использованию

оператора `disp` для вывода полученного результата название переменной `ans` в командном окне не отображается.

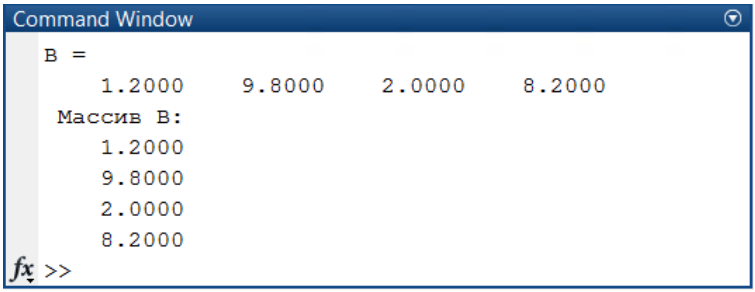


Рис. 3.18. Пример результата использования стандартной функции транспонирования для вывода одномерного массива-строки в виде столбца

В главе 5 приведены и подробно проанализированы другие примеры организации вывода массивов.

3.7.3. Обсчет данных, расположенных в массивах

Если массивы используются при обработке данных, они могут участвовать в любых операторах наравне со скалярными переменными в зависимости от содержания решаемой задачи. Кроме операций, которые применяются к массивам поэлементно, в MatLab существует большое количество стандартных операций, которые применяются к одномерным и двумерным массивам целиком.

Задачи по обработке данных с использованием массивов принято делить на два класса: обсчет данных и сортировка данных. Обсчет данных проводят обычно по заранее известным формулам, применяемым для каждого элемента массива. Наиболее распространенными задачами обсчета являются: заполнение массива, нахождение среднего значения и среднеквадратичного отклонения элементов массива. К наиболее распространенным задачам сортировки относятся задачи упорядочения данных по возрастанию, убыванию, поиск минимального и максимального значений, поиск всех значений больше (меньше) заданного и т.д.

Наиболее распространенными задачами ***обсчета данных, расположенных в массивах***, являются: заполнение массива с помощью простых математических формул или случайными числами; линейные преобразования массивов; арифметические действия с массивами (сложение, вычитание и т.д.); нахождение сумм и произведений элементов массива. Алгоритмы решения аналогичных задач для скалярных переменных рассматривались ранее. При решении задач с массивами рассмотренные ранее алгоритмы не изменятся, но их необходимо повторять (с помощью `for`-цикла) для каждого элемента массива.

Рассмотрим использование массивов в действиях с оператором присвоения (=) на нескольких примерах. Массивы могут использоваться слева от оператора присвоения (например, заполнение массива), справа от оператора присвоения (нахождения суммы элементов массива) и с обеих сторон от оператора присвоения (линейное преобразование массива).

Примеры использования массива слева от оператора присвоения.

Задание 3.18. Элементам одномерного массива (строки) `Y` присвойте следующие значения: 1, 6, 9, 2, 8, 4, 0, 5; полученный массив выведите на экран.

Программа.

```
Y=[1 6 9 2 8 4 0 5]
```

Пояснения. В конце оператора присвоения (=) нет точки с запятой (`;`), поэтому массив `Y` будет автоматически выведен на экран.

Задание 3.19. Элементам двумерного массива `C` присвойте следующие значения: первая строка: 1, 6, 9, 2; вторая строка: 8, 4, 0, 5; третья строка: 9, 9, 1, 1 (без вывода на экран).

Программа.

```
C=[1 6 9 2; 8 4 0 5; 9 9 1 1];
```

Пояснения. Значения, относящиеся к разным строкам, отделяются точкой с запятой (`;` – признак конца строки в двумерном массиве). В конце оператора присвоения (=) стоит точка с запятой (`;`), поэтому массив `C` на экран выводиться не будет.

Задание 3.20. Одномерный массив (строку) `Y` заполните значениями функции $f = \cos 0{,}3\pi x$, вычисленными на интервале $[a, b]$ с шагом dx; полученный массив выведите на экран.

Программа	*Пояснения*
`k = 1;`	Начальное значение индекса `k` массива `Y`
`for x = a:dx:b`	Изменение аргумента функции в пределах от `a` до `b` с шагом `dx`
`  Y(k) = cos(0.3*pi*x);`	Заполнение элемента массива `Y` с индексом `k`
`  k = k+1;`	Изменение значения индекса `k` массива `Y` (для следующего присвоения)
`end`	Конец цикла
`disp('Массив Y:')` `disp(Y)`	Вывод массива на экран с поясняющим текстом

Задание 3.21. *n* элементов одномерного массива (столбца) `Y` заполните значениями функции $f = \cos 0{,}3\pi x$, аргументы которой равномерно расположены на интервале $[a, b]$; полученный массив выведите на экран.

Программа	*Пояснения*
`dx = (b-a)/(n-1);`	Шаг изменения аргумента функции
`for k = 1:n`	Изменение индекса `k` массива `Y`
`  x = a+(k-1)*dx;`	Значение аргумента функции
`  Y(k,1)=cos(0.3*pi*x);`	Заполнение элемента массива `Y` с индексом `k`
`end`	Конец цикла
`disp('Массив Y:')` `disp(Y)`	Вывод массива на экран с поясняющим текстом

Задание 3.22. *n* элементов одномерного массива (строки) `X` заполните значениями, равномерно расположенными на интервале $[a, b]$; полученный массив выведите на экран как столбец.

Программа	*Пояснения*
`dx = (b-a)/(n-1);`	Расстояние между соседними значениями
`for k = 1:n`	Изменение индекса `k` массива `X`
`  X(k) = a+(k-1)*dx;`	Заполнение элемента массива `X` с индексом `k`
`end`	Конец цикла
`disp('Массив X:')` `disp(X')`	Вывод на экран транспонированного массива

В MatLab есть ***стандартная функция*** `linspace`, которая реализует данный алгоритм. Результат, эквивалентный предыдущему программному коду (без вывода массива), можно получить, применяя эту функцию:

```
X = linspace(a,b,n);
```

Пример использования массива справа от оператора присвоения.

Задание 3.23. Вычислите сумму элементов одномерного массива (строки) `A` длиной `n`: $S = \sum_{k=1}^{n} A_k$.

Программа	*Пояснения*
`S = 0;`	Начальное значение суммы
`for k =1: n`	Изменение значения индекса `k` массива `A`
`S = S+A(k);`	Сложение элементов массива `A` (рекуррентная формула)
`end`	

В MatLab есть ***стандартные функции*** для вычисления суммы элементов массива `sum` и для вычисления произведения элементов массива `prod`. Результат, эквивалентный предыдущему программному коду, можно получить, применяя функцию `sum`:

```
S = sum(A);
```

Примеры использования массивов с обеих сторон от оператора присвоения.

Задание 3.24. Из элементов одномерного массива `X` длиной `n` получите элементы одномерного массива `H` путем поэлементного линейного преобразования $H_p = K \cdot X_p + L$ (`K` и `L` – коэффициенты линейного преобразования).

Программа	*Пояснения*
`for p = 1: n`	Изменение значения индекса `p` массивов `X` и `H`
`H(p) = K*X(p)+L;`	Линейное преобразование для каждого элемента массива `X`
`end`	

Эквивалентный программный код:

```
H = K*X+L;
```

Задание 3.25. Из элементов двумерного массива S размером n×m получите элементы двумерного массива T путем поэлементного линейного преобразования $T_{p\,h} = K \cdot S_{p\,h} + L$ (K и L – коэффициенты линейного преобразования).

Программа	*Пояснения*
`for p =1: n` `for h =1: m`	Изменение значений индексов p, h массивов S и T
`T(p,h) = K*S(p,h)+L;`	Линейное преобразование для каждого элемента массива S
`end` `end`	

Эквивалентный программный код:

```
T = K*S+L;
```

Подробный анализ этих и других примеров обработки данных, расположенных в массивах, приведен в главе 5.

3.7.4. Простейшие задачи поиска элементов в массиве

Одна из наиболее распространенных задач сортировки – поиск минимального (имеющего наименьшее значение) элемента массива. Алгоритм решения этой задачи построен на сравнении элементов массива между собой.

Прежде чем обсуждать алгоритм сравнения элементов массива между собой, рассмотрим более простую задачу – сравнение элементов массива с некоторым заранее заданным числом.

Нахождение элементов массива, меньших заранее заданного числа z

Пусть изучаемые данные расположены в одномерном массиве A длиной n. Чтобы найти и вывести на экран элементы массива A, значения которых меньше заранее заданного (например, введенного с клавиатуры) числа z, необходимо:

1) значение каждого элемента массива A(k) сравнить с числом z;
2) если значение элемента массива A(k) меньше числа z, вывести на экран значение элемента массива A(k);
3) действия 1 и 2 повторить для всех элементов массива A, т.е. для таких элементов массива A, индексы которых k = 1, 2, … n.

Для реализации данного алгоритма на MatLab используем операторы if (пункт 1), disp (пункт 2) и for-цикл (пункт 3):

```
for k = 1: n
   if A(k)<z
      disp (['A(' int2str(k) ') = ' num2str(A(k))])
   end
end
```

Решение задачи о нахождении элементов массива, больших заданного числа z, имеет тот же алгоритм, если знак < (меньше) заменить знаком > (больше).

Нахождение минимального элемента массива

Пусть изучаемые данные расположены в одномерном массиве A длиной n. В отличие от предыдущего случая элементы массива надо сравнивать не с одним и тем же числом, а между собой. То есть число, с которым проводят сравнение (обозначим его Amin), не может быть произвольным. Его значение должно быть взято из массива. Если в процессе поиска будет найден элемент массива A(k), значение которого меньше значения, помещенного в переменную Amin, значение переменной Amin следует заменить значением найденного элемент массива A(k). Таким образом, в процессе поиска значение переменной Amin, с которой сравнивают элементы массива, может меняться. После окончания поиска переменная Amin будет содержать значение наименьшего элемента массива A.

В результате алгоритм нахождения минимального элемента одномерного массива A длиной n будет иметь следующую структуру:

1) перед началом поиска зададим начальное значение переменной Amin, с которой далее будем проводить сравнение. В качестве начального значения этой переменной возьмем один из элементов исследуемого массива A. Для удобства обычно берут A(1) или A(n) (последний элемент);

2) значение каждого элемента массива A(k) сравним с переменной Amin;
3) если значение элемента массива A(k) меньше текущего значения переменной Amin, заменим значение переменной Amin значением элемента массива A(k);
4) пункты 2 и 3 следует повторить для всех элементов массива A, т.е. для таких элементов массива A, индексы которых k = 1, 2, … n.

Для реализации данного алгоритма на MatLab используем операторы присвоения = (пункты 1,3), условного перехода if (пункт 2) и for-цикл (пункт 4):

```
Amin= A(1);    % начальное значение
for k = 1: n  % цикл по индексам элементов массива
   if A(k)<Amin  % сравнение A(k) и Amin
      Amin = A(k);  % изменение значения Amin
   end
end
```

Решение задачи о нахождении максимального элемента массива имеет тот же алгоритм, если знак < (меньше) заменить знаком > (больше).

Если в задаче требуется определить и минимальный и максимальный элементы массива, изложенный выше алгоритм применяют дважды, но с разными знаками (<, >) и с разными переменными для сравнения (например, Amin и Amax).

В MatLab существуют ***стандартные функции*** min и max для определения наибольшего и наименьшего значения среди элементов одномерного массива:

```
Amin = min(A);
Amax = max(A);
```

3.7.5. Простейшие задачи по изменению расположения данных в массиве

Взаимная перестановка двух элементов массива

Взаимная перестановка двух элементов массива – это обмен данными между этими элементами. Например, если элементы массива A(k) и A(p) имели значения A(k)=5 и A(p)=9, то после их перестановки эти

элементы будут содержать следующие значения: A(k)=9 и A(p)=5 (при неизменных значениях индексов k и p).

Чтобы элементы массива обменялись данными, необходимо иметь дополнительную переменную, например C, в которую на временное хранение помещается значение одного из переставляемых элементов массива (например, A(k)→C). После этого значение из элемента массива A(p) необходимо поместить в элемент массива A(k): A(p)→A(k). В этот момент элементы массива примут одинаковое значение. Далее значение из вспомогательной переменной C помещают в элемент массива A(p): C→A(p).

Для реализации данного алгоритма на MatLab используем три оператора присвоения:

```
C = A(k);
A(k) = A(p);
A(p) = C;
```

Сортировка элементов массива по убыванию

Сортировка элементов массива по убыванию – это такая перестановка элементов массива между собой, после которой каждый последующий (элемент с большим индексом) элемент имеет значение меньше предыдущего (элемента с меньшим индексом).

Для сортировки одномерного массива разработано несколько алгоритмов.

Сортировка по убыванию методом пузырька

В этом методе сравнивают значения двух соседних элементов массива A(k) и A(k+1). Эти два элемента будут расположены по убыванию, если A(k+1)<A(k). Если элементы расположены НЕ по убыванию (т.е. A(k)<A(k+1)), их нужно переставить между собой A(k) ↔ A(k+1): C = A(k); A(k) = A(k+1); A(k+1) = C;. Выполнение этой операции для всех пар элементов в массиве (для k от 1 до n-1) приведет к тому, что самое маленькое значение окажется в последнем элементе массива A(n).

Для упорядочивания оставшейся части массива описанные выше действия необходимо повторить еще n-1 раз, преобразуя каждый раз часть массива, длина которой постепенно уменьшается.

Для реализации данного алгоритма на MatLab используем операторы присвоения =, условного перехода `if` и `for`-цикла:

```
for p = 1:n-1
for k = 1:n-p    % длина цикла по k уменьшается с
                 % возрастанием p
if A(k)<A(k+1)   % сравнение соседних элементов
                 % массива
    C = A(k);          % перестановка соседних
    A(k) = A(k+1);     % элементов массива
    A(k+1) = C;
end
end
end
```

Для сортировки элементов массива методом пузырька по возрастанию используют тот же алгоритм, в котором при сравнении двух соседних элементов знак < (меньше) необходимо заменить знаком > (больше).

В MatLab существует ***стандартная функция*** только для сортировки элементов массива по возрастанию:

```
B = sort(A);
```

Подробный анализ этих и других примеров изменения расположения и значений элементов в массивах приведен в главе 5.

3.8. Нестандартные функции

Большие программы, предназначенные для решения сложных задач, но не разделенные на отдельные структурные элементы, сложны для разработки, отладки и понимания. Сложные задачи можно разделить на ряд более легких задач, для решения каждой из которых можно написать соответствующий фрагмент программы. Компоновать такие фрагменты удобнее, если каждый из них оформлен отдельно и имеет свое название. В MatLab такие фрагменты программы оформляют как функции. Программы, использующие функции, имеют, как правило, иерархическую структуру. Такие программы легче для разработки и проще для понимания.

Все функции языка программирования MatLab подразделяют на две группы: ***встроенные (стандартные)*** и ***определяемые программистом (нестандартные)***. Первые входят в комплект поставки языка и вызываются для выполнения по строго фиксированному имени. Вторые разрабатываются и именуются программистом. Функции, определяемые программистом, являются базой структурного программирования. Рассмотрим правила их создания и использования.

3.8.1. m-функции

m-функция – часть программы, имеющая собственное имя, предназначенная для решения определенной задачи, текст которой хранится в отдельном файле. Название файла должно совпадать с именем функции.

Среди пунктов раскрывающегося списка кнопки `New` в редакторе m-файлов есть пункт `Function`, позволяющий создать новый файл для написания m-функции (рис. 3.19).

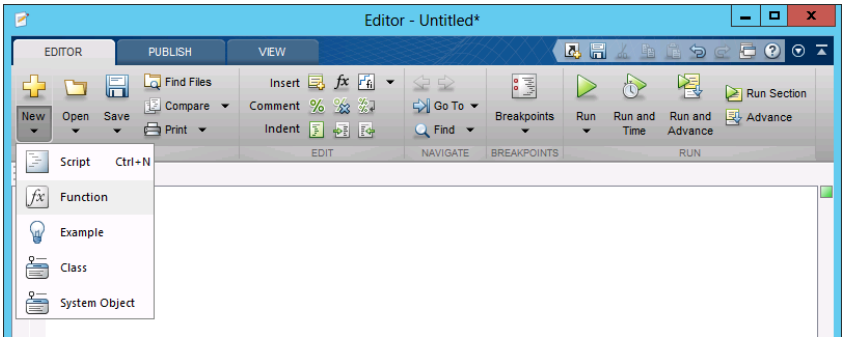


Рис. 3.19. Создание нового файла для написания m-функции в редакторе файлов MatLab

Выбор этого пункта на панели инструментов приводит к автоматическому созданию файла, содержащего заготовку структуры m-функции, типичную для MatLab.

Описание функции начинается со служебного слова `function`, после которого указывают имя функции между обозначениями результата ее работы и исходных данных.

Структура m-функции:

```
function [<результаты>]=<имя>(<исходные параметры>)
% Основной комментарий
% Дополнительный комментарий
  <операторы>
   <результаты> = <выражение>
   end
```

<Имя> функции – уникальный идентификатор, который в дальнейшем будет использоваться в программном коде управляющего файла

(m-сценария) или командного окна для обращения к функции. <Имя> функции должно совпадать с название m-файла, в котором она хранится.

<Исходные параметры> (или входные параметры) – это список переменных, содержащих исходные данные, необходимые для выполнения операций, записанных во фрагменте программы, обозначенном данной функцией. Если исходные данные должны быть представлены в нескольких переменных, они перечисляются в круглых скобках после имени функции через запятую в удобном с точки зрения программиста порядке. Если программный код, оформляемый в виде функции, не требует использования каких-либо исходных данных, то <исходные параметры> после имени функции указывать не нужно (функция без исходных параметров).

<Результаты> или <результирующие параметры> (выходные параметры) – это список переменных, которые после выполнения всех операций, запрограммированных в функции, будут содержать искомые значения. Если в результате работы функции необходимо получить значения нескольких переменных, они перечисляются в квадратных скобках перед именем функции через запятую в удобном с точки зрения программиста порядке. Если результирующая переменная одна, то квадратные скобки (операцию объединения) писать не нужно. Для всех переменных, перечисленных в списке <результирующие параметры>, необходимо указывать способ их вычисления (получения).

<Операторы> – набор операторов, позволяющих из исходных данных получить искомый результат. Последовательность операторов в тексте программного кода функции зависит от задачи, для решения которой она написана.

Текст функции должен заканчиваться таким количеством операторов присвоения, сколько <результирующих параметров> (переменных) описаны в заголовке функции.

Примеры.

Функция от двух исходных переменных с одним результирующим параметром:

```
   function r =  radius (x,y)
% вычисление расстояния от начала координат
% до точки с координатами (x, y)
   r = sqrt(x^2+y^2);
   end
```

Функция от одной исходной переменной с тремя результирующей параметрами:

```
   function [x,y,z] = koord (t)
% вычисление координат тела в момент времени t
   x = cos(5*t);
   y = sin(3*t+0.1);
   z = 0.2*t;
   end
```

Функция, не имеющая ни исходных, ни результирующих переменных (параметров):

```
   function Avtor
% выводит на экран данные автора
disp('Программа находит корни квадратного')
disp('           уравнения, ')
disp(' если ввести коэффициенты a, b, c')
disp('    автор Свиридов В.К. гр. 531')
   end
```

С помощью <исходных параметров> и <результатов> m-функция обменивается данными с другими m-файлами, которые ее используют (вызывают). Через <исходные параметры> m-функция получает исходные данные из внешних файлов или из командного окна. Через <результаты> – возвращает им вычисленные значения.

Если результатом операций, запрограммированных в функции, является значение одной переменной, то такие функции можно использовать (вызывать) в выражениях.

Для корректного использования (вызова) функций, результатом которых являются несколько переменных, их вызывают отдельно, присваивая такому количеству объединенных переменных, сколько результирующих параметров имеет функция.

Функции, не имеющие результирующих переменных, вызываются отдельно, как команды.

Примеры использования (вызова) функций:

```
% использование функции radius в выражении
   r1 = 3+radius(x1-x0,y1-y0)+radius(1,3)
% использование функции, вычисляющей координаты
% тела в разные моменты времени
   [x1,y1,z1] = koord(t1)
   [x2,y2,z2] = koord(t2)
   r=sqrt((x1-x2)^2+(y1-y2)^2+(z1-z2)^2)
% использование функции без исходных и
% результирующих параметров
   Avtor
```

Использование параметров (как исходных, так и результирующих) избавляет от необходимости отдельно программировать одинаковые действия для разных переменных. Все переменные (кроме параметров), встречающиеся в тексте программного кода функции, доступны для изменения и использования только в пределах данной m-функции и называются ***локальными***. Переменные, которые встречаются в управляющих программах (script-файлах), доступны для использования и изменения как в тексте программного кода данного файла, так и в командном окне. Поэтому, если текст программы не содержит оператора удаления переменных `clear`, повторное выполнение script-файла может использовать значения переменных, рассчитанные в предыдущем сеансе.

По умолчанию все переменные, встречающиеся в одном m-файле (функции, сценарии), кроме параметров функции, недоступны для использования в другом m-файле. Переменные, значения которых доступны для изменения и использования в разных m-файлах, называются ***глобальными***. Если использование значений переменных из управляющего файла, не являющихся параметрами функции, среди операторов m-функции является необходимым, такие переменные следует объявить глобальными и в тексте функции, и в управляющем файле (или командном окне) с помощью команды `global`.

Пример сравнения результатов выполнения (вычисления) функции, в которой исходные данные представлены как: исходные параметры; глобальные переменные.

Описание функции без глобальных переменных, но с исходными параметрами:

```
function r2 =  radius2 (x,y)
r2 = x^2+y^2;
end
```

Использование функции *в командном окне*:

```
x=3; y=2;
radius2(1,3)
ans =
   10
```

Описание функции без исходных параметров, но с глобальными переменными:

```
function r2 =  radius_2
global x y
r2 = x^2+y^2;
end
```

Использование функции *в командном окне*:

```
global x y
x=3; y=2;
radius_2
ans =
   13
```

Пояснения. Использование этих функций не в командном окне, а в управляющей программе происходит аналогичным образом. Только программный код и результат вычисления функции будут отображаться в разных окнах (в редакторе m-файлов и командном окне соответственно).

Изменение переменных внутри функции не приведет к изменению одноименных переменных в script-файле, если они являются локальными. Если переменные описаны как глобальные (`global`) и в script-файле (управляющей программе) и в функции, то изменение этих переменных в процессе выполнения программного кода функции приведет к их изменению и в управляющей программе.

3.8.2. Подфункции

Для программирования некоторых функций может потребоваться написание дополнительной (вспомогательной) функции. Если вспомогательная функция предназначена ТОЛЬКО для использования внутри основной функции, ее текст можно разместить в конце того же файла, где расположен текст основной функции. Такие (вспомогательные) функции недоступны для использования из других m-файлов и называются ***подфункциями***. В этом случае end ни в конце основной, ни в конце вспомогательной функции не пишется. В остальном текст каждой из функций в таком файле оформляется по общим правилам оформления m-функций, описанным в предыдущем параграфе. Основной считается первая из написанных в файле функций. Название файла должно совпадать с названием именно этой, основной функции.

Пример:

```
function r =  radius (t)
% основная функция
% вычисление расстояния от начала координат
% до тела в момент времени t
[x,y]=koord(t);
r = sqrt(x^2+y^2);
```

```
   function [x,y] = koord (t)
% подфункция
% вычисление координат тела в момент времени t
   x = cos(5*t);
   y = sin(3*t+0.1)+0.2*t;
```

Пояснения. В этом примере обе функции записаны в одном файле radius.m. Поэтому вызывать из управляющего m-файла или командного окна можно только функцию `radius(t)` по обычным правилам. Попытка вызвать функцию `koord(t)` в командном окне или любом файле, кроме radius.m, приведет к ошибке.

3.8.3. InLine-функции

Программировать функции, имеющие короткий программный код, можно без создания отдельных m-файлов, непосредственно в тексте `script`-файла в виде

```
<Имя> = inline ('<выражение>')
```

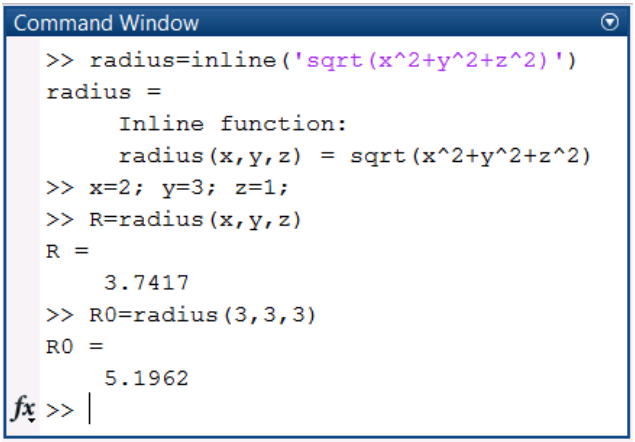


Рис. 3.20. Пример описания и использования InLine-функции в командном окне

<Имя> – уникальный идентификатор, по которому в дальнейшем будет выполняться вызов функции.

<Выражение> записывается в кавычках и задает правило вычисления функции. Количество исходных параметров определяется автоматически.

Для вызова InLine-функции необходимо указать ее имя и полный набор параметров (рис. 3.20).

InLine-функция доступна только из того m-файла (или командного окна), в котором она написана.

Глава 4. СТАНДАРТНЫЕ ФУНКЦИИ MATLAB

В комплект поставки среды MatLab входит огромное количество стандартных функций, позволяющих решать большинство типичных задач. Для использования стандартных функций не требуется никакого их предварительного подключения. Интерпретатор MatLab ищет файлы с используемыми в тексте программы функциями во всех зарегистрированных в среде MatLab папках.

4.1. Генераторы случайных чисел

Функция генерации псевдослучайных чисел `rand` (без параметров) позволяет получить ***случайное вещественное число, подчиняющееся равномерному распределению*** на интервале (0, 1). При использовании функции `rand` числа из разных участков интервала (0, 1) одинаковой длины появляются с одинаковой вероятностью.

Функция `rand` возвращает одно значение и может участвовать в выражениях, что позволяет путем линейных преобразований получать случайные числа в любом диапазоне.

Примеры генерации вещественного случайного числа в разных диапазонах.

Программа	*Пояснения*
`r1=rand`	Переменная `r1` примет вещественное значение в диапазоне от 0 до 1
`r2=3+2*rand`	Переменная `r2` примет вещественное значение в диапазоне от 3 до 5
`r3=a+(b-a)*rand`	Переменная `r3` примет вещественное значение в диапазоне от `a` до `b`

Та же функция, используемая с параметрами `rand(n,m)`, позволяет получить двумерный массив (матрицу) размером `n×m`, заполненную случайными числами, подчиняющимися равномерному распределению на интервале (0, 1). Функция с одним параметром `rand(n)` заполняет слу-

чайными числами от 0 до 1 квадратную матрицу размером n×n (а не одномерный массив).

Примеры заполнения массивов вещественными случайными числами в разных диапазонах.

Программа	*Пояснения*
`A1=rand(n,m)`	Заполнение двумерного массива `A1` размером `n×m` случайными вещественными числами в диапазоне от 0 до 1
`A2=a+(b-a)*rand(n,m)`	Заполнение двумерного массива `A2` размером `n×m` случайными вещественными числами в диапазоне от `a` до `b`
`A3=rand(n)`	Заполнение двумерного массива `A3` размером `n×n` случайными вещественными числами в диапазоне от 0 до 1
`A4=a+(b-a)*rand(n)`	Заполнение двумерного массива `A4` размером `n×n` случайными вещественными числами в диапазоне от `a` до `b`
`V1=rand(1,n)`	Заполнение одномерного массива (строки) `V1` длиной `n` случайными вещественными числами в диапазоне от 0 до 1
`V2=a+(b-a)*rand(1,n)`	Заполнение одномерного массива (строки) `V2` длиной `n` случайными вещественными числами в диапазоне от `a` до `b`
`V3=a+(b-a)*rand(n,1)`	Заполнение одномерного массива (столбца) `V3` длиной `n` случайными вещественными числами в диапазоне от `a` до `b`

Функция генерации псевдослучайных чисел `randi` позволяет получить ***случайное целое число***, подчиняющееся равномерному распределению. При использовании функции `randi` для получения одного случайного числа необходимо указать один параметр. Этот параметр позволяет управлять границами диапазона, в пределах которого осуществляется генерация.

Примеры генерации целого случайного числа в разных диапазонах.

Программа	*Пояснения*
`I1=randi(5)`	Переменная `I1` примет целое значение в диапазоне от 1 до 5
`I2=randi(n)`	Переменная `I2` примет целое значение в диапазоне от 1 до `n`
`I3=randi([3,13])`	Переменная `I3` примет целое значение в диапазоне от 3 до 13
`I4=randi([a,b])`	Переменная `I4` примет целое значение в диапазоне от `a` до `b`

Та же функция `randi`, используемая с дополнительными параметрами, позволяет получить двумерный массив (матрицу) размером `n×m`, заполненную целыми случайными числами. Размеры `n×m` матрицы (двумерного массива) указывают через запятую после границ диапазона случайных чисел.

Примеры заполнения массивов целыми случайными числами в разных диапазонах.

Программа	*Пояснения*
`B1=randi([a,b],n,m)`	Заполнение двумерного массива `B1` размером `n×m` случайными целыми числами в диапазоне от `a` до `b`
`B2=randi([a,b],n)`	Заполнение двумерного массива `B2` размером `n×n` случайными целыми числами в диапазоне от `a` до `b`
`V4=randi([a,b],1,n)`	Заполнение одномерного массива (строки) `V4` длиной `n` случайными целыми числами в диапазоне от `a` до `b`

Кроме функций `rand` и `randi`, генерирующих случайные числа, подчиняющиеся равномерному распределению, в MatLab есть стандартная функция `randn`, позволяющая генерировать ***случайные числа, подчиняющиеся нормальному распределению*** (распределению Гаусса). Нормальное распределение характеризуется математическим ожиданием и среднеквадратичным отклонением. Математическое ожидание – значение, вблизи которого случайные числа появляются с наибольшей вероят-

ностью. Среднеквадратичное отклонение характеризует средний разброс значений относительно математического ожидания. Стандартная функция `randn` (без параметров) генерирует числа с математическим ожиданием, равным нулю, среднеквадратичным отклонением, равным единице. То есть при ее многократном использовании чаще всего будут получаться числа, близкие к нулю (как положительные, так и отрицательные); 95% сгенерированных чисел будут лежать в интервале от –3 до 3.

Функция `randn` возвращает одно значение и может участвовать в выражениях.

Для получения случайных чисел вблизи другого значения математического ожидания и с другим среднеквадратичным разбросом используют линейное преобразование сгенерированных стандартной функцией `randn` значений.

Примеры генерации случайных чисел, подчиняющихся нормальному распределению в разных диапазонах:

Программа	*Пояснения*
`r1=randn`	Переменная `r1` примет вещественное значение вблизи нуля со среднеквадратичным отклонением, равным 1
`r2=a+b*randn`	Переменная `r2` примет вещественное значение вблизи `a` со среднеквадратичным отклонением, равным `b`
`A1=randn(n,m)`	Заполнение двумерного массива `A1` размером `n×m` случайными вещественными числами, расположенными вблизи нуля со среднеквадратичным отклонением, равным 1
`A2=a+b*randn(n,m)`	Заполнение двумерного массива `A2` размером `n×m` случайными вещественными числами, расположенными вблизи `a` со среднеквадратичным отклонением, равным `b`
`A3=a+b*randn(n)`	Заполнение двумерного массива `A3` размером `n×n` случайными вещественными числами, расположенными вблизи `a` со среднеквадратичным отклонением, равным `b`

Программа	*Пояснения*
`V1=randn(1,n)`	Заполнение одномерного массива (строки) `V1` длиной `n` случайными вещественными числами, расположенными вблизи нуля со среднеквадратичным отклонением, равным 1
`V2=a+b*randn(1,n)`	Заполнение одномерного массива (строки) `V2` длиной `n` случайными вещественными числами, расположенными вблизи `a` со среднеквадратичным отклонением, равным `b`

4.2. Операции с одномерными массивами

Все переменные в MatLab по умолчанию считаются двумерными массивами (матрицами). Поэтому наибольшее количество стандартных функций в MatLab связано с матричными операциями. Большинство из них можно применять и к одномерным массивам. Однако есть некоторые функции, применение которых к одномерным массивам имеет специфические особенности. Перечислим некоторые из них:

Функция	**Описание для одномерного массива `A`**
`prod(A)`	Произведение элементов массива
`sum(A)`	Сумма элементов массива
`mean(A)`	`mean(A)` = $\overline{A}$ – среднее арифметическое значение элементов
`std(A)` `std(A,flag)`	`std(A)` = `std(A,0)` = $=\sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(A_k-\overline{A})^2}$ – стандартное отклонение от среднего `std(A,1)` = $\sqrt{\frac{1}{n}\sum_{k=1}^{n}(A_k-\overline{A})^2}$ – среднеквадратичное отклонение от среднего
`linspace(a,b,n)`	Генерирует `n` значений, равномерно распределенных от `a` до `b`
`max(A)`	Возвращает наибольший элемент
`min(A)`	Возвращает наименьший элемент
`sort(A)`	Сортирует элементы по возрастанию

Применение функций `max`, `min`, `sort`, `prod`, `sum` к двумерным массивам (матрицам) приводит к выполнению этих же действий для каждого столбца матрицы. В результате получается строка соответствующих значений. Например, строка, каждый элемент которой – это сумма элементов, расположенных одном столбце матрицы.

4.3. Операции с матрицами

MatLab – система, ориентированная преимущественно на выполнение операций с матрицами. Она содержит большое количество стандартных функций для выполнения матричных расчетов. Стандартные операции с матрицами выполняются в соответствии с правилами линейной алгебры. Применение стандартных функций способствует уменьшению количества циклов, которые программист должен написать в программе явным образом для выполнения операций с матрицами. Перечислим некоторые из них:

Функция	**Описание для матрицы `A` (матриц `A` и `B`)**
`A+B`	Сложение матриц
`A-B`	Вычитание матриц
`A*B`	Умножение матриц
`A'`	Транспонирование матрицы
`A(:,k)=[]`	Удаление `k`-го столбца
`A(k,:)=[]`	Удаление `k`-й строки
`V=A(:,k)`	Создание одномерного массива `V` из `k`-го столбца
`V=A(k,:)`	Создание одномерного массива `V` из `k`-й строки
`A(:,k)=V`	Замена `k`-о столбца одномерным массивом `V`
`A(k,:)=V`	Замена `k`-й строки одномерным массивом `V`
`eye(n,m)`	Возвращает единичную матрицу размера `n`×`m`
`zeros(n,m)`	Возвращает матрицу размером `n`×`m`, состоящую из нулей
`ones(n,m)`	Возвращает матрицу размером `n`×`m`, состоящую из единиц
`rand(n,m)`	Возвращает матрицу размером `n`×`m`, заполненную вещественными случайными числами, подчиняющимися равномерному распределению на интервале (0, 1)

Функция	Описание для матрицы A (матриц A и B)
`randi([a,b],n,m)`	Возвращает матрицу размером n×m, заполненную целыми случайными числами, подчиняющимися равномерному распределению на интервале `[a,b]`
`randn(n,m)`	Возвращает матрицу размером n×m, заполненную случайными числами, подчиняющимися нормальному распределению с математическим ожиданием, равным нулю, и среднеквадратичным отклонением, равным единице
`[A,B]`	Горизонтальное объединение матриц A и B
`[A;B]`	Вертикальное объединение матриц A и B
`fliplr(A)`	Зеркально переставляет столбцы матрицы относительно вертикальной оси
`flipud(A)`	Зеркально переставляет строки матрицы относительно горизонтальной оси
`perms(V)`	Возвращает матрицу, которая содержит в себе все возможные перестановки элементов вектора V (каждая перестановка в отдельной строке)
`det(A)`	Вычисляет детерминант квадратной матрицы методом Гаусса
`rref(A)`	Приводит матрицу к треугольному виду методом Гаусса
`rrefmovie(A)`	Показывает пошаговое приведение матрицы к треугольному виду
`inv(A)`	Возвращает матрицу, обратную заданной

Глава 5. БАЗОВЫЕ КОНСТРУКЦИИ ПРОГРАММИРОВАНИЯ, РЕАЛИЗОВАННЫЕ НА ЯЗЫКЕ MATLAB

Описанные далее конструкции используются как в m-функциях, так и в управляющих программах. Описание переменных в MatLab происходит автоматически, зависит от значения и количества помещаемых в переменную данных и не требует написания специального кода в программе. Это не исключает необходимости преобразования данных из одного типа в другой. Поэтому пояснения к рассмотренным далее конструкциям включают описание типов данных, размещенных в используемых в этих примерах переменных. Выборочное комбинирование в программном коде рассмотренных далее конструкций позволяет эффективно разрабатывать программы для решения различных задач. При комбинировании данных конструкций в программном коде следует соблюдать последовательность этапов обработки информации: ввод (задание) начальных данных; обработка (получение, преобразование) данных; вывод результата.

5.1. Присвоение значений

Задание 5.1.1. Присвойте переменной `A` значение, равное 5,25.

Программа.

```
A = 5.25;
```

Пояснения. Слева от оператора присвоения (=) указывается идентификатор переменной, а справа – присваиваемое ей значение. Наличие в конце символа `;` (точка с запятой) подавляет вывод значений по умолчанию. Если этот символ (`;`) убрать, то присвоение значения переменной будет сопровождаться его выводом в командном окне.

Описание переменных. Тип переменных задается интерпретатором MatLab автоматически в соответствии с помещенным в переменную значением. Так как переменной `A` присваивается одно вещественное значение (`5,25`), то она автоматически описывается интерпретатором как числовой вещественный скаляр (`Double`).

Задание 5.1.2. Присвойте элементам одномерного массива X (строки) следующие значения: 1, 6, 9, 2, 8, 4, 0, 5.

Программа, вариант 1.

```
X(1) = 1; X(2) = 6; X(3) = 9; X(4) = 2;
X(5) = 8; X(6) = 4; X(7) = 0; X(8) = 5;
% Количество элементов
n = 8;
```

Программа, вариант 2.

```
      X = [1 6 9 2 8 4 0 5]
      disp ('Количество элементов')
      n = length (X)
```

Программа, вариант 3.

```
      X = [1, 6, 9, 2, 8, 4, 0, 5];
      % Количество элементов
      n = length (X);
```

Пояснения. В варианте 1 значения присваиваются (=) каждому конкретному элементу массива. Использование одного индекса для нумерации элементов массива свидетельствует о том, что значения присваиваются элементам одномерного массива-строки (номер строки по умолчанию считается равным единице). В вариантах 2 и 3 переменной X присваивается (=) совокупность данных, объединенных с помощью оператора `[]`. При этом данные объединяются в строку, если они разделены пробелами или запятыми. Количество n заполненных элементов массива, необходимое для организации циклов при дальнейшем использовании массива, можно либо присвоить заданному значению (вариант 1), либо определить с помощью стандартной функции `length`, в качестве параметра которой необходимо указать переменную, содержащую одномерный массив. Полученное с помощью функции `length` значение помещается (присваивается) в переменную n.

В варианте 2 заполненный массив X и количество n элементов в нем выводятся в командном окне с помощью вывода по умолчанию (символ `;` отсутствует у всех операторов). В остальных случаях присвоение значений происходит без вывода их значений в командном окне. Для этого в конце каждого оператора ставят символ `;`. В варианте 2 текст «Количество элементов» выводится в командное окно с помощью оператора `disp` для пояснения выводимой позже переменной n. В вариантах 1 и 3

тот же текст используется для пояснения фрагмента программного кода в виде комментария, так как значение переменной n присваивается без вывода в командном окне.

Описание переменных. Количество элементов массива – целое число, поэтому переменная n описывается как числовой целочисленный скаляр. Переменная X содержит несколько числовых значений, пронумерованных с помощью одного индекса, поэтому описывается как одномерный числовой (в данном примере целочисленный) массив (числовой массив размером 1×n).

Задание 5.1.3. Присвойте элементам одномерного массива X (столбца) следующие значения: 1, 6, 9, 2, 8, 4, 0, 5.

Программа, вариант 1.

```
X(1,1) = 1; X(2,1) = 6; X(3,1) = 9; X(4,1) = 2;
X(5,1) = 8; X(6,1) = 4; X(7,1) = 0; X(8,1) = 5;
% Количество элементов
n = 8;
```

Программа, вариант 2.

```
X = [1; 6; 9; 2; 8; 4; 0; 5]
disp ('Количество элементов')
n = length (X)
```

Программа, вариант 3.

```
X = [1; 6; 9; 2; 8; 4; 0; 5];
% Количество элементов
n = length (X);
```

Пояснения. В варианте 1 значения присваиваются (=) каждому конкретному элементу массива. Использование двух индексов для нумерации элементов массива так, что второй из них равен единице, свидетельствует о том, что значения присваиваются элементам одномерного массива-столбца (второй индекс – это номер столбца). В вариантах 2 и 3 переменной X присваивается (=) совокупность данных, объединенных с помощью оператора []. При этом данные объединяются в столбец, если они разделены символом точка с запятой (;). Количество n заполненных элементов массива, необходимое для организации циклов при дальнейшем использовании массива, можно либо присвоить заданному значению (вариант 1), либо определить с помощью стандартной функции length, в качестве параметра которой необходимо указать переменную, содер-

жащую одномерный массив. Полученное с помощью функции `length` значение помещается (присваивается) в переменную `n`.

В варианте 2 заполненный массив `X` и количество `n` элементов в нем выводятся в командном окне с помощью вывода по умолчанию (символ `;` отсутствует у всех операторов). В остальных случаях присвоение значений происходит без их вывода в командном окне. Для этого в конце каждого оператора ставится символ `;`. В варианте 2 текст «Количество элементов» выводится в командное окно с помощью оператора `disp` для пояснения выводимой позже переменной `n`. В вариантах 1 и 3 тот же текст используется для пояснения фрагмента программного кода в виде комментария, так как значение переменной `n` присваивается без вывода в командном окне.

Описание переменных. Количество элементов массива – целое число, поэтому переменная `n` описывается как числовой целочисленный скаляр. Переменная `X` содержит несколько числовых значений, пронумерованных с помощью двух индексов, один из которых принимает только фиксированное значение, равное единице, поэтому переменная `X` описывается как одномерный числовой (в данном примере целочисленный) массив (числовой массив размером `n×1`).

Задание 5.1.4. Присвойте элементам двумерного массива `A` следующие значения: первая строка: 1, 6, 9, 2; вторая строка: 8, 4, 0, 5; третья строка: 9, 9, 1, 1.

Программа, вариант 1.

```
A(1,1) = 1; A(1,2) = 6; A(1,3) = 9; A(1,4) = 2;
A(2,1) = 8; A(2,2) = 4; A(2,3) = 0; A(2,4) = 5;
A(3,1) = 9; A(3,2) = 9; A(3,3) = 1; A(3,4) = 1;
% n – количество строк, m – количество столбцов
n = 3; m = 4;
```

Программа, вариант 2.

```
A = [1 6 9 2; 8 4 0 5; 9 9 1 1]
disp ('Размер матрицы')
[n,m] = size (A)
```

Программа, вариант 3.

```
A = [1, 6, 9, 2; 8, 4, 0, 5; 9, 9, 1, 1];
% Размер матрицы
[n,m] = size (A);
```

Пояснения. В варианте 1 значения присваиваются (=) каждому конкретному элементу массива. Использование двух индексов для нумерации элементов массива свидетельствует о том, что значения присваиваются элементам двумерного массива размером 3×4 (первый индекс (номер строки) изменяется от 1 до 3, второй индекс (номер столбца) изменяется от 1 до 4). В вариантах 2 и 3 переменной `A` присваивается (=) совокупность данных, объединенных с помощью оператора `[]`. При этом данные объединяются в строку, если они разделены пробелами или запятыми. Данные, относящиеся к разным строкам, отделяются символом точка с запятой (`;`). Количество `n` строк и `m` столбцов в заполненном массиве, необходимые для организации циклов при дальнейшем использовании массива, можно либо присвоить заданным значениям (вариант 1), либо определить с помощью стандартной функции `size`, в качестве параметра которой необходимо указать переменную, содержащую двумерный массив. Полученные с помощью функции `size` значения помещаются (присваиваются) в переменные `n` и `m`.

В варианте 2 заполненный массив `A` и его размеры (`n` и `m`) выводятся в командном окне с помощью вывода по умолчанию (символ `;` отсутствует у всех операторов). В остальных случаях присвоение значений происходит без вывода их значений в командном окне. Для этого в конце каждого оператора ставится символ `;`. В варианте 2 текст «Размер матрицы» выводится в командное окно с помощью оператора `disp` для пояснения выводимых позже переменных `n` и `m`. В вариантах 1 и 3 текст «n – количество строк, m – количество столбцов» и «Размер матрицы» используется для пояснения фрагмента программного кода в виде комментария, так как значения переменных `n` и `m` присваиваются без вывода в командном окне.

Описание переменных. Количество строк и столбцов в массиве – целые числа, поэтому переменные `n` и `m` описывается как числовые целочисленные скаляры. Переменная `A` содержит несколько числовых значений, пронумерованных с помощью двух индексов, поэтому переменная `A` описывается как двумерный числовой (в данном примере целочисленный) массив (числовой массив размером `n×m`).

5.2. Ввод данных

Задание 5.2.1. Введите значение переменной A с клавиатуры.

Программа.

```
A = input ('Введите A : ');
```

Пояснения. Указанный в скобках оператора input параметр содержит поясняющий текст, какого рода информацию необходимо вводить. Введенная информация помещается (присваивается) в переменную, указанную слева от оператора присвоения (=). Если оператор input использовать без присвоения (=), то введенная информация автоматически присваивается специальной переменной ans.

Точка с запятой в конце оператора input блокирует вывод значения переменной A по умолчанию после окончания ввода данных.

Описание переменных. Данная конструкция используется для ввода информации любого типа. Поэтому переменная A после ввода данных может оказаться переменной любого типа, например, вещественной, целой (скаляром, числом) или массивом. Для конкретизации типа вводимой информации рекомендуется использовать в качестве параметра оператора input соответствующий поясняющий текст.

Задание 5.2.2. Введите значения переменных A и B с клавиатуры.

Программа.

```
A = input ('Введите A : ');
B = input ('Введите B : ');
```

Пояснения. Ввод реализуется последовательно сначала одной, затем другой переменной с помощью отдельных операторов input. Текст, указанный в качестве параметра оператора input, позволяет информировать пользователя, сколько значений необходимо ввести и каким переменным они будут присвоены. Содержание поясняющего текста не влияет на результат присвоения. Введенные значения помещаются в ту переменную, которая указана слева от оператора присвоения (=).

Точка с запятой в конце каждого из операторов input блокирует вывод значений переменных A и B по умолчанию после окончания ввода данных.

Описание переменных. A и B могут оказаться переменными любого типа, например, вещественными, целыми или массивами в зависимости

от типа введенной информации. Для конкретизации типа вводимой информации рекомендуется использовать в качестве параметра оператора `input` соответствующий поясняющий текст.

Задание 5.2.3. Введите значения переменных `a` и `b` так, чтобы было понятно, что `a` и `b` – границы интервала [*a*, *b*].

Программа, вариант 1.

```
a = input ('Введите начало интервала a : ');
b = input ('Введите конец  интервала b : ');
```

Программа, вариант 2.

```
disp ('Введите границы интервала [a, b]')
a = input ('a : ');
b = input ('b : ');
```

Пояснения. Описание смысла переменных в запросе (параметрах каждого из операторов `input`) перед вводом позволяет вводить данные более корректно. Например, если `a` и `b` – границы интервала, то следует вводить значение `b` больше, чем значение `a`. В обоих вариантах используется два оператора `input` для ввода каждой переменной поотдельности. В варианте 2 для вывода дополнительного пояснения о смысле, который будут иметь вводимые данные в программе, используется оператор `disp`. В этом случае можно использовать более краткие пояснения в операторах `input`.

Точка с запятой в конце каждого из операторов `input` блокирует вывод по умолчанию для значений переменных `a` и `b` после окончания ввода данных.

Описание переменных. По смыслу `a` и `b` – границы интервала, поэтому переменные `a` и `b` должны содержать вещественные значения и описываться вещественными скалярами.

Задание 5.2.4. Введите с клавиатуры `n` элементов одномерного массива `X` (строки).

Программа, вариант 1.

```
n = input ('Количество элементов массива n : ');
for k = 1:n
      X(k) = input (['X(' int2str(k) ') = ']);
end
```

Программа, вариант 2.

```
X = input ('Введите все элементы массива X : ');
disp ('Количество элементов')
n = length (X)
```

Пояснения. В варианте 1 значения элементов массива вводятся поэлементно: для ввода каждого элемента организован отдельный запрос с помощью отдельного оператора `input`. Для того чтобы поясняющий текст в операторе `input` содержал информацию о номере вводимого элемента, он разделен на три части: два строковых фрагмента `'X('` и `') = '`, а между ними – преобразованное в строковый формат значение номера `k` элемента массива. В качестве параметра оператора `input` можно использовать только одну строковую величину, поэтому три строковые фрагмента соединены в один с помощью оператора объединения `[]` (фрагменты в операторе объединения перечисляются через пробел). Повторение запросов о вводе элементов массива организовано с помощью оператора `for`-цикла по номерам `k` элементов массива от `1` до `n`. Количество `n` элементов массива, значения которых предстоит вводить, необходимо ввести с помощью отдельного оператора `input` перед циклом.

В варианте 2 с помощью оператора `input` организован один общий запрос на ввод нескольких значений, из которых по окончании ввода будет сформирован массив `X`. Чтобы из введенных данных был сформирован одномерный массив-строка, значения необходимо вводить в квадратных скобках через пробел (или через запятую). В этом случае пользователь может ввести любое количество значений, что приведет к формированию массива соответствующей длины. Чтобы в дальнейшем организовывать работу с этим массивом с помощью циклов, необходимо знать длину массива. Ее можно определить с помощью стандартной функции `length`, в качестве параметра которой необходимо указать идентификатор массива. Полученное данной функцией значение присваивается переменной `n`. Значение переменной `n` выводится в командном окне с помощью вывода по умолчанию, а поясняющий текст «Количество элементов» – с помощью оператора `disp`.

Точка с запятой в конце каждого из операторов `input` (в обоих вариантах) блокирует вывод значений элементов массива `X` по умолчанию после окончания ввода данных.

Описание переменных. По смыслу задачи переменные n и k содержат по одному целому значению: n – количество элементов; k – номер элемента в массиве. Переменные n и k – целочисленные скаляры. Переменная X содержит несколько значений – одномерный массив. В варианте 2 в зависимости от количества введенных пользователем значений и знаков между ними переменная X может оказаться не только одномерным массивом, но и скаляром или двумерным массивом. В варианте 1 вводимые пользователем данные помещаются именно в одномерный массив (т.е. в этом случае пользователь не может повлиять на способ размещения данных в массиве). Тип введенного массива зависит от типа (числового или строкового) введенной пользователем информации.

Задание 5.2.5. Введите с клавиатуры элементы двумерной матрицы A размером n×m.

Программа, вариант 1.

```
disp ('Введите матрицу A')
n = input ('Количество строк n : ');
m = input ('Количество столбцов m : ');
for k = 1:n
for p = 1:m
    A(k,p) = input (['A(' int2str(k) ',' ...
                          int2str(p) ') =']);
end
end
```

Программа, вариант 2.

```
A = input ('Введите все элементы матрицы A : ');
disp ('Размеры матрицы:')
[n, m] = size (A);
disp (['количество строк n = ' int2str(n)])
disp (['количество столбцов m = ' int2str(m)])
```

Пояснения. В варианте 1 значения элементов массива вводятся поэлементно: для ввода каждого элемента организован отдельный запрос с помощью отдельного оператора input. Для того чтобы поясняющий текст в операторе input содержал информацию о номере строки и номере столбца вводимого элемента, он разделен на пять части: три строковых фрагмента 'A(', ',' и ') = ', а между ними преобразованные в строковый формат значения номера строки k и номера столбца p элемен-

та массива. В качестве параметра оператора `input` можно использовать только одну строковую величину, поэтому пять строковых фрагментов соединены в один с помощью оператора объединения `[]` (фрагменты в операторе объединения перечисляются через пробел). Повторение запросов о вводе элементов массива организовано с помощью двух операторов `for`-цикла: по номерам `k` строк от `1` до `n`; по номерам `p` столбцов от `1` до `m`. Количество `n` строк и `m` столбцов элементов массива необходимо ввести с помощью отдельных операторов `input` перед циклами.

Для переноса слишком длинных операторов на следующую строку используется многоточие (`...`).

В варианте 2 с помощью оператора `input` организован один общий запрос на ввод нескольких значений, из которых по окончании ввода будет сформирован массив `A`. Чтобы из введенных данных был сформирован двумерный массив, значения необходимо вводить в квадратных скобках: значения, относящиеся к одной строке матрицы, вводят через пробел (или через запятую); строки отделяются друг от друга символом `;`. Пользователь может ввести любое количество значений, что приведет к формированию массива с соответствующим количеством строк и столбцов. Чтобы в дальнейшем организовывать работу с этим массивом с помощью циклов, необходимо знать размер матрицы (количество строк и столбцов). Их можно определить с помощью стандартной функции `size`, в качестве параметра которой необходимо указать идентификатор массива. Полученные данной функцией значения присваиваются переменным `n` и `m`. Значения переменных `n` и `m` выводятся в командном окне с помощью трех операторов `disp`. Один из них содержит общий заголовок «Размеры матрицы», а каждый из двух других – поясняющий текст и преобразованное в строковый формат значение переменной (`n` или `m`), объединенные в один текст оператором объединения `[]`.

Точка с запятой в конце каждого из операторов `input` (в обоих вариантах) блокирует вывод по умолчанию значений элементов массива `A` после окончания ввода данных.

Описание переменных. По смыслу задачи переменные `n` , `m`, `k` и `p` содержат по одному целому значению: `n`, `m` – количество строк, столбцов; `k`, `p` – номер строки, столбца. Переменные `n` , `m`, `k` и `p` – целочисленные скаляры. Переменная `A` содержит несколько значений – двумерный массив. В варианте 2 в зависимости от количества введенных пользователем значений и знаков между ними переменная `A` может оказаться не только двумерным массивом, но и скаляром или одномерным массивом. В вари-

анте 1 вводимые пользователем данные помещаются именно в двумерный массив (т.е. в этом случае пользователь не может повлиять на способ размещения данных в массиве). Будет ли этот массив `A` числовым или строковым, зависит от типа введенной пользователем информации.

5.3. Вывод данных

Задание 5.3.1. Выведите на экран значение переменной `x`.

Программа, вариант 1.

```
x
```

Программа, вариант 2.

```
disp (['x = ' num2str(x, '%6.3f')])
```

Пояснения. В программе, как правило, выводят значения нескольких переменных. Чтобы легче было ориентироваться, к какой переменной относится выведенное число, перед ним выводят поясняющий текст. В данном случае это `'x = '`. В варианте 1 используются возможности вывода по умолчанию. В этом случае вывод происходит в двух строках командного окна: в первой – название переменной, во второй – ее значение. В варианте 2 используется оператор `disp`. В этом случае вся информация выводится в одной строке командного окна. Параметр оператора `disp` может содержать только одну величину. Чтобы вывести две величины (строковое пояснение и значение переменной), их необходимо преобразовать к одинаковому типу (строковому) и объединить с помощью оператора `[]`. Объединяемые величины в операторе `[]` разделяются пробелами.

В варианте 1 вывод по умолчанию числовых данных осуществляется в формате, используемом в MatLab по умолчанию. Те же настройки используются и для преобразования чисел в строковый формат функцией `num2str`, если спецификация формата преобразования явно не указана. В варианте 2 сроковая спецификация формата `'%6.3f'` позволяет перед преобразованием в строку представить числовое значение в указанном формате (в данном примере в десятичном формате с фиксированной запятой и тремя знаками после нее).

Описание переменных. В варианте 1 `x` – переменная любого типа, например, целого, строкового или массив. В варианте 2 по использованию преобразования типов `num2str` можно сделать вывод, что пере-

менная x содержит числовые вещественные данные (например, числовой вещественный скаляр).

Задание 5.3.2. Выведите на экран значения вещественных переменных x и y в разных строках.

Программа, вариант 1.

```
x, y
```

Программа, вариант 2.

```
disp (['x = ' num2str(x, '%6.3f')])
disp (['y = ' num2str(y, '%6.3f')])
```

Пояснения. В варианте 1 используются возможности вывода по умолчанию. Несмотря на то, что в тексте программы переменные записаны в одной строке, вывод каждой их них происходит в двух строках командного окна: в первой – название переменной, во второй – ее значение. В варианте 2 для вывода каждой переменной используется оператор disp. В этом случае вся информация, являющаяся параметром оператора disp, выводится в одной строке командного окна. Параметр оператора disp может содержать только одну величину. Чтобы вывести две величины (строковое пояснение и значение переменной), их необходимо преобразовать к одинаковому типу (строковому) и объединить с помощью оператора []. Объединяемые величины в операторе [] разделяются пробелами. Аналогично выводится и значение второй переменной.

В варианте 1 вывод по умолчанию числовых данных осуществляется в формате, используемом в MatLab по умолчанию. Те же настройки используются и для преобразования чисел в строковый формат функцией num2str, если спецификация формата преобразования явно не указана. В варианте 2 сроковая спецификация формата '%6.3f' позволяет перед преобразованием в строку представить числовое значение в указанном формате (в данном примере в десятичном формате с фиксированной запятой и тремя знаками после нее).

Описание переменных. В варианте 1 x, y – переменные любого типа, например, целого, строкового или массивы. В варианте 2 по использованию преобразования типов num2str можно сделать вывод, что переменные x и y содержат числовые вещественные данные (например, числовые вещественные скаляры).

Задание 5.3.3. Выведите на экран значения вещественных переменных x и y в одной строке.

Программа, вариант 1.

```
disp (['x = ' num2str(x, '%6.2f') ...
     '  y = ' num2str(y, '%6.2f')])
```

Программа, вариант 2.

```
disp ('    x             y')
disp ([x y])
```

Пояснения. В обоих вариантах используется оператор disp, выведенные значения сопровождаются поясняющим текстом, к какой из переменных относится каждое из чисел. В варианте 1 поясняющий текст выводится перед соответствующим значением; в варианте 2 – над значением.

В варианте 1 одним оператором disp выводят два строковых пояснения и две числовые величины. Чтобы их можно было объединить в одном операторе [], необходимо числовые значения, хранящиеся в переменных, преобразовать в строковые с помощью функции num2str. Объединяемые оператором [] строковые величины отделяются пробелами. Благодаря использованию дополнительной спецификации '%6.2f' функции num2str, числовые значения выводятся в десятичном формате с фиксированной запятой и двумя знаками после нее.

Для переноса слишком длинных операторов на следующую строку используется многоточие (...).

В варианте 2 поясняющий текст выводится одним оператором disp, а значения переменных x и y – другим. Каждый из этих операторов disp выводит только однотипные данные (первый – только строковые, второй – только числовые), поэтому преобразование типов не требуется. Чтобы вывести значения двух однотипных переменных одним оператором disp, их необходимо объединить оператором []. Объединяемые оператором [] величины отделяются пробелами. Для отображения числовых данных используется формат, установленный в MatLab по умолчанию.

Описание переменных. По условию задачи переменные x и y содержат по одному числовому значению (числовые вещественные скаляры).

Задание 5.3.4. Выведите значения вещественных переменных y и z в разных строках так, чтобы было понятно, что они являются результатами некоторых вычислений.

Программа.

```
disp ('Результаты вычислений')
disp (['x = ' num2str(x, '%6.3f')])
disp (['y = ' num2str(y, '%6.3f')])
```

Пояснения. Для вывода поясняющего текста «Результаты вычислений» в отдельной строке командного окна используется оператор disp. Для вывода каждой переменной также используется оператор disp. Параметр оператора disp может содержать только одну величину (как при выводе пояснения). Чтобы вывести две величины (строковое пояснение и значение переменной), их необходимо преобразовать к одинаковому типу (строковому) и объединить с помощью оператора []. Объединяемые величины в операторе [] разделяются пробелами. Аналогично выводится и значение второй переменной.

Числовой формат, установленный в MatLab по умолчанию, используется для преобразования чисел в строковый формат функцией num2str, если спецификация формата преобразования явно не указана. Сроковая спецификация формата '%6.3f' позволяет перед преобразованием в строку представить числовое значение в указанном формате (в данном примере в десятичном формате с фиксированной запятой и тремя знаками после нее).

Описание переменных. По использованию преобразования типов num2str можно сделать вывод, что переменные x и y содержат числовые данные (числовые вещественные скаляры), что соответствует условию задачи.

Задание 5.3.5. Выведите значения n элементов одномерного массива (строки) X в строку.

Программа, вариант 1.

```
disp ('Массив')
X
```

Программа, вариант 2.

```
disp ('Массив')
disp (X)
```

Программа, вариант 3.

```
disp ('Массив')
disp (num2str(X, '%6.3f'))
```

Программа, вариант 4.

```
disp (['Массив  ' num2str(X, '%6.3f')])
```

Пояснения. В вариантах 1–3 в отдельной строке командного окна выводится поясняющий текст «Массив» с помощью оператора `disp`. В варианте 1 используется вывод по умолчанию – название переменной указано в отдельной строке без каких-либо команд и операторов. Так как массив является строкой, то и выводиться значения его элементов будут в строку. В этом случае между поясняющим текстом «Массив» и строкой цифр будет указано название переменной `X`. В варианте 2 массив-строка выводится с помощью оператора `disp` без поясняющего текста. Поэтому в отличие от варианта 1 в этом случае название переменной, содержащей массив, в командном окне не отображается. В вариантах 3 и 4 оператором `disp` выводятся значения элементов массива, предварительно преобразованные из чисел в строковый формат с учетом указанной спецификации `'%6.3f'` (в десятичном формате с фиксированной запятой и тремя знаками после нее). В этом случае удаляются дополнительные пробелы между числами (которые присутствуют в вариантах 1 и 2), на каждое число отводится по 6 позиций, включая пробелы. Варианты 3 и 4 отличаются расположением поясняющего текста. В варианте 4 поясняющий текст и строковое представление массива объединены с помощью оператора `[]` и выводятся в одной строке. А в варианте 3 поясняющий текст «Массив» выводится в отдельной строке.

В вариантах 1 и 2 числовые данные выводятся в формате, используемом в MatLab по умолчанию, а в вариантах 3 и 4 – в формате, указанном в спецификации функции `num2str` (десятичный формат с фиксированной запятой и тремя знаками после нее).

Описание переменных. В тех вариантах, где используется преобразование типов `num2str`, предполагается, что переменная `X` содержит числовые вещественные данные (числовой вещественный массив). Остальные варианты можно использовать для вывода массивов, содержащих информацию любого типа.

Задание 5.3.6. Выведите значения `n` элементов одномерного массива (строки) `X` в виде столбца.

Программа, вариант 1.

```
disp ('Массив')
for k = 1:n
    disp (['X(' int2str(k) ') =' ...
                num2str(X(k), '%6.3f')])
end
```

Программа, вариант 2.

```
      disp ('Массив')
      X'                    % ans
```

Программа, вариант 3.

```
      disp ('Массив')
      disp (X')             % без ans
```

Программа, вариант 4.

```
      disp ('Массив')
      disp (num2str(X', '%6.3f'))
```

Пояснения. Во всех вариантах в отдельной строке командного окна выводится поясняющий текст «Массив» с помощью оператора `disp`.

В варианте 1 массив выводится поэлементно (каждый элемент массива выводится отдельным оператором `disp` с поясняющим текстом и нумерацией; необходимое количество повторений действий оператора `disp` осуществляется с помощью `for`-цикла). Одним оператором `disp` выводят два строковых пояснения и две числовые величины. Чтобы их можно было объединить в одном операторе `[]`, необходимо числовое значение `X(k)` преобразовать в строковое с помощью функции `num2str`, а числовое значение `k` – с помощью функции `int2str`. Объединяемые оператором `[]` строковые величины отделяются пробелами.

Для переноса слишком длинных операторов на следующую строку используется многоточие (`...`).

В варианте 2 используется вывод по умолчанию. Но чтобы в командном окне массив выглядел не как строка, а как столбец, его перед выводом транспонируют (используют стандартную функцию транспонирования матриц `'`). Так как для хранения результата транспонирования не указана никакая переменная, он автоматически помещается в универсальную вспомогательную переменную `ans`. В этом случае между поясняющим текстом «Массив» и столбцом цифр будет указано название универсальной вспомогательной переменной `ans`.

В варианте 3 результат преобразования (транспонирования) строки в столбец выводится с помощью оператора `disp` без поясняющего текста.

Поэтому в отличие от варианта 2 в этом случае название переменной `ans`, содержащей транспонированный массив (между текстом «Массив» и числовыми значениями), в командном окне не отображается.

В варианте 4 оператором `disp` выводятся значения транспонированного массива, предварительно преобразованные из чисел в строковый формат с учетом указанной спецификации `'%6.3f'`. В результате числа в командном окне будут представлены в десятичном формате с фиксированной запятой и тремя знаками после нее независимо от числового формата, используемого в MatLab по умолчанию. В остальных вариантах числовые значения выводятся в формате, установленном в MatLab как формат по умолчанию.

Описание переменных. В тех вариантах, где используется преобразование типов `num2str`, предполагается, что переменная `X` содержит числовые вещественные данные (числовой вещественный массив). Остальные варианты можно использовать для вывода массивов, содержащих информацию любого типа. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массивах, т.е. это числовые целочисленные скаляры.

Задание 5.3.7. Выведите значения `n` элементов одномерного массива (строки) `X` по `p` элементов в каждой строке.

Программа, вариант 1.

```
disp ('Массив')
s = [];
for k = 1:n
   s = [s X(k)];
   if rem (k,p) == 0
      disp(s)
      s = [];
   end
end
disp(s)
```

Программа, вариант 2.

```
disp ('Массив')
for k = 1:p:(n-p)
     disp(X(k:k+p-1))
end
disp(X(k+p:n))
```

Пояснения. Оператор `disp` осуществляет вывод строки (целиком). Поэтому в варианте 1, прежде чем им воспользоваться, все относящиеся к одной строке данные объединяются во вспомогательной переменной `s` с помощью оператора объединения `[]`. Используется рекуррентная формула: текущее значение переменной `s` объединяется со значением элемента массива и вновь помещается в переменную `s` (значения накапливаются в переменной `s`). Для корректного выполнения рекуррентного объединения необходимо задать начальное значение переменной `s` (до цикла) как пустую совокупность. Если (`if`) номер `k` текущего элемента массива кратен `p` (остаток от деления `k` на `p` равен нулю), то накопленная в переменной `s` информация выводится с помощью оператора `disp` и эта переменная очищается с помощью оператора присвоения (=) пустой совокупности. Значения элементов массива выводятся без пояснения, под каким номером в массиве они располагаются.

Число элементов массива (`n`) может оказаться не кратным `p`, и последнее значение переменной `s` останется не выведенным. Поэтому после цикла добавлен оператор `disp(s)`, который выводит оставшуюся в переменной `s` информацию.

Когда вместо индекса элемента массива указывается последовательность, сформированная с помощью оператора `:`, то обращение происходит не к одному, а к совокупности элементов. Поэтому в варианте 2 с помощью оператора `disp` выводится совокупность, состоящая из `p` элементов (от `k` до `k+p-1`). Цикл, повторяющий вывод `p` элементов, организован по номерам `k` элементов, соответствующих началу выводимой в каждой отдельной строке группы элементов. Параметры цикла `k` изменяются с шагом `p` начиная с `1` так, чтобы номера выводимых элементов не выходили за границу массива (до `n-p`). После цикла добавлен оператор `disp`, который выводит оставшиеся элементы, если количество элементов массива не кратно `p` (используется последнее значение переменной `k`, вычисленное ранее в цикле).

В обоих вариантах с помощью оператора `disp` выводится текстовое пояснение «Массив», которое показывает, что выводимые далее значения являются массивом.

Числовые данные выводятся в формате, используемом в MatLab по умолчанию.

Описание переменных. Массив `X` может содержать данные любого типа. В варианте 1 переменная `s` является массивом переменной длины (от нуля перед циклом до `p`), содержащим данные того же типа, что

массив X. Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массивах, т.е. это числовые целочисленные скаляры. Переменная p – также целочисленный скаляр (по условию задачи).

Задание 5.3.8. Выведите значения n элементов двух одномерных массивов X и Y (строк) в два столбца (параллельно).

Программа, вариант 1.

```
for k = 1:n
  disp (['X(' int2str(k) ') = ' ...
              num2str(X(k), '%6.3f') ...
      '   Y(' int2str(k) ') = ' ...
              num2str(Y(k, '%6.3f'))])
end
```

Программа, вариант 2.

```
      disp ('     X         Y')
      for k = 1:n
         disp ([X(k) Y(k)])
      end
```

Программа, вариант 3.

```
      disp ('     k         X         Y')
      for k = 1:n
         disp ([k X(k) Y(k)])
      end
```

Программа, вариант 4.

```
      disp ('Массивы X и Y')
      [X; Y]'
```

Программа, вариант 5.

```
      disp ('Массивы X и Y')
      disp ([X; Y]')
```

Программа, вариант 6.

```
      disp ('Массивы X и Y')
      [[1:n]; X; Y]'
```

Программа, вариант 7.

```
      disp ('Массивы X и Y')
      disp ([[1:n]; X; Y]')
```

Пояснения. В одной строке командного окна должны располагаться по одному элементу из каждого массива, имеющие одинаковые индексы. В варианте 1 каждая пара значений `X(k)` и `Y(k)` выводится отдельным оператором `disp`, помещенным в `for`-цикл. Вывод значений `X(k)` и `Y(k)` в одной строке сопровождается поясняющим текстом, включающим номер элемента. Поэтому вся выводимая информация преобразована к строковому формату и объединена с помощью оператора `[]`. Использование символа `...` означает, что оператор (в данном случае `disp`) продолжается на следующей строке.

В варианте 2 каждая пара значений `X(k)` и `Y(k)` также выводится отдельным оператором `disp`, помещенным в `for`-цикл, но без поясняющего текста. Поэтому нет необходимости преобразовывать значения элементов массивов в строковый формат перед объединением. Однако если значения, расположенные в массивах, необходимо вывести в числовом формате, отличном от используемого в MatLab по умолчанию, совокупность `[X(k) Y(k)]` перед выводом можно преобразовать в строковый формат с использованием соответствующей спецификации (например, `'%6.3f'`). Тогда в цикле вместо оператора `disp([X(k) Y(k)])` следует написать `disp(num2str([X(k) Y(k)], '%6.3f'))`.

В этом и последующих вариантах в командном окне поясняющий текст будет располагаться над столбцами, поэтому он выведен с помощью отдельного оператора `disp` перед `for`-циклом.

В варианте 3 информация выводится в три столбца: перед столбцами со значениями элементов массивов `X(k)` и `Y(k)` добавлен столбец с номерами элементов. Вывод осуществляется также, как в варианте 2, с помощью оператора `disp` в цикле.

В вариантах 4 и 6 используется вывод по умолчанию. В варианте 4 перед выводом массивы `X` и `Y` необходимо объединить в матрицу, одна из строк которой будет состоять из элементов массива-строки `X`, а другая – из элементов массива-строки `Y` (символ `;` – признак конца строки при объединении строк в двумерный массив). В варианте 6 двумерный массив составлен из трех строк: первая – совокупность номеров элементов массивов; вторая – элементы массива `X`; третья – элементы массива `Y` (совокупность номеров элементов (от `1` до `n`) сформирована с помощью операторов двоеточия и объединения: `[1:n]`). После объединения полученную матрицу (двумерный массив) необходимо транспонировать (`'`), чтобы значения, расположенные в строках, выводились как столбцы. Так

как для хранения результата транспонирования никакая переменная не указана, он автоматически присваивается вспомогательной универсальной переменной `ans`. В конце данной операции символ `;` не указан, поэтому значение вспомогательной переменной `ans`, содержащей результат транспонирования, будет выведено в командном окне. Следовательно, в вариантах 4 и 6 между поясняющим текстом «Массивы X и Y» и столбцами значений будет выведено название переменной `ans`.

В вариантах 5 и 7 из имеющихся в массивах значений строится матрица так же, как в вариантах 4 и 6. Но для ее вывода используется оператор `disp`. Поэтому в вариантах 5 и 7 между поясняющим текстом «Массивы X и Y» и столбцами значений ничего выводиться не будет.

В вариантах 2–7 числовые данные выводятся в формате, используемом в MatLab по умолчанию. Те же настройки используются и для преобразования чисел в строковый формат функцией `num2str`, если спецификация формата преобразования явно не указана. В варианте 1 строковая спецификация формата `'%6.3f'` позволяет перед преобразованием в строку представить числовое значение в указанном формате (в данном примере в десятичном формате с фиксированной запятой и тремя знаками после нее).

Описание переменных. Только в варианте 1, где используется преобразование типов `num2str`, предполагается, что переменные `X` и `Y` содержат числовые данные (числовые вещественные массивы). Остальные варианты можно использовать для вывода массивов, содержащих информацию любого типа. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массивах, т.е. это числовые целочисленные скаляры.

Задание 5.3.9. Выведите матрицу `A` размером `n×m` на экран.

Программа, вариант 1.

```
disp ('Матрица A')
A
```

Программа, вариант 2.

```
disp ('Матрица A')
disp (A)
```

Программа, вариант 3.

```
disp ('Матрица A')
disp (num2str(A, '%6.2f'))
```

Программа, вариант 4.

```
disp ('Матрица A')
for k = 1:n
      disp (A(k,:))
end
```

Программа, вариант 5.

```
disp ('Матрица A')
for k = 1:n
   s = [];
   for p = 1:m
      s = [s A(k,p)];
   end
   disp(s)
end
```

Пояснения. Во всех вариантах перед матрицей с помощью оператора `disp` выведен поясняющий текст «Матрица A».

В варианте 1 используется вывод по умолчанию: в отдельной строке указан идентификатор двумерного массива `A` без точки с запятой и других операторов. В этом случае между поясняющим заголовком и значениями элементов будет выводиться название переменной, в которой они хранятся.

В варианте 2 двумерный массив `A` выводится с помощью оператора `disp`. В этом случае между поясняющим заголовком и значениями элементов никакой дополнительной информации не будет.

В варианте 3 также используется вывод с помощью оператора `disp`, но перед выводом двумерный массив преобразуется в строковый формат с помощью стандартной функции `num2str` с учетом спецификации `'%6.2f'`. В соответствии с выбранной спецификацией перед преобразованием в текст число будет представлено в десятичном формате с фиксированной запятой и двумя знаками после нее. Использование спецификации в функции `num2str` дает возможность (по сравнению с остальными вариантами) регулировать формат представления матрицы `A`, не изменяя формат, используемый по умолчанию, а также регулировать количество пробелов между числами в каждой из строк – в соответствии с выбранной спецификацией на каждое число выделяется по 6 позиций, включая пробелы.

В варианте 4 оператор `disp`, повторяемый в `for`-цикле, выводит по одной строке двумерного массива. Использование символа `:` вместо номера столбца говорит об обращении ко всем столбцам данной строки.

В варианте 5 из всех элементов, расположенных в одной строке, формируется совокупность (строка) `s` путем рекуррентного объединения элементов. Элементы, относящиеся к одной строке, расположены в разных столбцах, поэтому рекуррентное объединение осуществляется в цикле по столбцам (по `p` от `1` до `m`). Для корректного выполнения рекуррентных вычислений перед циклом по `p` задано начальное значение переменной `s`. После объединения в переменной `s` элементов, относящихся к одной строке, она выводится с помощью оператора `disp`. Эти же действия повторяются для всех строк с помощью оператора `for`-цикла (по `k` от `1` до `n`).

Если перед выводом числовых данных преобразование в строковый формат не выполняется (как в вариантах 1, 2, 4, 5) или при таком преобразовании спецификация формата в функции `num2str` не указывается, то числовые данные выводятся в формате, используемом в MatLab по умолчанию. В варианте 3 числовые данные (в соответствии с указанной в функции `num2str` спецификацией `'%6.2f'`) выводятся в десятичном формате с фиксированной запятой и двумя знаками после нее независимо от числового формата, используемого в MatLab по умолчанию.

Описание переменных. Только в варианте 3, где используется преобразование типов `num2str`, предполагается, что переменная `A` содержит числовые данные (числовой вещественный массив). Остальные варианты можно использовать для вывода массива, содержащего данные любого типа. Переменные `k`, `p`, `n` и `m` содержат информацию о номерах и количестве строк и столбцов (соответственно) в массивах, т.е. это числовые целочисленные скаляры.

5.4. Вычисления

Задание 5.4.1. Вычислите значение по формуле $f = x + 3\cos(x - 8)$.

Программа, вариант 1.

```
      f = x+3*cos(x-8);
```

Программа, вариант 2.

```
      f = func(x);
function f = func(x)     % в файле func.m
  f = x+3*cos(x-8);
end
```

Пояснения. Приоритетность операций MatLab такая же, как в математике: сначала вычисляются функции, далее умножение и деление, затем сложение и вычитание. Для изменения последовательности выполнения операций используют скобки.

В варианте 1 формула записана как математическое выражение справа от оператора присвоения. Если значение вычисляемого выражения необходимо поместить в некоторую переменную, то выражение располагается справа от знака присвоения (=), а переменная – слева.

В варианте 2 формула записана как m-функция в отдельном файле с названием `func.m`. В этом случае для вычисления по формуле необходимо вызвать функцию `func`, указав конкретное значение аргумента `x`. Чтобы результат вычисления m-функции поместить в некоторую переменную, переменную указывают слева от оператора присвоения (=), а m-функцию вызывают справа.

Точка с запятой в конце оператора присвоения (в обоих вариантах) подавляет применение вывода по умолчанию для значения переменной `f` как в функции, так и в управляющей программе.

Описание переменных. Учитывая, что результатом вычисления косинуса является вещественное число, переменная `f` являются числовым вещественным скаляром. Переменная `x` – числовой скаляр (вещественный или целочисленный).

Задание 5.4.2. Вычислите значения по несвязанным формулам $y = \sqrt{x^2 + 3}$ и $z = 6\cos x - 10$.

Программа, вариант 1.

```
y = sqrt(x*x+3);
z = 6*cos(x)-10;
```

Программа, вариант 2.

```
z = 6*cos(x)-10;
y = sqrt(x*x+3);
```

Программа, вариант 3.

```
[y,z] = f(x);
function [y,z] = f(x)    % в файле f.m
  y = sqrt(x*x+3);
  z = 6*cos(x)-10;
end
```

Пояснения. Имеющиеся формулы математически не связаны, поэтому порядок их написания в программе может быть любым (варианты 1 и 2). Точка с запятой в конце каждого из математических выражений подавляет применение вывода по умолчанию для значений переменных `y` и `z`. Последовательность выполнения математических операций в каждой формуле такая же, как в математике.

В вариантах 1 и 2 формулы записаны как математические выражения справа от оператора присвоения (=). А в варианте 3 – в виде m-функции, сохраненной в файле с названием `f.m`. В этом случае для вычисления по формулам необходимо вызвать функцию `f`, указав конкретное значение аргумента `x`. Результатом вычисления функции `f(x)` являются два числа `y` и `z`. Поэтому слева от оператора присвоения (=) указывают две объединенные оператором `[]` переменные, в которых необходимо сохранить полученный m-функцией результат, а справа – вызывают m-функцию.

Описание переменных. Учитывая, что результатом вычисления корня и косинуса являются вещественные числа, переменные `y` и `z` являются числовыми вещественными скалярами. Переменная `x` – числовой скаляр (вещественный или целочисленный).

Задание 5.4.3. Вычислите значения по взаимосвязанным формулам: $y = R_1 R_2 + f_1 - f_2$, где $R_1 = \dfrac{f_1}{f_2^2 + 4}$ и $R_2 = \sqrt{f_1^2 + f_2^2}$.

Программа, вариант 1.

```
      R1 = f1/(f2*f2+4);
      R2 = sqrt(f1*f1+f2*f2);
      y = R1*R2+f1-f2;
```

Программа, вариант 2.

```
      y = func(f1,f2)
function y = func(f1,f2)       % в файле func.m
  R1 = f1/(f2*f2+4);
  R2 = sqrt(f1*f1+f2*f2);
  y = R1*R2+f1-f2;
end
```

Пояснения. Имеющиеся формулы являются математически взаимосвязанными. Результаты вычислений значений `R1` и `R2` используются при вычислении значения `y`. Поэтому необходимо сначала рассчитать `R1`

и R2, а потом приступать к вычислению y. Точка с запятой в конце каждого из выражений подавляет применение вывода по умолчанию для значения переменных y, R1 и R2.

В варианте 1 все формулы записаны как математические выражения справа от операторов присвоения.

В варианте 2 все формулы размещены в m-функции в отдельном файле с названием func.m. В этом случае для вычисления искомого значения необходимо вызвать функцию func, указав конкретные значения ее аргументов f1 и f2. Чтобы результат вычисления m-функции поместить в некоторую переменную, переменную указывают слева от оператора присвоения (=), а m-функцию вызывают справа. Для управляющей программы, вызывающей функцию func(f1,f2), значения промежуточных переменных R1 и R2 недоступны.

Описание переменных. Учитывая, что результатом вычисления дроби и корня являются вещественные числа, переменные R1, R2 и y являются числовыми вещественными скалярами. Переменные f1 и f2 – числовые скаляры (вещественные или целочисленные).

Задание 5.4.4. Вычислите значение по формуле $f = \ln(a + b)$, если аргументы *a* и *b* удовлетворяют ее области определения.

Программа.

```
if a+b>0
      f = log(a+b);
end
```

Пояснения. Использование оператора if позволяет либо вычислить значение по имеющейся формуле (если аргумент логарифма неотрицательное число), либо пропустить данный фрагмент программы. Альтернативного варианта вычисления значения для случая, когда величина a+b отрицательная или равна нулю, в задаче не задано. Поэтому используется оператор if без блока else.

Описание переменных. Учитывая, что результатом вычисления натурального логарифма является вещественное число, переменная f является числовым вещественным скаляром. Переменные a и b – числовые скаляры (вещественные или целочисленные).

Задание 5.4.5. Вычислите значение по формуле $f = \sqrt{a+b} + \sqrt{b-7}$, если значения аргументов *a* и *b* позволяют получить вещественный результат. Выведите рассчитанное вещественное значение переменной *f*. Если заданным значениям аргументов *a* и *b* соответствует комплексное значение переменной *f*, выведите только соответствующее текстовое сообщение, а сами вычисления НЕ выполняйте.

Программа, вариант 1.

```
if (a+b>=0) & (b-7>=0)
   f = sqrt(a+b)+sqrt(b-7);
   disp (['f = ' num2str(f, '%6.3f')])
else
      disp ('Результат комплексный')
end
```

Программа, вариант 2.

```
if (a+b<0) | (b-7<0)
   disp ('Результат комплексный')
else
   f = sqrt(a+b)+sqrt(b-7);
   disp (['f = ' num2str(f, '%6.3f')])
end
```

Пояснения. В условии задачи описана альтернатива: либо вычислить значение и вывести результат вычислений на экран; либо вывести сообщение, предупреждающее, что вычисления не проводились. Поэтому в обоих вариантах используется оператор `if` с блоком `else`.

В варианте 1 проверяется «положительное» условие (когда вещественный результат получить возможно). Тогда в блоке `if` (после условия) выполняют два оператора: присвоения (=) с блокировкой вывода по умолчанию (`;`) и вывода (`disp` для вывода значения переменной `f`). Альтернативный случай (`else`) – вывод сообщения (один оператор `disp` с текстовым сообщением). Формула содержит два корня квадратных, и результат вычислений по ней не будет комплексным, только если оба подкоренных выражения окажутся неотрицательными. Поэтому в операторе `if` проверяется истинность двух логических выражений (`a+b>=0` и `b-7>=0`), объединенных логическим союзом И (`&`): условие считается истинным, если истинно и то и другое логическое выражение (одновременно).

В варианте 2 проверяется «отрицательное» условие (когда возможно получить только комплексный результат). Тогда в блоке `if` (после условия) выполняют один оператор – вывод сообщения (один оператор `disp` с текстовым сообщением). В альтернативном случае (`else`) – два оператора: присвоения (=) с блокировкой вывода по умолчанию (`;`) и вывода (`disp` для вывода значения переменной `f`). Формула содержит два корня квадратных, и результат вычислений по ней будет комплексным, если хотя бы одно из подкоренных выражений окажется отрицательным. Поэтому в операторе `if` проверяется истинность двух логических выражений (`a+b<0` и `b-7<0`), объединенных логическим союзом ИЛИ (`|`): условие считается истинным, если истинно либо одно логическое выражение, либо другое, либо оба одновременно.

Функция `num2str` со спецификацией `'%6.3f'` используется для вывода значения переменной `f` в десятичном формате с фиксированной запятой и тремя знаками после нее независимо от числового формата, используемого в MatLab по умолчанию.

Описание переменных. Учитывая, что результатом вычисления корня квадратного является вещественное число, переменная `f` является числовым вещественным скаляром. Переменные `a` и `b` – числовые скаляры (вещественные или целочисленные).

5.5. Заполнение массива

Задание 5.5.1. Заполните `n` элементов массива `Y` (строки) значениями, вычисленными по формуле $\cos 0{,}3\pi(k + 0{,}1)$, зависящей от номера `k` элемента массива.

Программа, вариант 1.

```
for k = 1:n
      Y(k) = cos(0.3*pi*(k+0.1));
end
```

Программа, вариант 2.

```
Y = cos(0.3*pi*([1:n]+0.1));
```

Программа, вариант 3.

```
for k = 1:n
      Y(k) = f(k);
end
```

Программа, вариант 4.

```
      Y = f([1:n]);

function y = f(k) % в файле f.m
%                 (для вариантов 3 и 4)
  y = cos(0.3*pi*(k+0.1));
end
```

Пояснения. По условию задачи каждый элемент массива необходимо вычислять по заданной формуле, выбирая разные значения `k` от `1` до `n`. Для этого в `for`-цикле по номерам `k` элементов каждому элементу массива `Y(k)` присваивается (=) значение, вычисленное по формуле (вариант 1).

В варианте 2 для заполнения массива значениями функции используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Аналогично вычисление большинства стандартных функций, таких как `cos`, `sin` и т.д., в MatLab также можно применять к аргументам, являющимся массивами (в результате получается массив той же длины, что и исходный, каждый элемент результирующего массива вычисляется как результат применения функции к элементу исходного массива с тем же индексом). Учитывая эти свойства стандартных функций MatLab, в варианте 2 с помощью оператора двоеточие (`:`) сформирована последовательность значений индексов элементов массива от `1` до `n`. С помощью оператора объединения `[]` из этой последовательности сформирован массив-строка, который последовательно преобразуется путем сложения и домножения. В результате аргументом косинуса будет являться одномерный массив строка длиной `n`, вычисление косинуса от которого позволяет получить искомый массив `Y`. В этом случае в явном использовании цикла нет необходимости.

В вариантах 3 и 4, в отличие от вариантов 1 и 2, формула записана в виде m-функции в файле с именем `f.m`. В качестве аргумента функции `f(k)` (также как для стандартных функций в вариантах 1 и 2) можно использовать как целое число `k`, так и массив целых чисел `[1:n]`, сформированный с помощью оператора двоеточие (`:`).

Во всех вариантах заполнение массива происходит без вывода значений его элементов (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Учитывая, что результатом вычисления косинуса является вещественное число, переменная `Y` является числовым вещественным массивом. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массивах, т.е. это числовые целочисленные скаляры.

Задание 5.5.2. Заполните n элементов массива `Y` (строки) случайными вещественными числами, подчиняющимися равномерному распределению на интервале [*a*, *b*].

Программа, вариант 1.

```
for k = 1:n
      Y(k)=a+(b-a)*rand;
end
```

Программа, вариант 2.

```
Y = a+(b-a)*rand(1,n);
```

Пояснения. Использование стандартной функции `rand` без параметров позволяет получить случайное вещественное число, подчиняющееся равномерному распределению на интервале `[0, 1]`. Полученное число можно преобразовать в число из интервала `[a,b]` с помощью линейного преобразования: умножение на константу `(b-a)` позволяет изменить длину интервала `[0, 1]` до размеров интервала `[a,b]`; прибавление константы `a` сдвигает значения, совмещая начало отсчета случайных чисел с началом заданного интервала.

В варианте 1 с помощью данного линейного преобразования заполняется каждый элемент массива в отдельности. Для этого значение результата линейного преобразования присваивается (=) элементу `Y(k)` массива в цикле по номерам `k` элементов от `1` до `n`.

В варианте 2 для заполнения массива значениями функции используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является масси-

вом, а другой скаляром. Учитывая эти свойства стандартных функций MatLab, в варианте 2 с помощью оператора `rand(1,n)` с параметрами сформирован одномерный массив-строка длиной `n`, значениями которого являются случайные вещественные числа, подчиняющиеся равномерному распределению в интервале `[0, 1]`. Полученный массив последовательно преобразуется путем домножения и сложения (линейное преобразование) и присваивается искомому массиву `Y`.

В обоих вариантах заполнение массива происходит без вывода значений его элементов (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Учитывая, что стандартная функция `rand` генерирует вещественное число, переменная `Y` является числовым вещественным массивом. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

По смыслу `a` и `b` – границы интервала. В общем случае они могут принимать как целые, так и дробные значения. Поэтому переменные `a` и `b` – это числовые скаляры (вещественные или целочисленные).

Задание 5.5.3. Заполните `n` элементов одномерного массива `X` (строки) значениями, равномерно расположенными на интервале [*a*, *b*].

Программа, вариант 1.

```
dx = (b-a)/(n-1);
for k=1:n
      X(k) = a+(k-1)*dx;
end
```

Программа, вариант 2.

```
dx = (b-a)/(n-1);
X(1) = a;
for k = 2:n
      X(k) = X(k-1)+dx;
end
```

Программа, вариант 3.

```
dx = (b-a)/(n-1);
X = a+([1:n]-1)*dx;    % X = [a:dx:b];
```

Программа, вариант 4.

```
X = linspace(a,b,n);
```

Пояснения. Согласно условию задачи на интервале [a,b] необходимо выбрать n разных значений, расстояния между которыми должны быть одинаковыми. Каждая пара соседних значений x образует отрезок. Длины всех отрезков, образованных парами соседних значений x, одинаковы (dx) и равны расстоянию между соседними значениями x. Чтобы рассчитать длину dx, необходимо длину отрезка (b–a) разделить на количество отрезков. Количество отрезков равно количеству пар соседних значений x: n–1.

Зная расстояние dx между соседними значениями x, можно последовательно получить все значения x двумя способами: либо последовательно добавляя dx к предыдущему значению x, начиная с a (вариант 2), либо записав взаимосвязь значения x с его номером k в явном виде: a+(k-1)*dx (вариант 1).

В вариантах 1 и 2 искомый массив X заполняется поэлементно. В варианте 1 каждому элементу X(k) массива присваивается результат вычисления по формуле, полученный в for-цикле по k от 1 до n. В варианте 2 значение первого элемента X(1) массива присваивается началу интервала a до цикла. В цикле по k от 2 до n значение каждого элемента X(k) вычисляется путем добавления к предыдущему элементу X(k-1) массива известного значения dx. Так как расстояние dx между всеми соседними значениями x одинаковы, то значение dx рассчитывается перед циклом.

В варианте 3 для заполнения массива значениями функции используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Учитывая эти свойства стандартных функций MatLab, в варианте 3 с помощью оператора двоеточие (:) сформирована последовательность значений индексов элементов массива от 1 до n. С помощью оператора объединения [] из этой последовательности сформирован массив-строка, который последовательно преобразуется путем сложения, вычитания и домножения. Результирующий массив целиком присваивается (=) переменной X. В этом случае в явном использовании цикла нет необходимости. Предварительное вычисление расстояния между искомыми значениями на интервале [a,b] позволяет также

сформировать совокупность значений от a до b с шагом dx с помощью оператора :. Для получения из этой совокупности одномерного массива-строки используется оператор объединения, результат действия которого целиком присваивается (=) переменной X. Этот способ решения задачи приводится в варианте 3 в виде комментария.

В варианте 4 для заполнения массива X используется стандартная функция linspace, которая формирует одномерный массив-строку из n значений, линейно (равномерно) изменяющихся от a до b.

Во всех вариантах массив X в командном окне не выводится: вывод по умолчанию подавляется с помощью символа ;, а оператор вывода disp не используется.

Описание переменных. Учитывая, что расстояние между точками на интервале [a,b] может оказаться дробным, переменная X является числовым вещественным массивом. Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

По смыслу a и b – границы интервала. В общем случае они могут принимать как целые, так и дробные значения. Поэтому переменные a и b, а также dx – это числовые вещественные скаляры.

Задание 5.5.4. Заполните n элементов одномерного массива Y (строки) значениями функции $f = 6\cos x + 3$, вычисленными на интервале [*a*, *b*] (аргументы *x* функции *f* располагаются на интервале [*a*, *b*] равномерно).

Программа, вариант 1.

```
dx = (b-a)/(n-1);
for k=1:n
      x = a+(k-1)*dx;
      Y(k) = 6*cos(x)+3;
end
```

Программа, вариант 2.

```
dx = (b-a)/(n-1);
x = a;
for k = 1:n
      Y(k) = 6*cos(x)+3;
      x = x+dx;
end
```

Программа, вариант 3.

```
dx = (b-a)/(n-1);
k = 1;
for x = a:dx:b
      Y(k) = 6*cos(x)+3;
      k = k+1;
end
```

Программа, вариант 4.

```
dx = (b-a)/(n-1);
Y = 6*cos([a:dx:b])+3;
```

Программа, вариант 5.

```
dx = (b-a)/(n-1);
for k=1:n
      x = a+(k-1)*dx;
      Y(k) = f(x);
end
```

Программа, вариант 6.

```
dx = (b-a)/(n-1);
x = a;
for k = 1:n
      Y(k) = f(x);
      x = x+dx;
end
```

Программа, вариант 7.

```
dx = (b-a)/(n-1);
k = 1;
for x = a:dx:b
      Y(k) = f(x);
      k = k+1;
end
```

Программа, вариант 8.

```
dx = (b-a)/(n-1);
Y = f([a:dx:b]);

function y = f(x) % в файле f.m
%              (для вариантов 5-8)
  y = 6*cos(x)+3;
end
```

Пояснения. Согласно условию задачи на интервале [a,b] необходимо выбрать n разных значений аргумента функции (x), расстояния между которыми должны быть одинаковыми. Каждая пара соседних значений x образует отрезок. Длины всех отрезков, образованных парами соседних значений x, одинаковы (dx) и равны расстоянию между соседними значениями x. Чтобы рассчитать длину dx, необходимо длину отрезка (b–a) разделить на количество отрезков. Количество отрезков равно количеству пар соседних значений x: n–1.

В данной задаче x – вспомогательная величина, вычисления которой могут быть организованы различными способами: можно последовательно добавлять dx к предыдущему значению x, начиная с x = a, с помощью рекуррентной формулы x = x+dx (вариант 2); можно записать взаимосвязь значения x с его номером k в явном виде x = a+(k–1)*dx (вариант 1); можно организовать for-цикл по значениям x от a до b с шагом dx (вариант 3).

Для каждого значения x вычисляют значения функции, а полученное значение помещают (присваивают) в элемент массива Y(k).

При любом варианте написания программы необходимо, чтобы перед вычислением значения функции было правильно задано значение аргумента x. Поэтому в варианте 1 формула для вычисления значения x расположена перед вычислением значения функции.

В варианте 2 для вычисления значения x используется рекуррентное соотношение (x = x+dx). В этом случае необходимо задавать начальное значение переменной x = a (до цикла, в котором выполняются рекуррентные вычисления). Так как начальное значение задано до цикла, то в цикле можно сразу вычислять значение функции, но далее необходимо подготовить x для следующего вычисления. Поэтому в варианте 2 вычисление значения x необходимо выполнять после вычисления значения функции.

В варианте 3 значения переменной x вычисляются автоматически с помощью оператора for-цикла. Поэтому значение функции (как и в варианте 2) можно сразу выполнять в начале цикла.

В вариантах 1–3 многократное повторение вычислений организовано с помощью for-цикла. Но в вариантах 1 и 2 цикл организован по номерам k элементов массива от 1 до n, а в варианте 3 – по значениям аргумента x функции. Учитывая, что значение аргумента функции и номер элемента являются взаимосвязанными (каждому элементу массива соответствует одно значение аргумента), нельзя изменять эти величины независимо. Поэтому

цикл организуется только по одной из этих величин (либо по k, либо по x). Вторая величина вычисляется. Если цикл организован по номерам элементов (варианты 1 и 2), аргумент функции вычисляется. Если цикл организован по аргументам x функции от a до b (вариант 3), то вычисляется номер k элемента массива, которому будет присвоено вычисленное значение функции. Для изменения номера элемента массива параллельно с аргументом x функции в варианте 3 используется рекуррентное соотношение k = k+1, которое увеличивает значение k на единицу после каждого вычисления значения функции так, чтобы при следующем выполнении цикла рассчитанное значение помещалось в следующий элемент массива. Начальное значение переменной k задается перед циклом так, чтобы первое вычисленное значение помещалось в элемент массива с номером 1.

В варианте 4 для заполнения массива значениями функции используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Аналогично вычисление большинства стандартных функций таких, как cos, sin и т.д., в MatLab также можно применять к аргументам, являющимся массивами (в результате получается массив той же длины, что и исходный, каждый элемент результирующего массива вычисляется как результат применения функции к элементу исходного массива с тем же индексом). Учитывая эти свойства стандартных функций MatLab, в варианте 4 с помощью оператора двоеточие (:) сформирована последовательность значений аргументов функции от a до b с шагом dx. С помощью оператора объединения [] из этой последовательности сформирован массив-строка, который используется в качестве аргумента косинуса. Последующие арифметические преобразования этого массива позволяют получить искомый массив Y. В этом случае в явном использовании цикла нет необходимости.

В вариантах 5–8, в отличие от вариантов 1–4, формула записана в виде m-функции в файле с именем f.m. В качестве аргумента функции f(x) (также как для стандартных функций в вариантах 1–4) можно использовать как вещественное число x (варианты 5–7), так и массив вещественных чисел [a:dx:b], сформированный с помощью операторов двоеточие и объединение (вариант 8).

Во всех вариантах значения элементов массива `Y`, а также вспомогательных переменных `x` и `dx` в командном окне не выводятся, так как в конце каждой операции присвоения используется символ `;`, подавляющий вывод по умолчанию, а оператор `disp` нигде не применяется.

Описание переменных. Учитывая, что результатом вычисления косинуса является вещественное число, переменная `Y` является числовым вещественным массивом. Расстояние между точками на интервале `[a,b]` может оказаться дробным, поэтому вспомогательная переменная `x` является числовым вещественным скаляром. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

По смыслу `a` и `b` – границы интервала. В общем случае они могут принимать как целые, так и дробные значения. Поэтому переменные `a` и `b`, а также `dx` – это числовые вещественные скаляры.

Задание 5.5.5. Заполните два одномерных массива `X` и `Y` (строки) одинаковой длины (по `n` элементов в каждом): массив `X` – значениями, равномерно расположенными на интервале [*a*, *b*]; массив `Y` – значениями функции $f = 6\cos x + 3$, в качестве аргументов которой взяты элементы массива `X` с тем же индексом.

Программа, вариант 1.

```
dx = (b-a)/(n-1);
for k=1:n
      X(k) = a+(k-1)*dx;
      Y(k) = 6*cos(X(k))+3;
end
```

Программа, вариант 2.

```
dx = (b-a)/(n-1);
X(1) = a;
for k = 1:n
      Y(k) = 6*cos(X(k))+3;
      X(k+1) = X(k)+dx;
end
```

Программа, вариант 3.

```
X = linspace(a,b,n);
for k = 1:n
      Y(k) = 6*cos(X(k))+3;
end
```

Программа, вариант 4.

```
X = linspace(a,b,n);
Y = 6*cos(X)+3;
```

Программа, вариант 5.

```
dx = (b-a)/(n-1);
X = [a:dx:b];
Y = 6*cos(X)+3;
```

Программа, вариант 6.

```
dx = (b-a)/(n-1);
for k=1:n
      X(k) = a+(k-1)*dx;
      Y(k) = f(X(k));
end
```

Программа, вариант 7.

```
dx = (b-a)/(n-1);
X(1) = a;
for k = 1:n
      Y(k) = f(X(k));
      X(k+1) = X(k)+dx;
end
```

Программа, вариант 8.

```
X = linspace(a,b,n);
for k = 1:n
      Y(k) = f(X(k));
end
```

Программа, вариант 9.

```
X = linspace(a,b,n);
Y = f(X);
```

Программа, вариант 10.

```
dx = (b-a)/(n-1);
X = [a:dx:b];
Y = f(X);
```

```
function y = f(x) % в файле f.m
%              (для вариантов 6-10)
  y = 6*cos(x)+3;
end
```

Пояснения. Согласно условию задачи на интервале [a,b] необходимо выбрать n разных значений аргумента функции, расстояния между которыми должны быть одинаковыми. Каждая пара соседних значений аргумента образует отрезок. Длины всех отрезков, образованных парами соседних значений аргумента, одинаковы (dx) и равны расстоянию между соседними значениями аргумента. Чтобы рассчитать длину dx, необходимо длину отрезка (b−a) разделить на количество отрезков. Количество отрезков равно количеству пар соседних значений аргумента: n−1.

В данной задаче все посчитанные значения аргумента необходимо поместить в массив X. Поэтому X – числовой массив, вычисления значений элементов которого могут быть организованы различными способами: можно последовательно добавлять dx к предыдущему значению X(k) начиная с X(1) = a с помощью рекуррентной формулы X(k+1) = X(k)+dx (вариант 2); можно записать взаимосвязь значения элемента X(k) массива с его номером k в явном виде X(k) = a+(k−1)*dx (вариант 1); можно заполнить массив X с помощью стандартной функции linspace (варианты 3 и 4) или сформировав последовательность значений с помощью оператора **:** (вариант 5).

Каждое значение элемента X(k) массива подставляют в функцию в качестве аргумента, а полученное значение помещают (присваивают) элементу массива Y(k) с тем же номером.

При любом варианте написания программы необходимо, чтобы перед вычислением значения функции было правильно задано значение аргумента. Поэтому в варианте 1 формула для вычисления значения X(k) расположена перед вычислением значения функции.

В варианте 2 для вычисления значения X(k+1) используется рекуррентное соотношение (X(k+1) = X(k)+dx). В этом случае необходимо задавать начальное значение переменной X(1) = a (до цикла, в котором выполняются рекуррентные вычисления). Так как начальное значение задано до цикла, то в цикле можно сразу вычислять значение функции, но далее необходимо подготовить X(k+1) для следующего вычисления. Поэтому в варианте 2 вычисление значения X(k+1) необходимо выполнять после вычисления значения функции.

В вариантах 3 и 4 для заполнения массива X используется стандартная функция linspace, которая формирует одномерный массив-строку из n

значений, линейно изменяющихся от a до b. Заполнение массива X происходит перед заполнением массива Y.

В варианте 5 с помощью оператора : формируется последовательность значений от a до b с шагом dx, которая объединяется в массив-строку с помощью оператора []. Полученный массив целиком присваивается (=) переменной X. Значение шага dx должно быть рассчитано перед формированием последовательности (заполнением массива X). А заполнение массива должно происходить перед заполнением массива Y.

В вариантах 1 и 2 оба массива заполняются поэлементно (в цикле по номерам k элементов массива от 1 до n). В варианте 3 поэлементно заполняется только массив Y, а массив X заполняется целиком с помощью стандартной функции.

В вариантах 4 и 5 массив X также заполняется целиком с помощью стандартной функции linspace, как в варианте 3, или с помощью операторов : и []. Для заполнения массива Y значениями функции используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Вычисление большинства стандартных функций таких, как cos, sin и т.д., в MatLab также можно применять к аргументам, являющимся массивами (в результате получается массив той же длины, что и исходный, каждый элемент результирующего массива вычисляется как результат применения функции к элементу исходного массива с тем же индексом). Учитывая эти свойства стандартных функций MatLab, в вариантах 4 и 5 в качестве аргумента косинуса используется массив X длиной n, вычисление косинуса от каждого элемента которого позволяет получить массив той же длины, а дальнейшие арифметические операции преобразуют его в соответствии с заданной функцией. Таким образом, в вариантах 4 и 5 искомый массив Y, как и массив X, заполняется целиком, но с использованием массива X.

В вариантах 6–10, в отличие от вариантов 1–5 формула записана в виде m-функции в файле с именем f.m. В качестве аргумента функции f(x) (также как для стандартных функций в вариантах 1–5) можно использовать как вещественное число X(k) (варианты 6–8), так и массив вещественных чисел X, сформированный с помощью стандартной функции linspace(a,b,n) (вариант 9) или оператора двоеточие [a:dx:b] (вариант 10).

Во всех вариантах значения элементов массивов X и Y, а также вспомогательной переменной dx в командном окне не выводятся, так как в

конце каждой операции присвоения используется символ ;, подавляющий вывод по умолчанию, а оператор disp нигде не применяется.

Описание переменных. Учитывая, что результатом вычисления косинуса является вещественное число, переменная Y является числовым вещественным массивом. Расстояние между точками на интервале [a,b] может оказаться дробным, поэтому переменная X также является числовым вещественным массивом той же длины. Переменные k и n содержат информацию о номерах и количестве (соответственно) элементов в массивах, т.е. это числовые целочисленные скаляры.

По смыслу a и b – границы интервала. В общем случае они могут принимать как целые, так и дробные значения. Поэтому переменные a и b, а также dx – это числовые вещественные скаляры.

Задание 5.5.6. Заполните n элементов массива Y (строки) значениями функции $f = 6\cos x + 3$, вычисленными на интервале $[a, b]$ так, что аргументы x функции f располагаются на интервале $[a, b]$ случайным образом.

Программа, вариант 1.

```
      for k = 1:n
            x = a+(b-a)*rand;
            Y(k) = 6*cos(x)+3;
      end
```

Программа, вариант 2.

```
            x = a+(b-a)*rand(1,n);
            Y = 6*cos(x)+3;
```

Программа, вариант 3.

```
      for k = 1:n
            x = a+(b-a)*rand;
            Y(k) = f(x);
      end
```

Программа, вариант 4.

```
            x = a+(b-a)*rand(1,n);
            Y = f(x);

function y = f(x) % в файле f.m
%              (для вариантов 3, 4)
  y = 6*cos(x)+3;
end
```

Пояснения. Согласно условию задачи на интервале `[a,b]` необходимо выбрать `n` значений аргумента функции `x` случайным образом.

Стандартная функция `rand` без параметров (вариант 1) позволяет получить случайное вещественное число, подчиняющееся равномерному распределению на интервале `[0, 1]`. Путем линейного преобразования число из интервала `[0, 1]` преобразуется в число из интервала `[a,b]`: значение функции `rand` домножается на константу `(b-a)` и прибавляется константа `a`.

Для полученного таким образом значения аргумента `x` вычисляется значение функции, которое помещается в элемент `Y(k)` массива через присвоение (=). Чтобы заполнить все `n` элементов массива `Y`, необходимо вычисление значений аргумента и функции повторить (в цикле) для всех элементов массива с номерами `k` от `1` до `n`.

Вспомогательная переменная x может быть не только скаляром (как в варианте 1), но и массивом (вариант 2). Тогда она должна содержать все аргументы, в которых далее будут вычисляться значения функции. Стандартная функция `rand(1,n)` с параметрами позволяет сформировать одномерный массив-строку из `n` случайных вещественных чисел, подчиняющихся равномерному распределению на интервале `[0,1]`. Большинство стандартных операций в MatLab расширено до операций с массивами как с матрицами. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Учитывая эти свойства стандартных функций MatLab, сформированный в варианте 2 с помощью функции `rand(1,n)` массив последовательно преобразуется путем сложения и умножения. Результирующий массив целиком присваивается (=) переменной `x`. Этот массив используется в качестве аргумента косинуса. По правилам вычисления функций от матрицы косинус вычисляется от каждого элемента массива `x`, формируя массив той же длины. Дальнейшие арифметические операции преобразуют его в соответствии с заданной формулой. Таким образом, в варианте 2 искомый массив `Y`, как и массив `x`, заполняется целиком.

В вариантах 3 и 4, в отличие от вариантов 1 и 2, формула записана в виде m-функции в файле с именем `f.m`. В качестве аргумента функции `f(x)` (также как для стандартных функций в вариантах 1 и 2) можно использовать как вещественное число (`x` в варианте 3), так и массив вещественных чисел (`x` в варианте 4).

Значения элементов массива `Y`, а также вспомогательной переменной `x` в командном окне не выводятся, так как в конце каждой операции присвоения используется символ `;`, подавляющий вывод по умолчанию, а оператор `disp` нигде не применяется.

Описание переменных. Учитывая, что результатом вычисления косинуса является вещественное число, переменная `Y` является числовым вещественным массивом. Стандартная функция `rand` генерирует вещественное число, поэтому вспомогательная переменная `x` является либо числовым вещественным скаляром, либо числовым вещественным одномерным массивом. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

По смыслу `a` и `b` – границы интервала. В общем случае они могут принимать как целые, так и дробные значения. Поэтому переменные `a` и `b` – это числовые вещественные скаляры.

5.6. Вычисления с массивами

Задание 5.6.1. Для массива `X` длиной `n` найдите сумму всех элементов $\sum_{k=1}^{n} X_k$.

Программа, вариант 1.

```
s = 0;
for k = 1:n
   s = s+X(k);
end
```

Программа, вариант 2.

```
s = sum(X);
```

Пояснения. В варианте 1 для вычисления значения суммы используется рекуррентное соотношение вида `s = s+...` . В зависимости от того, сумму чего надо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В данной задаче необходимо складывать элементы массива, поэтому в рекуррентном соотношении после знака «+» стоит элемент массива `X(k)`.

Чтобы к сумме добавить n элементов массива, необходимо повторять рекуррентные вычисления в цикле от 1 до n. В данном случае номер слагаемого совпадает с номером элемента массива (k).

Так как в каждом рекуррентном вычислении используется «старое» значение переменной s, для корректного выполнения первого вычисления перед циклом нужно задать начальное значение этой переменной так, чтобы оно не повлияло на результат суммирования (s = 0).

В варианте 2 для вычисления суммы элементов массива X используется стандартная функция MatLab sum, в качестве параметра которой необходимо указать одномерный числовой массив, сумму элементов которого требуется вычислить. Искомое значение присваивается переменной s.

В обоих вариантах вычисления происходят без вывода значения суммы (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Тип результата s зависит от типа информации, содержащейся в массиве X. Если в массиве X хранятся только целые числа, то их сумма s тоже будет целым числом (числовой целочисленный скаляр). Если в массиве X хранятся вещественные числа, то их сумма s тоже будет вещественным числом (числовой вещественный скаляр). Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.6.2. Для массива X длиной n найдите среднее (среднее арифметическое) значение всех элементов.

Программа, вариант 1.

```
s = 0;
for k = 1:n
      s = s+X(k);
end
sr = s/n;
```

Программа, вариант 2.

```
sr = 0;
for k = 1:n
      sr = sr+X(k);
end
sr = sr/n;
```

Программа, вариант 3.

```
sr = 0;
for k = 1:n
      sr = sr+X(k)/n;
end
```

Программа, вариант 4.

```
sr = mean(X);
```

Пояснения. Чтобы вычислить среднее (среднее арифметическое) значение всех элементов массива, необходимо сначала найти сумму всех элементов, а затем полученное значение разделить на количество элементов:

$$\frac{1}{n}\sum_{k=1}^{n} X_k, \text{ или } \sum_{k=1}^{n} \frac{X_k}{n}.$$

В вариантах 1–3 для вычисления значения суммы используется рекуррентное соотношение вида `s = s+...` (или `sr = sr+...`). В зависимости от того, сумму чего надо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В вариантах 1 и 2 складывают элементы массива, поэтому в рекуррентном соотношении после знака «+» стоит элемент массива `X(k)`. В варианте 3 используется формула, в которой `1/n` внесена под знак суммы, поэтому в рекуррентном соотношении после знака «+» стоит элемент массива, деленный на `n`: `X(k)/n`. Сумма должна содержать `n` слагаемых (элементов массива), поэтому в вариантах 1–3 рекуррентные вычисления повторяют в `for`-цикле по номерам `k` элементов массива от `1` до `n`.

Так как в каждом рекуррентном вычислении (в вариантах 1–3) используется «старое» значение переменной `s` (или `sr`), для корректного выполнения первого вычисления перед циклом нужно задать начальное значение этой переменной так, чтобы оно не повлияло на результат суммирования (`s = 0` или `sr = 0`).

В варианте 3 после окончания суммирования выполнение каких-либо дополнительных действий не требуется – переменная `sr` содержит искомое значение. В вариантах 1 и 2 после вычисления суммы необходимо выполнить дополнительные вычисления – разделить полученное значение на количество элементов в массиве.

В варианте 1 для накопления суммы и вычисления среднего используются две разные переменные (`s` и `sr`). В варианте 2 в переменной `sr`

сначала накапливается сумма элементов, а затем в нее же помещается и значение среднего. Данный вариант возможен, если, например, значение суммы элементов нужно только для нахождения среднего, и далее в программе нигде самостоятельно использоваться не будет.

В варианте 4 для вычисления среднего арифметического значения элементов массива `X` используется стандартная функция MatLab `mean`, в качестве параметра которой необходимо указать одномерный числовой массив, среднее значение элементов которого требуется вычислить. Рассчитанное значение присваивается переменной `sr`.

Во всех вариантах вычисления происходят без вывода значения суммы и (или) среднего (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Окажется ли значение переменной `s` (суммы элементов массива) вещественным или целочисленным числовым скаляром, зависит от того, какие значения хранятся в одномерном числовом массиве `X`. В любом случае среднее значение `sr` для любого числового массива `X` является числовым (чаще всего вещественным) скаляром. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.6.3. Для одномерного числового массива `X` длиной `n` найдите среднее значение элементов, номера которых лежат в диапазоне от `k1` до `k2` включительно.

Программа, вариант 1.

```
s = 0;
for k = k1:k2
      s = s+X(k);
end
sr = s/(k2-k1+1);
```

Программа, вариант 2.

```
sr = 0;
for k = k1:k2
      sr = sr+X(k);
end
sr = sr/(k2-k1+1);
```

Программа, вариант 3.

```
sr = mean(X(k1:k2));
```

Пояснения. Для нахождения среднего (среднего арифметического) нужно сумму элементов разделить на их количество. Так как в вычислениях участвуют элементы массива с номерами от `k1` до `k2` включительно, то количество элементов в сумме равно `k2-k1+1`. А суммирование удобно проводить по номерам `k` элементов массива от `k1` до `k2`:

$$\frac{1}{k2-k1+1}\sum_{k=k1}^{k2} X_k \ .$$

В вариантах 1 и 2 сумма вычисляется с помощью рекуррентного соотношения вида `s = s+...` или `sr = sr+...`, поэтому перед циклом, повторяющим вычисление по рекуррентной формуле, нужно задать начальное значение переменной `s` (или `sr`), накапливающей сумму, так чтобы оно не повлияло на результат вычислений (`s = 0` или `sr = 0`, соответственно). В зависимости от того, сумму чего необходимо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В данной задаче необходимо складывать элементы массива, поэтому в рекуррентном соотношении после знака «+» стоит элемент массива `X(k)`.

Чтобы к сумме добавить элементы массива, номера которых лежат в интервале от `k1` до `k2`, рекуррентные вычисления повторяют в `for`-цикле по `k` от `k1` до `k2`.

Для получения среднего значения после вычисления суммы ее значение необходимо поделить на количество слагаемых (элементов массива, участвовавших в суммировании).

В варианте 1 для накопления суммы и вычисления среднего используются две разные переменные (`s` и `sr`). В варианте 2 в переменной `sr` сначала накапливается сумма элементов, а затем в нее же помещается и значение среднего. Этот вариант возможен, если, например, значение суммы элементов нужно только для нахождения среднего и далее в программе нигде использоваться не будет.

В варианте 3 для вычисления среднего значения элементов массива, номера которых лежат в интервале от `k1` до `k2`, используется стандартная функция MatLab `mean`. Однако для получения искомого результата в качестве параметра этой функции используется не весь массив, а только часть его. Когда вместо индекса элемента массива указывается последовательность, сформированная с помощью оператора `:`, то обращение происходит не к одному, а к совокупности элементов в целом.

Во всех вариантах вычисления происходят без вывода значения суммы и (или) среднего (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Окажется ли значение переменной `s` (суммы элементов массива) вещественным или целочисленным числовым скаляром, зависит от того, какие значения хранятся в массиве `X`. В любом случае среднее значение `sr` для любого числового массива `X` является числовым (чаще всего вещественным) скаляром. Переменные `k`, `k1` и `k2` содержат информацию о номерах элементов массива, участвующих в вычислениях, т.е. это числовые целочисленные скаляры.

Задание 5.6.4. Для одномерного числового массива `X` длиной `n` найдите произведение всех элементов.

Программа, вариант 1.

```
p = 1;
for k = 1:n
   p = p*X(k);
end
```

Программа, вариант 2.

```
p = prod(X);
```

Пояснения. В варианте 1 для вычисления значения произведения используется рекуррентное соотношение вида `p = p*…` . В зависимости от того, произведение чего надо вычислить, рекуррентное соотношение дополняется конкретным видом сомножителя. В данной задаче необходимо перемножать элементы массива, поэтому в рекуррентном соотношении после знака «`*`» стоит элемент массива `X(k)`.

Чтобы в произведении `p` перемножить `n` элементов массива, необходимо повторять рекуррентные вычисления в цикле от `1` до `n`. В данном случае номер сомножителя совпадает с номером элемента массива (`k`).

Так как в каждом рекуррентном вычислении используется «старое» значение переменной `p`, для корректного выполнения первого вычисления перед циклом необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат перемножения (`p = 1`).

В варианте 2 для вычисления произведения элементов массива `X` используется стандартная функция MatLab `prod`, в качестве параметра которой необходимо указать одномерный числовой массив, элементы кото-

рого необходимо перемножить. Рассчитанное значение присваивается переменной p.

В обоих вариантах вычисления происходят без вывода значения произведения (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Тип результата p зависит от типа информации, содержащейся в массиве X. Если в массиве X хранятся целые числа, то их произведение p тоже будет целым числом (числовой целочисленный скаляр). Если в массиве X хранятся вещественные числа, то их произведение p тоже будет вещественным числом (числовой вещественный скаляр). Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.6.5. С помощью значений, расположенных в одномерном числовом массиве X длиной n, рассчитайте значение по формуле $Q = \sqrt{3 + \sum_{k=1}^{n} \sin \frac{X_k}{10}}$ (X_k – элемент массива).

Программа.

```
s = 0;
for k = 1:n
   s = s+sin(X(k)/10);
end
Q = sqrt(3+s);
```

Пояснения. Для вычисления значения по указанной формуле необходимо сначала вычислить сумму $s = \sum_{k=1}^{n} \sin \frac{X_k}{10}$, а затем с полученным значением выполнить остальные действия: $Q = \sqrt{3 + s}$.

Для вычисления значения суммы используется рекуррентное соотношение вида s = s+... . В зависимости от того, сумму чего надо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В данной задаче складывают значения синуса, в качестве аргумента которого используется элемент массива, деленный на 10, поэтому в рекуррентном соотношении после знака «+» стоит выражение sin(X(k)/10). Рекуррентные вычисления повторяют в for-цикле по номерам k элементов массива от 1 до n.

Так как в каждом рекуррентном вычислении используется «старое» значение переменной s, для корректного выполнения первого вычисления перед циклом необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат суммирования (s = 0).

После вычисления суммы необходимо выполнить дополнительные математические операции – прибавить 3 и извлечь корень квадратный. Полученное значение присваивается переменной Q.

Вычисления происходят без вывода значений (в конце каждого оператора присвоения используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. По условию задачи X – числовой массив. Учитывая, что результатом вычисления синуса является вещественное число, значение переменной s (суммы синусов) является вещественным числовым скаляром, если в массиве X хранятся целые и (или) вещественные значения. Корень квадратный из вещественного числа тоже является вещественным. Поэтому переменная Q – тоже вещественный числовой скаляр. Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.6.6. Найдите произведение диагональных элементов квадратной числовой матрицы A (размером n×n).

Программа.

```
p = 1;
for k = 1:n
      p = p*A(k,k);
end
```

Пояснения. Для вычисления значения произведения используется рекуррентное соотношение вида p = p*… . В зависимости от того, произведение чего надо вычислить, рекуррентное соотношение дополняется конкретным видом сомножителя. В данной задаче необходимо перемножать диагональные элементы квадратной матрицы, поэтому в рекуррентном соотношении после знака «*» стоит элемент массива A(k,k) – элемент, номер строки которого совпадает с номером столбца.

Чтобы перемножить все диагональные элементы матрицы, необходимо повторять рекуррентные вычисления в цикле по строкам от 1 до n (для диагональных элементов номера строк и столбцов совпадают). В данном случае номер сомножителя совпадает с номером строки (k).

Так как в каждом рекуррентном вычислении используется «старое» значение переменной p, для корректного выполнения первого вычисления перед циклом необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат перемножения (p = 1).

Вычисления происходят без вывода значения произведения (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Тип результата p зависит от типа информации, содержащейся в массиве A. Если все диагональные элементы матрицы A являются целыми числами, то их произведение p тоже будет целым числом (числовой целочисленный скаляр). Если диагональные элементы матрицы A являются вещественными числами, то их произведение p тоже будет вещественным числом (числовой вещественный скаляр). Переменные k и n содержат информацию о номере и количестве (соответственно) строк (столбцов) в матрице, т.е. это числовые целочисленные скаляры.

Задание 5.6.7. Найдите сумму всех элементов числовой матрицы A размером n×m.

Программа.

```
s = 0;
for k = 1:n
for p = 1:m
      s = s+A(k,p);
end
end
```

Пояснения. Для вычисления значения суммы используется рекуррентное соотношение вида s = s+… . В зависимости от того, сумму чего надо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В данной задаче необходимо складывать элементы матрицы, поэтому в рекуррентном соотношении после знака «+» стоит элемент двумерного массива A(k,p).

Чтобы к сумме добавить все элементы массива, необходимо перебрать все строки и все столбцы матрицы с помощью двух for-циклов: по строкам k от 1 до n; по столбцам p от 1 до m.

Так как в каждом рекуррентном вычислении используется «старое» значение переменной s, для корректного выполнения первого вычисле-

ния перед циклами необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат суммирования (`s = 0`).

Вычисления происходят без вывода значения суммы (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Тип результата вычислений `s` зависит от типа информации, содержащейся в массиве `A`. Если в массиве `A` хранятся только целые числа, то их сумма `s` тоже будет целым числом (числовой целочисленный скаляр). Если в массиве `A` хранятся вещественные числа, то их сумма `s` тоже будет вещественным числом (числовой вещественный скаляр). Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) строк в матрице, а переменные `p` и `m` – о номере и количестве (соответственно) столбцов, т.е. это числовые целочисленные скаляры.

Задание 5.6.8. Вычислите элементы массива `S` с помощью поэлементного линейного преобразования одномерного числового массива `X` длиной `n` с известными коэффициентами `K` и `L`.

Программа, вариант 1.

```
for k = 1:n
   S(k) = K*X(k)+L;
end
```

Программа, вариант 2.

```
S = K*X+L;
```

Пояснения. В варианте 1 поэлементное преобразование исходного массива реализуется с явным использованием `for`-цикла. Каждый элемент `X(k)` массива умножается на `K`, к результату прибавляется `L`. Полученное значение присваивается элементу массива `S` с тем же номером `k`. Чтобы преобразовать все элементы исходного массива, эти вычисления выполняются в `for`-цикле по номерам `k` элементов массива от `1` до `n`.

В варианте 2 для преобразования массива используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Учи-

тывая эти свойства стандартных функций MatLab, в варианте 2 исходный массив X преобразуется путем последовательного умножения и сложения (линейного преобразования). Полученный результат присваивается искомому массиву S целиком. Линейное преобразование выполняется поэлементно, но без явного использования цикла.

В обоих вариантах преобразование массива происходит без вывода значений элементов результирующего массива (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. По смыслу задачи переменные K и L – числовые скаляры, а переменная X – одномерный числовой массив. Результирующая переменная S также будет одномерным числовым массивом той же длины. Если все переменные K, L и X содержат только целочисленные данные, то результирующий массив также будет целочисленным. Если хотя бы одна из переменных K, L и X содержит вещественные данные, то результирующий массив будет вещественным. Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.6.9. Вычислите элементы матрицы S с помощью поэлементного линейного преобразования матрицы X размером n×m с известными коэффициентами K и L.

Программа, вариант 1.

```
for k = 1:n
for p = 1:m
   S(k,p) = K*X(k,p)+L;
end
end
```

Программа, вариант 2.

```
S = K*X+L;
```

Пояснения. В варианте 1 поэлементное преобразование исходной матрицы (двумерного массива) реализуется с явным использованием for-циклов. Каждый элемент X(k,p) массива умножается на K, к результату прибавляется L. Полученное значение присваивается элементу матрицы (массива) S с тем же номером k строки и тем же номером p столбца. Чтобы преобразовать все элементы исходного массива, эти вычисления выполняются в двух for-циклах: по номерам k строк от 1 до n; по номерам p столбцов от 1 до m.

В варианте 2 для преобразования массива используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Учитывая эти свойства стандартных функций MatLab, в варианте 2 исходная матрица `X` преобразуется путем последовательного умножения и сложения (линейного преобразования). Полученный результат присваивается искомой матрице `S` целиком. Линейное преобразование выполняется поэлементно, но без явного использования циклов.

В обоих вариантах преобразование матрицы происходит без вывода значений элементов результирующей матрицы (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. По смыслу задачи переменные `K` и `L` – числовые скаляры, а переменная `X` – двумерный числовой массив. Результирующая переменная `S` также будет двумерным числовым массивом. Если все переменные `K`, `L` и `X` содержат только целочисленные данные, то результирующий массив также будет целочисленным. Если хотя бы одна из переменных `K`, `L` и `X` содержит вещественные данные, то результирующий массив будет вещественным. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) строк в матрице, а переменные `p` и `m` – о номере и количестве (соответственно) столбцов, т.е. это числовые целочисленные скаляры.

5.7. Поиск минимального и максимального элементов массива

Задание 5.7.1. В одномерном массиве `X` (строке) длиной `n` найдите минимальное значение.

Программа, вариант 1.

```
minX = X(1);
for k = 1:n
      if X(k)<minX
            minX = X(k);
      end
end
```

Программа, вариант 2.

```
minX = min(X);
```

Пояснения. В варианте 1 текущее значение минимального элемента массива `minX` сравнивается (с помощью оператора `if`) поочередно с каждым элементом. Если значение элемента `X(k)` массива меньше, чем `minX`, значение переменной `minX` заменяется (путем присвоения) значением элемента `X(k)` массива. В этом случае дальнейшее сравнение происходит уже с измененным значением переменной `minX`.

Поиск проводят по всем элементам массива – в цикле по номерам `k` элементов массива от `1` до `n`.

До начала сравнения необходимо задать начальное значение переменной `minX`. Чтобы обеспечить корректность сравнения, в качестве начального значения переменной `minX` берут значение одного из элементов массива, например первого `X(1)`.

В варианте 2 для определения минимального значения среди элементов массива `X` используется стандартная функция MatLab `min`, в качестве параметра которой необходимо указать массив, в котором следует выполнять поиск. Найденное значение присваивается переменной `minX`.

В задаче запоминается только минимальное значение, а его расположение в массиве остается неизвестным.

В обоих вариантах поиск происходит без вывода минимального значения (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `X` может содержать любые данные, для которых определена операция сравнения «меньше», например числовые (числовой одномерный массив). Если переменная `X` содержит числовые данные, то искомое значение `minX` будет являться числовым скаляром. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это числовые целочисленные скаляры.

Задание 5.7.2. В одномерном массиве `X` (строке) длиной `n` найдите номер элемента, имеющего минимальное значение.

Программа.

```
Nmin = 1;
for k = 1:n
      if X(k)<X(Nmin)
            Nmin = k;
      end
```

```
end
```

Пояснения. Переменная `Nmin` будет хранить информацию о номере элемента массива, имеющего минимальное значение. Тогда `X(Nmin)` – значение минимального элемента. Это значение поочередно сравнивают (с помощью оператора `if`) с каждым элементом `X(k)` массива. Если значение некоторого элемента `X(k)` массива меньше, чем значение элемента массива с номером `Nmin`, то значение переменной `Nmin` заменяют (путем присвоения) на значение переменной `k` (запоминают номер элемента, значение которого меньше, чем текущее минимальное).

Поиск проводят по всем элементам массива – в цикле по номерам `k` элементов массива от `1` до `n`.

До начала поиска необходимо задать значение переменной `Nmin`. Чтобы обеспечить корректность сравнения, в качестве начального значения переменной `Nmin` берут номер одного из элементов массива, например первого (`Nmin = 1`).

При таком подходе запоминается только номер элемента массива, значение которого минимально, само минимальное значение в отдельную переменную не сохраняется, но легко может быть получено как `X(Nmin)`.

Поиск происходит без вывода номера минимального элемента массива (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `X` может содержать любые данные, для которых определена операция сравнения «меньше», например числовые (числовой одномерный массив). Учитывая, что элементы в массивах нумеруются целыми числами, начиная с единицы, искомое значение `Nmin` будет являться числовым целочисленным скаляром. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это тоже числовые целочисленные скаляры.

Задание 5.7.3. В матрице `A` размерами `n×m` в строке с номером `k` определите, в каком столбце расположен максимальный элемент.

Программа.

```
Pmax = 1;
for p = 1:m
      if A(k,p) > A(k,Pmax)
            Pmax = p;
      end
```

```
    end
```

Пояснения. Переменная `Pmax` будет хранить информацию о номере столбца, в котором расположен элемент, имеющий максимальное значение. Тогда `A(k,Pmax)` – значение максимального элемента матрицы в строке `k`. Это значение поочередно сравнивают с каждым элементом `A(k,p)` матрицы из строки `k` (с помощью оператора `if`). Если значение текущего элемента `A(k,p)` массива больше, чем значение элемента массива с номерами `k` и `Pmax`, то значение переменной `Pmax` заменяют (путем присвоения) на значение переменной `p` (запоминают номер столбца элемента, значение которого больше, чем текущее максимальное).

Поиск проводят в заданной строке `k` в цикле по номерам `p` столбцов матрицы от `1` до `m`.

До начала поиска необходимо задать начальное значение переменной `Pmax`. Чтобы обеспечить корректность сравнения, в качестве начального значения переменной `Pmax` можно взять номер одного из столбцов матрицы, например первого (`Pmax = 1`).

При таком подходе запоминается только номер столбца элемента матрицы, имеющего максимальное значение для данной строки, само максимальное значение в отдельную переменную не сохраняется, но легко может быть получено как `A(k,Pmax)`.

Поиск происходит без вывода номера столбца, в котором находится максимальный элемент (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `A` может содержать любые данные, для которых определена операция сравнения «больше», например числовые (числовой двумерный массив). Учитывая, что столбцы в массивах нумеруются целыми числами, начиная с единицы, искомое значение `Pmax` будет являться числовым целочисленным скаляром. Переменная `k` содержит информацию о номере строки матрицы, а переменные `p` и `m` – о номере и количестве (соответственно) столбцов, т.е. это тоже числовые целочисленные скаляры.

Задание 5.7.4. В матрице `A` размерами `n×m` в столбце с номером `p` определите, на каком месте расположен максимальный по модулю элемент.

Программа.

```
Kmax = 1;
for k = 1:n
      if abs(A(k,p)) > abs(A(Kmax,p))
            Kmax = k;
      end
end
```

Пояснения. Переменная `Kmax` будет хранить информацию о номере строки, в которой располагается элемент, имеющий максимальное по модулю значение. Тогда `A(Kmax,p)` – значение максимального по модулю элемента матрицы в столбце `p`. Модуль (`abs`) этого значения поочередно сравнивают (с помощью оператора `if`) с модулем каждым элементом `A(k,p)` матрицы из столбца `p`. Если модуль текущего элемента `A(k,p)` больше, чем модуль элемента массива с номерами `Kmax` и `p`, то значение переменной `Kmax` заменяют (путем присвоения) на значение переменной `k` (запоминают номер строки элемента, модуль значения которого больше, чем модуль текущего максимального).

Поиск проводят для заданного столбца `p` в цикле по номерам `k` строк матрицы от `1` до `n`.

До начала поиска необходимо задать начальное значение переменной `Kmax`. Чтобы обеспечить корректность сравнения, в качестве начального значения переменной `Kmax` можно взять номер одной из строк матрицы, например первой (`Kmax = 1`).

При таком подходе запоминается только номер строки элемента матрицы, модуль значения которого максимален для данного столбца, само максимальное по модулю значение в отдельную переменную не сохраняется, но легко может быть получено как `abs(A(Kmax,p))`.

Поиск происходит без вывода номера строки, в которой находится максимальный по модулю элемент (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `A` может содержать любые данные, для которых определена операция сравнения «больше», например числовые (числовой двумерный массив). Учитывая, что строки в массивах нумеруются целыми числами, начиная с единицы, искомое значение `Kmax` будет являться числовым целочисленным скаляром. Переменная `p`

содержит информацию о номере столбца матрицы, а переменные k и n – о номере и количестве (соответственно) строк, т.е. это тоже числовые целочисленные скаляры.

Задание 5.7.5. В матрице A размерами n×m определите, на каком месте расположен максимальный элемент.

Программа.

```
Kmax = 1;
Pmax = 1;
for k = 1:n
for p = 1:m
      if A(k,p)>A(Kmax,Pmax)
            Kmax = k;
            Pmax = p;
      end
end
end
```

Пояснения. Переменные Kmax и Pmax будут хранить информацию о номере строки и номере столбца, соответственно, элемента матрицы, имеющего максимальное значение. Тогда A(Kmax,Pmax) – значение максимального элемента. Это значение поочередно сравнивают (с помощью оператора if) с каждым элементом A(k,p) матрицы. Если текущее значение элемента A(k,p) массива больше, чем значение элемента массива с номерами Kmax, Pmax, то значение переменной Kmax заменяют (путем присвоения) на значение переменной k (запоминают номер строки элемента, значение которого больше, чем текущее максимальное), а значение переменной Pmax заменяют на значение переменной p (запоминают номер столбца).

Поиск проводят по всем строкам (для всех k от 1 до n) и всем столбцам (для всех p от 1 до m).

До начала поиска необходимо задать значения переменных Kmax и Pmax. Чтобы обеспечить корректность сравнения, в качестве начального значения переменной Kmax можно взять номер одной из строк матрицы, например первой (Kmax = 1), а в качестве начального значения переменной Pmax можно взять номер одного из столбцов, например тоже первого (Pmax = 1).

При таком подходе запоминаются только номера строки и столбца элемента матрицы, значение которого максимально, само максимальное значение в отдельную переменную не сохраняется, но легко может быть получено как элемент массива с номерами `Kmax` и `Pmax`: `A(Kmax,Pmax)`.

Поиск происходит без вывода номеров строки и столбца, в которых находится максимальный элемент (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `A` может содержать любые данные, для которых определена операция сравнения «больше», например числовые (числовой двумерный массив). Учитывая, что строки и столбцы в массивах нумеруются целыми числами, начиная с единицы, искомые значения `Kmax` и `Pmax` будет являться числовыми целочисленными скалярами. Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) строк в матрице, а переменные `p` и `m` – о номере и количестве (соответственно) столбцов, т.е. это тоже числовые целочисленные скаляры.

5.8. Поиск информации в массиве

Задание 5.8.1. Из одномерного массива `X` (строки) длиной `n` выведите на экран элементы: а) номера которых кратны числу `q`; б) значения которых меньше `z`.

Программа.

```
а) for k = 1:n
    if rem(k,q)==0
    disp(['X(' int2str(k) ') = ' ...
                              num2str(X(k), '%6.3f')])
    end
   end

б) for k = 1:n
    if X(k)<z
    disp(['X(' int2str(k) ') = ' ...
                              num2str(X(k), '%6.3f')])
    end
   end
```

Пояснения. В обоих случаях необходимо: для каждого элемента массива проверить, выполняется ли соответствующее условие; если условие выполняется – элемент массива выводится с помощью оператора `disp`.

Вывод найденных элементов `X(k)` массива в операторе `disp` сопровождается пояснением, какой порядковый номер `k` имеет данный элемент в массиве. Одним оператором `disp` выводят два строковых пояснения и две числовые величины. Чтобы их можно было объединить в одном операторе `[]`, необходимо числовое значение `X(k)` преобразовать в строковое с помощью функции `num2str` , а числовое значение `k` преобразовать в строковое с помощью функции `int2str`. Объединяемые оператором `[]` строковые величины отделяются пробелами. Спецификация `'%6.3f'` используется в функции `num2str` для преобразования значений найденных элементов к десятичному формату с фиксированной запятой и тремя знаками после нее независимо от формата, используемого в MatLab по умолчанию.

Для переноса слишком длинных операторов на другую строку используется многоточие (`...`).

Для выполнения части ***а*** необходимо проверять (с помощью оператора `if`), кратен ли номер `k` элемента массива заданному числу `q`. По правилам математики целое число `k` кратно целому числу `q`, если остаток от деления `k` на `q` равен нулю. Чтобы проверить, справедливо ли данное утверждение для текущего значения `k`, с помощью стандартной функции `rem(k,q)`, параметрами которой являются делимое и делитель, получают остаток от деления `k` на `q` и сравнивают полученное значение с нулем. Проверку выполняют для всех номеров `k` элементов массива в `for`-цикле по `k` от `1` до `n`.

Для выполнения части ***б*** необходимо сравнить (с помощью оператора `if`) значение элемента `X(k)` массива с заданным числом `z`: если значение элемента массива меньше числа `z`, это значение выводится оператором `disp`. Проверку выполняют для всех элементов массива в `for`-цикле по номерам `k` этих элементов от `1` до `n`.

Описание переменных. Переменная `X` может содержать любые данные, для которых определена операция сравнения «меньше», например числовые (числовой одномерный массив). Учитывая, что элементы в массивах нумеруются целыми числами начиная с единицы, переменная `q` должна являться числовым целочисленным скаляром. Переменная `z` должна являться числовым скаляром (целочисленным или веществен-

ным). Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это тоже числовые целочисленные скаляры.

Задание 5.8.2. Для одномерного массива X (строки) длиной n найдите сумму элементов массива: а) номера которых кратны числу q; б) значения которых меньше z.

Программа.

```
а)     s = 0;
       for k = 1:n
          if rem(k,q)==0
             s = s+X(k);
          end
       end

б)     s = 0;
       for k = 1:n
          if X(k)<z
             s = s+X(k);
          end
       end
```

Пояснения. В обоих случаях необходимо суммировать элементы. Для этого используется рекуррентное соотношение вида s = s+… . В зависимости от того, сумму чего надо вычислить, рекуррентное соотношение дополняется конкретным видом слагаемого. В данной задаче необходимо складывать элементы массива, поэтому в рекуррентном соотношении после знака «+» стоит элемент массива X(k). Так как в каждом рекуррентном вычислении используется «старое» значение переменной s, для корректного выполнения первого вычисления перед циклом необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат суммирования (s = 0).

Несмотря на то, что с помощью for-цикла перебираются номера всех элементов массива, суммировать необходимо только те, которые удовлетворяют заданным условиям. Поэтому в каждом цикле по k (номерам элементов массива) от 1 до n сначала происходит проверка соответствующего условия, и если оно истинно, то элемент X(k) массива добавляется к текущему значению суммы. Для выполнения части ***а*** и части ***б*** необходимо проверять два разных условия.

Для выполнения части ***а*** необходимо проверять (с помощью оператора `if`), кратен ли номер `k` элемента массива заданному числу `q`. По правилам математики целое число `k` кратно целому числу `q`, если остаток от деления `k` на `q` равен нулю. Чтобы проверить, справедливо ли данное утверждение для текущего значения `k`, с помощью стандартной функции `rem(k,q)`, параметрами которой являются делимое и делитель, получают остаток от деления `k` на `q` и сравнивают полученное значение с нулем.

Для выполнения части ***б*** необходимо сравнить (с помощью оператора `if`) значение элемента `X(k)` массива с заданным числом `z` – если значение элемента массива меньше числа `z`, то элемент массива прибавляется к текущему значению суммы.

В обоих случаях вычисления происходят без вывода значения суммы (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `X` может содержать любые данные, для которых определена операция сравнения «меньше», например числовые (числовой одномерный массив). Учитывая, что элементы в массивах нумеруются целыми числами начиная с единицы, переменная `q` должна являться числовым целочисленным скаляром. Переменная `z` должна являться числовым скаляром (целочисленным или вещественным). Переменные `k` и `n` содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это тоже числовые целочисленные скаляры. Если переменная `X` – числовой массив, то переменная `s`, содержащая значение суммы части элементов массива, будет числовым скаляром (вещественным или целочисленным в зависимости от того, какие данные хранятся в массиве).

Задание 5.8.3. Определите количество элементов числового одномерного массива `X` (строки) длиной `n`, значения которых меньше числа `z`.

Программа, вариант 1.

```
%p - количество найденных элементов
p = 0;
for k = 1:n
      if X(k) < z
            p = p+1;
      end
end
```

Программа, вариант 2.

```
% p - количество найденных элементов
p = 0;
for k = 1:n
  if X(k) < z
  disp(['X(' int2str(k) ') = ' ...
                              num2str(X(k, '%6.3f'))])
  p=p+1;
  end
end
```

Пояснения. Переменная `p` будет накапливать и хранить информацию о количестве элементов, удовлетворяющих заданному условию. Каждый раз, когда проверяемое условие окажется истинным, эту переменную необходимо увеличивать на единицу. Для этого используется рекуррентное соотношение вида `p = p+1`. Так как в каждом рекуррентном вычислении используется «старое» значение переменной `p`, для корректного выполнения первого вычисления перед циклом необходимо задать начальное значение этой переменной так, чтобы оно не повлияло на результат подсчета количества найденных элементов (`p = 0`).

Несмотря на то, что с помощью `for`-цикла перебираются номера всех элементов массива (`k` от `1` до `n`), переменную `p` необходимо увеличивать только в тех случаях, когда проверяемый с помощью условного оператора `if` элемент `X(k)` массива меньше числа `z`. Поэтому в цикле по `k` (номерам элементов массива) сначала происходит проверка условия.

В варианте 2 программа не только подсчитывает количество элементов, удовлетворяющих заданному условию, но и выводит эти элементы с помощью оператора `disp`. Это позволяет облегчить проверку правильности проведенных поисков.

Вывод найденных элементов `X(k)` массива в операторе `disp` сопровождается пояснением, какой порядковый номер `k` имеет данный элемент в массиве. Одним оператором `disp` выводят два строковых пояснения и две числовые величины. Чтобы их можно было объединить в одном операторе `[]`, необходимо числовое значение `X(k)` преобразовать в строковое с помощью функции `num2str` , а числовое значение `k` преобразовать в строковое с помощью функции `int2str`. Объединяемые оператором `[]` строковые величины отделяются пробелами. Спецификация `'%6.3f'`

используется в функции num2str для преобразования значений найденных элементов к десятичному формату с фиксированной запятой и тремя знаками после нее независимо от числового формата, используемого в MatLab по умолчанию.

Для переноса слишком длинных операторов на другую строку используется многоточие (...).

Вычисления происходят без вывода значения количества p элементов массива, удовлетворяющих заданному условию (в конце рекуррентной формулы используется символ ; для подавления вывода по умолчанию).

Описание переменных. Переменная X может содержать любые данные, для которых определена операция сравнения «меньше», например числовые (числовой одномерный массив). Учитывая, что элементы в массивах нумеруются целыми числами, начиная с единицы, переменная p должна являться числовым целочисленным скаляром. Переменная z должна являться числовым скаляром (целочисленным или вещественным). Переменные k и n содержат информацию о номере и количестве (соответственно) элементов в массиве, т.е. это тоже числовые целочисленные скаляры.

5.9. Изменение взаимного расположения элементов в массиве

Задание 5.9.1. В одномерном массиве X (строке) длиной n поменяйте местами значения элементов с номерами k1 и k2 ($X_{k1} \leftrightarrow X_{k2}$).

Программа.

```
c = X(k1);
X(k1) = X(k2);
X(k2) = c;
```

Пояснения. Чтобы две переменные (элемента массива) обменялись данными, необходима третья вспомогательная переменная c, в которую временно помещается (присваивается) одно из значений, например значение из элемента X(k1). Далее значение элемента X(k1) заменяется (присваивается) на значение из элемента X(k2), а значение элемента X(k2) массива заменяется (присваивается) на значение из вспомогательной переменной c.

Обмен данными происходит без вывода значений элементов массива и вспомогательной переменной (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Данный алгоритм можно использовать для обмена данными между элементами одномерного массива `X`, содержащего информацию любого типа. Тип вспомогательной переменной `c` зависит от типа информации, хранящейся в исходном массиве. Например, если в массиве `X` хранятся вещественные числа (числовой вещественный массив), то переменная `c` – числовой вещественный скаляр. По смыслу задачи переменные `k1` и `k2` – это номера элементов массива, поэтому они должны иметь целые положительные значения и являться числовыми целочисленными скалярами.

Задание 5.9.2. В одномерном массиве `X` (строке) длиной `n` поменяйте местами значения элементов по закону $X_1 \leftrightarrow X_2$, $X_3 \leftrightarrow X_4$, $X_5 \leftrightarrow X_6$…

Программа.

```
for k1 = 1:2:n-1
    k2 = k1+1;
      c = X(k1);
      X(k1) = X(k2);
      X(k2) = c;
end
```

Пояснения. Чтобы представить заданный в условии задачи закон в виде перестановки $X_{k1} \leftrightarrow X_{k2}$, необходимо установить соответствие между индексами элементов (`k1` и `k2`) в каждой паре так, чтобы, например, `k2` можно было вычислить через `k1`. Проанализировав заданный в условии задачи закон:

$X_1 \leftrightarrow X_2$	`k1 = 1`	`k2 = 2`
$X_3 \leftrightarrow X_4$	`k1 = 3`	`k2 = 4`
$X_5 \leftrightarrow X_6$	`k1 = 5`	`k2 = 6` …

можно получить искомую зависимость `k2 = k1+1`. При этом индекс `k1` первого из двух переставляемых элементов изменяется последовательно (1, 3, 5, …) от `1` до `n-1` через `2` (с шагом `2`). Последнее (максимальное) значение индекса `k1` должно быть таким, чтобы соответствующее ему значение индекса `k2` не превышало количество `n` элементов в массиве.

Такое изменение индекса k1 позволяет организовать for-цикл по k1 от 1 до n-1 с шагом 2. Внутри цикла сначала вычисляется значение индекса k2, а затем элементы X(k1) и X(k2) обмениваются данными.

Чтобы две переменные (элемента массива) обменялись данными, необходима третья вспомогательная переменная c, в которую временно помещается (присваивается) одно из значений, например значение из элемента X(k1). Далее значение элемента X(k1) заменяется (присваивается) на значение из элемента X(k2), а значение элемента X(k2) массива заменяется (присваивается) на значение из вспомогательной переменной c.

Обмен данными происходит без вывода значений элементов массива и вспомогательных переменных c и k2 (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Данный алгоритм можно использовать для преобразования одномерного массива X, содержащего информацию любого типа. Тип вспомогательной переменной c зависит от типа информации, хранящейся в исходном массиве. Например, если в массиве X хранятся вещественные числа (числовой вещественный массив), то переменная c – числовой вещественный скаляр. По смыслу задачи переменные k1 и k2 – это номера элементов массива, а n – количество элементов в массиве, поэтому эти переменные должны иметь целые положительные значения и являться числовыми целочисленными скалярами.

Задание 5.9.3. Отсортируйте элементы одномерного массива X длиной n по возрастанию.

Программа, вариант 1.

```
% метод пузырька
for p =1:n-1
   for k = 1:n-p
      if X(k)>X(k+1)
      % меняют местами элементы массива
         c = X(k);
         X(k) = X(k+1);
         X(k+1) = c;
      end
   end
end
```

Программа, вариант 2.

```
for p = 1:n-1
% определяют номер минимального элемента
   Kmin = p;
   for k = p:n
      if X(k)<X(Kmin)
            Kmin = k;
      end
   % меняют местами элементы массива
      c = X(p);
      X(p) = X(Kmin);
      X(Kmin) = c;
   end
end
```

Программа, вариант 3.

```
X=sort(X);
```

Пояснения. В методе пузырька (вариант 1) попарно переставляются соседние элементы `X(k)` и `X(k+1)` массива, если до перестановки предыдущий (`X(k)`) был больше, чем последующий (`X(k+1)`). В результате за один проход по массиву (цикл по `k`) самый большой элемент окажется на последнем месте. Повторение этой операции позволит поставить второй по величине элемент на предпоследнее место и т.д. Поэтому, чтобы полностью упорядочить массив, необходимо повторить эту операцию не более чем `n-1` раз (цикл по `p`).

В варианте 2 сначала определяется место, на котором расположен минимальный элемент (цикл по `k`), найденный элемент переставляется с первым. Далее действия повторяются, не затрагивая первый элемент (определяется расположение второго по малости элемента, он переставляет его на второе место). Чтобы упорядочить массив полностью, необходимо найти `n-1` положение `Kmin` минимальных значений элементов массива (цикл по `p`).

Чтобы определить положение минимального среди элементов массива, имеющих индексы от `p` до `n`, необходима вспомогательная переменная `Kmin`, в которой хранится номер минимального элемента. Поиски начинаются с элемента с номером `p`, последовательно сравнивая (с помощью оператора `if`) элементы `X(k)` массива с элементом `X(Kmin)`. Если значение элемента `X(k)` оказалось меньше `X(Kmin)`, в переменной `Kmin` запоминается (присваивается) номер `k` элемента `X(k)`.

В вариантах 1 и 2 для перестановки двух элементов массива ($X_k \leftrightarrow X_{k+1}$ или $X_p \leftrightarrow X_{Kmin}$) используется вспомогательная переменная `c` (как описано в задаче 5.9.1).

В варианте 3 для сортировки массива `X` по возрастанию используется стандартная функция MatLab `sort`, в качестве параметра которой необходимо указать массив, который следует отсортировать. Преобразованный массив вновь присваивается переменной `X`.

Во всех вариантах процесс преобразования происходит без вывода как массива, так и вспомогательных переменных (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `X` может содержать любые данные, для которых определены операции сравнения «меньше» и «больше», например числовые (числовой одномерный массив). Учитывая, что элементы в массивах нумеруются целыми числами, начиная с единицы, переменные `k`, `p`, `Kmin` должны являться числовыми целочисленными скалярами. Переменная `n` содержат информацию о количестве элементов в массиве, т.е. это тоже числовой целочисленный скаляр. Тип вспомогательной переменной `c` зависит от типа информации, хранящейся в исходном массиве. Например, если в массиве `X` хранятся вещественные числа (числовой вещественный массив), то переменная `c` – числовой вещественный скаляр.

Задание 5.9.4. В матрице `A` размерами `n×m` поменяйте местами строки с номерами `k1` и `k2` ($A^p_{k1} \leftrightarrow A^p_{k2}$).

Программа, вариант 1.

```
for p = 1:m
      c = A(k1,p);
      A(k1,p) = A(k2,p);
      A(k2,p) = c;
end
```

Программа, вариант 2.

```
      c = A(k1,:);
      A(k1,:) = A(k2,:);
      A(k2,:) = c;
```

Пояснения. В каждом столбце необходимо поменять местами два элемента матрицы, относящихся к строкам с номерами `k1` и `k2`.

В варианте 1 в `for`-цикле по столбцам `p` от `1` до `m` (явное использование `for`-цикла для перебора всех возможных значений номеров столбцов) меняют местами элементы $A_{k1}^{p} \leftrightarrow A_{k2}^{p}$, расположенные в указанных строках одного и того же столбца `p`.

Чтобы два элемента массива обменялись данными, необходима третья вспомогательная переменная `c`, в которую временно помещается (присваивается) одно из значений, например значение из элемента `A(k1,p)`. Далее значение элемента `A(k1,p)` массива заменяется (присваивается) на значение из элемента `A(k2,p)`, а значение элемента `A(k2,p)` массива заменяется (присваивается) на значение из вспомогательной переменной `c`. Эти действия повторяются в цикле по столбцам.

В варианте 2 обмениваются данными целые строки. Когда в качестве второго индекса двумерного массива указывается символ `:`, обращение происходит к строке в целом.

Чтобы две строки массива обменялись данными, необходима третья вспомогательная переменная `c`, в которую временно помещается (присваивается) одна из строк, например, строка `A(k1,:)` с номером `k1`. Далее строка `A(k1,:)` заменяется (присваивается) строкой `A(k2,:)` с номером `k2`, а строка `A(k2,:)`заменяется (присваивается) строкой из вспомогательной переменной `c`. В этом случае в организации цикла нет необходимости.

В обоих вариантах обмен данными происходит без вывода значений элементов массива и вспомогательной переменной (используется символ `;` для подавления вывода по умолчанию, а оператор `disp` нигде не применяется).

Описание переменных. Переменная `A` может содержать данные любого типа, например числовые (числовой двумерный массив). Учитывая, что элементы в массивах нумеруются целыми числами, переменные `k1`, `k2`, `p` должны являться числовыми целочисленными скалярами. Переменная `m` содержат информацию о количестве столбцов в матрице, т.е. это тоже числовой целочисленный скаляр. Тип вспомогательной переменной `c` зависит от типа информации, хранящейся в исходном массиве. Например, если в массиве `A` хранятся вещественные числа (числовой вещественный двумерный массив), то в варианте 1 переменная `c` – числовой вещественный скаляр, а в варианте 2 – числовой вещественный массив-строка.

5.10. Изменение значений элементов в массивах

Задание 5.10.1. В матрице A размерами $n \times m_1$ замените столбец с номером p1 на столбец с номером p2 из матрицы B размерами $n \times m_2$.

Программа, вариант 1.

```
for k = 1:n
      A(k,p1) = B(k,p2);
end
```

Программа, вариант 2.

```
A(:,p1) = B(:,p2);
```

Пояснения. Каждому элементу из столбца p1 матрицы A необходимо присвоить значение из столбца p2 матрицы B, расположенное в той же строке. Количество строк в обеих матрицах одинаково, поэтому данная операция может быть реализована.

В варианте 1 присвоение организовано для каждой строки поэлементно. В for-цикле по строкам k от 1 до n значение элемента B(k,p2) присваивают значению элемента A(k,p1) (явное использование for-цикла для перебора всех возможных значений номеров строк).

В варианте 2 организовано присвоение целых столбцов. Когда в качестве первого индекса двумерного массива указывается символ :, обращение происходит к столбцу в целом. В этом случае в явной организации цикла нет необходимости.

В обоих вариантах присвоение происходит без вывода массива (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Данная операция может быть реализована для матриц, содержащих информацию любого типа. Однако тип данных, хранящихся в матрице A, должен совпадать с типом данных, хранящихся в матрице B. Поэтому переменные A и B – однотипные двумерные массивы. Учитывая, что элементы в массивах нумеруются целыми числами, переменные k, p1, p2 должны являться числовыми целочисленными скалярами. Обе матрицы должны иметь одинаковое количество строк. Переменная n содержит информацию о количестве строк в каждой из матриц, т.е. это тоже числовой целочисленный скаляр.

Задание 5.10.2. В матрице A размерами n×m к строке с номером k1 прибавьте строку с номером k2, умноженную на некоторое число z.

Программа, вариант 1.

```
for p = 1:m
      A(k1,p) = A(k1,p)+z*A(k2,p);
end
```

Программа, вариант 2.

```
A(k1,:) = A(k1,:)+z*A(k2,:);
```

Пояснения. Каждый элемент из строки k2 необходимо умножить на число z, сложить с находящимся в том же столбце элементом из строки k1 и заменить полученным значением тот же элемент из строки k1.

В варианте 1 вычисления организованы поэлементно для каждого столбца. В for-цикле по столбцам p от 1 до m (явное использование for-цикла для перебора всех возможных значений номеров столбцов) вычисляется значение A(k1,p)+z*A(k2,p) и присваивается элементу A(k1,p).

В варианте 2 для преобразования массива используются стандартные действия с массивами как с матрицами, которые являются встроенными функциями MatLab. Например, встроенная функция умножения в MatLab расширена, в частности, до умножения матрицы на число (поэлементно по правилам линейной алгебры). Этот вариант умножения выполняется, когда один из сомножителей является массивом, а другой скаляром. Учитывая эти свойства стандартных функций MatLab, в варианте 2 вычисления организованы с целыми строками. Когда в качестве второго индекса двумерного массива указывается символ :, обращение происходит к строке в целом как к одномерному массиву. В этом случае в явной организации цикла по столбцам нет необходимости. Исходная строка A(k2,:) матрицы преобразуется путем последовательного умножения и сложения со строкой A(k1,:). Полученный результат целиком присваивается строке A(k1,:) матрицы.

В обоих вариантах до начала вычислений исходная матрица считается заданной. Хотя для расчетов используются рекуррентные формулы, но отдельно задавать начальное значение для этих формул нет необходимости, так как с помощью рекуррентных формул не вычисляются новые, а изменяются известные ранее данные.

В обоих вариантах вычисления происходят без вывода массива (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Учитывая, что с элементами матрицы необходимо выполнить операции сложения и умножения, переменная A является числовым массивом, а переменная z – числовым скаляром (вещественным или целочисленным). Элементы в массивах нумеруются целыми числами, поэтому переменные k1, k2, p должны являться числовыми целочисленными скалярами. Переменная m содержит информацию о количестве столбцов в матрице, т.е. это тоже числовой целочисленный скаляр.

Задание 5.10.3. Из матрицы A размерами n×m получите матрицу B путем вычеркивания строки с номером g и столбца с номером h.

Программа, вариант 1.

```
for k = 1:g-1
for p = 1:h-1
      B(k,p) = A(k,p);
end
end
for k = 1:g-1
for p = h+1:m
      B(k,p-1) = A(k,p);
end
end
for k = g+1:n
for p = 1:h-1
      B(k-1,p) = A(k,p);
end
end
for k = g+1:n
for p = h+1:m
      B(k-1,p-1) = A(k,p);
end
end
```

Программа, вариант 2.

```
B = A;
B(g,:) = [];
B(:,h) = [];
```

Пояснения. Вычеркиваемые строка и столбец делят исходную матрицу на четыре части. В варианте 1 для каждой из этих частей записано правило, в соответствии с которым данные из матрицы A записываются в

определенное место матрицы B. Данные из строк, расположенных перед вычеркиваемой (цикл по k от 1 до g-1), записываются в строки с тем же номером k, что у исходной матрицы, – части 1 и 2. Данные из строк, расположенных после вычеркиваемой (цикл по k от g+1 до n), записываются в строки с меньшим номером (k-1) по сравнению с исходной матрицей – части 3 и 4.

Аналогично данные из столбцов, расположенных перед вычеркиваемым (цикл по p от 1 до h-1), записываются в столбцы с тем же номером p, что у исходной матрицы, – части 1 и 3. Данные из столбцов, расположенных после вычеркиваемого (цикл по p от h+1 до m), записываются в столбцы с меньшим номером (p-1) по сравнению с исходной матрицей – части 2 и 4.

Только после выполнения всех четырех частей программы матрица B размером (n-1)×(m-1) будет заполнена полностью.

В варианте 2 сначала все данные из матрицы A копируются (присваиваются) в матрицу B. Далее в матрице B поочередно удаляются строка с номером g и столбец с номером h. Когда в качестве одного из индексов двумерного массива указывается символ :, обращение происходит либо к строке в целом (символ : указывается вместо номера столбца), либо к столбцу в целом (символ : указывается вместо номера строки). В этом случае в явной организации соответствующих циклов нет необходимости. Использование оператора объединения [] без значений внутри означает «пустую» совокупность элементов нулевой длины. При присвоении пустой совокупности [] строке (или столбцу, соответственно) элементы из указанной строки (столбца) удаляются.

В обоих вариантах заполнение матрицы B происходит без ее вывода (используется символ ; для подавления вывода по умолчанию, а оператор disp нигде не применяется).

Описание переменных. Данная операция может быть реализована для исходной матрицы, содержащей информацию любого типа. Учитывая, что данные копируются (присваиваются) из одной матрицы в другую, переменные A и B являются однотипными двумерными массивами. Элементы в массивах нумеруются целыми числами, поэтому переменные k, p, g и h являются числовыми целочисленными скалярами. Переменные n и m содержат информацию о количестве строк и столбцов в исходной матрице, т.е. это тоже числовые целочисленные скаляры.

Глава 6. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ MATLAB

Графическое представление результатов обработки информации существенно влияет на оценку их достоверности и значимости при решении разнообразных задач. Среди видов графических представлений можно выделить: двумерные и трехмерные; статические и динамические (анимированные). Для реализации каждого из видов графики в MatLab разработан ряд стандартных функций.

6.1. График функции одной переменной

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

Для построения графика необходимо:

1) подготовить числовые данные;

2) создать графическое окно;

3) задать необходимые настройки отображения графика.

Для построения двумерного графика (графика функции одной переменной) числовые данные необходимо представить в виде двух одномерных массивов одинаковой длины. Один массив должен содержать значения аргумента, а другой – значения функции. Нумерация аргументов и значений функции должна совпадать.

В большинстве случаев аргументы выбирают равномерно расположенными на некотором фиксированном интервале `[a,b]`. Тогда для вычисления аргументов точек графика можно воспользоваться либо оператором двоеточия (`:`), либо функцией `linspace`. Например,

```
X = a:dx:b        или     X = linspace(a,b,n),
```

где `dx` – шаг изменения аргумента на интервале `[a,b]`, `n` – количество точек на графике. Если аргументы функции расположены на интервале `[a,b]` неравномерно, то их значения можно задать с клавиатуры, путем присвоения или рассчитать каким-либо способом.

Если аналитический вид функции, график которой предстоит построить, известен, то правило ее вычисления можно оформить в виде m-функции. Тогда значения второго массива, необходимого для построе-

ния графика, можно рассчитать, применяя функцию ко всему массиву аргументов или к отдельным его элементам. Например,

```
Y = f(X);
```

или

```
for k=1:length(X)
      Y(k) = f(X(k));
end
```

где `f` – функция одной переменной, записанная в отдельном m-файле с названием `f.m`.

В MatLab для создания графического окна существует стандартная функция `figure`. Многократное использование функции `figure` позволяет разместить графическую информацию в нескольких графических окнах. Если программа предполагает размещение графической информации только в одном графическом окне, то функцию `figure` можно НЕ использовать. Графическое окно будет создано автоматически при первом использовании какой-либо графической функции.

Например, стандартная функция `plot` служит для отображения в графическом окне двумерного графика. При вызове функции `plot` необходимо указать одномерный массив аргументов (на первом месте) и одномерный массив значений функции (на втором месте):

```
plot(X,Y)
```

На графике изображается такое количество точек, сколько координат содержат массивы аргументов `X` и значений функции `Y`. Точки соединяются прямыми.

График строится в текущем (или вновь созданном) графическом окне.

В процессе выполнения программы, содержащей графические функции, генерируемое программой изображение автоматически сохраняется в той же папке, где хранится m-файл с программным кодом, в виде специального файла с расширением (типом) fig.

При написании программного кода настройки отображения графика (оси, масштаб, цвет графика и т.д.), кроме значений координат точек графика, можно оставить заданными по умолчанию. При необходимости настройки, заданные по умолчанию, можно изменить.

Например, можно задать тип и цвет линий и точек на графике, если указать третий параметр (спецификацию) у функции `plot`. Этот параметр должен быть строковым и представлять комбинацию трех символов: тип линии; тип точки; цвет.

Тип линии		Тип точки	
-	Сплошная	.	Точка
:	Двойной пунктир	o	Окружность
-.	Штрих-пунктир	x	Крест
_	Штриховая	+	Плюс
Цвет		*	Звездочка
y	Желтый	s	Квадрат
m	Фиолетовый	d	Ромб
c	Голубой	v	Треугольник (вниз)
r	Красный	^	Треугольник (вверх)
g	Зеленый	<	Треугольник (влево)
b	Синий	>	Треугольник (вправо)
w	Белый	p	Пятиугольник
k	Черный	h	Шестиугольник

Например, `plot(X,Y,'-sr')`

Пояснения. В этом случае точки будут изображаться квадратами и соединяться сплошными линиями красного цвета.

Виды двумерных графиков

Функция	Описание
`plot(…)`	График функции одной переменной в декартовой системе координат
`line(…)`	Ломаная, соединяющая прямыми линиями точки, координаты которых заданы в декартовой системе координат (частный случай: если заданы координаты двух точек, строится прямая)
`loglog(…)`	График функции одной переменной в логарифмическом масштабе (по обеим осям)
`bar(…)`	Столбцовая диаграмма (вертикальная)
`barh(…)`	Столбцовая диаграмма (горизонтальная)
`hist(…)`	Гистограмма
`errorbar(…)`	График функции одной переменной с зонами погрешности в каждой точке
`semilogx(…)`	График функции одной переменной в логарифмическом масштабе по оси X (по оси Y линейный масштаб)

Функция	Описание
`semilogy(…)`	График функции одной переменной в логарифмическом масштабе по оси Y (по оси X линейный масштаб)
`stem(…)`	Дискретный график функции одной переменной (не соединяет точки)
`polar(…)`	График функции в полярной системе координат
`pie(…)`	Круговая диаграмма по значениям одномерного массива

Если среди параметров функции построения двумерного графика указан только один одномерный массив, то он считается массивом значений функции. В качестве массива аргументов тогда берется последовательность индексов элементов массива (1, 2, …).

К любому из перечисленных видов двумерных графиков могут применяться дополнительные настройки.

Дополнительные настройки

Функция	Описание
`title('надпись')`	Позволяет задать титульную надпись
`xlabel('надпись')`	Позволяет задать надпись для оси X
`ylabel('надпись')`	Позволяет задать надпись для оси Y
`text(x,y,'надпись')`	Позволяет задать надпись в точке с координатами (`x`, `y`)
`gtext('надпись')`	Позволяет задать надпись в позиции курсора мыши (навести курсор и нажать любую клавишу)
`axis([Xmin Xmax Ymin Ymax])`	Устанавливает диапазон координат по осям X и Y
`axis auto`	Устанавливает параметры осей по умолчанию
`axis equal`	Устанавливает одинаковое значение между метками по осям X и Y
`axis off`	Убирает оси
`axis on`	Восстанавливает оси
`grid off`	Отключает сетку
`grid on`	Добавляет сетку

Функция	Описание
`legend('надпись1', 'надпись2',…, Pos)`	Добавляет к текущему графику легенду из перечисленных надписей; `Pos` задает положение легенды (0 – выбирать автоматически; 1 – верхний правый угол; 2 – верхний левый угол; 3 – нижний левый угол; 4 – нижний правый угол)
`legend off`	Удаляет легенду
`hold on`	Включает режим продолжения вывода графиков в текущее окно
`hold off`	Выключает режим продолжения вывода графиков в текущее окно
`subplot(n,m,p)`	Разбивает графическое окно на подокна (`n` – количество подокон по горизонтали; `m` – количество подокон по вертикали; `p` – номер текущего подокна)
`zoom(factor)`	Устанавливает коэффициент увеличения (уменьшения)
`zoom on`	Включает режим интерактивного изменения масштаба текущего графика
`zoom off`	Выключает режим интерактивного изменения масштаба текущего графика
`zoom xon`	Включает режим интерактивного изменения масштаба текущего графика только по оси X
`zoom yon`	Включает режим интерактивного изменения масштаба текущего графика только по оси Y
`[theta,r]= cart2pol(x,y)`	Преобразование декартовых координат в полярные
`[x,y]= pol2cart(theta,r)`	Преобразование полярных координат в декартовы

Если явные настройки не сделаны, то используются настройки по умолчанию. Масштаб графика подбирается автоматически.

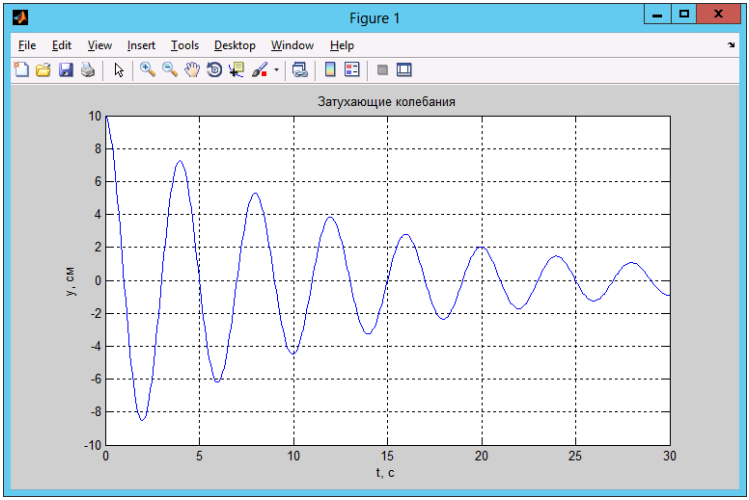


Рис. 6.1. Пример построения графика функции одной переменной

Пример (результат выполнения программного кода приведен на рис. 6.1):

```
% вычисление аргументов функции
   X = linspace(0,30,500);
% вычисление значений функции f(x)
   Y = f(X);
% построение графика
   plot(X,Y)
% дополнительные настройки
   title('Затухающие колебания');
   xlabel('t, c');
   ylabel('y, см');
   Ymax = max(Y); Ymin=-Ymax;
   axis([0 30 Ymin Ymax]);
   grid on

% функция f(x)= 10e^-0.08x cos(x·π/2) записана
% в отдельном файле с названием f.m
function y = f(x)
    y=10*exp(-0.08*x).*cos(pi/2*x);
end
```

Многие настройки можно также выполнить при помощи кнопок на панели инструментов и пунктов меню графического окна во время отображения графика, которые описаны в разделе 6.5.

6.2. График нескольких функции одной переменной

Каждая из функции построения двумерного графика позволяет на одном графике построить несколько функциональных зависимостей от одного и того же аргумента. Для отображения каждой функциональной зависимости необходимо задать одномерный массив аргументов `Xk`, одномерный массив значений функции `Yk` и, если нужно, спецификацию `sk` (`k` = 1, 2, …). Тогда для построения двумерного графика в декартовых координатах функцию `plot` следует записать в виде `plot(X1,Y1,s1, X2,Y2,s2, X3,Y3,s3,…)`.

Пример.

Задание 6.1. Постройте на одном графике две функциональные зависимости: 1) функции $f_1(x) = -x^2 + x + 4$ на интервале [–3, 5]; 2) функции $f_2(x) = 7\cos(2x^2)$ на интервале [–4, 4]. Точки первой функции отметьте окружностями.

Программа (результат выполнения программного кода приведен на рис. 6.2).

```
X1 = -3:0.5:5;           % аргументы первой функции
Y1 = f1(X1);             % значения первой функции
X2 = -4:0.01:4;          % аргументы второй функции
Y2 = f2(X2);             % значения второй функции
plot(X1,Y1,'-o', X2,Y2) % построение графиков
title('График двух функций');
```

```
% функция f1(x)=-x^2+x+4 записана
% в отдельном файле с названием f1.m
function y = f1(x)
   y=-x.^2+x+4;
end
```

```
% функция f2(x)= 7cos(2x^2) записана
% в отдельном файле с названием f2.m
function y = f2(x)
   y=7*cos(2*x.^2);
end
```

Пояснения. В этом примере для второй функции используется спецификация, установленная по умолчанию; кривые содержат разное количество точек.

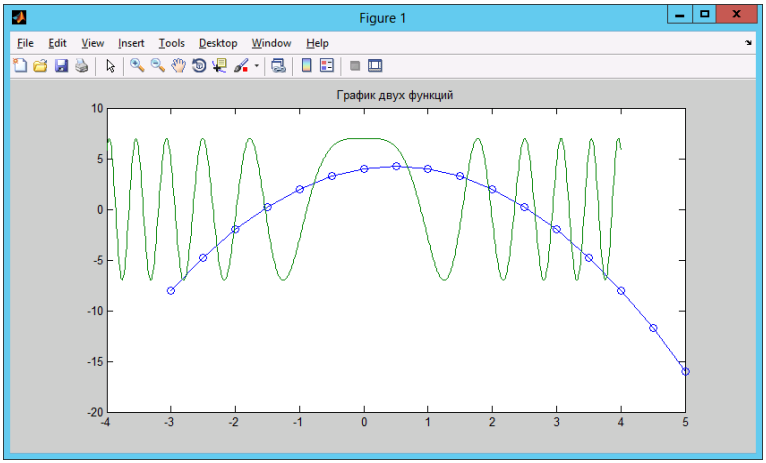


Рис. 6.2. Пример графика двух функций, заданных на разных интервалах

6.3. График функции двух переменных

Для построения графика двух переменных необходимо задать множество координат точек на плоскости XY и рассчитать значение функции в каждой из них. Данные должны быть представлены в виде трех массивов, например X, Y и Z.

Виды трехмерных графиков

Функция	Описание
`plot3(...)`	График функции двух переменных (поверхности) в декартовой системе координат – множество кривых
`mesh(...)`	График поверхности в виде сетки, цвет линий которой зависит от высоты
`meshc(...)`	График поверхности в виде сетки, цвет линий которой зависит от высоты, а также ее проекция в виде линий равного уровня
`surf(...)`	График поверхности в виде сетки, цвет ячеек которой зависит от высоты

Функция	Описание
`surfc(…)`	График поверхности в виде сетки, цвет ячеек которой зависит от высоты, а также ее проекция в виде линий равного уровня

Вызов функций для построения трехмерных графиков аналогичен вызову функций построения двумерных графиков. Только вместо двух массивов координат `X` и `Y` нужно указать три массива координат `X`, `Y` и `Z`. Далее при необходимости указываются спецификации для управления цветовой гаммой.

Для функции `plot3` все три массива `X`, `Y` и `Z` должны быть двумерными одинакового размера `m×n`. В этом случае поверхность изображается как множество из `m` кривых.

Пример.

Задание 6.2. Постройте график поверхности вида $f(x, y) = 0{,}1x^2 + 0{,}2y^2$ для $x \in [-10, 10]$, $y \in [-5, 5]$.

Программа (результат выполнения программного кода приведен на рис. 6.3).

```
n=60; m=30;
% массив X: m одинаковых строк, в каждой
% из которых n значений от -10 до 10
for k=1:m
   X(k,:) = linspace(-10,10,n);
end
% массив Y: n одинаковых столбцов, в каждом
% из которых m значений от -5 до 5
for k=1:n
   Y(:,k) = linspace(-5,5,m);
end
% массив Z: вычисляется для всех возможных пар
% значений X и Y
Z = f(X,Y);
plot3(X,Y,Z)
% функция f(x,y)= 0.1x^2+0.2y^2 записана
% в отдельном файле с названием f.m
function z = f(x,y)
   z=0.1*x.^2+0.2*y.^2;
end
```

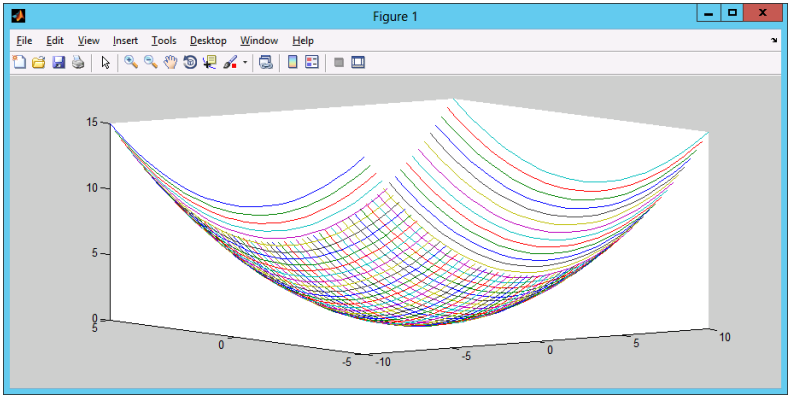


Рис. 6.3. Пример графика трехмерной поверхности, построенного с помощью функции `plot3`

Для остальных функций (`mesh, meshc, surf, surfc`) массивы аргументов `X` и `Y` должны быть одномерными одинаковой длины `n`, а массив `Z` – двумерным размером `n×n`. В этом случае поверхность изображается в виде сетки.

Пример.

Задание 6.1. Постройте график поверхности вида: $f(x,y) = 10e^{-0{,}1\sqrt{x^2+y^2}} \cos\left(\frac{\pi}{2}\sqrt{x^2+y^2}\right)$ для $x \in [-15, 15]$, $y \in [-5, 5]$.

Программа (результат выполнения программного кода приведен на рис. 6.4).

```
n=60;
% одномерный массив X длиной n со значениями
% от -15 до 15
   X = linspace(-15,15,n);
% одномерный массив Y длиной n со значениями
% от -5 до 5
   Y = linspace(-5,5,n);
for k=1:n
for p=1:n
% двумерный массив Z со значениями функции
```

```
% в каждой точке с координатами X(k), Y(p)
    Z(k,p) = f(X(k),Y(p));
end
end
surf(X,Y,Z) % построение поверхности

% функция f(x,y) записана в отдельном файле f.m
function z = f(x,y)
   r = sqrt(x^2+y^2); % вспомогательная переменная
   z=10*exp(-0.1*r)*cos(pi/2*r);
end
```

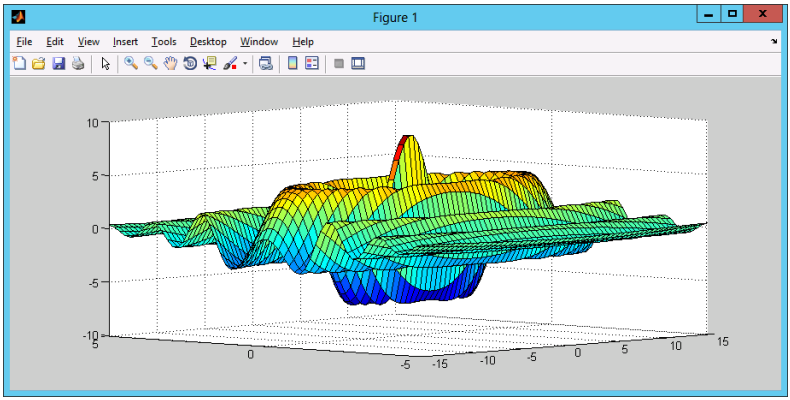


Рис. 6.4. Пример графика трехмерной поверхности, построенной с помощью функции `surf`

6.4. Анимация

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

Чтобы воспроизвести движение точки по заданной двумерной траектории, в MatLab используют стандартную функцию `comet(X,Y)`. Одномерные массивы `X` и `Y` одинаковой длины должны содержать последо-

вательность координат, через которые должна проходить точка в плоскости XY.

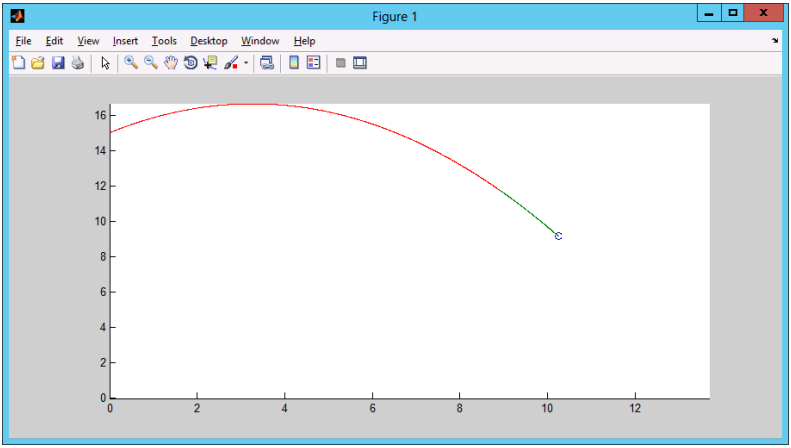


Рис. 6.5. Пример анимации движения тела, брошенного под углом к горизонту с некоторой высоты

Пример.

Задание 6.3. Анимируйте двумерное движение тела, брошенного под углом 45° к горизонту с высоты 15 м с начальной скоростью 8 м/с.

Программа (результат выполнения программного кода приведен на рис. 6.5).

```
% данные, необходимые для расчета координат
a=45*pi/180;                % угол в радианах
t=0; k=1; Y(1)=15; X(1)=0; % начальное положение
while Y(k)>0          % пока тело не упало на землю
   k=k+1; t=t+0.001;
   % вычисление координат тела
   [X(k),Y(k)]=koord(X(1),Y(1),a,t);
end
comet(X,Y)                  % анимация

% функция вычисления координат тела записана
% в отдельном файле с названием koord.m
```

```
function [x,y] = koord(x0,y0,a,t)
   x = x0+8*cos(a)*t;
   y = y0+8*sin(a)*t-9.8*t^2/2;
end
```

Функция `comet3` позволяет наблюдать движение точки в трехмерном пространстве. Для этого необходимо рассчитать по три координаты для каждого положения точки.

Для воспроизведения движения более сложных объектов необходимо создать последовательность кадров и воспроизвести их. В MatLab для этого используют следующие функции:

`movein(N)` – позволяет создать массив специального типа, в котором можно сохранить N кадров;

`getframe` – создает из текущего изображения кадр;

`movie(M)` – воспроизводит последовательность кадров, хранящихся в массиве `M` (каждый кадр хранится в отдельном столбце массива).

Пример.

Задание 6.4. Имитируйте смещение графика косинуса вдоль оси X, амплитуда которого последовательно то уменьшается, то увеличивается. Для этого создайте последовательность кадров, на каждом из которых с помощью функции `plot` изобразите графики функции $A\cdot\cos(0{,}5x+\varphi)$, отличающиеся значениями амплитуды A и начальной фазы φ. Изменение начальной фазы будет создавать иллюзию движения вдоль оси X. Амплитуду запишите, например, в виде $A = 10\cdot\cos\left(\pi\dfrac{n-1}{N}\right)$, а фазу – в виде $\varphi = \dfrac{n-1}{N-1}\pi$, где N – количество кадров, n – номер текущего кадра.

Программа.

```
N = 100;          % количество кадров
M = moviein(N);   % создает массив кадров
X = 0:0.1:6*pi;   % значение аргумента функции
for n=1:N
  % расчет данных и рисование одного кадра
    f =(n-1)*pi/(N-1);       % начальная фаза
    A = 10*cos((n-1)/N*pi); % амплитуда
    Y = A*cos(0.5*X+f);      % значения функции
```

```
  % создание изображения
    plot(X,Y);
    axis([0 6*pi -10 10]);
  % запоминание нарисованного кадра
  % в столбце массива
    M(:,n) = getframe;
end
% воспроизведение всех кадров
movie(M)
```

Кадры, используемые для анимации, при необходимости могут содержать и трехмерное изображение.

В MatLab есть еще несколько стандартных функций, используемых при создании кадров для анимации.

6.5. Настройка параметров графического окна и области отображения графика

6.5.1. Стандартные инструменты графического окна

Результаты работы стандартных функций построения и настройки графиков отображаются в графическом окне `figure` (см. рис. 6.1–6.5), которое имеет меню и панель инструментов (рис. 6.6) для детальной настройки параметров полученного изображения.

Пункты меню (рис. 6.6) графического окна `figure`:

- `File`: предоставляет возможность не только выполнять стандартные операции с файлами (`New`, `Open`, `Close`, `Save`, `Print`), но и генерировать программный код функции (**`Generate Code`**…), позволяющий программно воспроизвести все настройки, сделанные пользователем вручную;
- `Edit`: позволяет выделять, копировать, вырезать, удалять фрагменты размещенного в графическом окне изображения, а также осуществлять переход в режим отображения и редактирования свойств графического окна (`Figure Properties`…) или расположенной на нем области построения графика (`Axes Properties`…);

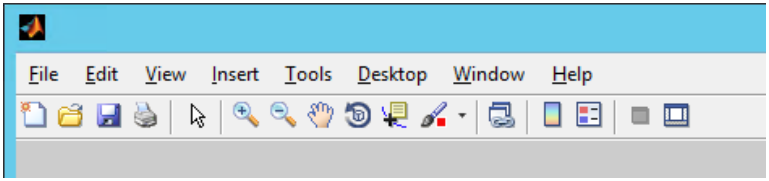


Рис. 6.6. Меню и панель инструментов графического окна

- `View`: позволяет управлять отображением следующих панелей инструментов: инструментов графического окна (`Figure Toolbar`), инструментов камеры (`Camera Toolbar`), инструментов редактирования графика (`Plot Edit Toolbar`), а также отображением редактора свойств (**`Property Editor`**) и др.;
- `Insert`: позволяет добавлять к имеющемуся изображению различные графические объекты (линии, прямоугольники, эллипсы, стрелки, текстовые заголовки и т.д.);
- `Tools`: позволяет начать или прекратить использование отдельных инструментов по настройке изображения;
- `Debug`: позволяет выполнить отладку программного кода, управляющего созданием редактируемого изображения;
- `Desktop`: позволяет закреплять (`Dock`) текущее графическое окно в основном окне среды MathLab;
- `Window`: если открыто несколько графических окон, позволяет осуществлять переходы между ними;
- `Help`: позволяет получать справочную информацию по отдельным графическим элементам.

Панель инструментов графического окна (рис. 6.6):

 позволяет создать новое графическое окно – New Figure;

 позволяет открыть файл – Open File;

 позволяет сохранить графическое окно – Save Figure;

 позволяет напечатать графическое окно – Print Figure;

 включает режим редактирования – Edit Plot;

 увеличение масштаба – Zoom In;

 уменьшение масштаба – Zoom Out;

 включает режим перемещения изображения – Pan;

 включает режим трехмерного вращения – Rotate 3D;

 включает режим отображения данных в позиции курсора – Data Cursor;

 включает режим выделения точек на графике выбранным цветом – Brush/Select Data;

 позволяет вставить цветовую шкалу – Insert Colorbar;

 позволяет вставить легенду – Insert Legend.

Эти и ряд других инструментов для редактирования графиков можно вызвать также с помощью пункта `Tools` в меню графического окна.

Для изменения свойств отдельных компонентов графика необходимо перейти в режим отображения редактора свойств, например, выбрав в меню окна `View` → `Property Editor`. Окно разделится на две части: одна содержит изображение графического окна, другая – свойства выделенного объекта. Выделяя однократным щелчком левой клавиши мыши объекты в графическом окне, можно отображать и редактировать их свойства.

Для двумерной кривой, построенной, например, с помощью функции `plot`, в пристыкованном внизу окне `Property Editor – Lineseries` доступны следующие свойства: тип графика (`Plot Type`), тип, толщина и цвет линии (`Line`), тип, размер и цвет маркеров (`Marker`) на кривой. Остальные свойства кривой доступны после нажатия на кнопку `More Properties...`

Для области построения графика в пристыкованом внизу окне `Property Editor – Axes` доступны следующие свойства (рис. 6.7): заголовок (`Title`), цвета фона и линий (`Colors`), сетка (`Grid`), шрифт (`Font`), а также для каждой из осей (`Axis`) координат: название (`Label`), границы (`Limits`), масштаб (`Scale`), расположение масштабных меток (`Ticks...`). Остальные свойства области построения графика доступны после нажатия на кнопку `More Properties...`

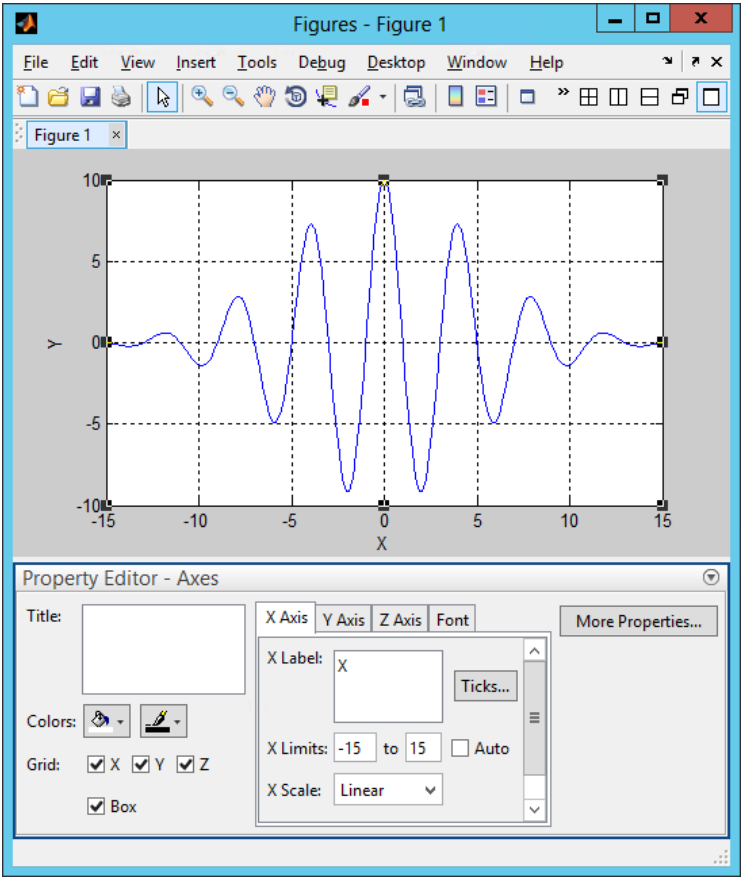


Рис. 6.7. Графическое окно в режиме редактирования свойств области построения графика

После изменения свойств графического окна или его отдельных элементов можно сгенерировать функцию, которая будет воспроизводить графическое окно вместе со всеми сделанными пользователем изменениями. Это избавляет от необходимости делать те же настройки при повторном выполнении программы. Для генерации программного кода можно использовать пункт `File` → `Generate Code` в меню графического окна.

6.5.2. Дескрипторная графика

При использовании для построения графиков стандартных функций автоматически создается определенная совокупность графических объектов. Количество и состав этих объектов зависят от используемых в программном коде стандартных графических функций. Чаще всего в процессе построения графиков создаются такие графические объекты, как `figure` (графическое окно), `axes` (область построения / расположения графика). Но в MatLab существуют и другие графические объекты, которые могут создаваться в процессе построения графиков:

- `figure` – графическое окно;
- `axes` – область построения / расположения графика;
- `line` – ломаная линия;
- `plot` – двумерная кривая;
- `surface` – трехмерная поверхность;
- `text` – текст;
- `rectangle` – прямоугольник;
- `light` – освещение;
- `uimenu` – меню;
- `uicontextmenu` – контекстное меню и др.

По существу, при вызове любой из перечисленных ранее стандартных функций, используемых для построения графиков, создаются соответствующие графические объекты.

Для каждого из этих объектов существует определенный набор команд, позволяющих управлять им. Например, `refresh` – обновить графическое окно; `clf` – очистить графическое окно; `cla` – очистить область построения графика.

Каждый из объектов имеет большое количество свойств. Если объект создается автоматически, то его свойства задаются по умолчанию и в процессе написания программного кода практически не регулируются. Если объект создается намеренно, для его идентификации в программе используется отдельная `<переменная>`, которая позволяет в дальнейшем обращаться к объекту и изменять его свойства в процессе выполнения программы. Для создания объектов используют название типа объекта (`figure`, `axes`, `line` и т.д.) следующим образом:

```
<переменная> = <тип объекта>;
```

Примеры:

```
f = figure; %  создание графического окна
```

```
h = axes;   % создание области построения графика
```

Список свойств объекта и их текущие значения можно вывести в командное окно MatLab с помощью команды `get`.

Пример:

```
f = figure;        %  создание графического окна
get(f)   % вывод свойств объекта f в командном окне
```

Использование этой команда позволяет получить информацию не только о доступных свойствах объекта и их значениях, но и о формате, в котором эти свойства необходимо задавать. Например, свойство `Position`, которое есть практически у всех графических объектов MatLab, необходимо задавать в виде одномерного массива-строки:

```
[x y width height],
```

где `x`, `y` – координаты нижнего левого угла объекта, `width` – ширина, `height` – высота объекта. Единицы измерения координат и размеров для разных объектов различны. Например, для графического окна они задаются в пикселах, для области построения графика – в долях от размера графического окна.

Свойства объектов можно задавать двумя способами: при создании объекта и с помощью команды `set`.

Чтобы каким-то свойствам объекта задать значения, отличные от значений по умолчанию, ***при создании объекта*** их перечисляют в круглых скобках через запятую после указания типа объекта как параметры:

```
<переменная> = <тип объекта>(<свойство>,
                <значение>,…);
```

Пример:

```
% создание графического окна размером 1200×800
  f = figure('Position', [300 50 1200 800]);
% создание области построения графика,
% занимающей 90% окна по горизонтали и по вертикали
  h = axes('Position', [0.06 0.06 0.9 0.9]);
```

Если при создании объекта необходимо задать значения нескольких свойств, они перечисляются через запятую в паре со своими значениями.

Пример задания двух свойств (`Position` и `Name` – заголовок окна) графического окна при его создании.

```
  f = figure('Position',[300 50 1200 800],
             'Name','Мой график');
```

Не перечисленные явно при создании объекта свойства остаются заданными по умолчанию.

Если настройка свойств объекта в программе происходит только при его создании (изменение свойств объекта в процессе дальнейшего выполнения программы не планируется), идентификатор объекта (переменную) можно опустить.

Пример:

```
figure('Position',[300 50 1200 800])
```

Если идентификатор объекта (переменная) явным образом при создании объекта не указан, MatLab сопоставляет такому объекту одну из вспомогательных переменных, но информацию об автоматически связанной с объектом переменной в командное окно не выводит. Для получения информации об этой переменной в таких случаях приходится использовать специальные команды. Таким образом, явное указание переменной для идентификации объекта при его создании упрощает дальнейшую разработку программы. Поэтому в последующих примерах переменные для идентификации объектов при их создании всегда будут указаны явно.

С помощью команды **set** можно изменять свойства ранее созданных объектов. В этом случае свойства объекта и их значения перечисляются как параметры функции `set` также попарно через запятую, но после `<идентификатора объекта>` (переменной, которой объект обозначен в программе):

`set(`**`<идентификатор объекта>`**`,<свойство>,<значение>,…)`

Без указания `<идентификатора объекта>` изменение его свойств с помощью функции `set` невозможно.

Пример задания названия шрифта (`FontName`) и его размера (`FontSize`) для области построения графика `axes` с помощью функции `set`:

```
h=axes;
. . . . .       % другой программный код
set(h,'FontSize', [14],'FontName' , 'Arial');
```

Свойства `XTick` и `YTick` области построения графика `axes` позволяют задавать расположение масштабных меток по осям X и Y соответственно. Например:

```
h=axes;
. . . . .       % другой программный код
set(h,'XTick',[-5 -2.5 0 2.5 5]);
% или set(h,'XTick',[-5:2.5:5]);
```

На созданных явно (не автоматически) объектах можно строить графики, в том числе с помощью стандартных функций (`plot` , `line` и т.д.). Если перед обращением к стандартной функции графическое окно и об-

ласть построения уже созданы, то график располагается на них. Если при обращении к стандартным функциям MatLab не находит графического окна и области построения, то он создает их автоматически.

Примеры:

```
f = figure('Name', 'Мой график');
plot(X,Y)
```

Пояснения: график строится в созданном ранее окне `f` с названием «Figure 1: Мой график»

```
plot(X,Y)
f = figure('Name', 'Мой график');
```

Пояснения: график строится в автоматически создаваемом окне с названием «Figure 1», а затем создается новое окно `f` с названием «Figure 2: Мой график».

Сочетание использования стандартных функций с программным созданием графических объектов позволяет сделать графическое представление результатов работы программы более эффективным.

Пример.

Задание 6.5. Постройте график функции $f = -x^2 + x + 4$ на интервале [–3, 5] в квадратном окне размером 500×500. Размер шрифта всех подписей на графике задайте равным 14.

Программа (результат выполнения программного кода приведен на рис. 6.8).

```
% создание окна размером 500x500
f=figure('Position',[300 50 500 500]);
% эквивалентная запись
%        figure('Position',[300 50 500 500]);
% так как в дальнейшем изменение свойств
% графического окна в программе не планируется
h=axes;      % создание области построения графика
set(h,'FontSize', [14]); % размер шрифта = 14
% эквивалентная запись двух предыдущих строк
%        h=axes('FontSize', [14]);
% или      axes('FontSize', [14]);
X = -3:0.01:5;     % аргументы функции
Y = f(X);          % значения функции f(x)
```

```
plot(X,Y)              % построение графика
title('График функции');       % заголовок

% функция f(x)= -x^2+x+4 записана в отдельном файле
% с названием f.m
function y = f(x)
   y=-x.^2+x+4;
end
```

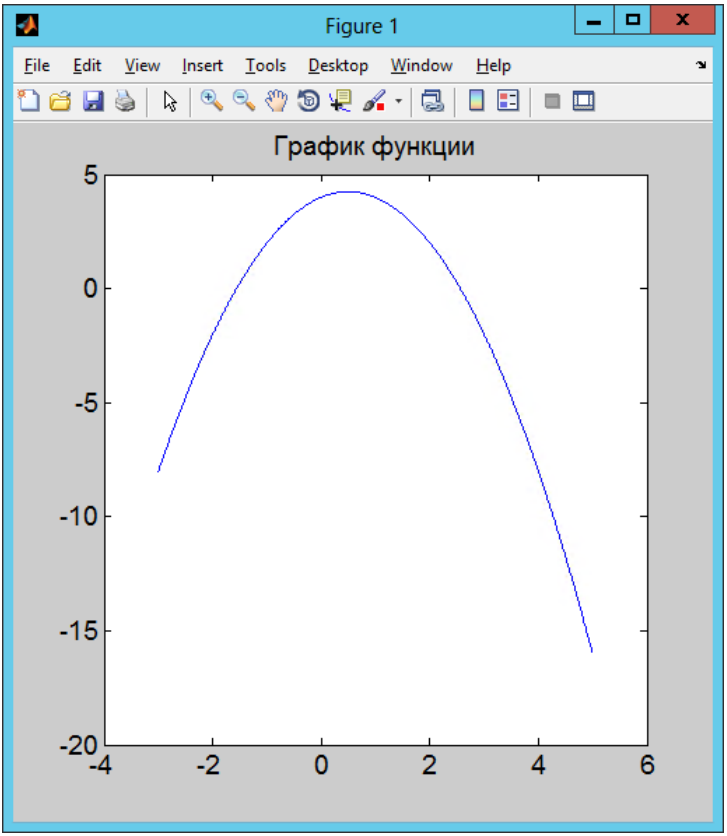


Рис. 6.8. Пример сочетания использования стандартных функций с программным созданием графических объектов

Аналогичным образом можно управлять свойствами остальных графических объектов, в том числе и `plot`, `line`, `legend` и т.д. Например, с помощью свойства `LineWidth` можно изменять толщину линии объекта `line` (ломаная).

Пример.

```
   z=line(X,Y);           % построение линии
set(z,'LineWidth',2.5)    % изменение толщины линии
% или z=line(X,Y,'LineWidth',2.5)
```

С помощью свойства `Color` можно изменять цвет кривой `plot` или линии `line` на графике. Стандартные цвета можно задавать с помощью буквенных обозначений, описанных в разделе 6.1. Остальные цвета задают в RGB формате в виде совокупности из трех чисел (от 0 до 1 каждое), соответствующих вкладу красного, зеленого и синего цветов (`[0 0 0]` – черный, `[1 1 1]` – белый, `[1 0 0]` – красный и т.д.).

Примеры:

```
   p=plot(X,Y);           % построение кривой
set(p,'Color','r')        % кривая красного цвета
% или
set(p,'Color',[1 0 0])    % тоже кривая красного цвета
   z=line(X,Y);           % построение линии
set(z,'Color','k')        % линия черного цвета
% или
set(z,'Color',[0 0 0])    % тоже линия черного цвета
% или z=line(X,Y,'Color',[0 0 0])
```

С помощью свойства `Location` можно изменять расположение легенды `legend`. Легенда может располагаться как внутри области построения графика, так и снаружи от нее. Свойство `Location` может принимать следующие значения: внутри – `North`, `NorthEast`, `East`, `SouthEast`, `South`, `SouthWest`, `West`, `NorthWest`; снаружи – `NorthOutside`, `NorthEastOutside`, `EastOutside`, `SouthEastOutside`, `SouthOutside`, `SouthWestOutside`, `WestOutside`, `NorthWestOutside`.

Пример:

```
d=legend('f1','f2','Location',' NorthEastOutside')
```

6.6. Визуально-ориентированное проектирование Windows-интерфейса

Среда MatLab позволяет создавать не только программы, выполняемые в текстовом режиме в командном окне, но и предоставляет средства разработки программ с Windows-интерфейсом (Graphic User Interface или сокращенно GUI). Такие программы обычно имеют прикладной характер и называются Windows-приложениями или GUI-приложениями.

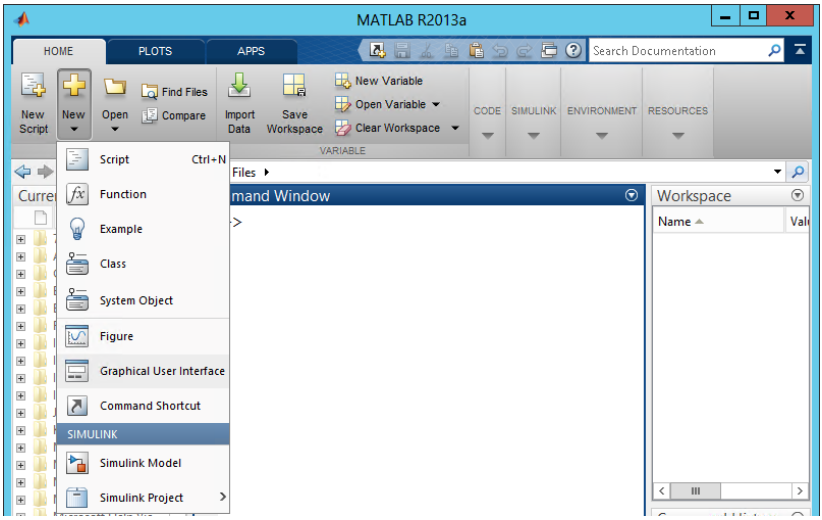


Рис. 6.9. Создание новой программы с Windows-интерфейсом из основного окна среды MatLab

6.6.1. Конструктор графического интерфейса

Чтобы приступить к созданию нового Windows-приложения, необходимо среди инструментов раскрывающегося списка кнопки New на панели инструментов основного окна среды Matlab (вкладка `Home`) выбрать `Graphic User Interface` (см. рис. 6.9).

В результате откроется окно конструктора GUIDE (GUI Designer). Но предварительно появится запрос, позволяющий выбрать тип создаваемого программного приложения (рис. 6.10) из следующего списка:

- `Blank GUI` – графическое (пустое) окно;
- `GUI with Uicontrols` – окно с кнопками и переключателями;
- `GUI with Axes and Menu` – окно с кнопкой, раскрывающимся списком, меню и областью построения графика;
- `Modal Question Dialog` – модальное диалоговое окно.

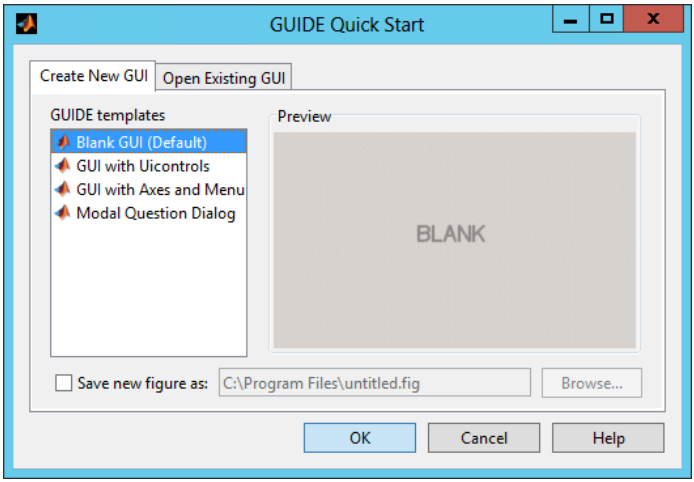


Рис. 6.10. Выбор типа создаваемого GUI-приложения

В результате выбора `Blank GUI`, предлагаемого по умолчанию, в открывшемся конструкторе GUIDE автоматически создается только графическое окно (рис. 6.11). Этот тип GUI-приложения включает минимальное количество автоматически создаваемых конструктором объектов (один). Остальные объекты разработчик приложения должен добавлять и настраивать самостоятельно. В остальных случаях кроме графического окна автоматически создается еще несколько объектов.

Окно конструктора GUIDE состоит из меню, двух панелей инструментов (вертикальной и горизонтальной) и рабочей области, в которой и происходит разработка интерфейса приложения.

Пункты меню (рис. 6.11) конструктора GUIDE:

- `File`: позволяет выполнять стандартные операции с файлами (`New`, `Open`, `Close`, `Save`, `Print`);
- `Edit`: позволяет выделять, копировать, дублировать, вырезать, удалять выделенные объекты, размещенные в графическом окне;

- `View`: позволяет управлять отображением в окне панелей инструментов, а также осуществлять переход к инспектору свойств (`Property Inspector`), обзору объектов (`Object Browser`), редактированию программного кода (`Editor`) и редактированию обработки событий (`View Callbacks`);
- `Layout`: позволяет управлять отображением вспомогательной сетки в графическом окне (`Snap to Grid`), а также перемещением отдельных объектов на задний (`Back`) или передний план (`Front`);
- `Tools`: содержит вспомогательные инструменты настройки объектов, а также настройки конструктора GUIDE;
- `Help`: позволяет получать справочную информацию по использованию конструктора GUIDE для создания приложений.

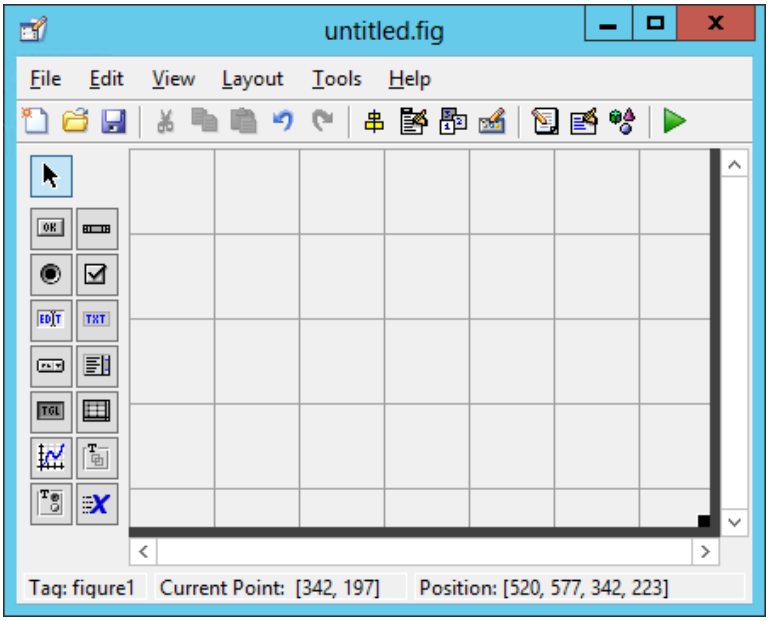


Рис. 6.11. Окно конструктора GUIDE с заготовкой приложения типа Blank GUI

Горизонтальная панель инструментов конструктора GUIDE (см. рис. 6.11):

 позволяет создать новое приложение (графическое окно) – New Figure;

 позволяет открыть файл сохраненного ранее приложения – Open Figure;

 позволяет сохранить приложение – Save Figure;

 позволяет вырезать объект – Cut;

 позволяет копировать объект – Copy;

 позволяет вставить объект – Paste;

 позволяет отменить последнюю операцию – Undo;

 позволяет восстановить отмененную операцию – Redo;

 позволяет выполнить выравнивание объектов – Align Objects;

 редактор меню для приложения – Menu Editor;

 редактор последовательности переходов между объектами – Tab Order Editor;

 редактор панели инструментов – Toolbar Editor;

 **редактор программного кода – Editor**;

 **инспектор свойств – Property Inspector**;

 обзор объектов – Object Browser;

 **запуск приложения на выполнение – Run**.

Некоторые из этих инструментов можно вызвать также с помощью пункта `Tools` в меню конструктора GUIDE.

6.6.2. Добавление объектов

Вертикальная панель инструментов конструктора GUIDE (см. рис. 6.11) позволяет добавлять к разрабатываемому приложению следующие объекты:

 режим выделения объектов – Select;

 **обычная кнопка – Push Button;**

 радиокнопка – Radio Button;

 **поле для ввода и редактирования текста – Edit Text;**

 открывающийся список – Pop-up Menu;

 кнопка – Toggle Button;

 **область построения графика – Axes;**

 область для размещения кнопок – Button Group;

 полоса прокрутки – Slider;

 переключатель – Check Box;

 **нередактируемый текст – Static Text;**

 список – Listbox;

 таблица – Table;

 область для размещения объектов – Panel;

 ActiveX Control.

Чтобы добавить к разрабатываемому приложению объект, необходимо сначала выбрать соответствующую кнопку на вертикальной панели инструментов в конструкторе GUIDE, а затем выполнить однократный щелчок левой клавишей манипулятора мышь в той части графического окна, где необходимо разместить объект. На поверхности окна появится выбранный объект, со всех сторон выделенный небольшими квадратными маркерами (рис. 6.12). Наличие маркеров вокруг объекта говорит о том, что все доступные в данный момент операции будут применяться к этому объекту (объект считается выделенным).

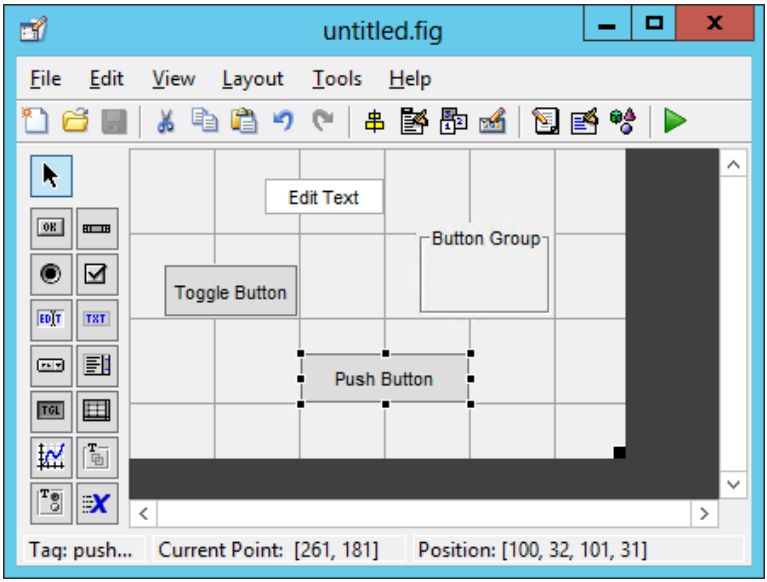


Рис. 6.12. Размещение объектов в графическом окне с помощью конструктора GUIDE

Изменение расположения объектов в графическом окне осуществляется при помощи манипулятора мышь (путем перетаскивания). Изменение размеров объектов производится путем перетаскивания в нужном направлении некоторых маркеров, выделяющих объект.

Для изменения других свойств объектов необходимо использовать инспектор свойств (`Property Inspector`).

6.6.3. Редактирование свойств объектов

Для получения доступа ко всем свойствам выбранного объекта необходимо использовать инспектор свойств. Открыть инспектор свойств можно следующим образом:

1. Выполнив двойной щелчок левой клавишей мыши на изображении объекта в окне редактируемого приложения.
2. Выбрав в меню конструктора GUIDE пункт `View` → `Property Inspector`.

3. Нажав на кнопку `Property Inspector` на горизонтальной панели инструментов конструктора GUIDE.

В результате любого из этих действий появится окно, в котором в виде таблицы будут перечислены все свойства и их текущие значения для выбранного объекта (в левом столбце отображаются свойства, а в правом – значения). Выделив значение одного из свойств, его можно отредактировать, соблюдая доступный для данного свойства тип данных.

Разные объекты имеют разный набор свойств. Но можно выделить несколько свойств, которые встречаются у всех объектов:

- `Tag` – идентификатор объекта (имя переменной);
- `Style` – тип объекта;
- `Position` – координаты и размеры объекта;
- `String` – текст, отображаемый на объекте;
- `BackgroundColor` – цвет фона объекта;
- `FontName` – название шрифта;
- `FontSize` – размер шрифта;
- `Value` – текущее значение;
- `Min` – минимальное значение;
- `Max` – максимальное значение.

Например, текст, написанный на кнопке (`Push Button`), хранится в ее свойстве `String`. Положение и размер кнопки можно изменять как с помощью мыши (как было описано выше), так и изменяя значения свойства `Position`. Эти же свойства можно изменять программно (путем написания программного кода) при обработке событий с различными объектами.

6.6.4. Программная обработка событий с объектами

Каждому объекту в GUI-приложении соответствует не только определенный набор свойств, но и определенный набор событий, которые могут происходить с этим объектом.

Список доступных для выделенного в конструкторе GUIDE объекта событий можно увидеть, выбрав в меню конструктора пункт `View` → `View CallBacks` или пункт `View CallBacks` в контекстном меню объекта, которое появляется после однократного щелчка правой клавишей мыши на объекте. Кроме того, все доступные для выделенного объекта события перечислены в таблице инспектора свойств (их названия заканчиваются на `...Fcn` и сопровождаются пиктограммой ).

Основные события, доступные для большинства объектов в конструкторе GUIDE:

- `Callback` – основная функция объекта (зависит от типа объекта);
- `KeyPressFcn` – нажата клавиша на клавиатуре;
- `CreateFcn` – создание объекта;
- `DeleteFcn` – удаление объекта;
- `OpenFcn` – открытие графического окна (только для объекта `figure`).

При первом сохранении разрабатываемого GUI-приложения автоматически генерируется программный код, описывающий создание и расположение добавленных к приложению в конструкторе объектов. Программный код обновляется при каждом сохранении приложения. Информация хранится в виде двух файлов: m-файл содержит программный код, а fig-файл – визуальное представление графического окна вместе со всеми расположенными на нем объектами. Однако действия, которые должна выполнять программа в случае наступления того или иного события с объектами, разработчик приложения должен написать сам.

Чтобы перейти к написанию или редактированию программного кода, который должен выполняться при наступлении какого-либо события, необходимо выбрать соответствующее событие в инспекторе свойств или в раскрывающемся списке `View CallBacks` контекстного меню объекта. При этом среда MatLab автоматически находит в связанном с приложением m-файле подфункцию, описывающую данное событие. Если такая подфункция отсутствует, то в m-файле автоматически создается заготовка данной подфункции.

Например, для события «нажатие на кнопку», соответствующего основной функции (`Callback`) объекта `Push Button`, имеющего в качестве идентификатора (`Tag`) значение `pushbutton1`, создается заготовка подфункции:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject,
                        eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined
%              in a future version of MATLAB
% handles    structure with handles and user data
%                                 (see GUIDATA)
```

Пояснения. Заготовка содержит только заголовок подфункции со списком исходных параметров. Остальной текст – комментарии, поясняющие назначение подфункции (`button press` – нажатие на кнопку) и смысл исходных параметров. В данной подфункции идентификатором `hObject` обозначена сама кнопка `pushbutton1`, с которой связано рассматриваемое событие. А параметр `handles` содержит информацию о структуре всех объектов в разрабатываемом приложении и позволяет обращаться к свойствам других объектов приложения. Параметр `eventdata` зарезервирован для использования в дальнейших версиях MatLab.

Заготовки подфункций для других событий имеют аналогичную структуру, но могут отличаться друг от друга в зависимости от типа объекта.

Заготовки для основной функции `Callback` каждого из объектов, используемых в приложении, создаются в m-файле автоматически.

Чтобы в программе задать свойство какого-либо объекта, используют функцию `set` в соответствии с правилами, описанными в предыдущем разделе.

`set(`**`<идентификатор объекта>`**`,<свойство>,<значение>,…)`

Если необходимо задать свойство того объекта, с которым связано событие редактируемой в данный момент подфункции, то в качестве идентификатора используют параметр `hObject`.

Например, чтобы после нажатия на кнопку `pushbutton1` на ней было написано слово «Выполнено», нужно задать / изменить ее свойство `String` в подфункции `pushbutton1_Callback`.

```
function pushbutton1_Callback(hObject,
                        eventdata, handles)
set(hObject, 'String', 'Выполнено')
```

Однако такие случаи встречаются редко. Чаще всего при нажатии на кнопку необходимо изменять свойства других объектов. К их идентификаторам можно обратиться через структуру `handles`.

Например, необходимо, чтобы после нажатия на кнопку слово «Выполнено» было написано не на самой кнопке, а на объекте `Static Text` с идентификатором `text1`, который был предварительно размещен на графическом окне в конструкторе GUIDE. Тогда в подфункции `pushbutton1_Callback` в качестве `<идентификатора объекта>` для функции `set` следует указать `handles.text1`.

```
function pushbutton1_Callback(hObject,
                        eventdata, handles)
set(handles.text1, 'String', 'Выполнено')
```

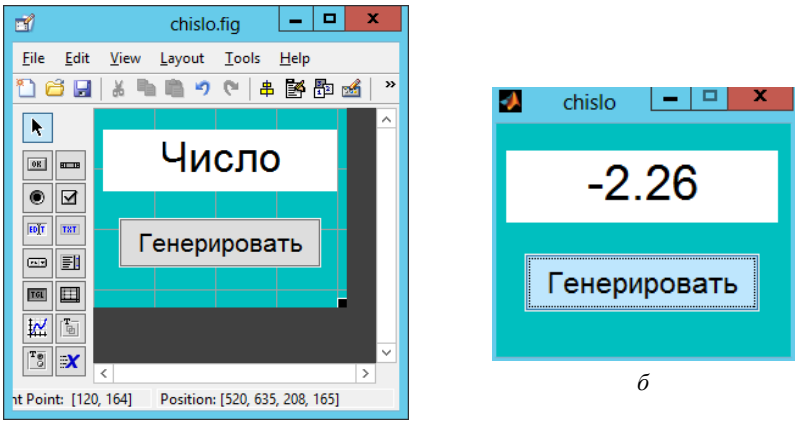
Пример.

Задание 6.6. Разработайте Windows-приложение (программу), в которой при каждом нажатии на кнопку генерируется новое случайное число.

Программа (результат визуального конструирования приложения приведен на рис. 6.13, *а*, а результат выполнения программного кода – на рис. 6.13, *б*).

```
function pushbutton1_Callback(hObject,
                              eventdata, handles)
x = randn;  % генерируется случайное число
% преобразование числа в строковый формат
s = num2str(x, '%6.2f');
% изменение надписи на объекте text1
set(handles.text1, 'String', s)
```

Пояснения. Для улучшения читаемости программный код приводится без стандартных комментариев, создаваемых автоматически в процессе первоначальной генерации заготовки программного кода подфункции. Эти комментарии рекомендуется не удалять из программы.



а *б*

Рис. 6.13. Пример изменения свойств одного объекта, когда событие происходит с другим

Прежде чем писать приведенный выше программный код, необходимо в конструкторе GUIDE (рис. 6.13, *а*) разместить на графическом окне два объекта: кнопку `Push Button` и текстовое поле `Static Text`, настроить размеры и расположение этих объектов. Далее в инспекторе свойств `Property Inspector` задать цвет (`Color`, `BackgroundColor`) окна и текстового поля, а также размер шрифта (`FontSize`) и текст (`String`), отображаемый на кнопке и в текстовом поле. После сохранения приложения происходит автоматический переход в редактор программного кода для его написания в подфункции `pushbutton1_Callback`. После запуска приложения и одного (или нескольких) нажатия на кнопку окно приложения примет вид, представленный на рис. 6.13, *б*.

Размещение в графическом окне объекта `Axes` (область построения графика) позволяет строить графики, используя стандартные функции (`plot`, `axis`, `grid` и т.д.), описанные разделах 6.1–6.4. Кроме того, наличие объекта `Axes` в графическом окне позволяет программисту использовать команды управления этим объектом, например, команду `cla` для очистки области построения графика (см. раздел 6.5).

Пример.

Задание 6.7. Разработайте Windows-приложение (программу), в котором при нажатии на одну кнопку область построения графика очищается, а при нажатии на другую – строится график функции $f(x) = -2 + 2\sin 3x + 0{,}1x^2$ на интервале [–5, 8].

Программа (результат визуального конструирования приложения приведен на рис. 6.14, *а*, а результат выполнения программного кода подфункции `pushbutton1_Callback` – на рис. 6.14, *б*).

```
% Нажатие на кнопку «Построить»
function pushbutton1_Callback(hObject,
                              eventdata, handles)
   % вычисление аргументов функции
   X=-5:0.05:8;
   % вычисление значений функции f
   Y=f(X);
   % построение графика точками
   plot(X,Y,'.')
   % диапазон координат по осям
   axis([min(X) max(X) min(Y) max(Y)])
   % нанесение сетки
   grid on
```

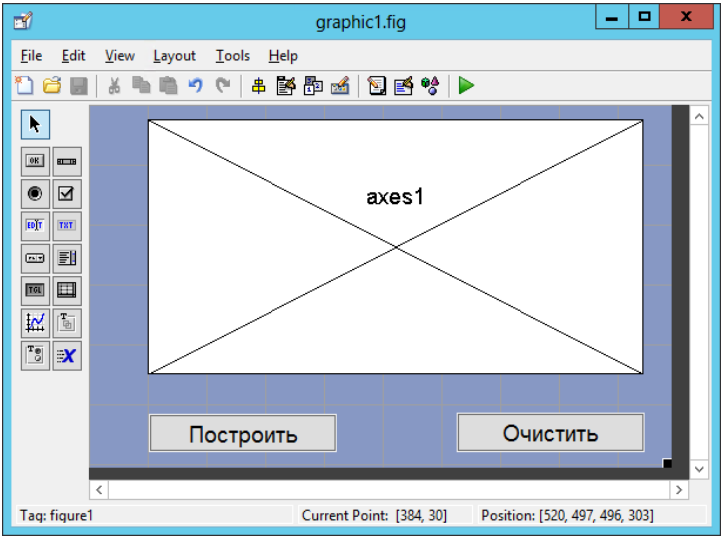
```
% Нажатие на кнопку «Очистить»
function pushbutton2_Callback(hObject,
                              eventdata, handles)
   % диапазон координат по осям по умолчанию
   axis([0 1 0 1])
   % удаление сетки
   grid off
   % очистка области построения графика
   cla

% функция f(x)= -2+2sin(3x)+0.1x^2
function y = f(x)
   y=-2+2*sin(3*x)+0.1*x.^2;
```

Пояснения. Для улучшения читаемости программный код приводится без стандартных комментариев, создаваемых автоматически в процессе первоначальной генерации заготовки программного кода подфункции. Эти комментарии рекомендуется не удалять из программы.

Программный код вычисления значения функции `f(x)` может быть оформлен либо как подфункция в одном файле с подфункциями обработки событий, либо как m-функция в отдельном файле с названием f.m.

Прежде чем писать приведенный выше программный код, необходимо в конструкторе GUIDE (рис. 6.14, *а*) разместить на графическом окне три объекта: две кнопки `Push Button` и область построения графика `Axes`, настроить размеры и расположение этих объектов. Далее в инспекторе свойств `Property Inspector` задать цвет (`Color`) и название (`Name`) окна, а также размер шрифта (`FontSize`) и текст (`String`), отображаемый на кнопках. После сохранения приложения происходит автоматический переход в редактор программного кода для его написания в подфункциях `pushbutton1_Callback` и `pushbutton2_Callback`. После запуска приложения и нажатия на кнопку «Построить» окно приложения примет вид, представленный на рис. 6.14, *б*. После нажатия на кнопку «Очистить» кривая и сетка в области построения графика исчезнут.



а



б

Рис. 6.14. Пример использования стандартных функций построения графиков в GUI-приложении

Среди объектов конструктора GUIDE есть объекты, позволяющие пользователю вводить / изменять информацию. Например, текстовое поле Edit Text. Введенная пользователем информация хранится в свойстве String этого объекта. Чтобы получить эту информацию, используют функцию get:

```
<переменная> = get(<идентификатор объекта>,
                                        <свойство>);
```

<Идентификатор объекта> указывает, к свойству какого объекта обращается программист. Название <свойства> записывается в строковом виде, например 'String'. Полученное в результате применения функции get значение свойства объекта помещается в <переменную>. Это обеспечивает возможность его дальнейшего использования.

Если обращение к свойству объекта выполняется в подфункции, связанной с событием этого же объекта, то в качестве <идентификатора объекта> используют hObject. Обращение к свойству объекта из подфункции, связанной с событием, произошедшим с другим объектом, выполняется через структуру handles. Например, чтобы в подфункции нажатия на кнопку получить значение свойства 'String' текстового поля Edit Text, имеющего идентификатор edit1, необходимо написать

```
s = get(handles.edit1, 'String');
```

В результате переменной s будет присвоено значение, хранящееся в свойстве String объекта edit1. Полученное значение может быть использовано для изменения свойств других объектов, расположенных в графическом окне.

Это позволяет неоднократно вводить новые данные и получать новые результаты, не прибегая к повторному запуску приложения.

Пример.

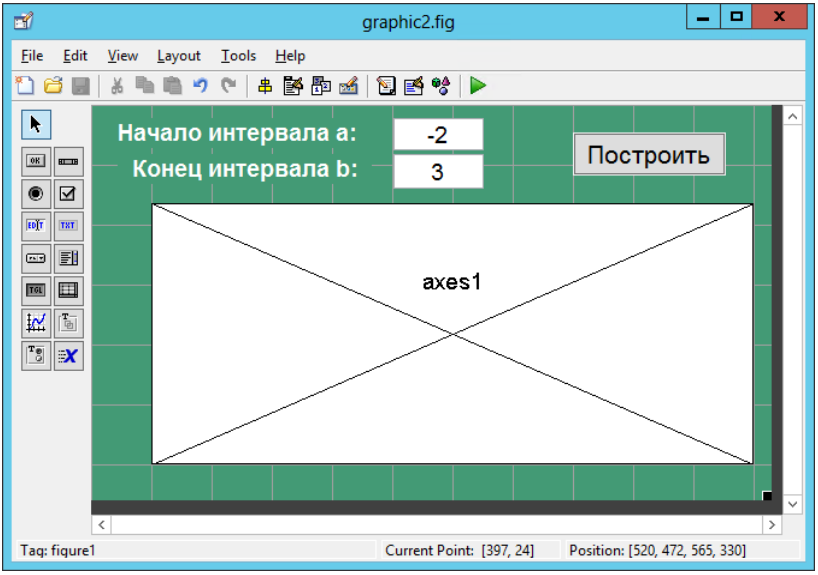
Задание 6.8. Разработайте Windows-приложение (программу), в котором график функции $f(x) = -2 + 2 \sin 3x + 0{,}1x^2$ строится на интервале, введенном пользователем.

Программа (результат визуального конструирования приложения приведен на рис. 6.15, *а*, а результат выполнения программного кода – на рис. 6.15, *б*).

```
% Нажатие на кнопку «Построить»
function   pushbutton1_Callback(hObject,   eventdata,
handles)
% в переменную a присваивается значение,
% введенное пользователем в поле объекта edit1
% в строковом формате
   a=get(handles.edit1,'String');
% преобразование значения переменной a
% из строкового формата в числовой
   a=str2num(a);
% аналогично для переменной b
   b=get(handles.edit2,'String');
   b=str2num(b);
% a – начало, b – конец интервала,
% на котором строится график функции
   dx=(b-a)/499;
   for k=1:500
     % вычисление аргументов функции
     X(k)=a+(k-1)*dx;
% вычисление значений функции f(x) для каждого X(k)
     Y(k)= f(X(k));
   end
% построение графика точками красного цвета
   p=plot(X,Y,'.r')
% диапазон координат по осям
   axis([min(X) max(X) min(Y) max(Y)])

% функция f(x)= -2+2sin(3x)+0.1x^2
function y = f(x)
   y=-2+2*sin(3*x)+0.1*x^2;
```

Пояснения. Для улучшения читаемости программный код приводится без стандартных комментариев, создаваемых автоматически в процессе первоначальной генерации заготовки программного кода подфункции. Эти комментарии рекомендуется не удалять из программы.



а



б

Рис. 6.15. Пример построения графика функции $f(x)= -2+2\sin(3x)+0,1x^2$ на заданном пользователем интервале

Программный код вычисления значения функции `f(x)` может быть оформлен либо как подфункция в одном файле с подфункциями обработки событий, либо как m-функция в отдельном файле с названием f.m.

Прежде чем писать приведенный выше программный код, необходимо в конструкторе GUIDE (рис. 6.15, *а*) разместить на графическом окне шесть объектов: два редактируемых поля `Edit Text`, два нередактируемых текста `Static Text`, кнопку `Push Button` и область построения графика `Axes`, настроить размеры и расположение этих объектов. Далее в инспекторе свойств `Property Inspector` задать цвет (`Color`) и название (`Name`) окна, а также размер шрифта (`FontSize`) и текст (`String`), отображаемый в текстовых полях и на кнопке по умолчанию. После сохранения приложения происходит автоматический переход в редактор программного кода для его написания в подфункции `pushbutton1_Callback`. После запуска приложения, ввода границ интервала и нажатия на кнопку «Построить» окно приложения примет вид, представленный на рис. 6.15, *б*. Не закрывая окно приложения, границы интервала можно изменить и вновь нажать на кнопку «Построить».

6.6.5. Диалоговые окна

Для дополнительного взаимодействия с пользователем в процессе выполнения GUI-приложения в MatLab предусмотрено большое количество диалоговых окон общего и специального назначения:

- `dialog` – окно общего назначения;
- **`msgbox` – окно сообщения**;
- `printdlg` – окно печати;
- `uigetfile` – выбор файла для чтения;
- `uiputfile` – выбор файла для записи (сохранения);
- `uisetcolor` – выбор цвета;
- `uisetfont` – выбор шрифта;
- `waitbar` – индикация процесса.

Для вывода сообщений используется стандартное окно `msgbox`:

```
msgbox(<сообщение>, <заголовок окна>)
```

Пример:

```
msgbox ('Недопустимое значение', 'Ошибка')
```

Для предоставления пользователю в процессе выполнения GUI-приложения возможности выбора цвета используется стандартное окно `uisetcolor` (рис. 6.16):

```
<имя переменной>=uisetcolor(<текущий цвет>,
                    <заголовок окна>)
```

В результате `<переменной>` присваивается выбранный пользователем в стандартном окне цвет. Текщий и выбранный пользователем цвет задаются в формате RGB в виде совокупности трех цифр (от 0 до 1 каждая), соответствующих вкладу красного, зеленого и синего цветов. Например, `[1 1 1]` – белый, `[0 0 0]` – черный.

Пример:

```
newColor=uisetcolor([1 1 1], 'Цвет графика');
```

Кроме перечисленных окна `msgbox` и `uisetcolor` имеют ряд дополнительных параметров (свойств), значения которых задаются по умолчанию, если они не указаны при вызове окна.

Аналогично организуется вызов других диалоговых окон.



Рис. 6.16. Диалоговое
окно выбора цвета

Использование диалоговых окон позволяет организовать ввод и вывод более разнообразной по типу (не только числа и строки, как в командном

окне) информации. Например, выбранный пользователем в стандартном окне цвет можно использовать для построения графика `plot` или линии `line`:

```
newColor=uisetcolor([1 1 1], 'Цвет графика');
plot(X,Y,'Color',newColor)
line(X,Y,'Color',newColor)
```

или для изменения цвета графика:

```
p=plot(X,Y)
. . . . .
newColor=uisetcolor([1 1 1], 'Цвет графика');
set(p,'Color',newColor)
```

Глава 7. ЧТЕНИЕ ДАННЫХ ИЗ ФАЙЛА. ЗАПИСЬ ДАННЫХ В ФАЙЛ

Под файлом понимают поименованную область внешней (например, дисковой) памяти компьютера.

В процессе обработки данных внешние файлы, расположенные на жестком диске компьютера, могут использоваться как для хранения исходных данных, так и для хранения результатов работы программы. Рассмотрим работу с файлами на примере текстовых (форматированных) файлов, расположенных на жестком диске компьютера (внешние файлы). Текстовые файлы могут содержать не только буквы, числовые значения целых и вещественных величин, но и любые символы псевдографики.

7.1. Обращение к файлу

Перед началом взаимодействия с конкретным внешним файлом необходимо ***открыть файл для чтения или записи*** информации. Для этого используют стандартную функцию `fopen`, которая имеет два параметра: 1) `<имя файла>` – название файла с расширением и указанием пути к файлу (если файл находится в той же папке, что и использующая его программа, путь к файлу можно не указывать); 2) `<спецификация>` указывает, с какой целью открывается файл.

Возможные значения <спецификации> для функции fopen

<Спецификация>	Описание
`'r'`	Открыть файл для чтения
`'r+'`	Открыть файл для чтения и записи
`'w'`	Открыть файл для записи (удалить содержимое существующего файла или создать новый)
`'a'`	Открыть файл для добавления информации

Для идентификации файла в программе необходимо использовать отдельную `<переменную>`, которая позволит в дальнейшем обращаться к связанному с ней файлу. Поэтому для корректного открытия файла функцию `fopen` рекомендуется использовать в виде

```
<переменная> = fopen (<имя файла>, <спецификация>);
```

Примеры:

```
% файл example1.dat открыт на чтение
FL = fopen('example1.dat', 'r');
% файл variant3.txt открыт для записи
F = fopen('variant3.txt', 'w');
```

Переменные `FL` и `F` используются в дальнейшем в программе для обращения к файлам `example1.dat` и `variant3.txt` соответственно.

Если необходимо задать расположение файла на жестком диске, используют полное имя файла, содержащее последовательно название диска, каталогов, в которых расположен файл, и имя файла с расширением.

Пример:

```
FL = fopen('c:\examples\example1.dat', 'r');
 F = fopen('d:\work\data\variant3.txt', 'w');
```

При использовании функции `fopen` со спецификацией `'r'` или `'w'` указатель файла устанавливается на его начало. При использовании функции `fopen` со спецификацией `'a'` указатель файла устанавливается на его конец.

При использовании функции `fopen` имя файла можно указывать как непосредственно на месте первого параметра, так и с помощью переменной, содержащей соответствующую информацию в строковом формате.

Примеры указания имени файла с помощью дополнительной переменной:

```
file_name = 'tema.txt';  FL = fopen(file_name, 'r');
Sfile = 'd:\work\var5.txt';   F = fopen(Sfile, 'w');
```

Когда необходимо использовать несколько файлов из одной папки, путь к папке бывает удобно представить в виде отдельной переменной и в дальнейшем объединять его с именами отдельных файлов с помощью оператора объединения `[]`:

```
Path= 'd:\work\';
F_dat = fopen([Path 'variant4.txt'], 'r');
F_res = fopen([Path 'result4.txt'], 'w');
```

После окончания работы с файлом необходимо ***закрыть файл*** с помощью функции `fclose`, параметром которой является `<переменная>` – идентификатор открытого ранее файла. Если в программе использовалось обращение к нескольким внешним файлам, необходимо закрыть каждый из них.

Примеры:

```
fclose (F_dat);
```

```
fclose (F);
fclose (F_res);
```

Своевременное закрытие файла с помощью процедуры `fclose` позволяет обеспечить сохранность информации в файле.

7.2. Чтение данных из файла

Ввод (чтение) данных из файла осуществляется с помощью функций `fgetl` и `fscanf`.

Функция `fgetl` читает информацию, хранящуюся в текущей строке файла, и возвращает ее в программу в строковой форме. Формат вызова функции `fgetl`:

```
<переменная> = fgetl(<идентификатор файла>);
```

Функция `fgetl` имеет один единственный параметр – переменную, используемую в программе как `<идентификатор файла>`, указывающий из какого файла необходимо читать информацию. Полученная из файла информация помещается в `<переменную>`, указанную слева от функции `fgetl` перед оператором присвоения.

Пример:

```
titul1 = fgetl(F_dat);
```

Функция `fscanf` позволяет читать информацию, хранящуюся в нескольких строках начиная с текущей, и возвращает ее в программу в виде двумерного массива. Причем чтение происходит по строкам, а массив формируется по столбцам (каждой строке в файле будет соответствовать столбец массива). Функция `fscanf` имеет три параметра и вызывается следующим образом:

```
<переменная> = fscanf(<идентификатор файла>,
                      <формат>, <размер>);
```

`<Идентификатор файла>` – полученная в результате выполнения функции `fopen` переменная указывает, из какого файла необходимо читать данные.

`<Размер>` указывает количество считываемых данных и может быть записан одним числом `n` (из данных формируется одномерный массив – столбец указанной длины) или в двумя числами `[m, n]` (из данных формируется двумерный массив размером `m×n`). Если `<размер>` не указан, происходит чтение всех данных до конца файла, которые преобразуются в один массив.

Параметр <формат> указывает, к какому типу нужно преобразовать полученные из файла данные, которые помещаются в <переменную>, указанную слева от функции fscanf перед оператором присвоения.

Возможные значения параметра <формат> для функции fscanf

<Формат>	Описание
'%c'	Последовательность символов
'%d'	Двоичное число
'%e', '%g'	Число с плавающей запятой
'%f'	Число с фиксированной запятой
'%i'	Целое число со знаком
'%o'	Восьмеричное число со знаком
'%s'	Последовательность непробельных символов
'%u'	Десятичное число со знаком
'%x'	Шестнадцатеричное число со знаком

Пример:

```
X = fscanf(F_dat,'%f',[2,3]);
```

Пояснения. В этом случае программа читает последовательно 6 вещественных чисел и располагает их по два в каждом столбце массива X. Чтение начинается с текущей позиции указателя файла.

Выбор операторов fgetl или fscanf, а также их последовательность и параметры определяются структурой данных в файле.

Пример ввода (чтения) элементов одномерного массива из файла.

Задание 7.1. Значения одномерного вещественного массива A длиной 8 введите (прочитайте) из файла d:\work\mas1.txt

Файл d:\work\mas1.txt

```
  4.78  5.34  8.91 10.14 17.77 13.24 11.15  9.67
```

Программа

```
F = fopen ('d:\work\mas1.txt','r');
   A = fscanf(F,'%f'); % читает все числа
fclose (F);
```

Пояснения. Результат: A – одномерный массив (столбец).

Задание 7.2. Значения одномерного вещественного массива A длиной 10 введите (прочитайте) из файла d:\work\mas2.txt

Файл d:\work\mas2.txt	Программа
4.78 5.34 8.91 10.14 17.77 13.24 11.15 9.67 5.08 3.11	F=fopen('d:\work\mas2.txt','r'); % читает все числа A = fscanf(F,'%f'); fclose (F);

Пояснения. Результат: A – одномерный массив (строка).

Пример ввода (чтения) элементов двумерного массива из файла.

Задание 7.3. Значения двумерного вещественного массива A размером 4×3 введите (прочитайте) из файла d:\work\mas3.txt

Файл d:\work\mas3.txt

```
     4.78    10.14    11.15     3.11
     5.34    17.77     9.67     7.32
     8.91    13.24     5.08    10.01
```

Программа.

```
F = fopen ('d:\work\mas3.txt','r');
   A = fscanf(F,'%f',[4,3]); % читает 12 чисел
fclose (F);
```

Пояснения. Данные, находящиеся в строке файла, помещаются в столбец массива. В массиве A получится 4 строки и 3 столбца.

Пример ввода (чтения) значений скалярных переменных из файла.

Задание 7.4. Введите (прочитайте) значения переменных Titul (строковая), x, y, z (вещественные) и nx, ny, nz (целые) из файла d:\work\data.txt

Файл d:\work\data.txt

```
   Координаты точки
   4.78   5.34   8.91
    20     55     17
   количество ячеек
```

Программа

```
FL = fopen('d:\work\data.txt','r');
    Titul = fgetl(FL)          % читает строку
    A = fscanf(FL,'%f',3);     % читает три числа
       x=A(1,1)          % помещает в x,y,z
       y=A(2,1),  z=A(3,1)
    A = fscanf(FL,'%i',3);     % читает три числа
       nx=A(1,1)          % помещает в nx,ny,nz
       ny=A(2,1),  nz=A(3,1)
fclose (FL);
```

7.3. Запись данных в файл

Вывод данных в файл осуществляется с помощью функции fprintf, которую вызывают следующим образом:

fprintf(<идентификатор файла>, <формат>, <переменная>)

<Идентификатор файла>, полученный в результате выполнения функции fopen, указывает, в какой файл необходимо записывать данные.

В качестве <переменной> могут быть указаны одна или несколько вычисленных в программе переменных. Если переменных несколько, они перечисляются через запятую. Данные из двумерных массивов берутся по столбцам, а запись в файл происходит по строкам (каждому столбцу массива в файле будет соответствовать строка).

Параметр <формат> указывает, к какому типу нужно преобразовать данные и как расположить их в файле (сколько значений в каждой строке).

Возможные значения параметра <формат> для функции fprintf

Символы	Описание
%c	Одиночный символ
%d	Двоичное число
%e, %E, %g, %G	Число с плавающей запятой

Символы	Описание
%f	Число с фиксированной запятой
%o	Восьмеричное число
%s	Строка символов
%u	Десятичное число со знаком
%x, %X	Шестнадцатеричное число
\n	Начало строки
\t	Символ табуляции
\b	Возврат на один символ
\r	Конец строки
\f	Новая страница
\\	\ (обратный слеш)
\'' или ''	' (одиночная кавычка)
%%	% (процент)

Значение параметра <формат> функции fprintf, как правило, представляет собой комбинацию перечисленных символов и цифр, которые позволяют регулировать количество выводимых в файл знаков (всего и после запятой) в числовых данных.

Пример:

'\n %5.3f %10.4e\r' или ' %5.3f %10.4e\n'

Такое значение параметра <формат> функции fprintf означает, что при выводе данных в каждой строке файла будет находиться по два числа: одно с тремя знаками после фиксированной запятой, другое в экспоненциальном виде (0,385 0,2419e–003). Начало и конец строки обозначены символами '\n' и '\r'.

Если символы начала и конца строки не используются, то все данные выводятся в одну строку, даже если для вывода информации функция fprintf используется несколько раз и для разных переменных.

Для разделения выведенных в файл значений пробелами рекомендуется указывать их между соответствующими спецификациями <формата> функции fprintf.

Пример вывода элементов двумерного массива в файл.

Задание 7.5. Значения элементов двумерного вещественного массива A размером 4×3 выведите в файл d:\work\mas3.txt

Программа.

```
A=[ 4.78  5.34  8.91; 10.14 17.77 13.24;...
   11.15  9.67  5.08;  3.11  7.32  10.01];
F = fopen('d:\work\mas3.txt','w');
    fprintf(F,' %8.2f %8.2f %8.2f %8.2f\n',A)
fclose (F);
```

Файл `d:\work\mas3.txt`

```
    4.78    10.14    11.15     3.11
    5.34    17.77     9.67     7.32
    8.91    13.24     5.08    10.01
```

В каждой строке файла записан столбец массива.

Пример вывода элементов одномерного массива в файл.

Задание 7.6. Значения элементов одномерного вещественного массива `A` длиной 10 выведите в файл `d:\work\mas2.txt` в виде столбца.

Программа	Файл `d:\work\mas2.txt`

`F=fopen('d:\work\mas2.txt','w');`
`fprintf(F,' %8.2f\n',A);`
`fclose (F);`
	4.78
	5.34
	8.91
	10.14
	17.77
	13.24
	11.15
	9.67
	5.08
	3.11

Кроме специальных символов параметр `<формат>` функции `fprintf` может содержать любой пояснительный текст, которым необходимо сопроводить выводимые значения.

Пример вывода значений скалярных переменных в файл.

Задание 7.6. Выведите значения переменных `x`, `y`, `z` (вещественные) и `nx`, `ny`, `nz` (целые) с текстовыми пояснениями в файл `d:\work\res.txt`

Программа.

```
FL = fopen('d:\work\res.txt','w');
   % вывод поясняющего текста в отдельной строке
   fprintf(FL,' %s\n', 'Координаты x,y,z');
   % вывод числовых значений
   fprintf(FL,' %5.2f  %5.2f  %5.2f\n', x,y,z);
   % вывод текста и значений в одной строке
   fprintf(FL,' nx=  %g ',nx);
   fprintf(FL,'ny=  %g \n',ny);
   fprintf(FL,' nz=  %g ',nz);
fclose (FL);
```

Файл d:\work\res.txt

```
Координаты x,y,z
  4.78   5.34   8.91
 nx=   20 ny=   55
 nz=   17
```

ЛИТЕРАТУРА

1. *Дьяконов В.П.* MATLAB. Полный самоучитель. М. : ДМК Пресс, 2014. 768 с.: ил.
2. *Дьяконов В.П.* MATLAB 7.*/R2006/R2007: Самоучитель. М. : ДМК Пресс, 2008. 768 с.: ил.
3. *Мещеряков В.В.* Задачи по математике с MATLAB&SIMULINK. М. : ДИАЛОГ-МИФИ, 2007. 528 с.
4. *Потемкин В.Г.* MatLAB 5 для студентов : справ. пособие. М. : ДИАЛОГ-МИФИ, 1998. 314 с.
5. *Лазарев Ю.Ф.* Начала программирования в среде MatLAB : учеб. пособие. Киев : НТУУ «КПИ», 2003. 424 с.
6. *Ануфриев И.Е., Смирнов А.Б., Смирнова Е.Н.* MATLAB 7. СПб. : БХВ-Петербург, 2005. 1104 с.: ил.
7. *Половко А.М., Бутусов П.Н.* MatLab для студента. СПб. : БХВ-Петербург, 2005. 320 с.: ил.

СОДЕРЖАНИЕ

Предисловие ... 3
Глава 1. Среда разработки программных решений MatLab ... 4
1.1. Запуск среды MatLab ... 4
1.2. Структура основного окна среды MatLab 2013 ... 4
1.2.1. Панель инструментов среды MatLab 2013 ... 5
1.2.2. Командное окно среды MatLab 2013 ... 7
1.2.3. Информационные окна ... 8
1.3. Создание m-файла ... 8
1.4. Редактирование программы ... 9
1.5. Сохранение программы на диске ... 10
1.6. Выполнение программы ... 11
1.7. Просмотр результатов работы программы ... 12
1.8. Встроенная справочная система ... 12
Глава 2. Основные элементы языка программирования MatLab ... 13
2.1. Структура программы ... 13
2.2. Идентификаторы ... 14
2.3. Типы данных ... 14
2.4. Константы и переменные ... 15
2.5. Операции ... 16
2.6. Комментарии ... 18
Глава 3. Элементы структурного программирования ... 19
3.1. Ввод информации с клавиатуры в диалоговом режиме ... 19
3.2. Вывод информации ... 21
3.3. Оператор присвоения ... 26
3.4. Оператор двоеточия (:) ... 29
3.5. Операторы условного перехода ... 29
3.5.1. Оператор условного перехода IF ... 30
3.5.2. Оператор выбора SWITCH ... 34
3.6. Операторы цикла ... 38
3.6.1. Оператор FOR-цикла ... 38
3.6.2. Оператор цикла WHILE ... 39
3.6.3. Особенности использования операторов циклов FOR, WHILE ... 39
3.6.4. Вычисление сумм и произведений ... 43

3.7. Массивы 47
3.7.1. Ввод массива с клавиатуры 48
3.7.2. Вывод значений, хранящихся в массиве 50
3.7.3. Обсчет данных, расположенных в массивах 55
3.7.4. Простейшие задачи поиска элементов в массиве 59
3.7.5. Простейшие задачи по изменению расположения данных в массиве 61
3.8. Нестандартные функции 63
3.8.1. m-функции 64
3.8.2. Подфункции 68
3.8.3. InLine-функции 69
Глава 4. Стандартные функции MatLab 71
4.1. Генераторы случайных чисел 71
4.2. Операции с одномерными массивами 75
4.3. Операции с матрицами 76
Глава 5. Базовые конструкции программирования, реализованные на языке MatLab 78
5.1. Присвоение значений 78
5.2. Ввод данных 83
5.3. Вывод данных 88
5.4. Вычисления 100
5.5. Заполнение массива 105
5.6. Вычисления с массивами 120
5.7. Поиск минимального и максимального элементов массива 131
5.8. Поиск информации в массиве 137
5.9. Изменение взаимного расположения элементов в массиве 142
5.10. Изменение значений элементов в массивах 148
Глава 6. Графические возможности MatLab 152
6.1. График функции одной переменной 152
6.2. График нескольких функции одной переменной 158
6.3. График функции двух переменных 159
6.4. Анимация 162
6.5. Настройка параметров графического окна и области отображения графика 165
6.5.1. Стандартные инструменты графического окна 165
6.5.2. Дескрипторная графика 169

6.6. Визуально-ориентированное проектирование Windows-интерфейса 175
6.6.1. Конструктор графического интерфейса 175
6.6.2. Добавление объектов 178
6.6.3. Редактирование свойств объектов 180
6.6.4. Программная обработка событий с объектами 181
6.6.5. Диалоговые окна 191
Глава 7. Чтение данных из файла. Запись данных в файл 194
7.1. Обращение к файлу 194
7.2. Чтение данных из файла 196
7.3. Запись данных в файл 199
Литература 203

Учебное издание

Ольга Геннадьевна Ревинская

ОСНОВЫ ОБРАБОТКИ ДАННЫХ В СРЕДЕ MATLAB 2013

Часть 1

Учебное пособие

Редактор Н.А. Афанасьева
Компьютерная верстка А.И. Лелоюр
Дизайн обложки Л.Д. Кривцовой

Подписано к печати 01.12.2015 г. Формат $60\times84^{1}/_{16}$.
Бумага для офисной техники. Гарнитура Times.
Усл. печ. л. 12.
Тираж 60 экз. Заказ № 1337.

Отпечатано на оборудовании
Издательского Дома
Томского государственного университета
634050, г. Томск, пр. Ленина, 36
Тел. 8+(382-2)–53-15-28
Сайт: http://publish.tsu.ru
E-mail: rio.tsu@mail.ru