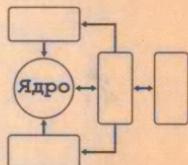
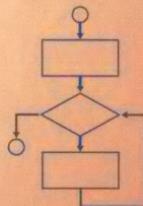
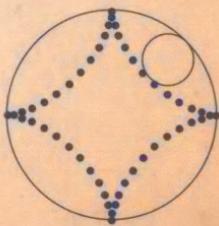


ОСНОВЫ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ

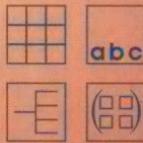
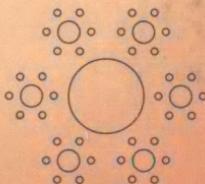
```
function [ ]=circle(r,x0,y0)
%CIRCLE создание окружности
% r - радиус окружности
% x0, y0 - координаты центра
t = 0:pi/64:2*pi;
x = x0 + r*cos(t);
y = y0 + r*sin(t);
plot(x,y)
```



$$\begin{array}{c} \square\square\square * \square\square = \square \\ \square\square * \square\square\square = \square\square\square\square \end{array}$$



$$Ax=b$$

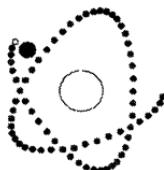
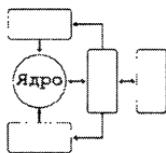


С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ
MATLAB

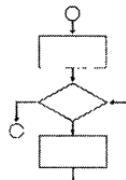
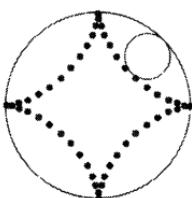


ОСНОВЫ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ

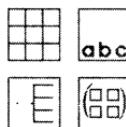
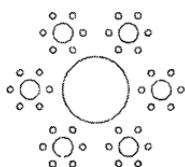
```
function []=circle(r,x0,y0)
%СIRCLE создание окружности
% r - радиус окружности
% x0 y0 - координаты центра
t = 0:pi/64:2*pi;
x = x0 + r*cos(t);
y = y0 + r*sin(t);
plot(x,y)
```



$$\begin{array}{c} \boxed{\quad} * \boxed{\quad} = \boxed{\quad} \\ \boxed{\quad} * \boxed{\quad\quad} = \boxed{\quad\quad\quad} \end{array}$$



$$Ax = b$$



С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ
MATLAB



УДК 681
ББК 32.97
К 82

КРИВИЛЁВ А.В.

К 82 Основы компьютерной математики с использованием
системы MATLAB. М.: Лекс-Книга, 2005. – 496 с. с ил.

ISBN 5-94558-013-9 (в пер.)

Рецензенты:

Кафедра системы автоматического управления Тульского го-
сударственного университета (заведующий кафедрой доктор
технических наук, профессор О.В. ГОРЯЧЕВ);

Л.Д. ПЕВЗНЕР, доктор технических наук, профессор

В настоящем учебном пособии представлено описание методики изучения одного из важнейших разделов математики – «Линейная алгебра и аналитическая геометрия» – с использованием системы компьютерной математики MATLAB. Приводится описание структурного стиля программирования и низкоуровневых операций по работе с файлами с применением высокогоуровневого языка программирования 4-го поколе-
ния. Осуществляется знакомство с фракталами на примерах построения геометрических фракталов.

В учебное пособие включено более 170 примеров и упражнений, около 200 индивидуальных заданий и более 100 контрольных вопросов.

Данное учебное пособие предназначено для студентов и специалистов, желающих освоить компьютерную математику и занимающихся разработкой новых алгоритмов и прикладных программ.

ISBN 5-94558-013-9

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав

**Учебное пособие выполнено при финансовой
и технической поддержке компании**



© Кривилев А.В., 2005
© Лекс-Книга, оформление, 2005

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ	9
Г л а в а 1. СИСТЕМЫ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ 15	
1.1. Классы систем компьютерной математики	15
1.2. Структура систем компьютерной математики	18
1.3. Состав системы MATLAB	20
Г л а в а 2. КЛАССЫ ДАННЫХ 25	
2.1. Арифметические и логический классы данных	25
2.1.1. Арифметические классы данных	25
2.1.2. Логический класс данных	29
2.1.3. Приоритет операций	32
2.2. Символьный класс данных	33
2.2.1. Создание переменных символьного класса	34
2.2.2. Преобразование переменной из символьного класса в арифметический класс	35
2.2.3. Вычисление символьных выражений	38
2.3. Массив структуры	39
2.3.1. Организация массива структуры в системе MATLAB	40
2.3.2. Создание переменных класса массив структуры	41
2.3.3. Просмотр переменных класса массив структуры	44
2.3.4. Сортировка имен полей	46
2.3.5. Удаление поля из массива структуры	47
2.4. Массив ячеек	48
2.4.1. Создание массива ячеек	49
2.4.2. Доступ к данным массива ячеек	50
2.4.3. Удаление ячеек и их содержимого из массива ячеек	51
2.5. Проверка класса данных	52
2.6. Выводы к главе 2	53

Г л а в а 3. РЕШЕНИЕ ЗАДАЧ ВЕКТОРНОЙ АЛГЕБРЫ	55
3.1. Векторы	55
3.1.1. Определение вектора	55
3.1.2. Линейные операции над векторами и их свойства	58
3.1.3. Длина вектора	60
3.1.4. Направляющие косинусы вектора	61
3.2. Скалярное произведение векторов	62
3.2.1. Определение скалярного произведения векторов	62
3.2.2. Скалярное произведение векторов, заданных в координатной форме	63
3.2.3. Свойства скалярного произведения	63
3.2.4. Определение угла между двумя векторами	64
3.3. Векторное произведение двух векторов	64
3.3.1. Определение векторного произведения	64
3.3.2. Свойства векторного произведения	65
3.4. Смешанное произведение трех векторов	66
3.4.1. Определение смешанного произведения	66
3.4.2. Свойства смешанного произведения	67
3.5. Внешнее произведение векторов	67
3.6. Норма вектора	68
3.6.1. Определение нормы вектора	68
3.6.2. Расстояние между векторами	70
3.7. Выводы к главе 3	71
Г л а в а 4. РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ	73
4.1. Матрицы	73
4.1.1. Основные понятия	73
4.1.2. Специальные матрицы	77
4.1.2.1. Диагональные матрицы	77
4.1.2.2. Треугольные и трапециевидные матрицы	77
4.1.2.3. Симметричные матрицы	78
4.1.2.4. Магический квадрат	79

4.1.3. Действия над матрицами	80
4.1.3.1. Линейные операции над матрицами и их свойства	80
4.1.3.2. Элементарные матричные преобразования	82
4.1.3.3. Произведение матриц	83
4.1.4. Определитель матрицы и его свойства	85
4.1.5. Невырожденные матрицы	87
4.1.5.1. Основные понятия	87
4.1.5.2. Обратная матрица и ее свойства	88
4.1.5.3. Ранг матрицы и его свойства	89
4.1.6. Норма матрицы	91
4.2. Системы линейных алгебраических уравнений	92
4.2.1. Основные понятия	92
4.2.2. Теорема Кронекера–Капелли	94
4.2.3. Матричный метод решения СЛАУ	94
4.2.4. Формулы Крамера	95
4.2.5. Метод Гаусса	97
4.2.6. Метод Гаусса с частичным выбором главного элемента	101
4.2.7. LU-разложение	104
4.2.8. Последовательность решения СЛАУ в системе MATLAB	106
4.3. Выводы к главе 4	107

Г л а в а 5. РЕШЕНИЕ ЗАДАЧ АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ	109
5.1. Аналитическая геометрия на плоскости	109
5.1.1. Системы координат на плоскости	109
5.1.2. Преобразование координат из одной системы в другую	112
5.1.3. Линии первого порядка.....	113
5.1.3.1. Общее уравнение прямой линии	113
5.1.3.2. Уравнение прямой линии с угловым коэффициентом	114
5.1.3.3. Уравнение прямой линии, проходящей через данную точку в заданном направлении	115
5.1.3.4. Уравнение прямой, проходящей через две точки	117
5.1.3.5. Уравнение прямой линии в отрезках	118
5.1.3.6. Уравнение прямой линии, проходящей через заданную точку перпендикулярно заданному вектору	121

5.1.3.7. Полярное уравнение прямой линии	122
5.1.3.8. Параметрический способ задания линии на плоскости	124
5.1.4. Линии второго порядка	126
5.1.4.1. Окружность	127
5.1.4.2. Эллипс	128
5.1.4.3. Гипербола	131
5.1.4.4. Парабола	136
5.1.5. Кривые высших порядков.....	142
5.2. Аналитическая геометрия в пространстве	146
5.2.1. Системы координат в пространстве	146
5.2.2. Трансформация системы координат в пространстве	148
5.2.3. Уравнения плоскости в пространстве	152
5.2.3.1. Общее уравнение плоскости	152
5.2.3.2. Уравнение плоскости, проходящей через три заданные точки	156
5.2.3.3. Уравнение плоскости в отрезках	157
5.2.3.4. Уравнение плоскости, проходящей через точку перпендикулярно данному вектору	160
5.2.4. Уравнения линии в пространстве	162
5.2.4.1. Общие уравнения прямой линии	162
5.2.4.2. Параметрические уравнения прямой линии	164
5.2.4.3. Канонические уравнения прямой линии	166
5.2.4.4. Уравнение прямой линии в пространстве, проходящей через две точки	169
5.2.5. Прямая и плоскость в пространстве. Основные задачи	171
5.2.5.1. Определение угла между двумя плоскостями	171
5.2.5.2. Вычисление расстояния от точки до плоскости	172
5.2.5.3. Определение угла между прямыми линиями	174
5.2.5.4. Определение взаимного расположения прямых линий в пространстве	175
5.2.5.5. Вычисление угла между прямой линией и плоскостью	175
5.2.5.6. Определение точки пересечения прямой линии и плоскости	176
5.2.6. Цилиндрические поверхности	178
5.2.7. Поверхности вращения	180
5.2.8. Поверхности второго порядка	183

5.2.8.1. Эллипсоид	184
5.2.8.2. Однополостный гиперболоид	186
5.2.8.3. Двуполостный гиперболоид	189
5.2.8.4. Эллиптический параболоид	190
5.2.8.5. Гиперболический параболоид	192
5.2.8.6. Конус второго порядка	196
5.3. Выводы к главе 5	198
 Г л а в а 6. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ MATLAB.....	201
6.1. Краткий обзор языков программирования	201
6.2. Понятие <i>m</i> -файла	203
6.3. Управляющие структуры	206
6.3.1. Структура выбора <i>if end</i> (ЕСЛИ)	206
6.3.2. Структура выбора <i>if else end</i> (ЕСЛИ ИНАЧЕ)	211
6.3.3. Структура выбора <i>if elseif else end</i> (ЕСЛИ ИНАЧЕ ЕСЛИ)	214
6.3.4. Структура множественного выбора <i>switch case end</i>	218
6.3.5. Структура повторения <i>for end</i>	222
6.3.6. Структура повторения <i>while end</i>	228
6.3.7. Функции <i>break()</i> и <i>continue()</i>	231
6.3.8. Заключение по структурному программированию	235
6.4. Функции	237
6.4.1. Описание функции	237
6.4.2. Вызов функции	242
6.4.3. Классы памяти и области видимости переменных	246
6.4.4. Параметры функции	252
6.4.4.1. Основные понятия	252
6.4.4.2. Инициализация входных формальных параметров	254
6.4.4.3. Обязательные и дополнительные выходные формальные параметры	258
6.4.4.4. Задание произвольного числа входных формальных параметров	260
6.4.4.5. Задание произвольного числа выходных формальных параметров	264

6.4.5. Подфункции	270
6.4.6. Рекурсивные функции	272
6.5. Файлы	283
6.5.1. Основные понятия	284
6.5.2. Текстовые файлы	285
6.5.2.1. Основные операции	286
6.5.2.2. Форматированный вывод данных	293
6.5.2.3. Форматированный ввод данных	299
6.5.3. Двоичные файлы	303
6.5.3.1. Основные операции	303
6.5.3.2. Произвольный доступ	309
6.6. Выводы к главе 6.....	312
ПРАКТИКУМЫ	317
Практикумы 1, 2. Знакомство с интерфейсом системы MATLAB.....	319
Практикум 3. Командное окно	351
Практикум 4. Классы данных	353
Практикум 5. Решение задач линейной алгебры	363
Практикумы 6, 7. Графические средства MATLAB	371
Практикум 8. Инstrumentальное средство Editor/Debugger.....	401
Практикум 9. Инstrumentальное средство Profiler	411
Практикум 10. Функции	431
Практикум 11. Геометрические фракталы	443
Практикум 12. Файлы	459
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	473
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	477

**Посвящается
моим родителям**

ПРЕДИСЛОВИЕ

*Всякое завоевание, всякий шаг вперед в познании
вытекает из мужества, из строгости к себе,
из чистоплотности в отношении себя ...*

Ф. Ницше. Ecce Homo. Как становятся самим собой

В последнее время наблюдается стремительное развитие систем компьютерной математики. Увеличивающаяся популярность математических пакетов сопряжена с усовершенствованием известных методов решения, а также с разработкой новых методов и средств, которые позволяют значительно сократить время, затрачиваемое на получение результата, визуализировать процесс решения и оформить полученное решение в виде отчета. Системы компьютерной математики находят применение при:

- проектировании систем управления летательными аппаратами и другими транспортными средствами;
- разработке современного коммуникационного, компьютерного и офисного оборудования;
- создании современной медицинской техники;
- анализе данных, полученных после различных исследований;
- прогнозировании ситуации на финансовых рынках и т.д.

Ориентация ведущих зарубежных и ряда отечественных компаний на использование систем компьютерной математики при выполнении своих проектов послужила дополнительным стимулом для компаний-разработчиков и связанных с ними институтов в работе над совершенствованием математических пакетов. На сегодняшний день системы компьютерной ма-

тематики достигли такого уровня развития, что овладение основными принципами их использования требует от пользователя хорошей математической подготовки. Для обеспечения задач современного образования фирмы-разработчики пошли на создание академических (студенческих) версий, которые можно использовать в высших учебных заведениях при выполнении семестровых курсовых работ и дипломных проектов. Системы компьютерной математики используются при обучении студентов в более чем 3500 институтов и университетов по всему миру.

В ряде случаев фирмы – разработчики математических пакетов не ограничились созданием только студенческих версий, а разработали специализированные программы, которые школьный учитель или преподаватель высшего учебного заведения может использовать при подготовке и во время проведения школьного урока, лекции или семинара соответственно, а также при проведении тестирования знаний учащихся.

С учетом изложенного необходимо как можно раньше начинать знакомство с этими системами, поскольку овладение навыками профессиональной работы с системами компьютерной математики позволит найти интересную и высокооплачиваемую работу в ведущих организациях как в России, так и за рубежом, а также легко влиться в коллектив высококвалифицированных специалистов.

Наряду с ростом популярности систем компьютерной математики из года в год увеличивается количество книг, посвященных этим системам, а также специализированным направлениям, в которых указанные системы используются в качестве инструмента для решения задач. Большинство книг на русском языке являются либо компиляцией материала из различных областей математики с небольшим числом примеров и поверхностным описанием интерфейса наряду с языком соответствующего математического пакета, использовавшимся при решении задач, либо переводом англоязычных источников (книг или справочной информации, поставляемой в электронном виде). В подавляющем большинстве книг отсутствует методическое изложение материала с большим количеством обобщающих примеров, которое является отличительной особенностью российской школы.

Целью данной книги является:

- представление методики изучения предмета с использованием системы компьютерной математики;
- описание структурного стиля программирования с использованием высококоуровневого языка 4-го поколения;
- знакомство с фракталами на примере геометрических фракталов.

В качестве предмета была взята дисциплина «Линейная алгебра и аналитическая геометрия», а в качестве инструмента – система MATLAB. Дисцип-

лии «Линейная алгебра и аналитическая геометрия» выбрана по двум причинам: во-первых, эта дисциплина является обязательной и изучается на начальных курсах; во-вторых, ее основные понятия используются при разработке современных технических устройств и написании компьютерных программ, активно использующих компьютерную графику. Выбор MATLAB обусловлен наибольшей популярностью этой системы в технических исследованиях и наличием в ее составе языка программирования 4-го поколения, который ориентирован на матричные операции.

Структурный стиль программирования позволяет решать математические задачи, а также служит начальной ступенью для изучения объектно-ориентированного стиля программирования.

Геометрические фракталы выбраны с целью наглядной демонстрации рекурсии, а также для ознакомления с природой фракталов, которые используются при проектировании сложных технических устройств и отображении на экране элементов компьютерной графики.

Материал подается в следующей последовательности:

- теоретическая часть;
- обобщающий пример или несколько примеров, которые наилучшим образом демонстрируют особенности представленного в теоретической части материала;
- практикум.

В некоторых случаях в теоретическую часть вставлены исторические справки о людях, которые внесли значительный вклад в развитие соответствующего направления или являются авторами описанных методов. В книге содержится более 170 примеров и упражнений, способствующих усвоению теоретического материала, а также принципов работы с графическим интерфейсом пользователя системы MATLAB. В практикумы включен дополнительный теоретический материал, который является необходимым при выполнении соответствующего практикума. Практикумы предназначены для приобретения навыков работы с системой MATLAB, а также для закрепления соответствующего материала, представленного в теоретической части. Большинство практикумов, помимо упражнений, содержат индивидуальные задания и контрольные вопросы. В общей сложности учебное пособие содержит более 200 индивидуальных заданий и более 100 контрольных вопросов.

Методические принципы и организация книги максимально облегчают процесс обучения и вырабатывают основные приемы для численного решения задач с последующей визуализацией результатов и их анализа.

Книга состоит из шести глав и двенадцати практикумов. Начиная со второй, каждая глава книги снабжена теоретическим материалом и примерами

решения соответствующих задач. В конце этих глав приводятся выводы со списком использовавшихся в них функций в виде таблицы.

В главе 1 содержится краткий обзор систем компьютерной математики и приводится описание состава системы MATLAB и ее пакетов расширения.

Глава 2 включает описание основных классов данных, которые используются при решении задач линейной алгебры и аналитической геометрии, а также классов, используемых для хранения разнородной информации. На примерах рассматривается использование арифметического, логического и символьного классов данных, а также массива ячеек и массива структур.

В главу 3 включены примеры решения задач векторной алгебры, описывающие создание векторов, вычисление скалярного, векторного, смешанного и внешнего произведений, а также различных векторных норм.

Глава 4 содержит описание решения задач линейной алгебры. На примерах продемонстрировано создание обычных и специальных матриц, действия над матрицами, вычисление определителя, ранга и норм матриц. Рассмотрены различные способы решения системы линейных алгебраических уравнений.

В главе 5 содержатся примеры решения задач, которые встречаются в аналитической геометрии. Рассматривается построение линий различных порядков на плоскости и поверхностей и линий в пространстве. Приводятся примеры, в которых используются аффинные преобразования.

Глава 6 содержит описание языка программирования MATLAB. Рассматривается применение структурного подхода для решения математических задач с использованием языка программирования высокого уровня 4-го поколения. Наряду с описанием основных языковых конструкций рассматриваются низкоуровневые операции по работе с текстовыми и двоичными файлами.

Практикумы 1 и 2 предназначены для знакомства и получения основных навыков работы с элементами графического интерфейса пользователя системы MATLAB.

Практикум 3 служит для более детального знакомства с приемами работы в командном окне системы MATLAB.

Практикум 4 является дополнением к главе 2 и предназначен для закрепления на практике приемов работы с соответствующими классами данных.

Практикум 5 представляет собой дополнение к главе 3 и главе 4 и служит для закрепления на практике широко известных методов решения систем линейных алгебраических уравнений.

Практикумы 6 и 7 служат для знакомства с объектами дескрипторной графики, графическим окном системы MATLAB, а также для приобретения

навыков построения линий на плоскости и поверхностей в пространстве. Эти практикумы являются дополнением к главе 5.

Практикум 8 предназначен для знакомства с инструментальным средством Editor/Debugger, которое используется для написания программ на языке MATLAB и их отладки.

Практикум 9 служит для знакомства с инструментальным средством Profiler, которое используется для создания профиля программы с целью анализа его эффективности.

Практикум 10 предназначен для закрепления навыков по созданию и использованию функций в математических задачах.

Практикум 11 служит для знакомства с геометрическими фракталами и закрепления навыков по работе с рекурсивными функциями.

Практикум 12 предназначен для закрепления навыков по работе с текстовыми и двоичными файлами с помощью низкоуровневых функций языка программирования MATLAB.

Практикумы 8, 9, 10, 11 и 12 являются дополнением к главе 6.

Вместе с книгой предлагается диск, который содержит примеры решения большинства задач, рассмотренных в книге. На диске также размещаются программы обучения работе с командным окном системы MATLAB (lab3_1, lab3_2 и lab 3_3). Программы могут быть использованы при формировании практикума 3. Они могут работать в двух режимах – в режиме обучения и в режиме тестирования. Программы предлагаются в m-файлах, и, следовательно, количество вопросов и сами вопросы могут быть изменены по усмотрению преподавателя. На диске находится программа lab67, позволяющая просмотреть результаты выполнения практикумов 6 и 7 согласно индивидуальному варианту. В директории lab11 располагаются файлы, с помощью которых можно просмотреть последовательность построения геометрических фракталов, а также задания на самостоятельную работу согласно индивидуальному варианту.

В основу книги лег материал курса «Информатика», читаемый автором совместно с Н.М. Козловой и К.М. Тихоновым студентам кафедр «Системы приводов авиационно-космической техники» и «Авиационные робототехнические системы» факультета «Робототехнические и интеллектуальные системы» Московского авиационного института (государственного технического университета).

Любые замечания, которые могут способствовать улучшению как содержания, так и формы книги, необходимо направлять на электронный адрес автора – k_alex@aport2000.ru.

Благодарности

Автор благодарен проф., докт. техн. наук В.Г. Стеблецову, который инициировал процесс написания учебного пособия и давал ценные рекомендации по ходу его подготовки.

Содействие и доброжелательное участие заслуженного деятеля науки и техники РФ, проф., докт. техн. наук Л.В. Рабиновича, доц., канд. техн. наук М.М. Медынского и доц., канд. техн. наук Н.М. Козловой в обсуждении основного круга идей, отраженных в книге, позволили улучшить ее содержание и структуру.

Особую признательность автор выражает доц., канд. техн. наук К.М. Тихонову, который подготовил практикум 12 и проверил все примеры, представленные в учебном пособии. Совместно с ним были написаны главы 1 и 2.

Большой труд по написанию графического интерфейса пользователя программы lab67 для практикумов 6 и 7 взял на себя студент кафедры «Системы приводов авиационно-космической техники» С.А. Гагарин.

Автор выражает огромную благодарность всем студентам, которые прослушали курс и во время его прослушивания выражали свое удовлетворение или недовольство материалом и/или формой его изложения, что в значительной мере позволило улучшить представленный в книге курс лекций и практикумов.

Глава 1. СИСТЕМЫ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ

Компьютерная математика – это совокупность методов и средств, обеспечивающих эффективную подготовку программ для решения математических задач любой сложности (с высокой степенью точности полученного результата и визуализации всех этапов решения).

1.1. КЛАССЫ СИСТЕМ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ

В настоящее время системы компьютерной математики (СКМ) условно делят на следующие классы [22]:

1. Системы для численных расчетов.
2. Табличные процессоры.
3. Матричные системы.
4. Системы статистических расчетов.
5. Системы специальных расчетов.
6. Системы аналитических расчетов.
7. Универсальные системы.

Рассмотрим кратко назначение каждого из указанных классов систем компьютерной математики.

Системы для численных расчетов решают следующие задачи: преобразование форматов представления чисел и углов, арифметические и алгебраические вычисления с целыми, вещественными и комплексными числами, вычисление тригонометрических и гиперболических функций, вычисление функций алгебры логики, выполнение простейших статистических расчетов и др.

Простейшими примерами систем данного класса являются система Calculator, которая поставляется вместе с операционной системой Microsoft Windows, или Calc98, созданная фирмой *Flow Simulation Ltd., USA*, для карманных компьютеров [69].

Табличные процессоры предназначены в первую очередь для работы с табличными данными. Имеют в своем составе большое число встроенных функций, позволяющих решать широкий спектр финансово-экономических задач и представлять результаты вычислений как в табличном виде, так и в виде различных графиков.

Первым табличным процессором для персональных компьютеров был VisiCalc, разработанный Дэном Бриклином (Dan Bricklin) и Бобом Франкстоном (Bob Frankston) в 1979 году [65, 70].

Наиболее распространенными системами этого класса на сегодняшний день являются электронные таблицы Lotus 1-2-3 (развитие VisiCalc) фирмы *Lotus Software Corp., USA*, Quattro Pro фирмы *Corel Corp., Canada*, и Excel

фирмы *Microsoft Corp., USA*. Все пакеты входят в состав офисных систем Lotus SmartSuite, Corel WordPerfect Office и Microsoft Office соответственно [66, 72, 77].

Матричные системы основаны на представлении всех объектов в виде матрицы. Даже одно число в таких системах рассматривается как матрица размером 1×1 . Такой подход позволяет применить единую идеологию к проведению вычислений как над простыми объектами, так и над структурами.

Основными представителями систем данного класса являются ранние версии программы MATLAB фирмы-производителя *MathWorks Inc., USA*, и программа SCILAB, которая разработана в 1990 году и сопровождается группой разработчиков из *INRIA Rocquencourt* (французский Национальный Институт Исследований в области Информатики и Автоматики) и *ENPC* (французская Национальная Школа Мостов и Дорог) [4, 75]. Отличительная особенность программы SCILAB от MATLAB – ее свободное распространение.

Системы статистических расчетов служат для извлечения многосвязных данных, проведения статистического анализа, визуализации результатов анализа, статистического контроля производственного процесса и т.п.

Наиболее распространенными в этом классе являются зарубежные системы GAUSS, S-PLUS, SPSS и STATISTICA, разрабатываемые фирмами *Ap-tech Systems Inc., USA*, *Insightful Corp., USA*, *SPSS Inc., USA* и *StatSoft Inc., USA* соответственно, а также отечественная система STADIA, созданная в 1988 году выпускником МАИ А.П. Кулакиным [3, 8, 10, 33, 55, 71, 79, 80].

Системы для статистических расчетов наиболее широко применяются в отечественных государственных органах (Госкомстат РФ, Центр стандартизации и метрологии), банковских структурах (ЦБ РФ), медицинских организациях (Московский научно-исследовательский институт диагностики и хирургии) и др.

Системы для специальных расчетов связаны с решением задач в отдельных областях науки и техники, где необходимо применение современных математических методов. В качестве примера можно привести систему конечно-элементного моделирования MSC.NASTRAN (фирма-производитель *MSC.Software Corp., USA* [76]), предназначенную для решения задач статической и динамической прочности конструкций, задач теплопроводности и др.

Системы имеют широкое распространение в области проектирования изделий космической (ГКНПЦ им. М.В. Хруничева) и авиационной (ОАО «ОКБ Сухого») техники, в автомобилестроении (АО «АвтоВАЗ») и др.

Системы аналитических расчетов предназначены для решения задач в символьном виде (вывод формул, преобразование выражений, символьное интегрирование и дифференцирование и др.). Это наиболее интеллектуальное направление из рассмотренных ранее систем компьютерной математики.

Отличительной особенностью таких систем является возможность получения результата в общем виде [2].

Например, необходимо вычислить производную от тригонометрической функции $\sin(x)$. При решении этой задачи в численном виде необходимо сформировать в некотором диапазоне с как можно меньшим шагом массив значений аргумента. Причем с уменьшением шага увеличивается объем, который занимает этот массив в памяти компьютера. Кроме того, для вычисления производной необходимо воспользоваться соответствующим численным методом. Программа таких вычислений будет достаточно сложной, а результат будет иметь погрешность. В системе символьной математики результатом решения поставленной задачи будет аналитическое выражение производной, в данном случае $\cos(x)$.

Первая система данного класса для персональных компьютеров была разработана Альбертом Ричем (Albert Rich) и Дэвидом Стутмайером (David Stoutemyer) в 1979 году и получила название *tiMATH*. Последовательницей этой системы стала система *DERIVE*, впервые выпущенная в 1988 году и применяющаяся в настоящее время как в персональных компьютерах, так и в графических микрокалькуляторах фирмы *Texas Instruments Inc., USA* (TI-89) [20, 24, 36, 67].

Такие системы наиболее целесообразно применять при обучении математике учащихся средних учебных заведений.

Универсальные системы объединили численные и аналитические расчеты, язык программирования высокого уровня, визуализацию результатов вычисления и возможность обмена информацией между системами с помощью различных форматов.

Появление на свет универсальных систем стало возможным после разработки универсальных алгоритмов решения математических задач и увеличения производительности персональных компьютеров.

Наиболее мощными системами этого класса являются *Maple* (*MapleSoft Corp., Canada*), *MathCad* (*MathSoft Inc., USA*), *Mathematica* (*Wolfram Research Inc., USA*) и *MATLAB* (*Mathworks Inc., USA*) [73, 74, 75, 81]. Из вышеназванных систем только *Mathematica*, впервые появившаяся на свет в 1988 году, разрабатывалась под руководством Стефана Вольфрама (Stephen Wolfram) как универсальная система [21]. Остальные системы были ведущими пакетами в своих классах и стали универсальными уже позднее.

Система *MathCad* являлась лидером в классе систем для численных расчетов и стала универсальной в 1994 году после интеграции в нее усеченной версии ядра *Maple* для аналитических расчетов [16, 32, 49].

Система *Maple* была разработана исследовательской группой, созданной Кейтом Геддом (Keith Geddes) и Гастоном Гонэ (Gaston Gonnet) на кафедре вычислительной техники в университете города Waterloo, в провинции Он-

tario (Канада). Сначала она была ориентирована на решение задач в символьном виде и перешла в класс универсальных систем после постепенного включения в нее численных алгоритмов. Несмотря на то что начальные версии системы Maple появились в 1982 году, первая коммерческая версия системы Maple вышла лишь в 1985 году, после пятилетней работы над проектом [11, 13, 39, 60, 61].

СКМ MATLAB (Matrix Laboratory – матричная лаборатория) была разработана Кливом Моулером (Cleve Moler) в конце 70-х годов как интерактивная среда доступа к математическим пакетам LINPACK и EISPACK. Пакеты LINPACK и EISPACK были написаны группой ученых и программистов, среди которых был и К. Моулер, в середине 70-х годов на языке программирования FORTRAN. Пакет LINPACK содержит функции для решения линейных уравнений, а EISPACK – для вычисления собственных значений. Система MATLAB была написана на языке FORTRAN и предназначалась для обучения студентов линейной алгебре. В момент создания первой версии К. Моулер был заведующим кафедрой Computer Science в University of New Mexico. После посещения Stanford University и демонстрации там своей программы он получил предложение от Джона Литтла (John Little) разработать совместно коммерческую версию программы. В 1983 году К. Моулер, Дж. Литтл и Стив Бангерт (Steve Bangert) написали на языке C коммерческую версию системы MATLAB, которая работала с графикой. Система MATLAB перешла в класс универсальных систем в 1994 году, т.е. после того, как ядро системы Mathematica 2.2 вошло в пакет расширения Mathematica Symbolic Toolbox. Однако, начиная с пятой версии, в системе MATLAB используется ядро Maple, которое входит в пакет расширения Symbolic Math Toolbox. Система MATLAB в свою очередь может использоваться системой Maple для проведения сложных численных расчетов. В 1984 году К. Моулером и Дж. Литтлом была создана фирма *MathWorks Inc.*, специалисты которой на протяжении уже 20 лет улучшают систему MATLAB [5, 15, 23, 31, 38, 41, 50, 51, 52, 58, 62, 63].

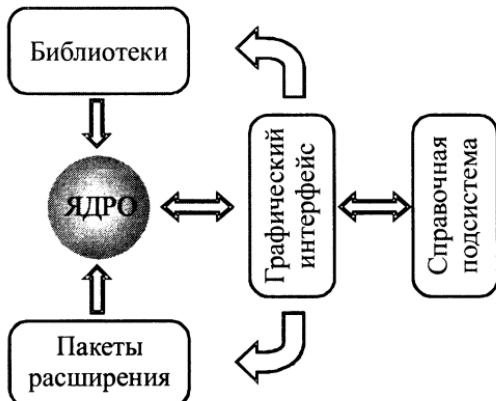
Самую свежую информацию о математических пакетах можно получить на образовательном сайте [68] или в фирме *SoftLine Group*, являющейся официальным поставщиком математического программного обеспечения на рынке РФ.

1.2. СТРУКТУРА СИСТЕМ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ

Несмотря на то что универсальные системы в той или иной мере отличаются друг от друга, они имеют сходную структуру, изображенную на рис. 1.1.

Основным элементом представленной структуры является **ядро** (core). Оно представляет собой комплекс откомпилированных (т.е. представленных в машинных кодах) функций, обеспечивающих работу всей системы в целом,

и в первую очередь оптимальную программную реализацию математических алгоритмов. Ядро в системах символьной математики играет особую роль, так как в нем содержится весь искусственный интеллект системы (правила преобразования выражений, вычисления пределов, производных и интегралов и др.). Ядро системы Mathematica 4.2 включает около 800 000 строк на языке программирования C, тогда как ядро Mathematica 1.0–15 000 строк [81].



Р и с . 1.1. Структура СКМ

Ядро должно обладать как можно большим интеллектом и работать максимально быстро, так как определяет быстродействие системы. Поэтому в него включают минимально необходимый набор функций. Все остальное, но также необходимое для функционирования системы, включают в состав *библиотек* (library). Как правило, это динамически компонуемые библиотеки (DLL), содержащие функции, к которым ядро обращается сравнительно редко.

В процессе эксплуатации той или иной системы расширяется круг решаемых ею задач. Эти задачи группируются по каким-либо признакам, из-за чего появляется необходимость дополнить систему набором специализированных функций. Эти специализированные процедуры оформляются в виде дополнительной библиотеки, которую принято называть *пакетом расширения* (Add-on, или Toolbox).

Управление процессом функционирования системы и решением задач осуществляется посредством *графического интерфейса* (front end). Все современные системы компьютерной математики имеют графический интерфейс пользователя (GUI), совместимый с операционной системой Windows. Например, с помощью графического интерфейса пользователь может использовать функцию из пакета расширения и просмотреть результат выполнения команды (Графический интерфейс → Пакет расширения → Ядро → Графический интерфейс).

Справочная подсистема (*help*) обеспечивает получение оперативных справок по работе с системой и решению тех или иных задач. Большинство справок сопровождается информативными, готовыми к выполнению примерами.

1.3. СОСТАВ СИСТЕМЫ MATLAB

Система MATLAB представляет собой набор средств для анализа данных, визуализации результатов вычисления, разработки приложений, моделирования и генерации программного кода (рис. 1.2) [75].

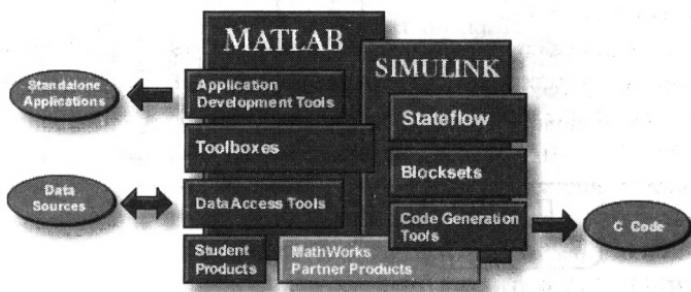


Рис. 1.2. Состав системы MATLAB

На рис. 1.2 используются следующие обозначения:

MATLAB представляет собой объединение математического ядра,ключающего язык программирования высокого уровня, библиотеки стандартных функций, графического интерфейса пользователя и справочной системы.

Application Development Tools – инструментальные средства разработки приложений позволяют создавать независимые программные продукты. В состав инструментальных средств входят: MATLAB Compiler, MATLAB Web Server, MATLAB Runtime Server, MATLAB COM Builder и MATLAB Excel Builder.

Toolboxes – пакеты расширения служат для увеличения возможностей систем MATLAB и SIMULINK в специализированных областях. Список пакетов расширения, входящих в состав системы MATLAB, приведен по категориям в табл. 1.1 применительно к версии MATLAB 6.5.1.

Data Acquisition and Access Tools – инструментальные средства обмена данными применяются для обмена информацией с внешними устройствами (осциллографами, генераторами периодических сигналов и др.), базами данных (Oracle, Access и др.) и другими приложениями (например, Excel).

MATLAB Student Version – студенческая версия системы MATLAB позволяет учащимся высших учебных заведений приобретать MATLAB, SIMULINK и пакеты символьной математики Symbolic/Extended Math Toolbox со значительными скидками. Информацию о стоимости студенческих версий MATLAB можно получить на сайте [78].

SIMULINK – пакет для визуальной разработки и моделирования непрерывных, дискретных и гибридных динамических систем с учетом нелинейностей.

Stateflow – среда разработки и моделирования конечных автоматов, изображаемых в виде графов, т.е. в виде состояний, переходов и условий переходов между состояниями.

Blocksets – наборы специализированных блоков, расширяющих возможности стандартной версии SIMULINK. Использование специализированных блоков позволяет моделировать сложные механические системы, системы гидро-, пневмо- и электроавтоматики.

Code Generation Tools – инструментальные средства создания кода предназначены с целью преобразования блок-схемы модели, реализованной в Simulink, в оптимальный код на языке программирования C для его последующей записи во встраиваемые приложения.

MathWorks Partner Products – пакеты расширения, выпущенные партнерами фирмы *MathWorks Inc.*. Пакеты расширения третьих фирм являются надстройкой над стандартным набором пакетов расширения, входящих в систему MATLAB. Полный список третьих фирм, разрабатывающих эти пакеты, можно просмотреть на сайте [75].

Таблица 1.1

Перечень пакетов расширения

Наименование продукта	Назначение
<i>Математика (Math and Analysis)</i>	
Curve Fitting	Параметрическая и непараметрическая аппроксимация данных с использованием полиномиальных, экспоненциальных и дробно-рациональных функций, а также сглаживающих сплайнов соответственно
Extended Symbolic Math	Выполнение аналитических вычислений с помощью функций, входящих в состав библиотеки расширений Maple

Продолжение табл. 1.1

Наименование продукта	Назначение
Mapping	Реализует предоставление географической информации в виде карт и осуществляет выполнение различных вычислений на картах
Neural Network	Создание, обучение и моделирование нейронных сетей для нелинейного управления нестационарными объектами, а также для предсказания поведения цен на рынке
Optimization	Определение максимумов и минимумов линейных и нелинейных функций, заданных в n -мерном пространстве
Partial Differential Equation	Решение нестационарных дифференциальных уравнений первого и второго порядка с частными производными методом конечных элементов. Используется для решения задач сопротивления материалов, электромагнетизма, тепломассопереноса и диффузии в плоскости
Spline	Создание, отображение и обработка сплайнов в наиболее употребительных формах: В-форме и в виде полиномов
Statistics	Содержит широкий набор функций для решения задач математической статистики и теории вероятности
Symbolic Math	Выполнение символьных вычислений с помощью стандартных функций, входящих в ядро Maple, и библиотеки линейной алгебры

Сбор и импорт данных (Data acquisition and import)

Data Acquisition	Обеспечивает обработку полученных от внешних устройств или извлеченных из файлов данных. Поддерживает целый ряд стандартного оборудования (порты ввода-вывода, звуковые карты и т.д.)
Database	Предоставляет графический интерфейс пользователя, с помощью которого можно осуществлять связь с наиболее известными базами данных
Image Acquisition	Включает в MATLAB возможности получения из внешних устройств графической и видеоинформации и ее визуализации

Продолжение табл. 1.1

Наименование продукта	Назначение
Instrument Control	Содержит функции, осуществляющие прямую связь МАТ-ЛАБ с внешними инструментами (осциллографами, генераторами импульсов и т.д.) посредством коммуникационных протоколов VISA и GPIB
<i>Обработка сигналов и изображений (Signal and Image Processing)</i>	
Communication	Разработка моделей цифровых и аналоговых систем и устройств связи и передачи информации. Используется для разработки и моделирования модемов
Filter Design	Предоставляет инструментальные средства для разработки и анализа различного вида фильтров
Image Processing	Цифровая обработка и анализ изображений
Signal Processing	Содержит большие возможности по спектральному анализу и фильтрации сигналов. Используется при обработке аудио- и видеоинформации, в телекоммуникационной сфере и т.д.
System Identification	Содержит инструментальные средства для создания математических моделей динамических систем на основе наблюдаемых входных и выходных данных
Wavelet	Импульсная декомпозиция сигналов и изображений
<i>Проектирование систем управления (Control Design)</i>	
Control System	Синтез и анализ линейных систем автоматического управления с использованием современных и классических методов
Fuzzy logic	Создание систем автоматического управления на базе теории нечетких множеств
LMI Control	Проектирование систем управления на основе теории линейных матричных неравенств
Model-Based Calibration	Предоставляет инструментальные средства конструирования для калибровки систем трансмиссии

Наименование продукта	Назначение
Model Predictive Control	Проектирование алгоритмов управления в дискретных и непрерывных системах на основе предсказания динамики их поведения
Robust Control	Проектирование робастных (грубых) систем управления
μ -Analysis and Synthesis	Проектирование системы управления с помощью теории оптимизации в пространствах H_2 и H_∞
<i>Финансовые приложения (Financial Modeling and Analysis)</i>	
Datafeed	Предоставляет инструментальные средства для сбора финансовой информации
Financial	Расчет процентных ставок и прибыли, анализ производных доходов и депозитов
Financial Derivatives	Выполняет анализ страхования от потерь и визуализацию результатов
Financial Time Series	Набор инструментальных средств для анализа данных финансовых рынков методом временных рядов
GARCH	Анализ изменчивости на финансовых рынках с помощью GARCH-моделей

Таким образом, основными функциями системы MATLAB являются быстрые численные алгоритмы, визуализация вычислений, интерактивная среда программирования, связь с языками программирования C, C++, FORTRAN и Java, а также обмен данными с другими приложениями.

Глава 2. КЛАССЫ ДАННЫХ

Данные в MATLAB группируются по *классам* согласно рис. 2.1 [75]. Все классы данных являются матричными и поддерживают работу с *разреженными матрицами*, где преобладают нулевые элементы. Для каждого класса данных устанавливаются свои операции.

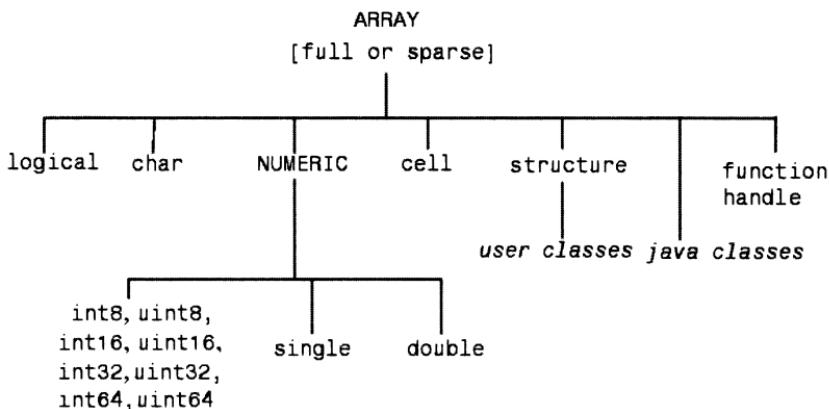


Рис. 2.1. Классы данных в MATLAB

Рассмотрим из всей совокупности классов лишь те классы, которые широко используются для решения задач линейной алгебры и аналитической геометрии, а также классы, используемые для хранения разнородной информации.

2.1. АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЙ КЛАССЫ ДАННЫХ

Арифметические классы данных (NUMERIC) и логический класс (logical) являются наиболее используемыми классами данных в системе MATLAB по сравнению с другими классами.

2.1.1. Арифметические классы данных

Арифметические классы данных делятся на *целые* и *вещественные*.

Данные целого класса могут быть числами со знаком (классы **int8**, **int16**, **int32** и **int64**) и числами без знака (**uint8**, **uint16**, **uint32** и **uint64**). Число в имени класса определяет размер памяти, отводимый для хранения числа в байтах. Например, целое со знаком **int8** – это однобайтовое число.

Данные вещественного класса могут быть числами *одинарной* (single – 4 байта) и *двойной* (double – 8 байт) *точности*.

Все числа при выполнении арифметических операций должны быть числами двойной точности. В противном случае в командное окно будет выведено сообщение об ошибке. Остальные арифметические классы используются для эффективного хранения данных на диске. Таким образом, любая переменная, перед выполнением операции над ней, должна быть преобразована к классу double.

Пример 2.1. Приравнить переменной k значение 2. Преобразовать созданную переменную к классу int8. Увеличить переменную на два. Преобразовать переменную обратно к классу double и опять увеличить на два.

Решение:

```
>> k = 2;
>> k = int8(k);
>> k = k+2;
??? Error using ==> +
Function '+' is not defined for values of class 'int8'.
(Ошибка выполнения операции сложения,
функция «+» не определена для значений класса int8).
>> k = double(k);
>> k = k+2
k =
4
```

При создании переменная k по умолчанию получает класс double (class – double array, size 1×1). Для преобразования переменной k другому классу необходимо вызвать функцию, имя которой соответствует целевому классу.

Все операции в MATLAB реализуются посредством функций. Например, для выполнения операции сложения MATLAB вызывает функцию plus() (вместо k+2 выполняется функция plus(k, 2)). Такая идеология реализована для всех операций: каждая операция имеет соответствующую ей функцию, к которой и обращается MATLAB при ее выполнении.

Перечень основных встроенных в систему MATLAB функций для арифметических классов приведен в табл. 2.1.

Наличие точки в операторах связано с тем, что все данные в системе MATLAB – массивы. Для массивов существуют две различные операции, связанные с умножением, делением и возведением в степень (например, умножение матриц и поэлементное умножение матриц).

Таблица 2.1

Основные математические функции

<i>1. Арифметические функции</i>	
plus()	сложение «+»
uplus()	унарный плюс «+»
minus()	вычитание «-»
uminus()	унарный минус «-»
times()	поэлементное умножение «.*»
mtimes()	матричное умножение «*»
power()	поэлементное возвведение в степень «.^»
mpower()	матричное возвведение в степень «^»
ldivide()	левое (справа налево) поэлементное деление «.\»
mldivide()	левое матричное деление «\»
rdivide()	правое (слева направо) поэлементное деление «./»
mrdivide()	правое матричное деление «/»
<i>2. Тригонометрические и гиперболические функции</i>	
sin(x), cos(x), tan(x), cot(x)	синус, косинус, тангенс и котангенс (аргумент x – в радианах)
sec(x), csc(x)	секанс и косеканс
asin(x), acos(x), atan(x), acot(x), asec(x), acsc(x)	обратные тригонометрические функции (возвращают результат в радианах)
sinh(x), cosh(x), tanh(x), coth(x), sech(x), csch(x)	гиперболические тригонометрические функции $(\sinh(x) = \frac{e^x - e^{-x}}{2}, \cosh(x) = \frac{e^x + e^{-x}}{2})$
asinh(x), acosh(x), atanh(x), acoth(x), asech(x),acsch(x)	обратные гиперболические тригонометрические функции
<i>3. Логарифмы, степенные функции, экспонента</i>	
exp(x)	экспоненциальная функция e^x
log(x)	натуральный логарифм ($\log_e x = \ln x$)
log10(x)	десятичный логарифм ($\log_{10} x = \lg x$)
log2(x)	логарифм по основанию 2 ($\log_2 x$)

<code>sqrt(x)</code>	квадратный корень (квадратный корень может быть извлечен и из отрицательного числа), например: <code>>> a=sqrt(-1); % a= 0+1i - мнимая единица</code> <code>>> a=sqrt(-5); % a= 0+2.2361i</code>
4. Остаток от деления*	
<code>mod(a,b)</code>	возвращает остаток от деления в соответствии с определением остатка, принятым в алгебре: <code>>> r=mod(5,3); %r=2, r=5-3</code> <code>>> r=mod(-5,3); %r=1, r=-5-(-6)=-5+6=1</code> <code>>> r=mod(-5,-3); %r=-2, r=-5-(-3)</code> <code>>> r=mod(5,-3); %r=-1, r=5-6</code>
<code>rem(a,b)</code>	<code>>> r=rem(-5,3); %r=-2, r=-5-(-3)</code> <code>>> r=rem(5,-3); %r=2, r=5-3</code> <code>>> r=rem(5,3); %r=2, r=5-3</code> <code>>> r=rem(-5,-3); %r=-2, r=-5-(-3)</code>
5. Округление	
<code>fix(x)</code>	округление до ближайшего целого в сторону нуля
<code>floor(x)</code>	округление до ближайшего целого слева от x
<code>ceil(x)</code>	округление до ближайшего целого справа от x
6. Знак числа и абсолютное значение	
<code>sign(x)</code>	возвращает знак числа (<code>sign(0)=0</code>)
<code>abs(x)</code>	возвращает абсолютное значение числа x

***Примечание.** Напомним понятие остатка от деления, принятое в алгебре: a – делимое, b – делитель, q – неполное частное, r – остаток; $a = b \cdot q + r$; при этом $a \in Z; b \in N; q \in Z; r \in N; 0 \leq r < b$, где N – множество всех натуральных чисел, Z – множество всех целых чисел.

Именно так принято в алгебре, и именно относительно такого понимания остатка сформулирована теорема о единственности пары $q \in Z$ и $r \in N$ для любой пары $a \in Z$ и $b \in N$ (теорема о делении с остатком).

Указанному пониманию остатка соответствует функция `mod()`, в случае если делитель (второй параметр функции) положителен.

В случае, если делитель отрицателен, для понимания результата достаточно мысленно повернуть числовую ось на 180° .

В отличие от функции `mod()` функция `rem()` построена на определении частного по отношению к ближайшему целому в направлении нуля.

2.1.2. Логический класс данных

Данные логического класса могут принимать одно из двух значений: true (истина) или false (ложь). «Истина» ассоциируется с единицей, а «ложь» – с нулем. Переменная логического класса занимает в памяти 1 байт.

Значения логического класса получаются как результат операций отношения (сравнения), список которых приведен в табл. 2.2. Операции отношения наиболее часто применяются в управляющих структурах (условный оператор и оператор повтора с условием).

Так же как и для арифметических операций, логические операции реализованы с помощью функций. В этом смысле знаки операций – условности. При их помощи логические выражения записываются, но реализуются они посредством вызова соответствующих функций.

Таблица 2.2

Операции сравнения

Операция	Функция	Описание					
<code>==</code>	<code>eq()</code>	Равно: <code>>> A=[1,2,3,4,5]; B=[5 4 3 2 1];...</code> <code>C = A==B % eq(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </table>	0	0	1	0	0
0	0	1	0	0			
<code>~=</code>	<code>ne()</code>	Не равно: <code>>> C = A~=B % ne(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> </table>	1	1	0	1	1
1	1	0	1	1			
<code><</code>	<code>lt()</code>	Меньше: <code>>> C= A<B % lt(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	1	1	0	0	0
1	1	0	0	0			
<code>></code>	<code>gt()</code>	Больше: <code>>> C= A>B % gt(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table>	0	0	0	1	1
0	0	0	1	1			
<code><=</code>	<code>le()</code>	Меньше или равно: <code>>> C= A<=B % le(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> </table>	1	1	1	0	0
1	1	1	0	0			
<code>>=</code>	<code>ge()</code>	Больше или равно: <code>>> C= A>=B % ge(A,B)</code> <code>C =</code> <table style="margin-left: 200px;"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> </table>	0	0	1	1	1
0	0	1	1	1			

В табл. 2.3 рассмотрены логические операции, которые могут быть поэлементными (element-wise), побитовыми (bit-wise) и короткими (short-circuit). При поэлементных операциях булевые функции вычисляются для соответствующих элементов массива. Побитовые операции выполняются над множеством натуральных чисел. Максимальное натуральное число, которое может быть представлено в системе MATLAB, можно посмотреть с помощью команды `>> bitmax`. Смысл коротких операций состоит в том, что процесс вычисления логического выражения прекращается, если результат его уже ясен.

Таблица 2.3

Логические операции

Операция	Функция	Описание
<i>1. Поэлементные операции</i>		
<code>~</code>	<code>not()</code>	Логическое отрицание «НЕ»: <code>>> A=[0 0 2 4]; B=[0 2 0 4]; ~B</code> <code>ans =</code> <code>1 0 1 0</code>
<code>&</code>	<code>and()</code>	Поэлементное «И» (конъюнкция): <code>>> A&B</code> <code>ans =</code> <code>0 0 0 1</code>
<code> </code>	<code>or()</code>	Поэлементное «ИЛИ» (дизъюнкция): <code>>> A B</code> <code>ans =</code> <code>0 1 1 1</code>
	<code>xor()</code>	«Исключающее ИЛИ» (сумма по модулю 2): <code>>> xor(A,B)</code> <code>ans =</code> <code>0 1 1 0</code>
<i>2. Побитовые операции</i>		
	<code>bitand()</code>	Побитовое «И»: <code>>> A=7; B=11; bitand(A,B) %7=<0111>, 11=<1011></code> <code>ans =</code> <code>3</code>
	<code>bitor()</code>	Побитовое «ИЛИ»: <code>>> bitor(A,B)</code> <code>ans =</code> <code>15</code>
	<code>bitxor()</code>	Побитовое «исключающее ИЛИ»: <code>>> bitxor(A,B)</code> <code>ans =</code> <code>12</code>

Окончание табл. 2.5

Операция	Функция	Описание
<i>3. Короткие операции</i>		
&&		Позлементное короткое (short-circuit) «И» $>> x>=2 \&\& x<=6$ Если x не удовлетворяет первому условию ($x>=2$), то выполнение команды прекращается, так как в любом случае в итоге будет логический ноль
		Позлементное короткое «ИЛИ» $>> x<=2 \mid\mid x>=6$ Если после выполнения первого сравнения ($x<=2$) получается логическая единица, то вычисления прекращаются

Пример 2.2. Определить, находятся ли точки $M_1(0.5;0.5)$ и $M_2(0.8;0.8)$ внутри логической области, изображенной на рис. 2.2.

Решение:

```
>> x=[0.5 0.8]; y=[0.5 0.8];
>> z=(x.^2+y.^2)>=1;
>> z=and(z, and(abs(x)<=1, abs(y)<=1))
z =
    0      1
```

Точка M_1 не попадает в заштрихованную область ($z=0$), а точка M_2 располагается внутри заштрихованной области ($z=1$).

Большой вклад в теорию логических вычислений внесли английский логик Джордж Буль и шотландский математик Огастес (Августус) де Морган.



Огастес де Морган (Augustes De Morgan) родился 27 июня 1806 года в Мадуре (Индия). Основные труды по алгебре, математическому анализу и математической логике. Один из основателей формальной алгебры. Учитель Ады Августы Лавлейс (дочери Джорджа Гордона Байрона), которая считается первым программистом и в честь которой назван один из языков программирования. В 1847 году опубликовал трактат «Формальная логика, или исчисление выводов необходимых и возможных», где изложил элементы логики высказываний и дал первую развитую систему логики отношений. В книге «Тригонометрия и двойная алгебра» (1849) развил мысль

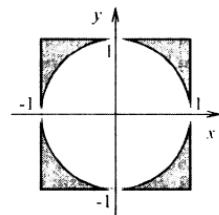
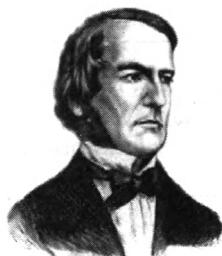


Рис. 2.2. Логическая область

Уильяма Гамильтона о распространении идей символьной алгебры на исчисление комплексных величин. Основатель и первый президент (1865) Лондонского математического общества. Умер 8 марта 1871 года в Лондоне.



Джордж Буль (George Boole) родился 2 ноября 1815 года в Линкольне (Англия). В 1847 году опубликовал статью по началам математической логики – «Математический анализ логики, или Опыт исчисления дедуктивных умозаключений», а в 1848 году – «Логическое исчисление». В 1849 году в городе Корк (Ирландия) открылось новое высшее учебное заведение – Квинз колледж; по рекомендации коллег-математиков Буль получил там профессуру, которую сохранил до своей смерти в 1864 году. В 1854 году появился главный его труд «Исследование законов мышления, на которых основаны математические теории логики и вероятностей», в котором он свел логику к алгебраической форме. Ввел в употребление операции логического отрицания «NOT», логического умножения «AND» (конъюнкция) и логического сложения «OR» (дизъюнкция). Логическое исчисление Буля получило название булевой алгебры. Одна из четырех дочерей, Этель Лилиан Буль, в замужестве Войнич, – автор популярного романа «Овод». Умер 8 декабря 1864 года в городе Корк.

2.1.3. Приоритет операций

В выражениях могут использоваться любые операции, а также их комбинации. При этом операции в выражении выполняются последовательно, в соответствии с установленным для них *приоритетом*. Последовательность выполнения операций может быть изменена с помощью круглых скобок. Рассмотрим приоритеты операций, начиная с наивысшего.

1. Наивысшим приоритетом обладают операции транспонирования (\cdot'), транспонирования с комплексным сопряжением ($'$), поэлементного возведения в степень (\cdot^{\wedge}) и матричного возведения в степень ($^{\wedge}$).
2. Следующий уровень приоритета – *унарные операции* ($+$, $-$, логическое отрицание (\sim)). Унарными называются операции, применяемые по отношению к одному операнду. Например, следующая операция является унарной: -3 .
3. *Мультипликативные операции* (умножение ($\cdot *$), правое деление ($\cdot /$), левое деление ($\cdot \backslash$), матричное умножение ($*$), матричное правое деление ($/$), матричное левое деление (\backslash)).
4. *Аддитивные операции* (сложение ($+$) и вычитание ($-$)).
5. Оператор « $::$ ».

6. Операторы отношения ($>$, \geq , $<$, \leq , $=$, $\sim=$).
7. Поэлементное «И» ($\&$).
8. Поэлементное «ИЛИ» (\mid).
9. Сокращенное «И» ($\&\&$).
10. Сокращенное «ИЛИ» ($\mid\mid$).

Пример 2.3. Вычислить арифметическое выражение

$$x = 3 + 2 \frac{a^2}{3} (a+1), \text{ где } a = -3.$$

Решение:

```
>> a=-3; x=3+2*a^2/3*(a+1)
x =
-9
```

При вычислении значения переменной x выполняются следующие шаги:

- a) вычисляется выражение в круглых скобках $(a+1) = -2$;
- б) переменная a возводится в квадрат $a^2 = 9$;
- в) полученное значение после возведения в квадрат последовательно умножается на два, делится на три и умножается на результат выражения в круглых скобках $2 * 9 / 3 = 6$;
- г) полученное в предыдущем пункте число прибавляется к трем $3 + 6 = 9$.

Пример 2.4. Вычислить логическое выражение $a \oplus b$ («исключающее ИЛИ») с помощью логических операций «И» и «ИЛИ», где $a = \text{true}$ и $b = \text{true}$.

Решение:

```
>> a=true; b=true; a_xor_b= ~a&b | a&~b
a_xor_b =
0
```

При вычислении переменной a_xor_b выполняются следующие шаги:

- а) последовательно выполняется инвертирование переменных $\sim a$ и $\sim b$;
- б) с полученными после инвертирования логическими значениями последовательно выполняется операция логического «И» в выражении $\sim a \& b$, а затем в выражении $a \& \sim b$;
- в) полученные в предыдущем пункте логические значения объединяются с помощью операции логического «ИЛИ».

2.2. СИМВОЛЬНЫЙ КЛАСС ДАННЫХ

Наряду с числовыми и логическими данными система MATLAB позволяет работать с данными, которые являются последовательностью символов. Переменные, содержащие только символьные данные, представляют собой *переменные символьного класса* (char).

2.2.1. Создание переменных символьного класса

Для создания переменной символьного класса достаточно присвоить ей значение, заключенное в апострофы.

Пример 2.5. Присвоить переменной text выражение «Hello, Word!».

Решение:

```
>> text = 'Hello, Word!'
text =
Hello, Word!
```

После выполнения команды в рабочей области MATLAB будет создана новая переменная символьного класса с именем text, которая состоит из 11 символов и занимает в памяти 22 байта. Таким образом, для представления одного символа в переменной символьного класса отводится 2 байта.

Система MATLAB позволяет создавать многострочные переменные символьного класса, являющиеся двухмерными массивами символов. При объединении символьных массивов в массивы более высокой размерности, т.е. в массивы строк, их размерность должна быть одинаковой. В противном случае в командное окно будет выведено сообщение об ошибке. Это связано с тем, что матрицы в системе MATLAB должны быть прямоугольными или, в частном случае, квадратными.

Пример 2.6. Создать символьную переменную student, в первой строке которой размещается фамилия студента, во второй – его имя.

Решение:

```
>> student = ['Иванов      '; 'Александр']
student =
Иванов
Александр
>> first_name='Александр'; last_name='Иванов      ';
>> student=[last_name; first_name]
student =
Иванов
Александр
>> name='Александр'; student=char('Иванов', name)
student =
Иванов
Александр
```

Пример был решен тремя способами. Для вертикального объединения строк использовалась последовательность символов «[;]». В результате переменная student является двухмерным массивом символов, состоящим из двух строк и девяти столбцов. При задании переменной student в первой строке для выравнивания числа символов в строках были добавлены три

пробела. Система MATLAB позволяет использовать при объединении строк в массивы символов имена других переменных, которые сами содержат массивы символов.

Система MATLAB позволяет объединять строки, содержащие разное количество символов. Эта операция реализуется с помощью функции `char()`, которая автоматически выравнивает длины строк, добавляя, где необходимо, пробелы в конец строки.

Пример 2.7. Создать символьную переменную `student`, в которой на одной строке размещаются фамилия и имя студента.

Решение:

```
>> student=['Иванов', ' ', 'Александр']
student =
Иванов Александр
>> surname='Иванов'; student=strcat(surname, ' Сергей')
student =
Иванов Сергей
```

Для горизонтального объединения строк можно использовать специальные символы «[,]» либо функцию `strcat()`. Для разделения фамилии и имени перед именем был добавлен пробел, так как функция `strcat()` удаляет все концевые и одиночные пробелы во входных параметрах.

2.2.2. Преобразование переменной из символьного класса в арифметический класс

Переменная символьного класса в системе MATLAB является массивом значений из таблицы кодировки символов ASCII (*American Standard Code for Information Interchange*), которые отображаются в виде соответствующих символов (табл. 2.4). Первая часть (0...127) символов таблицы является стандартной, а вторая часть (128...255) изменяется в зависимости от страны.

Таблица 2.4

Кодировка символов ASCII

32	48	0	64	@	80	P	96	`	112	p	128	Ђ	144	Ђ	160		176	°	192	À	208	P	224	à	240	P	
33	!	49	1	65	À	81	Q	97	à	113	q	129	Ѓ	145	‘	161	Ў	177	±	193	Б	209	C	225	б	241	C
34	"	50	2	66	В	82	R	98	њ	114	r	130	,	146	’	162	Ӧ	178	I	194	В	210	T	226	в	242	T
35	#	51	3	67	С	83	S	99	с	115	s	131	Ӯ	147	“	163	ҟ	179	i	195	Г	211	у	227	г	243	у
36	\$	52	4	68	D	84	T	100	đ	116	t	132	„	148	”	164	¤	180	г	196	đ	212	Ф	228	đ	244	Ф
37	%	53	5	69	E	85	U	101	е	117	u	133	...	149	•	165	ѓ	181	µ	197	Е	213	X	229	е	245	X
38	&	54	6	70	F	86	V	102	f	118	v	134	†	150	-	166	!	182	ќ	198	ќ	214	Ц	230	ж	246	ц
39	'	55	7	71	G	87	W	103	g	119	w	135	‡	151	-	167	§	183	·	199	З	215	Ч	231	з	247	Ч

40 (56 8	72 н	88 х	104 г	120 х	136 е	152 0	168 ё	184 е	200 и	216 ѿ	232 и	248 ѿ
41)	57 9	73 т	89 ў	105 і	121 ў	137 %	153 тм	169 @	185 #	201 й	217 ѿ	233 ѿ	249 ѿ
42 *	58 :	74 ж	90 з	106 ж	122 з	138 Ѽ	154 Ѽ	170 е	186 е	202 к	218 Ѽ	234 к	250 Ѽ
43 +	59 ;	75 к	91 [107 к	123 {	139 <	155 >	171 «	187 «	203 л	219 ѿ	235 л	251 ѿ
44 ,	60 <	76 л	92 \	108 л	124	140 Ѽ	156 Ѽ	172 -	188 ж	204 м	220 Ѽ	236 м	252 Ѽ
45 -	61 =	77 м	93]	109 т	125]	141 Ѱ	157 Ѱ	173 -	189 с	205 н	221 э	237 н	253 э
46 .	62 >	78 н	94 ^	110 п	126 ~	142 в	158 г	174 @	190 с	206 о	222 ѿ	238 о	254 ѿ
47 /	63 ?	79 о	95 _	111 о	127 0	143 ѿ	159 ѿ	175 Ѽ	191 і	207 п	223 я	239 п	255 я

Первые 32 символа этой таблицы являются непечатаемыми и используются для специальных целей (табуляция, окончание строки и т.д.).

Пример 2.8. Создать переменную символьного класса student, в которой содержится фамилия студента, и просмотреть значения символов, ее образующих.

Решение:

```
>> student='Иванов'; double(student)
ans =
    200    226    224    237    238    226
```

Пример 2.9. Создать массив чисел и просмотреть их символьное представление.

Решение:

```
>> surname = char([200    226    224    237    238    226])
surname =
Иванов
```

Для просмотра символьного представления массива чисел, согласно таблице ASCII, использовалась функция char().

В том случае, если в переменной символьного класса содержится числовое значение, которое необходимо присвоить переменной арифметического класса, следует использовать функции, приведенные в табл. 2.5.

В именах функций в табл. 2.5 цифра «2» используется как сокращение предлога «to» из-за одинакового произношения в английском языке этого предлога и числительного «2».

Просмотреть полную таблицу кодировки ASCII в командном окне MATLAB можно с помощью следующей команды:

```
>> [int2str([0:255]') blanks(256)' char([0:255]')]
```

Функция blanks() использовалась для формирования строки пробелов, длина которой определялась входным параметром. Апострофы использовались для преобразования строки в столбец.

Таблица 2.5

**Функции преобразования между строковым
и арифметическими классами**

Функция	Описание
int2str()	<p>Преобразует число или переменную класса double в строку:</p> <pre>>> a=2; [int2str(a), ' ', int2str(5)] ans = 2 5</pre> <p>Если входной параметр является вещественным числом, то происходит округление этого числа до ближайшего целого с помощью функции round() с последующим преобразованием:</p> <pre>>> a=2.4; [int2str(a), ' ', int2str(5.5)] ans = 2 6</pre> <p>Имя функции является сокращенной записью словосочетания integer to string</p>
num2str()	<p>Преобразует число или переменную класса double в строку, содержащую вещественное число с указанным количеством цифр для мантиссы (по умолчанию 4):</p> <pre>>> a=2.426; [num2str(a,3), ' ', num2str(5.55643)] ans = 2.43 5.5564</pre> <p>Имя функции представляет собой сокращенную запись словосочетания number to string</p>
mat2str()	<p>Преобразует числовую матрицу в ее вычисляемое символьное представление (более подробную информацию по работе с матрицами можно найти в § 3.2):</p> <pre>>> a=[2.1,1.2;1.5,5.2]; mat2str(a) ans = [2.1 1.2;1.5 5.2]</pre> <p>Данное символьное представление матрицы удобно использовать в качестве входного параметра функции eval().</p> <p>Имя функции – сокращенная запись matrix to string</p>
str2double()	<p>Преобразует строку в число двойной точности:</p> <pre>>> str='1.234'; str2double(str) ans = 1.2340</pre> <p>Строка может содержать научную запись числа:</p> <pre>>> str='-12.34e-01'; str2double(str) ans = -1.2340</pre>

Функция	Описание
str2num()	<p>Преобразует символьное представление матрицы или вектора в их числовое представление:</p> <pre>>> str='[1 2]'; str2num(str) ans = 1 2</pre> <p>Помимо преобразования матрицы из символьного представления в числовое, функция может использоваться для вычисления простых выражений, так как обращается к функции eval():</p> <pre>>> str = '1+2/3'; str2num(str) ans = 5/3</pre> <p>Если строка содержит только число, то следует использовать функцию str2double()</p>

Система MATLAB поддерживает и другие операции со строковыми переменными, которые осуществляют сравнение и замену строк, поиск части или всей строки в составе других строк, а также изменение регистров символов.

2.2.3. Вычисление символьных выражений

Помимо текстовой информации и чисел строки могут содержать различные выражения. Символьные выражения можно формировать по ходу выполнения программы в зависимости от тех или иных условий, что привносит дополнительную гибкость в работу программы. Так как символьное выражение является строкой, то при его формировании используются правила работы со строками, рассмотренные выше. Вычисление символьного выражения реализуется с помощью функции eval().

Пример 2.10. Вычислить символьные выражения

```
'(6+10)^(1/4)', '(a-5)/a' и '(b-2)/2',  
где a = 10.
```

Решение:

```
>> str='(6+10)^(1/4)'; eval(str)
ans =
    2
>> a=10; str='(a-5)/a'; eval(str)
ans =
    1/2
>> str='(b-2)/b'; eval(str)
??? Undefined function or variable 'b'.
```

Если в символьных выражениях содержатся переменные, то они должны быть переменными арифметического класса двойной точности. В том случае, если переменная, используемая в символьном выражении, не определена, то в командное окно выводится сообщение об ошибке.

Если переменные, используемые в символьных выражениях, являются переменными символьного класса, то во время выполнения функции eval() их содержимое преобразуется посредством вызова функции double() в матрицу значений кодов ASCII с последующим вычислением выражения.

Пример 2.11. Вычислить символьное выражение 'a-1',
где a='100'.

Решение:

```
>> a='100'; str='a-1'; eval(str)
ans =
```

48

47

47

Три числа в результате получились из-за того, что переменная a состоит из трех символов.

2.3. МАССИВ СТРУКТУРЫ

В различных языках программирования для хранения в одной переменной разнородной информации используются специальные типы данных. В языке Pascal примером такого типа данных является тип запись (record), в языке C – структура (struct). Язык MATLAB также поддерживает работу с переменными, в которых располагаются данные различных классов. Хранение разнородных данных в одной переменной в системе MATLAB реализуется с помощью класса данных, представляющего собой массив структуры (structure). Данные хранятся в отдельных контейнерах, называемых *полями* (fields). При этом структуры, состоящие из одинаковых полей и содержащие в них различные данные, могут образовывать массивы любой размерности. Переменные рассматриваемого класса широко используются при работе с базами данных с помощью пакета расширения Database Toolbox.

Данный класс берется за основу при создании пользовательских объектно-ориентированных классов данных (user classes), которые можно использовать для расширения стандартных возможностей системы MATLAB. Таким образом, с помощью системы MATLAB можно вести разработку программ, используя объектно-ориентированный подход.

2.3.1. Организация массива структуры в системе MATLAB

В системе MATLAB можно использовать два подхода при организации массива структуры: *матричный* (рис. 2.3,а) и *поэлементный* (рис. 2.3,б). Подходы различаются в размещении элементов структуры. В первом случае данные об именах, фамилиях, датах рождений и оценках всех студентов размещаются в одном соответствующем поле в матричном виде. При этом первые элементы в каждом из полей несут информацию о первом студенте, вторые элементы – о втором студенте и т.д. Этот подход является предпочтительным, когда необходимо выполнять операции над всеми элементами структуры. Во втором случае данные о студентах находятся в соответствующих полях каждого студента. Данный подход представляется предпочтительным, когда необходимо обрабатывать данные отдельных элементов массива структуры.

```
student
|
|           Иван
|- first_name = Петр
|           Семен
|
|           Иванов
|- last_name  = Петров
|           Семенов
|
|           1985
|- year      = 1986
|           1985
|
|           4
|- physics   = 5
|           5
|
|           5
|- mathematics = 4
|           5
```

а) матричная

```
student
|
|- student(1)
|   |- first_name = Иван
|   |- last_name  = Иванов
|   |- year      = 1985
|   |- physics   = 4
|   |- mathematics = 5
|
|- student(2)
|   |- first_name = Петр
|   |- last_name  = Петров
|   |- year      = 1986
|   |- physics   = 5
|   |- mathematics = 4
|
|- student(3)
|   |- first_name = Семен
|   |- last_name  = Семенов
|   |- year      = 1985
|   |- physics   = 5
|   |- mathematics = 5
```

б) поэлементная

Рис. 2.3. Организация структуры

Система MATLAB позволяет создавать также *иерархические структуры*, которые состоят из нескольких уровней (рис. 2.4). Так как большинство стандартных функций системы MATLAB поддерживают работу только с одним верхним уровнем иерархии, то по возможности следует избегать

использования вложенных структур. Если по каким-либо причинам необходимо использовать несколько уровней иерархии, то функции, которые работают с произвольным их числом, можно найти в обществе пользователей системы MATLAB и пакета SIMULINK по обмену файлами по адресу <http://www.mathworks.com/matlabcentral/> в разделе file exchange. На этой странице можно найти свободно распространяемые программы на языке MATLAB в исходниках. Примером одной из таких программ является программа Марка Джобсе (Mark Jobse), расположенная в файле `display_structure.m`, для просмотра иерархических структур.

```
st(1)
  first_name = Иван
  last_name  = Иванов
  year       = 1985
  first_semester
    physics = 4
    mathematics = 5
```

Р и с . 2.4. Иерархическая структура

2.3.2. Создание переменных класса массив структуры

Создать переменную класса *массив структуры* можно двумя способами:

- используя разделитель «.»;
- с помощью функции-конструктора `struct()`.

Пример 2.12. Используя первый способ создания массива структур, создать переменную `student` размерности 1×1 , которая содержит два поля – `surname` и `name` – с фамилией и именем студента соответственно

Решение:

```
>> student.surname='Иванов'; student.name='Иван'
student =
surname: 'Иванов'
name: 'Иван'
```

Переменная `student` занимает в памяти 268 байтов, из которых 124 байта отводится по умолчанию под каждое поле без учета размера данных этого поля. При этом первое поле содержит 12 байтов информации, а второе – 8 байтов. Переменная класса массив структуры в окне просмотра рабочей области обозначается с помощью пиктограммы  . Следовательно, с помощью первого способа создания переменной класса массив структуры можно не только создавать новую переменную, но и добавлять новые поля к уже созданной переменной.

Пример 2.13. Создать переменную `student`, которая состоит из трех полей: `first_name`, где находится имя студента, `last_name` с его фамилией и `year`, содержащим дату его рождения.

Решение:

```
>> student = struct('last_name', 'Иванов', ...
'first_name', 'Иван', 'year', 1985)
student =
    last_name: 'Иванов'
    first_name: 'Иван'
    year: 1985
```

При создании переменной с помощью функции-конструктора `struct()` нечетными входными аргументами в функцию являются имена полей, заключенные в апострофы, а четными – их соответствующие значения. После задания команды поля `first_name` и `last_name` содержат строковые значения, а поле `year` – числовое значение двойной точности. Переменная `student` занимает в памяти 400 байтов. Почему?

Таким образом, при использовании первого способа задания переменной класса массив структуры переменная формируется с помощью последовательного добавления полей, а во втором случае все поля, образующие структуру, формируются сразу одной командой.

Во время разработки структуры возможно применение комбинированного подхода, когда сначала формируются все необходимые на текущий момент поля, а затем, по мере необходимости, происходит добавление полей к разрабатываемой структуре.

Пример 2.14. Используя первую модель организации структуры, создать учетные записи для двух студентов в переменной `stud`.

Решение:

```
>> student = struct('last_name', 'Иванов', ...
'first_name', 'Иван', 'year', 1985);
>> student.physics=4; student.mathematics=5;
>> stud.last_name=char(student.last_name,'Семенов');...
stud.first_name=char(student.first_name,'Семен');...
stud.year=vertcat(student.year,1985);...
stud.physics=vertcat(student.physics,5);...
stud.mathematics=vertcat(student.mathematics,5)
stud =
    last_name: [2x7 char]
    first_name: [2x5 char]
        year: [2x1 double]
        physics: [2x1 double]
    mathematics: [2x1 double]
```

При создании новой переменной использовались поля уже расположенной в рабочей области переменной `student`, к которым были добавлены соответствующие данные. В результате получили переменную `stud`, которая содержит пять полей. В каждом из этих полей располагается массив, состоящий из двух строк. В первой строке находятся данные, которые относятся к первому студенту, а во второй строке – данные второго студента. Количество столбцов в массиве зависит от длины имени и фамилии в случае символьных данных. В момент добавления символьных данных массив поля увеличивается до необходимых размеров, чтобы сохранить равенство строк в матрице. Поля, содержащие арифметические данные, в приведенном примере состоят из одного столбца.

Пример 2.15. С помощью второй модели организации структуры создать учетные записи для двух студентов в переменной `student`.

Решение:

```
>> student = struct('last_name', 'Иванов', ...
'first_name', 'Иван', 'year', 1985);
>> student.physics=4; student.mathematics=5;
>> student(2) = struct('last_name','Семенов',...
'first_name','Семен', 'year', 1985, ...
'physics', 5, 'mathematics', 5)
student =
1x2 struct array with fields:
    last_name
    first_name
    year
    physics
    mathematics
```

При создании следующего элемента массива структуры необходимо после имени переменной указать в круглых скобках соответствующий номер элемента и заполнить все поля с помощью функции-конструктора или последовательно с помощью разделителя. В первом элементе располагаются данные, относящиеся к первому студенту, а во втором элементе – данные второго студента. В командное окно выводится результат выполнения команды, в котором указываются новая размерность переменной и названия полей. В том случае, если при использовании функции-конструктора одно из полей не будет указано, то в командное окно будет выведено сообщение об ошибке (`??? Subscripted assignment between dissimilar structures. - ???` Индексное назначение между различными структурами). В случае использования разделителя в неуказанных полях структуры будут располагаться массивы нулевой размерности (0×0), которые обозначаются [].

Пример 2.16. Создать структуру st, которая состоит из двух уровней иерархии. На первом уровне иерархии размещаются четыре поля: last_name, first_name, year и first_semester, в которых располагаются фамилия, имя, год рождения и оценки студентов за первый семестр соответственно. Поле first_semester содержит внутренние поля – physics и mathematics, в которых располагаются оценки студентов за первый семестр по физике и математике соответственно. В соответствующие поля ввести следующие значения: Иванов Иван, 1985 года рождения, физика – 4, высшая математика – 5.

Решение:

```
>> st=struct('last_name','Иванов',...
'first_name','Иван', 'year', 1985, ...
'first_semester', struct('physics',4,'mathematics',5))
st =
    last_name: 'Иванов'
    first_name: 'Иван'
        year: 1985
    first_semester: [1x1 struct]
```

После задания команды поле first_semester содержит структуру, образуя тем самым второй уровень иерархии.

2.3.3. Просмотр переменных класса массив структуры

Просмотреть имена полей можно либо с помощью задания имени переменной в качестве команды, либо с помощью функции fieldnames().

Пример 2.17. Просмотреть переменные student и stud класса массив структуры, созданные в примерах 2.14 и 2.15.

Решение:

```
>> student
student =
1x2 struct array with fields:
    last_name
    first_name
    year
    physics
    mathematics
>> fieldnames(stud)
ans =
    'last_name'
    'first_name'
    'year'
    'physics'
    'mathematics'
```

Во втором случае результат записывается в переменную `ans`, которая представляет собой массив ячеек размера 5×1 . Для просмотра вложенных полей необходимо указать полный путь к соответствующему полю с указанием номера элемента массива, в случае нескольких элементов в массиве.

Пример 2.18. Просмотреть в переменной `st`, созданной в примере 2.16, имена полей, располагающихся в поле `first_semester`.

Решение:

```
>> st.first_semester
ans =
    physics: 4
    mathematics: 5
>> fieldnames(st.first_semester)
ans =
    'physics'
    'mathematics'
```

В первом случае выводятся имена полей и их значения, так как ни одно из этих полей не содержит вложенную структуру. При этом результат записывается в переменную `ans` класса массив структуры. Если не указывать номер элемента переменной класса массив структуры, то будут выведены имена полей и их значения, содержащиеся в указанном поле, для всех элементов массива. Переменная `ans` будет содержать данные последнего поля. Во втором случае выводятся только имена полей, входящих в поле, которое указано в качестве входного аргумента в функцию `fieldnames()`.

Пример 2.19. Просмотреть значение поля `physics` переменной `st`, созданной в примере 2.16.

Решение:

```
>> getfield(st,'first_semester','physics')
ans =
    4
>> st.first_semester.physics
ans =
    4
```

Для просмотра значения поля можно использовать функцию `getfield()`, которая возвращает значение указанного в качестве входного параметра поля. Первым параметром функции идет имя переменной. Второй и третий параметры функции описывают полный путь к полю `physics`.

Пример 2.20. Вычислить среднюю оценку студента за первый семестр. Использовать данные, которые расположены в переменной `student`, из примера 2.14.

Решение:

```
>> str = [student(1).last_name, ' ', ...
getfield(student(1), 'first_name'), ': ', ...
num2str((student(1).physics + ...
getfield(student(1), 'mathematics'))/2)]
str =
Иванов Иван: 4.5
```

При использовании имен полей в реализации выражений следует обращать особое внимание на совместимость классов данных. В противном случае в командное окно будет выведен ошибочный результат.

2.3.4. Сортировка имен полей

В системе MATLAB имеется возможность *сортировки имен полей* в зависимости от расположения букв в таблице ASCII. Сортировка осуществляется с помощью функции `orderfields()`.

Пример 2.21. Упорядочить имена полей переменной `st` из примера 2.16 по алфавиту.

Решение:

```
>> st = orderfields(st)
st =
    first_name: 'Иван'
    first_semester: [1x1 struct]
    last_name: 'Иванов'
    year: 1985
>> s1=st.first_semester
s1 =
    physics: 4
    mathematics: 5
>> s1=orderfields(s1);
>> st.first_semester = s1;
>> st.first_semester
ans =
    mathematics: 5
    physics: 4
```

Производится сортировка только тех полей, которые расположены на верхнем уровне иерархии. Для сортировки полей, расположенных на нижних уровнях иерархии, их необходимо скопировать в отдельную переменную, произвести сортировку, а затем заменить старое поле новым с уже отсортированными полями.

Вышеописанные действия над полем `first_semester` следует выполнять для всех элементов массива `st`. Таким образом, каждый уровень иерархии следует сортировать отдельно и повторять сортировку столько раз, сколько элементов располагается в массиве.

Система MATLAB позволяет сортировать поля в произвольном порядке. Порядок сортировки полей указывается с помощью второго аргумента функции `orderfields(structure, [f1 f2 .. fn])`, являющегося вектором *перестановок*.

Пример 2.22. Провести сортировку полей переменной `student` из примера 2.14. После сортировки поля должны располагаться в следующем порядке: `year, last_name, first_name, mathematics и physics`.

Решение:

```
>> student=orderfields(student,[3 1 2 5 4])
student =
1x2 struct array with fields:
    year
    last_name
    first_name
    mathematics
    physics
```

При сортировке в произвольном порядке количество элементов вектора перестановок должно совпадать с количеством полей в переменной. В противном случае в командное окно будет выведено сообщение об ошибке:

```
?? Error using ==> orderfields
Permutation vector length must be the same as the number
of fields in the struct.
?? Ошибка использования ==> функции orderfields
```

Длина вектора перестановок должна равняться числу полей в структуре.

2.3.5. Удаление поля из массива структуры

Удаление поля осуществляется с помощью функции `rmfield()`.

Пример 2.23. Удалить из переменной `student` (из примера 2.14) поле `year`.

Решение:

```
>> student = rmfield(student,'year')
student =
1x2 struct array with fields:
    last_name
    first_name
    mathematics
    physics
```

Следует осторожно использовать функцию `rmfield()`, так как она удаляет данные без возможности их восстановления.

В табл. 2.6 представлены функции, используемые во время работы с переменными класса массив структуры.

Таблица 2.6

Функции для работы с массивом структуры

Функция	Описание
struct()	Создание переменной класса массив структуры с указанными в качестве входных параметров полями и их значениями
fieldnames()	Просмотр имен полей, располагающихся в поле, указанном во входном параметре
getfield()	Получение значения поля, указанного в качестве входного параметра
setfield()	Установка нового значения поля, которое указано во входном аргументе
orderfields()	Сортировка полей по алфавиту согласно таблице ASCII или согласно вектору номеров полей, указанных во втором входном аргументе в функцию
rmfield()	Удаление поля из массива структуры

2.4. МАССИВ ЯЧЕЕК

Массивом ячеек называется упорядоченный набор данных разнородных классов. Массивы ячеек могут иметь произвольный размер. На рис. 2.5 представлен двухмерный массив ячеек 2×3 , который содержит массив символь-

$\begin{bmatrix} \text{Гоголь} \\ \text{Николай} \\ \text{Васильевич} \end{bmatrix}$	$\begin{bmatrix} 20 & 3 & 1809 \\ 21 & 2 & 1852 \end{bmatrix}$	Russian Writer — Surname — Name - Patronymic Year
$\begin{bmatrix} \text{Ревизор} \\ \text{Шинель} \\ \text{Идиот} \\ \text{Портрет} \\ \text{Нос} \end{bmatrix}$	$\begin{bmatrix} \text{true} \\ \text{true} \\ \text{false} \\ \text{true} \\ \text{true} \end{bmatrix}$	

Рис. 2.5. Массив ячеек

ных данных 3×10 , массив арифметических данных 2×3 , массив структуры, массив символьных данных 5×7 , массив логических данных 5×1 и массив ячеек 2×3 . Массив ячеек является отличительной чертой системы MATLAB, так как позволяет использовать все преимущества работы с матричными данными. Массивы ячеек следует использовать при формировании массива строк различной длины, так как отпадает необходимость следить за равенством их длин. Массивы ячеек широко используются при создании функций, так как позволяют работать с произвольным числом входных и выходных параметров функции.

2.4.1. Создание массива ячеек

Массив ячеек можно создать двумя способами:

- последовательно присваивая значения ячейкам;
- используя функцию-конструктор `cell()`.

Пример 2.24. Используя первый способ, создать массив ячеек A, изображенный на рис. 2.5. Шестая ячейка содержит пустой массив.

Решение:

```
>> A{1,1}=char('Гоголь','Николай','Васильевич');
>> A{1,2}=[20 3 1809; 21 2 1852];
>> A{1,3}=struct('Russian_Writer', struct('Surname','Гоголь',...
'Name','Николай','Patronymic','Васильевич',...
'Date',[20 3 1809; 21 2 1852]));
>> A(2,1)={char('Ревизор','Шинель','Идиот','Портрет','Нос')};
>> A(2,2)={[true;true;false;true;true]};
>> A(2,3)={}
A =
    [3x10 char]    [2x3 double ]    [1x1 struct]
    [5x7  char]    [5x1 logical]    []
```

Фигурные скобки используются при присваивании данных соответствующим ячейкам. В трех первых командах фигурные скобки находятся слева от знака присваивания, а в трех последних – справа. В первом случае используется *индексация содержимого* (content indexing), а во втором случае – *индексация ячеек* (cell indexing). Первое число в индексе обозначает номер строки, а второе – номер столбца. Значения логического массива связаны с названиями произведений, которые находятся во второй ячейке. Так как автором произведения «Идиот» является Федор Михайлович Достоевский, то соответствующий элемент логического массива имеет значение `false`.

Функция конструктор `cell()` используется для предварительного выделения памяти под пустой массив ячеек указанного размера.

Пример 2.25. Используя второй способ создания массива ячеек, сформировать массив ячеек $B\{2 \times 3\}$ и первой ячейки присвоить массив ячеек из примера 2.24.

Решение:

```
>> B=cell(2, 3);  
>> B{1, 1}=A  
B =  
    {2x3 cell}      []      []  
        []      []      []
```

Следовательно, после создания массива ячеек с помощью функции `cell()` необходимо каждой ячейке присвоить соответствующие данные.

2.4.2. Доступ к данным массива ячеек

Доступ к данным массива ячеек осуществляется либо с помощью индексации содержимого, либо с помощью индексации ячеек. Результатом выполнения индексации содержимого будут данные, которые содержатся в соответствующих ячейках. Следовательно, класс возвращаемого результата соответствует классу данных, содержащихся в указанных ячейках. При выполнении индексации ячеек результатом будет массив ячеек, размер которого соответствует размеру индексов.

Обратиться к ячейке можно и с помощью одного индекса. Ячейки нумеруются по столбцам слева направо.

Пример 2.26. Используя индексацию ячеек и индексацию содержимого, присвоить переменной B третью ячейку массива ячеек A , а переменной C – содержимое третьей ячейки массива ячеек A . Массив ячеек A взять из примера 2.24.

Решение:

```
>> B=A(3) // Круглые скобки  
B =  
    [2x3 double]  
>> C=A{3} // Фигурные скобки  
C =  
    20      3      1809  
    21      2      1852
```

В результате переменная B является массивом ячеек, состоящим из одной ячейки, в которой размещается двухмерный массив арифметических данных. Переменная $C(2 \times 3)$ представляет собой переменную арифметического класса двойной точности. Следовательно, для извлечения данных из соответствующих ячеек необходимо использовать индексацию содержимого, а для создания нового массива ячеек или замены ячейки – индексацию ячеек.

Пример 2.27. Используя логическую индексацию, индексацию содержимого и массив ячеек из примера 2.24, определить, какие из данных произведений написал Н.В. Гоголь, и занести эти произведения в последнюю ячейку массива A. Сформировать предложение: Гоголь Николай Васильевич является автором следующих произведений (перечислить произведения).

Решение:

```
>> A{6}=A{2}(A{2,2},1:end);
>> Str = strcat(A{1,3}.Russian_Writer.Surname,...  
[' ',A{1,1}(2,:)], [' ',A{1,1}(3,:)],...  
' является автором следующих произведений:');
>> Str = char(Str,A{2,3}(1:end,:))
Str =
Гоголь Николай Васильевич является автором следующих
произведений:
Ревизор
Шинель
Портрет
Нос
```

Первая команда заполняет шестую ячейку данными из второй ячейки A{2} согласно значениям логического массива, который располагается на пересечении второй строки и второго столбца. Логический массив используется для указания номеров строк, подлежащих копированию в другую ячейку. Конструкция 1:end формирует массив указателей на все столбцы (символы) в строке. Сокращенной записью этой конструкции является один символ двоеточия – :. В результате выполнения этой команды в шестой ячейке будет находиться массив (4×7) символьного класса. Следующие две команды использовались для горизонтального объединения данных, расположенных в первой ячейке, с добавлением пробелов и вертикального объединения полученного массива символьного класса и данных, находящихся в шестой ячейке.

2.4.3. Удаление ячеек и их содержимого из массива ячеек

В том случае, если необходимо удалить содержимое ячейки, следует использовать индексацию содержимого, с помощью которой указывается порядковый номер ячейки, и символ пустого массива, который присваивается указанной ячейке. Если необходимо удалить ячейку, то следует использовать индексацию ячеек и символ пустого массива.

Пример 2.28. Удалить из массива ячеек A, созданного при решении предыдущего примера, данные из третьей ячейки и вторую строку.

Решение:

```
>> A{3} = []
A =
    [3x10 char]      []      [1x1 struct]
    [5x7  char]     [5x1 logical]   [4x7 char  ]
>> A(2, :) = []
A =
    [3x10 char]      []      [1x1 struct]
```

При удалении ячейки следует помнить, что удалять из массива с сохранением размерности исходного массива можно только целую строку или столбец ячеек. В противном случае массив ячеек изменит свою размерность и станет одномерным массивом (`>> A(3) = []`). Ячейки в полученном массиве будут располагаться согласно принятой нумерации, т.е. сначала ячейки первого столбца, затем второго и т.д.

2.5. ПРОВЕРКА КЛАССА ДАННЫХ

При реализации функций на языке программирования MATLAB часто возникают ситуации, когда необходимо проверить класс переменных, передаваемых в качестве входных параметров в функцию. В табл. 2.7 представлен список функций, использующихся для проверки рассмотренных выше классов переменных.

Таблица 2.7

Функции проверки классов переменной

Функция	Описание
<code>isa(имя_переменной, 'имя_класса')</code>	Возвращает логическую единицу, если класс первого входного аргумента функции совпадает с указанным классом во втором аргументе. В противном случае функция вернет логический ноль. <code>>> a=2; [isa(a,'double'), isa(a,'single')]</code> <code>ans =</code> <code>1 0</code>
<code>iscell()</code>	Возвращает логическую единицу, если входной аргумент является массивом ячеек, и логический ноль – в противном случае: <code>>> A{1,1}='Ячейка N1'; iscell(A)</code> <code>ans =</code> <code>1</code>

Окончание табл. 2.7

Функция	Описание
ischar()	Если входной аргумент есть массив символьного класса, то функция вернет единицу, в противном случае – логический ноль. <code>>> A{1,1}='N1'; [ischar(A), ischar(A{1})]</code> <code>ans =</code> <code> 0 1</code>
islogical()	Возвращает единицу, если входной аргумент – логического класса. В противном случае возвращает ноль. <code>>> c=1; a=2; islogical(a==c)</code> <code>ans =</code> <code> 1</code>
isnumeric()	Если входной аргумент арифметического класса, то – логическая единица. <code>>> a=3; b='a'; [isnumeric(a), isnumeric(b)]</code> <code>ans =</code> <code> 1 0</code>
isstruct()	Если входной аргумент класса массив структуры, то логическая единица, и в противном случае – логический ноль. <code>>> s=struct('Surname','Иванов'); isstruct(s)</code> <code>ans =</code> <code> 1</code>

2.6. ВЫВОДЫ К ГЛАВЕ 2

Во второй главе были рассмотрены классы данных системы MATLAB, которые используются при решении задач линейной алгебры и аналитической геометрии, а также классы данных, при помощи которых можно хранить разнородную информацию. Вне зависимости от класса данных любая переменная рассматривается как массив. В системе MATLAB можно использовать массивы произвольной размерности. Единственным ограничением является равенство элементов в соответствующей размерности. Например, в случае двухмерного массива допускаются матрицы, у которых в каждой строке или в каждом столбце содержится одинаковое число элементов.

Наиболее распространенными классами данных в MATLAB являются арифметические и логический классы данных. Все арифметические операции выполняются над переменными двойной точности, что приводит к высокой точности получаемого результата, а также к увеличению объема памя-

ти, используемого для вычисления результата. С целью уменьшения объема памяти, используемого для хранения данных, значения переменных в случае вещественных чисел можно хранить с одинарной точностью и – в случае целых чисел – в виде целого числа со знаком или без знака с фиксированной разрядностью. Однако при выполнении арифметических операций с использованием этих данных их класс необходимо привести к классу двойной точности.

Данные логического класса получаются после выполнения логических операций и операций сравнения. В ядро системы MATLAB встроены основные логические операции («НЕ», «ИЛИ» и «И»), на базе которых можно реализовать остальные логические операции.

Сложные выражения, состоящие из набора арифметических и/или логических операций, вычисляются слева направо согласно приоритету операций, их образующих.

Переменные символьного класса содержат строки текста и могут быть использованы для ввода текстовых надписей на рисунках. При работе с символьными переменными необходимо следить за равенством числа символов в строках, так как текст представляет собой матрицу, состоящую из символов. В систему MATLAB входят стандартные функции, осуществляющие преобразование переменной из символьного класса в арифметические, и наоборот – из арифметических классов в символьный.

Данные арифметических, символьного и логического классов могут храниться в массиве ячеек или в массиве структуры. Массив структуры хранит данные в отдельных полях. Для получения информации необходимо обратиться к соответствующему полю. Массив ячеек является отличительной особенностью системы MATLAB, так как представляет собой матрицу, элементы (ячейки) которой могут содержать данные различных классов.

Глава 3. РЕШЕНИЕ ЗАДАЧ ВЕКТОРНОЙ АЛГЕБРЫ

Используя встроенные типы данных и стандартные функции, входящие в систему MATLAB, можно решать задачи, встречающиеся в векторной алгебре.

3.1. ВЕКТОРЫ

3.1.1. Определение вектора

Целый ряд физических величин (сила, скорость и т.д.) характеризуется не только числовым значением, но и направлением. Например, при описании движения тела в данный момент времени необходимо указать не только скорость движения тела, но и направление его движения, т.е. направление скорости. Величины, в которых кроме числового значения необходимо также указать их направление, называются векторными.

Вектором называется направленный отрезок, имеющий определенную длину, у которого одна из ограничивающих его точек принимается за начало, а вторая – за конец. Если точка A – начало вектора и точка B – его конец, то вектор обозначается символом \overrightarrow{AB} . Начало вектора называют точкой его приложения. Вектор можно обозначить и малой латинской буквой с чертой над ней (например, \bar{a}).

Длиной вектора \overrightarrow{AB} называется длина отрезка, которая обозначается $|\overrightarrow{AB}|$. Длина вектора \bar{a} обозначается $|\bar{a}|$. Вектор, длина которого равна нулю, называется *нулевым вектором* и обозначается $\bar{0}$. Нулейвой вектор не имеет направления.

Вектор, длина которого равна единице, называется *единичным вектором* и обозначается через \bar{e} . Единичный вектор, направление которого совпадает с направлением вектора \bar{a} , называется *ортом* вектора \bar{a} и обозначается \bar{a}^0 .

Векторы \bar{a} и \bar{b} , расположенные на одной прямой или на параллельных прямых, называются *коллинеарными* и обозначают $\bar{a} \parallel \bar{b}$. Нулейвой вектор считается коллинеарным любому вектору.

Два вектора называются *равными*, если они являются коллинеарными и имеют одинаковые длину и направление. Если векторы \bar{a} и \bar{b} равны, то их обозначают с помощью символа $\bar{a} = \bar{b}$.

Три вектора в пространстве называются *компланарными*, если они лежат в одной плоскости или в параллельных плоскостях. Если среди трех векторов хотя бы один нуевой или два любых являются коллинеарными, то такие векторы компланарны.

Любой вектор может быть разложен единственным способом в базисе. Базисом на плоскости называется пара неколлинеарных векторов, взятых в определенном порядке. Базисом в пространстве называется тройка некомпланарных векторов, взятых в определенном порядке. Наиболее распространены базисы:

ненной формой представления вектора на плоскости и в пространстве является его представление с помощью прямоугольной (декартовой) системы координат, т.е. с помощью базиса из перпендикулярных попарно векторов.

Декартова система координат в пространстве образуется с помощью трех взаимно перпендикулярных осей с общим началом O и одинаковой масштабной единицей. Оси упорядочены, т.е. определено, какая из осей считается первой (она называется осью абсцисс и обозначается Ox), какая – второй (ось ординат Oy) и какая – третьей (ось аппликат Oz). Различают правую и левую системы декартовых прямоугольных координат.

Орты осей Ox , Oy и Oz обозначают через \bar{i} , \bar{j} и \bar{k} соответственно. Так как векторы \bar{i} , \bar{j} и \bar{k} некомпланарны, то они образуют базис, который называется *декартовым прямоугольным базисом*.

При разложении вектора \bar{a} по декартовому базису \bar{i} , \bar{j} и \bar{k} существует тройка чисел a_x , a_y и a_z , при которых справедливо равенство

$$\bar{a} = a_x \bar{i} + a_y \bar{j} + a_z \bar{k}.$$

Числа a_x , a_y и a_z называются *декартовыми прямоугольными координатами* вектора \bar{a} .

Запись $\bar{a} = (a_x; a_y; a_z)$ означает, что вектор \bar{a} имеет декартовы прямоугольные координаты a_x , a_y и a_z . Числа a_x , a_y и a_z являются проекциями вектора на координатные оси Ox , Oy и Oz соответственно.

Декартовыми прямоугольными координатами точки M называются проекции ее радиус-вектора \overline{OM} на соответствующие координатные оси. Запись $M(x;y;z)$ означает, что точка M имеет координаты x , y и z .

Координатные плоскости (плоскости, проходящие через пары координатных осей) делят все пространство на восемь частей, называемых *октантами*.

При задании вектора \bar{a} и точки M на декартовой координатной плоскости используют две координаты: a_x и a_y – $\bar{a} = (a_x; a_y)$ и $M(x;y)$ соответственно. Следовательно, задание вектора или точки на координатной плоскости является частным случаем их задания в координатном пространстве.

Рене Декарт (Rene Descartes) родился 31 марта 1596 года в небольшом городке Ла-Газ (Франция). В главном математическом труде «Геометрия» (1637) заложил основы аналитической геометрии. Впервые ввел в употребление: понятия переменной величины и функции; общепринятые знаки для переменных величин (x , y , z , ...), коэффициентов (a , b , c , ...) и обозначения степеней (x^2); метод координат и понятие уравнения кривой. Декарт был выдающимся философом. Разработал метод познания, в котором главенствующая роль в научном исследовании отводится разуму. Умер 11 февраля 1650 года в Стокгольме (Швеция).



В MATLAB вектор задается в координатной форме с помощью встроенных в ядро функций `horzcat()` и `vertcat()`, которые объединяют значения переменных и/или числовых констант в массив. Существует короткая запись задания вектора с помощью последовательности специальных символов (`«[...]»`, `«[..., ...]»` или `«[... ; ...]»`). Следовательно, вектор рассматривается как массив значений. Массив может иметь размер $1 \times n$ или $n \times 1$, где n – число элементов вектора. Таким образом, в MATLAB допускается создание как векторов-строк, так и векторов-столбцов. В дальнейшем под словом «вектор» понимается вектор-строка.

Пример 3.1. Создать с помощью специальных символов вектор-строку $\bar{a} = (2; 4; 6)$ и вектор-столбец $\bar{b} = (1; 8; -2)^T$ (для сокращения места вектор-столбец в ряде случаев записывается в строку с символом T , который обозначает транспонирование).

Решение:

```
>> A = [2 4 6], B = [1; 8; -2]
A =
    2      4      6
B =
    1
    8
   -2
```

Пример 3.2. Создать с помощью стандартных функций вектор-столбец $a = (1; 8; -2)^T$ и вектор-строку $\bar{b} = (2; 4; 6)$.

Решение:

```
>> A=vertcat(1,8,-2), B=horzcat(2,4,6)
A =
    1
    8
   -2
B =
    2      4      6
```

Использование специальных символов позволяет сократить время на запись команды.

Система MATLAB хранит в рабочей области значения координат вектора последовательно, начиная с первого. Получить доступ к элементу вектора можно, указав его имя и индекс. При индексации в MATLAB применяются правила, используемые в языке программирования FORTRAN (FORmula TRANslation). Следовательно, нумерация массивов по умолчанию начинается с единицы. Попытка обратиться к нулевому индексу вызовет ошибку (???) `Subscript indices must either be real positive integers or logicals` –

??? Индексы должны быть либо натуральными числами, либо логическими значениями).

Пример 3.3. Создать с помощью специальных символов вектор $\bar{a} = (-1; 5; 2)$ и изменить значение координаты a_y на 3.

Решение:

>> A=[-1, 5, 2], A(2)=3

A =

-1 5 2

A =

-1 3 2

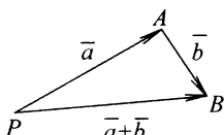
Таким образом, для создания вектора используются квадратные скобки, а для доступа к элементу вектора – круглые.

3.1.2. Линейные операции над векторами и их свойства

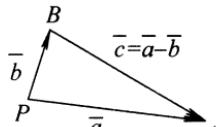
Линейными операциями над векторами являются операции вычисления суммы и разности векторов, а также умножение вектора на число.

Пусть \bar{a} и \bar{b} – два произвольных вектора. Возьмем произвольную точку P на координатной плоскости и построим вектор $\overline{PA} = \bar{a}$. От точки A отложим вектор $\overline{AB} = \bar{b}$. Вектор \overline{PB} , соединяющий начало вектора \bar{a} с концом вектора \bar{b} , называется *суммой векторов*: $\overline{PB} = \bar{a} + \bar{b}$ (см. рис. 3.1,а).

Разностью двух векторов \bar{a} и \bar{b} называется третий вектор $\bar{c} = \bar{a} - \bar{b}$, сумма которого с вычитаемым вектором \bar{b} дает вектор \bar{a} . Следовательно, если $\bar{c} = \bar{a} - \bar{b}$, то $\bar{c} + \bar{b} = \bar{a}$ (см. рис. 3.1,б).



а) сумма векторов



б) разность векторов

Рис. 3.1. Геометрическая интерпретация суммы и разности векторов

Пример 3.4. Вычислить сумму двух векторов $\bar{a} = (3; -1; 5)$ и $\bar{b} = (-4; 5; -1)$.

Решение:

>> A=[3, -1, 5]; B=[-4, 5, 1]; C=A+B

C =

-1 4 6

Суммой двух векторов \bar{a} и \bar{b} является вектор \bar{c} с координатами $(-1; 4; 6)$.

Пример 3.5. Вычислить разность двух векторов $\bar{a} = (3; -1; 5)$

и $\bar{b} = (-4; 5; -1)$.

Решение:

```
>> A=[3,-1,5]; B=[-4,5,-1]; C=A-B
C =
    7      -6      6
>> A=C+B
A =
    3      -1      5
```

Разностью двух векторов \bar{a} и \bar{b} является вектор \bar{c} с координатами $(7;-6;6)$.

При суммировании или вычитании векторов, заданных в координатной форме, суммируются или вычтываются значения соответствующих элементов этих векторов. Следовательно, при выполнении операции суммирования или вычитания векторы должны иметь одинаковый размер. В противном случае система MATLAB выдаст сообщение об ошибке.

Сумма векторов обладает следующими свойствами:

- 1) $\bar{a} + \bar{b} = \bar{b} + \bar{a}$ – переместительное свойство;
- 2) $(\bar{a} + \bar{b}) + \bar{c} = \bar{a} + (\bar{b} + \bar{c})$ – сочетательное свойство.

Пример 3.6. Проверить свойства суммы векторов, используя векторы $\bar{a}=(2;3;4)$, $\bar{b}=(3;5;2)$ и $\bar{c}=(1;1;1)$.

Решение:

```
>> A=[2,3,4]; B=[3,5,2]; C=[1,1,1];
>> isequal(A+B,B+A), isequal((A+B)+C,A+(B+C))
ans =
    1
ans =
    1
```

При проверке свойств использовалась функция `isequal()`, которая возвращает значение 1 (`true`), если сравниваемые величины равны, и 0 (`false`) – в противном случае. Таким образом, переместительное и сочетательное свойства имеют место при суммировании и вычитании векторов.

Произведением вектора \bar{a} на действительное число λ называется вектор \bar{b} , коллинеарный вектору \bar{a} , имеющий длину, равную $|\lambda| \cdot |\bar{a}|$, и *соправленный* вектору \bar{a} , если $\lambda > 0$, и *противоположно направленный* вектору \bar{a} , если $\lambda < 0$.

Следовательно, два вектора \bar{a} и \bar{b} коллинеарны тогда и только тогда, когда имеет место равенство $\bar{b} = \lambda \cdot \bar{a}$.

Из определения умножения вектора на число следует, что $\bar{a} = |\bar{a}| \cdot \bar{a}^0$, т.е. каждый вектор равен произведению его длины на единичный вектор того же направления.

В MATLAB умножение вектора на число выполняется с помощью встроенных в ядро функций `mtimes()` и `times()`. Эти функции могут быть вызваны и с помощью операторов «`*`» или «`.*`» соответственно. Первая функция выполняет матричное умножение входных параметров, а вторая – их поэлементное умножение. При умножении вектора на скаляр обе функции вернут одинаковый результат.

Пример 3.7. Умножить вектор $\bar{a} = (1; 2; 3)$ на число 2.

Решение:

```
>> A = [1 2 3]; B = 2*A  
B =  
    2      4      6
```

Таким образом, при умножении вектора на число каждый элемент вектора умножается на это число.

Умножение вектора на число обладает следующими свойствами:

1. $(\bar{a} + \bar{b}) \cdot \lambda = \bar{a} \cdot \lambda + \bar{b} \cdot \lambda$ – распределительное свойство;
2. $\lambda_1 \cdot (\lambda_2 \cdot \bar{a}) = (\lambda_1 \cdot \lambda_2) \cdot \bar{a}$ – сочетательное свойство.

Пример 3.8. Проверить свойства умножения вектора на число с помощью векторов $\bar{a} = (2; 3; 4)$ и $\bar{b} = (3; 5; 2)$ и чисел $\lambda = 5$, $\lambda_1 = 2$ и $\lambda_2 = 3$.

Решение:

```
>> A=[2,3,4]; B=[3,5,2]; lambda=5; lambda1=2; lambda2=3;  
>> A*lambda+B*lambda, isequal((A+B)*lambda,ans)  
ans =  
    25      40      30  
ans =  
    1  
>> (lambda1*lambda2)*A, isequal(ans,lambda1*(lambda2*A))  
ans =  
    12      18      24  
ans =  
    1
```

Следовательно, распределительное и сочетательное свойства имеют место при умножении вектора на число.

3.1.3. Длина вектора

Длина вектора \bar{a} равна квадратному корню из суммы квадратов его координат:

$$|\bar{a}| = \sqrt{\sum_{k=1}^n a_k^2},$$

где k – номер элемента вектора, n – общее число элементов в векторе. Для

векторов, определенных с помощью декартовой системы координат в пространстве, общее число элементов равно трем.

Пример 3.9. Вычислить длину вектора $\bar{a} = (2; 3; 4)$.

Решение:

```
>> A=[2 3 4]; sqrt(sum(A.*A))
ans =
5.3852
```

Длина вектора $\bar{a} = (2; 3; 4)$ равна 5.3852. При вычислении длины вектора использовались: оператор поэлементного умножения «.*», функция `sum()`, которая осуществляет суммирование элементов массива, функция `sqrt()`, вычисляющая корень из значения входного аргумента.

Пример 3.10. Вычислить единичный вектор \bar{a}^0 , сонаправленный вектору $\bar{a} = (1; 2; 3)$. Проверить, является ли вычисленный вектор \bar{a}^0 единичным.

Решение:

```
>> A=[1 2 3]; a0=A/sqrt(sum(A.*A))
a0 =
0.2673    0.5345    0.8018
>> sqrt(sum(a0.*a0))
ans =
1
```

Координаты орта вектора $\bar{a} = (1; 2; 3)$ равны (0.2673; 0.5345; 0.8018).

3.1.4. Направляющие косинусы вектора

Пусть дан вектор \bar{a} . Обозначим углы наклона этого вектора к осям Ox , Oy и Oz соответственно через α , β и γ . Три числа $\cos\alpha$, $\cos\beta$ и $\cos\gamma$ являются направляющими косинусами вектора \bar{a} . Направляющие косинусы вычисляются по формулам

$$\cos\alpha = \frac{a_1}{\sqrt{a_1^2 + a_2^2 + a_3^2}}, \quad \cos\beta = \frac{a_2}{\sqrt{a_1^2 + a_2^2 + a_3^2}}, \quad \cos\gamma = \frac{a_3}{\sqrt{a_1^2 + a_2^2 + a_3^2}}.$$

Направляющие косинусы удовлетворяют следующему соотношению:

$$\cos^2\alpha + \cos^2\beta + \cos^2\gamma = 1.$$

Пример 3.11. Вычислить углы наклона вектора $\bar{a} = (-1; 2; 5)$ к осям координат.

Решение:

```
>> A=[-1,2,5]; acos(A./sqrt(sum(A.*A)))*180/pi
```

```

ans =
    100.5197      68.5833      24.0948
>> sum(cos(ans./180*pi).^2)
ans =
    1

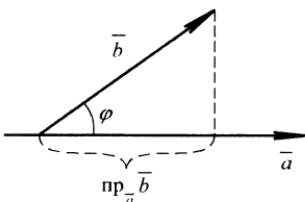
```

Углы наклона вектора $\bar{a} = (-1; 2; 5)$ $\alpha = 100.52^\circ$, $\beta = 68.58^\circ$ и $\gamma = 24.10^\circ$. При вычислении углов наклона использовались оператор поэлементного деления «./» и оператор матричного деления «/». При вычислении суммы квадратов направляющих косинусов использовался оператор поэлементного возведения в степень «.^». В результате сумма квадратов направляющих косинусов равна единице.

3.2. СКАЛЯРНОЕ ПРОИЗВЕДЕНИЕ ВЕКТОРОВ

3.2.1. Определение скалярного произведения векторов

Скалярным произведением двух ненулевых векторов \bar{a} и \bar{b} называется число, равное произведению длин этих векторов на косинус угла между ними:



Р и с . 3.2. Проекция вектора \bar{b} на ось, сонаправленную с вектором \bar{a}

$$\bar{a} \cdot \bar{b} = |\bar{a}| \cdot |\bar{b}| \cdot \cos \varphi,$$

где $\varphi = (\bar{a}, \bar{b})$. Так как $|\bar{b}| \cos \varphi = \text{пр}_{\bar{a}} \bar{b}$ (см. рис. 3.2) и $|\bar{a}| \cos \varphi = \text{пр}_{\bar{b}} \bar{a}$, то скалярное произведение может быть записано следующим образом:

$$\bar{a} \cdot \bar{b} = |\bar{a}| \text{пр}_{\bar{a}} \bar{b} = |\bar{b}| \text{пр}_{\bar{b}} \bar{a},$$

т.е. скалярное произведение двух векторов равно длине одного из них, умноженной на проекцию другого на ось, сонаправленную с первым вектором.

Физический смысл скалярного произведения: работа постоянной силы при прямолинейном перемещении точки ее приложения равна скалярному произведению вектора силы $\bar{F}(\bar{b})$ на вектор перемещения $\bar{s}(\bar{a})$ (рис. 3.2):

$$A = \bar{F} \cdot \bar{s} = |\bar{F}| \cdot |\bar{s}| \cdot \cos \varphi.$$

П р и м е р 3.12. Найти работу силы $\bar{F} = (2; 2)$ на перемещении $\bar{s} = (5; 2)$.

Р е ш е н и е :

```

>> F=[2 2]; S=[5 2];
>> phil=atan2(F(2),F(1))*180/pi, phi2=atan2(S(2),S(1))*180/pi
phil =
    45
phi2 =
   21.8014

```

```

>> phi=phi1-phi2
phi =
23.1986
>> A=sqrt(sum(F.*F))*sqrt(sum(S.*S))*cos(phi*pi/180)
A =
14.0000

```

Геометрический смысл скалярного произведения: если скалярное произведение равно нулю, то два ненулевых вектора ортогональны (перпендикулярны). Если скалярное произведение больше нуля, то угол между векторами острый, если меньше нуля – тупой.

3.2.2. Скалярное произведение векторов, заданных в координатной форме

Скалярное произведение двух векторов \bar{a} и \bar{b} , заданных в координатной форме, равно сумме парных произведений их одноименных координат:

$$\bar{a}\bar{b} = a_1b_1 + a_2b_2 + \dots + a_nb_n = \sum_{k=1}^n a_k b_k.$$

Пример 3.13. Вычислить скалярное произведение двух векторов $\bar{a} = (1; 2; 3)$ и $\bar{b} = (3; 2; 1)$.

Решение:

```

>> A=[1,2,3]; B=[3,2,1]; [sum(A.*B), dot(A,B)]
ans =
10      10

```

Скалярное произведение двух векторов \bar{a} и \bar{b} , заданных в координатной форме, вычисляется с помощью стандартной функции `sum()` и оператора поэлементного умножения «.*» либо с помощью стандартной функции `dot()`.

3.2.3. Свойства скалярного произведения

Скалярное произведение обладает следующими основными свойствами:

- 1) $\bar{a}\bar{b} = \bar{b}\bar{a}$ – переместительное свойство;
- 2) $\bar{a}^2 = \bar{a}\bar{a} = |\bar{a}|^2$, $|\bar{a}|^2$ – скалярный квадрат вектора;
- 3) $(\bar{a} + \bar{b}) \cdot \bar{c} = \bar{a}\bar{c} + \bar{b}\bar{c}$ – распределительное свойство;
- 4) $(\lambda\bar{a})\bar{b} = \lambda(\bar{a}\bar{b})$ – сочетательное свойство относительно числового множителя.

Пример 3.14. Проверить свойства скалярного произведения с помощью векторов $\bar{a} = (1; 2; 3)$, $\bar{b} = (3; 2; 1)$ и $\bar{c} = (3; 4; 5)$.

Решение:

```

>> A=[1 2 3]; B=[3 2 1]; C=[3 4 5];
>> isequal(dot(A,B),dot(B,A)) % переместительное свойство
ans =
    1
>> isequal(dot(A,A),sqrt(sum(A.*A)).^2)
ans =
    1
>> isequal(dot(A+B,C),dot(A,C)+dot(B,C)) % распределительное
ans =
    1
>> isequal(dot(3*A,B),3*dot(A,B)) % сочетательное
ans =
    1

```

Следовательно, все рассмотренные свойства справедливы для скалярного произведения.

3.2.4. Определение угла между двумя векторами

Угол между ненулевыми векторами \bar{a} и \bar{b} определяется по формуле

$$\cos \varphi = \frac{\bar{a} \cdot \bar{b}}{|\bar{a}| |\bar{b}|} = \frac{a_1 b_1 + a_2 b_2 + a_3 b_3}{\sqrt{a_1^2 + a_2^2 + a_3^2} \sqrt{b_1^2 + b_2^2 + b_3^2}}.$$

Пример 3.15. Вычислить угол между векторами $\bar{a}=(1;2;3)$ и $\bar{b}=(3;2;1)$.

Решение:

```

>> A=[1 2 3]; B=[3 2 1];
>> phi=acos(dot(A,B)/(sqrt(sum(A.*A))*sqrt(sum(B.*B))))
phi =
    0.7752
>> phi=phi*180/pi
phi =
    44.4153

```

Следовательно, угол между векторами $\bar{a}=(1;2;3)$ и $\bar{b}=(3;2;1)$ равен 44.42° , или 0.7752 радиан.

3.3. ВЕКТОРНОЕ ПРОИЗВЕДЕНИЕ ДВУХ ВЕКТОРОВ

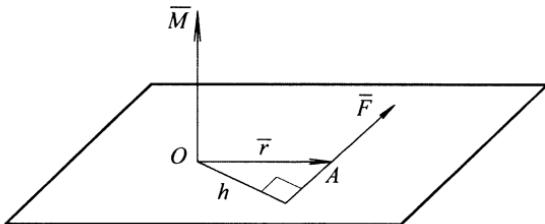
3.3.1. Определение векторного произведения

Векторным произведением двух векторов \bar{a} и \bar{b} называется новый вектор \bar{c} , длина которого равна площади параллелограмма, построенного на векторах \bar{a} и \bar{b} , приведенных к общему началу, и составляющего с вектора-

ми \bar{a} и \bar{b} правую тройку векторов (перпендикулярен перемножаемым векторам и направлен в такую сторону, чтобы кратчайший поворот от \bar{a} к \bar{b} вокруг полученного вектора происходил против часовой стрелки, если смотреть из конца вектора \bar{c}).

Векторное произведение обозначается $\bar{a} \times \bar{b}$, или $[\bar{a}, \bar{b}]$.

Физический смысл векторного произведения: векторное произведение силы \bar{F} на радиус-вектор \bar{r} , соединяющий некоторую точку O с точкой приложения силы \bar{F} , равно моменту силы относительно точки O (рис. 3.3).



Р и с . 3.3. Физический смысл векторного произведения

П р и м е р 3.16. Вычислить векторное произведение векторов $\bar{a} = (1; 2; 3)$ и $\bar{b} = (3; 2; 1)$.

Р е ш е н и е :

```
>> A=[1 2 3]; B=[3 2 1];
>> C=cross(A,B)
C =
    -4      8     -4
```

При вычислении векторного произведения использовалась стандартная функция `cross()`. В результате получился вектор размера 1×3 , где первый элемент вектора соответствует значению его проекции на ось x , второй – на ось y и третий – на ось z .

3.3.2. Свойства векторного произведения

Свойства векторного произведения:

- 1) $\bar{a} \times \bar{b} = -(\bar{b} \times \bar{a})$ – при перестановке сомножителей векторное произведение меняет свой знак;
- 2) $(\lambda \cdot \bar{a}) \times \bar{b} = \bar{a} \times (\lambda \cdot \bar{b}) = \lambda \cdot (\bar{a} \times \bar{b})$ – сочетательное свойство относительно скалярного множителя λ ;
- 3) $(\bar{a} + \bar{b}) \times \bar{c} = \bar{a} \times \bar{c} + \bar{b} \times \bar{c}$ – распределительное свойство;
- 4) если векторное произведение равно нулевому вектору, то либо один из двух векторов нулевой, либо векторы являются коллинеарными.

Пример 3.17. Проверить свойства векторного произведения с помощью векторов $\bar{a}=(1;1;0)$, $\bar{b}=(5;0;0)$ и $\bar{c}=(0;0;1)$, а также скалярного множителя $\lambda=2$.

Решение:

```
>> A=[1 1 0]; B=[5 0 0]; C=[0 0 1]; lambda=2;
>> isequal(cross(A,B), -cross(B,A))
ans =
    1
>> isequal(cross(lambda*A,B), lambda*cross(A,B))
ans =
    1
>> isequal(cross(A+B,C), cross(A,C)+cross(B,C))
ans =
    1
>> cross(A,A)
ans =
    0     0     0
```

При проверке четвертого свойства векторного произведения вектор \bar{a} умножался сам на себя. Так как два равных вектора коллинеарны, то в результате получился нулевой вектор. Таким образом, все рассмотренные свойства справедливы для векторного произведения.

3.4. СМЕШАННОЕ ПРОИЗВЕДЕНИЕ ТРЕХ ВЕКТОРОВ

3.4.1. Определение смешанного произведения

Смешанным (или векторно-скалярным) произведением векторов \bar{a} , \bar{b} и \bar{c} (взятых в указанном порядке) называется скалярное произведение вектора \bar{a} на векторное произведение векторов \bar{b} и \bar{c} : $\bar{a} \cdot (\bar{b} \times \bar{c})$.

Геометрический смысл смешанного произведения: смешанное произведение трех некомпланарных векторов \bar{a} , \bar{b} и \bar{c} , приведенных к общему началу, равно объему параллелепипеда, построенного на векторах \bar{a} , \bar{b} и \bar{c} как на ребрах (взятому со знаком «+» для правой тройки векторов и со знаком «-» – для левой).

Пример 3.18. Найти смешанное произведение векторов $\bar{a}=(5;0;0)$, $\bar{b}=(0;5;0)$ и $\bar{c}=(0;0;1)$, где векторы \bar{b} и \bar{c} перемножаются векторно, а их результат на вектор \bar{a} – скалярно.

Решение:

```
>> A=[5 0 0]; B=[0 5 0]; C=[0 0 1];
>> ABC=sum(A.*cross(B,C))
ABC =
    25
```

В результате смешанное произведение векторов \bar{a} , \bar{b} и \bar{c} равно 25.

3.4.2. Свойства смешанного произведения

Смешанное произведение обладает следующими свойствами:

1. Смешанное произведение не меняется при круговой перестановке его сомножителей, но перестановка двух соседних сомножителей меняет знак произведения:

$$\overline{abc} = \overline{bca} = \overline{cab} = -\overline{bac} = -\overline{acb} = -\overline{cba}.$$

2. Необходимым и достаточным условием компланарности трех ненулевых векторов является равенство нулю их смешанного произведения (\bar{a}, \bar{b} и \bar{c} компланарны $\Leftrightarrow \overline{abc} = 0$).

Пример 3.19. Проверить первое свойство смешанного произведения с помощью векторов $\bar{a}=(5;0;0)$, $\bar{b}=(0;5;0)$ и $\bar{c}=(0;0;1)$, а второе свойство – с помощью векторов $\bar{a}=(1;2;0)$, $\bar{b}=(2;2;0)$ и $\bar{c}=(3;2;0)$.

Решение:

```
>> A=[5 0 0]; B=[0 5 0]; C=[0 0 1];
>> ABC=sum(A.*cross(B,C)); BCA=sum(B.*cross(C,A));
CAB=sum(C.*cross(A,B)); BAC=sum(B.*cross(A,C));
ACB=sum(A.*cross(C,B)); CBA=sum(C.*cross(B,A));
>> [ABC BCA CAB BAC ACB CBA]
ans =
      25      25      25     -25     -25     -25
>> ABC+BCA+CAB+BAC+ACB+CBA
ans =
      0
>> A=[1,2,0]; B=[2,2,0]; C=[3,2,0]; sum(A.*cross(B,C))
ans =
      0
```

Следовательно, все рассмотренные свойства справедливы для смешанного произведения.

3.5. ВНЕШНЕЕ ПРОИЗВЕДЕНИЕ ВЕКТОРОВ

Внешним произведением векторов \bar{a} и \bar{b} называется матрица C размера $n \times m$, элементы которой вычисляются по формуле $c_{ij} = a_i \cdot b_j$.

Пример 3.20. Вычислить векторное произведение векторов $\bar{a}=(1;2;3)^T$ и $\bar{b}=(1;2;3)$.

Решение:

```
>> A = [1; 2; 3]; B = [1 2 3]; A*B
ans =
      1      2      3
      2      4      6
      3      6      9
```

При вычислении внешнего произведения используется оператор матричного умножения «`*`», являющийся сокращенной записью функции `mtimes()`. Следовательно, необходимым условием вычисления внешнего произведения является то, что вектор \bar{a} должен быть вектором-столбцом, а вектор \bar{b} – вектором-строкой. Если вектор \bar{a} является вектором-строкой, а вектор \bar{b} – вектором-столбцом, то в результате выполнения матричного умножения будет вычислено скалярное произведение.

Внешнее произведение имеет широкое применение в системе MATLAB, так как позволяет исключить операторы повтора (`for`, `while`), увеличивающие время вычисления.

3.6. НОРМА ВЕКТОРА

3.6.1. Определение нормы вектора

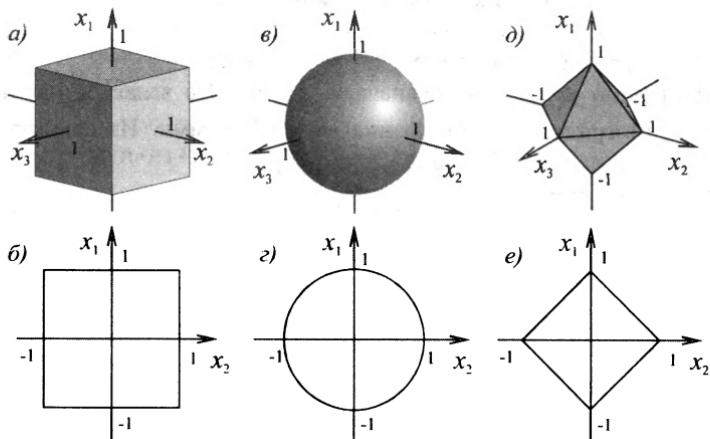
Нормой вектора \bar{a} называется число, обозначаемое символом $\|\bar{a}\|$ и удовлетворяющее следующим свойствам:

1. $\|\bar{a}\| \geq 0$; $\|\bar{a}\| = 0 \Leftrightarrow \bar{a} = 0$ – норма вектора неотрицательна. Норма вектора равна нулю, если вектор \bar{a} нулевой, и, наоборот, если вектор \bar{a} нулевой, то его норма равна нулю.
2. Для любого числа λ и любого вектора \bar{a} справедливо равенство $\|\lambda\bar{a}\| = |\lambda| \cdot \|\bar{a}\|$.
3. Для любых векторов \bar{a} и \bar{b} справедливо неравенство $\|\bar{a} + \bar{b}\| \leq \|\bar{a}\| + \|\bar{b}\|$.

Существует несколько видов нормы вектора:

- a) $\|\bar{a}\|_1 = \sum_{i=1}^n |a_i|$ – сумма абсолютных значений элементов вектора (первая, или кубическая норма);
- b) $\|\bar{a}\|_2 = \sqrt{\sum_{i=1}^n a_i^2}$ – квадратный корень из суммы квадратов элементов вектора (евклидова, или сферическая норма).
- v) $\|\bar{a}\|_\infty = \max_{1 \leq i \leq n} |a_i|$ – максимальное число из абсолютных значений элементов вектора (бесконечная, или октаэдрическая норма).

Множество векторов $x \in E$, удовлетворяющих условию $\|\bar{a}\| \leq 1$, называется *единичным шаром* в пространстве E . На рис. 3.4 представлены единичные шары в пространстве R^3 и R^2 для первой (a и b), второй (c и d) и бесконечной (e и f) норм.



Р и с . 3.4. Единичные шары векторных норм в пространстве R^2 и R^3

Первая норма порождает множество $\{(x_1, x_2, x_3) | |x_1| + |x_2| + |x_3| \leq 1\}$, т.е. куб с ребром, равным двум. Множество $\{(x_1, x_2, x_3) | x_1^2 + x_2^2 + x_3^2 \leq 1\}$, которое образует вторая норма, является сферой с радиусом, равным единице. Единичный шар для бесконечной нормы представляет собой октаэдр в пространстве R^3 .

В MATLAB предусмотрена возможность вычисления всех трех видов нормы вектора с помощью функции `norm()`, первый аргумент которой является именем вектора, а второй – видом нормы.

Пример 3.21. Вычислить первую, евклидову и бесконечную нормы вектора $a = (2; 4; 6; 8)$.

Решение:

```
>> A = [2 4 6 8]; norm(A,1), norm(A,2), norm(A,inf)
ans =
20
ans =
10.9545
ans =
8
```

Если не указывается вид нормы, то вычисляется вторая норма.

Векторные нормы применяются для вычисления расстояния между двумя векторами, а также для определения расстояния между исходной матрицей и множеством вырожденных матриц при оценке числа обусловленности во время решения системы линейных алгебраических уравнений. При оценке числа обусловленности применяют первую норму, так как для ее определения требуется меньше вычислений.

3.6.2. Расстояние между векторами

Величину $\|\bar{x} - \bar{y}\|$ (вторая, или евклидова норма) называют *расстоянием между векторами* \bar{x} и \bar{y} и обозначают $\rho(\bar{x}, \bar{y})$. Это является обобщением обычного геометрического расстояния между точками. Из свойств нормы вектора вытекают следующие свойства расстояния:

- 1) $\rho(\bar{x}, \bar{x}) = 0$; $\rho(\bar{x}, \bar{y}) > 0$ при $\bar{x} \neq \bar{y}$;
- 2) $\rho(\bar{x}, \bar{y}) = \rho(\bar{y}, \bar{x})$;
- 3) $\rho(\bar{x}, \bar{y}) + \rho(\bar{y}, \bar{z}) \geq \rho(\bar{x}, \bar{z})$.

Последнее свойство следует из неравенства треугольника:

$$\rho(\bar{x}, \bar{z}) = \|\bar{x} - \bar{z}\| = \|(\bar{x} - \bar{y}) + (\bar{y} - \bar{z})\| \leq \|\bar{x} - \bar{y}\| + \|\bar{y} - \bar{z}\| = \rho(\bar{x}, \bar{y}) + \rho(\bar{y}, \bar{z}).$$

Пример 3.22. Вычислить наименьшее расстояние между точками $A(1;3;4)$ и $B(1;-1;-2)$.

Решение:

```
>> A=[1, 3, 4]; B=[1, -1, -2]; norm(A-B)
ans =
    7.2111
```

Расстояние между точками равно 7.2111. В приведенном примере для вычисления расстояния использовалась вторая норма, которая определяет кратчайшее расстояние между двумя векторами. Для определения расстояния между векторами можно использовать и первую норму, которая вычисляет «манхэттенское» расстояние, т.е. число кварталов, которое необходимо пройти, чтобы попасть из пункта А в пункт В.

Пример 3.23. Вычислить «манхэттенское» расстояние между точками $A(1;3;4)$ и $B(1;-1;-2)$.

Решение:

```
>> A=[1 3 4]; B=[1 -1 -2]; norm(A-B,1)
ans =
    10
```

Таким образом, «манхэттенское» расстояние равно 10, что больше наименьшего расстояния, равного 7.2111.

Пример 3.24. Проверить свойства расстояния между векторами $\bar{a}=(1;3;4)$, $\bar{b}=(1;-1;-2)$ и $\bar{c}=(-1;5;2)$

Решение:

```
>> A=[1 3 4]; B=[1 -1 -2]; C=[-1, 5, 2];
```

```

>> norm(A-A)
ans =
    0
>> isequal(norm(A-B),norm(B-A))
ans =
    1
>> ge(norm(A-B)+norm(B-C), norm(A-C))
ans =
    1

```

Следовательно, все приведенные свойства справедливы для расстояния между векторами.

3.7. ВЫВОДЫ К ГЛАВЕ 3

В главе 3 рассмотрены способы задания векторов в системе MATLAB, а также примеры решения задач, встречающиеся в векторной алгебре:

- линейные операции над векторами;
- вычисление длины вектора;
- определение направляющих косинусов вектора;
- вычисление скалярного произведения;
- определение угла между векторами;
- вычисление векторного произведения двух векторов;
- вычисление смешанного произведения;
- вычисление внешнего произведения;
- вычисление первой, евклидовой и бесконечной норм вектора;
- определение расстояния между векторами.

Наряду с выполнением линейных и нелинейных операций рассмотрены их соответствующие свойства.

При решении задач векторной алгебры использовались стандартные функции системы MATLAB, перечень которых приведен в табл.3.1.

Таблица 3.1

Перечень функций для операций над векторами

Оператор	Имя функции	Описание
[], [,]	horzcat	Создание вектора-строки
[;]	vertcat	Создание вектора-столбца
+	plus	Суммирование элементов двух векторов

Окончание табл. 3.1

Оператор	Имя функции	Описание
-	minus	Вычитание из элементов первого вектора соответствующих элементов второго вектора
.*	times	1. Умножение элементов вектора на число. 2. Поэлементное умножение двух векторов
*	mtimes	1. Умножение элементов вектора на число. 2. Скалярное произведение (вектор-строка на вектор-столбец). 3. Внешнее произведение (вектор-столбец на вектор-строку).
/	mrdivide	Деление элементов вектора на число
./ (. \)	rdivide (ldivide)	Поэлементное правое (левое) деление векторов данных
.^	power	Поэлементное возведение в степень элементов вектора
,	ctranspose	Транспонирование вектора с комплексным со-пряжением
.'	transpose	Транспонирование вектора
	acos	Вычисление арккосинуса
	cross	Вычисление векторного произведения
	dot	Вычисление скалярного произведения
	isequal	Проверка равенства двух векторов
	norm	Вычисление нормы вектора

Глава 4. РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Быстрое и наглядное решение задач из курса линейной алгебры было основной целью Клива Моулера при создании MATLAB. Возможности, предоставляемые системой, позволяют говорить о MATLAB как об одной из наиболее предпочтительных систем при выборе инструмента для решения задач линейной алгебры.

4.1. МАТРИЦЫ

4.1.1. Основные понятия

Матрицей называется прямоугольная таблица, состоящая из n строк и m столбцов. Для определения матрицы используется следующее обозначение:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} = [a_{ij}]$$

Для любого элемента a_{ij} первый индекс i обозначает номер строки, а второй индекс j – номер столбца.

Если число строк не равно числу столбцов ($n \neq m$), то матрица называется *прямоугольной*, в противном случае – *квадратной*. Число строк или столбцов квадратной матрицы называется ее *порядком*. Диагональ квадратной матрицы, содержащая элементы $a_{11}, a_{22}, \dots, a_{nn}$, называется *главной*, а диагональ, которая содержит элементы $a_{1n}, a_{2n-1}, \dots, a_{n1}$ – *побочной* (или *вспомогательной*).

В прямоугольной матрице вида $n \times m$ возможен случай, когда $n = 1$. В этом случае матрица представляет собой *вектор-строку*. Если же $m = 1$, матрица является *вектором-столбцом*.

В MATLAB матрицы задаются с помощью специальных символов («[]», «[,]» и «[;]»), а также с помощью стандартных функций (`horzcat()` и `vertcat()`). Квадратные скобки используются для объединения значений, которые они охватывают. Матрицы в MATLAB являются двухмерными массивами.

Пример 4.1. Создать матрицу A с помощью специальных символов, а матрицу B – с помощью стандартных функций.

$$A = \begin{pmatrix} 3.4 & 3.4 & 3.4 \\ 2 & 2 & 2 \\ 1.8 & 1.8 & 1.8 \end{pmatrix}; \quad B = \begin{pmatrix} 3.4 & 2 & 1.8 \\ 3.4 & 2 & 1.8 \\ 3.4 & 2 & 1.8 \end{pmatrix}.$$

Р е ш е н и е :

```
>> A = [3.4 3.4 3.4; 2 2 2; 1.8 1.8 1.8]
A =
    3.4000    3.4000    3.4000
    2.0000    2.0000    2.0000
    1.8000    1.8000    1.8000
>> B=vertcat(horzcat(3.4,2,1.8),horzcat(3.4,2,1.8),...
horzcat(3.4,2,1.8))
B =
    3.4000    2.0000    1.8000
    3.4000    2.0000    1.8000
    3.4000    2.0000    1.8000
```

Символы пробел и запятая, используемые внутри квадратных скобок, имеют более высокий приоритет, чем символ точка с запятой. Следовательно, элементы сначала объединяются горизонтально, а затем вертикально. Тройточие используется для переноса команды на следующую строку.

При задании матриц необходимо следить за равенством длин строк, ее образующих. Если строки имеют различную длину, то в командное окно будет выведено сообщение об ошибке, и матрица не будет создана.

Матрицы в MATLAB можно задавать не только с помощью вертикального объединения строк, но и при помощи горизонтального объединения столбцов.

П р и м е р 4.2. Создать матрицы A и B из примера 4.1 с помощью специальных символов и стандартных функций соответственно, используя горизонтальное объединение столбцов.

Р е ш е н и е :

```
>> A = [[3.4;2;1.8] [3.4;2;1.8] [3.4;2;1.8]]
A =
    3.4000    3.4000    3.4000
    2.0000    2.0000    2.0000
    1.8000    1.8000    1.8000
>> B = horzcat(vertcat(3.4,3.4,3.4), ...
vertcat(2,2,2), vertcat(1.8,1.8,1.8))
B =
    3.4000    2.0000    1.8000
    3.4000    2.0000    1.8000
    3.4000    2.0000    1.8000
```

В результате выполнения команд были созданы две квадратные матрицы A и B третьего порядка.

При обращении к элементу матрицы в MATLAB после имени матрицы следует указать в круглых скобках два индекса, разделенных запятой. Первый индекс обозначает номер строки, а второй – номер столбца. Нумерация

строк и столбцов начинается с единицы. Обращаться к элементу массива в MATLAB можно также с помощью одного индекса. В MATLAB элементы массива располагаются в памяти по столбцам, как это принято в FORTRAN. При обращении к нулевому индексу в командное окно будет выведено сообщение об ошибке.

Пример 4.3. Создать матрицу A из примера 4.1 и изменить значения элементов, располагающихся на пересечении первой строки и второго столбца, а также второй строки и третьего столбца, на 5 и на 10 соответственно. К первому элементу обратиться с помощью двух индексов, а ко второму – с помощью одного индекса.

Решение:

```
>> A = [3.4 3.4 3.4; 2 2 2; 1.8 1.8 1.8]; A(1,2) = 5; A(8) = 10
A =
    3.4000    5.0000    3.4000
    2.0000    2.0000   10.0000
    1.8000    1.8000    1.8000
```

При обращении к строке (столбцу) матрицы необходимо указать номер строки в качестве первого (второго) индекса, а на месте второго (первого) индекса ввести символ двоеточие «`:`».

Пример 4.4. Создать матрицу B из примера 4.1 и присвоить всем элементам второго столбца значение 3, всем элементам первой строки, значение 0.5. Удалить третий столбец.

Решение:

```
>> B=[3.4,2,1.8; 3.4,2,1.8; 3.4,2,1.8]; B(:,2)=3; B(1,:)=0.5; B(:,3)=[]
B =
    0.5000    0.5000
    3.4000    3.0000
    3.4000    3.0000
```

Символ пустого массива используется для удаления выделенного фрагмента массива. Одна из размерностей удаляемого фрагмента должна равняться числу строк или столбцов массива, из которого удаляется фрагмент. В противном случае в командное окно будет выведено сообщение об ошибке.

Две матрицы называются *равными*, если они имеют одинаковое число строк, одинаковое число столбцов и их соответствующие элементы равны между собой.

Пример 4.5. Проверить на равенство следующие матрицы:

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}; \quad C = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}.$$

Решение:

```
>> A = [1 1; 2 2]; B = [1 1; 2 2]; C = [1 2; 2 1];
>> isequal(A,B), isequal(A,C)
ans =
    1
ans =
    0
```

Матрицы A и B равны между собой, так как они имеют одинаковый размер и их соответствующие элементы равны. Результатом сравнения матриц A и B является логическая единица. Во втором случае матрицы A и C не равны, так как не все соответствующие элементы этих матриц равны между собой. В результате сравнения матриц A и C получается логический ноль.

Если в матрице A размера $n \times m$ поменять местами строки со столбцами, то получится матрица размера $m \times n$, которая называется *транспонированной* по отношению к матрице A :

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \cdots & \cdots & \cdots & \cdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{pmatrix}.$$

В MATLAB операция транспонирования матрицы выполняется с помощью либо оператора «`'. '`», либо функции `transpose()`. В случае применения оператора «`'`» или функции `ctranspose()` выполняется транспонирование с комплексным сопряжением. Если значениями элементов матрицы являются действительные числа, то результаты выполнения операций транспонирования и транспонирования с комплексным сопряжением будут одинаковыми.

Пример 4.6. Транспонировать матрицу $A = \begin{pmatrix} 1 & 3 \\ 1 & 3 \end{pmatrix}$.

Решение:

```
>> A = [1 3; 1 3]; A', transpose(A)
ans =
    1      1
    3      3
ans =
    1      1
    3      3
```

В результате выполнения команд транспонирования исходной матрицы, состоящей из действительных чисел, с использованием функций транспонирования и транспонирования с комплексным сопряжением получился одинаковый ответ.

4.1.2. Специальные матрицы

В численных методах расчета используются специальные матрицы, форма которых позволяет проводить вычисления с меньшей потерей точности. Как правило, для задания матриц специального вида используются стандартные функции MATLAB.

4.1.2.1. Диагональные матрицы

В диагональных матрицах $a_{ij} = 0$, при всех i и j , кроме $i = j$. Если у диагональной матрицы все числа главной диагонали равны между собой, т.е. $a_{11} = a_{22} = \dots = a_{nn}$, то такая матрица называется *скалярной*. Если в скалярной матрице все числа главной диагонали равны единице, то матрица называется *единичной*.

Пример 4.7. Сформировать диагональную матрицу $D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$.

Решение:

```
>> V=1:3; D=diag(V)
```

$D =$

1	0	0
0	2	0
0	0	3

Для задания диагональной матрицы используется стандартная функция `diag()`, которая принимает в качестве входного аргумента вектор и размещает его на главной диагонали.

Пример 4.8. Создать единичную матрицу $A(3 \times 3)$, где $a_{11} = a_{22} = a_{33} = 1$.

Решение:

```
>> A=eye(3)
```

$A =$

1	0	0
0	1	0
0	0	1

Функция `eye()` формирует единичную матрицу, размер которой определяется входным аргументом. Единичная матрица используется для вычисления обратной матрицы методом Гаусса–Жордана.

4.1.2.2. Треугольные и трапециевидные матрицы

Квадратная матрица A является треугольной, если ее элементы, расположенные выше или ниже главной диагонали, равны нулю. Если нулевые элементы располагаются выше (ниже) главной диагонали, то такая матрица называется *нижней (верхней) треугольной матрицей*.

Треугольная матрица называется *унитреугольной*, если все элементы ее главной диагонали равны единице. Примером унитреугольной матрицы является матрица множителей L в LU-разложении с использованием метода исключения Гаусса.

Пример 4.9. Выделить из матрицы $A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$ нижнюю и верхнюю

треугольные матрицы.

Решение:

```
>> A=[1 4 7; 2 5 8; 3 6 9]; L=tril(A), U=triu(A)
```

$L =$

1	0	0
2	5	0
3	6	9

$U =$

1	4	7
0	5	8
0	0	9

Выделение треугольных матриц осуществляется с помощью стандартных функций `triu()` и `tril()`. Функция `tril()` выделяет нижнюю треугольную матрицу, а функция `triu()` – верхнюю треугольную.

Если матрица A является прямоугольной и элементы, расположенные выше или ниже главной диагонали, равны нулю, то матрица называется *трапециевидной*. Примером трапециевидной матрицы является расширенная матрица после завершения прямого хода в методе Гаусса.

4.1.2.3. Симметричные матрицы

Матрица является симметричной, если $a_{ij} = a_{ji}$ для всех i и j , или, что эквивалентно, $A = A^T$. К симметричным матрицам приводят многие физические задачи равновесия. Преимущества работы с симметричными матрицами по сравнению с матрицами общего вида состоят в следующем:

- для их хранения требуется примерно в 2 раза меньше памяти вычислительной машины;
- для большинства случаев требуется в 2 раза меньше затрат при вычислениях.

Примером симметричной матрицы является матрица Паскаля, которая образуется из элементов треугольника Паскаля (коэффициентов разложения бинома $(1+n)^j$).

Пример 4.10. Создать матрицу Паскаля $P(3 \times 3)$ и проверить ее симметричность.

Решение:

```
>> P=pascal(3), isequal(P,P')
P =
    1     1     1
    1     2     3
    1     3     6
ans =
    1
```

Следовательно, матрица Паскаля является симметричной матрицей.



Блез Паскаль (Blaise Pascal) родился 19 июня 1623 года в Клермон-Ферран (Франция). Был великим французским философом, математиком и физиком. Его математические работы относятся к периоду 40-х годов XVII века. В 1641–1642 годах сконструировал первую механическую суммирующую машину. К 1654 году закончил ряд работ по арифметике, теории чисел и алгебре, опубликованных в 1665 году, где впервые определил и применил метод математической индукции, разработал способ вычисления биномиальных коэффициентов («Трактат об арифметическом треугольнике»), нашел общий алгоритм для нахождения признаков делимости любого целого на любое другое целое число («О характере делимости чисел») и т.д. В его честь назван один из языков программирования, широко используемый для обучения программированию. Умер 19 августа 1662 года в Париже.

4.1.2.4. Магический квадрат

В *магическом квадрате* сумма значений элементов по строкам, столбцам, а также сумма главной и побочной диагоналям равны. Первым упоминанием о магическом квадрате в Европе считается его изображение на гравюре Альбрехта Дюрера «Меланхолия», датированной 1514 годом.

Сумма элементов по строкам (столбцам) называется *инвариантом магического квадрата* и обозначается μ_n . Значение инварианта зависит от порядка n матрицы и равно $\mu_n = n(n^2 + 1)/2$.



Пример 4.11. Создать магический квадрат размером 4×4 . Вычислить сумму значений элементов по строкам, столбцам, главной и побочной диагоналям.

Решение:

```
>> M = magic(4), col = sum(M,1),... row = sum(M,2) '
M =
    16      2      3     13
     5     11     10      8
     9      7      6     12
     4     14     15      1
col =
    34      34      34      34
row =
    34      34      34      34
>> main_diag = trace(M), sec_diag = trace(rot90(M))
main_diag =
    34
sec_diag =
    34
```

Магический квадрат задается с помощью функции `magic()`, входной аргумент которой определяет размерность матрицы. Для вычисления суммы элементов матрицы по строкам или столбцам следует использовать функцию `sum()`, второй аргумент которой определяет направление суммирования элементов (1 – столбцы, 2 – строки). При вычислении суммы элементов главной диагонали использовалась функция `trace()`, которая вычисляет след матрицы. При вычислении суммы элементов побочной диагонали матрицу M повернули на 90° и вычислили след полученной матрицы. После поворота матрицы на 90° элементы побочной диагонали располагаются на главной диагонали.

Система MATLAB позволяет создавать и другие виды специальных матриц, применяемых в специализированных задачах. Для ознакомления с остальными видами специальных матриц следует обратиться к [63].

4.1.3. Действия над матрицами

Под действиями над матрицами понимается сложение матриц, умножение матрицы на число, элементарные преобразования над матрицами, а также умножение матриц.

4.1.3.1. Линейные операции над матрицами и их свойства

Под линейными операциями понимается суммирование матриц и умножение матрицы на число.

Суммой матриц A и B называется матрица C, элементы которой равны сумме соответствующих элементов матриц A и B. Суммировать можно только матрицы, имеющие одинаковые размеры.

Пример 4.12. Выполнить суммирование матриц $A = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}$ и $B = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$.

Решение:

```
>> A=[2 3; 3 2]; B=[1 1; 2 2]; A+B
ans =
    3     4
    5     4
```

Следовательно, значения элементов итоговой матрицы после выполнения операции суммирования равно сумме значений соответствующих элементов исходных матриц. Суммирование матриц можно осуществлять с помощью прямого обращения к функции plus().

Сумма матриц обладает теми же свойствами, что и сумма векторов:

- 1) $A + B = B + A$ – переместительное свойство;
- 2) $(A + B) + C = A + (B + C)$ – сочетательное свойство.

Пример 4.13. Проверить свойства суммирования матриц A,

$B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $C = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$. Матрицу A взять из примера 4.12.

Решение:

```
>> A=[2 3; 3 2]; B = [1 1; 1 1]; C = [2 2; 2 2];
>> [isequal(A+B,B+A), isequal((A+B)+C,A+(B+C))]
ans =
    1     1
```

Таким образом, все рассмотренные свойства справедливы для суммы матриц.

Произведением матрицы на число называется такая матрица, каждый элемент которой равен произведению соответствующего элемента исходной матрицы на это число.

Пример 4.14. Вычислить произведение матрицы $A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ на число 2.

Решение:

```
>> A=[1 3; 2 4]; A*2
ans =
    2     6
    4     8
```

При выполнении операции умножения матрицы на число значение каждого элемента умножается на это число. Умножение матрицы на число можно выполнять с помощью прямого обращения к функциям `times()` или `mtimes()`.

Умножение матрицы на число обладает теми же свойствами, что и умножение вектора на число:

- 1) $(A + B) \cdot \lambda = A \cdot \lambda + B \cdot \lambda$ – распределительное свойство;
- 2) $\lambda_1 \cdot (\lambda_2 \cdot A) = (\lambda_1 \cdot \lambda_2) \cdot A$ – сочетательное свойство.

Пример 4.15. Проверить свойства умножения матрицы на число.

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}, \quad \lambda_1 = 2, \quad \lambda_2 = 3.$$

Решение:

```
>> A=[1 2; 1 2]; B=[1 1; 2 2]; lambda1=2; lambda2=3;
>> isequal((A+B)*lambda1, A*lambda1+B*lambda1)
ans =
    1
>> isequal(lambda1*(lambda2*A), (lambda1*lambda2)*A)
ans =
    1
```

Следовательно, приведенные свойства справедливы для умножения матрицы на число.

4.1.3.2. Элементарные матричные преобразования

Элементарными матричными преобразованиями являются:

- перестановка местами двух параллельных рядов матрицы;
- умножение всех элементов ряда матрицы на число, отличное от нуля;
- прибавление ко всем элементам ряда матрицы соответствующих элементов параллельного ряда, умноженных на одно и то же число.

Пример 4.16. Поменять местами 1-ю и 3-ю строки в матрице

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{pmatrix}.$$

Решение:

```
>> A=[1 1 1 1; 2 2 2 2; 3 3 3 3]; A=[A(3,:);A(2,:);A(1,:)]
A =
```

$$\begin{matrix} 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{matrix}$$

Пример 4.17. Умножить элементы второй строки матрицы A из предыдущего примера на число -2 .

Решение:

```
>> A=[1 1 1 1; 2 2 2 2; 3 3 3 3]; A=[A(1,:); A(2,:)*-2; A(3,:)]  
A =
```

$$\begin{matrix} 1 & 1 & 1 & 1 \\ -4 & -4 & -4 & -4 \\ 3 & 3 & 3 & 3 \end{matrix}$$

Пример 4.18. Умножить элементы третьей строки матрицы A из примера 4.16 на число -2 и прибавить их к соответствующим элементам первой строки.

Решение:

```
>> A=[1 1 1 1; 2 2 2 2; 3 3 3 3]; A=[A(1,:); A(2,:); A(3,:)*-2];...  
A=[A(1,:)+A(3,:); A(2,:); A(3,:)]
```

$$\begin{matrix} 1 & 1 & 1 & 1 \\ -5 & -5 & -5 & -5 \\ 2 & 2 & 2 & 2 \\ -6 & -6 & -6 & -6 \end{matrix}$$

Элементарные матричные преобразования используются во время приведения к треугольному виду исходной матрицы в методе Гаусса при решении системы линейных алгебраических уравнений.

4.1.3.3. Произведение матриц

Произведением матрицы $A = [a_{ij}]$ размера $n \times m$ на матрицу $B = [b_{jk}]$ размера $m \times r$ есть матрица $C = [c_{ik}]$ размера $n \times r$:

$$C = A \times B = [a_{ij}] \times [b_{jk}] = [c_{ik}], \text{ где } c_{ik} = \sum_{j=1}^m a_{ij} b_{jk}.$$

Таким образом, элемент c_{ik} матрицы C есть сумма произведений элементов i -й строки матрицы A на соответствующие элементы k -го столбца матрицы B . В каждом произведении матриц A и B число столбцов матрицы A должно равняться числу строк матрицы B (форма матриц должна быть согласованной). Из существования произведения $A \times B$ не следует существование произведения $B \times A$.

Рассмотрим умножение квадратных матриц второго порядка A и B :

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Произведением этих матриц будет матрица C следующего вида:

$$C = A \times B = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}.$$

Пример 4.19. Вычислить произведение матриц $A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ и $B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

Решение:

```
>> A=[1 3; 2 4]; B=[1 2; 3 4]; A*B, B*A
ans =
    10    14
    14    20
ans =
    5    11
   11    25
```

Первый результат соответствует умножению матрицы A на B , а второй – умножению матрицы B на A . Результаты получились разными. В том случае, если при $AB = BA$, то матрицы называются *перестановочными*. Умножение двух матриц реализуется с помощью оператора матричного умножения «*» либо прямым обращением к функции `mtimes()`.

Операция умножения матриц обладает следующими свойствами:

- 1) $A(BC) = (AB)C$ – сочетательное свойство;
- 2) $(A+B)C = AC + BC$ – распределительное свойство;
- 3) $\alpha(AB) = (\alpha A)B$ – сочетательное свойство относительно скаляра.

Пример 4.20. Проверить свойства операции умножения матриц с помощью матриц $A = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 2 \\ 4 & 4 \end{pmatrix}$ и $C = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$.

Решение:

```
>> A=[1,1;2,2]; B=[2,2;4,4]; C=[2,2;1,1]; alpha = 2;
>> [isequal(A*(B*C), (A*B)*C), isequal((A+B)*C, A*C+B*C), ...
isequal(alpha*(A*B), (alpha*A)*B)]
ans =
```

```
1      1      1
```

Следовательно, рассмотренные свойства справедливы для операции умножения матриц.

Система MATLAB позволяет выполнять поэлементное умножение матриц. При выполнении поэлементного умножения размеры матриц должны быть одинаковыми.

Пример 4.21. Выполнить поэлементное умножение матриц A и B .
Матрицы взять из примера 4.19.

Решение:

```
>> A=[1 3; 2 4]; B=[1 2; 3 4];
>> A.*B, times(A,B)
ans =
    1      6
    6     16
ans =
    1      6
    6     16
```

При поэлементном умножении матриц умножаются значения соответствующих элементов этих матриц и записываются в результирующую матрицу. В том случае, если размеры матриц при поэлементном умножении не совпадают, то в командное окно будет выведено следующее сообщение:

```
??? Error using ==> .* (Ошибка использования ==> .*)
Matrix dimensions must agree. (Размерности матриц должны
совпадать.)
```

4.1.4. Определитель матрицы и его свойства

Определителем (детерминантом)

$$D = \Delta = \det A = \det[a_{ij}] = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

квадратной матрицы называется число, равное сумме произведений элементов произвольной строки (или столбца) на их алгебраические дополнения:

$$D = \sum_{i=1}^n a_{ij} A_{ij} = \sum_{k=1}^n a_{jk} A_{jk} .$$

Алгебраическим дополнением элемента a_{ij} определителя Δ называется минор M_{ij} этого элемента, взятый со знаком $(-1)^{i+j}$.

Минором M_{ij} элемента a_{ij} определителя Δ называется новый определитель, который получается из данного вычеркиванием строки и столбца, содержащих элемент a_{ij} .

Пример 4.22. Вычислить определитель матрицы $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 2 \end{pmatrix}$, минор M_{12} и алгебраическое дополнение A_{12} .

Решение:

```
>> A=[1 2 1; 2 1 1; 2 1 2]; d=det(A),...
M12=det([A(2:3,1),A(2:3,3)]), A12=(-1)^(1+2)*M12
d =
-3
M12 =
2
A12 =
-2
```

При вычислении определителя используется встроенная в ядро функция `det()`. С помощью оператора «`:`» формировался диапазон номеров строк, которые участвуют в образовании новой матрицы. В том случае, если входным параметром функции `det()` является прямоугольная матрица, то в командное окно будет выведено следующее сообщение об ошибке:

```
??? Error using ==> det (Ошибка использования ==> det)
Matrix must be square. (Матрица должна быть квадратной.)
```

Определитель обладает рядом свойств:

1. Определитель не изменится, если его строки поменять местами с соответствующими столбцами.
2. При перестановке двух строк (или столбцов) определитель изменит свой знак на противоположный.
3. Общий множитель всех элементов строки (или столбца) можно вынести за знак определителя.
4. Определитель с двумя одинаковыми строками или столбцами равен нулю.
5. Если все элементы двух строк или столбцов определителя пропорциональны, то определитель равен нулю.
6. Если к какой-либо строке (столбцу) определителя прибавить соответствующие элементы другой строки (столбца), умноженные на одно и то же число, то определитель не изменит своей величины.
7. Треугольный определитель, у которого все элементы, лежащие выше (ниже) главной диагонали, являются нулями, равен произведению элементов главной диагонали.

Пример 4.23. Проверить свойства определителя.

Решение:

```
>> A = [2 4; 6 4]; isequal(det(A),det(A'))  
ans =  
    1  
>> A = [2 4; 6 4]; isequal(det(A),-det([A(:,2),A(:,1)]))  
ans =  
    1  
>> A = [2 4; 6 4]; isequal(4*det([2 1; 6 1]), det(A))  
ans =  
    1  
>> A=[1,2,3;1,2,3;2,3,4]; isequal(det(A),0)  
ans =  
    1  
>> A=[1,2,3;1,2,5;1,2,4]; isequal(det(A),0)  
ans =  
    1  
>> A=[1,2;1,3]; isequal(det(A),det([A(:,1) A(:,2)+2*A(:,1)]))  
ans =  
    1  
>> A=[1,2,3;0,2,3;0,0,3]; isequal(det(A),6)  
ans =  
    1
```

Следовательно, все приведенные свойства определителя справедливы.

4.1.5. Невырожденные матрицы

4.1.5.1. Основные понятия

Квадратная матрица A называется *невырожденной матрицей*, если определитель $\Delta = \det A \neq 0$. В противном случае матрица A называется *вырожденной*.

Союзной к матрице A называется матрица

$$A^* = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix},$$

где A_{ij} – алгебраическое дополнение элемента a_{ij} исходной матрицы A (определяется таким же образом, как и алгебраическое дополнение элемента определителя).

Пример 4.24. Определить, является ли матрица $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ невырожденной, и вычислить союзную к ней матрицу.

Решение:

```
>> d=det(A)
d =
0
>> A11=(-1)^2*det(A([2,3],[2,3])); A12=(-1)^3*det(A([2,3],[1,3]));...
A13=(-1)^4*det(A([2,3],[1,2]));
>> A21=(-1)^3*det(A([1,3],[2,3])); A22=(-1)^4*det(A([1,3],[1,3]));...
A23=(-1)^5*det(A([1,3],[1,2]));
>> A31=(-1)^4*det(A([1,2],[2,3])); A32=(-1)^5*det(A([1,2],[1,3]));...
A33=(-1)^6*det(A([1,2],[1,2]));
>> A_=[A11 A12 A13; A21 A22 A23; A31 A32 A33]
A_ =
-3      6      -3
 6     -12      6
 -3      6      -3
```

Так как определитель матрицы A равен нулю, то матрица является вырожденной. Союзной к матрице A является матрица

$$A^* = \begin{pmatrix} -3 & 6 & -3 \\ 6 & -12 & 6 \\ -3 & 6 & -3 \end{pmatrix}.$$

4.1.5.2. Обратная матрица и ее свойства

Если A является квадратной матрицей, то *обратной* по отношению к A называется матрица, которая при умножении на A (как слева, так и справа) дает единичную матрицу:

$$A^{-1} \cdot A = A \cdot A^{-1} = E.$$

Если обратная матрица A^{-1} существует, то матрица A называется *обратимой*. Для того чтобы квадратная матрица имела обратную, необходимо и достаточно, чтобы матрица A была невырожденной.

Обратная матрица находится по формуле

$$A^{-1} = \frac{\begin{vmatrix} A_{11}/D & A_{21}/D & \cdots & A_{n1}/D \\ A_{12}/D & A_{22}/D & \cdots & A_{n2}/D \\ \cdots & \cdots & \cdots & \cdots \\ A_{1n}/D & A_{2n}/D & \cdots & A_{nn}/D \end{vmatrix}}{D} = \frac{(A^*)^T}{D},$$

где D – определитель матрицы A , A_{ij} – алгебраические дополнения соответствующих элементов матрицы A .

Пример 4.25. Вычислить обратную матрицу матрицы $A = \begin{pmatrix} 4 & 2 \\ 1 & 2 \end{pmatrix}$.

Решение:

```
>> A=[4 2; 1 2]; inv(A)
ans =
    1/3      -1/3
   -1/6      2/3
```

Результат вычисления обратной матрицы представлен в дробном формате. Для вычисления обратной матрицы используется встроенная в ядро системы функция `inv()`.

Так как вычисление обратной матрицы в явном виде (вычисление определителя и соответствующих алгебраических дополнений) довольно «дорогая» операция с вычислительной точки зрения, то в функции `inv()` применяют метод Гаусса–Жордана. В методе Гаусса–Жордана расширенная матрица состоит из исходной и единичной матриц. Используя метод последовательных исключений (метод Гаусса), приводят исходную матрицу к единичной, а на месте единичной матрицы будет располагаться обратная матрица.

4.1.5.3. Ранг матрицы и его свойства

Рангом матрицы A называется такое число r , что, по крайней мере, один определитель порядка r , получаемый из этой матрицы при удалении некоторых строк и (или) столбцов, отличен от нуля, а все определители порядка $(r+1)$ равны нулю. Ранг матрицы равен наибольшему числу линейно независимых строк (или столбцов).

Пример 4.26. Вычислить ранг матрицы A из примера 4.24.

Решение:

```
>> A = [1 2 3; 4 5 6; 7 8 9]; rank(A)
ans =
    2
```

Ранг матрицы обладает следующими свойствами:

1. Ранг матрицы не меняется при транспонировании.
2. Ранг матрицы не меняется при перестановке ее строк (или столбцов).
3. Ранг матрицы не меняется при умножении строки (или столбца) на отличное от нуля число.
4. Ранг матрицы не меняется при сложении строк (или столбцов).

Пример 4.27. Проверить свойства ранга матрицы, используя матрицу A из примера 4.24.

Решение:

```
>> A=[1 2 3; 4 5 6; 7 8 9]; isequal(rank(A),rank(A'))
ans =
    1
>> isequal(rank(A),rank([A(3,:);A(2,:);A(1,:)]))
ans =
    1
>> isequal(rank(A),rank([A(:,1),2*A(:,2),A(:,3)]))
ans =
    1
>> isequal(rank(A),rank([A(:,1),A(:,1)+A(:,2),A(:,3)]))
ans =
    1
```

Таким образом, все рассмотренные свойства для ранга матрицы являются справедливыми.

Две матрицы называются **эквивалентными**, если ранг одной матрицы равен рангу другой ($A \sim B$).

Пример 4.28. Проверить равенство матриц

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \text{ и } B = \begin{pmatrix} 9 & 8 & 7 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix},$$

а также являются ли они эквивалентными.

Решение:

```
>> A=[1 2 3; 4 5 6; 7 8 9]; B=[9 8 7; 4 5 6; 1 2 3];
>> isequal(A,B), isequal(rank(A),rank(B))
ans =
    0
ans =
    1
```

Следовательно, матрицы A и B не равны между собой, но являются эквивалентными.

Ранг матрицы используется при решении системы линейных уравнений.

4.1.6. Норма матрицы

Нормой матрицы называется число, обозначаемое символом $\|A\|$ и удовлетворяющее следующим свойствам:

1. $\|A\| \geq 0; \|A\| = 0 \Leftrightarrow A = 0$ – норма матрицы неотрицательна. Норма матрицы равна нулю, если матрица A нулевая, и, наоборот, если матрица нулевая, то и ее норма равна нулю.
2. Для произвольной матрицы A и любого числа λ справедливо равенство $\|\lambda A\| = |\lambda| \cdot \|A\|$.
3. Для любых матриц справедливо неравенство $\|A + B\| \leq \|A\| + \|B\|$.
4. Для любых матриц справедливо неравенство $\|AB\| \leq \|A\| \cdot \|B\|$.

Существует несколько видов норм матриц:

- a) $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}|$ – максимальное число, получившееся после суммирования абсолютных значений элементов матрицы по столбцам (первая норма);
- б) $\|A\|_2 = \sqrt{\lambda_{\max}(A \cdot A^T)}$ – квадратный корень из максимального собственного числа симметричной матрицы (вторая норма);
- в) $\|A\|_e = \sqrt{\sum_i^n \sum_j^n a_{ij}^2}$ – сумма квадратов элементов матрицы (Евклидова или Фробениуса норма);
- г) $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$ – максимальное число, получившееся после суммирования абсолютных значений элементов матрицы по строкам (бесконечная норма).

Пример 4.29. Вычислить первую, вторую, евклидову и бесконечную

норму матрицы $A = \begin{pmatrix} -1 & 2 & -3 \\ 3 & 4 & 1 \\ 0 & 0 & 1 \end{pmatrix}$.

Решение:

```
>> A=[-1,2,-3
      3 4 1
      0 0 1];
```

```
>> n1=norm(A,1), n2=norm(A,2),n_fro=norm(A,'fro'),n_inf=norm(A,inf)
n1 =
6
n2 =
5.1318
n_fro =
6.4031
n_inf =
8
```

Для вычисления нормы матрицы используется встроенная в ядро функция `norm()`, второй входной аргумент которой определяет вид вычисляемой нормы. В том случае, если второй параметр не указан, то вычисляется вторая норма.

Матричные нормы используются для оценки числа обусловленности матрицы коэффициентов при неизвестных во время решения системы линейных алгебраических уравнений.

4.2. СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

4.2.1. Основные понятия

При решении инженерных задач достаточно часто приходится сталкиваться с решением *систем линейных алгебраических уравнений* (СЛАУ). СЛАУ, содержащей m уравнений и n неизвестных, называется система вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m, \end{cases} \quad (4.1)$$

где a_{ij} , $i = \overline{1, m}$, $j = \overline{1, n}$ называются *коэффициентами системы*, b_i – *свободными членами*, x_j – *неизвестными*. *Решением системы* (4.1) называется совокупность чисел $x_1 = a$, $x_2 = b$, ..., $x_n = n$, при подстановке которых все уравнения системы обращаются в тождества.

СЛАУ обычно записывают в компактной матричной форме:

$$A \cdot X = B,$$

где A – *матрица коэффициентов системы*, X – *вектор-столбец из неизвестных* и B – *вектор-столбец из свободных членов*:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

Расширенной матрицей называется матрица \bar{A} системы, которая получается после дополнения матрицы коэффициентов столбцом свободных членов:

$$\bar{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{pmatrix}.$$

СЛАУ называется *совместной*, если она имеет хотя бы одно решение, и *несовместной*, если она не имеет ни одного решения.

Совместная система называется *определенной*, если она имеет единственное решение, и *неопределенной*, если имеет более одного решения. В последнем случае каждое решение СЛАУ называется *частным решением*. Совокупность всех частных решений называется *общим решением*.

Решить СЛАУ – значит определить, является ли она совместной или нет. В том случае, если система совместна, то нужно найти ее общее решение.

СЛАУ называется *однородной*, если все свободные члены равны нулю:

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 0, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = 0, \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = 0. \end{array} \right.$$

Однородная СЛАУ всегда совместна, так как $x_1 = x_2 = \dots = x_n = 0$ является решением системы. Это решение СЛАУ называется *нулевым* или *тривиальным*.

Существуют *точные* и *приближенные (итерационные)* методы решения СЛАУ. Метод относится к классу точных, если в предположении отсутствия округлений он дает точное решение задачи после конечного числа арифметических и логических операций. Среди точных методов решения СЛАУ различают: матричный метод, метод с использованием формул Крамера, метод последовательных исключений (метод Гаусса), метод Гаусса с частичным выбором главного элемента, метод Гаусса–Жордана и т.д. К приближенным методам решения СЛАУ относятся: метод Якоби, метод Гаусса–Зейделя и т.д.

В данном пособии рассматриваются только указанные точные методы решения СЛАУ. Более подробную информацию о решении СЛАУ с использованием как точных, так и приближенных методов можно найти в [7, 19, 41, 54].

4.2.2. Теорема Кронекера–Капелли

Для определения совместности системы можно использовать теорему Кронекера–Капелли, смысл которой состоит в следующем: для того, чтобы система линейных уравнений была совместной, необходимо и достаточно, чтобы ранг матрицы A системы был равен рангу ее расширенной матрицы.

Если ранг совместной системы равен числу неизвестных, то система имеет единственное решение. Если ранг совместной системы меньше числа неизвестных, то система имеет бесчисленное множество решений.

Пример. 4.30. Исследовать на совместность следующую систему уравнений:

$$\begin{cases} x_1 + 3 \cdot x_2 + 2 \cdot x_3 = 4, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 1, \\ 3 \cdot x_1 - 5 \cdot x_2 + 4 \cdot x_3 = 2. \end{cases}$$

Решение:

```
>> A=[1 3 2; 2 -1 3; 3 -5 4]; B=[4 1 2]'; [rank(A), rank([A B])]
ans =
2      3
```

Так как ранг расширенной матрицы не равен рангу матрицы коэффициентов системы линейных алгебраических уравнений, то данная система несовместная, т.е. не имеет решений.

4.2.3. Матричный метод решения СЛАУ

Используя свойства матриц, приведенную выше матричную форму СЛАУ можно привести к следующему виду:

$$A^{-1}(AX) = A^{-1}B \Leftrightarrow (A^{-1}A)X = A^{-1}B \Leftrightarrow X = A^{-1}B,$$

где A^{-1} – обратная квадратная матрица.

Следовательно, для решения СЛАУ матричным методом необходимо выполнить следующие шаги:

1. Убедиться в том, что матрица A не является вырожденной (определитель ее не равен нулю), т.е. существует обратная матрица.
2. Найти обратную матрицу A^{-1} .
3. Вычислить произведение обратной матрицы на вектор-столбец свободных членов $A^{-1}B$.
4. Приравнять полученный результат вектору-столбцу X .

Пример 4.31. Решить с помощью матричного метода СЛАУ

$$\begin{cases} 5 \cdot x_1 + 4 \cdot x_2 = 15, \\ x_1 + x_2 = 5. \end{cases}$$

Решение:

1. Вычислим определитель матрицы A : $A = \begin{vmatrix} 5 & 4 \\ 1 & 1 \end{vmatrix} = 5 \cdot 1 - 4 \cdot 1 = 1 \neq 0$.
2. Так как определитель матрицы A не равен нулю, то существует обратная матрица A^{-1} :

$$A^{-1} = \begin{pmatrix} 1 & -4 \\ -1 & 5 \end{pmatrix}.$$

3. Вычислим произведение обратной матрицы на вектор-столбец свободных членов:

$$A^{-1} \cdot B = \begin{pmatrix} 1 & -4 \\ -1 & 5 \end{pmatrix} \cdot \begin{pmatrix} 15 \\ 5 \end{pmatrix} = \begin{pmatrix} 15 - 20 \\ -15 + 25 \end{pmatrix} = \begin{pmatrix} -5 \\ 10 \end{pmatrix}.$$

4. Присвоим полученный результат вектору-столбцу X : $X = \begin{pmatrix} -5 \\ 10 \end{pmatrix}$.

Для решения СЛАУ матричным методом с помощью MATLAB в командное окно следует ввести следующую команду:

```
>> X = inv([5, 4; 1, 1]) * [15; 5]
X =
    -5
    10
```

Главным недостатком матричного метода решения СЛАУ является необходимость определения обратной матрицы, для вычисления которой требуется значительное время. Из-за этого матричный метод решения СЛАУ не применяется в практике.

4.2.4. Формулы Крамера

В основе решения СЛАУ с помощью формул Крамера лежит теорема Крамера, которая формулируется следующим образом: система n уравнений с n неизвестными, определитель которой отличен от нуля, всегда имеет решение, и притом единственное. Оно находится следующим образом: значение каждого из неизвестных равно дроби, знаменателем которой является определитель системы, а числитель равен определителю системы, в котором вместо столбца коэффициентов при неизвестном располагается столбец свободных членов.

Пусть дана система n линейных уравнений с n неизвестными:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n = b_1, \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n = b_2, \\ \dots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \cdots + a_{nn} \cdot x_n = b_n. \end{cases}$$

Из коэффициентов при неизвестных формируется матрица A , из свободных членов – вектор-столбец B .

Определитель матрицы A обозначается Δ и называется *определителем системы*. Следовательно,

$$\Delta = \det A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \neq 0.$$

Если в определителе системы заменить поочередно столбцы коэффициентов при x_1, x_2, \dots, x_n на столбец свободных членов, то получится n определителей:

$$\Delta_{x_1} = \begin{vmatrix} b_1 & a_{12} & \cdots & a_{1n} \\ b_2 & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_n & a_{n2} & \cdots & a_{nn} \end{vmatrix}, \quad \Delta_{x_2} = \begin{vmatrix} a_{11} & b_1 & \cdots & a_{1n} \\ a_{21} & b_2 & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & b_n & \cdots & a_{nn} \end{vmatrix}, \dots,$$

$$\Delta_{x_n} = \begin{vmatrix} a_{11} & a_{12} & \cdots & b_1 \\ a_{21} & a_{22} & \cdots & b_2 \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & b_n \end{vmatrix}.$$

Формулы Крамера для решения системы уравнений имеют вид

$$x_1 = \frac{\Delta_{x_1}}{\Delta}, \quad x_2 = \frac{\Delta_{x_2}}{\Delta}, \quad \dots, \quad x_n = \frac{\Delta_{x_n}}{\Delta} \quad \text{или} \quad x_i = \frac{\Delta_{x_i}}{\Delta}, \quad \text{где } i = 1, 2, \dots, n.$$



Габриэль Крамер (Gabriel Cramer) родился 31 июля 1704 года в Женеве (Швейцария). Установил и опубликовал в 1750 году правило решения систем линейных уравнений с буквенными коэффициентами. Крамер заложил основы теории определителей, но не предложил для них обозначения. Он также занимался исследованием алгебраических кривых высших порядков (исследование особых точек, ветвей и т.п.). Умер 4 января 1752 года в Баньоль (Франция).

Пример 4.32. Решить СЛАУ с помощью формул Крамера

$$\begin{cases} 5 \cdot x_1 + 4 \cdot x_2 = 15, \\ x_1 + x_2 = 5. \end{cases}$$

Решение:

1. Вычислим определитель системы Δ , и если он не равен нулю, то определители Δ_{x_1} и Δ_{x_2} :

$$\Delta = \begin{vmatrix} 5 & 4 \\ 1 & 1 \end{vmatrix} = 1 \neq 0, \quad \Delta_{x_1} = \begin{vmatrix} 15 & 4 \\ 5 & 1 \end{vmatrix} = -5, \quad \Delta_{x_2} = \begin{vmatrix} 5 & 15 \\ 1 & 5 \end{vmatrix} = 10.$$

2. Применим формулы Крамера для определения значений x_1 и x_2 :

$$x_1 = \frac{\Delta_{x_1}}{\Delta} = \frac{-5}{1} = -5, \quad x_2 = \frac{\Delta_{x_2}}{\Delta} = \frac{10}{1} = 10.$$

Таким образом, решение системы есть $x_1 = -5, x_2 = 10$.

Выполним указанные операции в MATLAB:

```
>> A = [5 4; 1 1]; B = [15; 5]; d = det(A)
d =
    1
>> [B A(:,2)], dx1 = det(ans) % вычисление определителя dx1
ans =
    15      4
     5      1
dx1 =
   -5
>> [A(:,1) B], dx2 = det(ans) % вычисление определителя dx2
ans =
     5      15
     1      5
dx2 =
   10
>> x = [dx1/d; dx2/d] % получение решений системы уравнений
x =
   -5
   10
```

Основным недостатком метода Крамера является вычисление определителей, которое занимает очень много времени, из-за чего метод Крамера, как и матричный метод, редко используется в практике.

4.2.5. Метод Гаусса

Существо метода Гаусса состоит в последовательном исключении неизвестных. Систему из n уравнений сначала приводят к равносильной ей системе с треугольной матрицей коэффициентов (так называемый прямой ход). Затем выполняют обратный ход, вычисляя значения x_i и последовательно подставляя их в уравнения для получения значений x_{i-1} .

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array} \right) \Rightarrow \left(\begin{array}{cccc|c} a_{11}^* & a_{12}^* & \cdots & a_{1n}^* & b_1^* \\ 0 & a_{22}^* & \cdots & a_{2n}^* & b_2^* \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn}^* & b_n^* \end{array} \right) \Rightarrow$$

$$x_n = \frac{b_n^*}{a_{nn}^*}, \quad x_{n-1} = \frac{b_{n-1}^* - a_{n-1,n}^* \cdot x_n}{a_{n-1,n-1}^*}, \quad \dots,$$

$$x_1 = \frac{b_1^* - (a_{1n}^* \cdot x_n + a_{1,n-1}^* \cdot x_{n-1} + \dots + a_{12}^* \cdot x_2)}{a_{11}^*}.$$

Коэффициент a_{kk} матрицы A , который используется для исключения элементов a_{rk} , где $r = k+1, k+2, \dots, n$, называется k -м главным элементом. После выполнения прямого хода значения главных элементов в полученной треугольной матрице будут равны единице.

При выполнении прямого хода используют следующие преобразования:

1. Умножение или деление коэффициентов на одно и то же число (масштабирование).
2. Сложение и вычитание уравнений (замещение).
3. Перестановка уравнений системы.
4. Исключение из системы тех уравнений, в которых все коэффициенты при неизвестных и свободные члены равны нулю.



Карл Фридрих Гаусс (Carl Friedrich Gauss) родился 30 апреля 1777 года в Брауншвейге. В первом классе в возрасте 8 лет мгновенно решил задачу отыскания суммы первых 100 натуральных чисел. Построив правильный 17-угольник с помощью циркуля и линейки, первым решил многолетнюю проблему Евклида. После его смерти правильный 17-угольник, вписанный в круг, был выгравирован на его надгробном памятнике. В 1801 году вышла его первая работа «Арифметические исследования», которая определила дальнейшее развитие высшей алгебры и теории чисел. При определении орбиты малой планеты Цереры в 1801 году впервые использовал метод наименьших квадратов. Точное определение орбиты малой планеты, которая была невидимой в тот момент за Солнцем, принесло ему славу среди ученых Европы. В 1832 году совместно с Вебером создал абсолютную систему мер, в которой появились 1 сек., 1 мм, 1 кг и т.д. Поздние работы Гаусса включали исследования по теории вероятностей, электромагнетизму, геодезии и неевклидовой геометрии. Умер в Гётtingене 23 февраля 1855 года.

Пример 4.33. Решить методом Гаусса СЛАУ

$$\begin{cases} 3 \cdot x_1 + 2 \cdot x_2 - x_3 = 4, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 9, \\ x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3. \end{cases}$$

Решение:

Прямой ход:

Так как элемент a_{31} является единственным элементом в первом столбце, который равен единице, то его необходимо установить в качестве главного элемента. Переставим третье уравнение на место первого:

$$\begin{cases} x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 9, \\ 3 \cdot x_1 + 2 \cdot x_2 - x_3 = 4. \end{cases}$$

Расширенная матрица имеет вид

$$\left(\begin{array}{ccc|c} 1 & -2 & 2 & 3 \\ 2 & -1 & 3 & 9 \\ 3 & 2 & -1 & 4 \end{array} \right).$$

С целью исключения первого неизвестного из второго и третьего уравнений ($a_{21} = a_{31} = 0$) необходимо сначала умножить первую строку на 2 и вычесть из 2-й строки, а затем умножить на 3 и вычесть из 3-й строки:

$$\left(\begin{array}{ccc|c} 1 & -2 & 2 & 3 \\ 0 & 3 & -1 & 3 \\ 0 & 8 & -7 & -5 \end{array} \right).$$

Следующий шаг заключается в исключении второго неизвестного из третьего уравнения ($a_{32}^* = 0$). Разделим вторую строку на 3, полученные результаты умножим на 8 и вычтем из 3-й строки:

$$\left(\begin{array}{ccc|c} 1 & -2 & 2 & 3 \\ 0 & 1 & -1/3 & 1 \\ 0 & 0 & -13/3 & -13 \end{array} \right).$$

Полученная после ряда преобразований расширенная матрица соответствует следующей системе уравнений:

$$\left\{ \begin{array}{l} x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3, \\ x_2 - \frac{1}{3} \cdot x_3 = 1, \\ -\frac{13}{3} \cdot x_3 = -13. \end{array} \right.$$

Часть расширенной матрицы, связанной с коэффициентами при неизвестных, приведена к верхнему треугольному виду. Таким образом, завершен прямой ход.

Обратный ход:

Вычислим значение x_3 :

$$-\frac{13}{3} \cdot x_3 = -13 \Rightarrow x_3 = 3.$$

Подставим x_3 во второе уравнение и найдем x_2 :

$$x_2 - \frac{1}{3} \cdot x_3 = 1 \Rightarrow x_2 = 1 + \frac{1}{3} \cdot x_3 = 1 + 1 = 2.$$

Наконец, подставим x_2 и x_3 в первое уравнение и найдем x_1 :

$$x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3 \Rightarrow x_1 = 3 + 2 \cdot x_2 - 2 \cdot x_3 = 3 + 2 \cdot 2 - 2 \cdot 3 = 1.$$

Следовательно, решение СЛАУ имеет вид $x_1 = 1$, $x_2 = 2$, $x_3 = 3$.

Решить СЛАУ методом Гаусса с использованием системы MATLAB можно с помощью следующих команд:

```
>> % представление результатов команд в виде дроби
>> format rational
>> % задание расширенной матрицы
>> A = [1 -2 2; 2 -1 3; 3 2 -1]; B = [3; 9; 4]; AB = [A B]
AB =
    1          -2          2          3
    2          -1          3          9
    3           2         -1          4
>> % обнуление коэффициентов a21 и a31
>> AB1=[AB(1,:); AB(2,:)-AB(1,:).*AB(2,1);...
AB(3,:)-AB(1,:).*AB(3,1)]
AB1 =
    1          -2          2          3
    0           3         -1          3
    0           8         -7         -5
>> % обнуление коэффициента a*32
>> AB2=[AB1(1,:); AB1(2,:); AB1(3,:)-AB1(2,:)./AB1(2,2).*AB1(3,2)]
AB2 =
    1          -2          2          3
    0           3         -1          3
    0           0        -13/3       -13
>> x3=AB2(3,4)./AB2(3,3) % вычисление x3
x3 =
    3
>> x2=(AB2(2,4)-AB2(2,3).*x3)./AB2(2,2) % вычисление x2
x2 =
    2
>> x1=(AB2(1,4)-AB2(1,2).*x2-AB2(1,3).*x3)./AB2(1,1) % x1
x1 =
    1
>> % формирование вектора X
>> X = [x1; x2; x3]'
X =
    1          2          3
```

При выводе результатов решения СЛАУ методом Гаусса с помощью MATLAB использовался дробный формат. Несмотря на наличие различных форматов представления числа в командном окне и в окне редактора данных, ядро системы MATLAB выполняет арифметические действия с вещественными числами двойной точности. Следствием этого является наличие ошибок округления в полученном результате. С целью уменьшения влияния ошибок округления на решение СЛАУ применяют модифицированные методы. Одним из таких методов является *метод Гаусса с частичным выбором главного элемента*.

4.2.6. Метод Гаусса с частичным выбором главного элемента

При выполнении прямого хода в методе Гаусса с частичным выбором главного элемента выбирают главный элемент таким образом, чтобы он был максимальным по абсолютной величине в не приведенной части столбца.

При выполнении метода Гаусса с частичным выбором главного элемента используют следующие преобразования:

1. Масштабирование коэффициентов.
2. Перестановка уравнений (строк) системы.
3. Замещение уравнений.

Пример 4.34. Решить СЛАУ из примера 4.33 методом Гаусса с частичным выбором главного элемента. Оценить полученное решение.

Решение:

При использовании MATLAB для решения системы линейных уравнений методом Гаусса с частичным выбором главного элемента необходимо ввести в командное окно следующие команды:

```
>> % задание расширенной матрицы
>> A = [3 2 -1; 2 -1 3; 1 -2 2]; B = [4; 9; 3]; AB = [A B];
>> % обнуление коэффициентов a21 и a31
>> AB1=[AB(1,:); AB(2,:)-AB(1,:)*AB(2,1)/AB(1,1);...
AB(3,:)-AB(1,:)*AB(3,1)/AB(1,1)]
AB1 =
Columns 1 through 3
3.00000000000000e+000 2.00000000000000e+000 -1.00000000000000e+000
0 -2.3333333333333e+000 3.66666666666667e+000
0 -2.66666666666667e+000 2.3333333333334e+000
Column 4
4.00000000000000e+000
6.33333333333334e+000
1.66666666666667e+000
>> % перестановка 2-й и 3-й строк
>> AB2=[AB1(1,:); AB1(3,:); AB1(2,:)]
```

```

AB2 =
Columns 1 through 3
3.00000000000000e+000 2.00000000000000e+000 -1.00000000000000e+000
0 -2.66666666666667e+000 2.3333333333334e+000
0 -2.33333333333333e+000 3.66666666666667e+000
Column 4
4.00000000000000e+000
1.66666666666667e+000
6.33333333333334e+000
>> % обнуление коэффициента a*32
>> AB3=[AB2(1,:); AB2(2,:); AB2(3,:)-AB2(2,:)*AB2(3,2)/AB2(2,2)]
AB3 =
Columns 1 through 3
3.00000000000000e+000 2.00000000000000e+000 -1.00000000000000e+000
0 -2.66666666666667e+000 2.33333333333334e+000
0 0 1.62500000000000e+000
Column 4
4.00000000000000e+000
1.66666666666667e+000
4.875000000000001e+000
>> x3=AB3(3,4)/AB3(3,3) % вычисление x3
x3 =
3.00000000000000e+000
>> x2=(AB3(2,4)-AB3(2,3)*x3)/AB3(2,2) % вычисление x2
x2 =
2.000000000000001e+000
>> x1=(AB3(1,4)-AB3(1,2)*x2-AB3(1,3)*x3)/AB3(1,1) % x1
x1 =
9.99999999999996e-001
>> % формирование вектора X
>> X = [x1; x2; x3] '
X =
9.99999999999996e-001 2.000000000000001e+000 3.000000000000000e+000

```

Полученное с помощью системы MATLAB решение отличается от точно-го решения. Найденное решение оценивается с помощью двух мер по-грешности: вектора ошибки ($e = x - x^*$, где x – точное решение и x^* – вычисленное решение) и невязки ($r = B - Ax^* = A(x - x^*) = Ae$).

```

>> e = X - [1, 2, 3]
e =
-4.440892098500626e-016 8.881784197001252e-016 4.440892098500626e-016
В рассматриваемом примере ошибки округления малы ( $\approx 4\epsilon$ ). В боль-шинстве случаев точное решение системы неизвестно. В этом случае для оценки найденного решения можно использовать невязку. Невязка – это количественная мера несоответствия между левыми и правыми час-тями уравнений системы при подстановке в левую часть вычисленного решения.

```

```
>> r = (B-A*X) '
r =
4.440892098500626e-016
0 1.332267629550188e-015
```

Значения невязок малы. Однако в случае малых значений невязок не следует, что вектор ошибки также мал. Невязка связана с ошибкой посредством числа обусловленности матрицы A , которое обозначается $\text{cond}(A)$.

Обусловленность – это внутреннее свойство матрицы, которое не связано с методом решения. Число обусловленности матрицы A позволяет определить, насколько чувствительно решение СЛАУ к изменениям в A и B . Значения элементов матрицы A и вектора B в большинстве случаев задаются приближенно, так как являются результатами экспериментов или получены после округления при вычислении с помощью аналитических зависимостей. В зависимости от значения числа обусловленности различают хорошо и плохо обусловленные матрицы. Если число обусловленности большое (>15), то матрица считается плохо обусловленной, в противном случае – хорошо обусловленной. При хорошо обусловленных задачах малые значения невязок приводят к малой ошибке.

Пример 4.36. Вычислить число обусловленности матрицы A из примера 4.33.

Решение:

```
>> A = [3 2 -1; 2 -1 3; 1 -2 2]; cond(A)
ans =
6.4343
```

Вычисление числа обусловленности осуществляется с помощью вызова функции $\text{cond}()$. Так как число обусловленности равно 6.4343, то задача из примера 4.33 хорошо обусловленная. При решении СЛАУ с помощью стандартных функций системы MATLAB используется обратное число обусловленности $\text{rcond}(A)$. Чем ближе это число к единице, тем лучше обусловлена задача. Более подробную информацию о невязках, числах обусловленности и их вычислениях можно получить в [19, 26, 54].

Метод Гаусса с частичным выбором главного элемента является частным случаем метода Гаусса с полным выбором главного элемента, в котором в качестве главного элемента выбирается наибольший по абсолютной величине элемент в не приведенной части матрицы. При использовании метода с полным выбором ведущего элемента скорость увеличения ошибок решения в 2 раза меньше, чем при частичном выборе ведущего элемента [54]. Однако затраты на полный выбор главного элемента сравнимы со всем остальным процессом решения. Следовательно, в практике большее применение нашел метод Гаусса с частичным выбором главного элемента, матричной реализацией которого является LU-разложение.

4.2.7. LU-разложение

Треугольное или LU-разложение представляет собой матричную запись метода Гаусса с частичным выбором главного элемента. Матрица коэффициентов при неизвестных раскладывается на произведение более простых матриц: $A = PLU$, где U (*Upper*) – итоговая *верхняя треугольная матрица*, получаемая после завершения прямого хода; L (*Low*) – *нижняя унитарная треугольная матрица* множителей, использовавшихся при прямом ходе, с учетом перестановок строк; P (*Permutation*) – *матрица перестановок*, содержащая информацию о переупорядочении строк.

Треугольное разложение можно реализовать, не зная правой части. После вычисления матриц P , L и U решение СЛАУ сводится к последовательному решению трех более простых линейных систем:

$$Pz = b; \rightarrow Ly = z; \rightarrow Ux = y.$$

Математическая запись LU-разложения имеет вид

$$x = U^{-1}y = U^{-1}L^{-1}z = U^{-1}L^{-1}P^{-1}b = (PLU)^{-1}b = A^{-1}b.$$

Таким образом, решение при использовании LU-разложения находится за четыре шага:

1. Вычисление матриц P , L и U .
2. Вычисление столбца z .
3. Используя прямую подстановку, находится решение $Ly = z$.
4. Используя обратную подстановку, находится решение $Ux = y$.

Пример 4.37. Решить СЛАУ из примера 4.33 с помощью LU-разложения.

Решение:

Первый шаг состоит в определении матриц P , U и L .

Матрица перестановок имеет следующий вид: $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$,

что свидетельствует о перестановке 2-й и 3-й строк.

Матрица L включает множители, используемые при последовательном исключении неизвестных с учетом перестановки строк:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 7/8 & 1 \end{pmatrix}.$$

Матрица U является верхней треугольной матрицей коэффициентов при неизвестных, полученной после прямого хода:

$$U = \begin{pmatrix} 3 & 2 & -1 \\ 0 & -8/3 & 7/3 \\ 0 & 0 & 13/8 \end{pmatrix}.$$

Следующие шаги заключаются в последовательном решении трех систем:

$$Pz = b \Leftrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 9 \\ 3 \end{pmatrix} \Rightarrow z = \begin{pmatrix} 4 \\ 3 \\ 9 \end{pmatrix};$$

$$Ly = z \Leftrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 7/8 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 9 \end{pmatrix} \Rightarrow y = \begin{pmatrix} 4 \\ 5/3 \\ 39/8 \end{pmatrix};$$

$$Ux = y \Leftrightarrow \begin{pmatrix} 3 & 2 & -1 \\ 0 & -8/3 & 7/3 \\ 0 & 0 & 13/8 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 5/3 \\ 39/8 \end{pmatrix} \Rightarrow x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

При использовании MATLAB в командное окно необходимо ввести следующие команды для решения СЛАУ с помощью LU-разложения:

```
>> A=[3 2 -1; 2 -1 3; 1 -2 2]; B=[4; 9; 3];
```

```
>> [L,U,P]=lu(A) % вычисление матриц
```

L =

$$\begin{matrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 7/8 & 1 \end{matrix}$$

U =

$$\begin{matrix} 3 & 2 & -1 \\ 0 & -8/3 & 7/3 \\ 0 & 0 & 13/8 \end{matrix}$$

P =

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Так как полученные матрицы легко обратимы, то для последовательного получения векторов z , y и x следует применить функцию `inv()`:

```
>> Z=(inv(P)*B)', Y=(inv(L)*Z)', X=(inv(U)*Y)'
```

Z =

$$\begin{matrix} 4 & 3 & 9 \end{matrix}$$

Y =

$$\begin{matrix} 4 & 5/3 & 39/8 \end{matrix}$$

X =

$$\begin{matrix} 1 & 2 & 3 \end{matrix}$$

Операции транспонирования вектора введены для компактности представления результата.

4.2.8. Последовательность решения СЛАУ в системе MATLAB

В системе MATLAB реализованы и другие методы решения СЛАУ, которые для определенного класса задач являются более предпочтительными по сравнению с матричным методом Гаусса. Все методы решения СЛАУ, используемые в MATLAB, реализованы с помощью функций, расположенных в ядре системы. Описание этих функций можно просмотреть с помощью команды `>> help имя_функции` или открыть соответствующий файл, расположенный в директории ...\\MATLAB6p5\\toolbox\\MATLAB\\matfun.

Для решения системы линейных алгебраических уравнений с помощью MATLAB следует применять оператор «\» или функцию `mldivide()`, которые самостоятельно выбирают лучший метод для решения заданной системы уравнений. При этом решение СЛАУ любого порядка достигается одной командой:

```
>> X= (A\b) '
X =
    1           2           3
```

При решении системы линейных алгебраических уравнений с помощью оператора «\» система MATLAB выполняет следующие шаги [75]:

1. Проверяет, является ли матрица A треугольной матрицей, с точностью до перестановки ее строк или столбцов. Если матрица A треугольная, то система решается с помощью обратной подстановки. Если матрица не является треугольной, то это выявляется почти сразу, не занимая много времени.
2. Проверяет, является ли матрица A симметричной и положительно определенной. Если матрица A удовлетворяет условию, то выполняется разложение Холецкого (функция `chol()`), которое позволяет эффективно найти решение [19]. Матрицы, которые не являются положительно определенными, выявляются сразу.
3. Проверяет, является ли матрица A квадратной матрицей общего вида. Если матрица A удовлетворяет условию, то выполняется метод Гаусса с частичным выбором главного элемента (функция `lu()`).
4. Матрица A является прямоугольной матрицей. В этом случае выполняется QR-разложение (функция `qr()`) [19].

Таким образом, если заранее известен метод решения СЛАУ, то вместо оператора «\» следует использовать соответствующие функции (`lu()`, `chol()` и т.д.), исключая потерю времени на выбор метода решения.

При решении СЛАУ с использованием оператора «\» могут появляться следующие диагностические сообщения:

1. Если матрица A вырожденная, т.е. $\det(A) = 0$

Warning: Matrix is singular to working precision.

2. Если матрица A близка к вырожденной $\text{cond}(A) \gg 0$ ($r\text{cond} \approx 0$)

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = XXX.

При появлении диагностических сообщений решение либо не может быть найдено, либо может быть не точным.

4.3. ВЫВОДЫ К ГЛАВЕ 4

На примерах, приведенных в данной главе, продемонстрировано использование MATLAB:

1) при работе с матрицами:

- создание матриц, в том числе специальных;
- выполнение линейных операций над матрицами;
- выполнение элементарных матричных преобразований;
- произведение матриц;
- нахождение определителя матрицы;
- вычисление обратной матрицы;
- определение ранга матрицы;
- вычисление первой, второй и бесконечной норм матрицы;

2) при решении системы линейных алгебраических уравнений:

- матричным методом;
- используя формулы Крамера;
- с помощью метода Гаусса с частичным выбором главного элемента;
- используя LU-разложение;

Наряду с рассмотрением матричных операций рассмотрены их свойства, а также свойства определителя и ранга матрицы.

При выполнении примеров, приведенных в этой главе, использовались стандартные функции системы MATLAB, перечень которых представлен в табл. 4.1.

Таблица 4.1

Перечень функций для работы с матрицами и решения СЛАУ

Оператор	Имя функции	Описание
,	ctranspose	Транспонирование матрицы с комплексным со- прежнением
	det	Вычисление детерминанта

Оператор	Имя функции	Описание
[], [,]	horzcat	Горизонтальное соединение данных
	inv	Определение обратной матрицы
	isequal	Проверка равенства двух матриц
	lu	Решение СЛАУ с помощью LU-разложения
	magic	Создание магического квадрата
-	minus	Вычитание матриц
\	mldivide	Решение СЛАУ с автоматическим выбором метода решения
*	mtimes	Умножение двух матриц
	norm	Вычисление нормы матрицы
	pascal	Создание матрицы Паскаля, состоящей из коэффициентов бинома $(1+n)^n$
+	plus	Суммирование двух матриц
	rank	Определение ранга матрицы
	sum(M, n)	Вычисление суммы элементов матрицы M по строкам ($n = 2$) или столбцам ($n = 1$)
. *	times	Поэлементное умножение двух матриц
. '	transpose	Транспонирование матрицы
[;]	vertcat	Вертикальное соединение данных

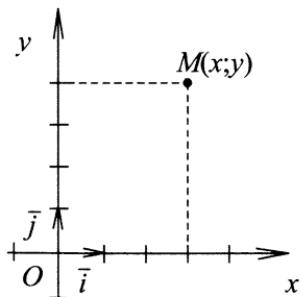
Г л а в а 5. РЕШЕНИЕ ЗАДАЧ АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ

Система MATLAB обладает развитыми средствами для визуализации графической информации. Используя предоставляемые системой графические средства, можно успешно решать различные задачи. В данной главе рассматривается использование графических возможностей системы MATLAB при построении геометрических примитивов на плоскости и в пространстве, а также решение ряда сопутствующих задач из курса аналитической геометрии.

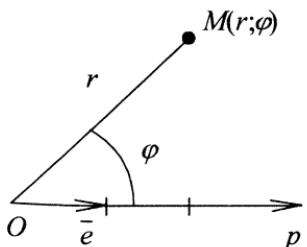
5.1. АНАЛИТИЧЕСКАЯ ГЕОМЕТРИЯ НА ПЛОСКОСТИ

5.1.1. Системы координат на плоскости

Под *системой координат* понимают способ, позволяющий численно описать положение точки на плоскости. Наибольшее распространение в практике получили две системы координат: *прямоугольная* (декартова) и *полярная*.



Р и с . 5.1. Декартова система координат



Р и с . 5.2. Полярная система координат

Прямоугольная система координат, представленная на рис. 5.1, задается двумя взаимно перпендикулярными прямыми – осями, на каждой из которых выбрано положительное направление и задан единичный (масштабный) отрезок. Систему координат обозначают Oxy (или Oij , где i, j – единичные орты соответствующих осей). Точку пересечения осей O называют *началом координат*. Координатами *произвольной точки* в системе координат Oxy называются два числа x и y , которые однозначно определяют точку на координатной плоскости. Координаты точки M записывают следующим образом: $M(x; y)$. При этом x называется *абсциссой точки*, а y – *ординатой*.

Полярная система координат, изображенная на рис. 5.2, задается точкой O , называемой *полюсом*, лучом Op , который носит название *полярной оси*, и единичным вектором \bar{e} того же направления, что и луч Op . Положение точки определяется двумя числами: ее расстоянием r до полюса и углом φ , образованным отрезком от точки до полюса O и полярной осью Op . При этом отсчет угла ведется от полярной оси в направлении, противоположном движению часовой стрелки. Числа r

и φ называются *полярными координатами* точки M . Координаты точки записывают следующим образом $M(r; \varphi)$, где r – называют *полярным радиусом*, φ – *полярным углом*.

Для получения аналитических выражений, которые связывают две системы координат между собой, необходимо совместить начало координат системы Oxy с полюсом O , а полярную ось Op – с положительной полуосью Ox .

При преобразовании одних координат в другие используются следующие аналитические зависимости:

$$\begin{cases} x = r \cdot \cos \varphi, \\ y = r \cdot \sin \varphi, \end{cases} \quad \begin{cases} r = \sqrt{x^2 + y^2}, \\ \operatorname{tg} \varphi = y/x. \end{cases}$$

Система MATLAB позволяет строить графики на плоскости в двух системах координат: прямоугольной и полярной.

Для построения графиков линий на плоскости в декартовой системе координат применяются функции `plot(x1,y1,s1, x2,y2,s2,...)` и `line()`. Функция `plot()` используется для построения одной или нескольких линий в одной координатной плоскости. С помощью входных аргументов функции `plot()` можно изменять свойства линии, принятые по умолчанию. Несмотря на то что функция `plot()` может принимать различное число входных аргументов, в ряде случаев последовательность их задания является фиксированной. Если задан один входной аргумент, то функция `plot()` строит график, где абсциссы являются номерами элементов входного массива, а ординаты – значениями соответствующих элементов массива. При задании двух аргументов первым должен быть *массив абсцисс*, а вторым – *массив ординат*. При этом массивы должны быть арифметического класса. В том случае, если необходимо изменить свойства выводимой линии, то применяется третий аргумент, состоящий из символов, которые заключены в апострофы. Строковый аргумент позволяет указать цвет и стиль линии, а также стиль маркеров, выводимых в координатных точках, которые соединяются прямой линией. Все возможные варианты маркеров, цветов и стилей линий представлены в табл. 5.1. Для построения еще одного графика задается следующая тройка параметров. При построении нескольких графиков третий параметр для каждого графика можно не указывать. В этом случае графики будут выводиться в формате, заданном по умолчанию. При выводе нескольких графиков с помощью одной команды используется следующая, применяемая по умолчанию, циклическая последовательность задания цветов: синий, темно-зеленый, красный, темно-голубой, темно-пурпурный, темно-желтый и серый. Функция `plot()` является надстройкой над функцией-конструктором объекта линии `line()`, которая создает линию внутри координатной плоскости, ограниченной координатными осями. С помощью функции `line()` можно не только строить график линии, но и задавать все свойства объекта Line

во время его создания, т.е. построения линии. Функции `plot()` и `line()` являются встроенными функциями в ядро системы MATLAB.

Построение графика линии на плоскости в полярной системе координат осуществляется с помощью функции `polar(phi, rho, s)`. Функция `polar()` в отличие от функции `plot()` позволяет строить только один график. Первый аргумент функции определяет полярный угол в радианах, второй – полярный радиус, третий аргумент, который является необязательным, служит для задания свойств линии и маркеров. Функция `polar()` при построении линии готовит необходимые данные для передачи в функцию `plot()`, производит построение *концентрических окружностей*, которые обозначают полярный радиус, и выводит обозначения полярных углов по окружности самого большого радиуса. При этом прямоугольная координатная плоскость, относительно которой происходит построение, делается невидимой. Возможные варианты задания типов линий и маркеров, а также их цветов на полярной координатной плоскости приведены в табл. 5.1.

Таблица 5.1

Таблица обозначений цветов и типов линий и маркеров

Цвет линии и маркера [код цвета в палитре RGB]	Тип маркера	Тип линии
b синий (blue) [0 0 1]	.	- сплошная
g зеленый (green) [0 1 0]	o круг	: пунктирная
r красный (red) [1 0 0]	x ×	-. штрих-пунктирная
c голубой (cyan) [0 1 1]	+	-- штриховая
m фиолетовый (magenta) [1 0 1]	*	звезда
y желтый (yellow) [1 1 0]	s квадрат (square)	
k черный (black) [0 0 0]	d ромб (diamond)	
w белый (white) [1 1 1]	p пятиконечная звезда (pentagram)	
	h шестиконечная звезда (hexagram)	
	^, <, >, v треугольники	

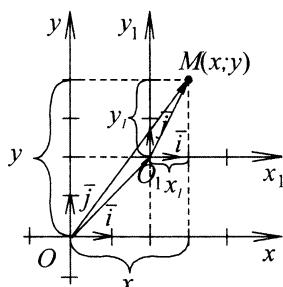
Перед построением линии в координатной плоскости система MATLAB выполняет следующие шаги:

- Осуществляет поиск текущего графического окна (`figure`). Текущим считается окно, которое было активным последним. Если графического окна нет, то оно создается.

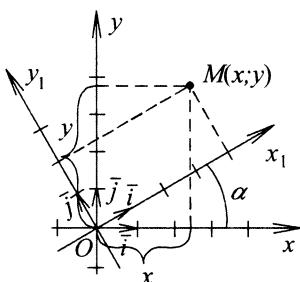
2. Осуществляет поиск текущей координатной плоскости (Axes) внутри текущего графического окна. Текущей считается та координатная плоскость, которая последней использовалась для построения линии. Если координатной плоскости нет, то она создается.

5.1.2. Преобразование координат из одной системы в другую

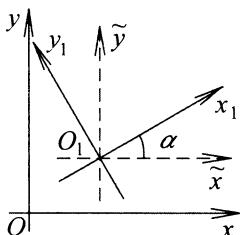
Переход от одной системы координат в какую-либо другую называется *преобразованием координат*.



Р и с . 5.3. Параллельный перенос осей



Р и с . 5.4. Поворот осей



Р и с . 5.5. Параллельный перенос и поворот осей

Рассмотрим два варианта преобразования: *параллельный перенос осей* и *поворот осей координат*. Под параллельным переносом осей координат понимают переход от системы координат Oxy к новой системе координат – $O_1x_1y_1$, при котором меняется положение начала координат, а направление осей и масштаб остаются неизменными (рис. 5.3). Аналитические выражения, которые используются при параллельном переносе осей координат, имеют вид

$$\begin{cases} x = x_0 + x_1, \\ y = y_0 + y_1. \end{cases}$$

Под поворотом осей координат называют такое преобразование координат, при котором обе оси поворачиваются на один и тот же угол, а начало координат и масштаб остаются неизменными (рис. 5.4). Формулы поворота осей имеют вид:

$$\begin{cases} x = x_1 \cdot \cos \alpha - y_1 \cdot \sin \alpha, \\ y = x_1 \cdot \sin \alpha + y_1 \cdot \cos \alpha. \end{cases}$$

Если новая система координат получена из старой путем параллельного переноса осей координат и поворотом осей на угол α (рис. 5.5), то формулы, описывающие эти преобразования, имеют следующий вид:

$$\begin{cases} x = x_1 \cdot \cos \alpha - y_1 \cdot \sin \alpha + x_0, \\ y = x_1 \cdot \sin \alpha + y_1 \cdot \cos \alpha + y_0. \end{cases}$$

На рис. 5.5 оси \tilde{x} и \tilde{y} являются осями вспомогательной системы координат.

5.1.3. Линии первого порядка

Линией на плоскости называется множество точек, которые обладают общим для них геометрическим свойством.

Уравнением прямой или кривой линии на плоскости Oxy называется такое уравнение $F(x,y) = 0$ с двумя переменными, которому удовлетворяют координаты x и y каждой точки линии и не удовлетворяют координаты любой точки, не лежащей на этой линии. Переменные x и y в уравнении линии носят название *текущих координат точек линии*.

5.1.3.1. Общее уравнение прямой линии

Общее уравнение прямой линии имеет вид

$$Ax + By + C = 0, \quad (5.1)$$

где A, B, C – произвольные числа, причем A и B не равны нулю одновременно.

Частные случаи формулы (5.1):

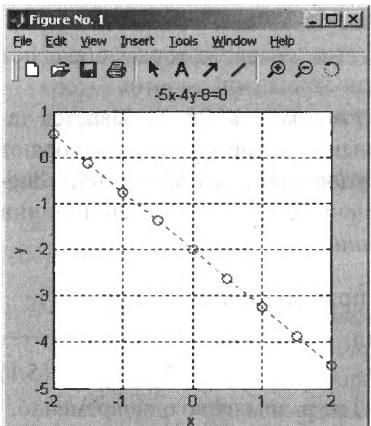
1. $A = 0$, уравнение прямой приводится к виду $y = -C/B$. Это уравнение прямой, параллельной оси Ox ;
2. $B = 0$, прямая параллельна оси Oy ;
3. $C = 0$, прямая линия проходит через начало координат.

Пример 5.1. Построить пунктирную линию красного цвета, заданную общим уравнением $-5x - 4y - 8 = 0$. Значения абсцисс точек линии изменяются в диапазоне $[-2;2]$ с шагом 0.5. В вычисленных точках вывести круговые маркеры красного цвета. Заголовком графика является общее уравнение прямой линии.

Решение:

```
>> A=-5; B=-4; C=-8; % задание коэффициентов уравнения
>> x=-2:0.5:2; % формирование диапазона абсцисс
>> y=-(A*x+C)/B; % вычисление значений ординат
>> plot(x,y,':ro') % построение графика прямой линии
>> grid on % визуализация координатной сетки
>> title('5x-4y+8=0') % задание заголовка
>> xlabel('x'), ylabel('y') % обозначение осей
```

Результаты приведенных выше команд показаны на рис. 5.6. Во время выполнения команд создается графическое окно с прямоугольной системой координат, в которой строится график прямой линии. Перед построением графика были созданы коэффициенты уравнения и два массива одинакового размера x и y . Третий входной аргумент функции `plot()` указывает на то, что линия – пунктирная (`:`), маркеры – круги (`o`), цвет – красный (`r`). Символы для задания типов линии и маркеров, а также их цвета могут записываться в любой последовательности. Тот же самый результат можно было получить с помощью следующей последовательности символов '`o:r`'. После построения графика прямой линии с помощью команды `>> grid on` выводится коор-



Р и с . 5.6. График прямой линии, заданной общим уравнением

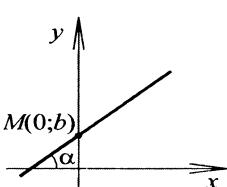
динатная сетка. Для удаления координатной сетки следует задать команду `>> grid off`. Функции `title()`, `xlabel()` и `ylabel()` принимают в качестве входного аргумента последовательность символов, заключенную в апострофы. Эти функции используются для вывода поясняющих текстовых надписей над координатной плоскостью (`title()` – заголовок графика), под координатной плоскостью (`xlabel()` – обозначение оси абсцисс) и слева от координатной плоскости (`ylabel()` – обозначение оси ординат).

Рассмотрим различные способы задания общего уравнения прямой линии, которые применяются в теории и практике.

5.1.3.2. Уравнение прямой линии с угловым коэффициентом

Пусть на плоскости Oxy задана произвольная прямая, не параллельная оси Oy . Ее положение полностью определяется ординатой b точки $M(0;b)$ пересечения с осью Oy и углом α между осью Ox и прямой.

Линия, изображенная на рис. 5.7, описывается уравнением прямой с угловым коэффициентом:



Р и с . 5.7. Линия с угловым коэффициентом

$$y = kx + b,$$

где число $k = \operatorname{tg} \alpha$ – угловой коэффициент прямой.

Общее уравнение прямой линии в данном случае:

$$y = -(A/B)x - (C/B).$$

П р и м е р 5.2. Построить три прямые линии с ординатой $b = 0.5$ и углами наклона $\alpha_1 = \pi/8$; $\alpha_2 = \pi/4$; $\alpha_3 = 3\pi/8$. Первую линию обозначить только маркерами синего цвета. Вторая линия – красного цвета и штрих-пунктирная без маркеров. Третья линия – зеленого цвета с маркерами в виде звезд. Значения абсцисс линий изменяются с шагом 0.5 в диапазоне $[-2;2]$. Координатная плоскость ограничена значениями абсцисс и ординат –1 и 2. Заголовком графиков являются уравнение прямой, значение ординаты и значения углов.

Р е ш е н и е :

```
>> b = 0.5; % задание общей ординаты прямых линий
>> x = -2:0.5:2; % диапазон изменения абсцисс
```

```
>> alpha = [pi/8; pi/4; 3*pi/8]; % вектор углов
>> k = tan(alpha); % угловые коэффициенты
>> y = k*x + b; % уравнение прямой линии
>> plot(x,y(1,:),'bo',x,y(2,:),'-.r') % вывод двух линий
>> hold on % включение режима добавления графиков
>> grid on % отображение координатной сетки
>> plot(x, y(3,:), '-g*') % график третьей линии
>> title('y=k*x+b, b=0.5, alpha=[pi/8; pi/4; 3*pi/8]')
>> xlabel('x'), ylabel('y') % обозначение осей x и y
>> axis([-1 2 -1 2]) % задание границ координатных осей
```

Результаты выполнения вышеприведенных команд показаны на рис. 5.8.

При построении графиков линий использовалась команда `>> hold on`, которая включает режим добавления графиков. Таким образом, при задании новой функции `plot()` новые графики будут выводиться в текущее графическое окно без удаления находящихся в координатной плоскости графиков. Для отключения режима добавления графиков следует задать команду `hold off`. Переключение между двумя режимами может осуществляться заданием команды `hold` без параметров. Для задания диапазона координатных осей использовалась функция `axis([xmin xmax ymin ymax])`, входным аргументом которой является массив, состоящий из четырех элементов. Значения первого и второго элементов этого массива задают минимальное и максимальное значения оси абсцисс соответственно. Значения третьего и четвертого элементов задают нижнюю и верхнюю границы диапазона значений оси ординат соответственно. Изменение диапазона отображения осей координат не влияет на число элементов в массивах данных, которые связаны с соответствующими осями.

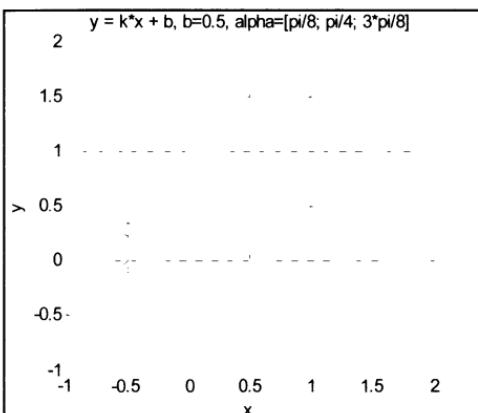


Рис. 5.8. Графики линий, заданных уравнением с угловым коэффициентом

ми осями. То есть во время изменения диапазона отображения координатной оси x с $[-2;2]$ на $[-1;2]$ переменная $x = [-2;2]$ не изменяется.

5.1.3.3. Уравнение прямой линии, проходящей через данную точку в заданном направлении

Пусть прямая линия проходит через точку $M(x_0, y_0)$ и ее направление характеризуется угловым коэффициентом k . Уравнение этой прямой можно

записать в виде $y = kx + b$, где b – неизвестная величина. Так как прямая линия проходит через точку $M(x_0; y_0)$, то координаты точки удовлетворяют уравнению прямой $y_0 = kx_0 + b$, откуда $b = y_0 - kx_0$. Подставив значение b в исходное уравнение, получим уравнение прямой, проходящей через данную точку в данном направлении: $y_0 = kx + y_0 - kx_0$ или $y - y_0 = k(x - x_0)$. Таким образом, уравнение прямой с угловым коэффициентом является частным случаем рассмотренного уравнения, когда $x_0 = 0$.

Пример 5.3. Построить две прямые линии, каждая из которых задана с помощью координат точки и углового коэффициента: $M_1(-1; 1)$, $\alpha_1 = \pi/8$ и $M_2(1; -1)$, $\alpha_2 = \pi/4$. Первая линия – сплошная, синего цвета с маркерами в виде кругов в вычисленных точках. Вторая линия – штрих-пунктирная, красного цвета с маркерами в виде треугольников, вершины которых направлены наверх. Вывести обозначения точек в виде $M_1(x_1; y_1)$ и $M_2(x_2; y_2)$. Обозначение второй точки оформить с помощью шрифта Times. Вывести обозначения осей и заголовок графиков. Заголовок представляет собой уравнение прямой линии, проходящей через данную точку в заданном направлении $y - y_0 = k(x - x_0)$.

Решение:

```
>> % исходные данные: значения абсцисс и угловые коэффициенты
>> x=-2:0.5:2; alpha = [pi/8, pi/4]; k = tan(alpha);
>> m1=[-1;1]; m2=[1;-1]; % задание точек
>> y1 = k(1)*(x-m1(1)) + m1(2); % ординаты первой линии
>> y2 = k(2)*(x-m2(1)) + m2(2); % ординаты второй линии
>> plot(x,y1,'-bo',x,y2,'-.r^') % построение графиков
>> grid on % показ координатной сетки
>> title('y - y_{0} = k * (x - x_{0})') % заголовок
>> xlabel('x'), ylabel('y') % обозначение осей координат
>> % обозначение точки M1(x1,y1)
>> text(-1.4,1.4,'M_{1}(x_{1},y_{1})')
>> % обозначение точки M2(x2,y2)
>> text(1,-1.5,'M_{2}(x_{2},y_{2})')
```

Результаты выполнения команд приведены на рис. 5.9. Для обозначения точек на координатной плоскости использовалась функция `text()`. Первые два аргумента функции задают координаты начала вывода текстовой информации в координатной плоскости. Первый аргумент соответствует оси x , а второй аргумент – оси y . Третий аргумент представляет собой выводимую в графическое окно текстовую информацию. Этот аргумент задается в виде последовательности символов, заключенных в апострофы. При задании заголовка и при обозначении точек на координатной плоскости использовалась запись издательской системы LaTeX, некоторые возможности которой

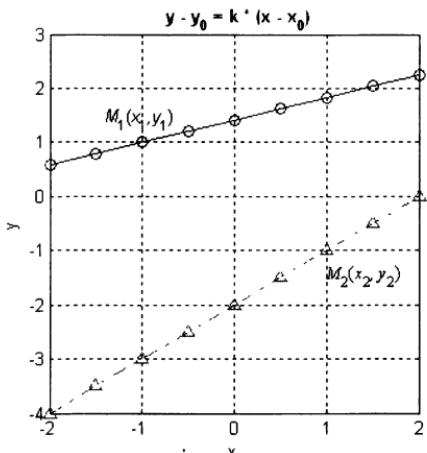


Рис. 5.9. Графики линий, заданных с помощью точки и угла наклона

встроены в систему MATLAB. Применяя данную издательскую систему, можно изменять шрифт, его свойства, а также вводить в графическое окно греческие и специальные символы. Издательская система LaTeX была разработана в 1985 году Лесли Лэмпортом (Leslie Lamport) на базе издательской системы TeX. Система верстки TeX была создана Дональдом Кнутом (Donald Knuth) в 1977 году и представляет собой специализированный язык программирования, который включает команды, макроопределения и является аппаратно независимым. Более полную информацию об издательской системе LaTeX можно получить в [29, 36]. В табл. 5.2 представлены некоторые команды системы LaTeX, которые можно использовать для оформления графиков в графическом окне системы MATLAB.

5.1.3.4. Уравнение прямой, проходящей через две точки

Пусть прямая линия проходит через две точки $M_1(x_1; y_1)$ и $M_2(x_2; y_2)$. Уравнение прямой, проходящей через точку M_1 , имеет вид $y - y_1 = k(x - x_1)$, где k – неизвестная величина, характеризующая угол наклона прямой линии. Так как прямая линия проходит и через точку M_2 , то координаты этой точки должны удовлетворять уравнению $y_2 - y_1 = k(x_2 - x_1)$. Откуда $k = (y_2 - y_1)/(x_2 - x_1)$. Подставляя найденное значение k в уравнение прямой линии, проходящей через данную точку в заданном направлении, получаем уравнение прямой, проходящей через две точки:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \Leftrightarrow y = \frac{(y_2 - y_1)(x - x_1)}{x_2 - x_1} + y_1.$$

Пример 5.4. Создать графическое окно для отображения четырех координатных плоскостей. В левой верхней (первой) координатной плоскости построить прямую линию, проходящую через две точки $M_1(1; -1)$ и $M_2(-1; 0)$. Абсцисса изменяется с шагом 0.5 в диапазоне $[-2; 2]$. Линия штриховая красного цвета. В местах вычисленных значений координат точек линии отобразить маркеры в виде пятиконечной звезды. Вывести обозначения точек $M_1(x_1; y_1)$ и $M_2(x_2; y_2)$. Обозначить координатные оси x и y со

ответственно, а также задать заголовок с помощью уравнения прямой, проходящей через две точки $(y - y_1)/(y_2 - y_1) = (x - x_1)/(x_2 - x_1)$. Вывести координатную сетку.

Решение:

```
>> m1 = [1;-1]; m2 = [-1;0]; % задание координат точек
>> x=-2:0.5:2; % значения абсцисс
>> % вычисление ординат прямой линии, заданной двумя точками
>> y=(m2(2)-m1(2))*(x-m1(1))/(m2(1)-m1(1))+m1(2);
>> % создание четырех областей в графическом окне
>> % и активизация первой области
>> subplot(2,2,1)
>> plot(x,y,'--rp') % построение графика линии
>> grid on % визуализация координатной сетки
>> xlabel('x'), ylabel('y') % обозначение осей координат
>> % вывод заголовка графика прямой линии
>> title('(y-y_{1})/(y_{2}-y_{1})=(x-x_{1})/(x_{2}-x_{1})')
>> text(0.8,-0.87,'M_{1}(x_{1};y_{1})') % точка M1(x1;y1)
>> text(-1.2,0.2,'M_{2}(x_{2};y_{2})') % точка M2(x2;y2)
```

В результате выполнения вышеприведенных команд получился график прямой линии, представленный на рис. 5.10, в левой верхней области графического окна. При построении графика использовалась функция `subplot()`, которая позволяет разбивать графическое окно на ряд областей с последующим размещением в каждой из них своей координатной системы. Первый входной аргумент в функцию определяет число областей по горизонтали, второй – по вертикали, а третий задает текущую область. Нумерация областей начинается с левой верхней области и выполняется построчно слева направо. Номер текущей области не должен превышать максимально допустимое число областей, т.е. произведение первых двух входных аргументов. В противном случае система MATLAB выдаст следующую ошибку: `??? Error using ==> subplot. Index exceeds number of subplots.` (`??? Ошибка при использовании ==> функции subplot. Индекс превышает число областей.`). Все последующие команды строят график и выводят обозначения для графика, осей и точек в координатной плоскости текущей области.

5.1.3.5. Уравнение прямой линии в отрезках

Пусть прямая пересекает ось Ox в точке $M_1(a;0)$, а ось Oy – в точке $M_2(0;b)$. В этом случае уравнение прямой линии будет иметь вид

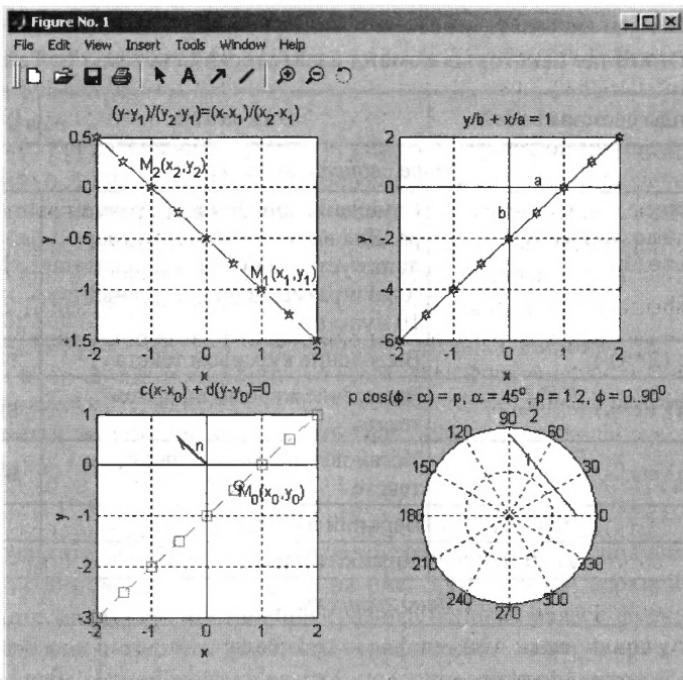
$$\frac{y-0}{b-0} = \frac{x-a}{0-a} \Leftrightarrow \frac{x}{a} + \frac{y}{b} = 1.$$

Это уравнение называется *уравнением прямой в отрезках*, так как числа a и b определяют, какие отрезки отсекает прямая линия при пересечении с осями x и y соответственно.

Таблица 5.2

Описание некоторых команд издательской системы LaTeX

Команды системы LaTeX	Описание	Результат
<i>Задание свойств шрифта</i>		
<code>\fontname{arial}</code> <code>\fontsize{10}</code> <code>M(x, y)</code>	Изменение шрифта и его размера. Для вывода точки и ее координат устанавливается шрифт Arial и размер шрифта, равный 10 пунктам	$M(x,y)$
<code>sin{\it(2*x)}</code>	Выделение курсивом текста	$\sin(2*x)$
<code>\bfsin\rm(2*x)</code>	Выделение жирным шрифтом текста	$\mathbf{\sin}(2*x)$
<code>\bfsin\rm(2*x)</code>	Установка прямого шрифта в тексте	$\sin(2*x)$
<code>x^2</code>	Верхний индекс	x^2
<code>x_{2}</code>	Нижний индекс	x_2
<i>Ввод греческих букв</i>		
<code>\alpha \beta \gamma \delta \varepsilon \eta</code> <code>\theta \kappa \lambda \mu \nu \xi</code> <code>\rho \sigma \tau \phi \chi \psi \omega</code> <code>\Gamma \Delta \Theta \Lambda \Xi</code> <code>\Sigma \Phi \Psi \Omega</code>		$\alpha \beta \gamma \delta \varepsilon \eta$ $\theta \kappa \lambda \mu \nu \xi$ $\rho \sigma \tau \phi \chi \psi \omega$ $\Gamma \Delta \Theta \Lambda \Xi$ $\Sigma \Phi \Psi \Omega$
<i>Ввод специальных символов</i>		
<code>\leftarrow \uparrow</code> <code>\rightarrow \downarrow</code> <code>\leftrightarrow</code>	Указатели	\leftarrow \rightarrow \leftrightarrow
<code>\newline</code>	Начало новой строки	
<code>\leq \geq \neq</code>	Операции сравнения: меньше либо равно, больше либо равно и не равно	$\leq \geq \neq$
<code>\oplus \vee \wedge</code>	Логические операции: сумма по модулю 2, дизъюнкция и конъюнкция	$\oplus \vee \wedge$
<code>\pm</code>	Плюс-минус	\pm
<code>\times \div</code>	Умножение и деление	$\times \div$
<code>\int \partial</code>	Интеграл и дифференциал	$\int \partial$
<code>\infty</code>	Бесконечность	∞
<code>\in</code>	Принадлежит	\in



Р и с . 5.10. Способы задания прямой линии

Пример 5.5. Построить сплошную прямую линию синего цвета, заданную в отрезках, в правом верхнем углу уже созданного графического окна. В местах вычисленных значений координат линии вывести маркеры в виде шестиконечной звезды. На координатной плоскости отобразить координатные оси в виде прямых линий черного цвета. Вывести обозначения отрезков, которые отсекает прямая линия при пересечении с координатными осями, а также обозначения осей. Заголовок представляет собой уравнение прямой линии, заданной в отрезках $y/b + x/a = 1$. Вывести координатную сетку.

Решение:

```
>> a=1; b=-2; x=-2:0.5:2; % начальные данные
>> y=(1-x/a)*b; % вычисление значений ординат
>> subplot(2,2,2) % активизация второй координатной плоскости
>> plot(x,y,'-bh')%, grid on % построение линии и вывод сетки
>> line([-2 2],[0 0], 'Color', 'black') % вывод оси x
>> line([0 0],[-6 2], 'Color', 'black') % вывод оси y
>> text(0.45,0.3,'a'), text(-0.2,-1,'b') % обозначение отрезков
>> xlabel('x'), ylabel('y') % обозначение осей
>> title('y/b+x/a=1') % вывод заголовка
```

Результаты выполнения команд приведены в графическом окне на рис. 5.10 в правом верхнем углу. Для визуализации координатных осей после построения графика прямой линии, заданной в отрезках, использовалась функция `line()`. Эта функция позволяет строить на координатной плоскости прямые линии по точкам, координаты которых являются входными аргументами в функцию. Первый входной аргумент функции представляет собой массив абсцисс точек, второй аргумент – массив ординат. При этом соответствующие элементы массивов образуют координаты точки. Далее следует пара входных аргументов, задаваемых в виде строки символов, заключенных в апострофы. Парные входные аргументы используются для задания свойств объектов, выводимых в графическом окне, одним из которых является линия. Первым аргументом в паре является имя свойства, вторым – его значение. Первая команда с функцией `line()` выводит на координатную плоскость ось абсцисс, а вторая команда – ось ординат. Обе оси имеют один и тот же черный цвет.

5.1.3.6. Уравнение прямой линии, проходящей через заданную точку перпендикулярно заданному вектору

Уравнение прямой линии, проходящей через заданную точку $M_0(x_0; y_0)$ перпендикулярно заданному ненулевому вектору $\bar{n} = (c; d)$, находится из равенства нулю скалярного произведения двух перпендикулярных векторов. Возьмем на прямой линии произвольную точку $M(x; y)$ и рассмотрим вектор $\overline{M_0M} = (x - x_0; y - y_0)$. Так как векторы \bar{n} и $\overline{M_0M}$ перпендикулярны, то их скалярное произведение равно нулю: $\bar{n} \cdot \overline{M_0M} = 0$, т.е.

$$c(x - x_0) + d(y - y_0) = 0.$$

Вектор $\bar{n} = (c; d)$, перпендикулярный прямой, называется *нормальным вектором этой прямой*, или *нормалью*.

Приведенное выше уравнение можно переписать в виде $Ax + By + C = 0$, где A и B – координаты нормального вектора, $C = -cx_0 - dy_0$ – свободный член.

Пример 5.6. Построить в левом нижнем углу графического окна штрих-пунктирную зеленого цвета прямую линию, проходящую через точку $M(0.6; -0.4)$ перпендикулярно вектору $\bar{n} = (-1; 1)$. Вывести маркеры в виде квадратов в местах вычисленных значений координат точек линии. Отобразить координатные оси черным цветом. Вывести обозначение заданной точки $M_0(x_0; y_0)$, вектора \bar{n} и координатных осей. Построить на координатной плоскости вектор \bar{n} . В качестве заголовка задать уравнение прямой, проходящей через заданную точку перпендикулярно заданному вектору.

Решение:

```

>> n=[-1;1]; % определение вектора
>> m=[0.6;-0.4]; % задание точки
>> subplot(2,2,3) % установка текущей области
>> y = m(2)-n(1)*(x-m(1))/n(2); % вычисление ординат
>> plot(x,y,'-.gs') % построение графика линии
>> % показ сетки и включение режима добавления графиков
>> grid on, hold on
>> % вывод координатных осей
>> line([-2 0; 2 0],[0 -3; 0 1],'Color','black')
>> xlabel('x'), ylabel('y') % обозначение осей
>> title('c*(x-x_{0})+d*(y-y_{0})=0') % заголовок
>> plot(m(1),m(2),'bo') % визуализация заданной точки
>> text(0.6,-0.6,'M_{0}(x_{0};y_{0})') % ее обозначение
>> line([0 -0.5 -0.42], [0 0.5 0.3], 'Color', [1 0 0], ...
'LineWidth',1.5) % визуализация вектора
>> text(-0.2,0.4,'n') % обозначение вектора

```

Результаты вышеприведенных команд представлены в графическом окне на рис. 5.10 в левом нижнем углу. При построении координатных осей в качестве аргументов функции `line()`, задающих координаты точек, использовались массивы размером 2×2 . Первый столбец этих массивов содержит координаты начала и конца оси абсцисс, а второй столбец – координаты оси ординат. Во втором случае функция `line()` была использована для изображения на координатной плоскости вектора, состоящего из трех точек, соединенных двумя прямыми линиями. Соединение точек происходит в той последовательности, в которой заданы их соответствующие координаты на входе в функцию `line()`. Для задания цвета линии в функции `line()` использовался массив, состоящий из трех элементов. Первый элемент обозначает красный цвет (`red`), второй – зеленый (`green`) и третий – синий (`blue`). То есть при задании цвета используется аддитивная цветовая модель `RGB`, в которой любой цвет является комбинацией трех основных цветов. Элементы этого массива могут принимать значения от нуля до единицы, характеризуя интенсивность соответствующего основного цвета. Объединение полной интенсивности красного, зеленого и синего цветов формирует белый цвет – `[1 1 1]`. Отсутствие интенсивности основных цветов дает в результате черный цвет – `[0 0 0]`. Свойство `LineWidth` позволяет задавать ширину линии в пунктах. Один пункт равен $1/72$ дюйма. По умолчанию ширина линии равна 0.5 пункта.

5.1.3.7. Полярное уравнение прямой линии

Таким же образом, как и для прямоугольной системы координат, вводится понятие уравнения линии и в полярной системе координат. Уравнение

$F(r; \varphi) = 0$ называется *уравнением линии в полярной системе координат*, если координаты любой точки, лежащей только на этой линии, удовлетворяют этому уравнению.

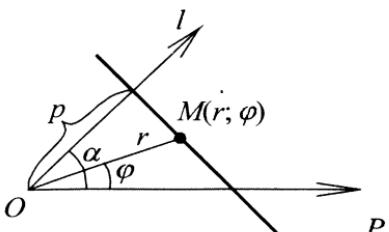


Рис. 5.11. Задание прямой линии в полярных координатах

Положение прямой линии, заданной в полярных координатах, можно определить, указав расстояние p от полюса O до прямой линии и угол α между полярной осью OP и осью l , проходящей через полюс O перпендикулярно прямой линии (см. рис. 5.11).

Для любой точки на заданной прямой линии выполняется следующее равенство:

$$\text{пр}_l \overline{OM} = p.$$

Однако $\text{пр}_l \overline{OM} = |\overline{OM}| \cdot \cos(\alpha - \varphi) = r \cdot \cos(\varphi - \alpha)$. Таким образом, уравнение прямой линии в полярных координатах имеет вид

$$r \cos(\varphi - \alpha) = p.$$

Пример 5.7. Построить сплошную линию черного цвета, заданную в полярных координатах ($\alpha = 40^\circ$, $p = 1.2$), в правом нижнем углу графического окна. Диапазон изменения полярного угла в градусах $[0; 90]$. В заголовке вывести полярное уравнение прямой линии $r \cos(\varphi - \alpha) = p$ и исходные данные.

Решение:

```
>> phi = 0:90; % диапазон изменения углов
>> alpha = 40; p = 1.2; % угол и расстояние до прямой
>> subplot(2,2,4) % установка текущей области
>> % вычисление вектора расстояний точек прямой до полюса
>> rho = p./cos(phi*pi/180-alpha*pi/180);
>> % построение графика в полярных координатах
>> polar(phi*pi/180,rho,'k')
>> % вывод заголовка (команда записывается в одной строке)
>> title('rho cos(phi-alpha)=p, \alpha=45^{\circ}, p=1.2, \phi=0..90^{\circ}' )
```

Результаты выполнения команд приведены в графическом окне на рис. 5.10 в правом нижнем углу. Координатная плоскость изображена в виде круга, в центре которого располагается полюс. Концентрические окружности внутри круга используются для обозначения меры удаленности от полюса. На рис. 5.10 изображены две окружности, которые соответствуют значениям 1 и 2. По внешней окружности располагаются значения углов в градусах. В отличие от функции `plot()`, которая является встроенной функцией

(built-in function) ядра системы MATLAB, функция `polar()` размещается в одноименном m-файле в директории ...\\MATLABбp5\\toolbox\\matlab\\graph2d и, таким образом, доступна для просмотра и внесения изменений. При работе с данной функцией необходимо помнить, что первый аргумент является массивом угловых значений, задаваемых в радианах.

5.1.3.8. Параметрический способ задания линии на плоскости

Линию на плоскости можно задать с помощью двух уравнений:

$$x = x(t), \quad y = y(t),$$

где x и y – координаты произвольной точки $M(x;y)$, лежащей на данной линии, а t – переменная, называемая *параметром*. Следовательно, параметр t определяет положение точки $(x;y)$ на плоскости.

В том случае, если параметр t изменяется, точка перемещается по плоскости, вычерчивая на ней линию. Описанный способ задания линии называется *параметрическим*, а приведенные выше уравнения – *параметрическими уравнениями линии*.

Пример 5.8. Построить в отдельной координатной плоскости одного графического окна три циклоиды (укороченную, обычную и удлиненную), заданные параметрические уравнения

$$x = r \left(t - \frac{a}{r} \sin t \right), \quad y = r \left(1 - \frac{a}{r} \cos t \right).$$

Циклоидой называется кривая, которую описывает некоторая точка P , расположенная на расстоянии a от центра круга радиуса r , катящегося без скольжения по прямой линии. Если точка P лежит на окружности круга ($r = a$), имеет место *обычная циклоида*, если она лежит внутри круга ($r > a$) – *уточченная циклоида*, если точка вне круга ($r < a$) – *удлиненная циклоида*. Два последних варианта называют также *трохоидами*. Циклоида относится к классу *трансцендентных линий*. Радиус круга $r = 1$. Расстояние от точки до центра: 0.5, 1 и 1.5 для первой, второй и третьей циклоид соответственно. Параметр t изменяется в диапазоне $[0;4\pi]$ с шагом $\pi/16$. Координатные плоскости разместить по горизонтали. Задать обозначения осей, координатную сетку и заголовок для каждой координатной плоскости. В заголовке вывести значение радиуса круга, расстояние от точки до центра и диапазон изменения параметра t . Диапазон изменения осей x и y всех координатных плоскостей $[0.4;4\pi r + 0.4]$ и $[-0.55;2.55]$ соответственно. Толщина линий всех циклоид равна 2 пунктам. При построении первой циклоиды в точке, где $t = \pi$, вывести маркер в виде кру-

га. Размер маркера равен 8 пунктам, толщина линии маркера равна 2 пунктам, а ее цвет – красный. Цвет поверхности маркера – зеленый. Во время построения второй циклоиды в точке, которая соответствует $t = 2\pi$, вывести маркер в виде треугольника, вершина которого направлена вниз. Размер маркера и толщина линии совпадают с соответствующими свойствами предыдущего маркера. Цвет линии – зеленый, а цвет поверхности – черный. На третьей координатной плоскости вывести маркер в виде круга, который соответствует $t = 3\pi$. Размер маркера и толщина линии совпадают с соответствующими свойствами предыдущих маркеров. Цвет линии – красный, цвет поверхности – зеленый. Напротив всех трех маркеров вывести соответствующие обозначения $t = \pi$, $t = 2\pi$ и $t = 3\pi$.

Решение:

```
>> t = 0:pi/16:4*pi; % диапазон изменения параметра
>> r = 1; % радиус круга
>> a = [0.5; 1; 1.5]; % расстояния от точки до центра
>> x = r*([t;t;t]-a./r*sin(t)); % массив 3x65 абсцисс
>> y = r*(1-a./r*cos(t)); % массив 3x65 ординат
>> figure % создание графического окна
>> subplot(1,3,1) % первый график – укороченная циклоида
>> plot(x(1,:),y(1,:), 'LineWidth', 2)
>> grid on, hold on, axis([-0.4 4*pi*r+0.4 -0.55 2.55])
>> xlabel('\itx'), ylabel('ity')
>> title('\itr\rm = 1, \ita\rm = 0.5, \itt\rm = [0..4\pi]')
>> % определение свойств маркера
>> plot(x(1,17), y(1,17), 'LineWidth', 2, ...
'Marker', 'o', 'MarkerSize', 8, ... % вид и размер
'MarkerEdgeColor', 'r', 'MarkerFaceColor', [0 1 0]) % цвет
>> text(1.9,1.6,'itt = \pi') % обозначение маркера
>> subplot(1,3,2) % второй график – циклоида
>> plot(x(2,:),y(2,:), 'LineWidth', 2)
>> grid on, hold on, axis([-0.4 4*pi*r+0.4 -0.55 2.55])
>> xlabel('\itx'), ylabel('ity')
>> title('\itr\rm = 1, \ita\rm = 1, \itt\rm = [0..4\pi]')
>> plot(x(2,33), y(2,33), 'LineWidth', 2, ...
'Marker', 'v', 'MarkerSize', 8, ...
'MarkerEdgeColor', [0 1 0], 'MarkerFaceColor', 'k')
>> text(5,-0.2,'itt = 2\pi')
>> subplot(1,3,3) % третий график – удлиненная циклоида
>> plot(x(3,:),y(3,:), 'LineWidth', 2)
>> grid on, hold on, axis([-0.4 4*pi*r+0.4 -0.55 2.55])
>> xlabel('\itx'), ylabel('ity')
>> title('\itr\rm = 1, \ita\rm = 1.5, \itt\rm = [0..4\pi]')
>> plot(x(3,49), y(3,49), 'LineWidth', 2, ...
'Marker', 'o', 'MarkerSize', 8, ...
'MarkerEdgeColor', [1 0 0], 'MarkerFaceColor', 'g')
>> text(5.1,2.45,'itt = 3\pi')
```

Результаты выполнения вышеприведенных команд показаны на рис. 5.12. Для вывода маркеров была использована функция `plot()`, в которую наряду с координатами маркера были переданы значения соответствующих свойств. Тип маркера определяется свойством `Marker`, размер маркера – свойством `MarkerSize`, цвет линии маркера – свойством `MarkerEdgeColor` и цвет поверхности маркера – `MarkerFaceColor`. Свойства `MarkerEdgeColor` и `MarkerFaceColor` могут принимать либо вектор 1×3 , в котором указывается интенсивность трех основных цветов, либо символ, заключенный в апострофы, который обозначает один из предопределенных цветов.

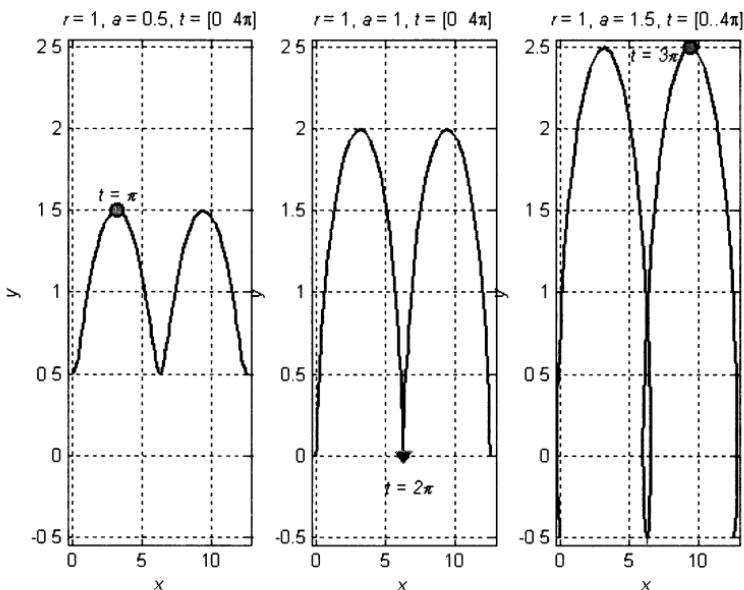


Рис. 5.12. Циклоиды

5.1.4. Линии второго порядка

Кривые второго порядка определяются при помощи уравнения второй степени относительно текущих координат:

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0.$$

В этом уравнении коэффициенты могут принимать любые действительные значения при условии, что коэффициенты A , B и C одновременно не равны нулю.

5.1.4.1. Окружность

Простейшей кривой второго порядка является окружность. *Окружностью радиуса R с центром в точке $M_0(x_0; y_0)$* называется множество точек $M(x; y)$ плоскости, удовлетворяющих условию $MM_0 = R$.

Используя формулу определения расстояния между двумя точками, получаем уравнение для окружности:

$$\sqrt{(x - x_0)^2 + (y - y_0)^2} = R \Rightarrow (x - x_0)^2 + (y - y_0)^2 = R^2.$$

Полученное уравнение окружности с центром в точке $M_0(x_0; y_0)$ и радиусом R называется *каноническим уравнением окружности*.

При использовании системы MATLAB для построения линий второго порядка на прямоугольной координатной плоскости их следует задавать с помощью соответствующих параметрических уравнений. Параметрические уравнения, описывающие окружность, имеют вид

$$x = x_0 + R \cos t, \quad y = y_0 + R \sin t.$$

Пример 5.9. Построить окружности единичного радиуса с центрами в точках $M_1(-2.1; 0)$, $M_2(-1.05; -1.5)$, $M_3(0; 0)$, $M_4(1.05; -1.5)$ и $M_5(2.1; 0)$.

Первую окружность вывести голубым цветом, вторую – желтым, третью – черным, четвертую – зеленым и пятую – красным. В заголовке графического окна написать *Olympic sign*. Убрать меню и задать графическое окно белого цвета. Установить одинаковый масштаб координатных осей и сделать невидимой координатную плоскость. В заголовке координатной плоскости написать «**MOSCOW 1980**» – место проведения XXII Олимпийских игр.

Решение:

```
>> x0=[-2.1; -1.05; 0; 1.05; 2.1]; y0=[0; -1.5; 0; -1.5; 0];
>> t = [0:pi/64:2*pi]; r = [1; 1; 1; 1; 1];
>> x=x0*ones(1,size(t,2))+r*cos(t);
>> y=y0*ones(1,size(t,2))+r*sin(t);
>> figure('NumberTitle','off', 'Name','Olympic sign',...
'MenuBar', 'none', 'Color', [1 1 1])
>> plot(x(1,:),y(1,:),'c', x(2,:),y(2,:),'y',...
x(3,:),y(3,:),'k',x(4,:),y(4,:),'g',x(5,:),y(5,:),'r')
>> axis equal, axis off
>> title('\bfMOSCOW 1980')
```

Результаты выполнения вышеприведенных команд представлены на рис. 5.13. Функция `figure()` содержит ряд входных аргументов, которые

используются для изменения внешнего вида графического окна, создаваемого по умолчанию. Свойство `NumberTitle` влияет на отображение номера графического окна в строке заголовка. По умолчанию значение этого свойства равно '`on`', т.е. при создании нового графического окна будет осуществляться вывод его номера в строке заголовка. С целью подавления вывода номера графического окна этому свойству присваивается значение '`off`'. Свойство `Name` влияет на вывод названия графического окна в строке заголовка. По умолчанию это свойство содержит пустой массив. Свойство `MenuBar` влияет на показ главного меню и панелей инструментов в графическом окне. По умолчанию значение этого параметра равно '`figure`', что соответствует отображению главного



Рис. 5.13. Построение окружностей

меню и стандартной панели инструментов в графическом окне. Чтобы скрыть главное меню и панели инструментов, следует присвоить этому свойству значение '`none`'. Цвет графического окна определяется свойством `Color`. Функция `axis()` применялась в первом случае (`axis equal`) с целью задания одинакового масштаба по двум осям, а во втором случае (`axis off`) для скрытия координатной плоскости. Если не задать команду `>> axis equal`, то окружности будут похожи на эллипсы. Для вывода координатных осей на экран следует задать команду `>> axis on`. Команда `>> axis` без параметров используется для переключения между двумя режимами отображения координатных осей.

5.1.4.2. Эллипс

Эллипсом называется множество точек на плоскости, сумма расстояний от каждой из которых до двух заданных точек, называемых *фокусами*, есть величина постоянная, большая, чем расстояние между фокусами.

Фокусы эллипса принято обозначать буквами F_1 и F_2 , расстояние между фокусами – через $2c$, сумму расстояний от любой точки эллипса до фокусов – через $2a$.

Каноническое уравнение эллипса имеет вид

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

где a , b и c связаны между собой равенством $a^2 - b^2 = c^2$ (или $b^2 - a^2 = c^2$).

В табл. 5.3 представлены два основных случая расположения эллипса относительно осей координат.

Эксцентриситетом эллипса называется отношение расстояния между фокусами к длине большой оси. Так как $2a > 2c$, то эксцентриситет всегда выражается правильной дробью, т.е. $0 \leq \varepsilon < 1$. Если величина эксцентриситета приближается к единице ($\varepsilon \approx 1$), то эллипс сильно вытянут; если значение эксцентриситета ближе к нулю ($\varepsilon \approx 0$), то эллипс имеет более округлую форму. Если эксцентриситет равен нулю, то эллипс вырождается в окружность.

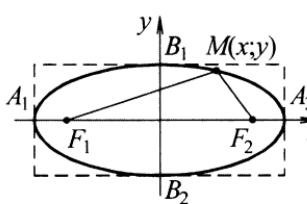
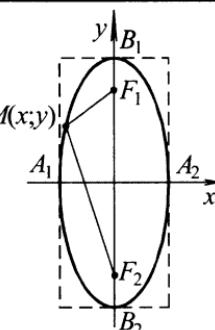
Параметрические уравнения эллипса имеют вид

$$\begin{cases} x = x_0 + a \cos t, \\ y = y_0 + b \sin t, \end{cases}$$

где x_0 и y_0 – координаты центра эллипса по осям абсцисс и ординат соответственно.

Таблица 5.3

Частные случаи расположения эллипсов относительно осей координат

		
Положение фокусов	$F_1, F_2 \in Ox$	$F_1, F_2 \in Oy$
Координаты фокусов	$F_1(-c;0), F_2(c;0)$	$F_1(0;c), F_2(0;-c)$
Соотношение между a и b	$a > b$	$b > a$
Большая ось	$ A_1A_2 = 2a$	$ B_1B_2 = 2b$
Малая ось	$ B_1B_2 = 2b$	$ A_1A_2 = 2a$
Фокусное расстояние	$ F_1F_2 = 2c$	$ F_1F_2 = 2c$
Эксцентриситет	$\varepsilon = c/a$	$\varepsilon = c/b$
Соотношение между a, b и c	$a^2 - b^2 = c^2$	$b^2 - a^2 = c^2$
Уравнение	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

Пример 5.10. Построить эллипс, полуоси которого повернуты на 60° . Большая полуось эллипса – $a = 1$, малая полуось – $b = 0.5$. Центр эллипса смещен относительно центра координатной плоскости на величину 0.1 по оси x и по оси y . Параметр t изменяется с шагом $\pi/16$. Эллипс красного цвета. Толщина линии 2 пункта. Отобразить штрих-пунктирными линиями черного цвета оси x_1 и y_1 системы координат, связанной с эллипсом. Обозначить координатные оси и вывести заголовок координатной плоскости, в котором указывается название линии полужирным наклонным шрифтом и значения большой и малой полуосей обычным шрифтом: *Ellipse* ($a = 1$, $b = 0.5$). Диапазон значений оси x $[-1;1]$, оси y $[-0.9;1.1]$. Задать одинаковый масштаб координатных осей и вывести координатную сетку. Визуализировать точку на эллипсе при $t = \pi/4$ в виде кругового маркера синего цвета размером 8 пунктов. Напротив точки вывести поясняющую текстовую надпись $M(x,y)$ и соединить точку с помощью двух сплошных прямых линий синего цвета с фокусами эллипса. Фокусы эллипса обозначить зелеными маркерами в виде звезды размером 8 пунктов. Рядом с нижним фокусом вывести надпись F_1 , а рядом с верхним – надпись F_2 .

Решение:

```

>> t=0:pi/16:2*pi; a=1; b=0.5; % Задание исходных данных
>> phi = 60*pi/180; x0 = 0.1; y0 = 0.1;
>> % Вычисление значений прямоугольных координат
>> x1=a*cos(t); y1=b*sin(t); x=x1*cos(phi)-y1*sin(phi)+x0;
>> y=x1*sin(phi)+y1*cos(phi)+y0;
>> plot(x,y,'LineWidth',2,'Color','r') %Построение эллипса
>> xlabel('\itx'), ylabel('\ity') % Оформление графика
>> title('|\it\bfEllipse\rm (\ita\rm=1, \itb\rm=0.5)')
>> axis equal, axis([-1 1 -0.9 1.1]), hold on, grid on
>> % построение и обозначение преобразованных координатных осей
>> x=-1:0.1:1; y=tan(phi)*(x-x0)+y0; plot(x,y,'-.k') % ось x
>> y=tan(phi+pi/2)*(x-x0)+y0; plot(x,y,'-.k') % ось y
>> text(0.7, 1.05, 'x'), text(-0.85, 0.75, 'y')
>> % Визуализация точки на эллипсе
>> xp = a*cos(pi/4)*cos(phi)-b*sin(pi/4)*sin(phi)+x0;
>> yp = a*cos(pi/4)*sin(phi)+b*sin(pi/4)*cos(phi)+y0;
>> plot(xp,yp,'Marker','o', 'MarkerSize', 8)
>> text(0.2,0.85, '\itM(x,y)') % Обозначение точки
>> % Вывод фокусов эллипса
>> x = [-sqrt(a^2-b^2) sqrt(a^2-b^2)]*cos(phi) + x0;
>> y = [-sqrt(a^2-b^2) sqrt(a^2-b^2)]*sin(phi) + y0;
>> plot(x(1),y(1),'g*','MarkerSize',8)
>> plot(x(2),y(2),'g*','MarkerSize',8)
>> text(-0.25,-0.7, '\itF\rm_{1}'), text(0.6,0.8, '\itF\rm_{2}')
>> % Построение двух линий, соединяющих точку и фокусы
>> line([x(1) xp x(2)], [y(1) yp y(2)])

```

Результаты вышеприведенных команд приведены на рис. 5.14. Местоположение фокусов было вычислено на основе соотношения между a , b и c , которые приведены в табл. 5.3. При построении эллипса, его фокусов, точки на эллипсе, а также координатных осей использовались формулы параллельного переноса осей координат и их поворота на заданный угол относительно нового центра.

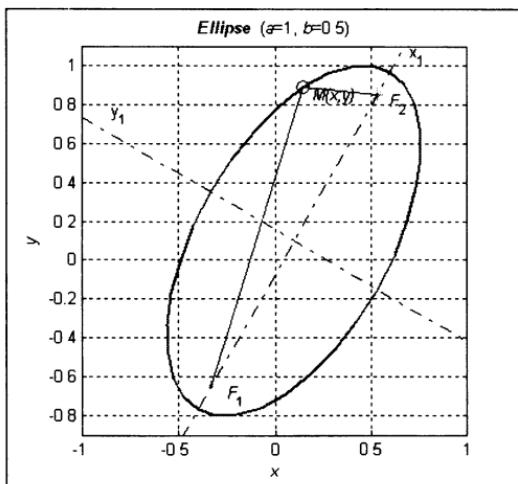


Рис. 5.14. Эллипс

5.1.4.3. Гипербола

Гиперболой называется множество точек на плоскости, модуль разности расстояний от каждой из которых до двух заданных точек, называемых *фокусами*, есть величина постоянная. Эта постоянная величина положительна и меньше расстояния между фокусами.

Фокусы гиперболы принято обозначать буквами F_1 и F_2 , расстояние между ними – через $2c$, а модуль разности между расстояниями от любой точки гиперболы до ее фокусов – через $2a$ ($2a < 2c$).

Каноническое уравнение гиперболы имеет вид

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \text{ или } \frac{y^2}{b^2} - \frac{x^2}{a^2} = 1$$

где a , b и c связаны между собой равенством $a^2 + b^2 = c^2$.

В табл. 5.4 представлены два основных случая расположения гиперболы относительно осей координат.

Таблица 5.4

**Частные случаи расположения гиперболы
относительно осей координат**

Положение фокусов Координаты фокусов Действительная ось Мнимая ось Фокусное расстояние Эксцентриситет Соотношение между a , b и c	$F_1; F_2 \in Ox$ $F_1(-c; 0), F_2(c; 0)$ $ A_1 A_2 = 2a$ $ B_1 B_2 = 2b$ $ F_1 F_2 = 2c$ $\varepsilon = c/a$ $c^2 - a^2 = b^2$	$F_1; F_2 \in Oy$ $F_1(0; c), F_2(0; -c)$ $ B_1 B_2 = 2b$ $ A_1 A_2 = 2a$ $ F_1 F_2 = 2c$ $\varepsilon = c/b$ $c^2 - b^2 = a^2$
Уравнение	$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$	$\frac{y^2}{b^2} - \frac{x^2}{a^2} = 1$

Прямоугольник со сторонами $2a$ и $2b$ называется *основным прямоугольником гиперболы*.

Эксцентриситетом гиперболы называется отношение расстояния между фокусами к длине действительной оси:

$$\varepsilon = \frac{2c}{2a} = \frac{c}{a} \quad \text{или} \quad \varepsilon = \frac{2c}{2b} = \frac{c}{b}.$$

Так как по определению $2a < 2c$, то эксцентриситет всегда выражается неправильной дробью, т.е. $\varepsilon > 1$.

Прямые линии l_1 и l_2 называются *асимптотами*; их уравнения имеют вид

$$y = \pm \frac{b}{a} x.$$

Равносторонней гиперболой называется гипербола, у которой длина действительной оси равна длине мнимой оси, т.е. $2a = 2b$ или $a = b$.

Параметрические уравнения гиперболы имеют вид

$$\begin{cases} x = a \operatorname{ch} t, & y = b \operatorname{sh} t - \text{правая ветвь}, \\ x = -a \operatorname{ch} t, & y = -b \operatorname{sh} t - \text{левая ветвь}, \end{cases} \quad (5.2)$$

где параметр $t \in (-\infty; \infty)$.

Если за полярную ось принять ось Ox прямоугольной системы координат (ось линии второго порядка), а за полюс – правый фокус, то уравнение в полярных координатах для гиперболы имеет вид

$$\rho = \frac{P}{1 - \varepsilon \cos \varphi}, \text{ где } P = \frac{b^2}{a} \text{ и } \varepsilon = \frac{c}{a}. \quad (5.3)$$

Пример 5.11. Построить в одном графическом окне три графика гиперболы, каждый из которых размещается в отдельной координатной плоскости. Первая прямоугольная координатная плоскость выводится в левом верхнем углу, вторая – в правом верхнем углу, а полярная координатная плоскость – под первыми двумя плоскостями и центрирована по горизонтали графического окна. При построении первого графика гиперболы использовались параметрические уравнения (5.2), координаты второго графика гиперболы вычислялись по параметрическим уравнениям

$$\begin{cases} x = a \operatorname{sh} t, & y = b \operatorname{ch} t - \text{верхняя ветвь}, \\ x = -a \operatorname{sh} t, & y = -b \operatorname{ch} t - \text{нижняя ветвь}, \end{cases}$$

а при визуализации третьего графика гиперболы использовалось уравнение (5.3). Параметр t для двух первых графиков изменяется в диапазоне $[-\pi/2; \pi/2]$ с шагом $\pi/16$. Полярный угол третьей гиперболы задается в радианах и изменяется в диапазоне $[5\pi/12; 19\pi/12]$ с шагом $\pi/18$ радиан. Длины действительной и мнимой полуосей для первого и третьего графиков $a = 1$ и $b = 1$ соответственно. При построении второго графика действительная ось $b = 1$, мнимая ось $a = 0.75$. Обозначить фокусы с помощью круговых маркеров и вывести рядом с ними поясняющие надписи. Отобразить в прямоугольных координатных плоскостях асимптоты гипербол и основные прямоугольники. Показать фокусы гипербол с помощью круговых маркеров красного цвета с поясняющими надписями F_1 и F_2 . В строке заголовка графического окна написать `Hyperboles`. Вывести координатные оси и их обозначения с помощью соответствующих параметрических уравнений, а также заголовки координатных плоскостей, в которых указывается полужирным наклонным шрифтом название кривой и обычным шрифтом – значения действительной и мнимой полуосей. Скрыть меню и стандартную палитру инструментов. Задать светло-серый цвет [0.9 0.9 0.9] фона графического окна.

Решение:

```

>> % создание графического окна
>> figure('NumberTitle', 'off', 'Name', 'Hyperboles',...
'Color', [0.9 0.9 0.9], 'MenuBar', 'none')
>> % построение первого графика
>> axes('Position', [0.08 0.58 0.35 0.35])
>> t=-pi/2:pi/16:pi/2; a=1; b=1; % параметры первой гиперболы
>> x_r=a*cosh(t); y_r=b*sinh(t); x_l=-a*cosh(t); y_l=-b*sinh(t);
>> % построение ветвей гиперболы
>> plot(x_r,y_r, 'LineWidth', 2) % правая ветвь
>> hold on, grid on, plot(x_l,y_l, 'LineWidth', 2) % левая ветвь
>> % обозначение графика
>> title('\it\bfHyperbole \rm(\ita\rm=1, \itb\rm=1)')
>> % вывод и обозначение координатных осей
>> line([-3 0; 3 0], [0 -2.2; 0 2.2], 'Color', 'k')
>> xlabel('\itx\rm = \pm\ita\rm ch(\itt\rm)')
>> ylabel('\ity\rm = \pm\itb\rm sh(\itt\rm)')
>> % вывод асимптот
>> x = -2.3::1:2.3; y_1 = b/a * x; y_2 = -b/a * x;
>> plot(x,y_1, 'k', x,y_2, 'k')
>> % визуализация и обозначение фокусов
>> c = [-sqrt(a^2+b^2) sqrt(a^2+b^2); 0 0];
>> plot(c(1),c(2), 'ro-', c(3),c(4), 'ro-');
>> text(-1.55,0.3,'itF\rm_{1}'), text(1.35,0.3,'itF\rm_{2}')
>> % построение прямоугольника гиперболы
>> rectangle('Curvature',[0 0], 'Position', [-a -b 2*a 2*b])
>> % обозначение осей гиперболы
>> text(.1,1.3,'itB\rm_{1}'), text(.1,-1.4,'itB\rm_{2}')
>> text(-0.93,-0.27,'itA\rm_{1}'), text(0.53,-0.27,'itA\rm_{2}')
>> axis equal % масштабирование осей
>> % построение второго графика
>> axes('Position', [0.58 0.58 0.35 0.35])
>> t = -pi/2:pi/16:pi/2; a = 0.75; b = 1;
>> x_r=a*sinh(t); y_r=b*cosh(t); x_l=-a*sinh(t); y_l=-b*cosh(t);
>> plot(x_r,y_r, 'LineWidth', 2), hold on, grid on
>> plot(x_l,y_l, 'LineWidth', 2)
>> title('\it\bfHyperbole \rm(\ita\rm=0.75, \itb\rm=1)')
>> line([-3.5 0; 3.5 0], [0 -2.65; 0 2.65], 'Color', 'k')
>> xlabel('\itx\rm = \pm\ita\rm sh(\itt\rm)')
>> ylabel('\ity\rm = \pm\itb\rm ch(\itt\rm)')
>> x = -2::1:2; y_1 = b/a * x; y_2 = -b/a * x;
>> plot(x,y_1, 'k', x,y_2, 'k')
>> c = [0 0; -sqrt(a^2 + b^2) sqrt(a^2+b^2)];
>> plot(c(1),c(2), 'ro-', c(3),c(4), 'ro-');

```

```

>> text(0.1,1.55,'\\itF\\rm_{1}'), text(0.1,-1.65,'\\itF\\rm_{2}')
>> rectangle('Curvature', [0 0], 'Position', [-a -b 2*a 2*b])
>> text(.1,0.6,'\\itB\\rm_{1}'), text(.1,-0.7,'\\itB\\rm_{2}')
>> text(-1.25,-0.4,'\\itA\\rm_{1}'), text(0.8,-0.4,'\\itA\\rm_{2}')
>> axis equal
>> % построение третьего графика
>> axes('Position', [0.275 0.01 0.45 0.45])
>> phi=75*pi/180:pi/18:285*pi/180;
>> a = 1; b = 1; P=b.^2./a; e=sqrt(a.^2+b.^2)./a;
>> rho=P./(1-e*cos(phi)); polar(phi,rho), hold on
>> title('\\itbfHyperbole \\rm{\\ita\\rm=1, \\itb\\rm=1}')
>> ylabel('\\rho\\rm=P/(1-\\epsilon\\cos(\\phi))')
>> line([-2 0; 2 0], [0 -2; 0 2], 'Color', 'k')
>> plot(0,0,'or'), text(0.15,0.15,'\\itF'), text(-0.7,-0.2,'\\itA')

```

На рис. 5.15 представлены результаты приведенных выше команд. Функция `figure()` была использована для создания нового графического окна, вывода в строке заголовка имени **Hyperboles**, задания цвета фона графического окна, соответствующего значениям [0.9 0.9 0.9] цветовой модели RGB, и скрытия строки главного меню и стандартной палитры инструментов.

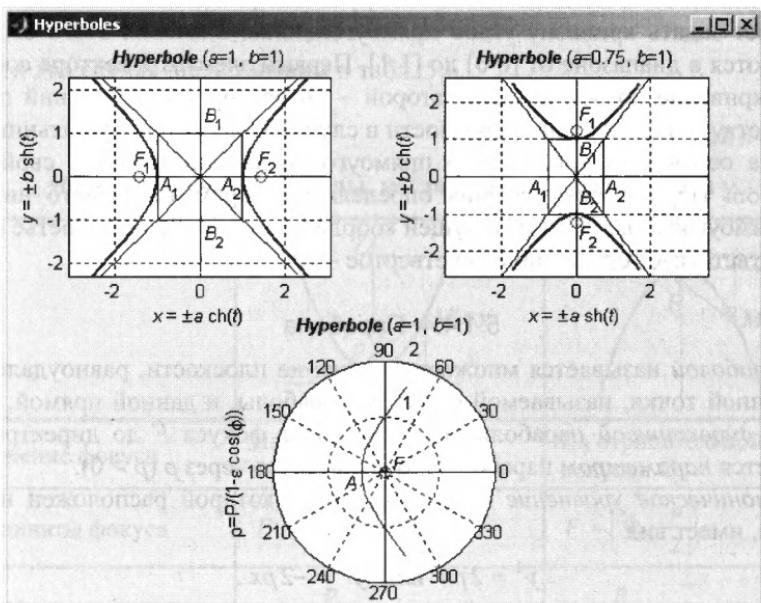


Рис. 5.15. Гиперболы

Функция `axes()` использовалась для создания новой координатной плоскости и ее размещения в произвольном месте графического окна. При задании функции `axes()` новая координатная плоскость становится текущей, куда выводятся результаты команд, задаваемых в командной строке. Значение свойства `Position` представляет собой вектор [`left bottom width height`]. Первый элемент обозначает расстояние от левого края графического окна до левого края координатной плоскости. Второй элемент соответствует расстоянию от нижнего края графического окна до нижнего края координатной плоскости. Третий элемент является длиной координатной плоскости по горизонтали, а четвертый – по вертикали. При задании размеров прямоугольных координатных плоскостей размеры обозначений осей и заголовка не учитываются. Расстояния задаются в нормализованных единицах, которые используются по умолчанию и изменяются в диапазоне от нуля до единицы. Следовательно, левый нижний угол графического окна соответствует координатам $(0;0)$, а правый верхний угол – координатам $(1;1)$. При задании полярной координатной плоскости обозначения полярного угла φ располагаются внутри прямоугольной области, в пределах которой строится полярная плоскость.

Для отображения прямоугольников гиперболы используется функция `rectangle()` со свойством `Curvature`, равным $[0\ 0]$. Свойство `Curvature` позволяет задать кривизну углов прямоугольника. Значения этого свойства изменяются в диапазоне от $[0\ 0]$ до $[1\ 1]$. Первый элемент в векторе соответствует кривизне по оси абсцисс, второй – по оси ординат. Верхний предел соответствует эллипсу или окружности в случае задания прямоугольника или квадрата соответственно. Размер прямоугольника определяется свойством `Position`. Первые два значения определяют координаты левого нижнего угла прямоугольника внутри текущей координатной плоскости, третье значение соответствует его ширине, а четвертое – его высоте.

5.1.4.4. Парабола

Параболой называется множество точек на плоскости, равноудаленных от заданной точки, называемой фокусом параболы, и данной прямой, называемой *директрисой* параболы. Расстояние от фокуса F до директрисы d называется *параметром* параболы и обозначается через p ($p > 0$).

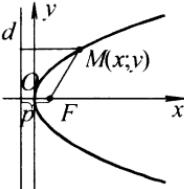
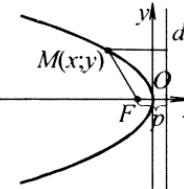
Каноническое уравнение параболы, фокус которой расположен на оси абсцисс, имеет вид

$$y^2 = 2px \text{ или } y^2 = -2px.$$

Эти два случая представлены в табл. 5.5.

Таблица 5.5

Частные случаи параболы, когда ее фокус лежит на оси x

		
Положение фокуса	На положительной полуоси Ox	На отрицательной полуоси Ox
Координаты фокуса	$F = \left(\frac{p}{2}; 0\right)$	$F = \left(-\frac{p}{2}; 0\right)$
Уравнение директрисы	$x = -\frac{p}{2}$	$x = \frac{p}{2}$
Уравнение параболы	$y^2 = 2px$	$y^2 = -2px$

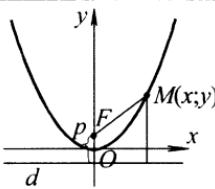
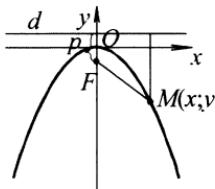
Каноническое уравнение параболы с фокусом на оси ординат имеет вид

$$x^2 = 2py \text{ или } x^2 = -2py.$$

Эти два случая представлены в табл. 5.6.

Таблица 5.6

Частные случаи параболы, когда ее фокус лежит на оси y

		
Положение фокуса	На положительной полуоси Oy	На отрицательной полуоси Oy
Координаты фокуса	$F = \left(0; \frac{p}{2}\right)$	$F = \left(0; -\frac{p}{2}\right)$
Уравнение директрисы	$y = -\frac{p}{2}$	$y = \frac{p}{2}$
Уравнение параболы	$x^2 = 2py$	$x^2 = -2py$

Уравнения соответствующих парабол, вершины которых не лежат в центре координатных плоскостей и ось симметрии которых параллельна одной из осей координатных плоскостей, имеют вид

$$(y - y_0)^2 = 2p(x - x_0) \text{ или } (y - y_0)^2 = -2p(x - x_0),$$

$$(x - x_0)^2 = 2p(y - y_0) \text{ или } (x - x_0)^2 = -2p(y - y_0).$$

Пример 5.12. Построить в одной координатной плоскости графического окна четыре параболы, вершины которых передвинуты от начала координат на единичное расстояние. Координаты фокуса F первой параболы $(1/2; 0)$, второй параболы $(0; 1)$, третьей параболы $(-3/2; 0)$ и четвертой параболы $(0; -2)$. Значения ординат первой и третьей парабол изменяется с шагом 0.1 в диапазоне $[-2; 2]$ и $[-2.5; 2.5]$ соответственно. Значения абсцисс второй и четвертой парабол отстоят друг от друга на расстоянии 0.1 внутри интервала $[-2; 2]$ и $[-3; 3]$ соответственно. Толщина линий всех парабол равна двум пунктам. Цвет первой параболы синий, второй параболы – красный, третьей параболы – зеленый и четвертой параболы – черный. Вывести в виде кругового маркера фокус каждой параболы и рядом с ним курсивом символ F . Цвет маркера и символа совпадает с цветом соответствующей параболы. Координатная плоскость, в которой строятся параболы, занимает в графическом окне область $[140 20 348 370]$, где первые два элемента вектора, заданные в пикселях, являются координатами левого нижнего угла координатной плоскости, третий элемент – ее ширина и четвертый – ее высота. Диапазон отображения абсцисс $[-3; 5]$ и ординат $[-3; 3]$. Напротив значений оси x $(-3 -2 -1 0 1 2 3 4 5)$ вывести в нижней части координатной плоскости засечки, из которых обозначить нечетные значения $(-x_{\max} -1 \text{ Center } 1 3 x_{\max})$. Напротив значений оси y $(-3 -2 -1 0 1 2 3)$ вывести в левой части координатной плоскости засечки и обозначить их соответствующими числовыми значениями. Вывести координатную сетку. В левой части графического окна отобразить уравнения всех четырех парабол соответствующими цветами. Записать по два уравнения в строке и повернуть их на 90° против часовой стрелки. Левый и нижний края графического окна отстоят от соответствующих краев экрана на 200 пикселяй. Ширина графического окна равна 500 пикселям, а его высота 400 пикселям. В строке заголовка выводится номер графического окна и слово *Parabole*. Графическое окно не содержит строки меню и стандартной панели инструментов.

Решение:

```
>> figure('Position',[200 200 500 400], 'MenuBar', 'none', ...
'Name', 'Parabole')
>> axes('Units','pixels','Position',[0 0 500 400], 'Visible','off')
```

```

>> % обозначение правой параболы
>> text('Units', 'pixels', 'Position', [40 133], 'String',...
'(\ity - y_{\rm0})^{(\rm2)} = 2\itp \rm(\itx - x_{\rm0}),', ...
'Color', [0 0 1], 'HorizontalAlignment', 'center', ...
'VerticalAlignment', 'bottom', 'Rotation', 90.0)
>> % обозначение верхней параболы
>> text('Units', 'pixels', 'Position', [40 266], 'String',...
'(\itx - x_{\rm0})^{(\rm2)} = 2\itp \rm(\ity - y_{\rm0}),', ...
'Color', [0 0.8 0], 'HorizontalAlignment', 'center', ...
'VerticalAlignment', 'bottom', 'Rotation', 90.0)
>> % обозначение левой параболы
>> text('Units', 'pixels', 'Position', [70 133], 'String',...
'(\ity - y_{\rm0})^{(\rm2)} = - 2\itp \rm(\itx - x_{\rm0}),', ...
'Color', [1 0 0], 'HorizontalAlignment', 'center', ...
'VerticalAlignment', 'bottom', 'Rotation', 90.0)
>> % обозначение верхней параболы
>> text('Units', 'pixels', 'Position', [70 266], 'String',...
'(\itx - x_{\rm0})^{(\rm2)} = - 2\itp \rm(\ity - y_{\rm0}),', ...
'Color', [0 0 0], 'HorizontalAlignment', 'center', ...
'VerticalAlignment', 'bottom', 'Rotation', 90.0)
>> % Вывод координатной плоскости и задание ее свойств
>> axes('Units', 'pixels', 'Position', [140,20,348,370],...
'XLimMode', 'manual', 'XLim', [-3 5], ...
'YLimMode', 'manual', 'YLim', [-3 3], ...
'XTickMode', 'manual', 'XTick', [-3 -2 -1 0 1 2 3 4 5], ...
'YTickMode', 'manual', 'YTick', [-3 -2 -1 0 1 2 3], ...
'XTickLabelMode', 'manual', 'XTickLabelMode', 'manual', ...
'XTickLabel', '-x_max||-1|Center|1||3||x_max', ...
'YTickLabel', '-y_max|-2|-1|Center|1|2|y_max', ...
'XGrid', 'on', 'YGrid', 'on', 'NextPlot', 'add')
>> % задание общих переменных
>> p=[1 2 3 4]; x0 = [1;0;-1;0]; y0 = [0;1;0;-1];
>> % построение правой и левой парабол, а также их фокусов
>> y = -2:0.1:2; x = (y-y0(1)).^2./p(1) + x0(1);
>> plot(x,y,'LineWidth',2, 'Color', [0 0 1])
>> x=x0(1)+p(1)/2; plot(x,y0(1), '-ob')
>> text(1.5, 0.2, '\itF', 'Color', [0 0 1])
>> y = -2.5:0.1:2.5; x = -(y-y0(3)).^2./p(3) + x0(3);
>> plot(x,y,'LineWidth',2, 'Color', [1 0 0])
>> x=x0(3)-p(3)/2; plot(x,y0(3), '-or')
>> text(0, 2.2, '\itF', 'Color', [0 0.8 0])
>> % построение верхней и нижней парабол
>> x=-2:0.1:2; y = (x-x0(2)).^2./p(2) + y0(2);
>> plot(x,y,'LineWidth',2, 'Color', [0 0.8 0])
>> y=y0(2)+p(2)/2; plot(x0(2),y, '-og')
>> text(-2.5, 0.2, '\itF', 'Color', [1 0 0])
>> x=-3:0.1:3; y = -(x-x0(4)).^2./p(4) + y0(4);
>> plot(x,y,'LineWidth',2, 'Color', [0 0 0])
>> y=y0(4)-p(4)/2; plot(x0(4),y, '-ok')
>> text(0, -2.8, '\itF', 'Color', [0 0 0])

```

Результаты выполнения вышеприведенных команд представлены на рис. 5.16. При создании графического окна с помощью свойства `Position` можно указать область, которую оно будет занимать на экране. Значение этого свойства представляет собой вектор 1×4 , значения которого по умолчанию для графического окна задаются в пикселях. Первое значение в векторе соответствует расстоянию от левой стороны экрана до левой стороны графического окна, второе – от нижней части экрана до нижней части графического окна, третье значение определяет ширину графического окна и четвертое –

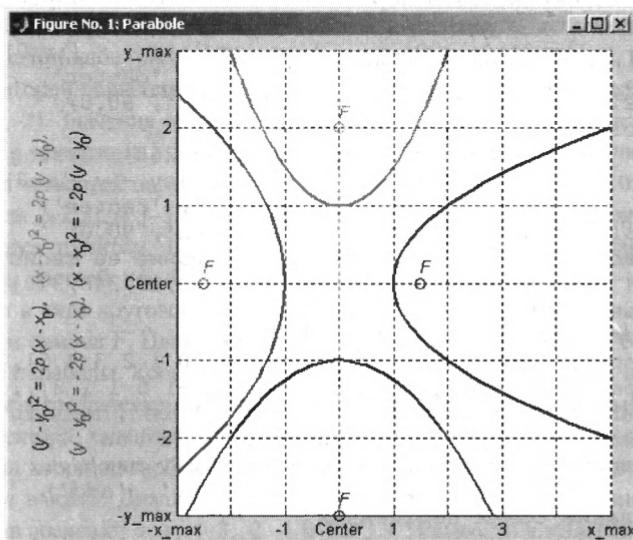


Рис. 5.16. Параболы

его высоту без учета строки заголовка. То есть свойство `Position` определяет размер клиентской области графического окна. Для изменения единицы измерения необходимо изменить значение свойства `Units`. Определение единицы измерения с помощью свойства `Units` должно идти перед указанием позиции графического объекта посредством свойства `Position`. Свойство `Units` может принимать следующие заранее определенные значения: `inches` (дюймы), `centimeters` (сантиметры), `normalized` (нормализованные единицы от 0 до 1) – используются по умолчанию для координатных плоскостей; `points` (пункты), `pixels` (пиксели) – используются по умолчанию для графических окон и `characters` (символы).

Для вывода текста в произвольном месте графического окна необходимо предварительно создать координатную плоскость, которая занимает всю клиентскую область этого графического окна, и сделать ее невидимой.

С помощью свойства `Position` определяется точка в текущей координатной плоскости, относительно которой будет выводиться текст, расположенный в свойстве `String`. *Выравнивание текста* относительно заданной точки определяется при помощи двух свойств: `HorizontalAlignment` и `VerticalAlignment`. Первое свойство задает *горизонтальное выравнивание*, а второе – *вертикальное*. Все возможные варианты значений этих свойств представлены на рис. 5.17. В приведенных командах текст выравнивался относительно заданной точки по центру и по нижней границе. Для изменения *ориентации текста* необходимо использовать свойство `Rotation`, которое позволяет задавать угол поворота текста в градусах относительно горизонтали против часовой стрелки. В приведенных примерах изменения ориентации текста угол поворота равнялся 90° , что соответствует расположению текста по вертикали.

Горизонтальное выравнивание `HorizontalAlignment`

по левой границе	по центру	по правой границе
<code>left</code>	<code>center</code>	<code>right</code>
		

Вертикальное выравнивание `VerticalAlignment`

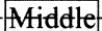
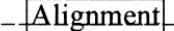
по верхней границе	по прописным буквам	по середине текста
<code>top</code>	<code>cap</code>	<code>middle</code>
		
по базовой линии	по нижней границе	
<code>baseline</code>	<code>bottom</code>	
		

Рис. 5.17. Виды выравнивания текста

Система MATLAB предоставляет большие возможности по оформлению координатной плоскости. С помощью свойств `XLim` и `YLim` можно задавать *фиксированный диапазон значений оси* абсцисс и оси ординат соответственно. По умолчанию диапазон значений обеих осей равняется $[0;1]$ и не является фиксированным, т.е. изменяется до полного включения графика в координатную плоскость. Для перехода из *автоматического режима определения диапазона осей* в *ручной режим* необходимо изменить значение свойств `XLimMode` и `YLimMode` с `auto`, которое задано по умолчанию, на `manual`.

Свойства `XTick` и `YTick` используются для вывода засечек в указанных значениях на соответствующих координатных осях. Как и в случае с предыдущими свойствами, для перехода в *ручной режим вывода засечек* следует изменить значения свойств `XtickMode` и `YtickMode` с `auto` на `manual`. С помощью свойств `XTickLabel` и `YTickLabel` выводятся значения соответствующих осей напротив определенных ранее засечек. Значения перечисляются через вертикальную черту. Если необходимо пропустить какое-либо значение, то следует задать две вертикальные черты подряд. Помимо числовых обозначений можно задавать текстовые обозначения, которые в некоторых случаях более информативны. Для перехода в ручной режим задания обозначений осей необходимо присвоить свойствам `XTickLabelMode` и `YTickLabelMode` значения `manual`. Для отображения координатной сетки используются свойства `XGrid` и `YGrid`, которые позволяют скрывать или выводить прямые линии, перпендикулярные соответствующей оси и пересекающие ее в определенных ранее засечках. Присваивание этим свойствам значений `on` соответствует заданию команды `grid on`. Свойство `NextPlot`, которое может принимать значения `replace` или `add`, используется для изменения режима построения графиков. Значение `replace`, используемое по умолчанию, соответствует тому, что перед построением следующего графика предыдущий график будет удаляться из координатной плоскости, то есть происходит замена графика. Значение `add` соответствует режиму добавления следующего графика к уже располагающимся графикам на координатной плоскости. Присваивание свойству `NextPlot` значения `add` соответствует команде `hold on`.

5.1.5. Кривые высших порядков

Еще математиками древности изучались линии второго порядка и рассматривались некоторые линии высших порядков. Однако только после изобретения Рене Декартом аналитической геометрии стало возможным систематическое изучение кривых высших порядков и их классификация.

Рассмотрим кривую линию, описываемую точкой подвижной окружности, которая касается изнутри неподвижной окружности вчетверо большего радиуса и катится по ней без скольжения. Эта кривая называется *астроидой* (от греческих слов *astron* – звезда и *eidos* – вид).

Уравнение астроиды в прямоугольных координатах имеет вид

$$x^{\frac{2}{3}} + y^{\frac{2}{3}} = a^{\frac{2}{3}},$$

где a – радиус неподвижной окружности.

Параметрические уравнения астроиды: $\begin{cases} x = a \cos^3 t, \\ y = a \sin^3 t. \end{cases}$

Решение:

```

>> %задание данных
>> t=0:pi/30:2*pi; a=1;
>> x=a*cos(t).^3; y=a*sin(t).^3;
>> % разрешение экрана
>> scrn = get(0,'ScreenSize');
>> % задание графического окна
>> width = 420; height = 380;
>> figure('Color', [1 1 1],...
'MenuBar', 'none',...
'Name','Астроида',...
'NumberTitle','off',...
'Position',...
[fix((scrn(3)-width)/2),...
fix((scrn(4)-height)/2),...
width, height])
>> % задание свойств плоскости
axes('Box','on','Color','none',...
'DataAspectRatioMode','manual',...
'DataAspectRatio',[1 1 1],...
'FontAngle','italic',...
'FontName','Times',...
'FontSize',12,...
'FontWeight','bold',...
'NextPlot','add',...
'XColor',[0.5 0 0],...
'XGrid','on','XDir','reverse',...
'XLimMode','manual',...
'XLim',[-1 1],...
'XTickMode','manual',...
'XTick',[-1,-.75,-.5,-.25,0,....
.25,.5,.75,1],...
'XTickLabelMode','manual',...
'XTickLabel',...
'-1||-0.5||0||0.5||1',...
'YAxisLocation','right',...
'YColor',[0.5 0 0],...
'YGrid','on','YDir','normal',...
'YLimMode','manual',...
'YLim',[-1 1],...
'YTickMode','manual',...
'YTick',[-1,-0.75,-.5,-.25,0,....
0.25,0.5,0.75,1],...
'YTickLabelMode','manual',...
'YTickLabel',...
'-1||-0.5||0||0.5||1',...
'YData',x,...
```

'TickDirMode','manual',...
'YData',y,...

```

'FontAngle','italic',...
'FontName','Times',...
'FontSize',12,...
'FontWeight','bold',...
'HorizontalAlignment','center',...
'Interpreter','tex',...
'Position', [0 -1.18],...
'String',...
'x=a\cdotcos^3(t),t=[0..2\pi]',...
'VerticalAlignment','middle');
>> % обозначение оси у
>> text('Color', [0.5,0,0],...
'FontAngle','italic',...
'FontName','Times',...
'FontSize',12,...
'FontWeight','bold',...
'HorizontalAlignment','center',...
'Interpreter','tex',...
'Position', [-1.2 0],...
'Rotation',90,...
```

'String',...
'y=a\cdotsin^3(t),t=[0..2\pi]',...
'VerticalAlignment','middle')

```

>> % построение неподв. окружности
>> rectangle('Curvature',[1,1],...
'LineWidth', 1.5,...
```

'Position', [-1 -1 2 2])

```

>> % построение подв. окружности
>> rectangle('Curvature',[1 1],...
'EdgeColor', [1 0 0],...
'FaceColor', [0.8 0.8 0.8],...
'LineWidth', 1.5,...
```

'Position', [-0.78,0.28,0.5,0.5])

```

>> % построение графика
>> line('Color',[0 0 1],...
'LineWidth',2.5,...
```

'LineStyle', '-.',...

```

'Marker', 'o',...
'MarkerEdgeColor', [0 0 0],...
'MarkerFaceColor', [0 1 0],...
'MarkerSize', 6,...
```

На рис. 5.18 представлены результаты вышеприведенных команд. Для размещения графического окна в центре экрана необходимо определить его текущее разрешение. Разрешение экрана монитора определяется с помощью функции `get()`, где в качестве первого входного аргумента располагается дескриптор со значением «ноль», а следующим входным аргументом идет свойство `Position`. Нулевой дескриптор является *корневым объектом* (`Root`) и обозначает экран, который является родительским объектом для графического окна. В результате функция возвращает значение указанного

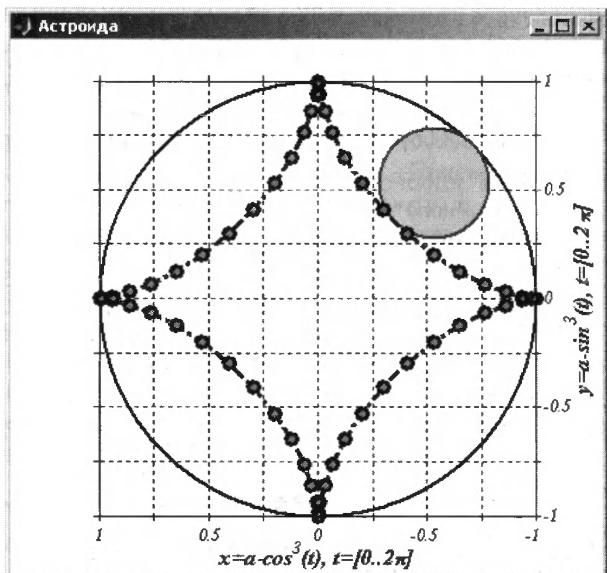


Рис. 5.18. Астроида

свойства корневого объекта. Результат представляет собой вектор 1×4 , где первые два элемента обозначают координаты левого нижнего угла экрана $(1;1)$, третий элемент – число пикселей по горизонтали (разрешение экрана по ширине), а четвертый элемент – число пикселей по вертикали (разрешение экрана по высоте). Графическое окно в свою очередь является родительским объектом для координатной плоскости. Следовательно, перед построением координатной плоскости следует создать графическое окно, и если необходимо, то задать значения свойств графического окна, которые отличаются от принятых по умолчанию. В приведенных командах был задан белый цвет графического окна; скрыты нумерация графических окон, выводимая в строке заголовка, строка главного меню и стандартная панель инструментов; задано имя графического окна, выводимое в строке заголовка; ука-

зана позиция графического окна на экране в пикселях, соответствующая выводу этого окна в центре экрана.

После задания свойств графического окна с помощью функции `axes()` была создана прямоугольная координатная плоскость и заданы ее свойства. Свойство `Box`, которое может иметь два значения: `on` или `off` (значение по умолчанию), используется для вывода прямоугольника, охватывающего координатную плоскость. Данная особенность может быть использована для улучшения внешнего вида. Свойство `DataAspectRatioMode` позволяет переключаться между автоматическим (`auto`) режимом, который принят по умолчанию, и ручным (`manual`) режимом задания масштаба координатных осей. Свойство `DataAspectRatio` позволяет установить масштаб по трем координатным осям. Первой осью считается ось `x`, второй – ось `y`, третьей – ось `z`. Вектор `[1, 1, 1]` соответствует однаковому масштабу по всем трем осям. Такого же результата можно добиться при задании команды `>> axis equal`. Следующая группа свойств определяет имя (`FontName`), размер (`FontSize`) и начертание (`FontAngle` и `FontWeight`) шрифта, используемого для вывода значений координатных осей. Изменить значение этого свойства можно также с помощью команд `>> hold on` или `>> hold off`. Для изменения направления оси абсцисс свойству `XDir` было присвоено значение `reverse`. Свойство `YAxisLocation`, которое может принимать значения `left` (по умолчанию) или `right`, определяет местоположение оси ординат относительно вертикальной плоскости. В приведенном примере ось ординат была выведена с правой стороны относительно координатной плоскости. Свойства `XColor` и `YColor` определяют цвет координатных осей, их значений и координатной сетки, которая связана с соответствующими осями.

После задания свойств координатной плоскости следует создание обозначений соответствующих координатных осей и задание их свойств. Обозначения координатных осей являются текстовыми объектами, для которых родительским объектом служит координатная плоскость. Следовательно, значения свойства `Position` задавались исходя из значений соответствующих координатных осей. В отличие от остальных объектов, для которых координатная плоскость является родительским объектом, *текстовый объект* (`text`) может размещаться вне границ координатной плоскости. Свойство `Interpreter` определяет, используется или нет интерпретатор команд системы верстки `TeX` для вывода текста, расположенного в свойстве `String`.

При построении подвижной и неподвижной окружностей внутри координатной плоскости использовался объект `rectangle` со значением свойства `Curvature`, равным `[1 1]`. Объект `rectangle` является дочерним объектом координатной плоскости. Свойства `EdgeColor` и `FaceColor` позволяют задать цвет контура круга и цвет его поверхности соответственно.

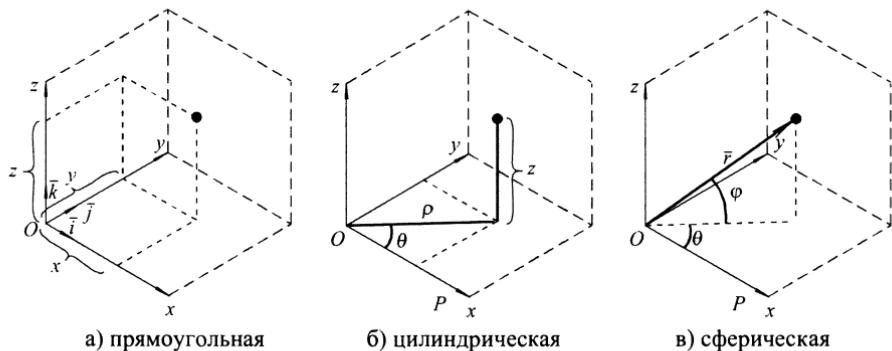
Для создания и задания свойств астроиды использовалась функция `line()`.

5.2. АНАЛИТИЧЕСКАЯ ГЕОМЕТРИЯ В ПРОСТРАНСТВЕ

5.2.1. Системы координат в пространстве

Под *системой координат* в пространстве понимают способ, позволяющий численно описать положение точки в пространстве. Наибольшее распространение в практике получили три системы координат: *прямоугольная, цилиндрическая и сферическая*.

Прямоугольная система координат, представленная на рис. 5.19, а, задается тремя взаимно перпендикулярными прямыми – осями, на каждой из которых выбрано положительное направление и задан единичный (масштабный) отрезок. Систему координат обозначают $Oxyz$ (или $Oijk$, где i, j, k – единичные орты соответствующих осей). Точку пересечения осей называют *началом координат*. Координатами произвольной точки в системе координат $Oxyz$ называются три числа – x, y и z , которые однозначно определяют точку в пространстве. Координаты точки M записывают следующим образом: $M(x; y; z)$. При этом x называется *абсциссой* точки, y – *ординатой*, а z – *аппликатой*.



Р и с . 5.19. Системы координат в пространстве

Цилиндрическая система координат, изображенная на рис. 5.19, б, является расширением полярной системы координат в пространство. Положение точки в цилиндрической системе координат определяется тремя координатами – θ, ρ и z . Первой координатой является угол θ , отсчет которого ведется от оси Op против часовой стрелки, если смотреть со стороны положительного направления оси z . Вторая координата (ρ) обозначает удаленность точки от начала координат в полярной плоскости. Третьей координатой является число z , определяющее высоту расположения точки над полярной плоскостью.

При преобразовании цилиндрической системы координат в прямоугольную и обратно необходимо совместить полярную ось Op с осью Ox прямо-

угольной системы координат, а также оси z обеих систем координат и отсчитывать полярный угол θ в плоскости Oxy . Используя эти соглашения, формулы преобразования имеют следующий вид:

$$\begin{cases} x = \rho \cos \theta, \\ y = \rho \sin \theta, \\ z = z. \end{cases} \quad \begin{cases} \theta = \operatorname{arctg}(y/x), \\ \rho = \sqrt{x^2 + y^2}, \\ z = z. \end{cases}$$

Сферическая система координат, представленная на рис. 5.19, в, позволяет определить положение точки в пространстве с помощью трех координат – θ , φ и r . Угол θ совпадает с углом θ в цилиндрической системе координат. Угол φ обозначает угол между радиусом-вектором \bar{r} и *полярной плоскостью*. Координата r определяет длину радиуса-вектора \bar{r} от центра сферической системы координат до точки.

При преобразовании сферической системы координат в декартову необходимо совместить луч Op с осью Ox и полярную плоскость – с плоскостью Oxy соответственно, а также обеспечить совпадение радиуса-вектора \bar{r} с положительным направлением оси z при $\varphi = 90^\circ$. Применяя эти соглашения, аналитические выражения, которые связывают декартову и сферическую системы координат, имеют вид

$$\begin{cases} x = r \cos \varphi \cos \theta, \\ y = r \cos \varphi \sin \theta, \\ z = r \sin \varphi. \end{cases} \quad \begin{cases} \theta = \operatorname{arctg}\left(\frac{y}{x}\right), \\ r = \sqrt{x^2 + y^2 + z^2}, \\ \varphi = \operatorname{arctg}\left(\frac{z}{\sqrt{x^2 + y^2}}\right). \end{cases}$$

Система MATLAB предлагает ряд функций, которые могут использоватьсь для преобразования координат между рассмотренными выше системами. Их список представлен в табл. 5.7.

Таблица 5.7

Список функций для преобразования координат

Функция	Описание
<code>[x, y, z]=pol2cart(theta, rho, z)</code>	Цилиндрическая \rightarrow прямоугольная
<code>[theta, rho, z]=cart2pol(x, y, z)</code>	Прямоугольная \rightarrow цилиндрическая
<code>[x, y, z]=sph2cart(theta, phi, r)</code>	Сферическая \rightarrow прямоугольная
<code>[theta, phi, r]=cart2sph(x, y, z)</code>	Прямоугольная \rightarrow сферическая

В случае использования стандартных функций для преобразования между системами координат углы следует задавать в радианах. Если число входных аргументов в функцию `pol2cart()` равно двум, то выполняется преобразование значений из полярной системы координат в прямоугольную.

5.2.2. Трансформация системы координат в пространстве

Под трансформацией системы координат понимается перемещение точки начала координат в другую позицию, а также вращение системы координат относительно координатной оси. Все случаи трансформации будут рассматриваться относительно прямоугольной системы координат.

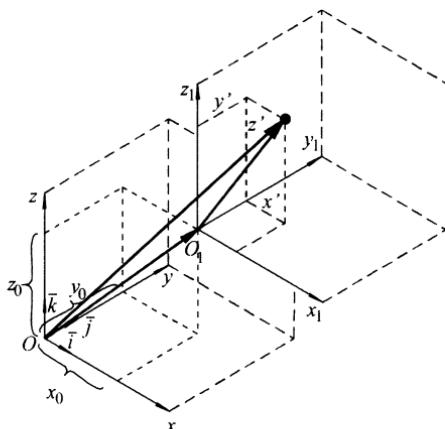
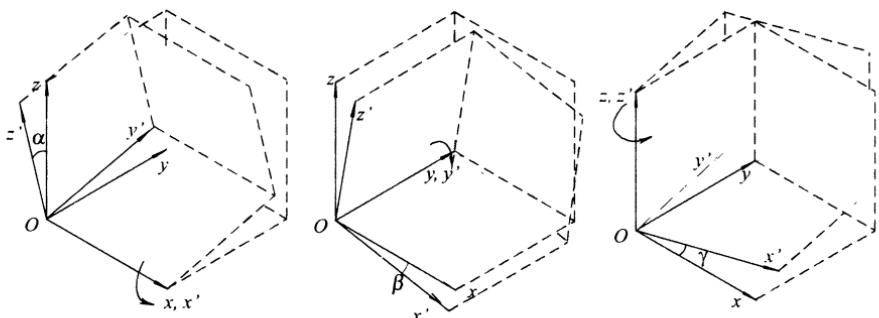


Рис. 5.20. Параллельный перенос осей координат

При параллельном переносе координатных осей происходит переход от системы координат $Oxyz$ к новой системе координат – $O_1x_1y_1z_1$. При этом происходит смена положения начала координат, но направление осей и их масштаб остаются неизменными (рис. 5.20). Формулы, которые применяются при параллельном переносе, выглядят следующим образом:

$$\begin{aligned}x &= x_0 + x', \quad y = y_0 + y', \\z &= z_0 + z'.\end{aligned}$$

Под *поворотом осей координат* называют такое преобразование координат, при котором две оси поворачиваются на один и тот же угол относительно третьей, а начало координат и масштаб остаются неизменными (рис. 5.21).



а) относительно оси x

б) относительно оси y

в) относительно оси z

Рис. 5.21. Поворот координатных осей

Для вращения координатных осей применяются следующие формулы, записанные в матричном виде:

$$\begin{aligned} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = & \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}, \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}, \\ \begin{bmatrix} x \\ y \\ z \end{bmatrix} = & \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}. \end{aligned}$$

Общее аналитическое выражение, описывающее поворот координатной системы вокруг всех трех осей координат, а также параллельный перенос осей, записывается следующим образом:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix},$$

где $R_x(\alpha)$, $R_y(\beta)$ и $R_z(\gamma)$ – матрицы поворота относительно соответствующих осей.

Приведенные выше матричные выражения могут использоваться также для вращения и перемещения объектов в текущей системе координат.

Пример 5.14. В одном графическом окне создать четыре координатных пространства. В первом октанте первого координатного пространства построить пирамиду с треугольным основанием. Во втором координатном пространстве изобразить пирамиду после ее перемещения на расстояние (0.25;0.25;0.25). В третьем координатном пространстве построить пирамиду после ее поворота на 45° относительно оси x . В четвертом координатном пространстве изобразить пирамиду, которая получена после ее перемещения на расстояние (0.25;0.25;0.25) и поворота на 45° относительно всех трех осей. Вывести обозначения всех осей и заголовки, где указать углы поворота относительно осей и расстояние при перемещении вдоль каждой оси. Вывести координатную сетку и заключить каждое координатное пространство в ограничивающий параллелепипед. В строке заголовка задать название: Вращение и перемещение фигуры в пространстве. Скрыть строку меню и стандартную панель инструментов. Выбрать белый цвет для фона графического окна.

Решение:

```
>> % начальные данные для построения пирамиды
>> p(:,:,1)=[0,1,0,0;0,0,1,0;0,0,0,0]; % 1-й треугольник
>> p(:,:,2)=[0,1,0,0;0,0,0,0;0,0,1,0]; % 2-й треугольник
```

```

>> p(:,:3)=[0,0,0,0;0,1,0,0;0,0,1,0]; % 3-й треугольник
>> p(:,:4)=[1,0,0,1;0,1,0,0;0,0,1,0]; % 4-й треугольник
>> a = 45*pi/180; b = 45*pi/180; g = 45*pi/180; % углы поворота
>> % расстояние при перемещении тетраэдра
>> l = ones(3,4)./4; % перемещение по треугольникам
>> % матрицы поворота
>> Rx=[1,0,0; 0,cos(a),-sin(a); 0,sin(a),cos(a)];
>> Ry=[cos(b),0,sin(b); 0,1,0; -sin(b),0,cos(b)];
>> Rz=[cos(g),-sin(g),0; sin(g),cos(g),0; 0,0,1];
>> % перемещение тетраэдра
>> p(:,:5)=p(:,:1)+l; p(:,:6)=p(:,:2)+l;
>> p(:,:7)=p(:,:3)+l; p(:,:8)=p(:,:4)+l;
>> % поворот тетраэдра относительно оси x
>> p(:,:9)=Rx*p(:,:1); p(:,:10)=Rx*p(:,:2);
>> p(:,:11)=Rx*p(:,:3); p(:,:12)=Rx*p(:,:4);
>> % перемещение и поворот
>> p(:,:13)=Rx*Ry*Rz*p(:,:1)+l; p(:,:14)=Rx*Ry*Rz*p(:,:2)+l;
>> p(:,:15)=Rx*Ry*Rz*p(:,:3)+l; p(:,:16)=Rx*Ry*Rz*p(:,:4)+l;
>> % визуализация результатов
>> figure('MenuBar','none','NumberTitle','off','Color','w',...
'Name','Вращение и перемещение фигуры в пространстве')
>> subplot(2,2,1)
>> fill3(p(1,:,:1),p(2,:,:1),p(3,:,:1),'r',...
>> p(1,:,:2),p(2,:,:2),p(3,:,:2),'b',p(1,:,:3),p(2,:,:3),...
>> p(3,:,:3),'y',p(1,:,:4),p(2,:,:4),p(3,:,:4),'m')
>> grid on, box on, xlabel('x'), ylabel('y'), zlabel('z')
>> tt = ['\alpha=0^{\circ}, \beta=0^{\circ}, \gamma=0^{\circ}, ...
'x_{\{0\}}=0, y_{\{0\}}=0, z_{\{0\}}=0']; title(tt)
>> axis equal, axis([0,1.25,0,1.25,0,1.25]), view(50,20)
>> subplot(2,2,2)
>> fill3(p(1,:,:5),p(2,:,:5),p(3,:,:5),'r',...
p(1,:,:6),p(2,:,:6),p(3,:,:6),'b',p(1,:,:7),p(2,:,:7),...
p(3,:,:7),'y',p(1,:,:8),p(2,:,:8),p(3,:,:8),'m')
>> grid on, box on, xlabel('x'), ylabel('y'), zlabel('z')
>> tt = ['\alpha=0^{\circ}, \beta=0^{\circ}, \gamma=0^{\circ}, ...
'x_{\{0\}}=.25, y_{\{0\}}=.25, z_{\{0\}}=.25']; title(tt)
>> axis equal, axis([0 1.25 0 1.25 0 1.25]), view(50,20)
>> subplot(2,2,3)
>> fill3(p(1,:,:9),p(2,:,:9),p(3,:,:9),'r',...
p(1,:,:10),p(2,:,:10),p(3,:,:10),'b',p(1,:,:11),p(2,:,:11),...
p(3,:,:11),'y',p(1,:,:12),p(2,:,:12),p(3,:,:12),'m')
>> grid on, box on, xlabel('x'), ylabel('y'), zlabel('z')
>> tt = ['\alpha=' num2str(a*180/pi),'^{\circ}, \beta=0^{\circ}, \gamma=0^{\circ}, ...
'\gamma=0^{\circ}, x_{\{0\}}=0, y_{\{0\}}=0, z_{\{0\}}=0'];
>> title(tt)
>> axis equal, axis([0 1.25 -0.75 .75 0 1.25]), view(50,20)
>> subplot(2,2,4)

```

```

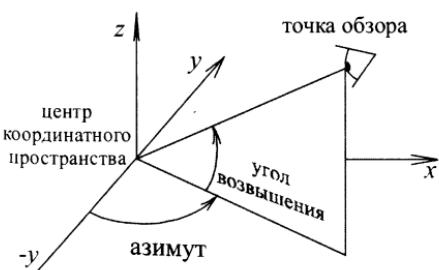
>> fill3(p(1,:,:13),p(2,:,:13),p(3,:,:13),'r',...
p(1,:,:14),p(2,:,:14),p(3,:,:14),'b',p(1,:,:15),p(2,:,:15),...
p(3,:,:15),'y',p(1,:,:16),p(2,:,:16),p(3,:,:16),'m')
>> grid on, box on, xlabel('x'), ylabel('y'), zlabel('z')
>> tt = ['\alpha=',num2str(a*180/pi),...
'^{\circ}, \beta=',num2str(b*180/pi),'^{\circ}, \gamma=',...
num2str(g*180/pi),'^{\circ}, x_{0}=1, y_{0}=1, z_{0}=1'];
>> title(tt), axis equal, axis([0,1.25,-0.75,.75,0,1.25]), view(50,20)

```

Результаты приведенных выше команд представлены на ил. 1. При построении пирамиды использовалась функция `fill3()`, которая позволяет отображать в пространстве закрашенные разными цветами многоугольники. В данном случае в функцию `fill3()` передавались координаты четырех треугольников, на основе которых строилась пирамида. При этом каждый треугольник изображался своим цветом. Функция `fill3()` является надстройкой над функцией-конструктором `patch()`, которая создает геометрические объекты на плоскости и в пространстве.

Функция `axis([xmin xmax ymin ymax zmin zmax])` позволяет ограничить выводимое в графическое окно координатное пространство. Эта функция принимает вектор, состоящий из шести элементов. Первая пара элементов определяет диапазон координатного пространства по оси x , вторая пара элементов – по оси y и третья пара элементов – по оси z . Та часть геометрических объектов, которая не попадает в ограниченное тремя осями координатное пространство, будет обрезана.

Для изменения точки обзора (`view()`) можно использовать функцию `view()`, которая принимает два входных аргумента. Первый входной аргумент соответствует углу *азимута* (*azimuth*), а второй – углу *возвышения* (*elevation*). Геометрическое описание этих углов представлено на рис. 5.22. *Азимут* представляет собой полярный угол, определяемый в плоскости xy . Положительные углы азимута отсчитываются от отрицательного направления оси y против часовой стрелки, если смотреть со стороны положительного



направления оси z . Угол *возвышения* определяет расположение наблюдателя над (положительные значения) или под (отрицательные значения) плоскостью xy . Все углы задаются в градусах. При построении графиков на плоскости xy по умолчанию применяются следующие значения: азимут равняется 0° , угол возвышения – 90° , а в случае объемных графиков – азимут составляет -37.5° и угол возвышения -30° .

Рис. 5.22. Определение точки обзора

5.2.3. Уравнения плоскости в пространстве

Поверхность в пространстве рассматривается как геометрическое место точек, удовлетворяющих какому-либо условию. *Уравнением поверхности* в выбранной системе координат называется такое уравнение ($F(x,y,z) = 0$ – в прямоугольной системе координат, $F(\theta,\rho,z) = 0$ – в цилиндрической системе координат, $F(\theta,\varphi,r) = 0$ – в сферической системе координат), которому удовлетворяют координаты каждой точки, принадлежащей поверхности, и не удовлетворяют координаты точек, не принадлежащих этой поверхности. Уравнение поверхности позволяет заменить изучение ее геометрических свойств исследованием уравнения поверхности. Например, для того, чтобы узнать, лежит ли точка на данной поверхности, достаточно подставить координаты рассматриваемой точки в уравнение поверхности вместо переменных. Если эти координаты удовлетворяют уравнению, то точка лежит на поверхности, если не удовлетворяет – не лежит. Простейшей поверхностью является *плоскость*. Плоскость в пространстве $Oxyz$ можно задать разными способами, каждому из которых соответствует определенный вид ее уравнения. Рассмотрим различные варианты задания плоскости в пространстве.

5.2.3.1. Общее уравнение плоскости

Общее уравнение плоскости имеет вид

$$Ax + By + Cz + D = 0,$$

где коэффициенты A, B, C и D – произвольные числа, причем A, B и C не равны нулю одновременно.

Частные случаи общего уравнения плоскости:

- Если $D = 0$, то оно примет вид $Ax + By + Cz = 0$. Этому уравнению удовлетворяет точка $O(0;0;0)$. Следовательно, в данном случае плоскость проходит через начало координат.
- Если $C = 0$, уравнение примет вид $Ax + By + D = 0$. Это уравнение совпадает с общим уравнением прямой линии на плоскости Oxy , т.е. является следом рассматриваемой плоскости в плоскости Oxy . Следовательно, рассматриваемая плоскость параллельна оси Oz . Если $B = 0$ – плоскость параллельна оси Oy , если $A = 0$ – параллельна оси Ox .
- Если $C = D = 0$, то плоскость проходит через точку $O(0;0;0)$ параллельно оси Oz , т.е. ось Oz лежит в плоскости $Ax + By = 0$. Аналогично уравнениям $Ax + Cz = 0$ и $By + Cz = 0$ соответствуют плоскости, в которых располагаются оси Oy и Ox соответственно.
- Если $A = B = 0$, то уравнение примет вид $Cz + D = 0$, т.е. $z = -D/C$. Плоскость параллельна плоскости Oxy . Аналогично уравнения $Ax + D = 0$

- и $By + D = 0$ обозначают плоскости, которые параллельны осям Oyz и Oxz соответственно.
5. Если $A = B = D = 0$, то уравнение примет вид $Cz = 0$, т. е. $z = 0$. Это уравнение плоскости Oxy . Аналогично $x = 0$ – уравнение плоскости Oyz , $y = 0$ – уравнение плоскости Oxz .

Пример 5.15. Построить плоскость, заданную общим уравнением $3x + 4y - 4z - 1 = 0$. Вывести обозначения осей и заголовок координатного пространства, в котором записано общее уравнение плоскости.

Решение:

```
>> A = 3; B = 4; C = -4; D = -1;
>> x = -1:0.4:1; y = -1:0.5:1;
>> [X,Y]=meshgrid(x,y);
>> Z = (-A*X-B*Y-D)/C;
>> surf(X,Y,Z), box on
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('3x + 4y - 4z - 1 = 0')
```

Результаты выполнения вышеприведенных команд представлены на ил. 2. При построении поверхности сначала формируются два вектора – x и y , которые определяют диапазоны изменения соответствующих координат и количество интервалов, на которые разбиваются эти диапазоны. Переменная x изменяется в диапазоне от -1 до 1 с шагом 0.4 , т.е. диапазон изменения этой переменной разбивается на пять интервалов. Переменная y изменяется в том же диапазоне, что и x , но с шагом 0.5 , т.е. диапазон разбивается на четыре интервала. С помощью функции `meshgrid()` начальные векторы x и y трансформируются в соответствующие матрицы X и Y . При этом число столбцов в матрицах X и Y совпадает с числом столбцов в векторе x , а число строк – с числом столбцов в векторе y . Следовательно, матрицы X и Y имеют размер 5×6 . В матрице X размещается построчно вектор x , а в матрице Y вектор y размещается по столбцам. Значения векторов x и y , а также матриц X и Y показаны на рис. 5.23. Таким образом, матрицы X и Y содержат координаты узловых точек, в которых будут вычисляться координаты плоскости по оси z . Значения этих координат после решения уравнения располагаются в матрице Z , которая имеет тот же самый размер, что и матрицы X и Y .

График плоскости строится с помощью функции `surf()`, первым входным аргументом которой является массив X , вторым – массив Y и третьим – массив Z . Построенная поверхность представляет собой набор клеток, которые соединены друг с другом в соответствующих узловых точках. По умолчанию функция `surf()` закрашивает каждую клетку поверхности определенным цветом, который зависит от значений элементов массива Z , образующих эту клетку. При этом для закрашивания клетки выбирается минимальное значение из четырех узловых точек. Цвет, соответствующий значению элемента

x	1	2	3	4	5	6	y	1	2	3	4	5
1	-1	-0.6	-0.2	0.2	0.6	1	1	-1	-0.5	0	0.5	1
X	1	2	3	4	5	6	Y	1	2	3	4	5
1	-1	-0.6	-0.2	0.2	0.6	1	1	-1	-1	-1	-1	-1
2	-1	-0.6	-0.2	0.2	0.6	1	2	-0.5	-0.5	-0.5	-0.5	-0.5
3	-1	-0.6	-0.2	0.2	0.6	1	3	0	0	0	0	0
4	-1	-0.6	-0.2	0.2	0.6	1	4	0.5	0.5	0.5	0.5	0.5
5	-1	-0.6	-0.2	0.2	0.6	1	5	1	1	1	1	1

Z	1	2	3	4	5	6
1	-2	-1.7	-1.4	-1.1	-0.8	-0.5
2	-1.5	-1.2	-0.9	-0.6	-0.3	0
3	-1	-0.7	-0.4	-0.1	0.2	0.5
4	-0.5	-0.2	0.1	0.4	0.7	1
5	0	0.3	0.6	0.9	1.2	1.5

Рис. 5.23. Значения векторов x и y , а также матриц X , Y и Z

массива Z , выбирает из цветовой палитры jet, которая используется по умолчанию для каждого графического окна. В табл. 5.8 представлены стандартные палитры системы MATLAB. Цветовая палитра представляет собой массив, состоящий из трех столбцов, строки которого соответствуют цветам палитры. Столбцы массива соответствуют цветам в аддитивной цветовой модели RGB.

Таблица 5.8

Стандартные палитры цветов системы MATLAB

Палитра	Описание цвета
autumn	Плавное изменение цвета от красного [1 0 0] до желтого [1 1 0]. Плавно изменяется зеленая составляющая цветовой модели RGB
bone	Оттенки серого цвета с добавлением оттенков синего цвета. К палитре gray добавлена палитра hot, в которой на месте столбца красного цвета использовался столбец синего цвета и, наоборот, на месте столбца синего цвета – столбец красного цвета
colorcube	Кубическое представление цвета. Описание такого представления цвета можно просмотреть на сайте http://www.colourcube.com
cool	Плавное изменение от голубого [0 1 1] до пурпурного [1 0 1] цвета. Интенсивность красного цвета постепенно увеличивается, а зеленого обратно пропорционально уменьшается
copper	Оттенки медного цвета. При получении оттенков медного цвета в качестве основы используется палитра gray. Значения красного цвета были умножены на 1.25, значения зеленого – на 0.7812 и значения синего – на 0.4975. Если результирующее значение красного цвета превышает единицу, то записывается единица

Окончание табл. 5.8

Палитра	Описание цвета
flag	Циклическое чередование красного [1 0 0], белого [1 1 1], синего [0 0 1] и черного [0 0 0] цветов
gray	Оттенки серого цвета. Плавное и синхронное изменение всех основных цветов от 0 до 1
hot	Плавное изменение цвета: черный [0 0 0] – красный [1 0 0] – желтый [1 1 0] – белый [1 1 1]. Палитра разбивается на три части в пропорциях 3:8, 3:8 и 2:8. В первой части реализуется переход цвета от черного к красному, во второй – от красного к желтому и в третьей – от желтого к белому
hsv	Цвет задается в терминах hue (оттенок), saturation (насыщенность) и value (значение). Используется следующая последовательность цветов: красный, желтый, зеленый, голубой, синий, пурпурный и красный. С помощью функции <code>hsv2rgb()</code> происходит преобразование матрицы значений цветовой модели HSV в цветовую модель RGB
jet	Разновидность цветовой палитры hsv. Цвета изменяются в следующей последовательности: темно-синий [0 0 0.5], синий [0 0 1], голубой [0 1 1], зеленый [0.5 1 0.5], желтый [1 1 0], оранжевый [1 0.5 0], красный [1 0 0] и темно-красный [0.5 0 0]. Данная палитра является палитрой по умолчанию
lines	Последовательность вывода цветов определяется с помощью свойства <code>ColorOrder</code>
pink	Палитра пастельных тонов розового цвета. Образуется путем смешивания палитры gray с палитрой hot, причем значения оттенков вычисляются по следующей формуле: $\sqrt{(2 \cdot \text{gray} + \text{hot})/3}$
prism	Циклическое чередование следующих цветов: красный, оранжевый, желтый, зеленый, синий и фиолетовый
spring	Оттенки пурпурного и желтого цветов. Начальным цветом является пурпурный [1 0 1], а конечным – желтый [1 1 0]
summer	Оттенки зеленого и желтого цветов. Палитра начинается с цвета [0 0.5 0.4] и заканчивается цветом [1 1 0.4], т.е. последовательно возрастает насыщенность красного и зеленого цветов
vga	Стандартная палитра операционной системы Windows, состоящая из 16 цветов
white	Один белый цвет
winter	Оттенки синего и зеленого цветов. Палитра изменяется от синего цвета [0 0 1] до цвета [0 1 0.5]

Для смены цветовой палитры графического окна необходимо задать команду `>> colormap(имя палитры(число_цветов))`. Число цветов по умолчанию равно 64.

5.2.3.2. Уравнение плоскости, проходящей через три заданные точки

Пусть даны точки $M_1(x_1; y_1; z_1)$, $M_2(x_2; y_2; z_2)$ и $M_3(x_3; y_3; z_3)$, не лежащие на одной прямой. Найдем уравнение плоскости, которая проходит через эти точки. Возьмем на плоскости произвольную точку $M(x; y; z)$ и образуем три вектора: $\overline{M_1 M} = (x - x_1; y - y_1; z - z_1)$, $\overline{M_1 M_2} = (x_2 - x_1; y_2 - y_1; z_2 - z_1)$, $\overline{M_1 M_3} = (x_3 - x_1; y_3 - y_1; z_3 - z_1)$. Эти три вектора лежат на плоскости, следовательно, они компланарны. Так как смешанное произведение компланарных векторов равно нулю, запишем уравнение плоскости в матричном виде:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0.$$

После ряда преобразований уравнение плоскости, заданной с помощью трех точек, можно записать в виде общего уравнения плоскости:

$$\begin{aligned} & \begin{vmatrix} y_2 - y_1 & z_2 - z_1 \\ y_3 - y_1 & z_3 - z_1 \end{vmatrix} x - \begin{vmatrix} x_2 - x_1 & z_2 - z_1 \\ x_3 - x_1 & z_3 - z_1 \end{vmatrix} y + \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} z + \\ & + \left(-\begin{vmatrix} y_2 - y_1 & z_2 - z_1 \\ y_3 - y_1 & z_3 - z_1 \end{vmatrix} x_1 + \begin{vmatrix} x_2 - x_1 & z_2 - z_1 \\ x_3 - x_1 & z_3 - z_1 \end{vmatrix} y_1 - \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} z_1 \right) = 0. \end{aligned}$$

Пример 5.16. Построить в координатном пространстве плоскость, заданную тремя точками: $M_1(1; 5; 2)$, $M_2(3; -2; 1)$ и $M_3(-4; 1; -2)$. Обозначить эти точки с помощью круговых маркеров красного цвета и поместить на против маркеров текстовые надписи: $M_1(x_1; y_1; z_1)$, $M_2(x_2; y_2; z_2)$ и $M_3(x_3; y_3; z_3)$ соответственно. Обозначить все координатные оси. Задать одинаковый масштаб всех осей и ограничить их кубом. Вывести координатную сетку. Задать цветовую палитру *autumn* и вывести в графическое окно ее шкалу. В строке заголовка графического окна написать: Плоскость, заданная тремя точками.

Решение:

```
>> x=-5:0.5:5; y=-5:0.5:5; % диапазоны изменения координат
>> M= [ [1;5;2], [3;-2;1], [-4;1;-2] ]; % координаты трех точек
>> % нижняя часть матрицы векторов
>> m=[M(1,2)-M(1,1), M(2,2)-M(2,1), M(3,2)-M(3,1); ...
M(1,3)-M(1,1), M(2,3)-M(2,1), M(3,3)-M(3,1)];
>> A=det(m(:,2:3)); B=det(m(:,[1,3]));... % определение коэффициентов
>> C=det(m(:,1:2)); D=-A.*M(1,1)+B*M(2,1)-C*M(3,1);
```

```

>> [X,Y]=meshgrid(x,y); % создание узловых координат внутри диапазона
>> Z = (-A*X+B*Y-D)/C; % решение уравнения
>> figure('NumberTitle','off', ... % создание графического окна
'Name','Плоскость, заданная тремя точками')
>> colormap(cool) % задание цветовой палитры
>> surf(x,y,Z) % построение плоскости
>> % оформление графика
>> box on, axis equal, axis square, hold on, colorbar
>> xlabel('x'), ylabel('y'), zlabel('z')
>> % построение и обозначение точек в пространстве
>> plot3(M(1,1),M(2,1),M(3,1),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor',[1 0 0])
>> text(M(1,1)-1.5, M(2,1)+0.05, M(3,1)+1,...
'\bf\it M_{\rm 1}(x_{\rm 1};y_{\rm 1};z_{\rm 1})')
>> plot3(M(1,2),M(2,2),M(3,2),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor',[1 0 0])
>> text(M(1,2)-1.5, M(2,2)+0.05, M(3,2)+1,...
'\bf\it M_{\rm 2}(x_{\rm 2};y_{\rm 2};z_{\rm 2})')
>> plot3(M(1,3),M(2,3),M(3,3),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor',[1 0 0])
>> text(M(1,3)-1.5, M(2,3)+0.05, M(3,3)+1,...
'\bf\it M_{\rm 3}(x_{\rm 3};y_{\rm 3};z_{\rm 3})')

```

Результаты приведенных выше команд представлены на ил. 3. При построении плоскости в координатном пространстве была задана команда `>> colorbar`, которая выводит в графическое окно справа от координатного пространства *шкалу текущей цветовой палитры*. Значения этой шкалы соответствуют значениям координаты z плоскости, т.е. минимальное значение координаты z соответствует минимальному значению шкалы, а максимальное значение координаты z – максимальному значению шкалы. Так как в палитре `cool` цвета изменяются плавно от голубого до пурпурного, то минимальные значения плоскости по координате z окрашивают ее в голубой цвет, а максимальные – в пурпурный. Команда `>> box on` охватывает координатное пространство в параллелепипед. Команда `>> axis square` превращает параллелепипед в куб, т.е. все ребра параллелепипеда имеют одинаковую длину. Команда `>> axis equal` задает одинаковый масштаб по всем трем координатным осям. Функция `plot3()` позволяет строить графики по точкам и выводить маркеры точек в координатном пространстве.

5.2.3.3. Уравнение плоскости в отрезках

Пусть плоскость отсекает на осях соответствующие отрезки a , b и c , т.е. проходит через три точки: $A(a;0;0)$, $B(0;b;0)$ и $C(0;0;c)$. Подставляя координаты этих точек в уравнение плоскости, проходящей через три точки, получаем

$$\begin{vmatrix} x-a & y & z \\ -a & b & 0 \\ -a & 0 & c \end{vmatrix} = 0 \Rightarrow bcx - abc + abz + acy = 0 \Rightarrow \frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1.$$

Полученное уравнение называется *уравнением плоскости в отрезках*.

Пример 5.17. Построить плоскость, заданную уравнением в отрезках, где точки $A=(-1;0;0)$, $B=(0;-2;0)$ и $C=(0;0;-2)$. В координатном пространстве построить черным цветом толщиной два пункта оси x , y и z , на которых в местах пересечений с плоскостью вывести круговые маркеры синего цвета и обозначить полужирным наклонным шрифтом координаты точек пересечения $A(a;0;0)$, $B(0;b;0)$ и $C(0;0;c)$ соответственно. Вывести обозначение осей и заголовок координатного пространства, в котором написать уравнение плоскости в отрезках. Вывести координатную сетку и охватывающий координатное пространство куб. Задать одинаковый масштаб и точку наблюдения с координатами $(50;21)$. Выбрать стандартную цветовую палитру *spring*.

Решение:

```
>> % задание плоскости
>> x=-2:0.5:1; y=-3:0.5:1; [X,Y]=meshgrid(x,y);
>> M=[[-1;0;0],[0;-2;0],[0;0;-2]]; % координаты трех точек
>> Z = (1-X./M(1,1)-Y./M(2,2))*M(3,3); % решение уравнения
>> % создание графического окна
>> figure('RendererMode','manual','Renderer','zbuffer')
>> colormap(spring) % задание цветовой палитры
>> % построение координатных осей
>> zer = zeros(1,2); xlim=[min(min(X)),max(max(X))];
>> ylim=[min(min(Y)),max(max(Y))];
>> zlim=[min(min(Z)),max(max(Z))];
>> line(xlim,zer,zer,'Color','k','LineWidth',2)
>> line(zer,ylim,zer,'Color','k','LineWidth',2)
>> line(zer,zer,zlim,'Color','k','LineWidth',2)
>> % задание свойств объекта axes
>> view(50,21), hold on, grid on
>> box on, axis equal, axis square
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('\itx/a + y/b + z/c \rm= 1')
>> surface('XData',X,'YData',Y,'ZData',Z, 'CData', Z,... % толщина линий сетки на плоскости
'FaceColor', 'interp',... % цвет поверхностей клеток
'EdgeColor', 'white') % цвет контуров клеток
>> % построение и обозначение точек в пространстве
>> plot3(M(1,1),M(2,1),M(3,1),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor', [0 0 1])
>> text(M(1,1)+.025,M(2,1)+.15,M(3,1)+.15,' \bf\itA(a;0;0) ')
```

```

>> plot3(M(1,2),M(2,2),M(3,2),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor',[0 0 1])
>> text(M(1,2)-0.1,M(2,2)-0.1,M(3,2)+0.6,'\bf{itB}(0;b;0)')
>> plot3(M(1,3),M(2,3),M(3,3),'Marker','o',...
'MarkerSize',8,'MarkerFaceColor',[0 0 1])
>> text(M(1,3),M(2,3)+.2,M(3,3)+.1,'\bf{itC}(0;0;c)')

```

Результаты вышеприведенных команд показаны на ил. 4. Одной из основных проблем при построении трехмерных сцен является проблема определения видимых объектов. Обычно для решения этой задачи применяют алгоритм z-buffer, встроенный в современные графические ускорители. Сущность этого алгоритма заключается во введении дополнительного буфера по координате z , где хранятся значения удаленности всех видимых пикселей изображения. Во время прорисовки очередного пикселя изображаемого объекта с координатами x и y его координата z сравнивается со значением координаты z , расположенной в z -буфере. Если новый пиксель имеет глубину больше, чем записанная в z -буфере, то эта точка объекта закрыта другим объектом и не отображается на экране. В противном случае значение записывается в z -буфер. Этот пиксель будет виден до тех пор, пока другой объект его не закроет. После вычисления трехмерной сцены в изображении используются все пиксели, которые остались в z -буфере. При выводе плоскости совместно с координатными осями некоторая часть координатных осей будет располагаться под плоскостью, т.е. скрыта от глаза наблюдателя. По умолчанию свойство `Renderer`, которое отвечает за метод прорисовки, установлено в `Painter`. Данный режим обеспечивает быструю прорисовку графического окна, содержащего простые объекты, но не обеспечивает реалистичное представление трехмерных сцен. Для того чтобы скрыть невидимые части необходимо изменить *метод прорисовки* (*rendering*) графического окна на `ZBuffer` или `OpenGL`. `OpenGL` (*Open Graphics Library*) является графическим стандартом в области компьютерной графики. Стандарт `OpenGL` был разработан в 1992 г. целым рядом ведущих в компьютерной индустрии фирм на основе графической библиотеки `IRIS GL` фирмы *Silicon Graphics*. Система `MATLAB` может работать с `OpenGL 1.3`. Использование метода `OpenGL` позволяет проводить прорисовку графического окна с помощью аппаратных средств, что увеличивает быстродействие. Только метод `OpenGL` поддерживает изменение прозрачности и освещенности объектов. Метод `ZBuffer` поддерживает только изменение освещенности, а метод `Painter` не поддерживает ни изменение прозрачности, ни изменение освещенности. В том случае, если возникают проблемы с прорисовкой графического изображения при использовании метода `OpenGL`, следует обратиться за дополнительной информацией по следующему электронному адресу: <http://www.mathworks.com/support/tech-notes/1200/1201.shtml>.

Для построения координатных осей была применена функция `line()`, которая строит линии внутри координатного пространства. Первые три входных аргумента в функцию `line()` являются векторами одинаковой размерности, которые определяют соответствующие координаты начала и конца линии.

Плоскость была построена с помощью функции-конструктора `surface`, которая служит для создания поверхности внутри координатного пространства. Для изменения отображения поверхности, которое принято по умолчанию в системе MATLAB, в функцию-конструктор передаются значения некоторых свойств. Свойства `XData`, `YData` и `ZData` определяют координаты узлов строящейся поверхности. Свойство `CData` определяет цвет координатных узлов поверхности. Минимальное и максимальное значения свойства `CData` соответствуют первому и последнему цветам в палитре цветов `Colormap`. Так как для графического окна была выбрана палитра `spring`, то плоскость окрашивается в цвета этой палитры. При этом пурпурный цвет преобладает в тех местах, где значения координат z близки к минимальному значению, а желтый цвет – в тех местах, где значения координат z приближаются к максимальному значению. До сих пор при отображении плоскости в пространстве каждая клетка плоскости окрашивалась в один цвет, который соответствует значению первого узла в клетке. Для задания плавного перехода от одного цвета к другому в пределах одной клетки свойству `FaceColor` необходимо присвоить значение `interp`. По умолчанию это значение равно `flat`. Если этому свойству присвоить значение, которое обозначает один цвет (например, `'FaceColor', [1 0 0]`), то вся поверхность будет окрашена в этот цвет. Для того чтобы скрыть поверхности клеток, этому свойству присваивается значение `none`. Сказанное выше для свойства `FaceColor` справедливо и для свойства `EdgeColor`, но в данном случае изменяется цвет не поверхности клетки, а линий, которые соединяют узловые точки поверхности. Цвет линий был изменен с черного, который используется по умолчанию, на белый. Толщина этих линий задается с помощью свойства `LineWidth`.

5.2.3.4. Уравнение плоскости, проходящей через точку перпендикулярно данному вектору

Пусть в пространстве $Oxyz$ задана плоскость S с помощью точки $M_0(x_0; y_0; z_0)$ и вектора $\bar{n} = (A; B; C)$, который перпендикулярен этой плоскости. Возьмем на плоскости произвольную точку $M(x; y; z)$ и составим вектор $\overline{M_0M} = (x - x_0; y - y_0; z - z_0)$. При любом расположении точки M на плоскости S векторы \bar{n} и $\overline{M_0M}$ взаимно перпендикулярны. Следовательно, их скалярное произведение равно нулю, т.е.

$$A(x - x_0) + B(y - y_0) + C(z - z_0) = 0.$$

Полученное уравнение называется *уравнением плоскости*, проходящей через данную точку $M_0(x_0; y_0; z_0)$ перпендикулярно вектору $\bar{n} = (A; B; C)$. Вектор называется *нормальным вектором плоскости*.

Пример 5.18. Построить плоскость, проходящую через точку $M(1.5; -1.5; -1)$ перпендикулярно вектору $\bar{n} = (1; 1; 1)$. Плоскость вывести в виде сетки. Цвет поверхности совпадает с цветом фона. Цвет линий, соединяющих узловые точки плоскости, плавно изменяется согласно цветовой палитре hsv. Обозначить точку в виде кругового маркера черного цвета размером 8 пунктов. Рядом с маркером расположить текстовую надпись: $M(x_0; y_0; z_0)$. Изобразить красным цветом толщиной 3 пункта вектор \bar{n} , начало которого совпадает с центром координатного пространства. Вывести обозначение вектора $n(A; B; C)$. Черным цветом толщиной 2 пункта изобразить координатные оси. Задать обозначения осей и заголовок координатного пространства. В заголовке написать уравнение $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$.

Решение:

```
>> % задание плоскости
>> x=-1.5:0.2:1.5; y=-1.5:0.2:1.5; [X, Y]=meshgrid(x, y);
>> M=[1.5; -1.5; -1]; n=[1; 1; 1]; % координаты точки и вектора
>> Z = (-n(1).* (X-M(1))-n(2).* (Y-M(2)))./n(3)+M(3);
>> % создание графического окна
>> figure('RendererMode','manual','Renderer','zbuffer')
>> colormap(hsv) % задание цветовой палитры
>> % построение координатных осей
>> zer = zeros(1,2); xlim=[min(min(X)), max(max(X))];
>> ylim=[min(min(Y)), max(max(Y))];
>> zlim=[min(min(Z)), max(max(Z))];
>> line(xlim,zer,zer,'Color','k','LineWidth',2)
>> line(zer,ylim,zer,'Color','k','LineWidth',2)
>> line(zer,zer,zlim,'Color','k','LineWidth',2)
>> % задание свойств объекта axes
>> view(70,15), hold on, grid on, box on, axis equal
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('\it A(x-x_{\rm 0})+B(y-y_{\rm 0})+C(z-z_{\rm 0})=0')
>> mesh(X, Y, Z)
>> shading interp
>> % построение и обозначение точек в пространстве
>> plot3(M(1), M(2), M(3), 'Marker', 'o',...
'MarkerSize', 8, 'MarkerFaceColor', [0 0 0])
>> text(M(1), M(2)+0.2, M(3), '\bf\it M(x_{\rm 0}; y_{\rm 0}; z_{\rm 0})')
>> line([0 n(1)], [0 n(2)], [0 n(3)], 'Color', 'r', 'LineWidth', 3)
>> text(n(1), n(2), n(3), '\bf\it n(A; B; C)')
```

Результаты вышеприведенных команд представлены на ил. 5. Функция `mesh()`, которая является надстройкой над функцией-конструктором `surface()`, выводит в координатное пространство поверхность в виде сетки. Цвет сетки определяется свойством `EdgeColor` объекта `surface`. Цвет поверхностей клетки совпадает с цветом фона. В рассматриваемом примере свойство `FaceColor` равно `[1 1 1]`, что соответствует белому цвету. Применение команды `shading` после создания объекта `surface` приводит к изменению свойств `EdgeColor` и `FaceColor` согласно табл. 5.9. В результате задания команды `shading` цвет сетки будет плавно изменяться от узла к узлу.

Таблица 5.9

Результаты задания команды `shading`

Свойство	Команда		
	<code>mesh</code>	<code>shading flat</code>	<code>shading interp</code>
<code>EdgeColor</code>	<code>flat</code>	<code>flat</code>	<code>interp</code>
<code>FaceColor</code>	Цвет фона	Цвет фона	Цвет фона
	<code>surf</code>	<code>shading flat</code>	<code>shading interp</code>
<code>EdgeColor</code>	<code>[0 0 0]</code>	<code>none</code>	<code>none</code>
<code>FaceColor</code>	<code>flat</code>	<code>flat</code>	<code>interp</code>

5.2.4. Уравнения линии в пространстве

Линию в пространстве можно определить несколькими способами. Рассмотрим различные способы задания линии в пространстве, а также ее визуализацию с помощью графических средств системы MATLAB.

5.2.4.1. Общие уравнения прямой линии

Прямую линию в пространстве можно задать как линию пересечения двух непараллельных плоскостей. Рассмотрим систему уравнений

$$\begin{cases} A_1x + B_1y + C_1z + D_1 = 0, \\ A_2x + B_2y + C_2z + D_2 = 0. \end{cases}$$

Каждое из приведенных уравнений определяет плоскость. В том случае, если заданные плоскости не параллельны (координаты векторов $\bar{n}_1 = (A_1; B_1; C_1)$ и $\bar{n}_2 = (A_2; B_2; C_2)$ не пропорциональны), то система уравнений определяет прямую линию L как геометрическое место точек пространства, координаты которых удовлетворяют каждому из этих уравнений. Представленные уравнения называют *общими уравнениями прямой*.

Пример 5.19. Построить линию, являющуюся пересечением двух плоскостей, заданных общими уравнениями $x - y - z + 1 = 0$ и $0.5x - y + z - 0.1 = 0$. При построении первой плоскости создать массив цветов, который изменяется от синего цвета к красному и содержит значения цвета для каждого координатного узла поверхности в терминологии RGB. При построении второй поверхности использовать стандартную цветовую палитру gray. Задать одинаковый масштаб всех осей и вывести их обозначения. Вывести координатную сетку. Охватить координатное пространство параллелепипедом. В заголовке координатного пространства вывести общие уравнения прямой линии. Задать точку наблюдения с координатами $(-70; 30)$.

Решение:

```
>> A=[1,-.5]; B=[-1,-1]; C=[-1,1]; D=[1,-.1]; % коэффициенты
>> % координаты узловых точек
>> x=-2:0.4:2; y=-2:0.4:2; [X,Y]=meshgrid(x,y);
>> % решение двух уравнений
>> Z(:,:,1)=(-A(1)*X-B(1)*Y-D(1))/C(1);
>> Z(:,:,2)=(-A(2)*X-B(2)*Y-D(2))/C(2);
>> % создание диалогового окна
>> figure('NumberTitle','off','Name','Общие уравнения линии',...
' Renderer','zbuffer')
>> % ----- ЗАДАНИЕ ЦВЕТА ДЛЯ 1-Й ПЛОСКОСТИ -----
>> % определение размерности страницы в массиве Z
>> [height_Z width_Z]=size(Z(:,:,1));
>> % формирование вектора изменения насыщенности цвета
>> r=(0:height_Z-1)'./(height_Z)+0.25;
>> r=min(r,1); % вписывание значений вектора в диапазон [0 1]
>> % задание массивов для основных цветов
>> R=r*ones(1,width_Z); G=r*zeros(1,width_Z);
>> B=flipud(r*ones(1,width_Z));
>> % формирование многомерного массива для цвета
>> TrueColor(:,:,:,1)=R; TrueColor(:,:,:,2)=G; TrueColor(:,:,:,3)=B;
>> %
>> surf(X,Y,Z(:,:,:1),TrueColor) % построение 1-й плоскости
>> hold on % включение режима добавления графиков
>> colormap(gray) % задание цветовой палитры для 2-й плоскости
>> surf(X,Y,Z(:,:,:2)) % построение 2-й плоскости
>> [cmin cmax] = caxis; % определение значений свойства CLim
>> caxis([cmin-4 cmax]) % увеличение диапазона CLim
>> box on, axis equal, view(-70,30) % оформление графика
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('itA_{\rm1}x+B_{\rm1}y+C_{\rm1}z+D_{\rm1}=0
itA_{\rm2}x+B_{\rm2}y+C_{\rm2}z+D_{\rm2}=0')
```

Результаты выполнения вышеприведенных команд представлены на ил. 6. Массив Z состоит из двух страниц, в которых размещаются значения по оси z

для первой и второй плоскостей соответственно. Размер каждой страницы совпадает с размером массивов X и Y , которые определяют координаты узловых точек плоскостей по осям x и y соответственно. Для задания цвета первой плоскости использовался многомерный массив `TrueColor` размера $m \times n \times 3$, где m – число строк в массивах данных X и Y , n – число столбцов в этих массивах. Первая страница массива `TrueColor` определяет насыщенность красного цвета для каждой точки плоскости, вторая страница – насыщенность зеленого цвета и третья страница – насыщенность синего цвета. Так как насыщенность основных цветов должна лежать в диапазоне $[0;1]$, то для выполнения этого условия использовалась функция `min()`, которая заменяла единицей все значения массива, выходящие за диапазон. При визуализации второй плоскости система MATLAB определяет диапазон изменения значений оси z и масштабирует выбранную палитру таким образом, чтобы конечные цвета соответствовали самым большим значениям массива, а начальные цвета палитры – минимальным значениям массива. С помощью функции `caxis()` сначала был получен диапазон изменения оси z $[-3;5]$, который располагается в свойстве `CLim` объекта `Axes`, а затем этот диапазон был изменен $[-7;5]$, что привело к увеличению белого цвета во второй плоскости.

5.2.4.2. Параметрические уравнения прямой линии

Положение прямой в пространстве вполне определено, если задать какую-либо точку $M_0(x_0; y_0; z_0)$, лежащую на прямой линии, и вектор \bar{V} , параллельный этой прямой. Вектор \bar{V} называется *направляющим вектором прямой линии*. Пусть прямая линия L задана ее точкой $M_0(x_0; y_0; z_0)$ и направляющим вектором $\bar{V} = (m; n; p)$. Возьмем на прямой линии L произвольную точку $M(x; y; z)$. Если обозначить радиусы-векторы точек M_0 и M через \bar{r}_0 и \bar{r} соответственно, то три вектора связаны соотношением

$$\bar{r} = \bar{r}_0 + \overline{M_0 M}.$$

Вектор $\overline{M_0 M}$, лежащий на прямой линии L , параллелен направляющему вектору \bar{V} , поэтому $\overline{M_0 M} = t \cdot \bar{V}$, где t – скалярный множитель, называемый *параметром*, может принимать различные значения в зависимости от положения точки M на прямой. Так как $\bar{r} = (x; y; z)$, $\bar{r}_0 = (x_0; y_0; z_0)$ и $t\bar{V} = (tm; tn; tp)$, то приведенное соотношение можно записать в виде системы уравнений:

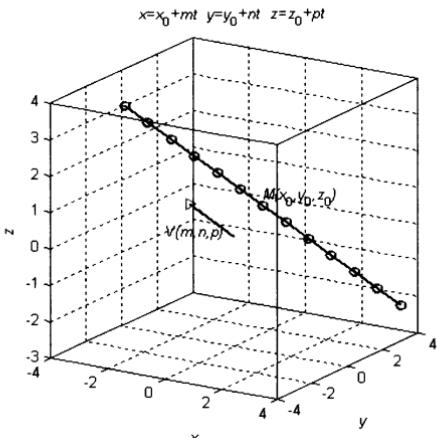
$$\begin{cases} x = x_0 + tm, \\ y = y_0 + tn, \\ z = z_0 + tp, \end{cases}$$

являющихся параметрическими уравнениями прямой линии в пространстве.

Пример 5.20. Построить прямую линию красного цвета, заданную параметрическими уравнениями $x = 1 - t$; $y = -t$; $z = 1 + t$. Параметр t изменяется в диапазоне от $[-3;3]$ с шагом 0.5. В местах вычисленных координат прямой линии вывести круговые маркеры. Обозначить на графике исходную точку $M_0(x_0;y_0;z_0)$. Построить направляющий вектор темно-зеленого цвета $[0\ 0.5\ 0]$ и обозначить его $V(m;n;p)$. В качестве направляющей стрелки вектора выбрать треугольный маркер соответствующего цвета и размером 6 пунктов, вершина которого направлена вправо. Вызвести координатную сетку и обрамляющий координатное пространство параллелепипед. Задать обозначения координатных осей и в заголовке координатного пространства записать параметрические уравнения прямой линии.

Решение:

```
>> % --- Задание линии в пространстве
>> t=[-3:0.5:3]; M=[1;0;1]; V=[-1;-1;1];
>> x=M(1)+V(1)*t; y=M(2)+V(2)*t; z=M(3)+V(3)*t;
>> % --- Создание графических объектов
>> Fig = figure; % создание графического окна
>> Ax = axes; % создание координатного пространства
>> L=plot3(x,y,z); % построение линии
>> Vec=line([0 V(1)], [0 V(2)], [0 V(3)]); % создание вектора
>> Mar=line(V(1),V(2),V(3)); % создание стрелки
>> text(-1.85,-1,0.1,'\'itV(m;n;p)') % обозначение вектора
>> text(1,0,1.2,'\'itM(x_{\rm 0};y_{\rm 0};z_{\rm 0})') % точка
>> Text = text(0,0,0,...,
'\itx=x_{\rm 0}+mt y=y_{\rm 0}+nt z=z_{\rm 0}+pt')
>> % --- Определение свойств созданных объектов
>> set(Fig,... определение свойств графического окна
'Color',[1 1 1], 'Name','Parametric line',...
'NextPlot','add', 'NumberTitle','off','Position', [200 200 400 400]);
>> set(Ax,... задание свойств координатного пространства
'Box','on', 'Title', Text,... задание заголовка
'View', [32, 17], ... точка обзора
'XGrid','on', 'XLabel', xlabel('\itx'),... обозначение оси x
'XLimMode','manual', 'XLim', [-4;4],... диапазон оси x
'YGrid','on', 'YLabel', ylabel('\ity'),... обозначение оси y
'YLimMode','manual', 'YLim', [-4;4],... диапазон оси y
'ZGrid','on', 'ZLabel', zlabel('\itz'),... обозначение оси z
'ZLimMode', 'manual', 'ZLim', [-3;4]) % диапазон оси z
>> set(L, 'Color',[1 0 0],... задание свойств линии
'LineWidth',2, 'Marker', 'o', 'MarkerSize', 6)
>> set(Vec, 'Color',[0 .5 0],'LineWidth',2) % задание свойств вектора
>> set(Mar,... задание свойств стрелки
'Color',[0 .5 0],'Marker','>', 'MarkerSize',6)
```



Р и с . 5.24. Прямая линии, заданная параметрическими уравнениями

`plot3()`, а вектор и стрелка на конце конструктора `line()`. На втором этапе, используя функцию `set()`, были изменены значения некоторых свойств созданных объектов для улучшения внешнего вида. Первым входным аргументом функции `set()` должен быть дескриптор объекта, свойства которого необходимо изменить. После имени объекта должны следовать пары аргументов: имя свойства и его значение. В ряде случаев на месте значения можно задавать имена других объектов или функций. При задании заголовка координатного пространства в качестве значения использовался текстовый объект `Text`. При обозначении координатных осей использовались соответствующие функции. Функция `set()` может также использоваться для просмотра возможных вариантов значений свойств указанного объекта: `>> set(имя объекта)`.

5.2.4.3. Канонические уравнения прямой линии

Пусть $\bar{V} = (m; n; p)$ – направляющий вектор прямой L и $M_0(x_0; y_0; z_0)$ – точка, лежащая на этой прямой. Вектор $\overline{M_0M}$, соединяющий точку с произвольной точкой прямой линии, параллелен вектору \bar{V} . Следовательно, координаты вектора $\overline{M_0M}$ и вектора \bar{V} пропорциональны:

$$\frac{x - x_0}{m} = \frac{y - y_0}{n} = \frac{z - z_0}{p}.$$

Полученные уравнения называются *каноническими уравнениями прямой линии в пространстве*.

Результаты выполнения вышеприведенных команд представлены на рис. 5.24. Процесс построения линии с направляющим вектором был разбит на два этапа. На первом этапе были созданы все необходимые графические объекты с помощью соответствующих функций-конструкторов. При создании объектов (графического окна, координатного пространства, трех линий и одного текстового объекта) им присваивалось имя (*дескриптор*). График линии в координатном пространстве был построен с помощью функции вектора – с помощью функции-конструктора `line()`.

На втором этапе, используя функцию `set()`, были изменены значения некоторых свойств созданных объектов для улучшения внешнего вида.

Обращение в нуль одного из знаменателей уравнений означает обращение в нуль соответствующего числителя:

$$\frac{x-1}{0} = \frac{y+3}{2} = \frac{z-2}{4}.$$

Прямая линия проходит через точку $M(1;-3;2)$ перпендикулярно оси Ox (проекция направляющего вектора на ось x равна нулю).

Если приравнять канонические уравнения к параметру t , то их можно преобразовать в параметрические уравнения прямой линии:

$$\frac{x - x_0}{m} = \frac{y - y_0}{n} = \frac{z - z_0}{p} = t.$$

При построении прямой линии, заданной каноническими уравнениями, в графическом окне системы MATLAB необходимо преобразовать канонические уравнения в параметрические уравнения линии и выполнить соответствующие команды для ее визуализации.

Пример 5.21. Построить прямую линию толщиной 2 пункта, заданную в виде канонических уравнений: $-(x + 3) / 0.5 = (y - 0.5) / 0.5 = -(z + 4) / 0.5$. Параметр t изменяется в диапазоне $[-3;3]$ с шагом 0.5. В местах вычисленных координат линии вывести круговые маркеры. Отобразить направляющий вектор черного цвета толщиной 2 пункта. В качестве стрелки вектора выбрать треугольный маркер размером 10 пунктов. Обозначить точку, лежащую на прямой, $M_0(x_0;y_0;z_0)$ и направляющий вектор V (направляющий вектор). Вызвать координатную сетку, задать одинаковый масштаб по осям и выбрать светло-серый цвет [0.95 0.95 0.95] в качестве фона координатного пространства. Установить точку наблюдения в позицию $(-57; 30)$. Обозначить оси и задать заголовок, в котором написать каноническое уравнение линии.

Решение:

```
>> t=[-3:0.5:3]; M=[-3;0.5;-4]; V=[-1.5;0.5;-0.5]; % задание коорд.
>> XYZ=M*ones(size(t))+V*t; % задание вектора
>> % создание объектов
>> figure; axes; L=plot3(XYZ(1,:),XYZ(2,:),XYZ(3,:));
>> Vec=line([0,V(1)], [0,V(2)], [0,V(3)]); Arrow=line(V(1),V(2),V(3));
>> text(M(1),M(2),M(3)-.5,'\\itM(x_{\\rm0};y_{\\rm0};z_{\\rm0})')
>> text(V(1),V(2)-1,V(3)+0.75,'\\itV(m;n;p)')
>> % Задание свойств
>> set(gcf,'Renderer','opengl', 'Color',[1 1 1],...
'Position', [200 200 330 400], 'CurrentObject',Arrow);
>> set(gca, 'View', [-57, 30],... точка обзора
'Color', [0.95 0.95 0.95],... светло-серый цвет коорд. пространства
'DataAspectRatioMode', 'Manual',...)
```

```
'DataAspectRatio', [1 1 1],... одинаковый масштаб по всем трем осям
'XGrid','on','YGrid','on','ZGrid','on',...
'Title',... заголовок
title('-(\itx\rm+3)/1.5 = (\ity\rm-0.5)/0.5 = -(\itz\rm+4)/0.5',...
'XLabel', xlabel('x'), 'YLabel', ylabel('y'), 'ZLabel', zlabel('z'));
>> set(L,'LineWidth',2,'Marker','o')
>> set(Vec,'LineWidth',2,'Color','k')
>> set(gco,'LineWidth',3,'Marker','','^','MarkerSize',10,'Color','k')
>> Text=findobj('Type','text','Position',[V(1),V(2)-1,V(3)+0.75,]);
>> set(Text,'FontWeight','bold')
```

Результаты выполнения команд показаны на рис. 5.25. При задании свойств объектам графического окна были использованы команды, которые позволяют получить дескрипторы объектов. Команда `>> gcf – get current figure` – позволяет получить *дескриптор текущего графического окна*. Эта

$$-(x+3)/1.5 = (y-0.5)/0.5 = -(z+4)/0.5$$

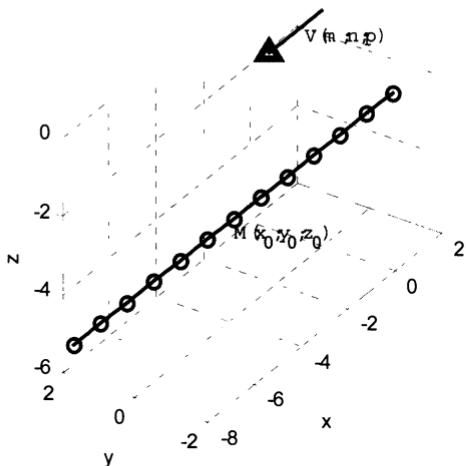


Рис. 5.25. График линии, заданной каноническими уравнениями

`current object` – позволяет получить *дескриптор текущего объекта* графического окна. Эта команда считывает значение свойства `CurrentObject` графического окна. По умолчанию свойство содержит пустой массив, что соответствует отсутствию в графическом окне текущего объекта. Назначить текущий объект можно с помощью присваивания свойству `CurrentObject` дескриптора созданного объекта или с помощью щелчка мыши на объекте

команда считывает значение свойства `CurrentFigure` корневого объекта (`Root`), обозначающего экран. Если на момент задания команды нет ни одного графического окна, то оно будет создано и его дескриптор будет результатом выполнения команды. Команда `>> gca – get current axes` – используется для получения *дескриптора текущего координатного пространства*. В том случае, если на момент задания команды не существовало координатного пространства в текущем графическом окне, то оно будет создано и его дескриптор будет результатом выполнения команды. Эта команда обращается к свойству `CurrentAxes` графического окна. Команда `>> gco – get`

в режиме редактирования. Для поиска объекта используется функция `findobj()`. Входными аргументами функции являются критерии поиска, задаваемые в виде пары: имя свойства и его значение. Входной аргумент `Type` определяет тип искомого объекта. Функция `findobj()` возвращает дескриптор или вектор дескрипторов объектов, которые удовлетворяют критериям поиска. По умолчанию поиск осуществляется последовательно по всем ветвям древовидной структуры объектов, т.е. во всех координатных пространствах каждого графического окна. Для ограничения поиска следует в качестве первого аргумента функции задать дескриптор объекта, внутри которого необходимо произвести поиск.

5.2.4.4. Уравнение прямой линии в пространстве, проходящей через две точки

Пусть прямая линия L проходит через точки $M_1(x_1; y_1; z_1)$ и $M_2(x_2; y_2; z_2)$. Вектор $\overline{M_1 M_2} = (x_2 - x_1; y_2 - y_1; z_2 - z_1)$ можно использовать в качестве направляющего вектора \vec{V} , т.е. $\vec{V} = \overline{M_1 M_2}$. Таким образом, $m = x_2 - x_1$, $n = y_2 - y_1$ и $p = z_2 - z_1$. Так как прямая линия L проходит через точку $M_1(x_1; y_1; z_1)$, то, согласно каноническим уравнениям прямой линии, уравнения прямой линии, проходящей через две заданные точки, имеют вид

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1}.$$

При построении прямой линии, заданной с помощью двух точек, необходимо преобразовать исходные уравнения в параметрические, которые будут иметь следующий вид:

$$x = x_1 + (x_2 - x_1) \cdot t; \quad y = y_1 + (y_2 - y_1) \cdot t; \quad z = z_1 + (z_2 - z_1) \cdot t.$$

Пример 5.22. Построить прямую линию черного цвета толщиной 3 пункта, заданную с помощью двух точек: $M_1(-0.2;-0.5;1)$ и $M_2(2;1;-3)$. Обозначить эти точки круговыми маркерами, толщина линий которых 3 пункта. Рядом с соответствующими маркерами вывести текстовые надписи: $M_1(x_1; y_1; z_1)$ и $M_2(x_2; y_2; z_2)$. Параметр t изменяется в диапазоне $[-1;2]$ с шагом 0.1. Вызвести координатную сетку, обозначить оси координат, задать одинаковый масштаб по всем осям и охватить координатное пространство параллелепипедом. Установить точку обзора в позицию с координатами (36;12). В заголовке графического окна вывести надпись: `Figure No. 1: Прямая линия.` Скрыть строку меню и стандартную панель инструментов. В качестве фона графического окна выбрать белый цвет. Левый нижний угол графического окна размещается на рабочем столе в позиции (200;200). Ширина окна составляет 290 точек, а высота – 320 точек.

Решение:

```

>> M = [[-.2;2], [-.5;1], [1;-3]]; % координаты точек
>> t = -1:0.1:2; % диапазон изменения параметра
>> x = M(1)+(M(2)-M(1))*t; y=M(3)+(M(4)-M(3))*t;
>> z = M(5)+(M(6)-M(5))*t;
>> % создание объектов
>> figure; axes; line(x,y,z);
>> point1=line(M(1),M(3),M(5)); % точка M1(x1;y1;z1)
>> point2=line(M(2),M(4),M(6)); % точка M2(x2;y2;z2)
>> text(0,0,0,'\itM_{\rm1}(x_{\rm1},y_{\rm1},z_{\rm1})')
>> text(0,0,0,'\itM_{\rm2}(x_{\rm2},y_{\rm2},z_{\rm2})')
>> % определение дескрипторов объектов
>> Fig = get(0,'CurrentFigure'); Ax = get(Fig,'CurrentAxes');
>> Ln = findobj(Ax,'Type','Line');
>> T1 = findobj(Ax,'Type','Text','String',...
'<i>\itM_{\rm1}(x_{\rm1},y_{\rm1},z_{\rm1})</i>');
>> T2 = findobj(Ax,'Type','Text','String',...
'<i>\itM_{\rm2}(x_{\rm2},y_{\rm2},z_{\rm2})</i>');
>> % задание значений свойствам объектов
>> set(Fig,'Color','w','MenuBar','none','Name','Прямая линия',...
'Position',[200 200 290 320])
>> set(Ax,'DataAspectRatioMode','Manual',...
'DataAspectRatio',[1 1 1],...
'Box','on','View',[36 12],...
'XLabel', xlabel('x'), 'YLabel', ylabel('y'), 'ZLabel', zlabel('z'), ...
'XGrid','on','YGrid','on','ZGrid','on');
>> set([point1 point2],'Marker','o','LineWidth',3);
>> set(Ln,'LineWidth',3,'Color','k');
>> set(T1,'Position',[M(1)+.5,M(3),M(5)]);
>> set(T2,'Position',[M(2)+.5,M(4),M(6)]);

```

Результаты приведенных команд представлены на рис. 5.26. Для получения дескрипторов графического окна и координатного пространства использовалась функция `get()`. В первом случае функция вернет значение свойства `CurrentFigure` корневого объекта. Если на данный момент нет ни одного графического окна, то функция `get()` возвратит пустой массив. Во втором случае функция `get()` возвращает значение свойства `CurrentAxes` графического окна. Если на момент выполнения функции в текущем графическом окне нет координатного пространства, то будет возвращен пустой массив. Рассмотренные ранее команды `gcf` и `gca` обращаются к функции `get()` при считывании значений соответствующих свойств.

При выполнении поиска объектов использовался дескриптор координатного пространства, внутри которого происходил поиск. Если не задать дескриптор, то система MATLAB будет последовательно производить поиск объектов типа `Line` и `Text` с указанными значениями выбранных свойств во всех

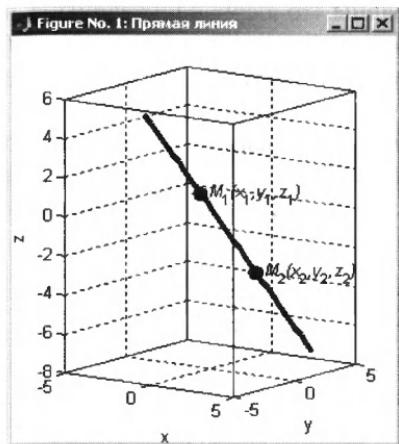


Рис. 5.26. Прямая линия, заданная с помощью двух точек

Таким образом, для получения значений свойств объектов следует использовать функцию `get()`, а для присваивания значений свойствам объектов – функцию `set()`.

5.2.5. Прямая и плоскость в пространстве. Основные задачи

Основными задачами аналитической геометрии при рассмотрении точек, прямых линий и плоскостей в пространстве являются определение угла пересечения между прямыми линиями или между плоскостью и прямой линией, вычисление расстояния от точки до прямой линии или плоскости, а также определение взаимного расположения прямых линий в пространстве. Рассмотрим решение этих задач с использованием системы MATLAB.

5.2.5.1. Определение угла между двумя плоскостями

Под углом пересечения между двумя плоскостями Q_1 и Q_2 понимается один из двугранных углов, образованных этими плоскостями.

Пусть заданы две плоскости с помощью общих уравнений:

$$A_1x + B_1y + C_1z + D_1 = 0, \quad A_2x + B_2y + C_2z + D_2 = 0.$$

Угол φ между нормальными векторами $\vec{n}_1 = (A_1; B_1; C_1)$ и $\vec{n}_2 = (A_2; B_2; C_2)$ плоскостей Q_1 и Q_2 равен одному из этих углов. Используя скалярное произведение двух векторов, найдем угол пересечения между плоскостями:

$$\cos \varphi = \frac{\vec{n}_1 \cdot \vec{n}_2}{|\vec{n}_1| \cdot |\vec{n}_2|} = \frac{A_1A_2 + B_1B_2 + C_1C_2}{\sqrt{A_1^2 + B_1^2 + C_1^2} \cdot \sqrt{A_2^2 + B_2^2 + C_2^2}}.$$

Если плоскости Q_1 и Q_2 перпендикулярны, то перпендикулярны и их нормали, т.е. $\vec{n}_1 \perp \vec{n}_2$. Следовательно, $\vec{n}_1 \cdot \vec{n}_2 = 0$ или $A_1 A_2 + B_1 B_2 + C_1 C_2 = 0$. Полученное равенство есть условие перпендикулярности двух плоскостей Q_1 и Q_2 .

Если плоскости Q_1 и Q_2 параллельны, то будут параллельны и их нормали. Следовательно, координаты нормальных векторов \vec{n}_1 и \vec{n}_2 будут пропорциональны: $A_1/A_2 = B_1/B_2 = C_1/C_2$. Это условие есть условие параллельности двух плоскостей Q_1 и Q_2 .

Пример 5.23. Найти угол пересечения между двумя плоскостями Q_1 и Q_2 , заданными в виде общих уравнений: $x - y - z + 1 = 0$ и $-x - 2y - 2z + 1 = 0$.

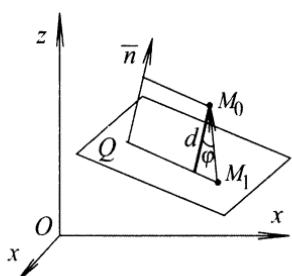
Решение:

```
>> A=[1;-1]; B=[-1;-2]; C=[-1;-2]; D=[1;1];
>> n1n2=A(1)*A(2)+B(1)*B(2)+C(1)*C(2);
>>
an1an2=sqrt(A(1)^2+B(1)^2+C(1)^2)*sqrt(A(2)^2+B(2)^2+C(2)^2);
>> phi=acos(n1n2/an1an2)*180/pi
>> phi =
54.7356
```

Угол между плоскостями Q_1 и Q_2 составляет 54.74° .

5.2.5.2. Вычисление расстояния от точки до плоскости

Расстояние d от точки $M_0(x_0; y_0; z_0)$ до плоскости Q равно модулю проекции вектора $\overline{M_1 M_0}$, где $M_1(x_1; y_1; z_1)$ – произвольная точка плоскости Q , на направление нормального вектора $\vec{n} = (A; B; C)$ (рис. 5.27). Таким образом,



$$\begin{aligned} d &= |\overline{M_1 M_0}| \cdot \cos(\varphi) = \frac{|\overline{M_1 M_0}|}{|\overline{M_1 M_0}|} \cdot \frac{\overline{M_1 M_0} \cdot \vec{n}}{|\overline{M_1 M_0}| \cdot |\vec{n}|} = \\ &= \frac{\overline{M_1 M_0} \cdot \vec{n}}{|\vec{n}|} = \frac{|(x_0 - x_1)A + (y_0 - y_1)B + (z_0 - z_1)C|}{\sqrt{A^2 + B^2 + C^2}} = \\ &= \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}}. \end{aligned}$$

Рис. 5.27. Расстояние от точки до плоскости

Так как точка M_1 располагается на плоскости Q , то $Ax_1 + By_1 + Cz_1 + D = 0 \Rightarrow D = -Ax_1 - By_1 - Cz_1$.

Пример 5.24. Найти среди двух точек $M_1(1;1.2;1)$ и $M_2(2;1.75;0)$ точку, которая ближе располагается к плоскости Q , заданной с помощью общего уравнения $x + y + z + 1 = 0$. Изобразить в координатном пространстве координатные оси в виде линий синего цвета толщиной 2 пункта, плоскость

в виде сетки и две точки в виде круговых маркеров синего цвета вместе с текстовыми обозначениями $M_1(x_1; y_1; z_1)$ и $M_2(x_2; y_2; z_2)$ соответственно. Обозначить координатные оси, сделать одинаковый масштаб по всем осям, вывести координатную сетку и охватить координатное пространство параллелепипедом. Установить точку обзора в позицию с координатами (68;8). Значения осей x и y лежат в диапазоне $[-2;3]$ и отстоят друг от друга на расстоянии 0.2.

Решение:

```
>> n=[1;1;1];D=[1];M=[[1;1.2;1],[2;1.75;0]]; % исходные данные
>> % Построение плоскости и точек в координатном пространстве
>> x=[-2:0.2:3];y=x;[X,Y]=meshgrid(x,y); Z=(-n(1)*X-n(2)*Y-D)/n(3);
>> surface(X,Y,Z,'FaceColor',[0.8 0.8 0.8]);
>> xlabel('x'), ylabel('y'), zlabel('z')
>> view(68,8), grid on
>> box on, axis equal, hold on
>> line([-2 3], [0 0], [0 0], 'LineWidth',2) % ось x
>> line([0 0], [-2 3], [0 0], 'LineWidth',2) % ось y
>> line([0 0],[0 0],[min(min(Z)),max(max(Z))], 'LineWidth',2) % ось z
>> line(M(1),M(2),M(3),'Marker','o'); % 1-я точка
>> text(M(1),M(2),M(3),'\it M_{\rm 1}(x_1; y_1; z_1)')
>> line(M(4),M(5),M(6),'Marker','o'); % 2-я точка
>> text(M(4),M(5),M(6),'\it M_{\rm 2}(x_2; y_2; z_2)')
>> % Определение расстояния между точками и плоскостью
>> d=abs(M'*n+ones(size(M',1),1)*D)./
norm(n)
d =
    2.4249
    2.7424
>> % минимальное расстояние
>> min(d)
ans =
    2.4249
```

Расстояние от точки M_1 до плоскости Q равно 2.4249, а от точки M_2 до плоскости Q – 2.7424. Таким образом, точка M_1 располагается к плоскости Q ближе, чем точка M_2 . Результаты построения плоскости Q и точек M_1 и M_2 в координатном пространстве представлены на рис. 5.28.

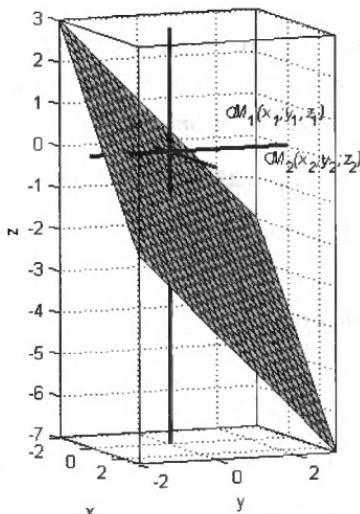


Рис. 5.28. Плоскость и точки

5.2.5.3. Определение угла между прямыми линиями

Под углом между прямыми линиями L_1 и L_2 понимается угол между направляющими векторами этих прямых $\bar{V}_1(m_1; n_1; p_1)$ и $\bar{V}_2(m_2; n_2; p_2)$.

Пусть заданы две прямые линии L_1 и L_2 . Первая задана каноническими уравнениями $(x - x_1)/m_1 = (y - y_1)/n_1 = (z - z_1)/p_1$, а вторая – с помощью общих уравнений прямой линии:

$$\begin{cases} A_1x + B_1y + C_1z + D_1 = 0, \\ A_2x + B_2y + C_2z + D_2 = 0. \end{cases}$$

Направляющий вектор линии, которая задана общими уравнениями, определяется с помощью векторного произведения нормалей поверхностей, ее образующих. Следовательно, $\bar{V}_2 = (m_2; n_2; p_2) = (A_1; B_1; C_1) \times (A_2; B_2; C_2)$.

Угол между направляющими двух прямых линий определяется с помощью следующей формулы:

$$\cos \varphi = \frac{\bar{V}_1 \cdot \bar{V}_2}{|\bar{V}_1| \cdot |\bar{V}_2|} = \frac{m_1 m_2 + n_1 n_2 + p_1 p_2}{\sqrt{m_1^2 + n_1^2 + p_1^2} \cdot \sqrt{m_2^2 + n_2^2 + p_2^2}}.$$

Если прямые линии L_1 и L_2 перпендикулярны, то скалярное произведение их направляющих векторов \bar{V}_1 и \bar{V}_2 равно нулю, т.е. $m_1 m_2 + n_1 n_2 + p_1 p_2 = 0 \Rightarrow \cos \varphi = 0$.

Если прямые линии L_1 и L_2 параллельны, то параллельны их направляющие векторы \bar{V}_1 и \bar{V}_2 . Следовательно, координаты этих векторов пропорциональны, т.е. $m_1/m_2 = n_1/n_2 = p_1/p_2$.

Пример 5.25. Найти угол между прямыми линиями L_1 и L_2 :

$$\frac{x}{2} = \frac{y-2}{-1} = \frac{z+2}{3} \quad \text{и} \quad \begin{cases} 2x + y - z - 1 = 0, \\ 2x - y + 3z + 5 = 0. \end{cases}$$

Решение:

```
>> V=[2;-1;3]; n=[[2;1;-1],[2;-1;3]];
>> V(:,2) = cross(n(:,1),n(:,2)); % направляющий вектор линии L2
>> phi=acos(V(:,1)'*V(:,2)/(norm(V(:,1))*norm(V(:,2))))*180/pi
phi =
```

90

Следовательно, прямые линии L_1 и L_2 перпендикулярны.

5.2.5.4. Определение взаимного расположения прямых линий в пространстве

Прямые линии L_1 и L_2 могут располагаться в пространстве на одной плоскости Q . Пусть прямая линия L_1 проходит через точку $M_1(x_1; y_1; z_1)$ параллельно направляющему вектору $\bar{V}_1 = (m_1; n_1; p_1)$, а прямая линия L_2 – через точку $M_2(x_2; y_2; z_2)$ параллельно направляющему вектору $\bar{V}_2 = (m_2; n_2; p_2)$. Для определения принадлежности плоскости Q линий L_1 и L_2 необходимо, чтобы смешанное произведение трех векторов \bar{V}_1 , \bar{V}_2 и $\overline{M_2 M_1}$ было равно нулю:

$$\begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ m_1 & n_1 & p_1 \\ m_2 & n_2 & p_2 \end{vmatrix} = 0.$$

При выполнении этого условия прямые линии L_1 и L_2 лежат в одной плоскости Q , т.е. пересекаются, если $\bar{V}_1 \neq \lambda \cdot \bar{V}_2$, либо параллельны $\bar{V}_1 \parallel \bar{V}_2$.

Пример 5.26. Определить взаимное расположение прямых линий L_1 и L_2 , проходящих через точки $M_1(x_1; y_1; z_1)$ и $M_2(x_2; y_2; z_2)$ параллельно направляющим векторам $\bar{V}_1 = (m_1; n_1; p_1)$ и $\bar{V}_2 = (m_2; n_2; p_2)$ соответственно.

Решение:

```
>> V=[1,-2,1;-1,2,-1]; % направляющие векторы линий V1 и V2
>> M=[2,-1,-1; 2,1,3]; % координаты точек M1 и M2
>> V=[M(2,:)-M(1,:); V] % матрица векторов.
V =
    0      2      4
    1     -2      1
   -1      2     -1
>> det(V)
ans =
    0
```

Следовательно, прямые линии L_1 и L_2 лежат в одной плоскости.

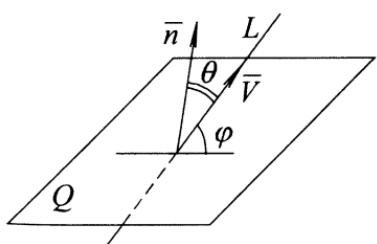


Рис. 5.29. Угол между прямой и плоскостью

5.2.5.5. Вычисление угла между прямой линией и плоскостью

Углом между прямой линией L и плоскостью Q называется любой из двух смежных углов, образованных прямой линией L и ее проекцией на плоскость Q (рис. 5.29). Пусть плоскость Q задана общим уравнением

$$Ax + By + Cz + D = 0,$$

а прямая L – каноническими уравнениями

$$\frac{(x - x_1)}{m_1} = \frac{(y - y_1)}{n_1} = \frac{(z - z_1)}{p_1}.$$

Обозначим через φ угол между плоскостью Q и прямой линией L , а через θ – угол между векторами $\bar{n} = (A; B; C)$ и $\bar{V} = (m_1; n_1; p_1)$. Так как $\varphi \leq \pi/2$, то

$$\sin \varphi = \sin \left(\frac{\pi}{2} - \theta \right) = \cos \theta = \frac{|Am + Bn + Cp|}{\sqrt{A^2 + B^2 + C^2} \cdot \sqrt{m^2 + n^2 + p^2}}.$$

Если прямая линия L параллельна плоскости Q , то векторы \bar{n} и \bar{V} перпендикулярны, т.е. $Am + Bn + Cp = 0$.

Если прямая линия L перпендикулярна плоскости Q , то векторы \bar{n} и \bar{V} параллельны, т.е. $A/m = B/n = C/p$.

Пример 5.27. Вычислить угол между прямыми линиями L_1 , L_2 и L_3 и плоскостью Q . Направляющий вектор прямой линии $L_1 - \bar{V}_1 = (1; -1; 0.5)$, линии $L_2 - \bar{V}_2 = (2; -1; 3)$ и линии $L_3 - \bar{V}_3 = (1; -4; -2)$. Поверхность Q задана общим уравнением $2x - 8y - 4z + 1 = 0$.

Решение:

```
>> V=[ [1;-1;0.5], [2;-1;3], [1;-4;-2] ]; % направляющие векторы
>> n=[2;-8;-4]; % координаты нормали поверхности
>> N = n*ones(1, size(V, 2)); % матрица координат нормали
>> costheta = abs(n'*V) ./ sqrt(sum(N.^2)) ./ sqrt(sum(V.^2));
>> % вычисление угла между прямой и плоскостью
>> theta = acos(costheta)*180/pi
theta =
    54.4147    90.0000         0
```

Таким образом, линия L_1 пересекает плоскость Q под углом 54.41° . Линия L_2 параллельна плоскости Q , так как угол между нормалью \bar{n} к плоскости Q и направляющим вектором \bar{V}_2 линии L_2 равен 90° . Линия L_3 перпендикулярна плоскости Q , так как угол между нормалью \bar{n} к плоскости Q и направляющим вектором \bar{V}_3 линии L_3 равен 0° .

5.2.5.6. Определение точки пересечения прямой линии и плоскости

Пусть требуется найти точку пересечения прямой линии L , заданной параметрическими уравнениями

$$x = x_0 + tm, \quad y = y_0 + tn, \quad z = z_0 + tp,$$

с плоскостью Q , заданной с помощью общего уравнения $Ax + By + Cz + D = 0$. Для определения точки пересечения прямой линии L и плоскости Q необхо-

димо решить заданные уравнения. Подставив выражения для x , y и z в уравнение плоскости получим уравнение

$$t(Am + Bn + Cp) + (Ax_0 + By_0 + Cz_0 + D) = 0.$$

Если прямая линия не параллельна плоскости ($Am + Bn + Cp \neq 0$), то получаем выражение для параметра t :

$$t = -\frac{Ax_0 + By_0 + Cz_0 + D}{Am + Bn + Cp}.$$

Подставляя данное выражение в параметрические уравнения прямой линии L вместо параметра t , найдем координаты точки пересечения прямой линии L с плоскостью Q :

$$x = x_0 - m \frac{Ax_0 + By_0 + Cz_0 + D}{Am + Bn + Cp}, \quad y = y_0 - n \frac{Ax_0 + By_0 + Cz_0 + D}{Am + Bn + Cp},$$

$$z = z_0 - p \frac{Ax_0 + By_0 + Cz_0 + D}{Am + Bn + Cp}.$$

Если прямая линия L параллельна плоскости Q ($Am + Bn + Cp = 0$), то она может либо принадлежать плоскости Q , либо не принадлежать. В первом случае одновременно выполняются оба равенства: $Am + Bn + Cp = 0$ и $Ax_0 + By_0 + Cz_0 + D = 0$, которые называются *условием принадлежности прямой*. Во втором случае $Ax_0 + By_0 + Cz_0 + D \neq 0$, следовательно, прямая линия L не принадлежит плоскости Q .

Пример 5.28. Определить координаты точек пересечения трех прямых линий L_1 , L_2 и L_3 с плоскостью Q . Линии заданы с помощью параметрических уравнений

$$L_1 : \begin{cases} x = 5 + t, \\ y = 1 - t, \\ z = -3 + 0.5 \cdot t, \end{cases} \quad L_2 : \begin{cases} x = -0.5 + 2 \cdot t, \\ y = -t, \\ z = 3 \cdot t, \end{cases} \quad L_3 : \begin{cases} x = 2 + 2 \cdot t, \\ y = 1 - t, \\ z = -2 + 3 \cdot t. \end{cases}$$

Плоскость Q задана общим уравнением $2x - 8y - 4z + 1 = 0$. Результат вывести в матричном виде.

Решение:

```
>> V=[1;-1;0.5], [2;-1;3], [2;-1;3]]; % координаты направляющих векторов
>> n=[2;-8;-4]; D=1; %координаты нормали к поверхности и коэффициент D
>> % координаты трех точек, принадлежащих соответствующей линии
>> M=[[5;1;-3], [-0.5;0;0], [2;1;-2]]; % начальные данные
>> x = M(1,:)-V(1,:).*(n'*M+D)./(n'*V); % значения по оси x
```

```

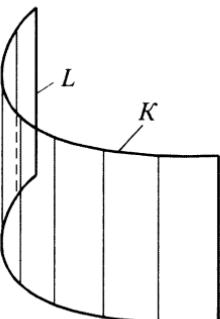
>> y = M(2,:) - V(2,:).* (n'*M+D)./(n'*V); % значения по оси у
>> z = M(3,:) - V(3,:).* (n'*M+D)./(n'*V); % z координаты
>> points = [x;y;z] % вывод полученных координат в матричном виде
points =
    3.1250      NaN      -Inf
    2.8750      NaN       Inf
   -3.9375      NaN      -Inf

```

Таким образом, линия L_1 пересекает плоскость Q в точке с координатами $(3.1250; 2.8750; -3.9375)$. Линия L_2 принадлежит поверхности Q , так как имеет место неопределенность $0/0$ (т.е. выполняются два условия), что в результате дает NaN. Линия L_3 параллельна плоскости Q , но не принадлежит ей, так как происходит деление на ноль (т.е. выполняется только одно условие), что в результате дает Inf или -Inf в зависимости от знака координаты направляющего вектора прямой линии. При выполнении этого примера можно отключить вывод предупреждающего сообщения о делении на ноль с помощью команды `>> warning off MATLAB:divideByZero`. После выполнения примера можно включить вывод предупреждающего сообщения с помощью команды `>> warning on MATLAB:divideByZero`.

5.2.6. Цилиндрические поверхности

Поверхность, образованная движением прямой L , которая перемещается в пространстве, сохраняя постоянное направление и пересекая каждый раз некоторую кривую K , называется *цилиндрической поверхностью* или *цилиндром* (рис. 5.30). При этом кривая K называется *направляющей цилиндра*, а прямая L – его *образующей*.



Если ось Oz прямоугольной системы координат параллельна образующей цилиндрической поверхности, то уравнение этой поверхности будет $F(x,y) = 0$. Уравнение $F(y,z) = 0$ есть уравнение цилиндра с образующими, параллельными оси Ox , а уравнение $F(x,z) = 0$ – с образующими, параллельными оси Oy .

Рис. 5.30. Цилиндрическая поверхность

Название цилиндра определяется названием направляющей. Если направляющей служит эллипс, то цилиндрическая поверхность называется *эллиптическим цилиндром*. Частным случаем эллиптического цилиндра является круговой цилиндр. Если направляющая представляет собой гиперболу, то цилиндрическая поверхность – гиперболический цилиндр. *Параболическим цилиндром* называется цилиндрическая поверхность, направляющей которой является парабола.

П р и м е р 5.29. Создать в одном графическом окне два координатных пространства. В первом координатном пространстве построить две цилиндрические поверхности. Первая поверхность является эллиптическим цилиндром, образующая которого параллельна оси Oz . Длина образующей – $L = 3$. Координаты центра эллипса – $(1;1)$. Параметр t параметрического уравнения эллипса изменяется от 0 до $3\pi/2$ с шагом $\pi/16$. Большая полуось эллипса – $a = 1.5$, а малая полуось – $b = 1$. Вторая поверхность представляет собой круговой цилиндр, направляющая которого лежит в плоскости Oyz . Радиус окружности $r = 0.5$. Длина образующей – $L = 2$. Для кругового цилиндра задать темно-зеленый цвет, а для эллиптического цилиндра – плавный переход цветов палитры jet. Во втором координатном пространстве построить две цилиндрические поверхности: гиперболический и параболический цилиндры, направляющие которых лежат в плоскости Oxy . Параметр t параметрических уравнений гиперболы изменяется от $-\pi/2$ до $\pi/2$ с шагом $\pi/16$. Действительная полуось – $b = 1$, мнимая полуось – $a = 0.5$. Длина образующей для обоих цилиндров – $L = 3$. Фокус параболы расположен на оси Ox . Вершина параболы находится в точке с координатами $(1;0)$. Координата параболы u изменяется от -2 до 2 с шагом 0.1. Для обеих цилиндрических поверхностей задать плавный переход цветов палитры jet. Направляющие всех цилиндрических поверхностей пересекают их образующие посередине.

Р е ш е н и е:

```
>> figure('NumberTitle','off','Name','Цилиндры второго порядка',...
' Renderer','opengl')
>> % образующие эллиптического цилиндра параллельны оси Oz
>> t = 0:pi/16:3*pi/2; a=1.5; b=1; x0=1; y0=1;
>> x=x0+a*cos(t); y=y0+b*sin(t);
>> height=3; step=10; delta=[-height/2:height/step:height/2];
>> X=ones(size(delta,1),1)*x; Y=ones(size(delta,1),1)*y;
>> z=height/2*ones(1,size(x,2)); Z=delta*z;
>> subplot(1,2,1), surf(X,Y,Z), shading interp, hold on
>> % образующие кругового цилиндра параллельны оси Ox
>> t=0:pi/16:2*pi; a=0.5; b=0.5; y0=1; z0=0;
>> y=y0+a*cos(t); z=z0+b*sin(t);
>> height=2; delta=[-height/2:height/step:height/2];
>> Y=ones(size(delta,1),1)*y; Z=ones(size(delta,1),1)*z;
>> x=height/2*ones(1,size(y,2)); X=delta*x;
>> surf(X,Y,Z,'FaceColor',[0 0.5 0])
>> xlabel('x'), ylabel('y'), zlabel('z'), axis equal, box on
>> title('\it Elliptical and circular cylinders')
>> % построение гиперболических цилиндров
>> t=-pi/2:pi/16:pi/2; a=0.5; b=1;
>> x_r=a*sinh(t); y_r=b*cosh(t); x_l=-a*sinh(t); y_l=-b*cosh(t);
```

```

>> height=3; step=10; delta=[-height/2:height/step:height/2]';
>> X_r=ones(size(delta,1),1)*x_r; Y_r=ones(size(delta,1),1)*y_r;
>> X_l=ones(size(delta,1),1)*x_l; Y_l=ones(size(delta,1),1)*y_l;
>> z=height/2*ones(1,size(x_r,2)); Z=delta*z;
>> subplot(1,2,2), surf(X_r,Y_r,Z), hold on, surf(X_l,Y_l,Z)
>> % построение параболического цилиндра
>> y0=0; x0=1; p=2; y=-2:0.1:2; x=(y-y0).^2/p + x0;
>> X=ones(size(delta,1),1)*x; Y=ones(size(delta,1),1)*y;
>> z=height/2*ones(1,size(x,2)); Z=delta*z;
>> surf(X,Y,Z), xlabel('x'), ylabel('y'), zlabel('z')
>> title('itHyperbolic and parabolic cylinders')
>> shading interp, axis equal, box on

```

Результаты вышеприведенных команд представлены на ил. 7.

5.2.7. Поверхности вращения

Поверхность, которая образуется вращением некоторой плоской кривой вокруг оси, лежащей в ее плоскости, называется *поверхностью вращения*. Пусть некоторая кривая L лежит в плоскости Oyz (рис. 5.31). Уравнения этой кривой будут записаны в виде

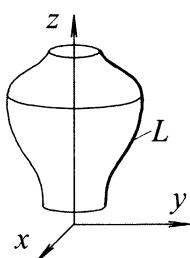


Рис. 5.31. Поверхность вращения

$$\begin{cases} F(y, z) = 0, \\ x = 0. \end{cases}$$

Уравнение поверхности, которая образуется вращением кривой L

$$\text{вокруг оси } Oz: F\left(\pm\sqrt{x^2 + y^2}, z\right) = 0,$$

$$\text{вокруг оси } Oy: F\left(\pm\sqrt{x^2 + z^2}, y\right) = 0.$$

Соответствующие параметрические уравнения поверхностей, которые образуются вращением кривой линии, заданной в плоскости Oyz , вокруг осей Oz и Oy , имеют вид

$$F(y \cdot \sin t, y \cdot \cos t, z) = 0 \text{ и } F(z \cdot \sin t, y, z \cdot \cos t) = 0,$$

где t – параметр уравнения, который изменяется в диапазоне $[0; 2\pi]$.

Если кривая лежит в плоскости Oxy ($z = 0$) и ее уравнение $F(x, y) = 0$, то уравнения поверхности вращения, образованной вращением кривой линии L вокруг оси Ox : $F\left(x, \pm\sqrt{y^2 + z^2}\right) = 0$ или $F\left(x, y \cdot \cos t, y \cdot \sin t\right) = 0$,

вокруг оси Oy : $F\left(\pm\sqrt{x^2 + z^2}, y\right) = 0$ или $F\left(x \cdot \cos t, y, x \cdot \sin t\right) = 0$.

Если кривая лежит в плоскости Oxz ($y = 0$) и ее уравнение $F(x,z) = 0$, то уравнения поверхности вращения, образованной вращением кривой линии L вокруг оси Ox : $F\left(x, \pm\sqrt{y^2 + z^2}\right) = 0$ или $F\left(x, z \cdot \sin t, z \cdot \cos t\right) = 0$,
вокруг оси Oz : $F\left(\pm\sqrt{x^2 + y^2}, z\right) = 0$ или $F\left(x \cdot \cos t, x \cdot \sin t, z\right) = 0$.

Кривая линия L в координатном пространстве может быть задана не только аналитическим уравнением, но и с помощью таблицы. Рассмотрим пример построения поверхностей вращения на основе кривых линий, заданных таблично.

Пример 5.30. Построить в отдельных координатных пространствах одногого графического окна поверхности вращения, образующие линии которых заданы с помощью таблиц. Первая кривая линия задана с помощью табл. 5.10, вторая линия – с помощью табл. 5.11, третья линия – с помощью табл. 5.12 и четвертая линия – табл. 5.13. При построении первой поверхности осью вращения является ось y , второй и третьей поверхностей – ось z , четвертой поверхности – ось x .

Таблица 5.10

Координаты образующей линии первой поверхности вращения

x	0	1	2	3	4	5	5	4	3	2	1	0
y	0	0.2	0.4	0.75	0.9	1	20	20.1	20.25	20.6	20.8	21

Таблица 5.11

Координаты образующей линии второй поверхности вращения

y	0	2	2.5	3.5	5
z	0	0	1	2	2.5

Таблица 5.12

Координаты образующей линии третьей поверхности вращения

y	3	1.5	0.5	0.5	2.2	3.1	4	5
z	1	1.5	3	12	14	16	18	19

Таблица 5.13

Координаты образующей линии четвертой поверхности вращения

z	3	1.5	0.5	0.5	2.2	3.1	4	5
x	1	1.5	3	12	14	16	18	19

Решение:

```

>> t=(0:pi/16:2*pi)'; % изменение параметра
>> figure('Name', 'Поверхности вращения', 'Renderer', 'opengl')
>> colormap(gray)
>> % построение первой поверхности
>> x=[0 1 2 3 4 5 5 4 3 2 1 0];
>> y=[0 0.2 0.4 0.75 0.9 1 2 20 20.1 20.25 20.6 20.8 21];
>> subplot(2,2,1)
>> surf(cos(t)*x,ones(size(t),1)*y,sin(t)*x,'FaceColor',[1,0.65,0.45])
>> axis equal, box on, set(gca,'Color','none')
>> xlabel('x'), ylabel('y'), zlabel('z')
>> % построение второй поверхности
>> y=[0 2 2.5 3.5 5]; z=[0 0 1 2 2.5];
>> subplot(2,2,2)
>> surf(cos(t)*y,sin(t)*y,ones(size(t),1)*z,'FaceColor',[0.95,0.95,0.95])
>> axis equal, box on, view(-35,30), set(gca,'Color','none')
>> xlabel('x'), ylabel('y'), zlabel('z')
>> % построение третьей поверхности
>> y=[3 1.5 0.5 0.5 0.5 2.2 3.1 4 5]; z=[1 1.5 3 9 12 14 16 18 19];
>> subplot(2,2,3)
>> surf(cos(t)*y,sin(t)*y,ones(size(t),1)*z)
>> axis off, axis equal, shading interp
>> % построение четвертой поверхности
>> z=[3 1.5 0.5 0.5 0.5 2.2 3.1 4 5]; x=[1 1.5 3 9 12 14 16 18 19];
>> subplot(2,2,4)
>> surf(ones(size(t),1)*x,cos(t)*z,sin(t)*z)
>> axis equal, box on, set(gca,'Color','none')
>> xlabel('x'), ylabel('y'), zlabel('z')

```

Результаты выполнения команд представлены на ил. 8.

Для дальнейшего изучения поверхностей вращения следует просмотреть демонстрационный пример 3D Drawing, реализация которого располагается в файле makevase.m.

Эллиптический тор является еще одним примером поверхности вращения, которая описывается следующей системой параметрических функций:

$$\begin{cases} x(t, \tau) = (r + a \cdot \cos t) \cdot \cos \tau, \\ y(t, \tau) = (r + a \cdot \cos t) \cdot \sin \tau, \\ z(t, \tau) = b \cdot \sin t, \\ \forall t, \tau \in [0; 2\pi], r > a, r > b, \end{cases}$$

где r – радиус направляющей окружности; a, b – полуоси образующего эллипса.

Приведенная система параметрических функций описывает эллиптический тор как поверхность вращения, которая получается с помощью вращения эллипса, построенного в плоскости Oxy , вокруг оси z .

Пример 5.31. Построить эллиптический тор, у которого полуоси образующего эллипса – $a = 0.75$, $b = 1.5$ и радиус направляющей окружности – $r = 3$. Направляющая окружность лежит в плоскости Oxy и ось z совпадает с ее центром. Параметр φ , который определяет длину линии, образующей эллипс, изменяется в диапазоне $[\pi/8; 15\pi/8]$ с шагом $\pi/32$. Параметр θ , определяющий длину линии, вдоль которой происходит вращение эллипса, изменяется в диапазоне $[0; 3\pi/2]$ с шагом $\pi/32$.

Решение:

```
>> a = 0.75; b = 1.5; r = 3; % значения осей и радиус окружности
>> phi = pi/8:pi/32:2*pi-pi/8; % параметр эллипса
>> theta = [0:pi/36:3*pi/2]'; % параметр вращения
>> X = cos(theta)*(r + a*cos(phi)); % массив значений по оси x
>> Y = sin(theta)*(r + a*cos(phi)); % массив значений по оси y
>> Z = ones(size(theta,1),1)*b*sin(phi); % массив значений по оси z
>> % Создание графического окна
>> figure('NumberTitle','off','Name','Эллиптический тор')
>> colormap(colorcube) % задание цветовой палитры
>> Ax = axes; % Создание координатного пространства
>> % создание поверхности и ориентирование цветов палитры по оси x
>> T = surf(X,Y,Z,Y);
>> % Изменение свойств поверхности
>> set(T,'LineStyle','-','FaceColor','interp','EdgeAlpha',0.2)
>> % Отображение координатных осей
>> line([min(min(X)) max(max(X))], [0 0], [0 0], 'LineWidth', 1)
>> line([0 0], [min(min(Y)) max(max(Y))], [0 0], 'LineWidth', 1)
>> line([0 0], [0 0], [min(min(Z)) max(max(Z))], 'LineWidth', 1)
>> % Оформление координатного пространства
>> set(Ax,'Color','none','Box','on',...
'XGrid','on','YGrid','on','ZGrid','on',...
'DataAspectRatio',[1 1 1], 'View', [36,20])
>> xlabel('x'), ylabel('y'), zlabel('z')
>> tt = ['\ita\rm = ', num2str(a,3), '\itb\rm = ', num2str(b,3),...
', \itr\rm = ', num2str(r,3)];
>> title(tt)
>> colorbar % вывод шкалы цветовой палитры
```

Результаты вышеприведенных команд представлены на ил. 9.

5.2.8. Поверхности второго порядка

Поверхностью второго порядка называется поверхность, уравнение которой в прямоугольной системе координат является алгебраическим уравнением второй степени. При исследовании поверхностей второго порядка применяют метод сечений: исследование вида поверхности производится при помощи изучения линий пересечения данной поверхности с координатными плоскостями или плоскостями, им параллельными.

5.2.8.1. Эллипсоид

Эллипсоидом называется поверхность, которая в некоторой прямоугольной системе координат определяется *каноническим уравнением*

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1.$$

Рассмотрим сечения этой поверхности с плоскостями, параллельными плоскости Oxy . Уравнение таких плоскостей: $z = h$, где h – любое число.

Линия, получаемая в сечении, определяется системой уравнений

$$\begin{cases} \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 - \frac{h^2}{c^2}, \\ z = h. \end{cases} \quad (5.4)$$

Рассмотрим полученные уравнения:

- a) если $|h| > c$ и $c > 0$, то $x^2/a^2 + y^2/b^2 < 0$. Точек пересечения рассматриваемой поверхности с плоскостями $z = h$ не существует;
- б) если $|h| = 0$, т.е. $h = \pm c$, то $x^2/a^2 + y^2/b^2 = 0$. Линия пересечения вырождается в две точки с координатами $(0;0;c)$ и $(0;0;-c)$. Плоскости $z = c$ и $z = -c$ касаются данной поверхности;
- в) если $|h| < c$, то уравнения линии примут вид

$$\begin{cases} \frac{x^2}{\left(a\sqrt{1-\frac{h^2}{c^2}}\right)^2} + \frac{y^2}{\left(b\sqrt{1-\frac{h^2}{c^2}}\right)^2} = 1, \\ z = h. \end{cases}$$

Следовательно, линия пересечения представляет собой эллипс с полуосями

$$a_1 = a\sqrt{1 - \frac{h^2}{c^2}} \text{ и } b_1 = b\sqrt{1 - \frac{h^2}{c^2}}.$$

При этом, чем меньше $|h|$, тем больше полуоси a_1 и b_1 . При $h = 0$ они достигают своих наибольших значений: $a_1 = a$, $b_1 = b$.

Аналогичные результаты будут получены, если рассматривать сечения поверхности, являющейся эллипсоидом, с плоскостями $x = h$ и $y = h$.

Величины a , b и c называются *полуосями эллипса*. Если все они различны, то эллипсоид называется *трехосным*; если какие-либо две полуоси равны, то трехосный эллипсоид трансформируется в *эллипсоид вращения*; если $a = b = c$, то – в *сфера* $x^2 + y^2 + z^2 = R^2$.

При построении поверхностей второго порядка в системе MATLAB используют их параметрические уравнения.

Параметрические уравнения эллипсоида имеют вид

$$\begin{cases} x = x_0 + a \cos \theta \cos \varphi, \\ y = y_0 + b \sin \theta \cos \varphi, \\ z = z_0 + c \sin \varphi, \end{cases}$$

где θ – долгота, φ – широта (из сферической системы координат), x_0 , y_0 и z_0 – координаты центра эллипсоида (из прямоугольной системы координат).

Пример 5.32. Построить трехосный эллипсoid со значениями осей $a = 6$, $b = 3$ и $c = 4$, полупрозрачную плоскость Q , параллельную плоскости Oxy ($h = 2$ – высота плоскости Q относительно центра эллипсоида) и красную линию L , полученную пересечением эллипсоида и плоскости Q . Часть эллипсоида, которая расположена выше пересечения с плоскостью Q , сделать невидимой.

Решение:

```
>> % задание данных для построения трехосного эллипса
>> x0 = 3; y0 = 1; z0 = 1; a = 6; b = 3; c = 4;
>> phi = [-pi/2:pi/128:pi/4]'; % диапазон изменения широты
>> theta = [0:pi/16:2*pi]; % диапазон изменения долготы
>> h = 2; % высота сечения эллипса плоскостью Q || Oxy
>> % вычисление значений координат поверхности
>> x = x0 + a*cos(phi)*cos(theta); y = y0 + b*cos(phi)*sin(theta);
>> z = z0 + c*sin(phi)*ones(1,size(theta,2));
>> figure('Name','Эллипсoid','NumberTitle','off')
>> colormap(gray*0.7) % задание цветовой палитры
>> surf(x,y,z), shading interp % создание поверхности
>> hold on % включение режима наложения графиков
>> % вычисление значений координат линии
>> x=x0+a*cos(pi/6)*cos(theta); y = y0 + b*cos(pi/6)*sin(theta);
>> z = z0 + h*ones(1,size(theta,2));
>> line(x,y,z,'LineWidth',2,'Color','r') % создание линии
>> % задание плоскости сечения эллипса
>> x=-4:10; y=-4:6; [X,Y]=meshgrid(x,y); Z=z0+h*ones(size(X));
>> surf(X,Y,Z,'FaceColor','w','EdgeColor','none','FaceAlpha',0.65)
>> xlabel('x'), ylabel('y'), zlabel('z'), title('\it\bfEllipsoid')
>> set(gca,'Color','none'), axis equal, box on
```

Результаты выполнения команд представлены на ил. 10. Система MATLAB позволяет задавать прозрачность графических объектов Surface. При создании прозрачных объектов система MATLAB автоматически переключает

режим прорисовки графического окна на OpenGL. Значение прозрачности поверхности лежит в диапазоне $[0;1]$. Нулевое значение соответствует полностью прозрачным поверхностям, а единичное значение – полностью непрозрачным. Созданный графический объект Surface в приведенном примере имеет одну прозрачность по всей поверхности, которая равна 0.65. Для задания прозрачности, одинаковой по всей поверхности, использовалось свойство FaceAlpha, которому было присвоено соответствующее прозрачности значение.

5.2.8.2. Однополостный гиперболоид

Однополостным гиперболоидом называется поверхность, которая в некоторой прямоугольной системе координат определяется каноническим уравнением

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1. \quad (5.5)$$

Рассмотрим сечения этой поверхности с плоскостями, параллельными плоскости Oxy . Уравнение таких плоскостей: $z = h$, где h – любое число.

Линия, получаемая в сечении, определяется системой уравнений

$$\begin{cases} \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 + \frac{h^2}{c^2}, \\ z = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{\left(a\sqrt{1+\frac{h^2}{c^2}}\right)^2} + \frac{y^2}{\left(b\sqrt{1+\frac{h^2}{c^2}}\right)^2} = 1, \\ z = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{a_1^2} + \frac{y^2}{b_1^2} = 1, \\ z = h. \end{cases}$$

Полученные уравнения являются уравнениями эллипсов с полуосами

$$a_1 = a\sqrt{1 + \frac{h^2}{c^2}} \text{ и } b_1 = b\sqrt{1 + \frac{h^2}{c^2}}.$$

Полуоси a_1 и b_1 достигают своего наименьшего значения при $h = 0$: $a_1 = a$ и $b_1 = b$. При возрастании $|h|$ полуоси эллипса будут увеличиваться.

Если пересекать однополостный гиперболоид (4.5) плоскостями $x = h$ ($|h| < a$) или $y = h$ ($|h| < b$), то в сечении будут иметь место гиперболы

$$\begin{cases} \frac{x^2}{a^2} - \frac{z^2}{c^2} = 1 - \frac{h^2}{b^2}, \\ y = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{\left(a\sqrt{1-\frac{h^2}{b^2}}\right)^2} - \frac{z^2}{\left(c\sqrt{1-\frac{h^2}{b^2}}\right)^2} = 1, \\ y = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{a_1^2} - \frac{z^2}{c_1^2} = 1, \\ y = h \end{cases}$$

или

$$\begin{cases} \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1 - \frac{h^2}{a^2}, \\ x = h, \end{cases} \Rightarrow \begin{cases} \frac{y^2}{\left(b\sqrt{1-\frac{h^2}{a^2}}\right)^2} + \frac{z^2}{\left(c\sqrt{1-\frac{h^2}{a^2}}\right)^2} = 1, \\ x = h, \end{cases} \Rightarrow \begin{cases} \frac{y^2}{b_1^2} - \frac{z^2}{c_1^2} = 1, \\ x = h. \end{cases}$$

Величина c называется *мнимой осью*, а величины a и b – *действительными полуосами* однополостного гиперболоида (5.5).

Пример 5.33. Построить однополостный гиперболоид с мнимой осью $c = 1$, действительными полуосами $a = 1$ и $b = 0.75$. Высота однополостного гиперболоида – $h = 8$. Прозрачность гиперболоида варьируется по его высоте. Края гиперболоида [4;3] и [-4;-3] имеют одну и ту же прозрачность ($\text{alpha} = 0.3$). При движении к центру гиперболоида его прозрачность уменьшается, и в центре он полностью непрозрачный ($\text{alpha} = 1$). В качестве цветовой палитры выбрать оттенки медного цвета (`copper`). Построить линии, которые получаются при пересечении однополостного гиперболоида плоскостями, параллельными плоскости Oxy и расположенным относительно плоскости Oxy на расстоянии $l = -4$, $l = 0$ и $l = 4$, а также плоскостью Oxz .

Решение:

```
>> a = 1; b = 0.75; c = 1; % параметры гиперболоида
>> h = [-4:0.2:4]; % сечения гиперболоида
>> theta=(0:pi/16:2*pi)'; % долгота
>> a1=a*sqrt(1+h.^2./c.^2); b1=b*sqrt(1+h.^2./c.^2); % оси эллипса
>> % координаты узловых точек однополостного гиперболоида
>> X=cos(theta)*a1; Y=sin(theta)*b1; Z=ones(size(theta,1),1)*h;
>> % создание графического окна и задание его свойств
>> figure('NumberTitle','off','Name','Однополостный гиперболоид')
>> colormap(min(copper*1.2,1)) % определение цветовой палитры
>> % задание карты прозрачности
>> alphamap('vup'), alphamap('increase',0.3)
>> % построение однополостного гиперболоида
>> surf(X,Y,Z,'EdgeColor','none','FaceColor','interp',...
'AlphaData',Z,'FaceAlpha','interp')
>> alim([-3 3]) % изменение диапазона переменной прозрачности по оси z
>> hold on % режим наложения графиков
>> line(cos(theta)*a1(1),sin(theta)*b1(1),... нижний эллипс
>> ones(size(theta,1),1)*h(1),'Color','k','LineWidth',2)
>> line(cos(theta)*a1(round(size(a1,2)/2)),... центральный эллипс
sin(theta)*b1(round(size(a1,2)/2)),...
ones(size(theta,1),1)*h(round(size(b1,2)/2)),'Color','k','LineWidth',2)
>> line(cos(theta)*a1(end),sin(theta)*b1(end),... верхний эллипс
ones(size(theta,1),1)*h(end),'Color','k','LineWidth',2)
```

```
>> line(cos(pi/2)*a1,sin(pi/2)*b1,... правая ветвь гиперболы
h,'Color','k','LineWidth',2)
>> line(cos(3*pi/2)*a1,sin(3*pi/2)*b1,... левая ветвь гиперболы
h,'Color','k','LineWidth',2)
>> % оформление координатного пространства
>> axis equal, box on, set(gca,'Color','none')
>> xlabel('x'), ylabel('y'), zlabel('z'), title('a=1, b=0.75, c=1, h=[-4;4]')

```

Результаты приведенных выше команд представлены на ил. 11. Система MATLAB позволяет изменять прозрачность вдоль поверхности. Для изменения прозрачности используется *карта прозрачности* (*alphamap*), которая является свойством графического окна. По умолчанию карта прозрачности представляет собой массив размером 64×1 . Элементы этого массива изменяются линейно от нуля до единицы, т.е. от полной прозрачности до ее отсутствия. С помощью команды `>> alphamap('vup')` были изменены значения элементов массива карты прозрачности. После выполнения команды значения элементов с 1 по 32 увеличиваются от нуля до единицы, а затем с 33 до 64 уменьшаются от единицы до нуля. Команда `>> alphamap('increase',n)` увеличивает текущие значения элементов массива карты прозрачности на величину n , т.е. пропорционально уменьшает прозрачность поверхности на заданное значение. Если результирующее значение элемента массива карты прозрачности превышает единицу во время выполнения команды, то оно приравнивается к единице. Кarta прозрачности используется для отображения на нее значений свойства *AlphaData* графического объекта согласно значению свойства *AlphaDataMapping*. Свойство *AlphaDataMapping* может принимать три значения: *none*, *scaled* и *direct*. Значение *scaled* позволяет отображать массив значений свойства *AlphaData* на карту прозрачности, используя пределы изменения прозрачности, хранящиеся в свойстве координатного пространства *ALim*. После создания поверхности свойство *ALim* содержит минимальное и максимальное значения массива *Z*, т.е. -4 и 4 соответственно. Следовательно, первая строка массива *AlphaData* соответствует первому элементу массива карты прозрачности, а последняя строка – последнему элементу. Уменьшение диапазона изменения прозрачности до $[-3;3]$ (`>> clim([-3 3])`) приводит к тому, что значения свойства *AlphaData*, которые лежат вне диапазона трансформируются к ближайшему граничному значению карты прозрачности (в данном примере – к 0.3). Значение *direct* позволяет отображать массив значений свойства *AlphaData* на карту прозрачности, рассматривая их в качестве индексов элементов массива *alphamap*. Значение *none* используется по умолчанию и позволяет задавать прозрачность поверхности с помощью значений свойства *AlphaData*, не используя карту прозрачности. Значения должны располагаться в диапазоне $[0;1]$. Задание свойству *FaceAlpha* значения *interp* приводит к плавному изменению прозрачности между узловыми точками поверхности.

5.2.8.3. Двуполостный гиперболоид

Двуполостным гиперболоидом называется поверхность, которая в некоторой прямоугольной системе координат определяется каноническим уравнением

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = -1. \quad (5.6)$$

Рассмотрим сечения этой поверхности с плоскостями, параллельными плоскости Oxy . Уравнение таких плоскостей: $z = h$, где h – любое число.

Линия, получаемая в сечении, определяется системой уравнений

$$\begin{cases} \frac{x^2}{a^2} + \frac{y^2}{b^2} = \frac{h^2}{c^2} - 1, \\ z = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{\left(a\sqrt{\frac{h^2}{c^2}-1}\right)^2} + \frac{y^2}{\left(b\sqrt{\frac{h^2}{c^2}-1}\right)^2} = 1, \\ z = h, \end{cases} \Rightarrow \begin{cases} \frac{x^2}{a_1^2} + \frac{y^2}{b_1^2} = 1, \\ z = h. \end{cases}$$

Рассмотрим частные случаи полученных уравнений:

- если $|h| < c$, то плоскости $z = h$ не пересекают двуполостный гиперболоид;
- если $|h| = c$, то плоскости $z = \pm c$ касаются двуполостного гиперболоида соответственно в точках $(0;0;c)$ и $(0;0;-c)$
- если $|h| > c$, то уравнения являются уравнениями эллипсов с полуосами

$$a_1 = a\sqrt{\frac{h^2}{c^2}-1} \text{ и } b_1 = b\sqrt{\frac{h^2}{c^2}-1}.$$

При этом полуоси эллипсов возрастают с увеличением значения h .

В том случае, если пересекать поверхность (5.6) координатными плоскостями Oyz ($x = 0$) и Oxz ($y = 0$), то в сечении получаются гиперболы, уравнения которых соответственно имеют вид

$$\frac{y^2}{b^2} - \frac{z^2}{c^2} = -1 \text{ и } \frac{x^2}{a^2} - \frac{z^2}{c^2} = -1.$$

У обеих гипербол ось Oz является *действительной осью*.

Пример 5.34. Построить в одном координатном пространстве три двуполостных гиперболоида, действительные оси которых z , y и x соответственно. Значения величин $a = 1$, $b = 1.2$ и $c = 1.1$ для всех гиперболоидов одинаковые.

Решение:

```
>> a=1; b=1.2; c=1.1; % параметры гиперболоидов
>> h_1 = [-3-c:0.1:-c, c:0.1:c+3]; % сечения 1-го гиперболоида
>> h_2 = [-3-a:0.1:-a, a:0.1:a+3]; % сечения 2-го гиперболоида
>> h_3 = [-3-b:0.1:-b, b:0.1:b+3]; % сечения 3-го гиперболоида
>> theta = [0:pi/16:2*pi]'; % долгота
```

```

>> % полуоси эллипсов
>> a1_1 = a*sqrt(h_1.^2./c.^2-1); b1_1 = b*sqrt(h_1.^2./c.^2-1);
>> b1_2 = b*sqrt(h_2.^2./a.^2-1); c1_2 = c*sqrt(h_2.^2./a.^2-1);
>> a1_3 = a*sqrt(h_3.^2./b.^2-1); c1_3 = c*sqrt(h_3.^2./b.^2-1);
>> % значения узловых точек двуполостных гиперболоидов
>> X_1=cos(theta)*a1_1; Y_1=sin(theta)*b1_1; Z_1=ones(size(theta),1)*h_1;
>> X_2=ones(size(theta),1)*h_2; Y_2=cos(theta)*b1_2; Z_2=sin(theta)*c1_2;
>> X_3=cos(theta)*a1_3; Y_3=ones(size(theta),1)*h_3; Z_3=sin(theta)*c1_3;
>> % создание графического окна и задания карты прозрачности
>> figure('NumberTitle','off','Name','Однополостный гиперболоид')
>> alphamap('vup'), alphamap('decrease',-0.3)
>> % построение гиперболоидов
>> surf(X_1,Y_1,Z_1,'EdgeColor','none','FaceColor','r',...
'AlphaData',Z_1,'FaceAlpha','interp'), hold on
>> surf(X_2,Y_2,Z_2,'EdgeColor','none','FaceColor','b',...
'AlphaData',X_2,'FaceAlpha','interp')
>> surf(X_3,Y_3,Z_3,'EdgeColor','none','FaceColor','g',...
'AlphaData',Y_3,'FaceAlpha','interp'), alim([-4.5;4.5])
>> % построение эллипсов
>> line(X_1(:,1),Y_1(:,1),Z_1(:,1),'Color','k','LineWidth',1.5);
>> line(X_1(:,end),Y_1(:,end),Z_1(:,end),'Color','k','LineWidth',1.5);
>> line(X_2(:,1),Y_2(:,1),Z_2(:,1),'Color','k','LineWidth',1.5);
>> line(X_2(:,end),Y_2(:,end),Z_2(:,end),'Color','k','LineWidth',1.5);
>> line(X_3(:,1),Y_3(:,1),Z_3(:,1),'Color','k','LineWidth',1.5);
>> line(X_3(:,end),Y_3(:,end),Z_3(:,end),'Color','k','LineWidth',1.5);
>> % оформление графика
>> xlabel('x'), ylabel('y'), zlabel('z')
>> axis equal, box on, set(gca,'Color','none'), view(-35,22)

```

Результаты приведенных выше команд представлены на ил. 12. Гиперболоиды окрашены в разные цвета: первый – в красный, второй – в синий и третий – в зеленый цвет. Значения элементов в карте прозрачности сначала увеличиваются от 0 до 0.7, а затем уменьшаются от 0.7 до 0. Следовательно, чем дальше от центра координатного пространства находится точка, принадлежащая гиперболоидам, тем большую прозрачность она имеет. Диапазон изменения прозрачности всех гиперболоидов равен [−4.5; 4.5].

5.2.8.4. Эллиптический параболоид

Эллиптическим параболоидом называется поверхность, которая в некоторой прямоугольной системе координат определяется каноническим уравнением

$$\frac{x^2}{p} + \frac{y^2}{q} = 2z, \quad (5.7)$$

где $p > 0$ и $q > 0$.

Рассечем эту поверхность плоскостями, которые параллельны плоскости Oxy ($z = h$, где h – любое число).

Линия, получаемая в сечении, определяется системой уравнений

$$\begin{cases} \frac{x^2}{p} + \frac{y^2}{q} = 2h, \\ z = h. \end{cases}$$

Рассмотрим частные случаи полученных уравнений:

- a) если $h < 0$, то плоскости $z = -h$ не пересекают поверхность;
- б) если $h = 0$, то плоскость $z = 0$ касается поверхности в точке $(0;0;0)$;
- в) если $h > 0$, то плоскости $z = h$ пересекают поверхность и в сечении имеет место эллипс, уравнение которого

$$\begin{cases} \frac{x^2}{2ph} + \frac{y^2}{2ph} = 1, \\ z = h. \end{cases}$$

Полуоси этого эллипса возрастают с увеличением значения h .

При пересечении поверхности (5.7) координатными плоскостями Oxz и Oyz получаются соответственно параболы $z = x^2/2p$ и $z = y^2/2q$.

Таким образом, поверхность, представляющая собой эллиптический параболоид, имеет вид бесконечно расширяющейся чаши.

Пример 5.35. Построить эллиптический параболоид с величинами $p = 1$ и $q = 3$. Высота параболоида равна 2.

Решение:

```
>> p = 1; q = 3; % параметры эллиптического параболоида
>> h = [0:0.2:2]; % сечения эллиптического параболоида
>> a = sqrt(2*h*p); b = sqrt(2*h*q); % полуоси эллипсов
>> theta = [0:pi/16:2*pi]'; % долгота
>> % значения координатных узлов эллиптического параболоида
>> X = cos(theta)*a; Y = sin(theta)*b; Z = ones(size(theta),1)*h;
>> % создание графического окна
>> figure('NumberTitle','off','Name','Эллиптический параболоид')
>> colormap(winter) % задание цветовой палитры
>> % построение поверхности
>> surf(X,Y,Z,'FaceColor','none','AlphaData',Z,...
'EdgeColor','interp','EdgeAlpha','interp','LineWidth',2)
>> % оформление графика
>> axis equal, box on, set(gca,'Color','none'), view(-36,23)
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('\itp \rm= 1; \itq \rm= 3; \itz \rm= [0;2]')
```

Результаты вышеприведенных команд представлены на ил. 13. Изменяя значения p и q , можно получать эллиптические параболоиды различной формы.

5.2.8.5. Гиперболический параболоид

Гиперболическим параболоидом называется поверхность, которая в некоторой прямоугольной системе координат определяется каноническим уравнением

$$\frac{x^2}{p} - \frac{y^2}{q} = 2z, \text{ где } p > 0 \text{ и } q > 0.$$

Рассечем эту поверхность плоскостями, которые параллельны плоскости Oxy ($z = h$, где h – любое число).

Линия, получаемая в сечении, определяется системой уравнений:

$$\begin{cases} \frac{x^2}{p} - \frac{y^2}{q} = 2h, \\ z = h. \end{cases}$$

Полученная система уравнений описывает гиперболу в координатном пространстве. Рассмотрим частные случаи этой системы уравнений

- если $h < 0$, то действительная ось гиперболы параллельна оси Oy ;
- если $h > 0$, то действительная ось гиперболы параллельна оси Ox ;
- если $h = 0$, то линия пересечения описывается следующим уравнением:

$$x^2/p - y^2/q = 0.$$

Следовательно, линия пересечения распадается на две пересекающиеся прямые, описываемые следующими уравнениями:

$$x/\sqrt{p} - y/\sqrt{q} = 0 \text{ и } x/\sqrt{p} + y/\sqrt{q} = 0.$$

Рассечем поверхность плоскостями, которые параллельны плоскости Oxz ($y = h$, где h – любое число).

Линии, получаемые в сечениях, являются параболами

$$\begin{cases} x^2 = 2p\left(z + \frac{h^2}{2q}\right), \\ y = h. \end{cases}$$

Ветви этих парабол направлены вверх. При $y = 0$ в сечении получается парабола с вершиной в начале координат и осью симметрии Oz .

Рассечем гиперболический параболоид плоскостями, которые параллельны плоскости Oyz ($x = h$, где h – любое число).

Линии, которые получаются в сечениях, представляют собой параболы

$$\begin{cases} y^2 = -2p \left(z - \frac{h^2}{2q} \right), \\ x = h. \end{cases}$$

Ветви этих парабол направлены вниз. При $x = 0$ в сечении получается парабола с вершиной в начале координат и осью симметрии Oz .

Пример 5.36. Построить в одном координатном пространстве два гиперболических параболоида: первый – с помощью параметрических уравнений гиперболы, второй – с помощью канонического уравнения. Величины $p = 1$, $q = 2$, а z изменяется в диапазоне $[-10; 10]$. Параметр t при построении ветвей двух гипербол изменяется в диапазоне $[-2; 2]$. В случае использования при построении гиперболического параболоида его канонического уравнения значения x и y располагаются в диапазоне $[-16; 16]$. Построить линию пересечения двух частей гиперболического параболоида.

Решение:

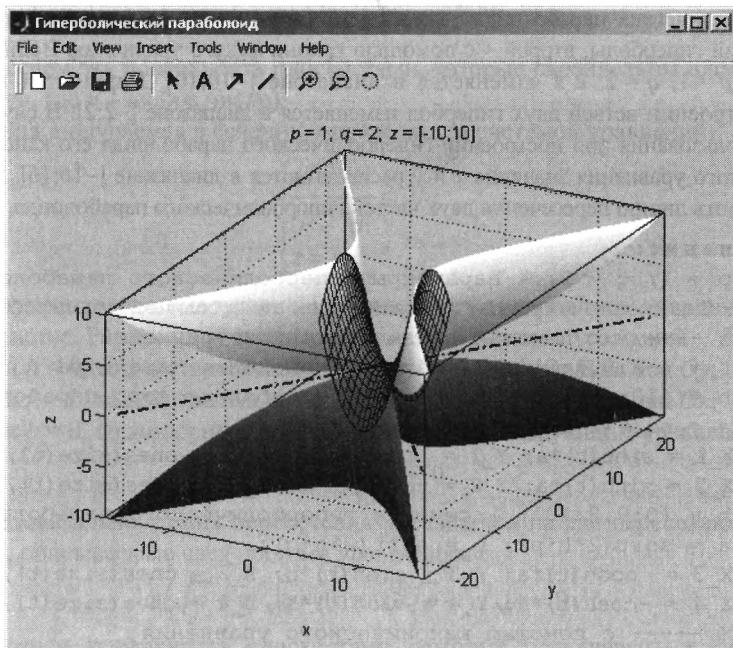
```
>> p = 1; q = 2; % параметры гиперболического параболоида
>> % Вычисление координат узловых точек гиперболического параболоида
>> % ----- с помощью параметрических уравнений
>> t = [-2:0.1:2]'; % диапазон изменения параметра
>> h = [10:-0.2:0]; % сечения гиперболического параболоида
>> a = sqrt(2*h*p); b = sqrt(2*h*q);
>> X_1 = sinh(t)*a; Y_1 = cosh(t)*b; Z_1 = -ones(size(t),1)*h;
>> X_2 = sinh(t)*a; Y_2 = -cosh(t)*b; Z_2 = -ones(size(t),1)*h;
>> h = [0:0.2:10]; % сечения гиперболического параболоида
>> a = sqrt(2*h*p); b = sqrt(2*h*q);
>> X_3 = cosh(t)*a; Y_3 = sinh(t)*b; Z_3 = ones(size(t),1)*h;
>> X_4 = -cosh(t)*a; Y_4 = sinh(t)*b; Z_4 = ones(size(t),1)*h;
>> % ----- с помощью канонического уравнения
>> x=[-4.5:0.5:4.5]; y=x; [X,Y]=meshgrid(x,y); Z=(X.^2./p-Y.^2./q)./2;
>> % Вычисление координат линий пересечений
>> x = [-16:16]; y_1 = x*sqrt(q)/sqrt(p); z_1 = zeros(size(x));
>> y_2 = -x*sqrt(q)/sqrt(p); z_2 = z_1;
>> % создание графического окна
>> figure('NumberTitle','off','Name','Гиперболический параболоид')
>> colormap(gray)
>> % построение поверхности
>> surf(X_1,Y_1,Z_1,'EdgeColor','none'), hold on
>> surf(X_2,Y_2,Z_2,'EdgeColor','none')
>> surf(X_3,Y_3,Z_3,'EdgeColor','none')
>> surf(X_4,Y_4,Z_4,'EdgeColor','none')
```

```

>> surf(X,Y,Z,'FaceColor','none')
>> % построение линии пересечения
>> line(x,y_1,z_1,'LineWidth',1.5,'Color',[0,0,0],'LineStyle','-.')
>> line(x,y_2,z_2,'LineWidth',1.5,'Color',[0,0,0],'LineStyle','-.')
>> % оформление графика
>> axis equal, box on, set(gca,'Color','none'), view(28,21)
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title(' \it p \rm= 1; \it q \rm= 2; \it z \rm= [-10;10] ')

```

Результаты команд, приведенных выше, изображены на рис. 5.32.



Р и с . 5.32. Гиперболический параболоид

В [44] приведен еще один способ построения гиперболического параболоида с помощью следующих *параметрических уравнений*:

$$x = a \cdot t \cdot \sin(\tau), \quad y = b \cdot t \cdot \cos(\tau), \quad z = -c \cdot t^2 \cdot \cos(2\tau),$$

где $t \in [0, \infty)$, $\tau \in [0, 2\pi]$.

Данный способ построения гиперболического параболоида является более предпочтительным, чем приведенный в примере 5.35, так как гиперболический параболоид отображается в координатном пространстве с помощью одной поверхности.

Пример 5.37. Построить гиперболический параболоид с параметрами $t = [0;2]$, $\tau = [0;2\pi]$ и величинами $p = 1$, $q = 2$. В качестве цветовой палитры графического окна выбрать палитру `summer` и изобразить линию пересечения с плоскостью Oxy .

Решение:

```
>> % диапазоны параметров
>> t = [0:0.1:2]'; tau = 0:pi/16:2*pi;
>> % значения коэффициентов
>> p = 1; q = 2; a = sqrt(p); b = sqrt(q); c = 1/2;
>> % значения координат гиперболического параболоида
>> X = a * t * sin(tau);
>> Y = b * t * cos(tau);
>> Z = -c * t.^2 * cos(2*tau);
>> % координаты линии пересечения
>> x1 = [ a * t(end:-1:1) * sin(pi/4); a * t * sin(5*pi/4)];
>> y1 = [ b * t(end:-1:1) * cos(pi/4); b * t * cos(5*pi/4)];
>> z1 = [-c*t(end:-1:1).^2*cos(2*pi/4); -c*t.^2*cos(10*pi/4)];
>> x2 = [ a * t(end:-1:1) * sin(3*pi/4); a * t * sin(7*pi/4)];
>> y2 = [ b * t(end:-1:1) * cos(3*pi/4); b * t * cos(7*pi/4)];
>> z2 = [-c*t(end:-1:1).^2*cos(6*pi/4); -c*t.^2*cos(14*pi/4)];
>> % оформление графического окна
>> fig = figure;
>> cl_fig = get(fig,'Color');
>> colormap(summer);
>> % построение поверхности с помощью параметрических уравнений
>> surf(X,Y,Z)
>> line(x1,y1,z1,'Color','k','LineWidth',2)
>> line(x2,y2,z2,'Color','k','LineWidth',2)
>> % ----- с помощью канонического уравнения
>> x=[-1:0.25:1]; y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=(X.^2./p-Y.^2./q)./2;
>> surf(X,Y,Z,'FaceColor','r')
>> % оформление координатного пространства
>> set(gca,'Color',cl_fig,'DataAspectRatio',[1 1 1]);
>> box on
>> xlabel('x'), ylabel('y'), zlabel('z')
>> str=['a = ', int2str(a), ', b = ', int2str(b), ', c = ',int2str(c)];
>> title(str)
```

Результаты вышеприведенных команд изображены на ил. 14.

5.2.8.6. Конус второго порядка

Конусом второго порядка называется поверхность, которая в некоторой прямоугольной системе координат определяется каноническим уравнением

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0.$$

Рассечем эту поверхность плоскостями, которые параллельны плоскости Oxy ($z = h$, где h – любое число).

Линия, получаемая в сечении, определяется системой уравнений

$$\begin{cases} \frac{x^2}{a^2} + \frac{y^2}{b^2} = \frac{h^2}{c^2}, \\ z = h. \end{cases}$$

Рассмотрим частные случаи этой системы уравнений:

- a) $h = 0$ – линия вырождается в точку с координатами $(0;0;0)$;
- б) $h \neq 0$ – в сечении будут иметь место эллипсы

$$\begin{cases} \frac{x^2}{a^2 h^2} + \frac{y^2}{b^2 h^2} = 1, \\ z = h. \end{cases}$$

Полуоси этих эллипсов будут возрастать при увеличении $|h|$.

Рассечем конус второго порядка плоскостью Oyz ($x = 0$). В сечении получится линия

$$\begin{cases} \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0, \\ x = 0. \end{cases}$$

Полученная линия распадается на две пересекающиеся прямые

$$\frac{y}{b} - \frac{z}{c} = 0 \quad \text{и} \quad \frac{y}{b} + \frac{z}{c} = 0.$$

При пересечении конуса второго порядка плоскостью Oxz ($y = 0$) в сечении будет иметь место линия

$$\begin{cases} \frac{x^2}{a^2} - \frac{z^2}{c^2} = 0, \\ y = 0, \end{cases}$$

которая распадается на пересекающиеся прямые

$$\frac{x}{a} - \frac{z}{c} = 0 \quad \text{и} \quad \frac{x}{a} + \frac{z}{c} = 0.$$

Поверхности, образованные из прямых линий, называются линейчатыми. Примерами линейчатых поверхностей являются цилиндрические и конические поверхности.

Пример 5.38. В одном графическом окне создать два координатных пространства. В первом координатном пространстве изобразить графический файл example.jpg, а во втором координатном пространстве построить конус второго порядка с коэффициентами $a = 1$, $b = 2$ и $c = 2$. На построенную поверхность наложить рисунок из файла example.jpg.

Решение:

```
>> a = 1; b = 2; c = 2; % параметры конуса второго порядка
>> h = [-2:0.2:2]; % сечения поверхности
>> theta = [0:pi/16:2*pi]'; % долгота
>> a1 = sqrt(a^2*h.^2/c^2); b1 = sqrt(b^2*h.^2/c^2);
>> % значения координатных узлов эллиптического параболоида
>> X=cos(theta)*a1; Y=sin(theta)*b1; Z=ones(size(theta),1)*h;
>> % считывание изображения в переменную
>> A = imread('example','jpg');
>> % создание графического окна
>> figure('Name','Конус второго порядка',...
'Color','w','MenuBar','none')
>> subplot(1,2,1)
>> % вывод в координатную плоскость графического файла
>> image(A), axis square, axis off
>> title('example.jpg')
>> subplot(1,2,2)
>> % построение поверхности
>> surf(X,Y,Z,'CData',A,'FaceColor','texture','EdgeColor','none')
>> % оформление графика
>> axis equal, box on, set(gca,'Color','none'), view(-36,23)
>> xlabel('x'), ylabel('y'), zlabel('z')
>> title('\ita \rm= 1; \itb \rm= 2; \itc \rm= 2')
```

Результаты команд представлены на ил. 15. Система MATLAB позволяет просматривать файлы различных форматов, содержащие графические изображения. Перед выводом содержимого графического файла на экран его необходимо считать в переменную. Для считывания содержимого графического файла в переменную используется функция `imread('имя_файла','тип_файла')`. В результате считывания данных из файла переменная представляет собой либо двухмерный массив, если изображение полутоновое, либо трехмерный, если изображение полноцветное. Функция `image()` отображает в коорди-

натной плоскости графическое изображение, которое хранится во входном аргументе. Для записи графического изображения в файл следует использовать функцию `imwrite(A, 'имя_файла', 'тип_файла')`. Дополнительную информацию о работе с изображениями можно посмотреть в [63].

5.3. ВЫВОДЫ К ГЛАВЕ 5

На примерах данной главы продемонстрировано использование системы MATLAB при решении следующих задач:

- 1) преобразование из одной системы координат в другую:
 - прямоугольная (плоскость и пространство);
 - полярная (плоскость);
 - цилиндрическая (пространство);
 - сферическая (пространство);
- 2) построение прямых линий на координатной плоскости, заданных с помощью:
 - общего уравнения;
 - уравнения с угловым коэффициентом;
 - уравнения линии, проходящей через данную точку в заданном направлении;
 - уравнения линии, проходящей через две точки;
 - уравнения линии в отрезках;
 - уравнения линии, проходящей через заданную точку перпендикулярно заданному вектору;
 - полярного уравнения;
 - параметрического уравнения;
- 3) построение линий второго порядка на координатной плоскости:
 - окружности;
 - эллипса;
 - гиперболы;
 - параболы;
- 4) построение линий высших порядков;
- 5) построение плоскостей в координатном пространстве, заданных в виде:
 - общего уравнения;
 - уравнения плоскости, проходящей через три точки;
 - уравнения плоскости в отрезках;
 - уравнения плоскости, проходящей через точку перпендикулярно данному вектору;
- 6) построение линий в координатном пространстве, заданных с помощью:
 - общих уравнений;
 - параметрических уравнений;

- канонических уравнений;
 - уравнение прямой линии, проходящей через две точки;
- 7) вычисление угла между плоскостями;
- 8) определение расстояния от точки до плоскости;
- 9) вычисление угла между прямыми линиями;
- 10) определение взаимного расположения прямых линий в пространстве;
- 11) вычисление угла между прямой линией и плоскостью;
- 12) вычисление точки пересечения прямой линии и плоскости;
- 13) построение цилиндрических поверхностей;
- 14) построение поверхностей вращения;
- 15) построение поверхностей второго порядка:
 - эллипсоида;
 - однополостного гиперболоида;
 - двуполостного гиперболоида;
 - эллиптического параболоида;
 - гиперболического параболоида;
 - конуса;
- 16) построение поверхностей высших порядков.

При решении вышеуказанных задач использовались стандартные функции системы MATLAB, список имен которых вместе с кратким описанием приведен в табл. 5.14.

Таблица 5.14

Функции для отображения и обозначения графических объектов

Функция	Описание
axes()	Создание координатного пространства и задание его свойств
axis()	Задание свойств координатного пространства
box on box off	Управление видимостью параллелепипеда, ограничивающего координатное пространство
colorbar()	Вывод шкалы цветовой палитры в графическое окно
colormap()	Задание цветовой палитры графического окна
fill3()	Создание многоугольников внутри координатного пространства и задание цветов их поверхностей и ребер
findobj()	Поиск объектов
figure()	Создание графического окна и задание его свойств

Окончание табл. 5.14

Функция	Описание
gca()	Получение дескриптора текущего координатного пространства графического окна
gcf	Получение дескриптора текущего графического окна
gco()	Получение дескриптора текущего объекта графического окна
get()	Получение значений свойств указанного объекта
grid on grid off	Управление видимостью координатной сетки (on – показ сетки, off – удаление сетки)
hold on hold off	Переключение между режимами вывода графиков в координатное пространство (on – добавление нового графика, off – замена текущего графика)
line()	Создание линии и задание ее свойств
mesh()	Создание поверхности в виде сетки
meshgrid()	Формирование двухмерных массивов на основе одномерных
plot()	Построение двухмерных графиков линий с указанием их цвета и типа, а также типа маркеров
plot3()	Построение трехмерных графиков линий с указанием их цвета и типа, а также типа маркеров
rectangle()	Создание прямоугольника внутри координатного пространства и задание его свойств
set()	Задание значений свойств указанного объекта
shading()	Задание цветового перехода в пределах ячейки поверхности и многоугольника
subplot()	Создание нескольких координатных пространств в одном графическом окне
surface() surf()	Создание поверхности и задание ее свойств
text()	Создание текста и задание его свойств
title()	Вывод заголовка координатного пространства
view()	Задание угловых координат точки наблюдения
xlabel()	Обозначение оси абсцисс
ylabel()	Обозначение оси ординат
zlabel()	Обозначение оси аппликат

Глава 6. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ MATLAB

Использование языка программирования позволяет автоматизировать исследование влияния значений коэффициентов, входящих в уравнения кривой и поверхности, на их форму и положение внутри координатного пространства.

6.1. КРАТКИЙ ОБЗОР ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Языком программирования называется язык, предназначенный для записи программ, исполняемых на ЭВМ.

Программа – это предписание ЭВМ на языке программирования, позволяющее решить требуемую задачу.

Синтаксис языка программирования – это набор правил построения конструкций языка.

Семантика языка программирования – это совокупность значений (смысла) всех конструкций языка.

Различают языки программирования *низкого и высокого уровня*. Языки низкого уровня ориентированы на систему команд процессора и аппаратную реализацию ЭВМ. К языкам низкого уровня относят машинные языки и языки ассемблера. Языки высокого уровня позволяют записывать программу с помощью абстрактных конструкций и символьных обозначений, которые понятны не только ЭВМ, но и человеку. Для преобразования программы, написанной на языке высокого уровня, в последовательность машинных команд необходимо использовать *транслятор*, который может работать в режиме *компилятора* (преобразование всей программы до ее выполнения) или *интерпретатора* (последовательное преобразование команд во время выполнения). Применение языков программирования высокого уровня позволяет сократить время на создание программ, решающих математические задачи.

Совокупность правил, лежащих в основе синтаксиса и семантики языка программирования, позволяет отнести его к одному из *стилей программирования*. Различают следующие стили программирования: *неструктурный, структурный, логический, объектно-ориентированный и функциональный*.

Неструктурный стиль программирования допускает использование в явном виде команды безусловного перехода (*goto*), присущей языкам ассемблера. Использование неструктурного стиля программирования затрудняет разработку больших программ.

Структурный стиль программирования был окончательно сформирован на рубеже 60-х и 70-х годов. В основе этого стиля лежат две идеи: *декомпо-*

зация задачи и использование трех управляющих конструкций (*последовательное выполнение, ветвление и цикл*). Под *декомпозицией задачи* понимается разбиение исходной задачи на ряд более мелких подзадач, которые решаются своей подпрограммой (функцией или процедурой). Принципы структурного стиля программирования заложены в языки программирования ALGOL (ALGOrithmic Language, 1960), BASIC (Beginner's All-purpose Symbolic Instruction Code, 1964), C (1972), FORTRAN (1958 и 1977) и Pascal (1971).

Логический стиль программирования не использует последовательные структурные абстракции, которые являются следствием машинной архитектуры фон Неймана, а базируется на создании совокупности фактов и правил, полностью описывающих некоторую ситуацию или предметную область. Следовательно, программист должен только описать постановку задачи, а поиски путей решения возлагаются на транслятор. Логическое программирование стало возможным после появления в 1972 г. языка PROLOG (PROgramming in LOGic). На сегодняшний день создано несколько расширений этого языка. Предполагается, что логические языки окажут сильное влияние на развитие систем с искусственным интеллектом.

Объектно-ориентированное программирование (ООП) является надстройкой над структурным программированием, т.е. абстракцией более высокого порядка. В основе ООП лежит отображение объектов реального мира с помощью описания их свойств и методов работы с ними. Примерами языков программирования, которые используют объектно-ориентированный подход, являются Ada (1983), C++ (1986), C# (2000–2001), Java (1995), Modula-3 (1988), Object Pascal (1986) и Smalltalk (1980). Многие из вышеназванных языков являются гибридными, так как поддерживают объектно-ориентированный и структурный стили программирования. ООП является оптимальным вариантом при реализации графического интерфейса пользователя, а также для разработки сетевых программ, поддерживающих архитектуру «клиент-сервер» и работающих через произвольный браузер.

Функциональный стиль программирования основан на использовании при вычислениях функций, которые определяются через другие функции или через себя (рекурсия). Программа, реализованная с помощью функционального стиля программирования, представляет собой определения функций. В процессе ее выполнения функции получают параметры, вычисляют и возвращают результат. При необходимости вычисляются значения других функций. Примерами функциональных языков являются LISP (LISt Processing, 1956) и его многочисленные диалекты. Языки функционального программирования применяются в области искусственного интеллекта.

Дополнительную информацию о стилях и языках программирования можно посмотреть в [1, 14, 17, 18, 25, 27, 28, 29, 34, 40, 42, 43, 46, 47, 53, 64, 77].

Язык MATLAB является языком высокого уровня, который позволяет решать научные и технические задачи, используя структурный и объектно-ориентированный стили программирования. Дальнейшее изложение материала базируется на использовании структурного стиля программирования при решении математических задач.

6.2. ПОНЯТИЕ М-ФАЙЛА

В систему MATLAB включен язык программирования высокого уровня, с помощью которого можно решать рассмотренные выше задачи векторной алгебры, линейной алгебры и аналитической геометрии с помощью матричной записи, а также позволяющий как автоматизировать, так и визуализировать процесс исследования влияния параметров, входящих в уравнения, на положение и форму линии и поверхности в координатном пространстве.

Система MATLAB ориентирована на работу в диалоговом режиме, т.е. необходимо в командной строке ввести команду и передать ее ядру системы. После проверки синтаксиса команда обрабатывается ядром и результат выводится в командное и/или графическое окно. После выполнения команды система переходит в режим ожидания ввода новой команды. В том случае, если во введенной команде имеются синтаксические ошибки, в командное окно выводится соответствующее сообщение и система переходит в режим ожидания ввода новой команды без обработки введенной. Если необходимо просмотреть итоговый результат, но с другими начальными данными, следует заново ввести в командную строку все необходимые команды, так как система MATLAB не имеет возможности динамического обновления результата при изменении значения исходной переменной. Существует две возможности автоматизации повторного ввода серии команд. Первый способ заключается в использовании окна Command History, в котором содержатся введенные ранее команды. Этот способ хорошо подходит для повторного выполнения небольшого количества команд. Однако в некоторых случаях – в зависимости от текущих значений переменных – требуется выполнять различные последовательности команд заранее неизвестное число раз. В таких случаях следует применять второй подход, основанный на применении m-файлов, которые могут содержать команды и управляющие структуры языка MATLAB. После создания m-файла его вызов осуществляется заданием имени, а если необходимо – заданием входных и выходных параметров, в командной строке.

M-файлы представляют собой текстовые файлы, сохраненные с расширением m. Так как m-файл является текстовым файлом, то его можно создавать в любом текстовом редакторе. В состав системы MATLAB входит Editor/Debugger, который позволяет как создавать m-файлы, так и выполнять их отладку, что приводит к сокращению времени создания рабочего m-файла. Следовательно, при создании m-файлов наиболее предпочтительным вариан-

том является использование встроенного редактора с возможностями отладки и выделения синтаксических конструкций языка MATLAB. Открыть Editor/Debugger для создания нового m-файла можно с помощью команды M-file, расположенной в подменю New главного меню File, или с помощью нажатия на кнопку New панели инструментов рабочего стола системы MATLAB.

M-файлы подразделяются на два типа: сценарии (script) и функции (function).

Сценарии, являющиеся самым простым типом m-файлов, содержат последовательности команд, которые задаются в командной строке, и комментарии, начинающиеся со знака %. Отличительными особенностями сценариев являются:

- выполнение команд в режиме интерпретации, т.е. команды преобразуются в исполнительный код и выполняются построчно;
- работают только с переменными, расположенными в рабочей области MATLAB.

Следовательно, необходимо быть внимательным при создании переменных во время выполнения сценария, так как новые переменные, имена которых совпадают с именами переменных, расположенных в рабочей области, заменят их, что может привести к потере необходимой для дальнейшей работы информации.

Пример 6.1. Вычислить углы наклона вектора $\bar{a} = (-1; 2; 5)$ к осям координат. Проверить результат. Решение оформить в виде сценария с именем example_6_1.

Решение:

Листинг 6.1

```
% Задание исходного вектора
A = [-1, 2, 5]
% Вычисление углов наклона
A = acos(A ./ sqrt(sum(A.*A))) * 180/pi
% Проверка полученного результата
A = sum(cos(A ./ 180*pi).^2)
```

Результаты выполнения сценария, представленного на листинге 6.1.

```
>> example_6_1
A =
    -1      2      5
A =
   100.5197    68.5833    24.0948
A =
    1
```

Во время выполнения сценария переменная A сначала содержит значения координат вектора \bar{a} . После выполнения второй команды переменная A содержит значения в градусах углов наклона вектора \bar{a} к осям Ox , Oy и Oz соответственно. После окончания выполнения третьей команды завершается выполнение сценария. Переменная A, которая располагается в рабочей области системы MATLAB, содержит результат выполнения третьей команды. Так как $A = 1$, то углы наклона вектора к координатным осям вычислены правильно.

Функции наряду со сценариями также содержат команды, но являются более сложным типом m-файлов по сравнению со сценариями. Отличительными особенностями функций от сценариев являются:

- возможность компилирования всей функции в исполняемый код с последующим размещением его в памяти;
- наличие собственной рабочей области, где хранятся локальные переменные;
- наличие входных и выходных параметров.

После выполнения функции ее рабочая область удаляется из памяти. Следовательно, все переменные, которые были созданы во время выполнения функции и располагались в ее рабочей области, удаляются из памяти.

Пример 6.2. Вычислить углы наклона вектора $\bar{a} = (-1; 2; 5)$ к осям координат. Вычисление оформить в виде функции с именем angles.

Решение:

Листинг 6.2

```
function [a] = angles(A)
%ANGLES вычисление углов наклона вектора к осям координат
a = acos(A./sqrt(sum(A.*A)))*180/pi;
```

Результаты выполнения функции, представленной на листинге 6.2.

```
>> A = [-1,2,5]; [a] = angles(A)
a =
    100.5197    68.5833    24.0948
```

После задания команды, в которой имеется вызов функции, управление передается этой функции. В функцию angles() с помощью входного аргумента A передаются координаты вектора, а с помощью выходного аргумента a возвращается результат выполнения функции. Результатом является вектор, размер которого совпадает с размером входного вектора.

Как функции, так и сценарии позволяют использовать управляющие структуры, которые служат для управления последовательностью выполнения содержащихся в них команд.

6.3. УПРАВЛЯЮЩИЕ СТРУКТУРЫ

Серьезным шагом в развитии программирования считается переход от неструктурного стиля к структурному стилю программирования. Термин «структурное программирование» (structured programming) был введен голландским программистом Эдсгером Дейкстра (Edsger Dijkstra). Одним из ощутимых результатов исследований в области структурного программирования в 60-х годах было создание в 1971 г. швейцарским математиком Никлаусом Виртом языка программирования Pascal, который наиболее широко используется при обучении программированию в университетах разных стран. Бома (Bohm) и Джакопини (Jacopini) доказали, что все программы могут быть написаны с использованием всего трех управляющих структур: **следования, выбора и повторения**. Последовательное выполнение команд в зависимости от приоритета операций было рассмотрено в гл. 2. Язык MATLAB предоставляет три структуры выбора: **if end – структура с единственным выбором**, **if else end – структура с двойным выбором** и **switch – структура с множественным выбором**. В языке MATLAB присутствует два типа структур повторения: **while end – цикл с условием** и **for end – цикл с параметром**.

6.3.1. Структура выбора if end (ЕСЛИ)

Структура выбора **if end** используется в том случае, когда должно выполниться действие или последовательность действий в зависимости от некоторого условия. Структура выбора **if end** имеет следующую форму:

```
if логическое_выражение
    инструкция_1
    инструкция_2
    .....
    инструкция_n
end
```

Если условие, являющееся логическим выражением в терминологии языка MATLAB, удовлетворено, т.е. условие есть `true(1)`, то выполняются инструкции, расположенные в теле структуры. В противном случае программа переходит на следующую инструкцию, расположенную после ключевого слова `end`. Логическое выражение в языке MATLAB может возвращать массив логических значений. В этом случае инструкции, расположенные внутри структуры, будут выполняться, если все элементы массива имеют значение `true(1)`.

При записи инструкций в структуре `if end` следует делать отступ для улучшения читаемости программы. Ступенчатая, или рельефная, запись программ является хорошим стилем программирования. Выбранную длину от-

ступа следует использовать во всей программе. Как правило, длина отступа составляет два или три символа.

Структуру выбора `if end` можно записать в одну строку, отделяя логическое выражение и инструкции друг от друга с помощью запятой или/или точки с запятой. Например:

```
if лог._выражение, инстр._1, инстр._2; ..., инстр._n; end
```

Запись в одну строку следует использовать, если структура выбора `if end` содержит одну инструкцию. В противном случае следует использовать ступенчатую запись.

Структуры выбора при работе с системой MATLAB применяются главным образом для вывода сообщений в командное окно или для проверки класса переменной при работе функции или числа входных параметров функции с целью досрочного завершения выполнения функции. При выделении из исходного массива данных нового массива или изменении значений группы элементов в исходном массиве, значения которых удовлетворяют заданному условию, в системе MATLAB следует использовать *логическое индексирование*. Для выполнения логического индексирования необходимо с помощью логических операций, операций сравнения или функций, возвращающих результат логического класса, сформировать логический массив, содержащий одинаковое число элементов, с исходным массивом. Сформированный логический массив состоит из нулей и единиц, где единичные значения соответствуют выполнению условия, а нулевые значения – невыполнению этого условия. После формирования логического массива его необходимо использовать в качестве индексов исходного массива данных. При выполнении логического индексирования индекс исходного массива определяется через индекс логического массива, а выбор элемента из исходного массива осуществляется из условия истинности значения соответствующего элемента логического массива. Следовательно, при реализации логического индексирования в неявном виде используется структура единственного выбора `if end`. В том случае, если логическое индексирование используется слева от знака присваивания, то выбранным элементам исходного массива присваивается значение, находящееся справа от знака присваивания. Если логическое индексирование используется справа от знака присваивания или является одиночной инструкцией, то создается новый массив из выбранных элементов исходного массива. Преимуществом логического индексирования по сравнению с использованием структуры `if end` в явном виде является сокращение времени выполнения программы, а недостатком – выделение дополнительной памяти для хранения значений элементов логического массива, используемых в качестве индексов. Количество элементов в исходном массиве определяет число байт, которое будет занимать логический массив в памяти.

Логическое индексирование использовалось при отображении логической области в графическом окне во время выполнения практикума 4.

Пример 6.3. Задать массив целых положительных чисел и определить, сколько в нем простых чисел. Если простые числа найдены, то вывести их количество и сами эти числа. Решение оформить в виде сценария с именем example_6_3.

Решение:

Листинг 6.3

```
% ----- Определение кол-ва простых чисел в массиве -----
% Приглашение для ввода массива положительных чисел
answer = input('Введите массив положительных чисел: ');
if any(isprime(answer(:))) % поиск простого числа в массиве
    s = sum(isprime(answer(:))); % подсчет суммы элементов
    % формирование строки сообщения
    string = ['Вы ввели ' num2str(s) ' простых чисел:'];
    disp(string) % вывод строки сообщения
    answer(isprime(answer(:))) % вывод простых чисел
end % окончание структуры выбора
```

Результаты выполнения инструкций, представленных на листинге 6.3, при задании первых двадцати натуральных чисел:

```
>> example_6_3
Введите массив положительных чисел: 1:20
Вы ввели 8 простых чисел:
ans =
     2      3      5      7     11     13     17     19
```

Следовательно, в первых двадцати натуральных числах содержится восемь простых чисел.

Первая строка на листинге 6.3 содержит *комментарий*, который объясняет цель последующих инструкций. Комментарии начинаются с символа % и продолжаются до конца строки. Комментарии позволяют документировать программу и облегчить ее понимание. Во время выполнения команд интерпретатор игнорирует текст, помещенный в комментарии. В языке MATLAB имеется только односторонний комментарий. Использование комментариев считается хорошим стилем программирования при написании программ любой сложности.

Функция input() использовалась для организации диалога с пользователем. Входной аргумент функции выводится в командную строку, и система MATLAB ожидает ввода массива чисел, приостанавливая выполнение инструкции. После ввода массива целых чисел система MATLAB присваивает его переменной answer класса double и продолжает интерпретацию следующих инструкций.

Функция `any()` возвращает логическое значение `true` в том случае, если в одномерном массиве имеется хотя бы одно значение, отличное от нуля. В случае двухмерного массива отдельно проверяются столбцы и возвращается вектор нулей и/или единиц. Если в рассматриваемом примере не преобразовать исходный массив в одномерный массив, то в ряде случаев при наличии в нем простых чисел результатом логического выражения будет значение `false`. Почему? Функция `isprime()` возвращает массив логического класса той же самой размерности, что и исходный числовой массив `answer`. Истинные значения в итоговом массиве свидетельствуют о том, что в соответствующей позиции исходного массива располагается простое число. С целью получения одного логического значения после выполнения логического выражения массив входных значений `answer` предварительно преобразуется в одномерный массив (вектор-столбец) с помощью инструкции `answer(:)`. Полученный одномерный массив передается функции `isprime()`, которая просматривает его и формирует соответствующий логический массив. Возвращаемый функцией `isprime()` логический массив передается в функцию `any()`, которая возвращает значение логического класса, используемое для входа в тело структуры `if end` или для его обхода.

Первая инструкция в теле структуры предназначена для вычисления суммы простых чисел в исходном массиве. Система MATLAB перед выполнением арифметических операций осуществляет проверку классов переменных, входящих в выражение, и их приведение, если необходимо, к арифметическому классу. Следовательно, при выполнении суммирования значений элементов одномерного массива логического класса с помощью функции `sum()` осуществляется предварительное приведение исходного массива логического класса к массиву арифметического класса с последующим вычислением суммы элементов. С целью получения единственного числа исходный массив был преобразован в одномерный массив. Таким образом, при вычислении количества простых чисел во введенном массиве выполняются следующие действия:

- исходный массив трансформируется в одномерный массив `(:)`;
- на месте простых чисел в преобразованном массиве помещаются единицы, а в остальных местах – нули (`isprime()`);
- суммирование единиц в логическом массиве (`sum()`).

Следующая строка в теле структуры `if end` является комментарием, раскрывающим назначение инструкции на следующей строке.

При формировании итогового текстового сообщения использовалась функция `num2str()`, которая преобразует число в его символьное представление. Сформированное сообщение присваивается переменной `string`, которая становится символьного класса.

Функция `disp()` используется для вывода текстового сообщения в командное окно. В рассматриваемом примере функции `disp()` передается переменная `string`, содержимое которой выводится в командную строку.

Последняя инструкция в теле структуры `if end` присваивает переменной `ans` массив простых чисел, находящихся во введенном массиве, и выводит ее в командное окно. Для компактного вывода использовалась операция транспонирования, которая преобразует вектор-столбец в вектор-строку. При выполнении этой инструкции использовалось логическое индексирование, с помощью которого было выполнено выделение простых чисел из исходного массива данных и их запись в переменную `ans`. После выполнения этой инструкции завершается выполнение структуры `if end` и интерпретатор переходит к обработке следующей за ключевым словом `end` инструкции.

Как правило, при написании программ с использованием структурного стиля программирования применяют графическое описание программы или ее фрагмента в виде **блок-схемы**. На рис. 6.1 изображена блок-схема сценария

из примера 6.3. Блок-схема состоит из соединенных с помощью линий связи специальных символов, таких, как прямоугольники, параллелограммы, ромбы, овалы и круги.

Символ овала используется с целью обозначения начала и конца программы. Следовательно, любая программа должна начинаться с символа овала, который содержит слово «Начало», и заканчиваться символом овала, содержащим слово «Конец».

При обозначении фрагментов программы символы овалов заменяются на символы малых окружностей, называемых также *символами слияния*.

Символ прямоугольника, называемый *узлом обработки*, обозначает вычисления, которые выполняются в программе.

Рис. 6.1. Блок-схема к примеру 6.3

Символ параллелограмма обозначает ввод информации в программу, а также ее вывод. Под вводом информации в MATLAB понимается загрузка данных, операций, имен функций или целых выражений из файла или ввод их пользователем. Вывод информации может осуществляться как в файл, так и в командное окно.

Символ ромба, называемый *узлом проверки условия*, позволяет определить, какая инструкция будет выполняться следующей. Символ ромба обозначает логическое выражение в структуре выбора.

В сценарии, представленном на листинге 6.3, некоторые действия выполняются несколько раз, что приводит к увеличению времени выполнения. Попробуйте сократить количество действий при выполнении сценария.

6.3.2. Структура выбора if else end (ЕСЛИ ИНАЧЕ)

Структура `if end` позволяет определить одну группу инструкции, которая выполняется в том случае, если условие истинно, и пропускается в противном случае. Структура выбора `if else end` используется для определения двух групп инструкций, из которых в зависимости от условия выполняется только одна группа инструкций. Структура выбора `if else end` имеет следующий вид:

```
if логическое_выражение
    инструкция_true_1
    инструкция_true_2
    .....
    инструкция_true_n
else
    инструкция_false_1
    инструкция_false_2
    .....
    инструкция_false_m
end
```

Если результатом логического выражения является логическая единица или массив логических значений, состоящий из логических единиц, то выполняются инструкции, расположенные между логическим выражением и ключевым словом `else`, в противном случае – инструкции, находящиеся между ключевыми словами `else` и `end`.

Пример 6.4. Задать массив целых положительных чисел и определить, сколько в нем простых чисел. Если простые числа найдены, то вывести в окно команд их количество и сами эти числа. В противном случае вывести сообщение: Введенный массив не содержит простых чисел. Решение оформить в виде сценария.

Решение:

Листинг 6.4

```
% Приглашение для ввода массива положительных чисел
answer = input('Введите массив положительных чисел: ');
```

Окончание листинга 6.4

```

tic % включение таймера
a = isprime(answer(:)); % логический массив
if any(a) % проверка введенного массива
    s = sum(a); % подсчет суммы элементов
    % формирование строки сообщения
    string = ['Вы ввели ' num2str(s) ' простых чисел:'];
    disp(string) % вывод строки сообщения
    answer(a') % вывод простых чисел
else
    % формирование строки сообщения
    string = ['Введенный массив не содержит простых чисел'];
    disp(string) % вывод строки сообщения
end % окончание структуры выбора
toc % выключение таймера и вывод вычисленного времени

```

Результаты выполнения инструкций, представленных на листинге 6.4:

```

>> example_5_4
Введите массив положительных чисел: [1,4,6,8]
Введенный массив не содержит простых чисел
elapsed_time =
    0.0400
>> example_6_4
Введите массив положительных чисел: 1:20
Вы ввели 8 простых чисел:
ans =
    2      3      5      7     11     13     17     19
elapsed_time =
    0.0800

```

Таким образом, если во введенном массиве простые числа отсутствуют, то выполняется следующая за ключевым словом `else` инструкция, которая выводит сообщение в командное окно. После вывода сообщения завершается выполнение структуры `if else end` и программа переходит на инструкцию, расположенную после ключевого слова `end`.

Для оценки времени выполнения инструкций, входящих в каждую ветвь структуры `if else end`, использовались функции `tic()` и `toc()`. Функция `tic()` служит для запуска таймера, а функция `toc` осуществляет его останов с последующим выводом зафиксированного времени в командное окно. Так как при истинном значении логического выражения кроме формирования сообщения и его вывода в командное окно выполняется подсчет количества простых чисел и их вывод, то время выполнения сценария в этом случае больше. Увеличение количества элементов в исходном массиве, а также количества простых

чисел в этом массиве приведет к увеличению времени выполнения сценария. Попробуйте повторить пример 6.4, задав входной массив, состоящий из 1000 первых натуральных чисел.

Блок-схема сценария, который представлен на листинге 6.3, изображена на рис. 6.2.

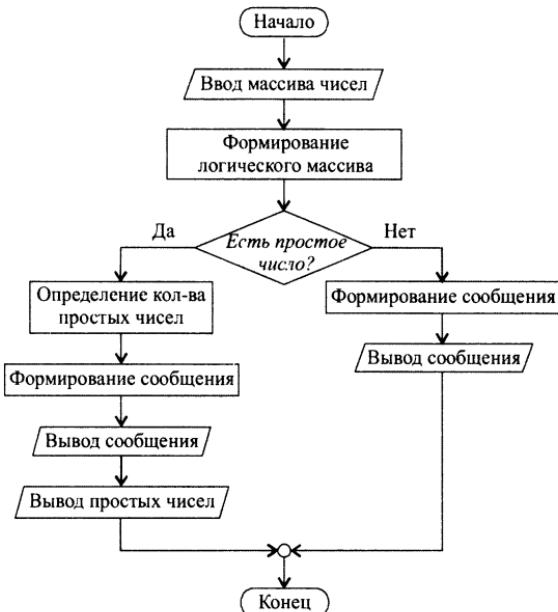


Рис. 6.2. Блок-схема к примеру 6.4

При выполнении вычислений, связанных с выделением из исходного массива группы или двух групп элементов, которые удовлетворяют определенному условию, можно обойтись без структуры выбора `if else end`. При этом необходимо использовать логическое индексирование, которое позволяет сократить время выполнения программы.

Пример 6.5. Используя логическое индексирование, сформировать на основе массива $A = [1, 2, 3, 5, 4, 6, 7, 8]$ новый массив B , в котором сначала располагаются нечетные элементы исходного массива, а затем – его четные элементы. Решение оформить в виде сценария с именем `example_6_5`.

Решение:

Листинг 6.5

```

A = [1, 2, 3, 5, 4, 6, 7, 8];
B = [A(rem(A, 2) ~= 0) A(rem(A, 2) == 0) ]

```

Результаты выполнения инструкций, представленных на листинге 6.5:

```
>> example_6_5
B =
    1     3     5     7     2     4     6     8
```

Формирование нового массива было выполнено с помощью логического индексирования и операции горизонтального соединения массивов. При логическом индексировании на месте индексов можно использовать не только имена логических массивов, но и логические выражения. В рассматриваемом примере логические выражения содержали вызов функции `rem(A, 2)`, которая возвращает массив остатков от деления значений соответствующих элементов массива A на два, и операцию сравнения. В первом случае с помощью оператора «`~=`» осуществляется проверка на неравенство нулю остатков от деления на два, а во втором случае с помощью оператора «`==`» – проверка их на равенство нулю. Каждое логическое индексирование в примере 6.5 соответствует одной ветви в структуре выбора `if else end`.

6.3.3. Структура выбора if elseif else end (ЕСЛИ ИНАЧЕ ЕСЛИ)

Рассмотренные выше структуры выбора `if end` и `if else end` являются структурами с одиночным и двойным выбором соответственно. В ряде случаев возникают ситуации, когда необходимо выбирать между тремя альтернативами. Например, число может быть больше, равно либо меньше другого числа. При существовании трех альтернативных путей выполнения вычислений следует применять структуру выбора `if elseif else end`, которая имеет следующий вид:

```
if логическое_выражение_1
    группа_инструкций_1
elseif логическое_выражение_2
    группа_инструкций_2
else
    группа_инструкций_3
end
```

Первая группа инструкций выполняется, если результатом первого логического выражения является логическая единица или массив логических значений, состоящий из логических единиц. Вторая группа инструкций выполняется, если в результате вычисления первого логического выражения получился хотя бы один логический ноль, а в результате вычисления второго логического выражения – логическая единица или массив одних логических единиц. В противном случае будет выполняться третья группа инструкций.

Пример 6.6. Задать массив целых положительных чисел и определить, сколько в нем простых чисел. Если простые числа найдены, то вывести их количество и сами эти числа. При формировании поясняющей надписи о количестве введенных простых чисел необходимо следить за окончанием слов: простое(ых) и число(а, чисел). В противном случае вывести сообщение: Введенный массив не содержит простых чисел. Решение оформить в виде сценария с именем example_6_6.

Решение:

Листинг 6.6

```
answer = input('Введите массив положительных чисел: ');
a = isprime(answer(:)); % логический массив
if any(a) % проверка наличия простых чисел в массиве
    s = sum(a); % вычисление кол-ва простых чисел
    str = int2str(s); % преобразование числа в строку
    d = size(str,2); % определение числа разрядов
    % определение значения правого десятичного разряда в числе
    lsd = str2num(str(d));
    % формирование сообщения о количестве простых чисел в массиве
    if ((d>1)&&(str2num(str(d-1))==1)) || ~((lsd>0)&(lsd<5))
        string = ['Вы ввели ' num2str(s) ' простых чисел'];
    elseif lsd == 1
        string = ['Вы ввели ' num2str(s) ' простое число'];
    else
        string = ['Вы ввели ' num2str(s) ' простых числа'];
    end
    disp(string) % вывод информации о количестве простых чисел
    answer(a') % вывод простых чисел
else
    string ='Во введенном массиве нет простых чисел';
    disp(string)
end
```

Результаты выполнения инструкций, представленных на листинге 6.6:

```
>> example_6_6
Введите массив положительных чисел: 1:31
Вы ввели 11 простых чисел
ans =
    2     3     5     7    11    13    17    19    23    29    31
>> example_6_6
Введите массив положительных чисел: 1:73
Вы ввели 21 простое число
```

```

ans =
    Columns 1 through 11
    2     3     5     7    11    13    17    19    23    29    31
    Columns 12 through 21
    37    41    43    47    53    59    61    67    71    73
>> example_6_6
Введите массив положительных чисел: 1:79
Вы ввели 22 простых числа
ans =
    Columns 1 through 11
    2     3     5     7    11    13    17    19    23    29    31
    Columns 12 through 22
    37    41    43    47    53    59    61    67    71    73    79

```

Блок-схема сценария, который представлен на листинге 6.6, показана на рис. 6.3.

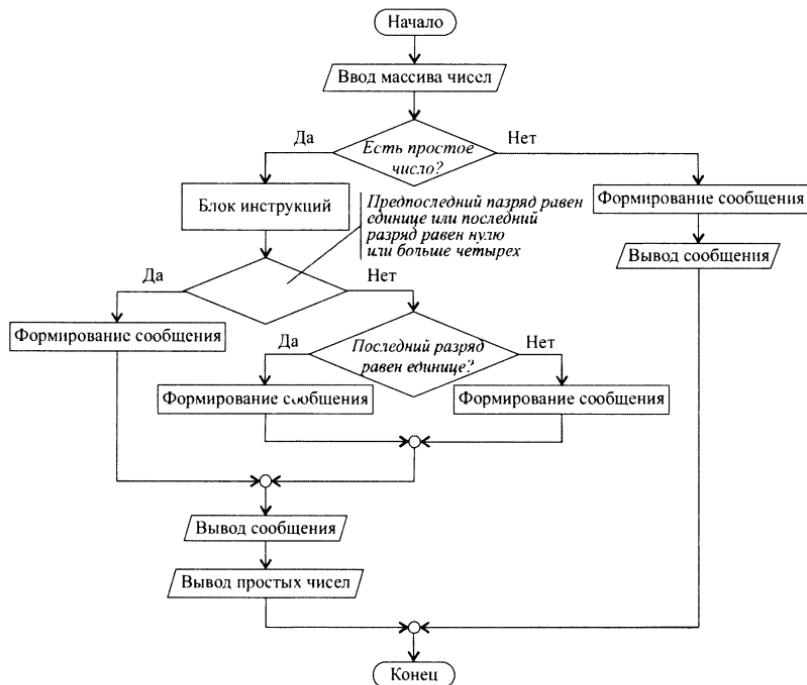


Рис. 6.3. Блок-схема сценария из примера 6.6

При решении примера 6.6 были использованы две структуры выбора: `if else end` и `if elseif else end`. Проверка на наличие простых чисел

в массиве осуществлялась с помощью структуры с двойным выбором, а формирование сообщения о количестве простых чисел – с помощью структуры с тройным выбором. Так как формирование сообщения о количестве простых чисел необходимо выполнять только в том случае, если введенный массив их содержит, то сначала следует проверить наличие в нем простых чисел. Следовательно, структуру с тройным выбором необходимо вставить в группу инструкций структуры с двойным выбором.

Перед формированием сообщения о количестве простых чисел во введенном массиве выполняются следующие действия: определение количества простых чисел в массиве, преобразование полученного числа в его символьное представление, определение количества разрядов в числе, которое соответствует количеству простых чисел в массиве, и определение последней цифры в этом числе. Если результатом вычисления первого логического выражения в структуре с тройным выбором является логическая единица, то имеет место число, оканчивающееся на 0, 5, 6, 7, 8 или 9, или числа 11, 12, 13 или 14, которым соответствует словосочетание «простых чисел». При формировании первого логического выражения использовались короткие логические операции (`&&` и `||`), которые позволяют досрочно прекратить вычисление логического выражения, если его результат уже определен. Если количество простых чисел в массиве меньше десяти (`d=1`), то вычисление значения второго разряда и проверка его на равенство единице (`str2num(str(d-1))==1`) выполняться не будет, так как выполнение этих действий приведет к аварийному завершению выполнения сценария. Ядро системы MATLAB перейдет к вычислению второй части логического выражения (`((1sd>0) & (1sd<5))`). Если второй разряд числа, соответствующего количеству простых чисел в массиве, равен единице, то вторая часть логического выражения выполняться не будет, так как результатом логического выражения в любом случае будет истина. В том случае, когда результатом вычисления первого логического выражения является логический ноль, ядро системы MATLAB переходит к вычислению второго логического выражения, которое располагается за ключевым словом `elseif`. Второе логическое выражение состоит из оператора сравнения, который используется для проверки значения последнего разряда числа, соответствующего количеству простых чисел, на равенство единице. Если логическое выражение возвращает логическую единицу, то число оканчивается на 1, которой соответствует словосочетание «простое число». В противном случае число оканчивается на цифру 2, 3 или 4, которым соответствует словосочетание «простых числа». После формирования сообщения о количестве простых чисел в массиве ядро системы MATLAB переходит к выполнению инструкции, следующей за ключевым словом `end` структуры с тройным выбором `if elseif else end`. Выполняется вывод сформированного сообщения в командное окно и присваивание пере-

менной `ans` всех простых чисел во введенном массиве с последующим ее выводом в командное окно. После выполнения этих инструкций осуществляется выход из структуры с двойным выбором и завершение работы сценария.

В том случае, если во введенном массиве нет простых чисел, то выполняются инструкции, расположенные после ключевого слова `else` структуры с двойным выбором. Эти инструкции формируют и выводят в командное окно сообщение: Во введенном массиве нет простых чисел. После ее выполнения осуществляется выход из структуры с двойным выбором с последующим завершением работы сценария.

На рис. 6.3 прямоугольник с надписью: «Блок инструкций» включает четыре последовательные инструкции: вычисление количества простых чисел, преобразование числа в строку, определение количества разрядов и определение правой цифры в вычисленном числе.

6.3.4. Структура множественного выбора `switch case end`

Помимо рассмотренных ранее структур с единственным, двойным и тройным выбором, язык MATLAB содержит структуру с множественным выбором, которая позволяет выполнять разные группы инструкций в зависимости от значений вычисленных выражений. Структура множественного выбора имеет следующий вид:

```
switch switch_выражение
    case case_выражение
        инструкция_1, ..., инструкция_n
    case {case_выражение_1, ..., case_выражение_k}
        инструкция_1, ..., инструкция_m
    otherwise
        инструкция_1, ..., инструкция_p
end
```

При входе в структуру множественного выбора выполняется вычисление `switch_выражения`. Результатом вычисления `switch_выражения` может быть скалярное значение либо массив символов. После получения результата осуществляется переход к первому ключевому слову `case` и выполняется вычисление следующего за ним `case_выражения`. После вычисления `case_выражения` полученный результат сравнивается с результатом `switch_выражения`. Если результатом вычисления выражений являются скалярные величины, то сравнение выполняется с помощью операции «==» (функции `eq()`), а в случае массивов символов – с помощью функции `strcmp(switch_выражение, case_выражение)`. Если результатом сравнения является логическая единица, то выполняется группа инструкций, которая следует за `case_выражением`. После ее выполнения происходит выход из структуры множественного выбора.

бара. В том случае, если результаты case_выражения и switch_выражения не совпадают, происходит переход к следующему ключевому слову – case, если оно есть, либо к первой инструкции, следующей за ключевым словом otherwise. После выполнения последней инструкции в ветви otherwise осуществляется выход из структуры множественного выбора. Допускается использование нескольких case_выражений за ключевым словом case. В этом случае их необходимо рассматривать как массив ячеек, т.е. заключить все case_выражения в фигурные скобки. В результате сравнения switch_выражения с массивом case_выражений логическая единица будет в том случае, если результат switch_выражения совпадает хотя бы с одним результатом case_выражений. Блок-схема структуры множественного выбора switch case end представлена на рис. 6.4.

Несмотря на то что структура switch может быть реализована без ветви otherwise, при написании хорошо структурированных и документированных программ рекомендуется ее использовать. Ветвь otherwise необходимо всегда размещать в структуре множественного выбора последней.

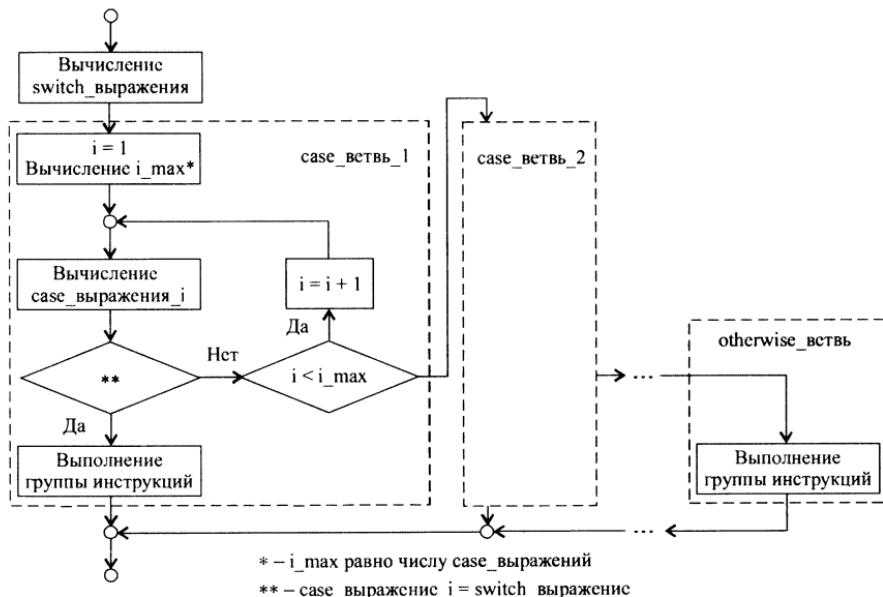


Рис. 6.4. Блок-схема структуры множественного выбора switch

Пример 6.7. Задать в командном окне два вектора и арифметическую операцию. С помощью структуры switch выполнить соответствующую арифметическую операцию и вывести результат в командное окно. Если

арифметическая операция задана с ошибкой, вывести сообщение об ошибке. Решение оформить в виде сценария с именем example_6_7.

Решение:

Листинг 6.7

```
a1 = input('Введите значения элементов первого вектора: ');
operation = input('Введите арифметическую операцию: ','s');
a2 = input('Введите значения элементов второго вектора: ');
switch operation
    case '+' % Сложение
        a3 = a1+a2; res = ['Результат: ' num2str(a3)]; disp(res)
    case '-' % Вычитание
        a3 = a1-a2; res = ['Результат: ' num2str(a3)]; disp(res)
    case '*' % Умножение
        a3 = a1.*a2; res = ['Результат: ' num2str(a3)]; disp(res)
    case '/' % Деление
        a3 = a1./a2; res = ['Результат: ' num2str(a3)]; disp(res)
    otherwise
        disp('Введена неопределенная арифметическая операция')
end
```

Результаты выполнения инструкций, представленных на листинге 6.7:

```
>> example_6_7
Введите значения элементов первого вектора: [1 3 -4 8]
Введите арифметическую операцию: +
Введите значения элементов второго вектора: [2 1 6 3]
Результат: 3 4 2 11
```

При решении примера 6.7 для ввода числовых значений и символа арифметической операции с помощью клавиатуры использовалась функция `input()`. По умолчанию эта функция возвращает значение арифметического класса. Передача второго входного параметра в виде '`s`' задает символьный класс возвращаемого значения. В качестве `switch`-выражения использовалась переменная символьного класса `operation`, значение которой сравнивалось с символами арифметических операций, расположенными на месте `case`-выражений. При совпадении значения `operation` с одним из заданных символов выполняется соответствующая этому символу группа инструкций. В противном случае выводится сообщение о неправильно заданной арифметической операции.

Пример 6.8. Используя структуру `switch`, определить класс переменной `a` и вывести соответствующее сообщение (например, «Переменная логического класса»). Решение оформить в виде сценария с именем `example_6_8`. До запуска сценария создать переменную `a`.

Решение:**Листинг 6.8**

```

switch class(a)
case 'logical'
    disp('Переменная логического класса')
case {'double','int8','uint8','int16','uint16','int32','uint32'}
    disp('Переменная арифметического класса')
case 'char'
    disp('Переменная символьного класса')
case 'struct'
    disp('Переменная класса массив структуры')
case 'cell'
    disp('Переменная класса массив ячеек')
case 'function_handle'
    disp('Дескриптор функции')
otherwise
    disp('Переменная производного класса или класса языка Java')
end

```

Результаты выполнения инструкций, представленных на листинге 6.8:

```

>> a=5; example_6_8, a='5'; example_6_8, a={5}; example_6_8
Переменная арифметического класса
Переменная символьного класса
Переменная класса массив ячеек

```

При решении примера 6.8 в качестве switch_выражения использовалась функция class(), вычисляющая класс входного аргумента и возвращающая результат в виде строки. Все case_выражения были оформлены в виде строковых констант. После второго зарезервированного слова case следует массив строковых констант, которые соответствуют названиям различных классов, образующих арифметический класс. Если результат switch_выражения совпадает со строковой константой, то в командное окно выводится соответствующее сообщение. В том случае, если ни одна из строковых констант не совпадала с результатом switch_выражения, то выполняется инструкция, расположенная после ключевого слова otherwise. Сценарий example_6_8 был запущен три раза. Перед каждым запуском сценария переменной a присваивалось значение отличного от предыдущих вариантов класса. В первом случае переменной a было присвоено число 5, во втором случае – строка, состоящая из одного символа '5', и в третьем случае – массив ячеек, состоящий из одного элемента, в котором находилось число 5. В результате получились разные сообщения, которые соответствуют текущему на момент выполнения сценария классу переменной a.

6.3.5. Структура повторения for end

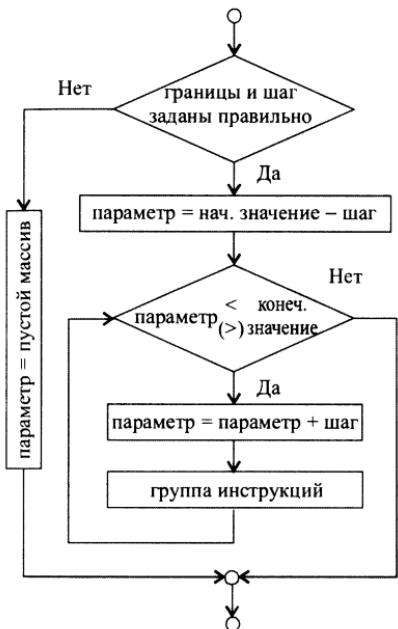
При реализации алгоритмов на языке MATLAB следует применять векторный подход, который позволяет написать программу в более компактном виде и повысить скорость ее выполнения. В большинстве случаев при использовании векторного подхода язык MATLAB позволяет выполнять математические операции с массивами данных без структур повторения, которые широко применяются в таких языках программирования, как BASIC, C, C++, FORTRAN, Object Pascal и т.д. Однако при выполнении вычислений с их последующей визуализацией возникают ситуации, когда нельзя обойтись без использования структур повторения. Например, выполнение повторяющихся инструкций при создании анимационных роликов, их демонстрации, решении многомерной задачи и т.д.

В язык MATLAB включены две структуры повторения, одной из которых является структура `for end`. Структуру повторения `for end` необходимо использовать в том случае, если заранее известно количество повторений или оно может быть вычислено. Структура повторения `for end` имеет две формы:

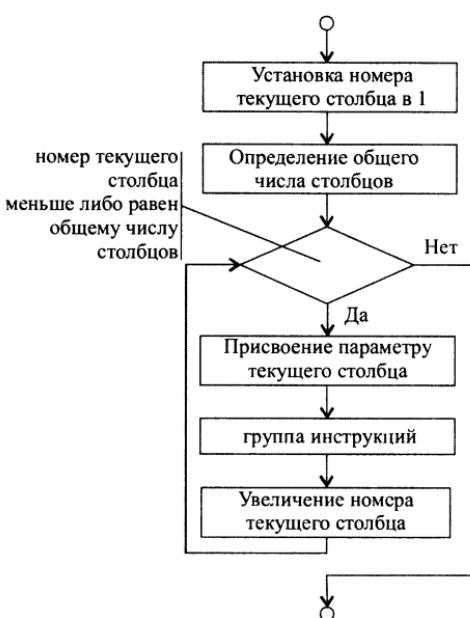
```
for параметр = начальное_значение:шаг:конечное_значение
    инструкция_1, инструкция_2, ..., инструкция_n
end
for параметр = переменная
    инструкция_1, инструкция_2, ..., инструкция_n
end
```

Параметр в первой форме является счетчиком количества повторений. *Шаг приращения* может быть как положительной, так и отрицательной величиной, а также как целым числом, так и вещественным. Если шаг равен единице, его можно не указывать. Из-за наличия параметра в структуре повторения `for end` ее называют также *циклом (loop) с параметром*. При входе в цикл проверяется правильность задания начального значения параметра, его конечного значения и шага. В том случае, если начальное значение меньше конечного и шаг отрицательный либо начальное значение больше конечного и шаг положительный, структура повтора не выполняется, а параметру цикла присваивается пустой массив. В противном случае осуществляется вход в структуру повторения. Параметру присваивается начальное значение, и выполняются все инструкции, расположенные внутри структуры повторения. Если новое значение параметра меньше (больше) его конечного значения, то происходит увеличение (уменьшение) значения параметра на величину его шага и заново выполняются все инструкции. В противном случае происходит выход из цикла. Следовательно, значение параметра после выхода из цикла равно тому значению, при котором последний раз

повторялись инструкции. Блок-схема первой формы структуры повторения `for end` изображена на рис. 6.5,а. При использовании второй формы структуры повторения `for end` число повторов равно числу столбцов в переменной. Параметр цикла является вектором-столбцом, который равен текущему столбцу переменной. Блок-схема второй формы изображена на рис. 6.5,б.



а) первая форма



б) вторая форма

Р и с . 6.5. Структура повторения `for end`

Система MATLAB предоставляет все необходимые средства для создания анимации. Используя эти средства, можно создавать анимацию двумя способами. Первый способ заключается в последовательной записи текущего состояния графического окна или координатного пространства в переменную с последующим воспроизведением сохраненных изображений произвольное количество раз. Второй способ представляет собой непрерывную прорисовку объектов в графическом окне. Первый способ следует использовать в случаях, когда графические изображения достаточно сложные и при их перерисовке появляются блики, либо в случае создания файла с расширением .avi (Audio Video Interleave, видеофильмы, синхронизованные со звуком).

ком) для последующего просмотра анимации без предварительной загрузки системы MATLAB. Второй способ позволяет использовать различные методы прорисовки графических объектов для создания дополнительных эффектов. Структуру повторения `for end` можно использовать при создании *анимации* как с помощью первого, так и с помощью второго способа. При ее использовании необходимо заранее определить количество повторов (кадров).

Методы прорисовки и удаления графических объектов задаются с помощью свойства `erasemode`. Система MATLAB предоставляет пользователю четыре метода прорисовки: `normal`, `none`, `background` и `xor`.

В первом методе, принятом по умолчанию, выполняется объемный анализ сцены с последующей точной прорисовкой всех ее объектов. Метод `normal` выполняет самую лучшую прорисовку объектов сцены и, как следствие, является самым медленным из всех. Этот метод используется во время печати.

При методе `none` во время перерисовки не происходит удаления предыдущего состояния объектов. Несмотря на то что предыдущие состояния являются видимыми на экране, система MATLAB их не сохраняет. Следовательно, при изменении размеров графического окна во время анимации или после ее завершения все предыдущие состояния удаляются из графического окна. При печати графического окна после анимации с режимом `none` будет распечатано только текущее состояние.

При методе `background` во время перерисовки удаляются не только объекты, которые перерисовываются, но и все объекты, которые располагались под ними. После удаления пустое место закрашивается цветом фона координатной плоскости или цветом графического окна, если плоскость невидима. Используя метод `background`, можно визуализировать работу ластика.

Метод `xor` наиболее широко применяется при создании анимации, так как пересовывает только те объекты, которые изменили свое положение.

Пример 6.9. Создать в графическом окне циферблат, часовую и минутную стрелки. Используя второй метод создания анимации, выполнить анимацию. Во время выполнения анимации часовая стрелка должна сделать полный оборот. Для секундной стрелки задать метод прорисовки `xor`, а для минутной – `none`. Решение оформить в виде сценария с именем `example_6_9`.

Решение:

Листинг 6.9

```
% Создание графического окна и задание его параметров
f = figure('Position',[200 200 300 300],'MenuBar','none',...
    'Name','Часы','NumberTitle','off','Color',[1 1 1]);
alpham = -6*pi/180; % угол поворота минутной стрелки в радианах
```

Окончание листинга 6.9

```

alphah = -30*pi/180; % угол поворота часовой стрелки и надписей
% построение окружности
plot(10*cos(0:pi/64:2*pi),10*sin(0:pi/64:2*pi),...
    'LineWidth',3,'Color',[0 .5 0])
hold on % включение режима добавления графиков
% оформление координатной плоскости
axis equal, axis([-12 12 -12 12]), axis off
x1 = 0; y1 = 11.5; % координаты отсчета текстовых надписей
for hours = 1:12 % цикл задания текстовых надписей
    % Вращение координат текстовой надписи
    xt = x1*cos(alphah*hours)-y1*sin(alphah*hours);
    yt = x1*sin(alphah*hours)+y1*cos(alphah*hours);
    text(xt,yt,num2str(hours),... % Создание текстовой надписи
        'HorizontalAlignment','Center','Rotation',hours*-30,...
        'FontSize',12)
end
x1 = [0 0]; y1 = [0 8]; % задание координат стрелок
hh = plot(x1,y1, ... % построение часовой стрелки
    'LineWidth',3,'Color','b', 'EraseMode','none');
hm = plot(x1,y1, ... % построение минутной стрелки
    'LineWidth',2,'Color','r','EraseMode','xor');
% анимация
for hours = 1:12 % часы
    for minutes = 1:59 % минуты
        % Вращение минутной стрелки
        xm = x1*cos(alpham*minutes)-y1*sin(alpham*minutes);
        ym = x1*sin(alpham*minutes)+y1*cos(alpham*minutes);
        set(hm,'XData',xm,'YData',ym) % установка новых значений
        pause(.025) % пауза
    end % окончание структуры повторения для минутной стрелки
    % Вращение часовой стрелки
    xh = x1*cos(alphah*hours)-y1*sin(alphah*hours);
    yh = x1*sin(alphah*hours)+y1*cos(alphah*hours);
    set(hh,'XData',xh,'YData',yh)
end % окончание структуры повторения для часовой стрелки

```

Результаты выполнения сценария в начальный и конечный моменты времени представлены на ил. 16,а и ил. 16,б соответственно. Первая структура повторения использовалась для задания текстовых надписей по окружности. Вторая структура повторения, которая выполняет анимацию, включая

ет команды для вращения часовой стрелки. Так как часовая стрелка должна поворачиваться на один час после каждого полного оборота секундной стрелки, то в состав инструкций структуры повторения для часовой стрелки включена структура повторения для секундной стрелки. Инструкции, расположенные во внутренней структуре повторения, выполняются 60 раз при каждом повторе внешней структуры повторения. Инструкции, расположенные внутри внешней структуры повторения, выполняются 12 раз для полного поворота часовой стрелки.

Во время анимации для временного прекращения выполнения инструкций используется функция `pause()`. Входным аргументом в функцию является число, которое задает паузу в секундах. В том случае, если число отсутствует, система MATLAB приостанавливает выполнение инструкций до тех пор, пока пользователь не нажмет любую клавишу. Команда `pause on` включает обработку пауз системой MATLAB, а команда `pause off` отключает этот режим. По умолчанию режим включен.

В начале анимации минутная стрелка совпадает с часовой стрелкой. При совпадении объектов во время анимации происходит смешение цветов. Минутная стрелка задана с помощью красной линии толщиной 2 пункта, а часовая стрелка – с помощью синей линии толщиной 3 пункта. Результирующим цветом в данном случае является зеленый цвет. Следовательно, минутная стрелка в начале анимации будет изображаться зеленым цветом. После окончания анимации все положения часовой стрелки, в которых она находилась во время анимации, будут отображены в графическом окне. Данная особенность является следствием выбранного для часовой стрелки метода прорисовки `none`, при котором сохраняются все предыдущие состояния. Так как для минутной стрелки был выбран метод `xog`, то в графическом окне отображается только ее текущее состояние. Удалить предыдущие состояния часовой стрелки можно с помощью изменения размеров графического окна.

Пример 6.10. Решить с помощью метода Гаусса с частичным выбором главного элемента следующие системы линейных алгебраических уравнений:

$$\begin{cases} 3 \cdot x_1 + 2 \cdot x_2 - x_3 = 4, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 9, \\ x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3 \end{cases} \text{ и } \begin{cases} 3 \cdot x_1 + 2 \cdot x_2 - x_3 = 9, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 3, \\ x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 4. \end{cases}$$

Переменную A использовать для матрицы коэффициентов при неизвестных, переменную B – для матрицы свободных членов, переменную AB – для расширенной матрицы и x – для матрицы неизвестных. Метод Гаусса с частичным выбором главного элемента реализовать в виде сценария с именем `example_6_10`.

Решение:**Листинг 6.10**

```
% Решение СЛАУ методом Гаусса с частичным выбором главного элемента
AB = [A B]; % объединение массивов
row_count = size(AB,1); % число строк = числу неизвестных
col_count_A = size(A,2); % число столбцов в матрице A
col_count_B = size(B,2); % число столбцов в матрице B
row_ones = ones(1,col_count_B); % строка единиц
x = zeros(col_count_A,col_count_B); % выделение памяти под x
% прямой ход
for col = 1:col_count_A-1 % шаг равен +1
    [absmax, index] = max(abs(AB(:,end,col)));
    if index > 1 % перестановка строк
        AB([index+col-1 col],:) = AB([col index+col-1],:);
    end
    AB(:,col+1:end) = AB(:,col+1:end)-...
        (AB(:,col+1:end,col)./AB(:,col,col))*AB(:,col:end);
end
% обратный ход
for row = row_count:-1:1
    if row == row_count
        % Вычисление последнего неизвестного
        x(:,row) = AB(:,col_count_A+1:end)./AB(:,row);
    elseif row == row_count-1
        % Вычисление предпоследнего неизвестного
        x(:,row) = (AB(:,row+1:end)) - ...
            AB(:,row+1).*x(:,row+1,:)) ./ AB(:,row);
    else
        % Вычисление остальных неизвестных
        x(:,row) = (AB(:,row+1:end)) - ...
            dot(AB(:,row+1:col_count_A)'*row_ones, ...
                x(:,row+1:end,:)))./AB(:,row);
    end
end
```

Результаты выполнения сценария example_6_10:

```
>> A = [3 2 -1; 2 -1 3; 1 -2 2]; B = [[4; 9; 3] [9; 3; 4]];
>> example_6_10
>> x
x =
    1.0000    3.8462
    2.0000   -2.4615
    3.0000   -2.3846
```

```
>> A*x  
ans =  
    4.0000    9.0000  
    9.0000    3.0000  
    3.0000    4.0000
```

Структура повторения используется в данном случае наряду с векторными операциями, так как во время выполнения прямого и обратного хода требуются «двухмерные вычисления». При использовании приведенных в листинге 6.10 инструкций необходимо, чтобы определитель матрицы коэффициентов при неизвестных не был равен нулю. В противном случае результат будет неверным.

6.3.6. Структура повторения while end

Помимо структуры повторения `for end` язык MATLAB содержит еще одну структуру повторения – `while end`. Ее отличительной особенностью является то, что инструкции, расположенные в теле структуры повторения, начинают выполняться и повторно выполняются только в том случае, если некоторое условие истинно. Как только условие становится ложным, происходит выход из структуры повторения. Если условие ложно во время входа в структуру повторения, то управление передается на инструкцию, расположенную после ключевого слова `end`. Таким образом, структуру повторения `while end` можно использовать в случаях, когда заранее неизвестно число повторений. Структура повторения `while end` имеет следующие формы:

<code>while</code>	<code>логическое_выражение</code>	<code> </code>	<code>while</code>	<code>переменная</code>	<code> </code>	<code>while</code>	<code>число</code>
	<code>инструкция_1</code>	<code> </code>		<code>инструкция_1</code>	<code> </code>		<code>инструкция_1</code>
	<code>инструкция_2</code>	<code> </code>		<code>инструкция_2</code>	<code> </code>		<code>инструкция_2</code>
	<code>.....</code>	<code> </code>		<code>.....</code>	<code> </code>		<code>.....</code>
	<code>инструкция_n</code>	<code> </code>		<code>инструкция_n</code>	<code> </code>		<code>инструкция_n</code>
<code>end</code>		<code> </code>	<code>end</code>		<code> </code>	<code>end</code>	

При логическом выражении инструкции выполняются в том случае, если результатом вычисления логического выражения является истина. При использовании переменной, которая может быть логического класса, арифметического класса двойной точности или символьного класса, инструкции выполняются в случае, если переменная не содержит пустой массив или элементы с нулевыми значениями. Числовое значение, отличное от нуля, приводит к бесконечному выполнению инструкций («зацикливанию»). Прекратить выполнение инструкций в данном случае можно только с помощью комбинации клавиш `Ctrl+C`. Применение бесконечного цикла может понадобиться при создании анимации. Для избежания «зацикливания» в двух первых случаях необходимо влиять на результат логического выражения или

значений элементов переменной с помощью инструкций внутри тела структуры повторения. Блок-схема структуры повторения `while end` для всех форм представлена на рис. 6.6.



Рис. 6.6. Блок-схема структуры `while end`

Пример 6.11. Найти наибольший общий делитель соответствующих элементов двух массивов. Решение оформить в виде сценария с именем `example_6_11`.

Решение:

Листинг 6.11

```

% Определение наибольшего общего делителя двух положительных чисел
% Задание начальных значений делимого, делителя и остатка
num = abs(m); den = abs(n); remainder=ones(size(m));
while any(remainder(:))
    l = (remainder > 0); % значения для лог. индексирования
    remainder(l) = rem(num(l),den(l)); % вычисление остатка
    num(l) = den(l);
    den(l) = remainder(l);
end
num % Вывод в командное окно наибольшего общего делителя

```

Результаты выполнения сценария `example_6_11`:

```

>> m = [52 2166; 27 20];
>> n = [133 6099; 36 45];
>> example_6_11
num =
     1      57
     9      5

```

Поиск наибольших общих делителей соответствующих элементов двух массивов был реализован с помощью алгоритма Евклида. Во время их поиска выполняются следующие действия:

- Определение элементов массивов делимых и делителей, которые используются при вычислении остатков и выполнении операций присвоения.
- Вычисление остатков от поэлементного деления массива делимых на массив делителей и присвоение их соответствующим элементам массива остатков.
- Поэлементное присвоение элементам массива делимых значений элементов массива делителей.
- Поэлементное присвоение элементам массива делителей значений элементов массива остатков.
- Проверка на равенство нулю всех элементов массива остатков. Если все остатки равны нулю, то завершается вычисление наибольших общих делителей.

Так как заранее неизвестно, сколько потребуется повторов для вычисления наибольших общих делителей элементов двух массивов, то следует использовать структуру повторения `while end`. Функция `any()` возвращает логическую единицу, если хотя бы один элемент отличен от нуля. Следовательно, инструкции, расположенные в теле структуры повторения `while end`, будут выполняться до тех пор, пока все элементы массива `reminder` не будут равны нулю.

С целью избежания дальнейшего поиска наибольшего общего делителя для тех элементов исходных массивов, соответствующие элементы массива остатков которых равны нулю, используется логическое индексирование.

Пример 6.12. Создать анимацию, демонстрирующую вращение спутника вокруг планеты. Планета вращается вокруг Солнца с постоянным углом поворота $-2\pi/180$. В графическом окне изобразить траекторию вращения спутника при следующих углах его поворота: а) $-3\pi/180$; б) $3\pi/180$; в) $-5\pi/180$; г) $5\pi/180$; д) $-\pi/20$; е) $9\pi/180$. Решение оформить в виде сценария `example_6_12`.

Решение:

Листинг 6.12

```
% создание графического окна и задание его свойств
figure('Position', [200 200 300 300], 'Name', 'Анимация',...
    'NumberTitle', 'off', 'Color', 'w')
% создание солнца
hsn = plot(0,0,'Marker','o','MarkerSize',40,...%
    'MarkerEdgeColor','y','MarkerFaceColor','r','EraseMode','none');
hold on % режим добавления графиков
```

Окончание листинга 6.12

```
% создание планеты
planet_r = 9; planet_phi = 0; % начальные координаты планеты
x_planet = planet_r*cos(planet_phi);
y_planet = planet_r*sin(planet_phi);
planet_alpha = -2*pi/180; % угловое перемещение планеты
hp = plot(x_planet,y_planet,'Marker','o',...
    'MarkerSize',15,'MarkerFaceColor','b','EraseMode','xor');
% создание спутника
sattle_r = 2; sattle_phi = 0; % начальные координаты спутника
x_sattle = x_planet + sattle_r*cos(sattle_phi);
y_sattle = y_planet + sattle_r*sin(sattle_phi);
sattle_alpha = -4*pi/180; % угловое перемещение спутника
hst = plot(x_sattle,y_sattle,'Marker','o',...
    'MarkerSize',5,'MarkerFaceColor','g','EraseMode','none');
% Оформление координатных осей
axis([-13 13 -13 13]), axis equal, axis off
while 1
    % Перемещение планеты
    planet_phi = planet_phi + planet_alpha;
    x_planet = planet_r*cos(planet_phi);
    y_planet = planet_r*sin(planet_phi);
    % Перемещение спутника
    sattle_phi = sattle_phi + sattle_alpha;
    x_sattle = x_planet + sattle_r*cos(sattle_phi);
    y_sattle = y_planet + sattle_r*sin(sattle_phi);
    % визуализация нового положения планеты
    set(hp,'XData',x_planet,'YData',y_planet)
    set(hst,'XData',x_sattle,'YData',y_sattle)
    pause(0.05)
end
```

Результаты выполнения инструкций, представленных на листинге 6.11, при разных углах поворота спутника изображены на ил. 17.

6.3.7. Функции break() и continue()

Функции `break()` и `continue()` используются для изменения порядка выполнения инструкций в теле структур повторения `for` `end` и `while` `end`.

При выполнении функции `break()` происходит выход из структуры повторения. После выхода из структуры повторения выполняется следую-

щая за ней инструкция. В том случае, если функция `break()` вызывается из вложенной структуры повторения, то выполняется выход из текущей структуры повторения с переходом на один уровень вложенности наверх. Следовательно, назначение функции `break()` – досрочно прервать выполнение инструкций внутри структуры повторения и осуществить выход из текущей структуры.

Пример 6.13. Написать программу, преобразующую массив чисел, которые заданы в десятичной системе счисления, в массив чисел римской системы счисления. Решение оформить в виде сценария `example_6_13`.

Решение:

Листинг 6.13

```
[r c] = size(decimal); % Определение размерности массива
dec = decimal(:); % преобразование в вектор-столбец
% Массив римских чисел
romans = {'I', 'IV', 'V', 'IX', 'X', 'XL', 'L', 'XC', 'C', 'CD', 'D', 'CM', 'M'};
% Массив арабских чисел
arabics = [1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000];
% удаление переменной rom из памяти, если она там есть
if exist('rom'), clear rom, end
% Выделение места под массив чисел в римской системе счисления
rom(1:size(dec,1),1) = {};
% Преобразование массива чисел
for row = 1:size(dec,1) % повторения для каждого числа в массиве
    for column = size(arabics,2):-1:1 % перебор арабских чисел
        while (dec(row) >= arabics(column))
            rom{row} = [rom{row} romans{column}]; % добавление символа
            dec(row) = dec(row) - arabics(column); % уменьшение числа
        end
        if dec(row) == 0, break, end % выход из цикла for
    end
end
rom = reshape(rom, r, c); % Преобразование размерностей
rom % Вывод результата в командное окно
```

Результаты выполнения инструкций, представленных на листинге 6.13:

```
>> decimal = [8 500; 1999 23];
>> example_6_13
rom =
    'VIII'      'D'
    'MCMXCIX'   'XXIII'
```

Вызов функции `break()` прекращает выполнение структуры повторения `for end`, в которой происходит перебор чисел из массива `arabics`. В этом массиве содержатся числа в десятичной системе счисления, которые соответствуют числам в римской системе счисления, находящимся в массиве `romans`. В том случае, если исходное число преобразовано из десятичной в римскую систему счисления, а перебор чисел в массиве не закончился, то выполняется выход из вложенной структуры повторения `for end`. В рассмотренном примере инструкции во вложенной структуре повторения выполняются 15, 3, 10 и 16 раз для преобразования чисел 8, 500, 1999 и 23 соответственно. Аналогичный результат работы программы можно получить, если заменить структуру повторения `for end` и функцию `break()` на структуру повторения `while end` с двумя условиями, связанными логической операцией «И»:

```
for row = 1:size(dec,1)
    column = size(arabics,2);
    while ((dec(row)>0)&&(column > 0))
        while (dec(row) >= arabics(column))
            rom{row} = [rom{row} romans{column}];
            dec(row) = dec(row) - arabics(column);
        end
        column = column - 1;
    end
end
```

Вызов функции `continue()` сопровождается пропуском оставшихся инструкций в теле структуры повторения и переходом к новой итерации, если она допустима. В противном случае выполняется выход из структуры повторения.

Пример 6.14. Написать программу, которая подсчитывает количество введенных в командную строку простых чисел. Ввод чисел реализовать с помощью структуры повторения `while end` и функции `input()`. При вводе отрицательного или нулевого значения выполнить выход из цикла с последующим выводом количества введенных простых чисел. Если введенное число не является простым числом, то перейти с помощью функции `continue()` к вводу нового числа. В противном случае вывести сообщение, что введенное число является простым.

Решение:

Листинг 6.14

```
number = 0;
while 1
    a = input('Введите положительное число (0 - выход) :');
```

```

if a <= 0
    break % выход из цикла
end
if ~isprime(a)
    continue % переход к вводу нового числа
end
s = ['Вы ввели простое число']; disp(s)
number = number + 1;
end
s = ['Было введено ' num2str(number) ' простых чисел']; disp(s)

```

Результаты выполнения инструкций, представленных на листинге 6.14:

```

>> example_6_14
Введите положительное число (0 - выход):2
Вы ввели простое число
Введите положительное число (0 - выход):3
Вы ввели простое число
Введите положительное число (0 - выход):4
Введите положительное число (0 - выход):0
Было введено 2 простых числа

```

Аналогичного результата можно добиться, используя элементы структурного программирования:

```

number = 0;
a = 1;
while (a>0)
    a = input('Введите положительное число (0 - выход):');
    if isprime(a)
        s = ['Вы ввели простое число'];
        disp(s)
        number = number + 1;
    end
end
s = ['Было введено ' num2str(number) ' простых числа']; disp(s)

```

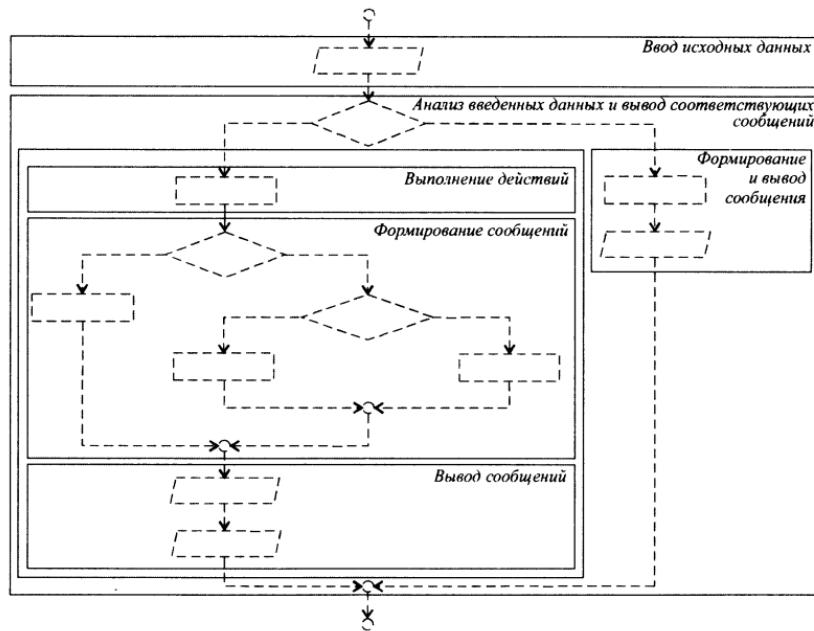
Использование функций `break()` и `continue()` приводит к нарушению стиля структурного программирования. Результаты вызова этих функций могут быть достигнуты с помощью конструкций структурного программирования, что приводит к улучшению читаемости программы. Однако вызовы функций с последующим выходом из структуры или переходом к новой итерации выполняются быстрее, чем соответствующие приемы структурного программирования.

6.3.8. Заключение по структурному программированию

Во время разработки программ, решающих математические задачи с помощью языка программирования MATLAB, следует применять правила структурного стиля программирования:

1. Разработка программы начинается с простейшей блок-схемы.
2. Каждый прямоугольник (действие) может быть заменен двумя последовательными прямоугольниками (более детальными действиями). Правило называется *правилом пакетирования*.
3. Каждый прямоугольник (действие) может быть заменен любой структурой управления `if end`, `if else end`, `if elseif else end`, `switch end`, `for end` и `while end`. Правило носит название *правила вложения*.
4. Правила 2 и 3 могут применяться неограниченно и в любом порядке.

На рис. 6.7 показано применение правил структурного программирования при решении примера 6.6.



Р и с . 6.7. Применение правил структурного программирования

Разработка программы начинается с двух блоков, расположенных на первом уровне иерархии: ввод исходных данных и их анализ с последующим выводом соответствующих сообщений. Так как первый блок состоит из одной инструкции, то на втором шаге с помощью правила вложения детализируются действия второго блока. Этот блок содержит структуру управления

`if else end`. В каждой из ветвей формируется соответствующее сообщение с последующим его выводом в командное окно. Третий шаг заключается в детализации действий в одной из двух ветвей. Левая ветвь (результат логического выражения есть истина) с помощью правила пакетирования заменяется на три последовательно расположенных блока. В этих блоках осуществляется выполнение необходимых действий, формирование сообщения и вывод результатов. При необходимости с помощью правил пакетирования или вложения блок выполнения действий может быть заменен блоками или конструкциями соответственно. Допустим, что блок описывает одну инструкцию. Следовательно, дальнейшая детализация не имеет смысла. На четвертом шаге блок формирования сообщения с помощью правила вложения заменяется структурой `if elseif else end`. Каждая из трех ветвей состоит из одной инструкции, которая формирует соответствующее сообщение. Блок вывода результатов с помощью правила пакетирования заменяется двумя блоками. Первый блок соответствует инструкции, которая выводит сформированные сообщения в командное окно, а второй блок – инструкции, которая выводит введенные простые числа. На этом заканчивается пятый шаг и декомпозиция левой ветви структуры выбора. Шестой шаг заключается в детализации правой ветви (результат логического выражения есть ложь), которая состоит из одного блока, осуществляющего формирование и вывод сообщения в командное окно. С помощью правила пакетирования блок заменяется двумя блоками, которые выполняют формирование и вывод сообщения соответственно. Блоки представляют собой простые инструкции. На этом заканчивается декомпозиция правой ветви, второго блока первого уровня иерархии и всей задачи в целом. Рассмотренный способ не является единственным решением задачи. Начать детализацию можно с трех блоков: ввод, анализ и вывод. Самостоятельно продолжите декомпозицию задачи для этого варианта.

Итальянские ученые Бом и Джакопини доказали, что при разработке программ с применением структурного стиля программирования достаточно использовать три формы управления: следование, выбор и повторение.

Следование подразумевает построчное выполнение инструкций. В том случае, если инструкции размещаются на одной строке, они выполняются последовательно слева направо. Разделителями инструкций в языке MATLAB служат запятая и точка с запятой. Точка с запятой используется для подавления вывода результата выполнения инструкции на экран.

Выбор осуществляется одним из четырех способов:

`if end` – структура единственного выбора;

`if else end` – структура двойного выбора;

`if elseif else end` – структура тройного выбора;

`switch end` – структура множественного выбора.

Для реализации любой формы выбора достаточно одной структуры `if end`. Однако использование других структур выбора позволяет сделать программу более читабельной и легче изменяемой впоследствии.

При написании программ на языке MATLAB для реализации выбора следует использовать логическое индексирование. Применение логического индексирования приводит к выигрышу в скорости выполнения инструкций по сравнению со структурами выбора, которые свойственны языкам третьего поколения. Структуры выбора при программировании на языке MATLAB следует использовать при вводе или выводе данных в командное окно.

Повторение обеспечивается одним из двух способов:

`for end` – структура повторения с параметром;

`while end` – структура повторения с условием.

Для реализации двух форм повторения достаточно одной структуры `while end`.

При написании программ на языке программирования MATLAB следует по возможности избегать использования структур повторения. Решить целый ряд математических задач без использования структур повторения можно с помощью применения операции внешнего произведения.

6.4. ФУНКЦИИ

Система MATLAB содержит большое количество функций, позволяющих решать задачи из различных областей математики и представлять результаты как в текстовом, так и в графическом виде. В систему MATLAB также входит ряд сервисных функций, которые обеспечивают связь с операционной системой и позволяют выполнять некоторые действия на ее уровне (например, создание, перемещение и удаление директории). Несмотря на огромное количество стандартных и встроенных функций, возникают ситуации, когда необходимо создавать новые функции для решения задач в специализированных областях. Язык MATLAB предоставляет все необходимые средства для их создания. Функции являются единственным видом подпрограмм в языке MATLAB, которые могут работать как с глобальными, так и с локальными данными. С подпрограммами связаны следующие понятия: описание подпрограммы, ее вызов, локальные и статические переменные, а также параметры подпрограммы.

6.4.1. Описание функции

Описание функции представляет собой совокупность действий, которые активизируются при ее вызове. В общем случае функция состоит из заголовка, строки быстрой помощи, строк полного описания назначения и тела

функции. Пример оформления функции представлен на листинге 6.15. Приведенная функция предназначена для преобразования входного массива целых положительных чисел, записанных в десятичной системе счисления и лежащих в диапазоне от 1 до 3999, в массив соответствующих чисел, записанных в римской системе счисления.

Листинг 6.15

```
function [roman] = dec2rom(decimal)
%DEC2ROM преобразование десятичных чисел в римские числа
% decimal - входной параметр, содержащий массив десятичных чисел.
% Значения элементов массива должны лежать в диапазоне [1;3999].
% roman - выходной параметр, содержащий массив римских чисел

% Автор: А.В. Кривилев
% Версия: 1.0. Дата: 10.02.2004

% ===== Проверка значений входного аргумента ===== %
% Входной параметр является пустым массивом
if isempty(decimal), roman = ''; return, end
% Проверка класса входного параметра
if ~isnumeric(decimal)
    error('Входной параметр должен быть арифметического класса')
end
% Проверка диапазона
dec = decimal(:); % преобразование в вектор-столбец
if any((dec > 3999) | (dec < 1))
    error('Значение одного из элементов массива лежит вне диапазона')
end
% Являются ли элементы входного массива целыми числами
if find(dec ~= fix(dec))
    error('Элементы входного массива должны быть целыми числами')
end

% ===== Выполнение преобразования ===== %
[r c] = size(decimal); % Определение размера исходного массива
% Формирование массива римских чисел
romans = {'I','IV','V','IX','X','XL','L','XC','C','CD','D','CM','M'};
% Формирование массива арабских чисел
arabics = [1,4,5,9,10,40,50,90,100,400,500,900,1000];
% Определение количества элементов в выходном массиве
roman(1:size(dec,1),1) = {[[]};
```

Окончание листинга 6.15

```
% Преобразование массива чисел
for row = 1:size(dec,1) % повторения для каждого числа в массиве
    column = size(arabics,2);
    while ((dec(row)>0)&&(column > 0))
        while (dec(row) >= arabics(column))
            roman{row} = [roman{row} romans{column}]; % добавление символа
            dec(row) = dec(row) - arabics(column); % уменьшение числа
        end
        column = column - 1;
    end
end
roman = reshape(roman,r,c); % Преобразование размера
```

Результаты вызова функции, представленной на листинге 6.15:

```
>> d = [8 500; 1999 23];
>> rom = dec2rom(d)
rom =
    'VIII'          'D'
    'MCMXCIX'       'XXIII'
```

Первая строка на листинге 6.15 представляет собой *строку заголовка функции*. Стока заголовка функции начинается с ключевого слова `function`. После ключевого слова `function` следует перечисление выходных параметров, заключенных в квадратные скобки. Выходные параметры перечисляются через запятую. В данном случае функция `dec2rom()` возвращает результат с помощью одного выходного параметра – `roman`, в котором находится массив чисел, записанных в римской системе счисления. После перечисления выходных параметров следует знак равенства, за которым располагается имя функции и заключенный в круглые скобки список входных параметров. При задании имени функции необходимо следовать требованиям, которые предъявляются к заданию имени переменной: различаются первые 63 символа в имени, имя должно начинаться с буквы и содержать любую комбинацию латинских букв, знаков подчеркивания и цифр. Длина различаемых символов в имени функции зависит от применяемой операционной системы. Узнать максимальную длину строки символов, которые различаются системой MATLAB, можно с помощью команды `>> namelengthmax`. Несмотря на то что имя функции может отличаться от имени файла, в котором она описана, настоятельно рекомендуется давать одинаковые имена функции и файлу. Входные параметры перечисляются через запятую. Список входных параметров в рассматриваемом примере состоит из одного параметра, с помощью которого в функцию передается массив чисел, записанных в десятичной системе счисления.

Вторая строка является *строкой быстрой помощи* (H1 line), содержащей краткое описание функции. Эта строка начинается с символа % и представляет собой комментарий. Как правило, за знаком комментария располагается имя функции, записанное в верхнем регистре. Несмотря на то что имена функций в строке быстрой помощи принято записывать в верхнем регистре, вызов функции следует осуществлять заданием ее имени в нижнем регистре. После имени функции идет ее краткое описание. Эта строка просматривается во время поиска, заданного с помощью команды >> lookfor критерии поиска. Например,

```
>> lookfor dec2
DEC2ROM Преобразование десятичных чисел в римские числа
DEC2BASE Convert decimal integer to base B string.
DEC2BIN Convert decimal integer to a binary string.
DEC2HEX Convert decimal integer to hexadecimal string.
DEC2BINVEC Convert decimal number to a binary vector.
DEC2OCT Convert nonnegative decimal numbers to octal numbers.
```

Следовательно, после задания команды `lookfor` можно получить информацию о всех функциях, в строках быстрой помощи которых встречается заданная последовательность символов. При задании команды `lookfor` просматриваются файлы, которые расположены в директориях, указанных в пути поиска системы MATLAB (search path).

Строки, располагающиеся после строки первой помощи до первой пустой строки или первой инструкции, называются *строками помощи* (help text). Каждая из этих строк является комментарием, так как начинается с символа %. Строки помощи вместе со строкой быстрой помощи выводятся в ответ на команду `help имя_функция`:

```
>> help dec2rom
DEC2ROM преобразование десятичных чисел в римские числа
decimal - входной параметр, содержащий массив десятичных чисел.
Значения элементов массива должны лежать в диапазоне [1;3999].
roman - выходной параметр, содержащий массив римских чисел
```

Как правило, в строках помощи описывается интерфейсная часть функции: имена, классы, ограничения и назначение соответствующих входных и выходных параметров. В ряде случаев приводятся результаты вызовов функции с различными значениями некоторых входных параметров (например, `sum()`), а также ссылки на имена функций, в описании которых можно найти дополнительную информацию (see also PROD, CUMSUM, DIFF).

После пустой строки, которая завершает строки помощи, задаются *реквизиты функции*: имена разработчиков, название организации, в которой она создавалась, а также номер версии и дата ее создания. Эти строки являются комментариями, так как начинаются с символа %.

После заголовка функции, строки быстрой помощи, строк помощи и реквизитов функции идет *тело функции*. Как правило, тело функции состоит из двух частей. В первой части осуществляется проверка количества, классов и значений входных параметров, а также числа выходных параметров. В том случае, если во время проверки обнаружена комбинация входных данных, которая может привести к неправильному результату или возникновению исключительной ситуации, выполнение функции необходимо прекратить и вывести в командное окно сообщение, где указываются причины прекращения ее выполнения. Во второй части тела функции задаются инструкции, которые необходимы для вычисления значений выходных аргументов на основе входных данных, т.е. во второй части тела функции описывается алгоритм.

В рассматриваемом примере выполняется четыре вида проверки входного аргумента функции. Все виды проверок реализованы с помощью структуры выбора `if end`. Сначала выполняется проверка того, является ли входной параметр пустым массивом. Если входной параметр – пустой массив, то выходному параметру присваивается пустой массив и выполнение функции завершается:

```
>> decimal = []; r = dec2rom(decimal)
r =
    ''
```

Досрочное завершение выполнения функции в этом случае реализовано с помощью инструкции `return`, которая осуществляет передачу управления в область, из которой она вызывалась. В данном случае вызов функции был выполнен из командного окна.

С помощью второй структуры выбора `if end` проверяется класс входного параметра. Если входной параметр не арифметического класса, то завершается выполнение функции с выводом в командное окно сообщения об ошибке:

```
>> d = 'string'; answer = dec2rom(d)
??? Error using ==> dec2rom
Входной параметр должен быть арифметического класса
```

Прекращение выполнения функции с выводом сообщения о неправильном задании входного параметра осуществляется с помощью вызова функции `error()`, входным параметром которой является выводимое в командное окно сообщение.

Третья структура выбора `if end` используется для проверки значений элементов во входном массиве. Если значение хотя бы одного элемента в этом массиве лежит вне диапазона [1;3999], то выполняется выход из функции с выводом соответствующего сообщения в командное окно.

```
>> decimal = [8 500; 4999 23]; answer = dec2rom(decimal)
??? Error using ==> dec2rom
Значение одного из элементов массива лежит вне диапазона
```

Границы диапазона соответствуют минимальному и максимальному числу в римской системе счисления.

Четвертая структура выбора if end служит для проверки того, являются ли элементы входного массива целыми числами. Если значение хотя бы одного элемента в массиве отличается от целого значения, то выполнение функции прекращается и в командное окно выводится соответствующее сообщение:

```
>> decimal = [8.5 500; 2999 23]; answer = dec2rom(decimal)
??? Error using ==> dec2rom
Элементы входного массива должны быть целыми числами
```

В теле функции вместе с инструкциями следует использовать комментарии, которые начинаются с символа % и заканчиваются концом строки. Комментарии могут размещаться как на отдельных строках, так и вместе с инструкциями. Во втором случае инструкции должны располагаться перед комментарием. Комментарии позволяют документировать инструкции, что приводит к улучшению читабельности функции. Использование комментариев в программе считается хорошим стилем программирования.

6.4.2. Вызов функции

Вызов функции наряду с ее описанием является основной характеристикой функции. Вызов функции используется во всех случаях, где используется переменная. Исключением является случай, когда переменная располагается слева от знака присваивания.

В языке MATLAB существуют два формата вызова функции: *функциональный* и *командный*. Оба формата вызова функции используются как в т-файлах, так и в командной строке.

Функциональный формат является наиболее распространенным и аналогичен вызову функций в языках программирования высокого уровня третьего поколения. Отличительной особенностью функционального формата в языке MATLAB является возможность задания нескольких выходных аргументов. Вызов функции в функциональном формате имеет следующие виды:

```
имя_функции(вход_1, вход_2, ..., вход_n)
имя_переменной = имя_функции(вход_1, вход_2, ..., вход_n)
[выход_1, выход_2, ..., выход_m] = имя_функции(вход_1, вход_2, ..., вход_n)
```

В первом случае результат, возвращаемый функцией, присваивается переменной ans, во втором случае – переменной, имя которой располагается слева от знака присваивания. В третьем случае значения, возвращаемые

функцией, присваиваются соответствующим переменным, список которых задается слева от знака присваивания.

Пример 6.15. Написать функцию angles2(), вычисляющую углы наклона заданных векторов к координатным осям. Входными аргументами в функцию являются координаты векторов по трем осям – x, y и z, выходными аргументами – углы наклона alpha, beta и gamma к соответствующим осям. Углы наклона необходимо возвращать в градусах.

Решение:

Листинг 6.16

```
function [alpha,beta,gamma] = angles2(x,y,z)
%ANGLES2 вычисление углов наклона векторов к координатным осям
% x, y и z координаты векторов
% alpha, beta и gamma - углы между векторами и соответствующими осями

% Проверка входных аргументов отсутствует
[m n] = size(x); % Определение размера исходных массивов
x=x(:)'; y=y(:)'; z=z(:)'; % Преобразование в одномерные массивы
length = sqrt(x.*x + y.*y + z.*z); % Вычисление длин векторов
alpha = acos(x./length)*180/pi; % угол alpha
beta = acos(y./length)*180/pi; % угол beta
gamma = acos(z./length)*180/pi; % угол gamma
% Преобразование размера
alpha = reshape(alpha,m,n);
beta = reshape(beta,m,n);
gamma = reshape(gamma,m,n);
```

Варианты вызова функции angles():

```
>> x = 3; y = 0; z = 1; [a b c] = angles2(x,y,z)
a =
    18.4349
b =
    90
c =
    71.5651
>> angles2(x,y,z)
ans =
    18.4349
>> a = angles2(x,y,z)
a =
    18.4349
>> [a b] = angles2(x,y,z)
a =
    18.4349
```

```

b =
    90
>> x=[1 2 3; 4 5 6]; y=[2 -1 0; 0 -1 2]; z=[-3 -2 -1; 0 0 0];
>> [a b c] = angles2(x,y,z)
a =
    74.4986    48.1897    18.4349
            0    11.3099    18.4349
b =
    57.6885   109.4712    90.0000
    90.0000   101.3099    71.5651
c =
   143.3008   131.8103   108.4349
   90.0000    90.0000    90.0000

```

В первом варианте вызова (`[a b c] = angles2(x, y, z)`) используются три входных параметра и три выходных параметра. Во время вызова функции значения входных параметров передаются в функцию. После окончания выполнения всех необходимых инструкций функция возвращает три значения, которые записываются в переменные `a`, `b` и `c`. В том случае, если переменные `a`, `b` и `c` не были определены в рабочем пространстве MATLAB, они создаются, и в них записываются соответствующие значения. Если переменные `a`, `b` и `c` уже содержали произвольные значения до вызова функции, то эти значения удаляются и на их место записываются новые. Во время присваивания переменным новых значений при необходимости меняется класс переменной.

Во втором варианте (`angles2(x, y, z)`) функция `angles2()` вызывается с тремя входными параметрами, но без указания выходных параметров. В этом случае, как и в предыдущем варианте вызова функции, будут выполнены все необходимые инструкции, т.е. вычислены все выходные параметры, указанные в описании функции. Однако функция возвратит одно значение, которое запишется в переменную `ans`. В переменную `ans` записывается значение выходного аргумента, имя которого первым указано в списке выходных аргументов функции. Переменная `ans` применяется по умолчанию системой MATLAB, если не указаны выходные аргументы. Замечания по созданию и присваиванию значения переменной `ans` аналогичны замечаниям по созданию и присваиванию значений переменным `a`, `b` и `c` из предыдущего варианта.

Третий вариант вызова (`a=angles2(x, y, z)`) эквивалентен второму варианту ее вызова. Отличие заключается в том, что результат в этом случае присваивается не переменной `ans`, а переменной `a`.

В четвертом варианте вызова (`[a b] = angles2(x, y, z)`) используются три входных параметра и два выходных параметра. В этом случае функция принимает три значения, а возвращает два значения. В переменную `a`

записывается значение первого выходного параметра (*alpha*) в списке выходных параметров, а в переменную *b* – значение второго выходного параметра (*beta*). Третий выходной параметр (*gamma*) вычисляется, но не возвращается функцией, так как при ее вызове указываются только два выходных параметра.

Пятый вариант вызова функции *angle2()* соответствует первому варианту, но в этом случае в функцию передаются координаты векторов, заданных в виде матриц. В данном случае с помощью одного вызова функции были вычислены углы наклона шести векторов к соответствующим координатным осям. При разработке собственных функций следует использовать возможности языка MATLAB, который позволяет оперировать с матрицами. Использование матричной записи позволяет в большинстве случаев исключить структуры повторения. Если использовать скалярный подход при вычислении углов наклона шести векторов к координатным осям, то следует 6 раз вызывать функцию с координатами соответствующего вектора.

При вычислении углов наклона векторов к соответствующим координатным осям вызов функции *acos()* выполнялся внутри выражений. При вызове функции, расположенной внутри выражения, результатом ее выполнения является массив чисел, который используется для дальнейшего вычисления выражения.

Командный формат вызова функций применяется сравнительно редко, так как обладает рядом ограничений. При командном вызове функция не возвращает значения, так как выходные параметры отсутствуют. Входными параметрами могут быть только массивы символов. Вызов функции в командном формате имеет следующий вид:

```
имя_функции вход_1 вход_2 ... вход_n
```

При командном вызове функции ее имя и входные параметры отделяются друг от друга пробелами. Командный формат вызова функции используется при удалении переменных из рабочей области с помощью функции *clear()*.

Пример 6.16. Используя командный формат вызова функции *clear()*, удалите из рабочей области переменные *a*, *b* и *c*, созданные в предыдущем примере.

Решение:

```
>> clear a b c
```

После окончания выполнения функции *clear()* переменные *a*, *b* и *c* будут удалены из рабочей области системы MATLAB. Таких же результатов можно было добиться с помощью функционального вызова функции *clear()* со строковыми входными параметрами: *clear('a','b','c')*.

Следовательно, при командном формате в функцию передаются не значения переменных, а их имена. В том случае, если на месте переменных при командном вызове функции располагаются числа, то они преобразуются в строковые символы.

Во время первого вызова функции система MATLAB преобразует исходный m-файл, в котором располагается функция, в псевдокод, загружает его в память и начинает его выполнение. При преобразовании в псевдокод происходит синтаксическая проверка, приводящая к увеличению времени выполнения инструкции, в которой имеется вызов функции. После завершения работы функции псевдокод остается в памяти с целью исключения потери времени на повторные преобразования при следующих ее вызовах. Функция в виде псевдокода остается в памяти до тех пор, пока явно не будет удалена из нее с помощью вызова функции `clear()` или пока не завершится сеанс работы с системой MATLAB. В системе MATLAB существует возможность исключить потерю времени на преобразование функции в псевдокод при первом ее вызове. Для этого необходимо заранее преобразовать m-файл, в котором содержится описание функции, в псевдокод. Преобразование m-файла, содержащего описание функции, в р-файл, в котором располагается ее псевдокод, возможно с помощью задания команды `>> pcode имя_файла`. Во время выполнения этой команды выполняется синтаксический анализ указанного m-файла, и в случае отсутствия синтаксических ошибок в текущей директории создается файл с тем же самым именем, но с расширением r, куда записывается псевдокод. При создании р-файла исходный m-файл сохраняется. Файлы с расширением r имеют более высокий приоритет при вызове, чем файлы с расширением m. Следовательно, если в одной директории размещаются файлы с одинаковыми именами, но расширение одного – m, а другого – r, то при вызове функции, которая содержится в обоих файлах, в память будет загружаться файл с расширением r. Помимо уменьшения времени, затрачиваемого на вызов функции, р-файлы позволяют скрыть описание алгоритма, реализованного в функции.

6.4.3. Классы памяти и области видимости переменных

До сих пор при рассмотрении переменной во внимание принимались атрибуты, которые можно просмотреть в окне *Workspace* (имя, размер, количество занимаемых в памяти байт и класс) и в окне *Array Editor* (значение). Помимо указанных выше атрибутов переменная имеет и другие атрибуты: *класс памяти* и *область видимости*.

Класс памяти переменной определяет период или время жизни, в течение которого эта переменная существует в памяти. Одни переменные существуют на протяжении всего времени выполнения команды или сценария,

другие существуют небольшой промежуток времени, третья постоянно создаются и уничтожаются.

Областью видимости (областью действия) переменной называется область, в которой на данную переменную можно сослаться. На некоторые переменные можно сослаться в любом месте функции или сценария, а на другие – только из определенного места.

Принято различать два класса памяти переменных: *автоматический класс памяти с локальным временем жизни* и *статический класс памяти с глобальным временем жизни*.

К классу с локальным временем жизни относятся переменные, создаваемые в теле функции. Например, переменные `dec`, `r`, `c`, `romans`, `arabics`, `row` и `column`, которые создаются во время выполнения функции `dec2rom()`, являются переменными с локальным временем жизни. Такие переменные для краткости называются *локальными переменными*. Локальные переменные после их создания размещаются в *рабочей области функции* и существуют до тех пор, пока не закончится выполнение функции или пока они явно не будут удалены из ее рабочей области. Рабочая область функции создается во время вызова функции и существует до тех пор, пока функция не закончит свое выполнение. Рабочие области функций и рабочая область системы MATLAB, которая называется *основной рабочей областью*, не пересекаются. Таким образом, в рабочих областях функций и в основной рабочей области могут храниться переменные с одинаковыми именами, но с разными значениями. Во время завершения выполнения функции происходит автоматическое удаление ее рабочей области, а вместе с ней и всех ее локальных переменных. Рабочую область функции во время ее выполнения можно просмотреть с помощью инструментального средства *Workspace Browser*. Пример просмотра рабочей области функции `dec2rom()` во время ее выполнения представлен на рис. 6.8.

Переменные статического класса памяти с глобальным временем жизни бывают двух видов: *глобальные переменные* и *локальные сохраняемые переменные*.

Глобальные переменные создаются с помощью указания *ключевого слова* `global` перед именем переменной или списком переменных, перечисляемых через пробел. Например, для создания трех глобальных переменных: `ONE`, `TWO` и `THREE` необходимо задать следующую инструкцию:

```
global ONE TWO THREE
```

Как правило, глобальные переменные задаются с помощью букв в верхнем регистре. Задание имен глобальных переменных в верхнем регистре позволяет отличать их от одноименных локальных переменных. Можно не следовать этому соглашению, а выработать свой стиль обозначения глобаль-

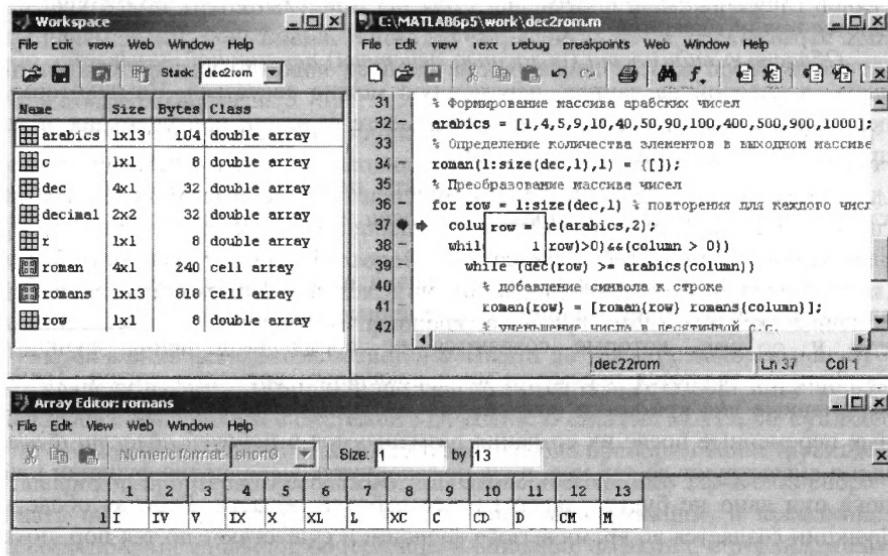


Рис. 6.8. Просмотр рабочей области функции dec2rom()

тных переменных, которого следует придерживаться при разработке большого проекта, состоящего из нескольких функций. При создании глобальных переменных им присваиваются пустые массивы. Глобальные переменные размещаются в *глобальной рабочей области*, которая отлична от основной рабочей области и от рабочих областей функций. Глобальные переменные можно создавать как в командном окне, так и во время вызова функций. Инструкция, которая создает глобальные переменные в функции, пропускается при повторных ее вызовах.

Удалить глобальные переменные из памяти можно с помощью вызова функции `clear()`: `clear global` – приведет к удалению всех переменных из глобальной рабочей области; `clear('global','two')` – приведет к удалению из глобальной рабочей области только одной переменной `two`.

Локальные сохраняемые переменные создаются внутри функции с помощью указания ключевого слова `persistent` перед именем переменной или списком переменных, перечисляемых через пробел. Например, для создания трех локальных сохраняемых переменных: `ONE`, `TWO` и `THREE` необходимо задать следующую инструкцию:

```
persistent ONE TWO THREE
```

При задании локальных сохраняемых переменных следует использовать буквы в верхнем регистре. Сказанное выше относительно задания имен гло-

бальных переменных справедливо и для задания имен локальных сохраняемых переменных. Во время создания локальных сохраняемых переменных им присваивается пустой массив. Локальные сохраняемые переменные сохраняют свои значения в течение всего времени существования функции. При каждом следующем вызове этой функции локальные сохраняемые переменные содержат те значения, которые они имели при завершении предыдущего ее вызова. Локальные сохраняемые переменные можно удалить только вместе с функцией. Например, при вызове функции `clear()`, входным параметром которой является имя удаляемой функции, из памяти будет удалена не только указанная функция, но и все локальные сохраняемые переменные, которые были созданы во время ее выполнения.

Областью действия, или видимости, переменной называется та часть функции или сценария, в которой на переменную можно ссылаться. Существуют три области действия: *область действия сценарий*, *область действия функция* и *область действия системы MATLAB*.

Переменные, которые создаются во время выполнения сценария или с помощью задания команды в командном окне, имеют область действия сценарий. Эти переменные размещаются в основной рабочей области системы MATLAB и доступны всем сценариям, а также командам, вводимым в командном окне. На переменные с областью действия сценарий нельзя ссылаться внутри функции. Переменные с областью действия сценарий являются самыми распространенными переменными при работе с командным окном системы MATLAB. При решении задач векторной алгебры, линейной алгебры и аналитической геометрии использовались переменные только с областью действия сценарий.

Переменные, создаваемые во время выполнения функции, имеют область действия функция. Эти переменные размещаются в рабочей области соответствующих функций и видимы только в той функции, где они были созданы. Функции позволяют использовать один из наиболее фундаментальных принципов разработки хорошего программного обеспечения – скрытие информации. Таким образом, на переменные, которые создаются внутри функции, нельзя ссылаться в сценарии и при задании команд в командном окне. Автоматические и сохраняемые локальные переменные, созданные с помощью ключевого слова `persistent`, имеют область действия функция.

Переменная, которая создается с помощью ключевого слова `global`, имеет область действия системы MATLAB. Эти переменные доступны как внутри сценариев, так и внутри функций. Использование в функциях или сценариях глобальных переменных может приводить к побочным эффектам. При выполнении инструкций функция может случайно изменить значение глобальной переменной, что, возможно, приведет к неверному результату.

выполнения всего приложения. При разработке собственных функций следует избегать использования глобальных переменных, за исключением тех случаев, когда основным критерием является производительность программы, а не ее расширяемость, возможность повторного использования набора инструкций и т.д.

На листинге 6.17 представлен пример, который демонстрирует использование области видимости глобальных переменных, автоматических локальных переменных и сохраняемых локальных переменных. На листинге 6.17 представлено описание четырех функций, которые размещаются в одноименных файлах и не имеют входных и выходных параметров.

При выполнении первой инструкции функции `main()` создается глобальная переменная `x`, которой присваивается пустой массив. После выполнения второй инструкции глобальная переменная `X` содержит значение, равное единице. Третья инструкция создает локальную автоматическую переменную `x`, которая видна только внутри этой функции, и присваивает ей значение, равное пяти. Шесть следующих инструкций поочередно в указанной последовательности вызывают функции `a()`, `p()` и `g()`. При вызовах функций не указываются значения параметров, так как при их описании входные и выходные параметры отсутствуют. После вызовов функций в командное окно выводится значение локальной переменной `x`, область видимости которой, функция `main()`. Так как локальная переменная `x` с областью действия функция, не видима в функциях, которые вызываются из функции `main()`, то значение этой переменной не изменится и будет равно пяти. После вывода локальной переменной `x` происходит увеличение значения глобальной переменной `X` на единицу и вывод ее нового значения в командное окно. Завершается выполнение функции `main()` удалением глобальной переменной `X` из глобальной рабочей области системы MATLAB. Так как глобальные переменные видны во всех функциях и сценариях, то их значения могут изменяться во время вызова функций `a()`, `p()` и `g()`. В рассматриваемом примере при каждом вызове функции `g()` происходит вывод в командное окно значения глобальной переменной, которое было до и после выполнения этой функции. Во время ее выполнения значение глобальной переменной `X` увеличивается на единицу. Так как при выполнении функции `main()` функция `g()` вызывается дважды, то значение переменной будет увеличено на два. Следовательно, последним значением глобальной переменной `X` перед ее удалением будет 4.

При выполнении функции `a()` создается локальная переменная `x`, которая видима в этой функции и существует только во время ее выполнения. Попытка обратиться к переменной `x` без ее инициализации приведет к ошибке, несмотря на то, что в функции `main()` была создана переменная с таким же именем. Так как при завершении выполнения функции `a()` ее локальные

переменные *x* и *str* уничтожаются, то при повторном вызове этой функции в командное окно будут выведены те же самые значения (1 и 2) локальной переменной *x*, что и при первом ее вызове.

Во время выполнения функции *p()* происходит увеличение значения сохраняемой локальной переменной *x* на единицу. При первом вызове этой функции создается сохраняемая локальная переменная *X*, ей присваивается начальное значение, равное одному, а затем происходит увеличение значения на единицу. После завершения выполнения этой функции из памяти удаляется автоматическая переменная *str*, а сохраняемая переменная *x* остается в памяти. Так как при выполнении функции *main()* функция *p()* вызывается дважды, то значение сохраняемой переменной *x* при завершении работы функции *main()* будет равно трем. Какое значение будет находиться в сохраняемой переменной *X* при повторном вызове функции *main()*?

Листинг 6.17

```
function [] = main()
% пример на классы памяти и области видимости переменных
global X % создание глобальной переменной X
X = 1; % задание начального значения глобальной переменной
x = 5; % создание локальной переменной x
a; p; g; a; p; g; % последовательные вызовы функций
str = ['Локальная переменная x в main = ' num2str(x)]; disp(str)
X = X+1; str = ['Глобальная переменная X = ' num2str(X)]; disp(str)
clear global X

function [] = a()
% Работа с автоматической локальной переменной x
x = 1; str = ['Локальная переменная x в a = ' num2str(x)]; disp(str)
x = x+1; str = ['Локальная переменная x в a = ' num2str(x)]; disp(str)

function [] = p()
% Работа с сохраняемой локальной переменной
persistent X .
if isempty(X), X = 1; end
str = ['Сохраняемая локальная переменная x в p = ' num2str(X)]; disp(str)
X = X+1;
str = ['Сохраняемая локальная переменная x в p = ' num2str(X)]; disp(str)

function [] = g()
% Работа с глобальной переменной
global X
```

```

if (X == []), X = 0, end
str = ['Глобальная переменная X = ' num2str(X)]; disp(str)
X = X+1; str = ['Глобальная переменная X = ' num2str(X)]; disp(str)

```

Вызов функции `main()` из командной строки:

```

>> main
Локальная переменная x в a = 1
Локальная переменная x в a = 2
Сохраняется локальная переменная x в p = 1
Сохраняется локальная переменная x в p = 2
Глобальная переменная X = 1
Глобальная переменная X = 2
Локальная переменная x в a = 1
Локальная переменная x в a = 2
Сохраняется локальная переменная x в p = 2
Сохраняется локальная переменная x в p = 3
Глобальная переменная X = 2
Глобальная переменная X = 3
Локальная переменная x в main = 5
Глобальная переменная X = 4

```

6.4.4. Параметры функции

6.4.4.1. Основные понятия

При работе с функциями различают *фактические* и *формальные* параметры. Фактические параметры указываются во время вызова функции, а формальные параметры – при ее описании. Во время вызова функции происходит замена формальных параметров на фактические, которая называется *передачей параметров*. Например, на листинге 6.15 переменные `decimal` и `roman` являются формальными параметрами, а переменные `d` и `rom` – фактическими параметрами. Во время вызова функции `dec2rom()` формальные параметры `decimal` и `roman` заменяются фактическими параметрами `d` и `rom` соответственно. Имена формальных и фактических параметров могут совпадать (см. пример 6.2). Фактические параметры по отношению к функции являются динамическим контекстом вызова, так как от вызова к вызову они могут изменяться. В языке MATLAB связь между фактическими и формальными параметрами является *позиционной*, т.е. каждому формальному параметру соответствует фактический параметр, который имеет тот же порядковый номер в списке входных или выходных параметров функции соответственно, что и формальный параметр.

В заголовке функции при ее описании можно определить *входные* (in-параметры) и *выходные* (out-параметры) *формальные параметры*. Фор-

мальные параметры используются в теле функции точно так же, как любые переменные. Следовательно, при работе с функцией можно присваивать значения как выходным, так и входным формальным параметрам.

По своему назначению фактические параметры подразделяются на три вида: *входные* (in-параметры), *выходные* (out-параметры) и *совмещенные* (inout-параметры). Входные параметры представляют собой значения, которые передаются в функцию для ее выполнения. Следовательно, входными параметрами могут быть числа (`sin(10)`), переменные (`sin(x)`), выражения (`sin(a+b)`), которые также могут включать вызов функции (`sin(sqrt(x))`). Входным параметром может быть также дескриптор функции. Во время вызова функции система MATLAB позволяет задавать произвольное количество входных фактических параметров, а также присваивать входным формальным параметрам начальные значения, если соответствующие фактические параметры не указаны. Присваивание начальных значений входным формальным параметрам называется *инициализацией параметров*. Выходными параметрами являются переменные, так как только в них можно записать результаты, возвращаемые функцией. Например, при задании команды `>> a = sin(10)` будет вызвана функция `sin()` с входным параметром в виде константы 10 и выходным параметром в виде переменной `a`. Язык MATLAB позволяет задавать вызов функции без указания выходных фактических параметров. В этом случае значение первого формального параметра присваивается переменной `ans`, которая используется по умолчанию в качестве единственного фактического параметра (см. пример 6.15). Класс выходного формального параметра определяет класс соответствующего выходного фактического параметра. В том случае, если при вызове функции имя входного параметра совпадает с именем выходного параметра, то этот параметр является совмещенным (`>> a = sin(a)`). Совмещенные параметры могут быть только переменными. Совмещенные параметры отличаются от выходных параметров наличием определенных значений перед вызовом функции.

Передача входных параметров в языке MATLAB осуществляется двумя способами: *по значению* и *по ссылке*. При передаче параметра по значению во время вызова функции создается новая переменная с именем формального параметра, в которую копируется значение соответствующего фактического параметра. Следовательно, при выполнении функции все операции, в которых присутствует имя формального параметра, осуществляются с переменной, которая является копией фактического параметра. Так как при завершении работы функции эта переменная уничтожается, то новые значения, которые могли быть присвоены этой переменной, не сохраняются. Передача параметра по значению позволяет повысить надежность программы, так как исключается возможность случайного изменения входного фактиче-

ского параметра. Однако при передаче параметра по значению на создание копии необходимо затратить время. При работе с матрицами больших размеров, на что ориентирована система MATLAB, копирование значений фактических параметров может занять значительное время, что приведет к резкому снижению производительности. При передаче параметра по ссылке формальному параметру передается не значение фактического параметра, а его адрес. Таким образом, функция работает не с копией, а с оригиналом. В данном случае исключаются потери времени на копирование, но увеличивается вероятность случайного изменения значений фактических параметров. В языке MATLAB при передаче параметров используется комбинированный подход. По умолчанию передача параметров осуществляется по ссылке, что обеспечивает максимально возможную производительность программы, в которой используются вызовы функций. В том случае, если в описании функции входной формальный параметр находится слева от знака присваивания, то передача соответствующего фактического параметра во время вызова функции осуществляется по значению. Таким образом, исключается случайное изменение фактического параметра. Следовательно, для сохранения высокой производительности не следует в теле функции присваивать входному формальному параметру новое значение. При передаче параметра по ссылке в формальный параметр копируется значение адреса памяти, где располагается значение фактического параметра. Таким образом, передача параметра по ссылке моделируется с помощью передачи параметра по значению.

6.4.4.2. Инициализация входных формальных параметров

Во время вызова функции можно не указывать ряд входных фактических параметров. В этом случае необходимо предусмотреть *инициализацию* соответствующих входных формальных параметров. Под инициализацией входных формальных параметров понимается задание им начальных значений. Формальные параметры, которые можно инициализировать в теле функции, называются *дополнительными входными параметрами*. Так как в языке MATLAB используется позиционная форма передачи параметров, то опускаться во время вызова функции могут лишь последние параметры, указанные в заголовке функции.

Пример 6.17. Создать функцию `circle()`, осуществляющую построение окружности с центром в точке $A(x_0; y_0)$ и радиусом r . Функция имеет три соответствующих формальных параметра: r , x_0 и y_0 . Формальные параметры x_0 и y_0 являются дополнительными. Выполнить вызов без указания фактических параметров, с указанием одного, двух и трех параметров.

Решение:**Листинг 6.18**

```
function [] = circle(r,x0,y0)
%CIRCLE построение окружности радиусом r с центром в точке (x0,y0)
% r - радиус окружности, x0,y0 - координаты центра окружности

% Определение числа заданных входных фактических параметров
if (nargin < 1)
    error('Слишком мало входных фактических параметров')
end
% Задание начальных значений дополнительным параметрам
if (nargin < 3), y0 = 0; end % инициализация параметра y0
if (nargin < 2), x0 = 0; end % инициализация параметра x0

% локальные переменные
t = 0:pi/64:2*pi; % параметр
x = x0 + r*cos(t); % координаты точек окружности по оси x
y = y0 + r*sin(t); % координаты точек окружности по оси y
% построение окружности в графическом окне
plot(x,y), axis equal, grid on, hold on
xlabel('x'), ylabel('y'), title('circle')
```

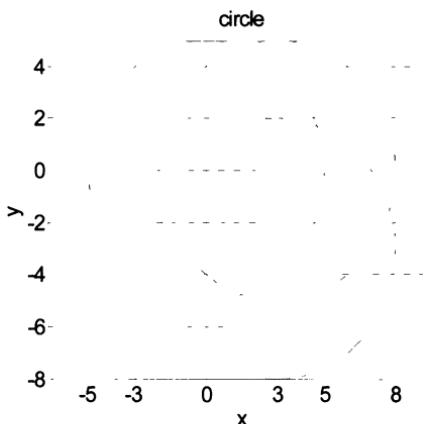
Вызовы функции `circle()` с различным числом входных параметров:

```
>> circle(5,3,-3)
>> circle(5,3)
>> circle(5)
>> circle
??? Error using ==> circle
Слишком мало входных фактических параметров
```

Результаты вышеприведенных вызовов функции `circle()` представлены на рис.6.9.

Тело функции `circle()` состоит из двух частей. В первой части осуществляется определение количества входных фактических параметров и, если необходимо, инициализация входных формальных параметров. Во второй части выполняется вычисление значений координат точек окружности и ее построение в графическом окне.

Для определения количества входных фактических параметров в теле функции `circle()` использовался вызов встроенной в ядро MATLAB функции `nargin()` без указания входных параметров. Функция `nargin()` возвращает число фактических параметров, с которыми осуществлялся вызов функции, в теле которой имеется обращение к функции `nargin()`. Следова-



**Рис. 6.9. Построение окружностей
с помощью параметров по умолчанию**

тельно, функцию `nargin()` можно вызывать только из т-файла, в котором имеется описание функции. Вызов функции `nargin()` из командного окна приведет к выводу в командное окно сообщения об ошибке использования функции `nargin()`.

Первый вызов функции `circle()` был выполнен в обычном виде с указанием полного количества фактических параметров. После выполнения функции в графическом окне была построена окружность с центром в точке с координатами $(x_0; y_0)$ и радиусом r .

Второй вызов функции `circle()` выполнялся с помощью указания двух фактических параметров. В этом случае после вызова функции последний формальный параметр (y_0) является неопределенным. Для нормальной работы ему необходимо присвоить значение по умолчанию. Значение по умолчанию неопределенным формальным параметрам присваивается с помощью структуры выбора `if end`. При задании двух фактических параметров будет выполнена инструкция, содержащаяся в теле структуры выбора `if end` с логическим выражением `nargin < 3`. В результате выполнения этой инструкции формальному параметру y_0 будет присвоено нулевое значение. Инструкции, расположенные в остальных структурах выбора `if end`, выполнены не будут, так как результатом их логических выражений является значение `false`. Следовательно, при вызове функции с двумя фактическими параметрами будет построена окружность с центром в точке $(x_0; 0)$ и радиусом r .

Третий вызов функции `circle()` сопровождается заданием одного фактического параметра. В этом случае после вызова будут неопределенными два формальных параметра: y_0 и x_0 . Задание начальных значений этим параметрам выполняется с помощью инструкций, расположенных во второй

и третьей структурах выбора соответственно. После инициализации формальных параметров `x0` и `y0` они будут содержать нулевые значения. Таким образом, после выполнения функции будет построена окружность с центром в точке $(0;0)$ и радиусом 5.

Четвертый вызов функции `circle()` выполнялся без указания фактических параметров. После вызова функции выполнение передается на первую инструкцию в теле функции, которой является структура выбора `if end`. Так как в результате вычисления логического выражения `nargin < 1` получается истина, то выполняется инструкция, расположенная в теле структуры выбора. Выполнение этой инструкции приведет к досрочному завершению работы функции и выводу в командное окно сообщения о недостаточном числе фактических параметров. Таким образом, первая структура выбора `if end` используется для задания минимального числа фактических параметров или числа обязательных формальных параметров. В рассматриваемом примере количество обязательных формальных параметров равно одному, что соответствует постоянному заданию значения радиуса отображаемой в графическом окне окружности при вызове функции.

Осуществлять проверку числа входных фактических параметров можно с помощью вызова функции `nargchk(low,high,nargin)`. Первые два параметра задают соответственно минимальное и максимальное число параметров, которые можно передавать в функцию. Третий параметр соответствует действительному числу входных параметров, передаваемых в функцию. Если число действительных фактических параметров выходит за установленные границы, то функция `nargchk()` возвращает сообщение об ошибке. В противном случае функция `nargchk()` вернет пустой массив. Следовательно, первую структуру выбора `if end`, представленную на листинге 6.18, можно заменить на следующую инструкцию: `error(nargchk(1,3,nargin))`. Если после внесения изменений вызвать функцию `circle()` без задания входных фактических параметров, то выполнение функции будет прервано при выполнении новой инструкции и в командное окно будет выведено сообщение о недостаточном количестве входных параметров (*Not enough input arguments*). Если при вызове функции `circle()` задать четыре и более входных фактических параметра, то выполнение функции также будет досрочно прекращено, но в командное окно будет выведено сообщение о слишком большом количестве входных фактических параметров (*Too many input arguments*).

Таким образом, наличие формальных параметров, задаваемых по умолчанию, позволяет вызывать одну и ту же функцию с разным числом фактических параметров. При этом диапазон изменения числа фактических параметров является фиксированным и попытка выйти за его пределы приведет к ошибке.

6.4.4.3. Обязательные и дополнительные выходные формальные параметры

Язык MATLAB позволяет во время вызова функции не только задавать не полный список входных фактических параметров, но и опускать ряд выходных фактических параметров. После выполнения инструкций функция возвращает результат в заданные фактические параметры либо в переменную *ans*, если отсутствуют выходные фактические параметры, а в заголовке функции имеются выходные формальные параметры. По умолчанию во время выполнения функции вычисляются значения всех выходных формальных параметров, даже несмотря на задание неполного количества выходных фактических параметров (см. пример 6.15). Вычисление значений выходных формальных параметров, которые не возвращаются из функции, приводит к неэффективному ее использованию. Добиться более эффективного использования функции можно путем включения в ее тело блока, в котором определяется число выходных фактических параметров и выполняются соответствующие этому числу действия. Для определения числа выходных фактических параметров необходимо обращаться к встроенной в ядро MATLAB функции *nargout()* без задания входных параметров.

Пример 6.18. На базе функции *circle()* из предыдущего примера создать функцию *circle1()*, которая имеет два выходных формальных параметра – *x* и *y*. При вызове функции без выходных фактических параметров выполняется построение окружности без вычисления выходных формальных параметров. В случае вызова функции с одним выходным фактическим параметром выполняется построение окружности, как в предыдущем случае, а также первому выходному формальному параметру присваиваются значения координат точек окружности по оси абсцисс. При вызове функции с двумя фактическими параметрами выходным формальным параметрам присваиваются значения координат точек по соответствующим осям. Построение окружности в последнем случае не выполняется.

Решение:

Листинг 6.19

```
function [x,y] = circle1(r,x0,y0)
%CIRCLE1 построение окружности радиусом r с центром в точке (x0,y0)
% r - радиус окружности
% x0,y0 - координаты центра окружности

% Анализ числа заданных входных фактических параметров
error(nargchk(1,3,nargin))
```

Окончание листинга 6.19

```
% Задание начальных значений дополнительным параметрам
if (nargin < 3), y0 = 0; end % инициализация параметра y0
if (nargin < 2), x0 = 0; end % инициализация параметра x0
% локальные переменные
t = 0:pi/64:2*pi;           % параметр
x1=x0 + r*cos(t); y1=y0 + r*sin(t); % координаты точек окружности
% построение окружности в графическом окне
if (nargout < 1)
    plot(x1,y1), axis equal, grid on, hold on
    xlabel('x'), ylabel('y'), title('circle')
elseif (nargout < 2)
    x = x1;
    plot(x1,y1), axis equal, grid on, hold on
    xlabel('x'), ylabel('y'), title('circle')
else
    x = x1;
    y = y1;
end
```

Тело функции, представленной на листинге 6.19, состоит из трех частей.

В первой части производится определение числа входных фактических параметров и при необходимости – инициализация входных формальных параметров. Во второй части вычисляются координаты точек, на основе которых строится окружность. Третья часть служит для повышения эффективности выполнения функции и состоит из структуры выбора `if elseif else end`. Первая ветвь этой структуры содержит инструкции, которые будут выполняться при вызове функции без указания выходных фактических параметров. Вторая ветвь включает инструкции, выполняющиеся в том случае, если указан один выходной фактический параметр. Инструкции, расположенные в третьей ветви, будут выполнены при задании двух выходных фактических параметров.

Проверка числа выходных фактических параметров выполняется с помощью функции `nargoutchk(low,high,nargout)`. Первые два параметра задают соответственно минимальное (обязательное) и максимальное число выходных фактических параметров. Выходные формальные параметры, соответствующие выходным фактическим параметрам, которые необходимо указывать всегда, называются *обязательными*. Выходные формальные параметры, соответствующие выходным фактическим параметрам, которые можно опускать, называются *дополнительными*. Третий параметр соответствует действительному числу выходных фактических параметров. Если число действительных выходных фактических параметров выходит за установлен-

ные границы, то функция `nargoutchk()` возвращает сообщение об ошибке (Not enough output arguments или Too many output arguments). В противном случае результатом будет пустой массив.

С целью просмотра функции `nargoutchk()` в действии следует изменить тело функции `circle1()`. После проверки числа входных фактических параметров включить проверку числа выходных фактических параметров. Задать минимальное число выходных фактических параметров, равное единице, и заменить структуру выбора `if elseif else end` на `if else end`, исключив первую ветвь.

6.4.4.4. Задание произвольного числа входных формальных параметров

Язык MATLAB позволяет передавать произвольное число входных параметров. Передача произвольного числа параметров реализуется с помощью задания входного формального параметра `varargin`, который имеет класс массив ячеек. Если в заголовке функции кроме формального параметра `varargin` присутствуют и другие входные формальные параметры, то они записываются первыми, так как параметр `varargin` является дополнительным. Во время вызова функции все заданные фактические параметры, которые позиционно соответствуют формальному параметру `varargin`, преобразуются в одномерный массив ячеек и помещаются в `varargin`. Так как значения фактических параметров содержатся в массиве ячеек с именем `varargin`, то для их получения необходимо обратиться к соответствующей ячейке по номеру (например, `varargin{1}` – обращение к первой ячейке массива ячеек). Так как во время вызова функции, содержащей формальный параметр `varargin`, происходит копирование соответствующих фактических параметров в этот формальный параметр, то в этом случае осуществляется передача параметров по значению.

Пример 6.19. Написать функцию `circles()`, которая принимает произвольное число входных параметров, кратное трем, и строит соответствующие окружности в графическом окне. Первый параметр в тройке обозначает радиус окружности, второй и третий – координаты центра окружности по осям абсцисс и ординат соответственно.

Решение:

Листинг 6.20

```
function [] = circles(varargin)
%CIRCLE построение окружностей
% varargin - массив ячеек, которые содержат по три элемента.
% r - радиус соответствующей окружности (первый элемент)
% x0,y0 - координаты центров окружностей по осям x и y
```

Окончание листинга 6.20

```
% Проверка числа входных параметров
if nargin < 1
    r = 1; x0 = 0; y0 = 0; % параметры по умолчанию
else
    % Преобразование входного массива ячеек в массив чисел
    len = length(varargin);
    for k = 1:len
        r(k) = varargin{k}(1); % радиус k-й окружности
        x0(k) = varargin{k}(2); % центр окружности по оси x
        y0(k) = varargin{k}(3); % центр окружности по оси y
    end
end
% Вычисление координат точек окружностей
x0 = x0(:); y0 = y0(:); r = r(:); % векторы-столбцы
t = 0:pi/64:2*pi; % параметр
x = x0*ones(1,size(t,2))+r*cos(t); % коорд. точек по x
y = y0*ones(1,size(t,2)) + r*sin(t); % коорд. точек по y
% построение окружностей в графическом окне
x = x.'; y = y.'; % транспонирование массивов
line(x,y,'LineWidth',2)
axis equal, grid on, hold on, box on
xlabel('x'), ylabel('y'), title('circles')
```

Вызовы функции `circle()` с числом входных фактических параметров, равным четырем, и с числом параметров, равным шести:

```
>> a = [1 0 0]; circles(a,[2 0 0],[0.5 0 1.5],[0.5 0 -1.5])
>> circles(a,[2,0,0],[0.5,0,1.5],[0.5,0,-1.5],[0.5,1.5,0],[0.5,-1.5,0])
```

Результаты вызовов функции `circles()` приведены на ил. 18. В первом вызове количество фактических параметров равно четырем, а во втором – шести. Каждый фактический параметр определяет параметры окружности и представляет собой вектор, состоящий из трех элементов. В том случае, если при вызове функции `circles()` входные фактические параметры не указываются, то локальным переменным `r`, `x0` и `y0` присваиваются соответствующие значения по умолчанию.

В рассматриваемом примере каждая ячейка представляет собой вектор, состоящий из трех элементов. Обращение к элементу вектора задается после обращения к ячейке (например, `varargin{1}(2)` – обращение ко второму элементу вектора, расположенному в первой ячейке). Для определения числа ячеек в одномерном массиве (числа фактических параметров) используется функция `length()`, которая возвращает количество элементов в векторе.

Структура повторения `for end` применяется для копирования значений элементов векторов, расположенных в ячейках, в соответствующие локальные переменные (создаются вторые копии значений фактических параметров). После копирования входных данных выполняется вычисление координат точек окружностей и их построение в графическом окне. Таким образом, использование в функциях возможности произвольного задания числа фактических параметров не только увеличивает гибкость при работе с функцией, но и снижает эффективность ее работы, так как необходимо затратить время и память на создание и размещение копий входных фактических параметров.

Функции с произвольным числом формальных параметров широко применяются при работе с дескрипторной графикой. Ряд стандартных функций (`axis()`, `title()`, `xlabel()`, `ylabel()` и т.д.) при вызове принимают произвольное число входных фактических параметров. Входной формальный параметр `varargin` после вызова этих функций в большинстве случаев содержит в двух соседних ячейках строковые значения: имя свойства объекта дескрипторной графики и его значение.

Пример 6.20. Написать функцию `circles2()`, которая принимает в качестве входных фактических параметров радиус окружности `r`, координаты центра окружности `x0` и `y0` по осям `x` и `y` соответственно, а также произвольное число пар значений, задающих свойство объекта дескрипторной графики `Line` и значение этого свойства. Формальный параметр `r` является обязательным, а все остальные параметры – дополнительными.

Решение:

Листинг 6.21

```
function [] = circle2(r,x0,y0,varargin)
%CIRCLE2 построение окружности с заданными свойствами
% r - радиус окружности (обязательный параметр)
% x0, y0 - координаты центра окружности (дополнительные параметры)
% varargin - свойства объекта дескрипторной графики line

% Проверка числа входных фактических параметров
if nargin < 1, error('Отсутствуют входные фактические параметры'), end
if nargin < 2, x0 = 0; y0 = 0; end
if nargin < 3, y0 = 0; end
% Формирование координат точек окружности
t = 0:pi/64:2*pi; % параметр
x = x0 + r*cos(t); % координаты точек окружности по оси x
y = y0 + r*sin(t); % координаты точек окружности по оси y
```

Окончание листинга 6.21

```
% Построение окружности
line(x,y,varargin(:))
% Задание свойств координатной плоскости
set(gca,'XGrid','on','YGrid','on','DataAspectRatio',[1 1 1])
xlabel('x'), ylabel('y')
```

Вызовы функции `circle2()` с произвольным числом входных фактических параметров:

```
>> r = 3; x0 = 2; y0 = 1;
>> circle2(r,x0,y0,'LineWidth',9,'LineStyle',':')
>> circle2(2,0,0,'LineWidth',4)
>> circle2(1)
>> circle2
??? Error using ==> circle2
Отсутствуют входные фактические параметры
```

Результаты задания вышеприведенных команд изображены на рис. 6.10.

В первом вызове функции `circle2()` присутствуют семь входных фактических параметров. Первый фактический параметр соответствует радиусу окружности, второй и третий – координатам центра окружности по осям x и y соответственно. Остальные две пары входных параметров изменяют значения свойств `LineWidth` и `LineStyle` объекта дескрипторной графики `Line`, которые приняты по умолчанию. Ширина линии задается равной 9 пунктам, а стиль линии – пунктирным. Второй вызов функции `circle2()` сопровождается заданием пяти входных фактических параметров. В этом случае задаются радиус окружности, координаты центра окружности по соответствующим осям и толщина линии. Во время третьего вызова присутствует

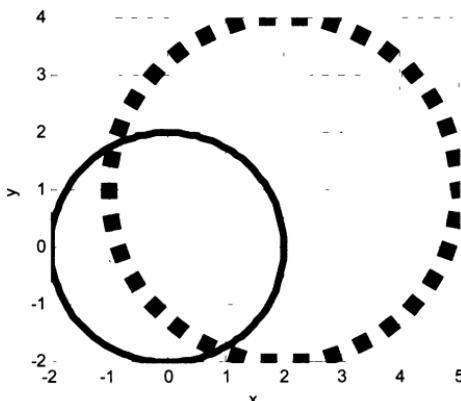


Рис. 6.10. Построение окружностей

только один входной фактический параметр, который определяет радиус окружности, отображаемой в графическом окне. Все остальные параметры окружности и стиль линии задаются по умолчанию. При четвертом вызове функции `circle2()` отсутствуют входные фактические параметры, что приводит к выводу в командное окно сообщения об ошибке и к досрочному прекращению выполнения функции.

6.4.4.5. Задание произвольного числа выходных формальных параметров

Язык MATLAB позволяет передавать наряду с произвольным числом входных параметров и произвольное число выходных параметров. Передача произвольного числа выходных параметров из функции реализуется с помощью задания выходного формального параметра `varargout`, который имеет класс массив ячеек. Если в заголовке функции кроме формального параметра `varargout` присутствуют и другие выходные формальные параметры, то они записываются первыми, так как параметр `varargout` является дополнительным параметром. При завершении работы функции значения, содержащиеся в ячейках выходного формального параметра `varargout`, копируются в соответствующие выходные фактические параметры.

Пример 6.21. Написать функцию `circles1()`, которая принимает одно значение, равное числу отображаемых в графическом окне окружностей. В теле функции сформировать соответствующее число радиусов и координат центров окружностей по каждой оси. Значения радиусов и координат центров задаются случайным образом с помощью функции `rand()`. В графическом окне необходимо отобразить все окружности. После построения окружностей следует передать указанным выходным фактическим параметрам значения радиусов и координат центра соответствующей окружности. Каждый выходной фактический параметр является массивом, содержащим три элемента, которые определяют окружность: радиус и координаты центра по осям абсцисс и ординат.

Решение:

Листинг 6.22

```
function [varargout] = circles1(number)
%CIRCLE1 построение окружностей с возвращением их параметров
% number - число окружностей

% Проверка числа входных фактических параметров
if (nargin < 1) || (nargin > 1)
    error('Неправильно задано число входных фактических параметров')
end
```

Окончание листинга 6.22

```

if (number < 1)
    error('Входное значение должно быть больше нуля')
end
if (nargout > number)
    error('Слишком много выходных фактических параметров')
end
% Определение радиусов и координат центров окружностей
r = fix(5*rand(1,number))'; % случайные значения радиусов (<=5)
x0 = (5 - 10*rand(1,number))'; % коорд. центров по оси x [-5;5]
y0 = (5 - 10*rand(1,number))'; % коорд. центров по оси y [-5;5]
% Вычисление координат точек окружностей
t = [0:pi/64:2*pi]; % параметр
x = x0*ones(1,size(t,2)) + r*cos(t); % по оси x
y = y0*ones(1,size(t,2)) + r*sin(t); % по оси y
% Построение окружностей и оформление координатной плоскости
plot(x.',y.'), axis('equal'), xlabel('x'), ylabel('y'), grid on
% Формирование выходных формальных параметров
i = 0; % локальная переменная
while ((i < nargout) && (i < number))
    i = i + 1;
    varargout{i}(1) = r(i);
    varargout{i}(2) = x0(i);
    varargout{i}(3) = y0(i);
end

```

Вызовы функции `circles1()` с произвольным числом выходных фактических параметров:

```

>> [c1,c2,c3] = circles1(0)
??? Error using ==> circles1
Входное значение должно быть больше нуля

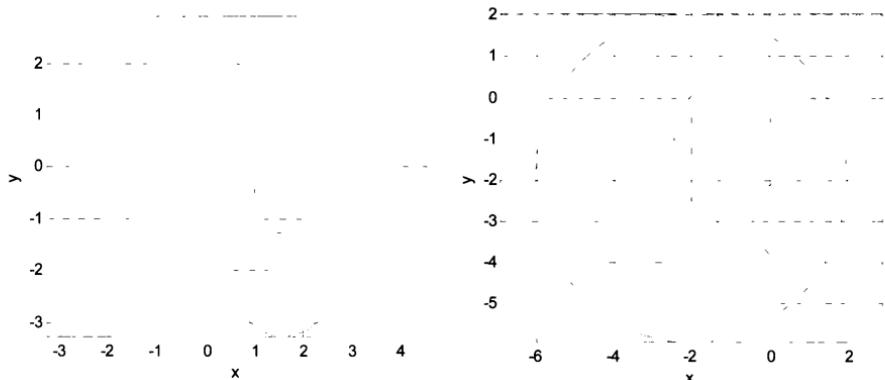
>> [c1,c2] = circles1(2)
c1 =
    1.0000    1.5881   -2.2711
c2 =
    1.0000   -0.3408    1.9071
>> [c1,c2,c3] = circles1(3)
c1 =
    4.0000   -2.0274   -1.9457
c2 =
    2.0000   -0.4657   -1.2131
c3 =
    1.0000    0.5512   -2.9482

```

```
>> [c1,c2,c3] = circles1(2)
??? Error using ==> circles1
Слишком много выходных фактических параметров

>> [c1,c2,c3] = circles1
??? Error using ==> circles1
Неправильно задано число входных фактических параметров
```

Графические результаты вызовов функции `circles1()` представлены на рис. 6.11.



a) $\gg [c1, c2] = \text{circles1}(2)$ b) $\gg [c1, c2] = \text{circles1}(3)$

Р и с . 6.11. Графические результаты вызовов функции `circles1()`

При первом вызове функции `circles1()` значение входного фактического параметра равно нулю, что приводит к досрочному завершению выполнения функции и выводу в командное окно сообщения об ошибочно заданном значении входного параметра.

Во время второго вызова значение входного фактического параметра равно двум, а выходными фактическими параметрами являются $c1$ и $c2$. Во время выполнения функции в графическом окне строятся две окружности (рис. 6.11,а) и формируется выходной формальный параметр `varargout`, состоящий из двух ячеек. В каждой ячейке находится массив 1×3 , первый элемент которого содержит радиус соответствующей окружности, а второй и третий элементы – координаты центра по осям x и y соответственно. При завершении работы функции содержимое первой ячейки копируется в выходной фактический параметр $c1$, а содержимое второй ячейки – в параметр $c2$. После окончания работы функции значения переменных $c1$ и $c2$ выводятся в командное окно.

Третий вызов функции соответствует предыдущему вызову, за исключением того, что число выходных фактических параметров и значение входного фактического параметра равны трем. В результате в графическом окне будут построены три окружности со случайными значениями радиусов и их центров, которые при завершении работы функции передаются параметрам `c1`, `c2` и `c3` соответственно. После завершения выполнения функции значения этих переменных отображаются в командном окне. Следовательно, в этом случае выходной формальный параметр `varargout` содержит три ячейки.

При четвертом вызове функции `circles1()` были заданы три выходных фактических параметра и один входной фактический параметр, значение которого равно двум. Так как количество выходных параметров превышает количество окружностей, которые необходимо создать, то выполнение функции завершится досрочно с выводом сообщения об ошибке в командное окно.

Во время пятого вызова функции `circles()` отсутствуют входные фактические параметры. Так как обязательно должен присутствовать один входной фактический параметр, то выполнение функции будет прервано и в командном окне появится соответствующее сообщение об ошибке.

Пример 6.22. Написать функцию `circles2()`, которая принимает произвольное число входных параметров и возвращает произвольное число выходных параметров. В качестве первого обязательного входного параметра функция принимает количество окружностей, которые необходимо построить. Остальные входные параметры являются необязательными и используются для переопределения свойств объекта дескрипторной графики `axes`, в котором изображаются окружности. Первые два выходных формальных параметра являются обязательными. Они содержат координаты окружностей по осям `x` и `y` соответственно. Остальные выходные параметры являются необязательными и содержат основные параметры соответствующей окружности – ее радиус и координаты центра по осям `x` и `y`.

Решение:

Листинг 6.23

```
function [x,y, varargout] = circles2(number,varargin)
%CIRCLE2 построение окружностей и возвращение их параметров
% number - число окружностей

% Проверка числа входных и выходных фактических параметров
if (nargin < 1) % отсутствуют входные фактические параметры
    error('Неправильно задано число входных фактических параметров')
end
```

Окончание листинга 6.23

```

if (nargout < 2) % выходных фактических параметров меньше двух
    error('Слишком мало выходных фактических параметров')
end
if (number < 1)
    error('Входное значение должно быть больше нуля')
end
% Определение радиусов и координат центров окружностей
r = fix(5*rand(1,number))'; % случайные значения радиусов (<=5)
x0 = (5 - 10*rand(1,number))'; % координаты центров по x [-5;5]
y0 = (5 - 10*rand(1,number))'; % координаты центров по y [-5;5]
% Вычисление координат точек окружностей
t = [0:pi/64:2*pi]; % параметр
x = (x0*ones(1,size(t,2)) + r*cos(t))'; % координаты точек по x
y = (y0*ones(1,size(t,2)) + r*sin(t))'; % координаты точек по y
% Построение окружностей
plot(x,y)
% Оформление координатной плоскости
xlabel('x'), ylabel('y') % обозначение осей координат
if nargin < 2
    axis('equal'), grid on % равный масштаб по осям, коорд. сетка
else
    set(gca,varargin{:}); % свойства, определяемые пользователем
end
% Формирование выходных формальных параметров
i = 0; % локальная переменная
while ((i < nargout - 2) && (i < number))
    i = i + 1;
    varargout{i}(1) = r(i);
    varargout{i}(2) = x0(i);
    varargout{i}(3) = y0(i);
end

```

Пример вызова функции `circles2()` с произвольным числом входных и выходных фактических параметров:

```

>> [x,y,c1,c2] = circles2(4,'XGrid','on','YGrid','on',...
    'DataAspectRatio',[1 1 1]);

```

Графические результаты вызова функции `circles2()` представлены на рис. 6.12. Во время вызова функции указываются четыре выходных и семь входных фактических параметра. Первый входной параметр задает число создаваемых окружностей. Остальные входные параметры используются для

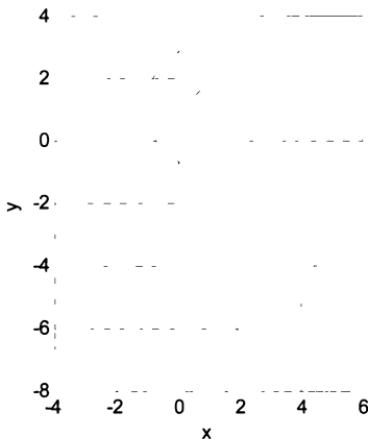


Рис . 6.12. Результаты вызова функции `circles2()`

переопределения значения свойств объекта дескрипторной графики Axes, заданных по умолчанию. Первые две пары этих параметров задают отображение координатной сетки по осям x и y соответственно. Третья пара параметров устанавливает одинаковый масштаб по всем осям. После окончания выполнения работы функции в основной рабочей области MATLAB будут располагаться переменные x , y , $c1$ и $c2$. Переменные $x(129 \times 4)$ и $y(129 \times 4)$ в каждом столбце содержат координаты точек соответствующей окружности. Так как значение первого входного фактического параметра равно четырем, то

каждая переменная содержит по четыре столбца. Количество строк в этих переменных соответствует количеству элементов в переменной t , которая является параметром при описании окружности в параметрическом виде. Переменная $c1(1 \times 3)$ содержит основные параметры первой окружности: радиус ($c1(1) = 2$), координаты центра по оси x ($c1(2) = 0.98196$) и по оси y ($c1(3) = 1.0587$). В переменной находятся основные параметры второй окружности: радиус ($c2(1) = 2$), координаты центра по оси x ($c2(2) = 1.9231$) и по оси y ($c2(3) = -0.030145$). Результаты представлены в формате `shortg`. Значения основных параметров остальных окружностей удаляются из памяти во время окончания выполнения функции.

В некоторых пакетах расширения системы MATLAB встречаются функции, в которых наличие выходных фактических параметров влияет на построение графиков в графическом окне при их выполнении. В том случае, если задаются выходные фактические параметры, которые после выполнения функции будут содержать координаты точек линий, то линии в графическом окне не отображаются. В противном случае происходит построение линий. Измените функцию `circles2()` таким образом, чтобы происходило построение графиков только в том случае, если отсутствуют выходные фактические параметры. Если присутствуют два и более выходных фактических параметра, то вычисляются только значения окружностей. В случае задания одного выходного фактического параметра выводится сообщение об ошибке.

6.4.5. Подфункции

При решении сложной задачи ее разбивают на ряд подзадач. Решение каждой подзадачи реализуется в виде отдельной функции. Оформление подзадачи с помощью отдельной функции позволяет облегчить проектирование, реализацию, работу, а также сопровождение программы. До сих пор для описания функции создавался новый файл. Язык MATLAB позволяет размещать описания нескольких функций в одном файле. В том случае, если файл содержит описания нескольких функций, то функция, которая описана первой, называется *главной функцией*, а все остальные функции – *подфункциями*. При активизации файла управление передается на первую инструкцию в главной функции. Подфункции можно вызывать как из главной функции, так и из другой подфункции, которая располагается в том же самом файле. Подфункция имеет более высокий приоритет при вызове, чем простая функция. Следовательно, если система MATLAB при выполнении инструкций в теле функции встречает вызов функции, то сначала просматриваются имена подфункций, описанных в том же файле, а уже потом имена m-файлов в пути поиска системы MATLAB. Причем имена m-файлов будут просматриваться только в том случае, если отсутствует подфункция с этим именем. Самый высокий приоритет среди функций имеют встроенные функции. Следовательно, имена подфункций должны отличаться от встроенных функций.

Пример 6.23. Создать файл с именем `circles3.m`, в котором размещаются две функции: `circle3()` и `circle_length()`. Главная функция `circle3()` предназначена для вычисления координат точек окружности, а подфункция – длины этой окружности. Основная функция принимает три входных параметра: `r` (радиус), `x0` и `y0` (координаты центра окружности) и возвращает три выходных параметра: `x`, `y` (координаты точек окружности) и `len` (длина окружности). Параметры `x0`, `y0` и `len` являются дополнительными. В основной функции выполнить вызов подфункции. Подфункция принимает один входной параметр `radius` (радиус окружности) и возвращает один выходной параметр `len` (длина окружности).

Решение:

Листинг 6.24

```
function [x,y,len] = circle3(r,x0,y0) % основная функция
%CIRCLE3 вычисление координат окружности и определение ее длины
% Входные параметры:
% r - радиус окружности; x0, y0 - координаты центра окружности
% Выходные параметры:
% x, y - координаты точек окружности; len - длина окружности
```

Окончание листинга 6.24

```
% Проверка числа входных и выходных фактических параметров
if (nargin < 1) % отсутствуют входные фактические параметры
    error('Неправильно задано число входных фактических параметров')
end

if (nargout < 2) % выходных фактических параметров меньше двух
    error('Слишком мало выходных фактических параметров')
end

if (nargin < 2), x0 = 0; end % инициализация x0
if (nargin < 3), x0 = 0; y0 = 0; end % инициализация x0 и y0

% Вычисление координат точек окружности
t = [0:pi/64:2*pi]; % параметр
x = x0 + r*cos(t); % координаты точек окружности по оси x
y = y0 + r*sin(t); % координаты точек окружности по оси y

% Вычисление длины окружности
if (nargout == 3) % Проверка наличия третьего выходного параметра
    len = circle_length(r);
end

function [len] = circle_length(radius) % подфункция
% Вычисление длины окружности
len = 2*pi*radius; % тело подфункции
```

Примеры вызова главной функции `circle3()` из командной строки:

```
>> [x,y] = circle3(1); plot(x,y); xlabel('x'); ylabel('y');
>> axis('equal'); grid('on'); hold('on')
>> [x,y,len] = circle3(0.5,1,1); plot(x,y,'linewidth',3);
>> len
len =
    3.1416
```

Результаты вызовов функции `circle3()` представлены на рис. 6.13. В первом вызове функции `circle3()` были указаны только обязательные входные и выходные фактические параметры. В результате первого вызова были получены координаты точек окружности, радиус которой равен единице с центром в точке с координатами (0;0). С помощью функций `plot()`, `xlabel()`, `ylabel()`, `axis()`, `grid()` и `hold()` в графическом окне на основании ранее вычисленных координат отображается окружность, задаются обозначения осей абсцисс и ординат, устанавливается равный масштаб по осям, отображается сетка и включается режим наложения графиков соответственно. Во втором вызове функции `circle3()` задаются все фактические параметры. Функция возвращает значения координат точек окружности с центром в точке (1;1) и радиусом 0.5, а также значение длины окружности,

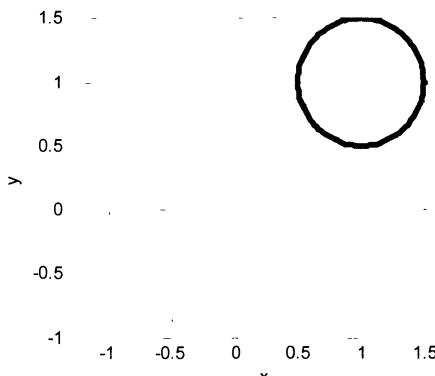


Рис. 6.13. Результаты вызова функции `circle3()`

которое записывается в переменную `len`. С помощью следующей команды в графическом окне отображается вторая окружность толщиной линии 3 пункта. Последняя команда выводит в командное окно значение длины окружности, которое находится в переменной `len`.

Использование подфункций не приводит к выигрышу во времени при выполнении всей программы по сравнению с использованием отдельных функций, так как и простые функции, и подфункции перед использованием преобразуются в р-код, размещаются в памяти, а уже затем начинают выполняться. Так как поиск необходимых файлов в пути поиска системы MATLAB выполняется быстро, то потеря времени на поиск файла является несущественной. Подфункцию следует использовать в тех случаях, когда ее тело состоит из небольшого числа инструкций или когда требуется получить один р-файл, который позволит сохранить секретность уникального алгоритма. В остальных случаях следует размещать функции в отдельных файлах.

6.4.6. Рекурсивные функции

При рассмотрении предыдущих примеров программы состояли из сценариев или функций, которые вызывали другие функции в строго иерархическом порядке. В ряде случаев при написании программы полезно иметь функции, которые вызывают сами себя напрямую или через другие функции. Такие функции принято называть *рекурсивными*.

Язык MATLAB является матричным языком и ориентирован на *итеративную схему вычислений*, в которой повторение некоторых действий реализуется с помощью структур повторения. При использовании *рекурсивной схемы вычислений* повторение действий осуществляется с помощью повторного обращения к функции. Язык MATLAB позволяет при написа-

нии программ использовать как итеративную, так и рекурсивную схему вычислений.

При использовании рекурсивной схемы вычислений решение задачи во многих случаях описывается проще и нагляднее, чем при итеративной схеме. Однако с точки зрения затрат машинного времени и памяти рекурсивные программы менее эффективны, так как требуется организовывать вызов функции и возврат из нее. Следует избегать использования рекурсивных функций, когда требуется высокая эффективность. Любую задачу, которую можно решить рекурсивно, можно решить и итеративно. Как правило, при программировании на языке MATLAB рекурсивный подход предпочитают итеративному подходу в тех случаях, когда итеративное решение не является очевидным.

В языке MATLAB существует ограничение на количество рекурсивных вызовов. Максимально допустимое число рекурсивных вызовов хранится в свойстве `RecursionLimit` корневого объекта и по умолчанию равно 500.

При рекурсивном решении задачи необходимо учитывать следующие особенности:

1. Рекурсивная функция вызывает саму себя.
2. Каждый рекурсивный вызов решает идентичную задачу меньшего размера.
3. Проверка базисных условий позволяет остановить рекурсивный процесс.
4. Одна из задач должна оказаться *базовой*.

Рассмотрим два примера: вычисление факториала и построение геометрического фрактала. Краткие сведения по фракталам приведены в практикуме 11. При решении этих задач демонстрируется различие между рекурсивными функциями, возвращающими некоторые значения, и рекурсивными функциями, не возвращающими никаких значений.

Пример 6.24. Написать рекурсивную функцию `fact()` для вычисления факториала числа N . Визуализировать процесс вычисления. Рекурсивное определение факториала числа N записывается следующим образом:

$$N! = \begin{cases} 1, & \text{если } N = 0, \\ N \times (N - 1)!, & \text{если } N > 0. \end{cases}$$

Решение:

Листинг 6.25

```
function [out_num] = fact(in_num)
%FACT вычисление факториала числа N
if in_num == 0
    out_num = 1
else
    out_num = in_num*fact(in_num-1)
end
```

Пример вызова функции `fact()`:

```
>> fact(3)
out_num =
    1
out_num =
    1
out_num =
    2
out_num =
    6
ans =
    6
```

Написанная функция соответствует четырем критериям, которые были представлены выше:

1. Функция `fact()` вызывает саму себя.
2. При каждом рекурсивном вызове функции `fact()` значение ее входного параметра уменьшается на единицу.
3. Факториал нуля функции вычисляется иначе, чем факториалы остальных чисел. Если значение входного параметра больше нуля, то генерируется рекурсивный вызов. При нулевом значении входного параметра значение функции равно 1, т.е. прекращается рекурсивный процесс. Таким образом, базовой задачей является вычисление факториала нуля.
4. В том случае, если значение входного параметра неотрицательно, то п. 2 гарантирует, что в процессе вычислений будет достигнута *базовая задача*.

Вычисление факториала целого положительного числа $N=3$ с помощью рекурсивной программы происходит согласно схеме, изображенной на рис. 6.14. По рекурсивной схеме $3!$ вычисляется как произведение $2!$ на 3. Следовательно, чтобы вычислить $3!$, необходимо найти $2!$, что требует вычисления $1!$, для которого необходимо вычислить $0!$. Последнее, согласно алгоритму, есть 1. При распознавании базовой задачи заканчивается *рекурсивный спуск*. После решения базовой задачи осуществляется *рекурсивный подъем*. Во время рекурсивного подъема вычисляются произведения значений, которые возвращаются после соответствующего вызова рекурсивной функции. Следовательно, $1!=1\times 0!=1\times 1=1\Rightarrow 2!=2\times 1=2\Rightarrow 3!=3\times 2=6$. В результате $3!=6$. Следовательно, функция `fact()` возвращает число 6, которое записывается в переменную `ans` и выводится в командное окно. В командное окно также выводятся промежуточные значения, которые возвращает функция после рекурсивных вызовов. В системе MATLAB имеется стандартная функция `factorial()`, которая используется для вычисления факториала. В этой функции реализована итеративная схема вычисления факториала положительного целого числа. При использовании итеративной схемы факториал числа N вычисляется с помощью следующей формулы: $N!=1\times 2\times 3\times \dots \times N$.

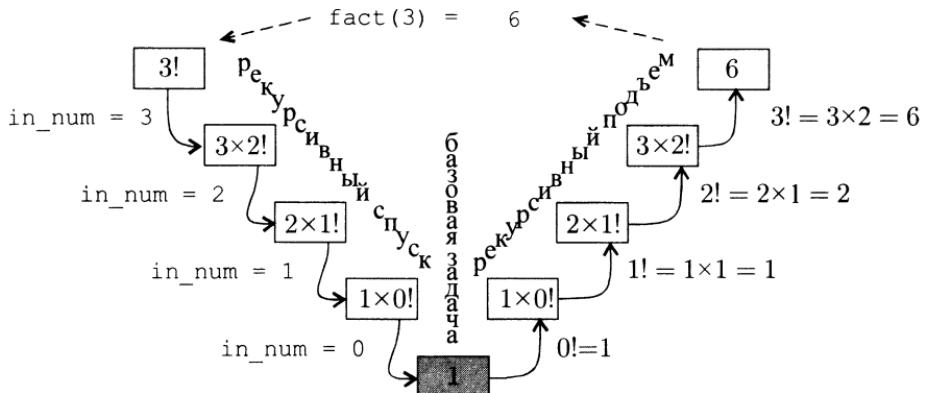


Рис. 6.14. Последовательность рекурсивных вызовов при вычислении $3!$

Глубиной рекурсии называется количество одновременно хранящихся в памяти копий рекурсивной функции. В рассмотренном примере максимальная глубина рекурсии равнялась четырем.

Пример 6.25. Написать функцию `rec_fractal()`, которая осуществляет построение геометрического фрактала, состоящего из окружностей и изображенного на рис. 6.15. В качестве входного параметра функция принимает значение, которое соответствует глубине фрактального изображения или рекурсии. В том случае, если входной параметр не задан, то по умолчанию он принимается равным четырем. Радиус начальной окружности – $r_1 = 1$. Координаты центра начальной окружности – $x_1 = 0$, $y_1 = 0$. Центры окружностей следующего уровня располагаются от центра окружности текущего уровня на расстоянии, которое равно удвоенному радиусу окружности текущего уровня $r_{i+1} = 2 \times r_i$. Угловое расстояние между соседними окружностями составляет 60° . Центром вращения окружностей является центр окружности предыдущего уровня. Радиусы окружностей следующего уровня отличаются в 0.3 раза от радиусов окружностей предыдущего уровня $r_{i+1} = 0.3 \times r_i$. Вычисление координат окружностей и их отображение в графическом окне необходимо оформить

помощью рекурсивной функции `picture()`.

Листинг 6.26

```
function [] = rec_fractal( n )
%REC_FRACTAL построение фрактала по рекурсивной схеме
if (nargin < 1), n = 4; end % Анализ числа входных параметров
fig = figure; % Создание графического окна
clFig = get(fig,'Color'); % Извлечение цвета графического окна
```

Окончание листинга 6.26

```

ax = axes(... % создание координатного пространства
    'Color', 'none',...
    'DataAspectRatio',[1 1 1],...
    'NextPlot','add',...
    'XColor', clFig,'YColor',clFig);
picture(0,0,1,2,n-1) % Вызов функции picture()
end

function [] = picture(x,y,r,r1,n)
%PICTURE рекурсивная функция
if n > 0
    % Вычисление координат точек окружности
    x_c = x + r.*cos([0:pi/32:2*pi]);
    y_c = y + r.*sin([0:pi/32:2*pi]);
    % построение окружности
    line(x_c,y_c);
    % параметры окружностей следующего уровня
    r1 = 2.0*r; r = 0.3*r;
    x1 = x + r1.*cos(pi.*[1:6]./3);
    y1 = y + r1.*sin(pi.*[1:6]./3);
    for i = 1:6
        picture(x1(i),y1(i),r,r1,n-1); % рекурсивный вызов
    end
end
end

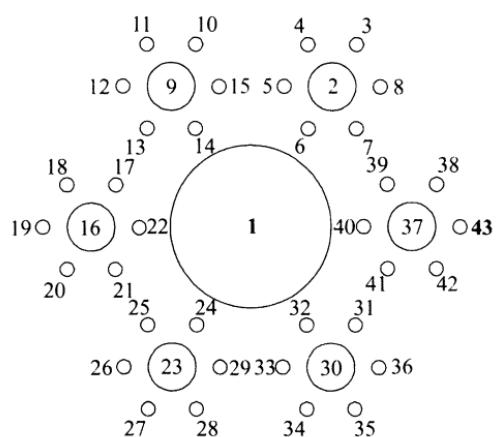
```

Представленная функция `picture()` соответствует критериям рекурсивной функции:

1. Функция `picture()` вызывает саму себя.
2. При каждом новом вызове радиус выводимой окружности уменьшается.
3. При достижении максимальной глубины рекурсии рекурсивные вызовы прекращаются.
4. Базовой задачей является построение окружности нулевой длины. Для решения этой задачи ничего не нужно делать.

В качестве альтернативы п. 3 и 4 можно использовать прекращение рекурсивных вызовов при достижении минимального радиуса окружности и ее отображения в графическом окне соответственно. Самостоятельно напишите рекурсивную функцию, в которой базовой задачей является построение окружности минимального радиуса.

Результаты вызова функции `rec_fractal(3)` представлены на рис. 6.15. На рисунке показана последовательность построения окружностей. Эта последовательность соответствует выводу окружностей в графическое окно во время рекурсивного спуска.



Р и с . 6.15. Последовательность построения окружностей при рекурсивном спуске

Первой строится окружность самого большого радиуса $r = 1$. После ее построения вычисляются параметры окружностей следующего уровня и управление передается структуре повторения `for end`, в которой реализуется шесть рекурсивных вызовов. Во время первого рекурсивного вызова ($i=1$) выполняется построение второй окружности, вычисляются параметры окружностей следующего уровня, которые будут размещаться вокруг второй окружности, и выполняется рекурсивный вызов с параметрами

первой окружности третьего уровня. Во время второго рекурсивного вызова в графическое окно выводится третья окружность, вычисляются параметры окружностей четвертого уровня, которые будут располагаться вокруг третьей окружности, и осуществляется третий рекурсивный вызов с параметрами первой окружности четвертого уровня. Так как максимальная заданная глубина фракタルного изображения равна трем, то во время третьего рекурсивного вызова не будет выполняться никаких действий. Следовательно, была выявлена базовая задача. Те же самые результаты будут при последующих пяти рекурсивных вызовах, выполняемых внутри структуры повторения. После выполнения восьмого рекурсивного вызова происходит подъем на один уровень (уровень окружности 2). Во время девятого рекурсивного вызова функции `picture()` передаются параметры окружности 4. Для окружности 4 повторяются те же самые действия, которые имели место при создании окружности 3. Рекурсивные спуски и подъемы выполняются до тех пор, пока не будет построена последняя окружность 43. После ее построения выполняется еще шесть рекурсивных вызовов, которые соответствуют шести окружностям четвертого уровня, располагающимся вокруг окружности 43, и происходит возврат в вызывающую функцию `rec_fractal()`. Таким образом, при построении окружностей во время рекурсивного спуска сначала в графическом окне по одной выводятся окружности больших радиусов, а затем соответствующие им окружности меньших радиусов. Построение

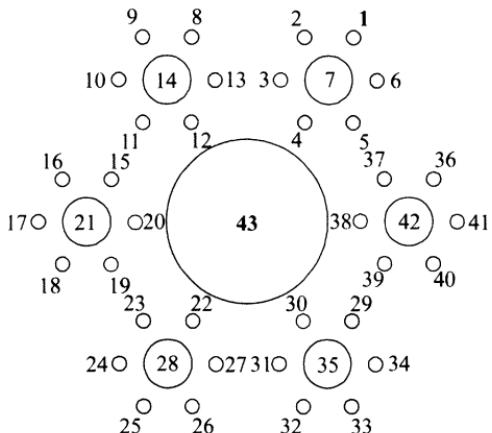
следующей окружности большего радиуса выполняется только в том случае, если построены все окружности малых радиусов, соответствующие текущей окружности большего радиуса.

Построение фрактального изображения, приведенного в примере 6.25, можно выполнять не только во время рекурсивных спусков, но и во время рекурсивных подъемов. Листинг 6.27 содержит описание рекурсивной функции `picture()`, при выполнении которой окружности строятся во время рекурсивного подъема. Единственным отличием этой функции от одноименной функции, приведенной в примере 6.25, является местоположение встроенной функции `line()`. В данном случае она располагается после структуры повторения, в которой выполняются рекурсивные вызовы.

Листинг 6.27

```
function [] = picture(x,y,r,r1,n)
%PICTURE рекурсивная функция
if n > 0
    % Вычисление координат точек окружности
    x_c = x + r.*cos([0:pi/32:2*pi]);
    y_c = y + r.*sin([0:pi/32:2*pi]);
    % параметры окружностей следующего уровня
    r1 = 2.0*r; r = 0.3*r;
    x1 = x + r1.*cos(pi.*[1:6]./3);
    y1 = y + r1.*sin(pi.*[1:6]./3);
    for i = 1:6
        picture(x1(i),y1(i),r,r1,n-1); % рекурсивный вызов
    end
    % построение окружности
    line(x_c,y_c);
end
end
```

На рис. 6.16. приведена последовательность построения окружностей во время рекурсивного подъема. Первая окружность, которая располагается на третьем уровне и имеет самый маленький радиус, отображается в графическом окне после завершения восьмого рекурсивного вызова. Следующие пять окружностей также располагаются на третьем уровне вокруг седьмой окружности, которая находится на втором уровне. Окружность самого большого радиуса будет построена после завершения вывода в графическое окно всех окружностей второго уровня. Следовательно, при отображении окружностей в графическом окне во время рекурсивного подъема сначала выводятся окружности меньших радиусов, а затем соответствующие им окружности больших радиусов. Причем окружность большего радиуса выво-



Р и с . 6.16. Последовательность построения окружностей при рекурсивном подъеме

дится только после построения всех соответствующих ей окружностей малого радиуса.

Во время построения геометрического фрактала, изображенного на рис. 6.15 и 6.16, функция `picture()` вызывалась 259 раз. Разница между числом окружностей и количеством вызовов функции является следствием наличия базовой задачи, при которой выполняется построение окружности нулевой длины, т.е. не выполняется никаких действий. Такой тип рекурсивных вызовов носит название *остаточной*, или *хвостовой*, рекурсии.

С целью повышения эффектив-

ности рекурсивных программ следует удалять хвостовую рекурсию. На листинге 6.28 представлено описание функции и подфункции `picture()`, в которой отсутствует хвостовая рекурсия, и построение геометрического фрактала выполняется во время рекурсивного спуска. После внесения изменений число окружностей соответствует числу вызовов функции. В функцию `picture()` были также внесены изменения, которые касаются цвета окружности. При построении окружностей используется семь цветов, которые соответствуют цветовой палитре `lines`. Цвет окружности зависит от ее порядкового номера.

Листинг 6.28

```
function [] = rec_fractal(n)
%REC_FRACTAL построение фрактального изображения
% Анализ входного параметра
if (nargin < 1) | (n < 1) | (n > 4), n = 4; end
fig = figure; % Создание графического окна
clFig = get(fig,'Color'); % Извлечение цвета графического окна
ax = axes(... % создание координатного пространства
'Color', 'none',...
'DataAspectRatio',[1 1 1],...
'NextPlot','add',...
'XColor', clFig,'YColor',clFig);
picture( 0, 0, 1, 2, n, 0) % Вызов функции picture()
end
```

```

function [] = picture( x, y, r, r1, n, number)
%PICTURE рекурсивная функция
    % Вычисление координат точек окружности
    x_c = x + r.*cos([0:pi/32:2*pi]);
    y_c = y + r.*sin([0:pi/32:2*pi]);
    switch rem(number,7) % Построение окружности в графическом окне
        case 0, line(x_c,y_c,'Color',[0 0 1]); % Синий
        case 1, line(x_c,y_c,'Color',[0 0.5 0]); % Темно-зеленый
        case 2, line(x_c,y_c,'Color',[1 0 0]); % Красный
        case 3, line(x_c,y_c,'Color',[0 191/255 191/255]); % Темно-голубой
        case 4, line(x_c,y_c,'Color',[191/255 0 191/255]); % Фиолетовый
        case 5, line(x_c,y_c,'Color',[191/255 191/255 0]); % Темно-желтый
        otherwise line(x_c,y_c,'Color',[0.25 0.25 0.25]); % Серый
    end
    if n - 1 > 0
        % Определение координат центра окружности и ее радиуса
        r1 = 2.0*r; r = 0.3*r;
        x1 = x + r1.*cos(pi.*[1:6]./3);
        y1 = y + r1.*sin(pi.*[1:6]./3);
        for i = 1:6
            number = number + 1;
            picture(x1(i),y1(i),r,r1,n-1,number);
        end
    end
end

```

Результаты выполнения функций, представленных на листинге 6.28, при входном параметре функции `rec_fractal()`, заданном по умолчанию, изображены на ил. 19.

Так как система MATLAB ориентирована на итеративную схему решения задач, то рассмотрим пример построения фрактального изображения из предыдущего примера с ее использованием.

Пример 6.26. Построить фрактальное изображение из предыдущего примера с помощью итеративной схемы, которую необходимо реализовать в функции `it_fractal()`. Входной параметр функции соответствует порядку геометрического фрактала.

Решение:**Листинг 6.29**

```

function it_fractal(n)
%IT_FRACTAL построение фрактала изображения при помощи итераций

if nargin < 1, n = 4; end % Анализ числа входных данных
x = 0; y = 0; r = 32; % Задание начальных условий
t = [0:pi/64:2*pi]; % Задание значений параметра t
xx = []; yy = []; % Создание массивов для хранения данных
% Вычисление координат начальной окружности
x_c = x + r*cos(t); y_c = y + r*sin(t);
fig = figure; % Создание графического окна
clFig = get(fig,'Color'); % Копирование цвета окна в переменную
ax = axes(... Создание координатного пространства
    'Color', 'none',...
    'DataAspectRatio',[1 1 1],...
    'NextPlot','add',...
    'XColor', clFig,'YColor',clFig);
line(x_c,y_c); % Построение линии
% Вычисление начальных значений для входа в структуру повторения
r1 = 2.0*r; r = 0.3*r; c = 0;
for a=1:n-1
    % определение числа отображаемых в цикле окружностей
    max_range = prod(6.*ones(1,a))/6;
    for index = 1:max_range
        % вычисление центров шести окружностей
        center_x = x + r1*cos(pi.*[1:6]./3);
        center_y = y + r1*sin(pi.*[1:6]./3);
        % вычисление радиусов окружностей
        col = r*ones(size(center_x,2),1);
        % вычисление координат окружностей
        x_c = center_x'*ones(1,size(t,2)) + col*cos(t);
        y_c = center_y'*ones(1,size(t,2)) + col*sin(t);
        % запись координат центров окружностей
        if a < n-1, xx = [xx, center_x]; yy = [yy, center_y]; end
        for k = 1:6 % вывод окружностей в графическое окно
            switch rem(a,7)
                case 6, line(x_c(k,:),y_c(k,:),'Color',[0.25 0.25 0.25]);
                case 5, line(x_c(k,:),y_c(k,:),'Color',[191/255 191/255 0]);
                case 4, line(x_c(k,:),y_c(k,:),'Color',[191/255 0 191/255]);
                case 3, line(x_c(k,:),y_c(k,:),'Color',[0 191/255 191/255]);

```

Окончание листинга 6.29

```

    case 2, line(x_c(k,:),y_c(k,:),'Color',[1 0 0]);
    case 1, line(x_c(k,:),y_c(k,:),'Color',[0 0.5 0]);
    otherwise line(x_c(k,:),y_c(k,:),'Color',[0 0 1]);
end
end
if ~((a == n-1) && (index == max_range))
    % изменение координат центра окружности предыдущего уровня,
    % относительно которой будут выводиться шесть окружностей
    % текущего уровня
    c = c + 1; x = xx(c); y = yy(c);
end
end % окончание построения окружностей соответствующего уровня
if a < n-1
    x = xx(c); y = yy(c); % изменение координат центров окружностей
    % определение расстояния между центрами окружностей
    % текущего и следующего уровней
    r1 = 2.0 * r;
    r = 0.3 * r; % радиуса окружностей следующего уровня
end
end

```

Результаты вызова функции `it_fractal()` без входного параметра представлены на ил. 20. С помощью цветов была передана последовательность построения окружностей. Первой выводится окружность самого большого радиуса, которая располагается в центре изображения. Эта окружность отображается синим цветом. При выполнении первой итерации осуществляется построение шести окружностей зеленого цвета, которые располагаются вокруг первой окружности. Во время выполнения второй итерации происходит построение 36 окружностей красного цвета, по шесть окружностей вокруг соответствующих зеленых окружностей. Третья итерация сопровождается построением 216 окружностей голубого цвета, по шесть окружностей вокруг соответствующих окружностей красного цвета.

Таким образом, при использовании итеративной схемы построение окружностей выполняется в одном направлении последовательно по уровням, начиная с первого.

В табл. 6.1 представлены сравнительные результаты времени выполнения функций `rec_fractal()` и `it_fractal()` с различными входными значениями. Данные были получены на ЭВМ с характеристиками: процессор – РИ-400, объем ОЗУ – 128 Мб. В скобках указано число вызовов функции `picture()`.

Таблица 6.1

Результаты времени выполнения функций

n	4	5	6
rec fractal(n)	0.781 сек. (259)	2.914 сек. (1555)	15.321 сек. (9331)
it fractal(n)	0.671 сек.	2.313 сек.	12.348 сек

Из табл. 6.1 видно, что с увеличением значения входного параметра функций итеративная схема дает больший выигрыш по времени выполнения, так как отсутствуют рекурсивные вызовы функции. Однако рекурсивный алгоритм в данном случае является более наглядным, чем итеративный.

6.5. ФАЙЛЫ

Ввод данных с клавиатуры и вывод результатов расчета в командное или графические окна удобны, когда работа выполняется с временными данными, максимальное время жизни которых составляет один сеанс работы с системой MATLAB. Во время завершения сеанса работы закрываются созданные графические окна и командное окно, а также удаляются все данные, которые размещаются в основной рабочей области системы MATLAB. С целью длительного хранения данных и их последующего использования в других сеансах работы с системой MATLAB применяются *дисковые файлы*. *Дисковым файлом* называется логически связанная совокупность данных, которая имеет имя и располагается на внешнем носителе. Дисковый файл является частным случаем понятия файла, который является промежуточным объектом между программой и физическим устройством, куда или откуда осуществляется ввод или вывод данных. В дальнейшем изложении под файлом понимается только дисковый файл. Содержимое файла остается неизменным, пока его не модифицируют человек или программа. Использование файлов позволяет обрабатывать большие объемы информации и сократить время на ввод одних и тех же данных.

Система MATLAB предназначена главным образом для обработки информации, представленной в матричном виде. Если обработка данных выполняется только средствами системы MATLAB, то сохранять данные, расположенные в основной рабочей области, следует в файле с расширением mat, а графические результаты – в файле с расширением fig. Однако возможности системы MATLAB не ограничиваются только обработкой информации. Система MATLAB позволяет обмениваться информацией с устройствами, которыми могут быть экран, клавиатура, мышь, дисковый файл, принтер, осциллограф и т.д. Для обмена данными с устройствами через последовательный порт, а также с автономными (stand-alone) приложения-

ми, которые не позволяют импортировать данные в форматах, поддерживаемых системой MATLAB, следует использовать низкоуровневые функции ввода-вывода. При использовании этих функций обмен данными осуществляется с помощью файлов.

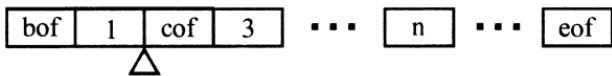
6.5.1. Основные понятия

Все файлы условно принято делить на две группы: *файлы последовательного доступа* и *файлы прямого доступа*. В файлах последовательного доступа данные считываются из файла и записываются в файл последовательно. Получить доступ к определенной записи можно только после последовательного перебора предыдущих записей. Файл последовательного доступа можно сравнить с информацией на магнитной ленте (аудио- или видеокассета). В языке MATLAB файлом с последовательным доступом является *текстовый файл*. В файлах прямого доступа данные могут записываться и считываются в произвольном порядке. Следовательно, любая запись в таком файле доступна напрямую. Файл прямого доступа можно сравнить с данными, которые размещаются в оперативной памяти компьютера или на накопителях на гибких и жестких магнитных дисках. В языке MATLAB файлом с прямым доступом является *двоичный файл*. В случае работы с двоичным файлом считываение или запись данных можно осуществлять как последовательно, так и произвольно.

Работа с файлами состоит из трех этапов: *открытие файла, считывание или запись данных и закрытие файла*. При открытии файла определяется режим работы с файлом: является ли он текстовым или двоичным, открывается ли файл для чтения, записи или выполнения обеих этих операций. По умолчанию во время открытия файла устанавливается двоичный режим работы с доступом только для чтения. Открытие файла сопровождается созданием *логического файла*, который ставится в соответствие *физическому файлу*, и идентификатора или *дескриптора файла*, указывающего на логический файл. После операции открытия физического файла все действия выполняются с логическим файлом, а результаты этих действий отражаются на состоянии физического файла. После выполнения необходимых операций с файлом он должен быть закрыт. Если не выполнить операцию закрытия файла, то некоторая часть данных может быть потеряна. Логический и физический файлы соединены между собой с помощью *буфера ввода-вывода*. Запись в физический файл происходит только после полного заполнения буфера ввода-вывода. После выполнения записи в физический файл данных из буфера ввода-вывода он автоматически очищается. При закрытии файла вся информация, которая содержится в не полностью заполненном буфере обмена, записывается в файл.

В языке MATLAB существует два стандартных файла вывода, которые не требуют операции открытия, связаны с экраном и всегда доступны во время сеанса работы с системой MATLAB. Первым файлом с идентификатором, равным единице, является файл, ассоциированный со стандартным потоком вывода. Вторым файлом, идентификатор которого равен двум, является файл, связанный с потоком ошибок. *Потоком* называется последовательность байт данных. Поток байт, принимаемый программой, называется *потоком ввода*, а поток, который посыпается программой на устройства, называется *потоком вывода*. Для организации работы с окном Command Window система MATLAB использует еще один стандартный поток. Этим потоком является поток ввода, связанный с клавиатурой.

Файл состоит из записей, которые расположены последовательно. *Записью* называют единицу обмена данными между программой и файлом. Нулевая запись содержит метку *начала файла* bof (Beginning Of File). Последняя запись в файле представляет собой символ *конца файла* eof (End Of File). При работе с файлами необходимо иметь информацию о текущей позиции, из которой будет осуществляться считывание данных или в которую будет происходить их запись. Текущая позиция в файле обозначается с помощью *маркера текущей позиции*, изображенного на рис. 6.17. Маркер текущей позиции не является физическим элементом, а используется только для указания на текущую позицию в файле cof (Current position in File). Маркер перемещается при переходе от записи к записи. На рис. 6.17 используются следующие обозначения: прямоугольники – записи, треугольник – маркер текущего положения в файле, n – n-я запись в файле.



Р и с . 6.17. Маркер текущей позиции в файле

Проверка конца файла реализуется с помощью функции feof(fid). Функция feof() принимает один параметр, который является идентификатором файла. В том случае, если достигнут конец файла, функция возвращает единичное значение. В остальных случаях результат равен нулю.

Язык MATLAB включает набор низкоуровневых функций, которые предназначены для работы с файлами. Эти функции являются расширением соответствующих функций из стандартной библиотеки языка С, которые учитывают векторные операции, присущие языку MATLAB.

6.5.2. Текстовые файлы

Текстовые файлы представляют собой последовательность символов в формате ASCII. Информация, содержащаяся в текстовом файле, может

рассматриваться как одна строка или как совокупность строк. Во втором случае каждая строка завершается специальной *меткой конца строки*. В операционной системе Windows в качестве метки конца строки используется последовательность двух символов из таблицы ASCII с номерами 13 (возврат каретки – Carriage Return) и 10 (переход на новую строку – Line Feed). В текстовых файлах с помощью символов может отображаться не только текстовая, но и числовая информация.

6.5.2.1. Основные операции

Основными операциями при работе с текстовыми файлами являются открытие файла, чтение данных из файла, запись данных в файл и закрытие файла.

Открытие текстового файла осуществляется с помощью функции `fopen()`. Во время вызова функции `fopen()` с целью открытия текстового файла следует задавать два входных и один выходной параметр:

```
[fid] = fopen('имя_файла', 'режим_работы_с_файлом')
```

Первый входной фактический параметр является именем открываемого текстового файла, а второй параметр определяет режим работы с файлом. Возможные значения второго параметра перечислены в табл. 6.2. Выходной параметр представляет собой *идентификатор*, или дескриптор, файла (`fid – file identifier`). При задании имени файла следует указывать полный путь. Если путь не указан, то система MATLAB осуществляет поиск в текущей директории.

Таблица 6.2

Способы доступа к текстовым файлам

Обозначение	Описание
'rt'	Открытие файла для чтения. Аббревиатура <code>rt</code> является сокращенной записью словосочетания <code>read text</code> (читать текст). В том случае, если файл не найден, функция <code>fopen()</code> возвращает значение <code>-1</code>
'wt'	Открытие файла для записи. Буква <code>w</code> обозначает слово <code>write</code> (записать). Если файл не существует, то он создается. Если файл существует, то он удаляется без предупреждения, а взамен создается новый пустой файл с тем же именем
'at'	Открытие файла для добавления данных. Буква <code>a</code> в обозначении способа является первой буквой слова <code>append</code> (добавлять). В случае отсутствия файла он создается. Если файл существует, новые данные дописываются в его конец

Окончание табл. 6.2

Обозначение	Описание
'rt+'	Открытие файла для чтения и записи. Если файл не существует, то функция возвращает значение -1. Если файл существует, новые данные записываются с начала файла, затирая уже имеющуюся информацию
'wt+'	Открытие файла для чтения и записи. Если файл не существует, то он создается. Если файл существует, то он очищается и новые данные записываются в очищенный файл
'at+'	Открытие файла для чтения и добавления данных. Если файл не существует, то он создается. Если файл существует, новые данные дописываются в его конец

После открытия файла необходимо всегда проверять значение его дескриптора, а операции, связанные с файлом, выполнять только в том случае, если его дескриптор не равен -1. Функция `fopen()` может вернуть значение -1 в следующих ситуациях:

- задание неправильного (несуществующего) имени файла при открытии с доступом для чтения 'rt' или чтения и записи 'rt+';
- попытка открыть файл на диске, который не готов к вводу-выводу (диск не отформатирован);
- попытка открыть файл в несуществующей директории или на несуществующем диске.

При работе с текстовыми файлами существует два способа чтения и записи информации: бесформатный (построчный) и форматный. При использовании бесформатного способа работы с текстовыми файлами чтение и запись осуществляются с помощью функций `fgetl()` или `fgets()` и `fprintf()` соответственно. Различие между функциями `fgetl()` и `fgets()` заключается в том, что при чтении строки с помощью функции `fgetl()` из нее удаляется символ конца строки.

После окончания работы с файлом его необходимо закрыть. Закрытие файла осуществляется с помощью вызова функции `fclose(fid)`, входной параметр которой является дескриптором файла.

Пример 6.27. Создать функцию `linechar()`, в которой выполняется подсчет числа строк и количества символов в файле. В качестве входного параметра задать имя файла. Функция возвращает два значения: число строк и число символов. Во время выполнения функции считанные из файла строки необходимо выводить в командное окно. Перед вызовом функции `linechar()` создать с помощью текстового редактора текстовый файл с расширением txt в текущей директории системы MATLAB и во время вызова задать его имя.

Решение:**Листинг 6.30**

```
function [line_number,char_number] = linechar(filename)
%LINECHAR вычисление числа строк и символов в текстовом файле
% line_number - число строк в файле
% char_number - число символов в файле
% filename - имя файла

% Анализ количества входных фактических параметров
if (nargin ~= 1)
    error('Число входных фактических параметров не равно 1')
end
% Проверка класса входного параметра
if ~ischar(filename)
    error('Входной параметр должен быть строкой')
end
% Открытие файла для чтения
[fid, mes] = fopen(filename,'r');
if (fid == -1) % Проверка правильности открытия файла
    disp('Ошибка во время открытия файла')
    error(mes) % Вывод описания причины возникновения ошибки
else % файл открыт
    % Задание начальных значений выходным параметрам
    line_number = 0; char_number = 0;
    % Считывание строк из файла
    while ~feof(fid) % сравнение с символом конца файла
        string = fgetl(fid); % считывание строки из файла
        % увеличение счетчиков строк и символов
        line_number = line_number + 1;
        char_number = char_number + size(string,2);
        if strmatch(string,'') % Вывод строки в командное окно
            disp(' ') % Вывод пустой строки
        else
            disp(string)
        end
    end
    fclose(fid); % закрытие файла
end
```

Результаты вызова функции linechar():

```
>> [lines, characters] = linechar('nietzsche.txt')
Плохо отплачивает тот учителю,
кто навсегда остается только учеником.
```

```
Заратустра, о дарящей добродетели
lines =
    4
characters =
    101
```

Если в теле функции linechar() заменить вызов функции fgetl() на fgets(), то результат будет иметь вид:

```
>> [lines, characters] = linechar('nietzsche.txt')
Плохо отплачивает тот учителю,
кто навсегда остается только учеником.
```

```
Заратустра, о дарящей добродетели
lines =
    4
characters =
    109
```

В первом случае, когда используется функция fgetl(), при считывании строк из текстового файла количество символов в файле равно 101, а во втором случае – 109. Так как файл содержит четыре метки окончания строки, то функция fgetl() удалила четыре пары символов CR и LF. Система MATLAB при выводе строк в командное окно автоматически добавляет символ перевода на новую строку. Следовательно, во втором случае строки, содержащие этот символ, выводятся в командном окне через строку. Наряду с добавлением символа конца строки система MATLAB проверяет, является ли строка пустой. Если строка не содержит ни одного символа, то система MATLAB не выводит ее в командное окно. Так как третья строка в текстовом файле nietcshe.txt является пустой (содержит только символ перевода на новую строку), то при ее чтении с помощью функции fgetl() с последующим выводом в командное окно система MATLAB ее не отобразит. Для отображения пустых строк в функцию linechar() включена структура выбора if else end, в которой пустая строка заменяется на строку, содержащую символ «пробел».

Пример 6.28. Написать функцию modestxt(), с помощью которой можно исследовать влияние различных режимов работы с текстовыми файлами на их содержимое.

Решение:

Листинг 6.31

```
function [] = modestxt()
%MODESTXT пример открытия файла в разных режимах

% Ввод имени файла
filename = input('Введите полное имя файла: ','s');
flag = true; % установка флага
while flag
    while 1 % Ввод режима работы с файлом
        mode = input('Введите режим работы с файлом: ','s');
        if strcmp(mode,'rt') || strcmp(mode,'rt+') || ...
            strcmp(mode,'wt') || strcmp(mode,'wt+') || ...
            strcmp(mode,'at') || strcmp(mode,'at+')
            break % Выход из цикла, если режим задан правильно
        else
            disp('Ошибка при задании режима работы с файлом')
        end
    end
    fid = fopen(filename,mode); % Открытие файла
    if fid ~= -1
        if ~strcmp(mode,'rt') % Изменение содержимого файла
            string = input('Введите строку текста:\n','s');
            string = [string '\n']; % Символ конца строки
            fprintf(fid,string); % Запись строки в файл
            fclose(fid); % Закрытие файла (очистка буфера)
            fid = fopen(filename,'rt'); % Открытие файла для чтения
        end
        % Вывод содержимого файла в командное окно
        while ~feof(fid) % Сравнение с символом конца файла
            string = fgetl(fid); % Считывание строки из файла
            if strcmp(string,''), disp(' '), else disp(string), end
        end
        fclose(fid); % Закрытие файла
    else
        fprintf(2,'Ошибка при задании имени файла\n')
        flag = false; % Выход из программы
    end
end
```

Окончание листинга 6.31

```

if flag %
    answer = input('Прекратить выполнение программы? (y/n)', 's');
    if strcmp(answer, 'y') || strcmp(answer, 'Y'), flag = false; end
end
end

```

Результаты вызова функции modestxt():

```

>> modestxt
Введите полное имя файла: nietzsche1
Введите режим работы с файлом: tr
Ошибка при задании режима работы с файлом
Введите режим работы с файлом: rt
Ошибка при задании имени файла
>> modestxt
Введите полное имя файла: nietzsche1
Введите режим работы с файлом: rt+
Ошибка при задании имени файла
>> modestxt
Введите полное имя файла: nietzsche1
Введите режим работы с файлом: wt
Введите строку текста:
Friedrich Nietzsche
Friedrich Nietzsche
Прекратить выполнение программы? (y/n) y
>> modestxt
Введите полное имя файла: nietzsche.txt
Введите режим работы с файлом: at
Введите строку текста:
Friedrich Nietzsche
Плохо отплачивает тот учителю,
кто навсегда остается только учеником.

Заратустра, о дарящей добродетели
Friedrich Nietzsche
Прекратить выполнение программы? (y/n) n
Введите режим работы с файлом: rt+
Введите строку текста:
Friedrich Nietzsche
Friedrich Nietzsche
    учителю,
кто навсегда остается только учеником.

Заратустра, о дарящей добродетели
Friedrich Nietzsche
Прекратить выполнение программы? (y/n) n

```

```
| Введите режим работы с файлом: wt+
| Введите строку текста:
| Friedrich Nietzsche
| Friedrich Nietzsche
| Прекратить выполнение программы? (y/n) y
```

При вызове функции `modestxt()` появляется приглашение для ввода имени файла. После ввода имени файла (`nietzsche1`) в командном окне появляется приглашение для задания режима работы с файлом. Так как в первом случае заданная последовательность символов не совпадает с обозначениями режимов работы, приведенными в табл. 6.2, то в командное окно выводится сообщение об ошибке и появляется приглашение для повторного ввода режима работы. Во втором случае задается режим работы текстового файла с доступом только для чтения. Так как файла `nietzsche1` нет, то в командное окно выводится сообщение об ошибочном задании имени файла. Сообщение об ошибке передается стандартному потоку ошибок, посредством которого оно выводится на экран. Передача сообщения потоку ошибок осуществляется с помощью вызова функции `fprintf()`. В качестве первого входного параметра функция принимает идентификатор файла, куда направляется текстовое сообщение. Само текстовое сообщение передается с помощью второго входного параметра. В конце текстового сообщения присутствует символ конца строки `\n`, используемый в языке MATLAB для перехода на новую строку. После вывода сообщения в командное окно функция `modestxt()` завершает свое выполнение.

Во время второго вызова функции выполняется попытка открыть несуществующий файл `nietzsche1` для чтения и записи (`rt+`). Как и в предыдущем случае, в командное окно будет выведено сообщение об ошибке с последующим завершением работы функции `modestxt()`.

При третьем вызове функции `modestxt()` файл `nietzsche1` открывается только для записи. Так как этого файла `nietzsche1` не было в текущей директории во время вызова функции, он создается, и маркер текущей позиции устанавливается в начало файла. После открытия файла в командное окно выводится приглашение для ввода строки, которая после нажатия на клавишу `Enter` записывается в заданный файл. После записи строки в файл он закрывается и открывается снова только для чтения с целью отображения его содержимого в командном окне. Файл содержит одну строку с именем немецкого философа Фридриха Ницше. После вывода содержимого файла `nietzsche1` в командное окно он закрывается и в командном окне появляется вопрос: «Прекратить выполнение программы?» с возможными вариантами ответов (`y/n`). При вводе `y` (`yes – да`) завершается выполнение функции `modestxt()`.

Во время четвертого вызова функции `modestxt()` открывается файл `nietzsche.txt` в режиме добавления текстовых данных. Вводится имя филосо-

фа, которое записывается в конец файла. После добавления текста файл закрывается и открывается снова, но уже в режиме только для чтения. Из файла считывается все содержимое, которое выводится в командное окно. Из выведенного содержимого файла видно, что фамилия философа была добавлена в конец файла. После отображения файла в командном окне он закрывается, и в командное окно выводится вопрос о завершении программы. Так как в ответ на вопрос вводится символ `n`, то программа продолжает выполняться. В командном окне появляется новое приглашение о вводе режима работы с файлом `nietzsche.txt`. Задается режим чтения и записи (`rt+`). После задания режима работы с файлом вводится строка `Friedrich Nietzsche`, которая записывается в начало файла. После записи строки в файл он закрывается и открывается снова только для чтения с последующим выводом содержимого файла в командное окно. Выведенное содержимое файла свидетельствует о том, что строка записывается в начало файла, удаляя все символы, на место которых она записывается. Так как перед записью в файл в конец введенной строки добавляется символ переноса на новую строку, то последующий текст выводится с новой строки. В следующем цикле выполнения функции `modestxt()` файл `nietzsche.txt` открывается для чтения и записи с предварительной очисткой его содержимого (`wt+`). В файл записывается та же строка, что и в предыдущих случаях. При выводе содержимого файла выводится только одна строка, которая содержит имя немецкого философа. После положительного ответа на вопрос о завершении работы с программой выполнение функции `modestxt()` прекращается.

6.5.2.2. Форматированный вывод данных

Функции ввода и вывода, которые рассматривались до сих пор, работали только со строкой символов, содержащей текстовую информацию. В большинстве случаев файлы содержат не только текстовые, но и числовые данные. Для считывания и записи как текстовой, так и числовой информации в языке MATLAB применяются функции `fscanf()` и `fprintf()` соответственно. Эти функции являются расширением одноименных функций из стандартной библиотеки языка C, которые учитывают матричную структуру данных, используемых в языке MATLAB.

Функция `fprintf()` предназначена для форматированного вывода данных в файл. Заголовок функции имеет следующий вид:

```
count = fprintf(fid,format,varargin)
```

Первый входной параметр (`fid`) является идентификатором файла, куда направляется вывод информации. Если идентификатор файла отсутствует, то вывод направляется в командное окно.

Второй входной параметр (`format`) представляет собой строку формата, которая указывает функции `fprintf()`, как оформлять выводимые данные.

Необязательные входные параметры (`varargin`) после строки формата – это переменные и выражения, значения которых нужно вывести.

Выходной параметр (`count`) содержит число, которое равно числу байт, записанных в файл.

Строка формата может содержать спецификации вывода, которые начинаются с символа `%` и состоят из компонентов:

`% [флаг] [ширина_поля] [. [точность]] символ_формата`

В квадратных скобках указаны необязательные компоненты. Единственной обязательной частью спецификации вывода (кроме символа `%`) является символ формата. Символы формата и их значения перечислены в табл 6.3.

Таблица 6.3

Символы формата в спецификации вывода

Символ	Значение
<code>o</code>	Целое восьмеричное число без знака. <code>>> num = 27; fprintf('%o', num)</code> <code>33</code>
<code>u</code>	Целое десятичное число без знака. <code>>> num = 27; fprintf('%u', num)</code> <code>27</code>
<code>x, X</code>	Целое шестнадцатеричное число без знака в нижнем (x) или в верхнем (X) регистре. <code>>> num = 27; fprintf('%x %X', num, num)</code> <code>1b 1B</code>
<code>d, i</code>	Целое десятичное число со знаком. В том случае, если число содержит дробную часть, оно записывается в экспоненциальном формате. <code>>> num1 = -13; num2 = 15; fprintf('%i %d', num1, num2)</code> <code>-13 15</code> <code>>> num = 27.27; fprintf('%d', num)</code> <code>2.727000e+001</code>
<code>f</code>	Вещественное число в фиксированном формате. Если не указана точность, то выводится шесть цифр после десятичной точки. <code>>> num = 27.01; fprintf('%f', num)</code> <code>27.010000</code>

Окончание табл. 6.3

Символ	Значение
e, E	Вещественное число в экспоненциальном формате. Задавая спецификацию e или E, можно управлять регистром символов. Если точность не указана, то после десятичной точки выводится шесть цифр. <pre>>> num = 27.01; fprintf('%e %E',num,num) 2.701000e+001 2.701000E+001</pre>
g, G	Используется формат e, E или f. Формат e или E применяется в том случае, если показатель степени меньше -4 или больше, чем точность (по умолчанию 6). В противном случае используется формат f. <pre>>> num = 0.00002701; fprintf('%g',num) 2.701e-005 >> num = 0.0002701; fprintf('%g',num) 0.0002701 >> num = 270100; fprintf('%G',num) 270100 >> num = 2701000; fprintf('%G',num) 2.701E+006</pre>
c	Одиночный символ. Если переменная содержит строку, то в файл будет записана строка <pre>>> character = 'Y'; fprintf('%c',character) Y</pre>
s	Строка символов <pre>>> string = 'Yes'; fprintf('%s',string) Yes</pre>

Спецификация ширины поля задает минимальное число символов, которое должно быть выведено. Эта спецификация может иметь одну из следующих форм:

- десятичное целое число, не начинающееся с нуля. Выводимое значение дополняется пробелами слева, чтобы заполнить указанную ширину. Например,

```
>> number = 10; fprintf('%5d', number)
      10
```
- десятичное целое число, начинающееся с нуля. Выводимое значение дополняется нулями слева, чтобы заполнить указанную ширину. Например,

```
>> number = 10; fprintf('%05d', number)
00010
```

- символ «*». Рассматриваются два аргумента в списке значений. В качестве значения ширины берется первый аргумент, а значение второго выводится. Например,

```
>> width = 4; number = 10; fprintf('%*d', width, number)  
10
```

Спецификация точности состоит из десятичной точки «.», за которой может следовать число. Спецификация точности применима только к символам формата e, E, f, g, G и s. Она указывает количество цифр после десятичной точки, которые следует вывести, а в случае спецификации s – количество выводимых символов. Если после десятичной точки ничего не указано, то принимается нулевая точность. В случае использования фиксированного формата и нулевой точности число округляется до ближайшего целого и выводится в соответствующий файл. Например,

```
>> number = 10.56; fprintf('%12.4f', number)  
10.5600  
>> number = 10.56; fprintf('%12.4e', number)  
1.0560e+001  
>> number = 10.56; fprintf('%12.4g', number)  
10.56  
>> number = 10.56; fprintf('%12.e', number)  
1e+001  
>> number = 10.56; fprintf('%12.f', number)  
11
```

Последним необязательным компонентом спецификации вывода является флаг, который должен располагаться непосредственно после символа «%». Существуют четыре допустимых флага:

- символ «←». Выводимое значение выравнивается по правому краю поля вместо левого, как это делается по умолчанию. Например,

```
>> number = 27; fprintf('%5d', number)  
27  
>> number = 27; fprintf('%-5d', number)  
27
```

- символ «+». Число обязательно выводится со знаком «плюс» или «минус». Например,

```
>> number = 27; fprintf('%+5d', number)  
+27  
>> number = 27; fprintf('%+-5d', number)  
+27
```

- символ «пробел» означает, что перед положительным числом всегда должен стоять пробел. Например,

```
>> number = 27; fprintf('%04d% 04d',number,number)
0027 027
```

- символ «#». Применяется только к спецификациям x, X или o. Указывает, что ненулевые шестнадцатеричные значения (в случаях x или X) должны отображаться с символами 0x или 0X, а восьмеричные (в случае o) – с нулем перед ним. Например,

```
>> number = 27; fprintf('%#6x', number)
0x1b
>> number = 27; fprintf('%#6X', number)
0X1B
>> number = 27; fprintf('%#6o', number)
033
```

При управлении выводом в строке формата могут находиться не только спецификации вывода, но и управляющие символы, представляющие собой escape-последовательности. В табл. 6.4 представлены некоторые управляющие символы, используемые в языке MATLAB.

Т а б л и ц а 6.4
Управляющие символы

Символ	Значение
\a	Звуковой сигнал
\b	Возврат на один символ назад >> string = 'ПоП'; fprintf('%s\b%s',string,string) ПоПоП
\n	Перевод строки >> string = 'ПоП'; fprintf('%s\n%s',string,string) ПоП ПоП
\t	Горизонтальная табуляция >> string = 'ПоП'; fprintf('%s\t%s',string,string) ПоП ПоП

Пример 6.29. Написать функцию num2tbl(), с помощью которой выполняется построение таблицы, состоящей из трех столбцов. В первом столбце выводятся числа в десятичной системе счисления, во втором – соответствующие числа в восьмеричной системе счисления и в третьем – соответствующие числа в шестнадцатеричной системе счисления. Числа необходимо выровнять по левому краю с отступом в один пробел. Вывести заголовки каждого столбца и заголовок таблицы, а также отобразить ее границы.

Решение:

Листинг 6.32

```
function [] = num2tbl(number)
%NUM2TBL запись чисел в табличном виде

% Заголовок таблицы
numbers      = 'Числа';
decimal      = 'Десятичные';
octal        = 'Восьмеричные';
hexadecimal = 'Шестнадцатеричные';
% Массив (3xnumber) десятичных чисел
num = [0:number;0:number;0:number];
fid = fopen('numbers.txt','wt'); % открытие файла для записи
if (fid ~= -1)
    % Запись в файл numbers.txt таблицы чисел от 0 до 16
    fprintf(fid,'-----\n');
    fprintf(fid,'|%-26s          |%n',numbers);
    fprintf(fid,'|-----|-----|\n');
    fprintf(fid,'|%-13s|%-15s|%-20s|\n',decimal,octal,hexadecimal);
    fprintf(fid,'|-----|-----|\n');
    fprintf(fid,'|%-13d| %#14o| %#19x|\n',num);
    fprintf(fid,'-----\n');
    fclose(fid); % Закрытие файла
    fid = fopen('numbers.txt','rt'); % Открытие файла для чтения
    while ~feof(fid)
        string = fgetl(fid); % Считывание строки из файла
        if strcmp(string,''), disp(' '), else disp(string), end
    end
    fclose(fid); % Закрытие файла
else
    fprintf(2,'Ошибка при открытии файла.\n')
end
```

Результаты вызова функции num2tbl():

```
>> num2tbl(16)
```

Числа		
Десятичные	Восьмеричные	Шестнадцатеричные
0	0	0
1	01	0x1
2	02	0x2
3	03	0x3
4	04	0x4
5	05	0x5
6	06	0x6
7	07	0x7
8	010	0x8
9	011	0x9
10	012	0xa
11	013	0xb
12	014	0xc
13	015	0xd
14	016	0xe
15	017	0xf
16	020	0x10

Так как функция `fprintf()` поддерживает операции с массивами, то при записи массива `num` в файл отсутствует структура повторения, без которой нельзя было обойтись в случае использования языка С.

6.5.2.3. Форматированный ввод данных

Форматированный ввод данных реализуется с помощью функции `fscanf()`, которая является расширением одноименной функции из стандартной библиотеки языка С с учетом матричной структуры данных, которая используется в языке MATLAB. Заголовок функции `fscanf()` имеет следующий вид:

```
[A,count] = fscanf(fid,format,size)
```

Первый обязательный входной параметр (`fid`) является дескриптором файла, который возвращается функцией `fopen()`. Для получения данных из командного окна, ассоциированного с дескриптором «1», следует использовать встроенную функцию `input()`.

Второй обязательный входной параметр (`format`) представляет собой строку формата, которая задает правила считывания данных из файла. В строке формата допускаются следующие элементы:

- пробелы и символы табуляции, которые игнорируются, но позволяют сделать строку более удобочитаемой;
- спецификации ввода, состоящие из символа % и следующих за ним обязательных и необязательных компонент в определенном порядке. Обязательным компонентом в спецификации ввода является специальный символ, который указывает функции `fscanf()`, как ей интерпретировать вводимые данные. После считывания данных они соответствующим образом преобразуются в класс `double`. Специальные символы совпадают со специальными символами, используемыми для вывода данных, которые приведены в табл. 6.3. Одна спецификация ввода соответствует одному вводимому значению. В строке формата должна быть хотя бы одна спецификация ввода.

Если считанное значение является символом класса `char` (%c), то он преобразуется в число согласно номеру в таблице ASCII. В случае строки символов (%s) осуществляется преобразование всех символов, образующих строку.

Между символом % и обязательным символом формата вводимых данных могут присутствовать необязательные компоненты.

Первым необязательным компонентом в спецификации ввода является флаг пропуска поля ввода (*), который следует сразу за символом %. Если он присутствует в спецификации ввода, то функция `fscanf()` выполняет ввод соответствующего значения, но результат ввода не возвращается функцией.

Следующим элементом может быть ширина поля, которая выражается десятичным числом и обозначает количество символов в поле ввода. Полем ввода называется последовательность непустых символов, которая считается законченной, как только встретился первый пустой символ (пробел, табуляция и т.п.) или как только полностью прочитана указанная ширина поля.

Завершает список необязательных компонентов модификатор точности, который может быть символом h (`short`) или l (`long`).

Третий дополнительный входной параметр (`size`) позволяет задать размер считываемых из файла данных. Параметр `size` может иметь следующий вид:

- число n. Считывается n значений из файла в указанном формате. Все прочитанные значения записываются в выходной параметр в виде вектора-столбца;
- специальное значение inf. Выполняется считывание данных из файла до тех, пока не встретится метка конца файла. Все прочитанные данные записываются в вектор-столбец;
- два числа внутри квадратных скобок [m, n]. Выполняется считывание данных в матричном виде. При считывании данных формируется матри-

ца $n \times m$, где n – число строк, а m – число столбцов. На месте переменной n может присутствовать специальное значение `inf`.

Первый выходной параметр (`A`) является именем массива класса `double`, куда производится запись данных при завершении работы функции `fscanf()`.

Второй необязательный выходной параметр (`count`) содержит количество прочитанных элементов данных.

Пример 6.30. Написать функцию `tbl2num()`, которая считывает из таблицы, созданной в предыдущем примере, заданное число строк и выводит в командное окно прочитанные данные с учетом системы счисления. При выполнении считывания данных необходимо пропустить заголовки таблицы и каждого столбца, которые располагаются на пяти первых строках.

Решение:

Листинг 6.33

```
function [numbers] = tbl2num(file_name, row_number)
%TBL2NUM преобразование таблицы в массив чисел
% file_name - имя файла, который содержит таблицу данных
% row_number - число строк, которые необходимо считать из таблицы
% numbers    - считанные из таблицы данные, представленные в виде
%               массива строк

% Проверка класса входного параметра
if ~ischar(file_name)
    error('Ошибка при задании входного параметра');
end
% Открытие файла для чтения
fid = fopen(file_name,'rt');
if fid ~= -1
    fprintf(1, '\nФайл %s открыт в режиме %s\n', file_name, 'rt')
    % Пропуск заголовков таблицы и колонок (пять строк)
    fgetl(fid); fgetl(fid); fgetl(fid); fgetl(fid); fgetl(fid);
    % Считывание данных из таблицы
    [num,count]= fscanf(fid, '%*s%d %*s%14o%*s %x%*s', [3, row_number]);
    % Закрытие файла
    fclose(fid);
if count ~= row_number*3
    error('Ошибка при чтении данных из файла');
end
```

```

else
    % Создание разделительного столбца
    spaces = char(ones(1,size(num,2)).*32)';
    % Формирование выходного массива с учетом системы счисления
    numbers = ...
        [num2str(num(1,:)'),spaces,dec2base(num(2,:)',8),spaces,...
        dec2hex(num(3,:'))];
end
else
    string = ['Ошибка при открытии файла ' file_name];
    error(string)
end

```

Результаты вызова функции `tbl2num()`:

```

>> tbl2num('numbers.txt',11)

Файл numbers.txt открыт в режиме rt
ans =
  0 00 0
  1 01 1
  2 02 2
  3 03 3
  4 04 4
  5 05 5
  6 06 6
  7 07 7
  8 10 8
  9 11 9
 10 12 A

```

Текстовые файлы представляют собой файлы с последовательным доступом данных. Следовательно, для считывания данных, располагающихся с шестой строки таблицы, необходимо считать первые пять строк. Для считывания первых пяти строк используется функция `fgetl()`.

Начиная с шестой строки, считывание данных осуществляется с помощью функции `fscanf()`, в которой задается формат считывания данных. Так как ограничителем столбцов таблицы является символ «|», то он пропускается при чтении данных (`%*s`). Первый столбец содержит целое число в десятичной системе счисления (`%d`), второй столбец – целое число в восьмеричной системе счисления (`%14o`) и третий столбец – целое число в шестнадцатеричной системе счисления (`%x`). При считывании второго столбца указывается его ширина.

Третий входной параметр в функцию `fscanf()` [3 `row_count`] содержит размер считываемых из файла данных. Из файла происходит считывание данных, располагающихся в трех столбцах и в `row_count` строках. Считанные данные размещаются в переменной `num` в виде трех строк и `row_count` столбцов. Следовательно, при считывании данных происходит смена строк и столбцов (транспонирование). Кроме смены строк и столбцов все числа, записанные в различных системах счисления, преобразуются в числа десятичной системы счисления.

Для обратного преобразования в шестнадцатеричную систему счисления используется функция `num2hex()`, а в восьмеричную систему счисления – функция `num2base()`, второй входной параметр которой является числом 8. После окончания выполнения эти функции возвращают массив символьного класса. При формировании выходного массива, содержащего данные в трех системах счисления, в качестве разделителя столбцов использовался массив пробелов `spaces`.

6.5.3. Двоичные файлы

Язык MATLAB позволяет работать с двоичными файлами, в которых данные хранятся в двоичном представлении, без специальной служебной информации. Все данные записываются ичитываются в неизменном виде, без разделения на строки. Символ конца строки (`\n`) воспринимается, как и любой другой байт данных. Двоичные файлы являются компактными и используются для хранения больших объемов информации.

6.5.3.1. Основные операции

Основными операциями при работе с двоичными файлами, так же как и с текстовыми, являются открытие файла, чтение данных из файла, запись данных в файл и закрытие файла.

Открытие файла выполняется с помощью функции `fopen()`. Во время вызова функции `fopen()` файла следует задавать два входных и один выходной параметр:

```
[fid] = fopen('имя_файла', 'режим_работы_с_файлом')
```

Первый входной фактический параметр является именем открываемого двоичного файла, а второй параметр определяет режим работы с файлом. Режимы работы с двоичными файлами совпадают с режимами работы с текстовыми файлами, которые представлены в табл. 6.1. Для указания на работу с двоичным, а не с текстовым файлом, необходимо в спецификации режима

работы с файлом заменить символ `t` на `b` или просто удалить символ `t`. Например, инструкция

```
fid = fopen('matrix.bin','rb')
```

открывает двоичный файл `matrix.bin` только для чтения.

Выходной параметр `fid` является дескриптором файла, который используется в последующих операциях с файлом. В том случае, если дескриптор файла содержит значение `-1`, значит, при открытии файла произошла ошибка. Возможные варианты возникновения ошибок описаны при рассмотрении текстовых файлов.

После выполнения необходимых операций с файлом его необходимо закрыть. Закрытие файла осуществляется, как и в случае работы с текстовыми файлами, с помощью вызова функции `fclose(fid)`, входной параметр которой является дескриптором файла.

Сохранение данных в двоичном файле выполняется с помощью функции `fwrite()`, один из возможных вариантов вызовов которой имеет вид

```
count = fwrite(fid,A,precision)
```

Первым обязательным входным параметром является дескриптор файла, куда будет производиться запись данных. Данные содержатся в массиве `A`, который является вторым обязательным входным параметром функции. Третий необязательный входной параметр `precision` является строкой и задает формат (класс на языке MATLAB или тип на языках FORTRAN и C), в котором будут записываться данные. Возможные варианты формата представлены в табл. 6.5. Формат можно задавать двумя способами:

- задавать имена классов числовых и символьных данных, применяемые в языке MATLAB (наиболее предпочтительный способ, так как не зависит от платформы);
- использовать ключевые слова, применяемые в языках программирования высокого уровня, – FORTRAN и C.

В том случае, если параметр `precision` не указан, то данные записываются в файл в формате `uchar`.

В переменную `count` при завершении работы функции `fwrite()` записывается число прочитанных из файла элементов.

Считывание данных из двоичного файла выполняется с помощью функции `fread()`, один из возможных вариантов вызова которой имеет вид

```
[A,count] = fread(fid,size,precision)
```

Первый входной параметр (`fid`) обозначает файл, откуда будет производиться считывание данных.

Второй параметр (`size`) определяет количество считываемых элементов и порядок их размещения в результирующем массиве `A`. Параметр `size` мо-

жет быть числом, специальным значением `inf` или вектором `[m n]`, где `m` – число столбцов в считываемом массиве, а `n` – число строк. При передаче прочитанных данных в результирующий массив `A` строки и столбцы меняются местами. Если параметр `size` не указан, то он приравнивается к `inf`. Следовательно, чтение данных будет выполняться до тех пор, пока не будет достигнут конец файла. Все прочитанные данные будут возвращены функцией в виде вектора-столбца.

Таблица 6.5

Варианты форматов данных в двоичном файле

Ключевое слово на языке MATLAB	Ключевое слово на языке FORTRAN	Ключевое слово на языке C	Кол-во байт	Диапазон значений
<code>schar</code>		<code>signed char</code>	1	<code>-128...127</code>
<code>uchar</code>		<code>unsigned char</code>	1	<code>0...255</code>
<code>int8</code>	<code>integer*1</code>	<code>schar</code>	1	<code>-128...127</code>
<code>int16</code>	<code>integer*2</code>	<code>short</code>	2	<code>-32 768...32 767</code>
<code>int32</code>	<code>integer*4</code>	<code>int, long</code>	4	<code>-2 147 483 648...2 147 483 647</code>
<code>int64</code>			8	<code>-9 223 372 036 854 775 808...9 223 372 036 854 775 807</code>
<code>uint8</code>		<code>uchar</code>	1	<code>0...255</code>
<code>uint16</code>		<code>ushort</code>	2	<code>0...65 535</code>
<code>uint32</code>		<code>uint, ulong</code>	4	<code>0...4 294 967 295</code>
<code>uint64</code>			8	<code>0...18 446 744 073 709 551 615</code>
<code>single</code>	<code>real*4</code>	<code>float</code>	4	$\approx \pm 10^{-38} \dots 10^{38}$
<code>double</code>	<code>real*8</code>	<code>double</code>	8	$\approx \pm 10^{-308} \dots 10^{308}$

Третий входной параметр задает формат считываемых данных. Возможные варианты форматов представлены в табл. 6.5. При считывании данных необходимо указывать тот же самый формат, который применялся при записи этих данных в файл. При считывании числовых данных происходит автом

матическое приведение классов данных к классу double. В том случае, если по какой-либо причине необходимо оставить исходный формат данных, отличный от класса double, или преобразовать данные в другой формат, необходимо использовать полную запись третьего параметра, которая имеет следующий вид:

```
'исходный_формат=>целевой_формат'
```

Например, запись 'int32=>single' указывает на то, что необходимо считать массив 4-байтовых целых чисел со знаком и преобразовать их в соответствующие вещественные числа одинарной точности. По умолчанию целевой формат равен double.

Первый выходной параметр (A) является результирующим массивом, куда передаются прочитанные данные. Размер массива A задается входным параметром size. Если параметр size не указан, то прочитанные данные передаются в массив A в виде вектора-столбца.

Второй необязательный выходной параметр (count) содержит количество прочитанных элементов функцией fread(). В случае ошибки или достижения конца файла значение в переменной count может быть меньше, чем число элементов, задаваемое параметром size.

Пример 6.31. Написать функцию `binfile()`, во время выполнения которой открывается для записи указанный в качестве первого входного параметра двоичный файл и в него записываются массивы данных, содержащиеся в остальных входных параметрах. Второй входной параметр записывается в формате uchar, третий – в формате двойной точности double, а четвертый – в формате int32. После записи данных файл необходимо закрыть и открыть снова, но на этот раз в режиме чтения. Прочитать три массива из файла в переменные strstr, AA и bb соответственно. При чтении первого массива сохранить формат исходных данных, а при чтении третьего массива преобразовать данные в класс uint32.

Решение:

Листинг 6.34

```
function [strstr,AA,bb] = binfile(file_name,str,A,b)
%TXTFILE запись данных в двоичный файл
% file_name      - имя файла, куда будут записываться данные
% str, A, b      - массивы входных данных
% strstr, AA, bb - массивы выходных данных
% Анализ входных данных
if nargin < 4
    error('Должно быть четыре входных параметра');
end
```

Продолжение листинга 6.34

```

if (~ischar(file_name) || ~ischar(str))
    err_str = ['Первые два входных параметра должны быть '...
               'символьного класса'];
    error(err_str);
end
if (~isnumeric(A) || ~isnumeric(b))
    err_str = ['Третий и четвертый входные параметры'...
               'должны быть арифметического класса'];
    error(err_str);
end
% Открытие файла для записи
fid = fopen(file_name,'wb');
if (fid ~= -1)
    % Определение размера переменных
    colstr = size(str,2);
    [rowA colA] = size(A);
    [rowb colb] = size(b);
    % Запись строки в формате uchar
    fwrite(fid,str);
    % Запись матрицы A в файл в формате двойной точности
    fwrite(fid,A,'double');
    % Запись переменной b в формате int32
    fwrite(fid,b,'int32');
    fclose(fid); % Закрытие файла
    fid = fopen(file_name,'rb'); % Открытие файла только для чтения
    % Считывание данных в переменную strstr
    [strstr, count] = fread(fid,colstr,'uchar=>uchar');
    if (count < colstr)
        fclose(fid); % Закрытие файла
        error('Ошибка при чтении текстовых данных из файла')
    else
        strstr = char(strstr)';
    end
    % Считывание данных в переменную AA
    [AA count] = fread(fid,[rowA colA],'double');
    % Проверка количества считанных элементов в переменную AA
    if (count < rowA.*colA)
        fclose(fid); % Закрытие файла
        error('Ошибка при чтении данных из файла в массив AA');
    end

```

```
% Считывание данных в переменную bb
[bb count] = fread(fid,[rowb colb],'int32=>uint32');
% Проверка количества считанных элементов в переменную bb
if (count < rowb.*colb)
    fclose(fid); % Закрытие файла
    error('Ошибка при чтении данных из файла в массив bb');
end
fclose(fid); % Закрытие файла
else
    error('Ошибка при открытии файла');
end
```

Результаты вызова функции `binfile()`:

```
>> str = 'Двоичный файл';
>> A=[1.1 1.2 1.3; 2.1 2.2 2.3; 3.1 3.2 3.3];
>> b=[9;10;11];
>> [a1,a2,a3] = binfile('data.bin',str,[A A],b-10)
a1 =
Двоичный файл
a2 =
    1.1000    1.2000    1.3000    1.1000    1.2000    1.3000
    2.1000    2.2000    2.3000    2.1000    2.2000    2.3000
    3.1000    3.2000    3.3000    3.1000    3.2000    3.3000
a3 =
    0
    0
    1
```

При выполнении функции создается двоичный файл `data.bin`, в который записывается строка, состоящая из 13 символов, матрица 3×6 и вектор 3×1 . Символы строки записываются в формате `uchar`, значения элементов матрицы – в формате `double`, а значения элементов вектора – в формате `uint32`. Следовательно, размер двоичного файла равен 169 ($13 + 144 + 12$) байтам. При считывании из двоичного файла значений элементов вектора изменялся класс данных с `int32` на `uint32`. Так как в диапазон чисел класса `uint32` не входят отрицательные числа, то все считанные из файла отрицательные числа преобразуются в нули. В том случае, если считываемое число превышает максимально возможное число целевого класса, функция `fread()` возвратит максимально возможное число целевого класса.

При работе с двоичными файлами запись и считывание информации могут происходить как по одному байту, так и по блокам. В приведенном примере использовался блочный ввод-вывод. В некоторых случаях двоичные

файлы содержат данные одного класса или типа. Такие файлы принято называть *типовыми файлами*.

6.5.3.2. Произвольный доступ

При работе с файлами встречаются ситуации, когда необходимо считывать данные, расположенные в произвольных местах этого файла. Например, файл содержит десять наборов данных, которые получены в результате проведенного эксперимента с различными начальными условиями. В зависимости от текущих требований во время работы с программой необходимо выполнить считывание из файла соответствующего набора данных. Если необходимо последовательно считать данные из девятого и шестого наборов, то при последовательном доступе к файлу перед считыванием девятого набора данных надо считать и удалить восемь первых наборов данных, а перед считыванием шестого набора – вернуться в начало файла и считать с последующим удалением пять первых наборов. Так как обмен информацией с дисковыми файлами является одной из самых медленных операций, выполняемых программой, то эффективность ее работы при считывании произвольного блока данных из файла с последовательным доступом является низкой. С целью повышения эффективности работы программы, которая постоянно обменивается информацией с файлами, необходимо использовать произвольный доступ к данным, расположенным в файле. Произвольный доступ означает, что данные можно считывать из файла или записывать в файл в любой позиции без перебора всех предыдущих байт. Язык MATLAB разрешает произвольный доступ к данным только при работе с двоичными файлами.

Считывание и запись данных в файл выполняются в текущую позицию в файле, которая определяется с помощью маркера текущей позиции. При последовательном доступе к файлу перемещение маркера текущей позиции выполняется автоматически. При произвольном доступе маркер текущей позиции необходимо перемещать вручную. Перемещение маркера выполняется с помощью функции `fseek()`, вызов которой имеет следующий вид:

```
status = fseek(fid,offset,origin)
```

Первый входной параметр (`fid`) обозначает файл, в котором будет осуществляться перемещение маркера текущей позиции.

Второй параметр (`offset`) определяет расстояние, на которое необходимо переместить маркер. Если `offset>0`, то перемещение происходит по направлению к концу файла. При `offset=0` положение маркера не изменяется. В том случае, если `offset<0`, маркер перемещается к началу файла.

Третий параметр (`origin`) задает точку отсчета и может иметь одно из трех значений строкового класса: '`'bof'`', '`'cof'`' и '`'eof'`'. В первом случае

перемещение маркера текущей позиции будет выполняться относительно начала файла, во втором – относительно текущей позиции и в третьем случае – относительно конца файла.

Выходной параметр `status` содержит число 0 или -1, которое возвращается функцией. Если функция возвращает нулевое значение, то перемещение маркера выполнено успешно. В противном случае при перемещении маркера возникла ошибка и после завершения выполнения функции `fseek()` он не находится в желаемой позиции. Ошибка при выполнении функции `fseek()` может возникнуть в том случае, если перемещение маркера выполняется за граничные записи файла, которыми являются метки его начала и конца. Получить описание ошибки и ее номер можно с помощью вызова функции `ferror()`.

Частным случаем функции `fseek()` является функция `frewind()`, при выполнении которой маркер текущей позиции перемещается в начало файла.

Определить положение маркера текущей позиции можно с помощью функции `ftell()`, которая принимает один входной параметр, являющийся идентификатором файла.

Пример 6.32. Написать функцию `mag3file()`, при вызове которой последовательно выполняются следующие действия:

- создается магический квадрат $M(3 \times 3)$;
- сохраняются значения элементов созданной матрицы в двоичном файле `magic3.bin` в формате `uint16`;
- читываются из файла последовательно шесть элементов в следующем порядке: шестой, первый, второй, третий, восьмой и седьмой;
- выводятся в командное окно магический квадрат и значения элементов матрицы, считанных из файла.

Решение:

Листинг 6.35

```
function [] = mag3file()
%MAG3FILE запись магического квадрата M(3x3) в файл

M = magic(3); % Создание магического квадрата
% Имя формата, в котором записываются и считываются данные
class_name = 'uint16';
class_size = 2; % Размер класса в байтах
fid = fopen('magic3.bin','wb'); % Открытие файла для записи
if (fid ~= -1)
    fwrite(fid,M,class_name); % Запись магического квадрата в файл
    fclose(fid); % Закрытие файла
    fid = fopen('magic3.bin','rb'); % Открытие файла для чтения
```

Окончание листинга 6.3.5

```

status = fseek(fid,5*class_size,'bof'); % Перемещение маркера
if (status ~= -1)
    A=fread(fid,1,class_name);% Считывание значения шестого элемента
else
    fclose(fid); % Закрытие файла
    error(ferror(fid)); % Выход из функции
end
frewind(fid); % Перемещение маркера в начало файла
% Считывание первых трех элементов
A = [A fread(fid,[1 3],class_name)];
% Перемещение маркера к восьмому элементу
status = fseek(fid,-2*class_size,'eof');
if (status ~= -1)
    % Считывание значения восьмого элемента
    A = [A fread(fid,1,class_name)];
else
    fclose(fid); % Закрытие файла
    error(ferror(fid)); % Выход из функции
end
% Перемещение маркера к седьмому элементу
status = fseek(fid,-2*class_size,'cof');
if (status ~= -1)
    % Считывание значения седьмого элемента
    A=[A fread(fid,1,class_name)];
else
    fclose(fid); % Закрытие файла
    error(ferror(fid)); % Выход из функции
end
fclose(fid);
M, A % Вывод магического квадрата и считанных из файла данных
else
    error('Ошибка при открытии файла');
end

```

Результаты вызова функции mag3file():

```

>> mag3file
M =
     8      1      6
     3      5      7
     4      9      2
A =
     9      8      3      4      7      6

```

Процесс считывания шестого, первого, второго третьего, восьмого и седьмого значений элементов матрицы из двоичного файла состоит из двух действий:

- установка маркера текущей позиции в необходимую позицию;
- считывание необходимого числа байт из файла с последующим их преобразованием в необходимый формат и присвоением полученного числа выходному фактическому параметру.

При перемещении маркера необходимо учитывать размер формата данных. Так как значения элементов матрицы сохранялись в формате uint16, то данные располагаются в двух байтах. Следовательно, для считывания значения шестого элемента матрицы необходимо переместить маркер текущей позиции на десять байт от начала файла. При считывании данных маркер текущий позиции автоматически перемещается по направлению к концу файла на число байт, равное формату данных. Таким образом, для считывания значения седьмого элемента матрицы после того, как было считано значение восьмого элемента, необходимо переместить маркер по направлению к началу файла на удвоенное число байт, соответствующее формату сохраненных данных.

После каждого перемещения маркера текущей позиции, выполняемого функцией `fseek()`, необходимо проверять значение, которое возвращает функция. Если возвращаемое значение равно -1 , то при перемещении маркера произошла ошибка. В этом случае необходимо принять соответствующие меры. В рассматриваемом примере при наличии ошибки файл закрывается, выполнение функции `mag3file()` завершается и в командное окно выводится сообщение об ошибке с причиной ее возникновения.

6.6. ВЫВОДЫ К ГЛАВЕ 6

В гл. 6 описан язык программирования MATLAB, который является интерпретируемым высокогоревневым языком программирования 4-го поколения. Язык MATLAB позволяет решать научные и технические задачи в численном виде, используя структурный и объектно-ориентированные стили программирования.

При использовании структурного стиля программирования перед описанием алгоритма решения поставленной задачи на конкретном языке программирования (в данном случае – на языке MATLAB) его (алгоритм) необходимо изобразить в виде блок-схемы, применяя следующие правила:

- разработка программы начинается с простейшей блок-схемы;
- каждый прямоугольник (действие) может быть заменен двумя последовательными прямоугольниками (более детальными действиями);

- каждый прямоугольник (действие) может быть заменен любой структурой управления;
- последние два правила могут применяться неограниченно и в любом порядке.

Во время выполнения третьего правила применяют следующие структуры управления:

- единственного выбора `if end`;
- двойного выбора `if else end`;
- тройного выбора `if elseif else end`;
- множественного выбора `switch case end`;
- повторения с параметром `for end`;
- повторения с условием `while end`.

Первые четыре структуры являются структурами выбора, последние две – структурами повторения. Структура последовательного выполнения инструкций встроена в язык MATLAB. Инструкции, образующие выражения, выполняются одна за другой согласно их приоритету. Изменить последовательность выполнения инструкций, входящих в выражение, можно с помощью круглых скобок.

При написании программ на языке MATLAB для реализации выбора следует использовать логическое индексирование. Применение логического индексирования приводит к выигрышу в скорости выполнения инструкций по сравнению со структурами выбора, которые свойственны языкам третьего поколения. Структуры выбора при программировании на языке MATLAB следует использовать при вводе или выводе данных в командное окно, а также при проверке числа заданных входных и выходных параметров во время вызова функции.

При реализации алгоритмов на языке программирования MATLAB следует по возможности избегать использования структур повторения. Решить целый ряд математических задач без использования структур повторения можно с помощью применения операции внешнего произведения.

При решении сложных задач, состоящих из нескольких подзадач, необходимо решение каждой подзадачи описывать в виде отдельной функции. Данные между подзадачами передаются с помощью входных и выходных параметров функций. Действия, которые выполняются внутри одной функции, являются невидимыми для других функций. При решении задач можно использовать целый ряд функций:

- стандартные функции, поставляемые вместе с системой MATLAB (встроенные в ядро системы, расположенные в стандартных библиотеках в виде m-файлов, p-фалов или mex-файлов);
- специализированные функции, созданные в компании *MathWorks Inc.* или третьими фирмами и организованные в виде пакетов расширения (Toolbox);

- свои собственные функции, написанные на языке MATLAB или на языках С и FORTRAN с последующей компиляцией последних в тех-файлы.

При использовании собственных функций, написанных на языке MATLAB, необходимо создать описание функции и организовать ее вызов.

Описание функции состоит из четырех частей:

- заголовок функции;
- строка быстрой помощи;
- строки помощи;
- тело функции.

Заголовок включает списки входных и выходных параметров, а также имя функции. Стока быстрой помощи содержит емкое описание назначения функции, которое отображается в командном окне во время поиска. Строки помощи содержат детальное описание назначения функции, ее входных и выходных параметров, а также ее атрибутов (фамилии авторов, номер версии и т.д.). Тело функции содержит инструкции, которые описывают на языке MATLAB решение задачи.

Вызов функции можно организовать как из командной строки, так и во время выполнении другой функции. Во время вызова функции значения входных фактических параметров передаются входным формальным параметрам, а при окончании выполнения инструкций, расположенных в теле функции, значения выходных формальных параметров передаются в выходные фактические параметры. Функция в языке MATLAB является универсальным видом подпрограмм, так как позволяет работать как с входными и выходными параметрами, так и без них. Следовательно, существуют четыре комбинации вызова функции:

- с входными и выходными параметрами;
- только с входными параметрами;
- только с выходными параметрами;
- без входных и выходных параметров.

Первый случай является наиболее распространенным. Второй случай используется для отображения информации в графическом или командном окне. Третий случай применяется при получении данных от пользователя и передачи их с помощью выходных параметров в программу. Последний случай является менее распространенным, так как содержит автономные инструкции, которые могут работать только с глобальными переменными или статическими переменными этой функции.

Во время вызова функции язык MATLAB позволяет варьировать число входных и выходных параметров при их фиксированном максимальном количестве. Дополнительными входными параметрами являются параметры, которые можно не указывать во время вызова. Если дополнительный параметр не указан, то ему необходимо присвоить значение по умолчанию. Дополнительные параметры должны располагаться в конце списка формальных

параметров, так как в языке MATLAB используется только позиционная передача параметров.

Язык MATLAB позволяет задавать произвольное число входных и выходных параметров. Эта особенность используется главным образом для создания объектов дескрипторной графики и задания их свойствам необходимых значений, которые отличаются от значений, принимаемых по умолчанию.

При работе с переменными различают классы памяти и область видимости переменных. Класс памяти переменной определяет период или время жизни, в течение которого эта переменная существует в памяти. Областью действия или видимости переменной называется та часть функции или сценария, в которой на переменную можно сослаться. Существуют три области действия: область действия сценарий, область действия функция и область действия системы MATLAB.

Существует два класса памяти переменных: автоматический класс памяти с локальным временем жизни и статический класс памяти с глобальным временем жизни. К автоматическому классу памяти с локальным временем жизни относятся переменные, создаваемые в теле функции и существующие только во время ее выполнения. Переменные статического класса памяти с глобальным временем жизни бывают двух видов: глобальные переменные и локальные сохраняемые переменные. Глобальные переменные размещаются в глобальной рабочей области и доступны из различных функций. Локальные сохраняемые переменные создаются внутри функции во время ее первого вызова и сохраняются в памяти между ее вызовами. Локальные сохраняемые переменные удаляются из памяти во время удаления из памяти функции.

Несмотря на то что язык программирования MATLAB ориентирован на итеративную схему решения задач, он также поддерживает и рекурсивную схему. В итеративной схеме повторение действий реализуется с помощью структур повторения, а в рекурсивной схеме – с помощью повторного вызова функции. При использовании рекурсивной схемы решение задачи во многих случаях описывается проще и нагляднее, чем при итеративной схеме. Однако с точки зрения затрат машинного времени и памяти рекурсивные программы менее эффективны. Как правило, при программировании на языке MATLAB рекурсивный подход предпочитают итеративному подходу в тех случаях, когда итеративное решение не является очевидным.

Язык MATLAB поддерживает низкоуровневые операции с текстовыми и двоичными файлами: открытие файла, считывание и/или запись данных и закрытие файла. Работа с файлами может быть организована в различных режимах. Низкоуровневые операции следует использовать только для обмена информацией со специализированными приложениями, которые не поддерживают стандартные форматы данных, применяемые в системе MATLAB.

ПРАКТИКУМЫ

Все, чему мы хотим научиться, мы учимся, делая.

Аристотель. Этика Никомахи II (325 г. до н. э.)

Практикумы 1, 2. ИНТЕРФЕЙС СИСТЕМЫ MATLAB

Цель: Знакомство и получение навыков работы с основными элементами интерфейса системы MATLAB.

Рабочий стол (Desktop)

Для загрузки системы MATLAB необходимо дважды щелкнуть на иконке MATLAB, расположенной на рабочем столе, или выполнить следующую последовательность действий: из панели задач выбрать Пуск → Программы → MATLAB 6.5 → MATLAB 6.5. После загрузки MATLAB появляется Рабочий стол MATLAB (рис. П-1.1), содержащий визуальные инструменты для управления файлами, переменными и приложениями, связанными с системой MATLAB.

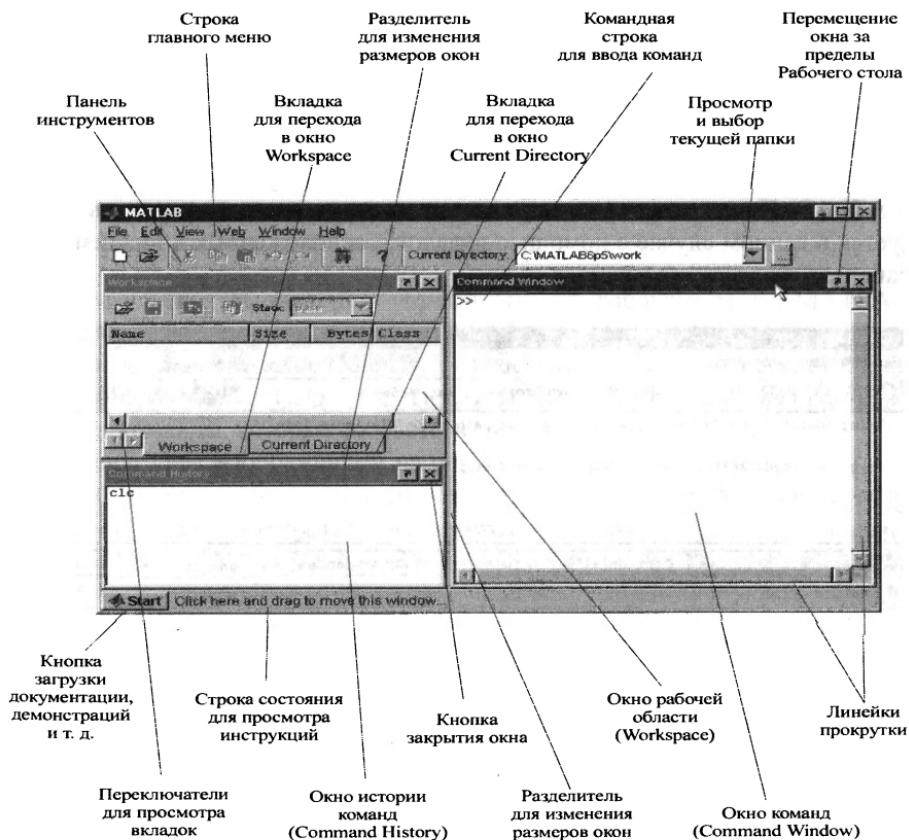


Рис. П-1.1. Рабочий стол системы MATLAB

Рабочий стол, изображенный на рис. П-1.1, состоит из строки заголовка, строки главного меню, панели инструментов, области для размещения окон инstrumentальных средств и строки состояния с кнопкой **Start**.

С помощью рабочего стола можно управлять следующими окнами инструментальных средств:

- окном запуска приложений (Launch Pad);
- окном команд (Command Window);
- окном истории команд (Command History);
- окном просмотра помощи (Help Browser);
- окном просмотра текущей директории (Current Directory Browser);
- окном просмотра рабочей области (Workspace Browser);
- окном Редактора данных (Array Editor);
- Редактором m-файлов (Editor/Debugger);
- Профайлером (Profiler), который служит для оценки быстродействия команд.

На рабочем столе может присутствовать любое сочетание инструментальных средств, рассмотренных выше. На рис. П-1.2 представлен вариант рабочего стола, на котором располагаются семь инструментальных средств с активным командным окном, занимающим весь рабочий стол системы MATLAB. Переключение между окнами инструментальных средств реализуется в данном случае с помощью вкладок, расположенных в нижней части рабочего стола.

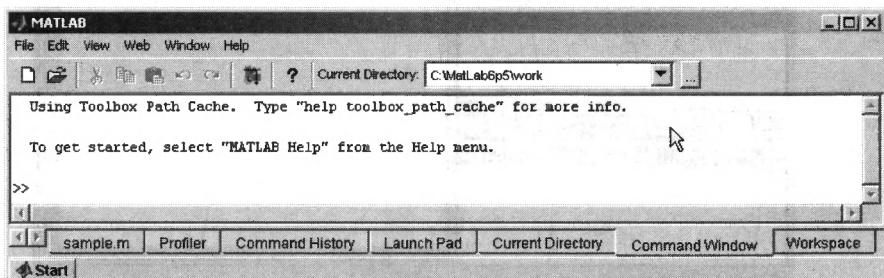


Рис. П-1.2 Вариант расположения инструментальных средств на рабочем столе

Помимо указанных выше инструментов, в систему MATLAB для визуализации результатов вычислений входят графические окна (**Figures**), которые не управляются посредством рабочего стола.

Используя вертикальные и горизонтальные разделители, можно изменять размер окон инструментальных средств в пределах рабочего стола. Наряду с изменением размеров окон система MATLAB предоставляет возможность

их перемещения по рабочему столу. Перемещение окон реализуется с помощью перетаскивания строки заголовка соответствующего окна. При перемещении окна его новое положение обозначается контуром. Переключение между окнами реализуется с помощью щелчка левой кнопки мыши в области соответствующего окна или на одной из вкладок, если нужное окно полностью закрыто другим окном. Используя кнопки закрытия () и отделения окна от рабочего стола () можно закрывать окно инструментального средства и отделять его от рабочего стола с последующим его перемещением по экрану соответственно.

Упражнение 1. Работа с окнами инструментальных средств.

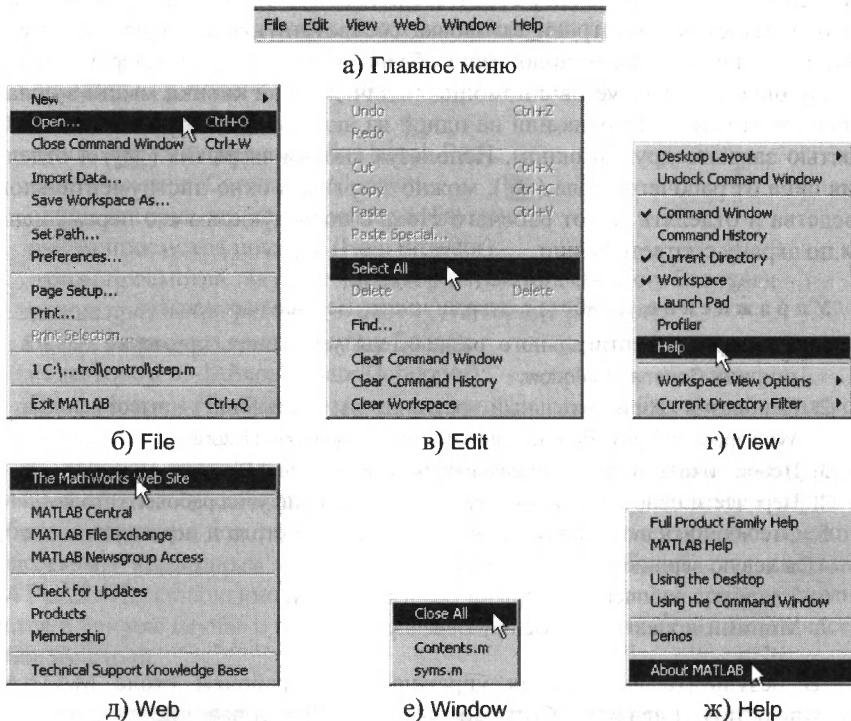
1. С помощью вертикального разделителя уменьшить примерно на 1/3 ширину Command Window.
2. Используя горизонтальный разделитель, установить высоту окна Workspace в 3 раза больше высоты окна Command History.
3. Переключиться между окнами Workspace и Current Directory.
4. Перенести окно Current Directory в правый нижний угол рабочего стола.
5. Отсоединить окно Current Directory от рабочего стола и переместить его в левую верхнюю часть экрана.
6. Удалить с рабочего стола окно Command History.
7. Минимизировать окно Current Directory.

В результате выполнения упражнения на рабочем столе MATLAB должны остаться два окна: Command Window и Workspace.

Строка главного меню (Main Menu) состоит из шести пунктов: File, Edit, View, Web, Window и Help (рис. П-1.3, а). Каждый пункт главного меню включает команды, которые соответствуют его названию.

В меню View, изображенном на рис. П-1.3, г, размещаются команды, которые позволяют отображать, скрывать, отделять и прикреплять инструментальные средства к рабочему столу системы MATLAB.

Команда Desktop Layout открывает меню, в котором размещаются команды, предназначенные для отображения на рабочем столе предустановленных комбинаций инструментальных средств. Команда Default возвращает внешний вид рабочего стола, который принят по умолчанию (рис. П-1.1). Данное расположение инструментальных средств является наиболее используемым при комплексной работе с системой MATLAB. Команда Command Window Only удаляет с рабочего стола все инструментальные средства, кроме командного окна. Интерфейс становится похожим на версии MATLAB 4.x и 5.x. Данную особенность полезно использовать, если работа с системой MATLAB осуществляется только с помощью командного окна. Команда Simple служит для отображения на рабочем столе двух инструментальных средств: окна



Р и с . П-1.3. Меню рабочего стола системы MATLAB

истории команд – с левой стороны и окна команд – с правой стороны. Данная комбинация инструментальных средств на рабочем столе является более предпочтительной, чем одно окно команд, так как позволяет с помощью окна предыстории просматривать ранее введенные команды и при необходимости повторно выполнять различные последовательности этих команд. Команды Short History и Tall History предназначены для отображения инструментальных средств Command Window, соединенных Current Directory и Workspace, а также Command History, где последнее инструментальное средство в первом случае занимает небольшое пространство рабочего стола, а во втором случае располагается по всей длине левого края рабочего стола. Эти варианты являются наименее применимыми, так как окно команд, куда вводятся команды и выводятся результаты их выполнения в текстовом виде, занимает мало места на рабочем столе. Команда Five Panels отображает на рабочем столе пять панелей или окон инструментальных средств: Command Window, Current Directory, Workspace, Command History и Launch Pad. Данную

особенность не следует использовать, так как в версии MATLAB 6.5 появился более быстрый и интуитивный путь к возможностям Launch Pad, который реализован с помощью кнопки Start.

Команда Undock активное окно отделяет текущее активное окно от рабочего стола и размещает его в поле экрана. В случае открепленного от рабочего стола окна эта команда будет заменена командой Dock окно, которая прикрепляет это окно к рабочему столу.

Следующие семь команд (Command Window, Command History, Current Directory, Workspace, Launch Pad, Profiler и Help) управляют выводом соответствующего инструментального средства на экран. Индикатором присутствия инструмента на экране является «✓», расположенная слева от имени команды. Система MATLAB помнит последнее расположение инструментального средства и отображает его на том же самом месте при задании соответствующей команды.

Последние две команды – Workspace View Options и Current Directory Filter – позволяют настраивать отображение информации в окнах Workspace и Current Directory соответственно. Эти команды появляются в меню View только в том случае, если соответствующие инструментальные средства присутствуют на экране.

Упражнение 2. Обзор команд меню View.

1. Просмотреть по порядку результаты выполнения всех команд меню View.
2. Вывести на рабочий стол различные сочетания инструментальных средств.
3. Возвратить вид рабочего стола, принятый по умолчанию.

В меню File, представленном на рис. П-1.3, б, располагаются команды, которые позволяют осуществлять операции на файловом уровне. Меню File разделено по функциональным признакам на шесть частей.

С помощью подменю New можно открыть приложение для создания нового m-файла (Editor/Debugger), динамической модели (SIMULINK) и графического интерфейса пользователя (GUIDE – Graphic User Interface Development Environment), а также создать новое графическое окно для визуализации результатов вычислений. Команда Open... открывает стандартное диалоговое окно, в котором можно выбрать необходимый файл. Открыть диалоговое окно Open можно с помощью комбинации клавиш Ctrl+O. Команда Close Command Window закрывает командное окно. Эта команда является контекстно-зависимой, так как закрывает текущее внутри рабочего стола окно. В данном случае текущим окном является окно Command Window. Наиболее быстрый доступ к команде реализуется с помощью сочетания клавиш Ctrl+W.

Следующая группа команд позволяет импортировать данные (**Import Data...**) с диска в рабочую область системы MATLAB, а также сохранять данные, расположенные в рабочей области, на диске (**Save Workspace as ...**). Система MATLAB обеспечивает импорт и экспорт данных, представленных в следующих форматах: текстовом, двоичном и HDF (Hierarchical Data Format – объектно-ориентированный формат обмена данными). В текстовом формате значения данных хранятся в коде ASCII (American Standard Code Information Interchange). Система MATLAB позволяет импортировать данные в текстовом формате из файлов с расширением .txt (Text), .dat (Data), .csv (Comma-Separated Values – разделенные запятыми значения). Данные в двоичном формате могут содержать изображения, аудио- и видеинформацию: файлы с оцифрованной звуковой информацией (.wav – Waveform Audio), мультимедийные файлы (.avi – Audio-video Interleaved), файлы изображений (.bmp – Bitmap Picture, .cur – Cursor (изображения курсоров), .gif – Graphic Interchange Format, .ico – Icon (пиктограммы в Windows), .tif – Tagged Image Format и др.). Формат HDF является независимым от платформы форматом общего назначения, который используется для хранения научных данных. Формат HDF был разработан National Center for Supercomputing Application (NCSA – Национальный центр по приложениям для суперкомпьютеров). Формат HDF может содержать многомерные числовые массивы или текстовые данные. Импорт данных осуществляется либо с помощью встроенных функций, задаваемых в командной строке, либо с помощью специальных приложений Import Wizard или HDF Import Tool.

Следующие две команды – **Set Path ...** и **Preferences...** – открывают диалоговые окна, в которых можно настраивать список путей доступа к файлам и параметры работы системы MATLAB соответственно. Список путей доступа используется при поиске необходимого файла во время выполнения команды. С помощью диалогового окна **Set Path** можно создавать новые пути доступа, перемещать уже созданные по списку, а также удалять их из списка. Для увеличения быстродействия выполнения команд следует помешать в начало (наверх) списка имена тех папок, которые содержат наиболее часто используемые файлы. С помощью диалогового окна **Preferences** настраиваются параметры работы инструментальных средств системы MATLAB: размер и цвет шрифта, цвет фона, формат вывода числовой информации и т.д.

Команды **Print Setup ...**, **Print...** и **Print Selection...** служат для открытия диалоговых окон, в которых производится настройка страницы для печати, выбор принтера и передача на печать содержимого текущего инструментального средства, а также распечатка выделенной части соответственно.

Следующая часть меню содержит до четырех последних имен файлов, с которыми производилась работа в системе MATLAB.

Последняя команда – **Exit MATLAB** – осуществляет закрытие системы.

Упражнение 3. Импорт данных и сохранение переменных на диске.

- Импортировать данные из файла C:\MATLAB6p5\labs\ones.csv. Выбрать из меню File команду Import Data Появится диалоговое окно Import, где в качестве текущей папки необходимо выбрать C:\MATLAB6p5\labs, в фильтре выбора типа файла установить Spreadsheet (*.csv, *.xls, *.wk1) и выбрать файл ones.csv, в котором содержится матрица единиц 3×3. После нажатия на кнопку Open появится диалоговое окно Import Wizard, изображенное на рис. П-1.4. В верх-

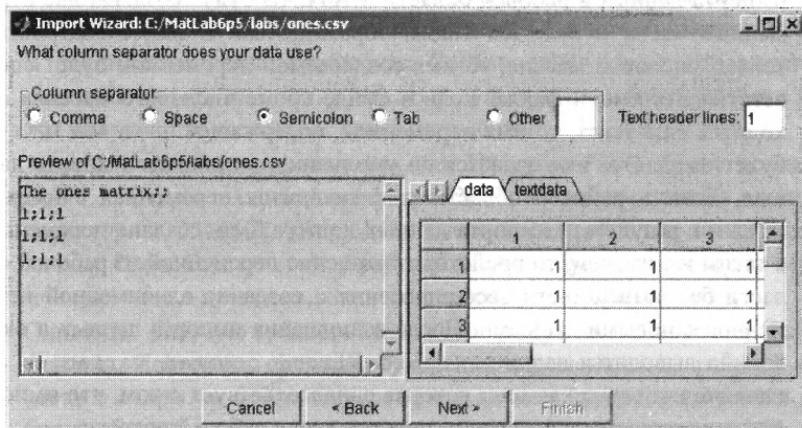


Рис. П-1.4. Окно Import Wizard

ней части окна представлены разделители между столбцами данных и поле Text header lines, в которое вводится количество строк заголовка данных, не входящих в числовой массив. Разделителями столбцов могут быть следующие символы: запятая (Comma), пробел (Space), точка с запятой (Semicolon), табуляция (Tab), а также любой другой символ, введенный в поле Other. В левой части окна отображается выбранный файл, а в правой части выводится результат его преобразования. Содержание файла будет преобразовано в две переменные – data и textdata, значения которых размещаются в одноименных вкладках. После нажатия на кнопку Next > содержимое диалогового окна Import Wizard изменится и появится возможность выбора переменных, которые будут импортированы в рабочую область MATLAB. В левой части окна выводятся имена переменных, их размер, объем в байтах и класс. Флажок напротив имени переменной свидетельствует о том, что переменная будет импортирована в рабочую область MATLAB. В правой части окна отображается содержание выделенной в левой части переменной. Для импорта только одной переменной data в рабочую область

необходимо убрать флажок с поля, располагающегося напротив имени переменной `textdata` и нажать на кнопку `Finish`. В результате в рабочей области будет создана одна переменная `data` размером 3×3 , которая занимает в памяти 72 байта. Просмотреть результаты импорта данных в рабочую область MATLAB можно в окне `Workspace`.

2. Импортировать единичную матрицу 3×3 с символами табуляции в качестве разделителей столбцов данных из текстового файла `C:\MATLAB6p5\labs\matrix.txt`. В результате выполнения операции импорта данных в рабочей области MATLAB будут созданы две переменные: `data` и `matrix`. Таким образом, если в файле содержится только числовой массив, то имя создаваемой переменной будет соответствовать имени файла. Если в файле кроме числового массива находится еще текст, то имя переменной, содержащей числовой массив, будет `data`. Это имя задается по умолчанию в процессе импорта данных. Если в рабочей области уже находится переменная с именем `data` и в результате импорта данных должна быть создана переменная с таким же именем, то происходит удаление переменной из рабочей области без возможности восстановления и создание одноименной переменной с новыми данными. После выполнения импорта данных в окне команд выводится надпись: `Import Wizard created variables in the current workspace.`, которая свидетельствует о том, что импорт данных завершился созданием переменных в рабочей области.
3. Сохранить рабочую область на диске в файле `data.mat`. Для сохранения переменных, расположенных в рабочей области, необходимо из меню `File` выбрать команду `Save Workspace as ...`. Появится диалоговое окно `Save to MAT-File`, где необходимо выбрать директорию, в которой будет сохранен файл, и задать имя файла. После нажатия на кнопку `Save` файл будет сохранен с расширением `.mat`. Файлы с расширением `.mat` являются двоичными файлами и используются системой MATLAB для хранения текстовых или числовых данных. Задайте команду `Save Workspace as...` и выберите директорию `C:\MATLAB6p5\labs`. Введите в поле `File Name:` имя файла `data` и нажмите на кнопку `Save`. В результате на диске в указанной директории будет создан файл `data.mat`, в котором размещаются переменные `data` и `matrix`.

Меню `Edit`, показанное на рис. П-1.3, в, содержит команды, которые позволяют: отменять (`Undo`) или возвращать (`Redo`) отмененное действие, обмениваться информацией с буфером обмена (`Cut` – Вырезать в буфер, `Copy` – Копировать в буфер, `Paste` – Вставить из буфера, `Paste Special...` – Специальная вставка), полностью выделять содержимое текущего окна (`Select All`), осуществлять поиск фрагмента текста в текущем окне инструментального

средства (*Find...*), удалять выделенный текст (*Delete*), а также очищать окно команд (*Clear Command Window*), окно истории команд (*Clear Command History*) и содержимое рабочей области (*Clear Workspace*). Некоторые команды в этом меню могут обозначаться серым цветом, который свидетельствует о том, что команда в данный момент недоступна.

Упражнение 4. Работа с командами меню *Edit*.

1. Удалить все переменные из рабочей области. С целью удаления содержимого рабочей области MATLAB сделать активным окно *Workspace*, выбрать из меню *Edit* команду *Select All*, а затем команду *Delete*. В появившемся окне сообщения *Confirm delete* для подтверждения удаления нажать на кнопку *Yes*. В результате из рабочей области будут удалены две созданные ранее переменные – *data* и *matrix*.
2. Загрузить данные из файла *data.mat*. Из меню *File* выбрать команду *Open...*. Откроется диалоговое окно *Open*, в котором сделать текущей директорию *C:\MATLAB6p5\labs*, выбрать файл *data.mat* и нажать на кнопку *Open*. В результате в рабочей области будут созданы две переменные – *data* и *matrix*, которые размещаются в файле *data.mat*.
3. Очистить рабочую область и заново загрузить данные из файла *data.mat*. Повторить операцию удаления переменных из рабочей области и открытия файла *data.mat*, но в данном случае для очистки рабочей области использовать команду *Clear Workspace* из меню *Edit*.

С помощью команд меню *Web*, показанных на рис. П-1.3, д, можно перейти на соответствующую страницу сайта фирмы–разработчика системы MATLAB.

В меню *Window*, представленном на рис. П-1.3, е, располагаются команды, которые позволяют закрывать все окна приложений, связанных с MATLAB, переключаться между окнами m-файлов, открытых в редакторе *Editor/Debugger*, окном библиотек блоков приложения *SIMULINK*, окнами моделей систем, разрабатываемых с помощью этого приложения, и т.д.

В меню *Help*, которое изображено на рис. П-1.3, ж, располагаются команды, обеспечивающие доступ к справочной информации о системе MATLAB и демонстрационным файлам по функциям этой системы.

Панель инструментов рабочего стола (*Desktop toolbar*), показанная на рис. П-1.5, предоставляет быстрый доступ к часто используемым командам. Для отображения подсказки о функции кнопки необходимо переместить курсор на кнопку и немного подождать.

С помощью панели инструментов можно создавать новые m-файлы, открывать уже существующие файлы, вырезать, копировать и вставлять информацию в/из буфера обмена, отменять и возвращать последнюю отменен-

нюю команду, запускать окно библиотек системы моделирования SIMULINK, переходить в окно просмотра помощи, просматривать и устанавливать текущую директорию системы MATLAB. При выполнении команды система MATLAB просматривает имена функций, входящих в команду, сначала в текущей директории, а затем в пути доступа (Path).

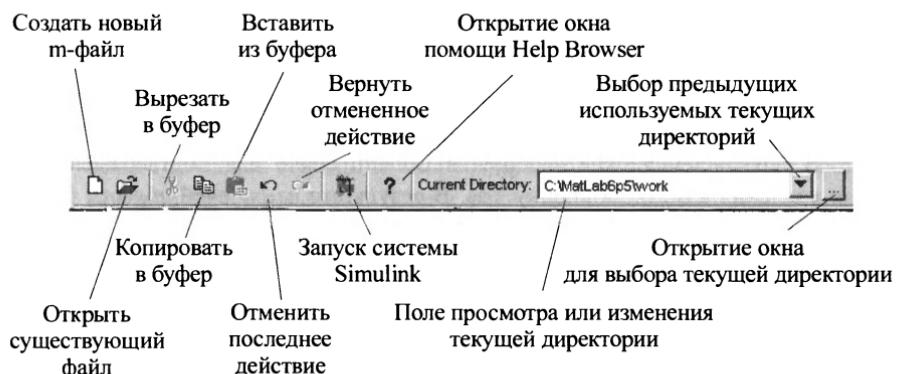


Рис. П-1.5. Панель инструментов рабочего стола

Упражнение 5. Работа с панелью инструментов рабочего стола.

- Установить текущую директорию ...\\MATLAB6p5\\labs. Открыть окно выбора текущей директории с помощью кнопки Browse For Folder . Выбрать в появившемся диалоговом окне директорию с именем labs, которая находится в директории MATLAB6p5, и нажмите ОК. Имя текущей директории появится в списке Current Directory:.
- Удалить переменные из рабочей области и загрузить данные из файла data.mat. Предварительно активизировав окно Workspace, выделить все переменные, расположенные в рабочей области, с помощью комбинации клавиш Ctrl+A. Нажатием на кнопку Delete удалить выделенные переменные из рабочей области. С помощью щелчка на кнопке Open File открыть диалоговое окно Open. При активизации диалогового окна Open в нем по умолчанию отображаются имена файлов, которые расположены в текущей директории. Из текущей директории выбрать файл data.mat. После открытия файла в рабочей области будут созданы две переменные – data и matrix.

Строка состояния (Status Bar), представленная на рис. П-1.6, предназначена для отображения текущего состояния системы (Initializing... – Инициализация..., Busy – Работа, Waiting for input – Ожидание ввода, Continue entering statement – Продолжение ввода команды и т.д.) и для показа подска-

зок по работе с системой во время перемещения курсора (Click here and drag to move the window – Щелкните и перетащите для перемещения окна).



Рис. П-1.6. Стока состояния

Контекстно-зависимые меню (Context Menus) предоставляют быстрый доступ к командам, используемым в данной ситуации. Контекстно-зависимые меню вызываются щелчком правой кнопки мыши в окнах инструментальных средств (Command Window, Command History, Workspace Browser и т.д.). Серым цветом обозначаются пункты меню, которые недоступны в данном случае.

Командное окно (Command Window)

Командное окно используется для ввода команд и вывода результатов их выполнения в текстовом виде. Работа с командным окном происходит в диалоговом режиме: пользователь вводит команду и передает ее ядру MATLAB, ядро обрабатывает полученную команду и возвращает результат. Все команды вводятся в командную строку после приглашения `>>`, которое свидетельствует о готовности ядра системы MATLAB к обработке команды.

Упражнение 6. Работа с командным окном

1. Ввести в командную строку `a = 0.2` (переменной `a` присваивается значение 0.2) и нажать Enter. Нажатие клавиши Enter определяет окончание ввода команды и ее передачу ядру системы MATLAB. После получения команды ядро сначала проверяет команду на синтаксические ошибки. Если нет синтаксических ошибок, то начинается обработка команды. В противном случае выводится сообщение об ошибке. После обработки команды ядро передаст командному окну результат ее выполнения и подготовится к приему новой команды `>>`. На рис. П-1.7 показан пример ввода синтаксически неправильной команды и сообщение, которое выводится в командном окне в ответ на эту команду. Сообщение состоит из трех строк. В первой строке после вопросительных знаков выводится команда. Во второй строке печатается вертикальная черта, которая указывает место первой встреченной в команде синтаксической ошибки. В третьей строке выводится поясняющая эту ошибку надпись. Система MATLAB указывает на то, что был пропущен оператор, запятая или точка с запятой (рис. П-1.7). В данном случае между десятичной точкой и цифрой был введен пробел, который привел к ошибочной команде. В ответ на правильно введенную команду в командное окно выводится результат этой команды, который состоит из

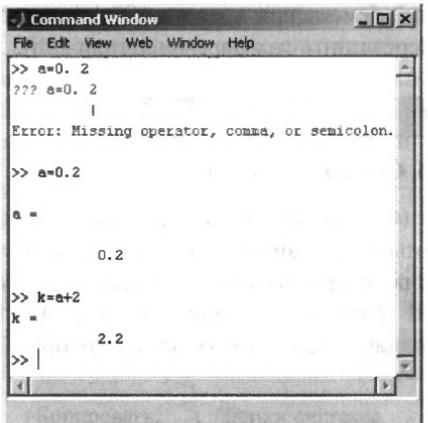


Рис. П-1.7. Командное окно

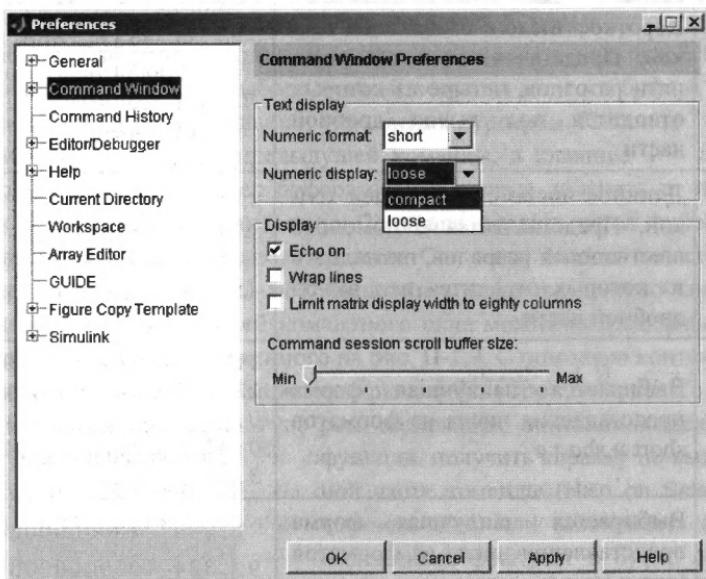
имени переменной, знака равенства и значения переменной. В командном окне возможны два режима отображения команд и результатов: широкий (*loose*) и компактный (*compact*). В первом случае между командой и результатом, а также внутри результата между именем переменной и данными выводятся пустые строки для лучшего восприятия информации. Во втором случае пустые строки не используются. Для изменения режима

работы командного окна необходимо выполнить следующие действия: из меню **File** выбрать команду **Preferences...**, в открывшемся диалоговом окне **Preferences** сделать активным инструментальное средство **Command Window** и в группе **Text display** изменить значение в поле **Numeric display** (рис. П-1.8).

2. Ввести новую команду $k=a+2$ и просмотреть результат. Результат выводится на двух строках. Переменная k равна 2.2. Система MATLAB запоминает в рабочей области значения всех переменных, которые вводятся в командной строке. Следовательно, их можно использовать в выражениях для создания новых или изменения уже созданных переменных. Выражения должны всегда располагаться справа от знака присваивания, иначе появится сообщение об ошибке. Имена переменных могут располагаться как справа, так и слева от знака присваивания.

По умолчанию в командное окно выводится результат в числовом формате **short**. В табл. П-1.1 представлены форматы вывода чисел. Изменение формата вывода числа не влечет за собой изменение самого числа. Числа, с которыми работает система MATLAB, занимают в памяти 8 байт. Мантисса числа занимает 55 бит, а в оставшихся 9 битах размещается порядок числа и его знак. Изменить формат вывода числа можно с помощью диалогового окна **Preferences**, где после активизации инструментального средства **Command Window** из раскрывающегося списка поля **Numeric format**, расположенного внутри группы **Text display**, следует выбрать необходимое значение (рис. П-1.8). Изменить формат вывода чисел можно не только с помощью диалогового окна, но и с помощью

задания команды в командном окне. Например, для смены текущего формата на long e необходимо в командной строке ввести команду format long e.



**Рис. П-1.8 Окно настроек инструментального средства
Command Window**

Таблица П-1.1

Форматы вывода числовых данных

Формат	Описание	Пример
short	Короткое число с фиксированной точкой. Четыре разряда отводится под вывод дробной части. При необходимости производится округление до четвертого знака	1.0200 10.0200 100.0201 (100.02005)
long	Длинное число с фиксированной точкой. Четырнадцать разрядов отводится под вывод дробной части. При необходимости производится округление	1.00100200300401 (1.001002003004005)

Формат	Описание	Пример
short e	Короткое число с плавающей точкой. Представляется с помощью пяти разрядов, четыре из которых отводится под вывод дробной части	1.2346e+000 (1.234567) 1.2346e+001 (12.34567)
long e	Длинное число с плавающей точкой. Представляется с помощью шестнадцати разрядов, пятнадцать из которых отводится под вывод дробной части	8.765432112345678e+007
short g	Выбирается наилучшая форма представления числа из форматов short и short e	1.2398e+005 (123979.4) 123.46 (123.456789) 0.34523 (0.3452331) 0.00034523 (0.0003452331) 3.4523e-005 (0.0000345233)
long g	Выбирается наилучшая форма представления числа из форматов long и long e	1.239794000000000e+005 99.33440000000000
hex	Число выводится в шестнадцатеричной форме	4010cccccccccccd (4.2)
bank	Формат представления денежных единиц – долларов и центов. Целая часть отводится под доллары, дробная часть, состоящая из двух разрядов, – под центы	69.96
+	Символьное обозначение числа: «+» – положительное число; «-» – отрицательное число; пробел – нулевое значение	+ (5) - (-2) (0)
rational	Число выводится в дробном виде	1/3

Упражнение 7. Ознакомление с форматами вывода чисел и вызовом ранее введенных команд.

- Сменить формат представления чисел на short e и просмотреть значение переменной a в командном окне. Для просмотра значения пере-

менной в командном окне необходимо в командную строку ввести имя соответствующей переменной и нажать на клавишу Enter.

- Последовательно изменения формат представления чисел, просмотреть значение переменной *a* в различных форматах. С целью ускорения вызова ранее введенной команды можно использовать кольцевой буфер хранения команд. Для выбора нужной команды из этого буфера необходимо использовать клавиши управления курсором. Клавиша \uparrow предназначена для выбора предыдущей команды, а клавиша \downarrow – последующей команды. Размер буфера обмена можно изменять от *min* до *max* с помощью бегунка поля *Command session scroll buffer size* в разделе инструментального средства *Command Window* диалогового окна *Preferences* (рис. П-1.8). Выполнить ранее введенную команду с помощью возможностей командного окна можно посредством контекстного меню, изображенного на рис. П-1.9. С помощью контекстно-зависимого меню также можно открыть файл (*Open Selection*), в котором

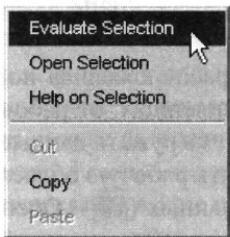


Рис. П-1.9. Контекстное меню окна команд

содержится описание выделенной функции, получить справку по выделенной части команды (*Help on Selection*), вырезать (*Cut*) или скопировать (*Copy*) выделенный фрагмент в буфер обмена и вставить (*Paste*) информацию из буфера обмена. Операции *Cut* и *Paste* действительны только для текущей командной строки. Для повторного задания команды или выполнения части команды необходимо выделить всю

команду или ее часть соответственно и щелчком правой кнопки мыши на выделенном фрагменте вызвать контекстное меню, в котором выбрать команду *Evaluate Selection*.

- Используя контекстное меню, просмотреть значение переменной *k* в командном окне.
- Установить формат вывода числовой информации, который принят по умолчанию.

Окно просмотра рабочей области (Workspace Browser)

Окно просмотра рабочей области, изображенное на рис. П-1.10, предназначено для быстрого просмотра атрибутов переменных, располагающихся в рабочей области. С помощью окна просмотра рабочей области можно увидеть имя переменной (*Name*), ее размер (*Size*), число байтов (*Bytes*), зани-

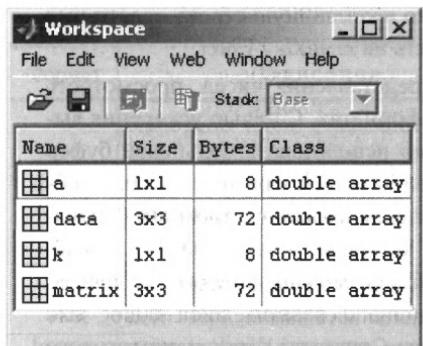


Рис. П-1.10 Окно просмотра рабочей области

честву байт и классу. Более быстрый способ сортировки переменных по атрибутам, реализуется щелчком левой кнопки мыши на имени соответствующего атрибута.

Палитра инструментов Workspace Browser, расположенная под строкой заголовка или под строкой меню в случае открепленного от рабочего стола окна, позволяет выполнять следующие операции: загружать данные из файла в рабочую область (– Load Data File), сохранять рабочую область в файле (– Save Workspace As), открывать Редактор данных (– Open) с целью просмотра или редактирования значений выделенной переменной, удалять выделенную переменную (– Delete), а также переключаться с помощью списка Stack между основной рабочей областью (Base) и рабочей областью функций во время их отладки.

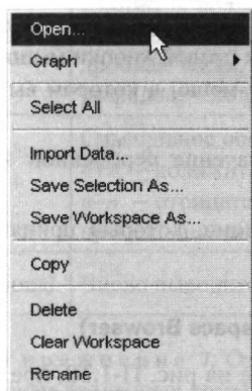


Рис. П-1.11. Контекстное меню Workspace

маемых переменной в памяти, и ее класс (Class). Для идентификации класса переменной слева от имени используется соответствующая иконка.

Окно Workspace Browser можно вывести на экран либо с помощью соответствующей команды меню View, либо с помощью задания команды workspace в командном окне. С помощью подменю Workspace View Options меню View можно изменять внешний вид окна Workspace (скрывать или показывать поля Size, Bytes и Class), а также сортировать переменные по имени, размерности, коли-

честву байт и классу. Более быстрый способ сортировки переменных по атрибутам, реализуется щелчком левой кнопки мыши на имени соответствующего атрибута.

Палитра инструментов Workspace Browser, расположенная под строкой заголовка или под строкой меню в случае открепленного от рабочего стола окна, позволяет выполнять следующие операции: загружать данные из файла в рабочую область (– Load Data File), сохранять рабочую область в файле (– Save Workspace As), открывать Редактор данных (– Open) с целью просмотра или редактирования значений выделенной переменной, удалять выделенную переменную (– Delete), а также переключаться с помощью списка Stack между основной рабочей областью (Base) и рабочей областью функций во время их отладки.

Дополнительные функции по работе с Workspace Browser можно получить при помощи контекстно-зависимого меню, изображенного на рис. П-1.11. Это меню содержит команды, выполняющие следующие действия: открытие Редактора данных (Open...), открытие дополнительных подменю для построения различных графиков (Graph ▶), выбор всех переменных в окне просмотра рабочей области (Select All), импорт данных в рабочую область из файла (Import Data ...), сохранение выделенных переменных в файле (Save Selection As...),

сохранение всей рабочей области в файле (Save Workspace As...), копирование выделенных переменных в буфер обмена (Copy), удаление выделенных переменных из рабочей области (Delete), очистку рабочей области (Clear Workspace) и переименование выделенной переменной (Rename). Целый ряд пунктов контекстно-зависимого меню обеспечивает более быстрый доступ к командам, расположенным в меню рабочего стола.

Упражнение 8. Работа с инструментальным средством Workspace Browser.

1. Используя контекстно-зависимое меню Workspace Browser, сохранить в файле C:\MATLAB6p5\labs\data_a_k.mat созданные ранее переменные a и k. Для этого необходимо сначала выделить переменные a и k. При выделении переменных, располагающихся в разных местах списка переменных, использовать клавишу Ctrl. Затем нужно вызвать контекстное меню и в нем выбрать команду Save Selection As... В открывшемся диалоговом окне Save to MAT-File: выбрать директорию C:\MATLAB6p5\labs, в поле File name ввести имя файла data_a_k.mat и нажать на кнопку Save.
2. Убрать из окна Workspace Browser поле Class. Из меню View выбрать раскрывающееся подменю Workspace View Options, в котором при помощи щелчка на имени команды снять флажок с команды Show Class. Наличие флажка свидетельствует о том, что соответствующее поле будет показано в окне просмотра рабочего стола.
3. Отсортировать имена файлов по алфавиту в обратном порядке. Нажать левой кнопкой мыши на имени поля Name. Повторное нажатие на этом имени изменяет направление сортировки на противоположное.
4. Изменить имя переменной k на b. Выделить переменную k, открыть контекстное меню и выбрать команду Rename. Окно просмотра переменных перейдет в режим редактирования имени переменной. Ввести новое имя и нажать на клавишу Enter.
5. Используя контекстное меню, очистить рабочую область. Из контекстного меню необходимо выбрать команду Clear Workspace.
6. С помощью панели инструментов окна Workspace загрузить данные из файла data_a_k.mat. Нажать на кнопку Load Data File. В появившемся диалоговом окне установить директорию C:\MATLAB6p5\labs, выбрать соответствующий файл и нажать на кнопку Open.
7. Используя контекстное меню, импортировать числовые данные из файлов matrix.txt и ones.csv.
8. Сделать видимым поле Class.

Просматривать содержимое рабочей области, загружать и удалять данные можно не только с помощью окна *Workspace Browser*, но также с помощью *Command Window*. Для просмотра имен переменных, находящихся в рабочей области, необходимо в командную строку ввести команду *who*. Результат задания команды *who* соответствует внешнему виду окна *Workspace Browser* при отключенных полях *Size*, *Bytes* и *Class*. Полную информацию о содержании рабочей области можно получить с помощью команды *whos*. В результате информация, выведенная в командное окно, будет соответствовать представленной в окне просмотра рабочей области при всех включенных полях, с тем исключением, что в командное окно будет выведено общее количество переменных и общий объем занимаемой ими памяти. Для удаления переменной из рабочей области необходимо ввести в командную строку команду *clear имя_переменной*. Очистка рабочей области осуществляется с помощью команды *clear* без параметров. Загрузка всех данных из файла реализуется командой *load имя_файла*, а выборочная загрузка – командой *load имя_файла имя_переменной*. Для сохранения рабочей области на диске необходимо ввести команду *save имя_файла*. Данные будут сохранены в файле с расширением *.mat*. Выборочное сохранение переменных из рабочей области обеспечивается командой *save имя_файла имя_переменной*.

Упражнение 9. Операции с рабочей областью посредством окна команд *Command Window*.

1. Просмотреть имена переменных, расположенных в рабочей области – *who*.
2. Сохранить переменную *k* в файле C:\MATLAB6p5\labs\data_k.mat – *save C:\MATLAB6p5\labs\data_k.mat k*.
3. Удалить переменную *k* из рабочей области – *clear k*.
4. Просмотреть в командном окне полную информацию о рабочей области – *whos*.
5. Очистить рабочую область – *clear*.
6. Загрузить в рабочую область из файла C:\MATLAB6p5\labs\data_a_k.mat переменную *a* – *load c:\matlab6p5\labs\data_a_k.mat a*.
7. Загрузить данные из файла C:\MATLAB6p5\labs\data_k.mat – *load data_k*.

Окно Редактора данных (Array Editor)

Редактор данных, изображенный на рис. П-1.12, предназначен для просмотра и редактирования значений переменных. Под редактированием переменных подразумевается не только изменение массива значений переменной, но также и изменение ее размера.

Редактор данных вызывается двойным щелчком на имени переменной в окне просмотра рабочей области или заданием в командном окне команды `openvar('имя_переменной')`. Окно Редактора данных состоит из панели инструментов и области просмотра значений переменных. В случае открепленного от рабочего стола окна редактирования данных в окне присутствует главное меню, которое совпадает с главным меню рабочего стола, и строка состояния (рис. П-1.12). В окне редактора данных можно отображать несколько переменных. Переключение между переменными реализуется щелчком на соответствующей вкладке, которая располагается в нижней части окна Array Editor. На рис. П-1.12 в окне редактирования данных находятся две переменные: `a` и `k`.

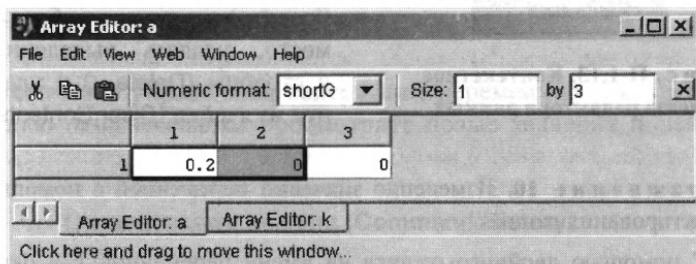


Рис. П-1.12. Окно редактора данных

Панель инструментов окна редактирования данных используется для перемещения (– Cut) и копирования (– Copy) в буфер обмена выделенных значений, вставки (– Paste) значений из буфера, изменения числового формата (Numeric format) просмотра значений переменной и изменения числа строк и столбцов переменной (Size: by). При перемещении выделенных значений в буфер обмена на их месте будут записаны нули. Количество и размерность числовых значений при вставке из буфера обмена должны совпадать с областью, выделенной для вставки. Изменение числового формата просмотра данных действует только в окне Редактора данных и не распространяется на изменение числового формата в окне команд. Описание форматов данных можно просмотреть в табл. П-1.1. Изменение размера переменной осуществляется при помощи изменения значений в текстовых полях Size. Левое поле соответствует количеству строк, правое – количеству столбцов. При увеличении размера переменной с помощью текстовых полей Size в область просмотра будут добавлены новые ячейки, содержащие нулевые значения. Новые столбцы и строки добавляются соответственно за столбцами и строками с максимальными номерами. При уменьшении размерности происходит

удаление строк и/или столбцов с наибольшими номерами. Щелчком на номере строки или столбца полностью выделяется соответствующая строка или столбец. Для выделения всех значений необходимо нажать на поле, которое находится на пересечении строки номеров столбцов и столбца номеров строк переменной, либо из меню **Edit** выбрать команду **Select All (Ctrl+A)**.

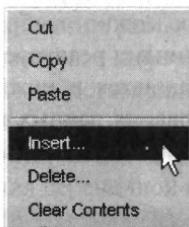


Рис. П-1.13. Контекстное меню редактора данных

Контекстное меню редактора данных, представленное на рис. П-1.13, содержит команды, которые позволяют: вырезать (**Cut**) и копировать (**Copy**) выделенные значения в буфер обмена, вставлять (**Paste**) значения из буфера обмена, а также вставлять (**Insert...**) строки и столбцы в указанное место, удалять выделенные строки и столбцы (**Delete...**) и удалять значение из ячейки (**Clear Contents**).

Упражнение 10. Изменение значений переменной с помощью окна редактирования данных.

- С помощью двойного щелчка на имени переменной *k* в окне *Workspace* отобразить содержание переменной *k* в окне *Array Editor*.
- С помощью задания соответствующей команды в командном окне отобразить в окне редактирования данных содержание переменной *a*.
- Используя панель инструментов, изменить размер переменной *a* на размер 1×3 .
- С помощью контекстного меню скопировать в буфер обмена значение из первой ячейки и вставить его в третью ячейку, используя палитру инструментов.
- Удалить вторую ячейку. Для удаления второй ячейки ее необходимо выделить, вывести контекстное меню и выбрать команду **Delete**.
- Изменить значение второй ячейки на 5.5. Переместить указатель мыши на вторую ячейку и дважды нажать левой кнопкой мыши. Ячейка перейдет в режим редактирования. Ввести значение 5.5 и нажать на клавишу *Enter*.
- Вставить две ячейки между первой и второй ячейкой. Выделить вторую ячейку и вызвать контекстное меню, в котором выбрать команду **Insert**. После выполнения команды будет создана одна ячейка, которая займет вторую позицию, отодвинув вторую ячейку на третью позицию. Повторить действия, но уже для третьей ячейки.

8. Используя контекстное меню, переместить в буфер значение четвертой ячейки.
9. С помощью палитры инструментов вставить значение из буфера обмена в ячейки с номерами 2, 3 и 4. Выделить три ячейки и нажать на кнопку Paste.
10. Скопировать весь массив значений переменной a в буфер обмена.
11. Перейти к просмотру значений переменной k.
12. Изменить размер переменной k на размер 2×4.
13. Во вторую строку значений переменной k вставить данные из буфера обмена.
14. Изменить формат отображения числовых значений на rational.
15. Закрыть окно редактирования.

После окончания упражнения содержание переменных a и k изменится. С помощью окна Workspace просмотрите новые значения полей переменных a и k.

Окно истории команд (Command History)

Окно истории команд, показанное на рис. П-1.14, служит для просмотра заданных ранее в командной строке Command Window команд. В окне истории команд можно также просмотреть дату и время начала сеанса работы с системой MATLAB. Сеанс работы с системой MATLAB начинается после ее загрузки в память и вывода на экран ее рабочего стола. Завершение сеанса работы сопровождается закрытием рабочего стола системы MATLAB.

С помощью контекстного меню окна истории команд, изображенного на рис. П-1.15, можно выполнять следующие действия: копировать (Copy) выделенные строки в буфер обмена, повторно выполнять команду или серию выделенных команд (Evaluate Selection), создавать новый m-файл (Create M-File) путем копирования выделенных строк в редактор m-файлов Editor/Debugger,

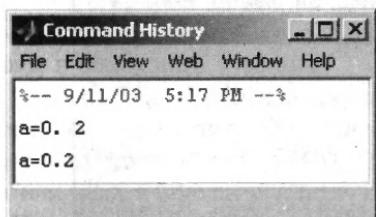


Рис . П-1.14. Окно истории команд

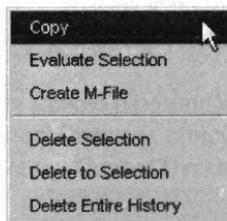


Рис . П-1.15. Контекстное меню окна истории команд

а также удалять выделенные строки (*Delete Selection*), удалять все строки из окна истории команд до выделенной строки (*Delete to Selection*) и полностью очищать окно истории команд (*Delete Entire History*).

Изменять параметры работы окна истории команд можно с помощью диалогового окна *Preferences*, представленного на рис. П-1.16, при активном инструментальном средстве *Command History*. Диалоговое окно вызывается из меню *File* командой *Preferences...* В окне присутствуют две группы опций: *Settings* (Настройки) и *Saving* (Сохранение). В первой группе опция включается установкой флажка напротив ее имени. Во второй группе опции переключаются при щелчке на соответствующем имени. В группе *Settings* можно включать или выключать следующие опции: *Save exit/quit command* – сохранять в истории команду *exit/quit* (при задании одной из этих команд система MATLAB завершает работу); *Save consecutive duplicate commands* – сохранять одинаковые команды, заданные в командном окне друг за другом; *Save commands entered at an input prompt* – сохранять команды, введенные в командную строку в ответ на команду *input* (например, `>> n=input('Введите номер варианта');`); *Allow Drag and Drop editing* – разрешить перенос команд из окна истории команд в командное окно (перенос возможен только в том случае, если окна расположены в пределах рабочего стола). В группе *Saving* можно переключаться между следующими опциями:

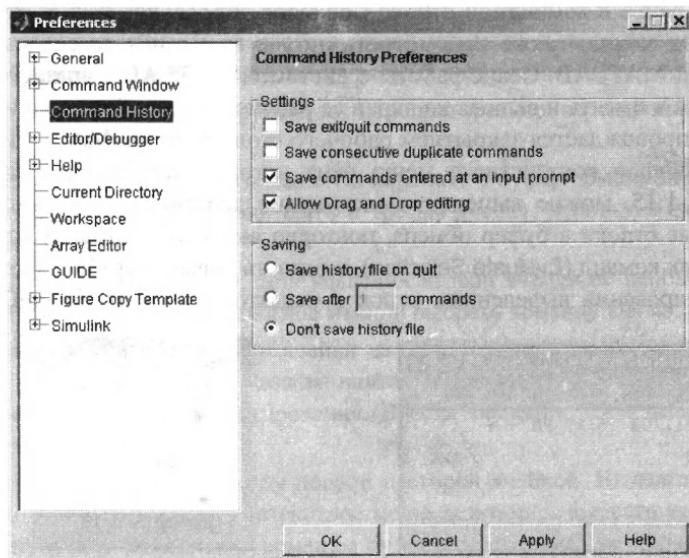


Рис. П-1.16. Диалоговое окно настроек работы
окна истории команд

Save history file on quit – сохранять файл истории при выходе (история команд сохраняется в файле `history.m`), **Save after n commands** – сохранять файл истории после ввода каждой n команды, **Don't save history file** – не сохранять файл истории команд (несмотря на включенную опцию, в течение сеанса работы в окне истории команд сохраняются все команды, которые будут удалены при выходе).

Упражнение 11. Работа с окном истории команд.

1. В диалоговом окне **Preferences** задать опции, как показано на рис. П-1.16.
2. Используя контекстное меню окна истории команд, повторно выполнить команду `a=0.2`.
3. Перенести из окна истории команд в командное окно команду `b=a+2`.
4. С помощью контекстного меню удалить из текущего сеанса работы все команды `a=0.2` и `b=a+2`.
5. Задать команду `n=input('Введите номер варианта: ')` и в ответ на приглашение ввести свой вариант. Просмотреть результаты работы в окне истории команд.
6. В диалоговом окне **Preferences** убрать флажок с опции **Save commands entered at an input prompt** и повторить пятое задание.
7. Очистить окно истории команд с помощью команды **Clear Command History** из меню **Edit**.

Окно просмотра текущей директории (**Current Directory Browser**)

Окно просмотра текущей директории, изображенное на рис. П-1.17, предоставляет большие возможности по работе с файлами и директориями. С помощью средств **Current Directory Browser** можно создавать, переименовывать и удалять файлы и директории, перемещаться по структуре директорий, просматривать содержание текущей директории и атрибутов файлов, а также открывать файлы для работы в системе MATLAB. Под атрибутами файлов подразумевается его тип (File Type), дата последнего изменения (Last Modified) и краткое описание файла (Description). Имеется дополнительная возможность просматривать в окне **Current Directory Browser** содержание mat-файлов и описание m-файлов.

Открыть окно просмотра текущей директории можно различными способами: щелчком левой кнопкой мыши на вкладке **Current Directory**, присутствующей на рабочем столе системы MATLAB, выбрав команду **Current Directory** из меню **View** или задав в **Command Window** команду `filebrowser`.

На панели инструментов окна **Current Directory Browser** размещаются элементы управления, которые позволяют переключаться с помощью выпадающего списка между недавно используемыми текущими директориями,

открывать окно выбора текущей директории (– Browse For Folder), перемещаться на один уровень наверх (– Go Up One Level) в иерархической структуре директорий операционной системы Windows, создавать директорию (– New Folder) внутри текущей директории и осуществлять поиск фрагментов текста внутри файлов (– Find in Files). Работа с первыми двумя элементами панели инструментов окна Current Directory Browser аналогична работе с соответствующими элементами панели инструментов рабочего стола системы MATLAB. Следующие два элемента являются стандартными для операционной системы Windows. Перемещаться на один уровень наверх можно при нажатии на клавишу Back Space (←) при активном окне Current Directory Browser.

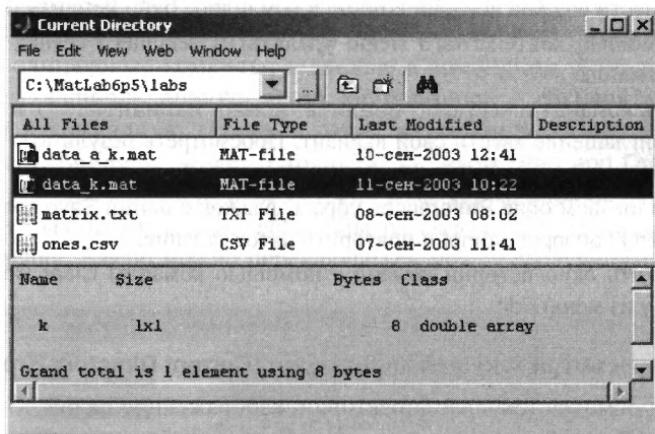


Рис. П-1.17. Окно просмотра текущей директории

Используя диалоговое окно Preferences с активным инструментальным средством Current Directory Browser, представленное на рис. П-1.18, можно выполнять следующие настройки окна просмотра текущей директории: установить количество запоминаемых в списке последних текущих директорий, очистить список директорий (кнопка Clear History), а также отобразить или скрыть поле типа файла (Show file types), поле даты последней модификации файла (Show last modified date), описания m-файлов (Show M-file descriptions) и комментарии в m-файлах и содержание mat-файлов (Show M-file comments and MAT-file contents). При включении обеих или одной из двух последних опций в нижней части окна Current Directory Browser появляется область с серым фоном, куда будут выводиться описания и комментарии выделенных m-файлов и содержания mat-файлов соответственно.

Наиболее эффективная работа с окном Current Directory Browser осуществляется с помощью контекстно-зависимого меню этого окна, показанного

на рис. П-1.19. Контекстно-зависимое меню содержит команды, которые позволяют: открывать файл в соответствующем Редакторе (Open); запускать выделенный m-файл (Run); открывать окно просмотра справочной информации (Help Browser); открывать содержание файла в текстовом виде с помощью установленного текстового редактора или в редакторе Editor/Debugger (Open as Text); импортировать данные из файла (Import Data ...); создавать новый (New ...) m-файл (M-File), модель (Model) и директорию. (Folder); переименовывать (Rename) файл и директорию; удалять (Delete) выделенные

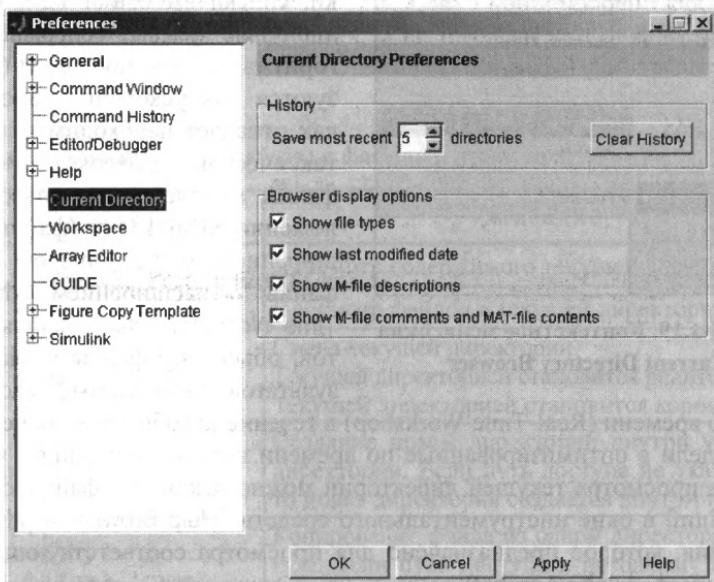


Рис . П-1.18. Диалоговое окно настроек Current Directory Browser

файлы и директории; вырезать (Cut) и копировать (Copy) в буфер обмена выделенные файлы и директории; вставлять из буфера обмена (Paste); изменять параметры фильтра отображения файлов и директорий в окне Current Directory Browser (File Filter ...); добавлять к путям доступа к файлам (Add to Path ...) текущую директорию (Current Directory), выбранные директории (Selected Folders) и выбранные директории и поддиректории (Selected Folders and Subfolders); обновлять содержимое текущей директории (Refresh).

В окне просмотра текущей директории можно просматривать как все файлы, расположенные в текущей директории, так и только выбранные файлы системы MATLAB. К файлам системы MATLAB относятся следующие файлы: M-files – текстовые файлы, содержащие описание алгоритмов работы программ на языке MATLAB; MAT-files – двоичные файлы, предназначенные для

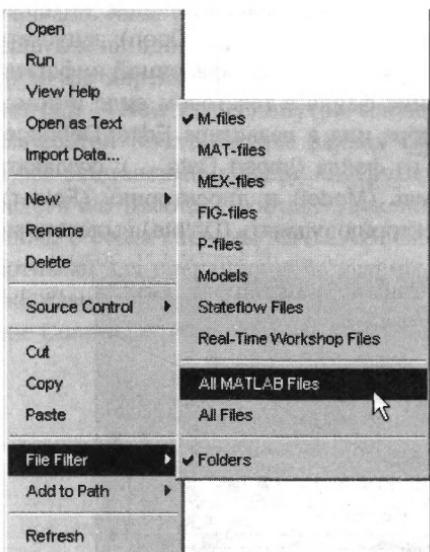


Рис. П-1.19. Контекстное меню окна
Current Directory Browser

реального времени (Real-Time Workshop) в течение преобразования исходного файла модели в оптимизированные по времени выполнения файлы. С помощью окна просмотра текущей директории можно открывать файлы с расширениями html в окне инструментального средства Help Browser и pdf в окне приложения, которое предназначено для просмотра соответствующих файлов (например, Adobe Acrobat).

Упражнение 12. Работа с окном просмотра текущей директории.

1. Просмотреть содержимое директории C:\MATLAB6p5\labs.
2. Уберать из окна Current Directory поля Last Modified и Description.
3. Используя контекстное меню, включить опцию просмотра всех файлов системы MATLAB.
4. Включить просмотр содержания mat-файлов.
5. Просмотреть содержание файлов data_a_k.mat и data_k.mat.
6. Создать внутри директории C:\MATLAB6p5\labs новую директорию gr07_10n, где n – номер группы.
7. Используя буфер обмена, скопировать файлы data_k.mat и data_a_k.mat из директории C:\MATLAB6p5\labs в недавно созданную директорию.

хранения данных; MEX-files – файлы, которые содержат откомпилированные алгоритмы работы программы, реализованных на языках С или FORTRAN; FIG-files – файлы графических окон, которые содержат графические результаты работы и/или управляющие элементы (кнопки, списки выбора и т.д.); P-files – двоичные файлы, содержащие алгоритмы работы программ (используются для ускорения работы, так как отпадает необходимость в синтаксической проверке); Models – файлы моделей, созданных в приложении SIMULINK (файлы с расширением mdl); Stateflow Files – файлы с расширением cdr; Real-Time Workshop Files – файлы проектов, объектные файлы и файлы результатов, создаваемые мастерской

8. Удалить файлы data_a_k.mat и data_k.mat из следующей директории C:\MATLAB6p5\labs.
9. Включить просмотр всех типов файлов.
10. Включить просмотр полей Last Modified и Description.
11. Выключить просмотр содержания mat-файлов.
12. Установить в качестве текущей директории C:\MATLAB6p5\work.

Функции по работе с директориями и файлами доступны не только из окна Current Directory Browser, но и из окна Command Window. Список команд для работы с файлами и директориями из командного окна приведен в табл. П-1.2.

Таблица П-1.2

Команды для работы с файлами в командном окне

Команда	Описание
dir	Просмотр содержимого текущей директории
what	Просмотр файлов в текущей директории
cd путь_доступа cd .. cd \	Смена текущей директории Текущей директорией становится родительская Текущей директорией становится корневая
mkdir('путь_доступа', 'имя_директории') или mkdir('имя_директории')	Создание новой директории внутри указанной директории. Если путь доступа не указывается, то новая директория создается внутри текущей
copyfile('путь_доступа\имя_файла', 'путь_доступа') или copyfile('путь_доступа\имя_файла', 'новое_имя_файла')	Копирование файла из одной директории в другую. Если путь доступа к исходному файлу не указывается, то он должен находиться в текущей директории. При отсутствии пути доступа ко второму файлу создается копия исходного файла с указанным именем в той же самой директории
movefile('путь_доступа\имя_файла', 'путь_доступа') или movefile('путь_доступа\имя_файла', 'новое_имя_файла')	Перемещение файла из одной директории в другую. Если путь доступа к исходному файлу не указывается, то он должен находиться в текущей директории. При отсутствии пути доступа ко второму файлу происходит смена имени файла
delete путь_доступа\имя_файла	Удаление файла из указанной директории. При отсутствии пути доступа файл должен размещаться в текущей директории
rmdir('путь_доступа\имя_директории')	Удаление указанной директории

Команды копирования и перемещения файлов могут применяться также и для копирования и перемещения директорий соответственно.

Упражнение 13. Файловые операции в окне команд.

1. Сменить текущую директорию на C:\MATLAB6p5\labs.
2. Просмотреть имена директорий и файлов, расположенных внутри текущей директории.
3. Просмотреть имена файлов, связанных с системой MATLAB.
4. Создать в директории C:\MATLAB6p5\labs\gr07_10n (n – номер группы) директорию lab01.
5. Скопировать файл data_a_k.mat из директории C:\MATLAB6p5\labs\gr07_10n в директорию C:\MATLAB6p5\labs\gr07_10n\lab01.
6. Переместить файл data_k.mat из директории C:\MATLAB6p5\labs\gr07_10n в директорию, указанную в предыдущем пункте.
7. Изменить текущую директорию на C:\MATLAB6p5\labs\gr07_10n\lab01.
8. Просмотреть текущую директорию.
9. Изменить имя файла data_a_k.mat на data_ak.mat. При необходимости выполнить команду Refresh из контекстного меню окна Current Directory для обновления содержимого этого окна.
10. Сделать текущей директорией C:\MATLAB6p5\work.

Окно просмотра справочной информации (Help Browser)

Окно просмотра справочной информации, изображенное на рис. П-1.20, служит для просмотра справочной информации в формате html (hypertext markup language – язык разметки гипертекста) и демонстрационных m-файлов по системе MATLAB и по продуктам, выпускаемым фирмой *MathWorks, Inc.* Окно Help Browser открывается после щелчка на кнопке ?, расположенной на панели инструментов рабочего стола, выбором команды Help из меню View или после ввода в командное окно команды helpbrowser. Okno Help Browser также можно открыть с помощью команд меню Help.

Окно Help Browser состоит из двух областей: области навигации (Help Navigator), расположенной с левой стороны, и области просмотра, расположенной с правой стороны. Область навигации включает управляющие элементы для установки фильтра поиска (Product Filter) и пять вкладок Contents (Содержание), Index (Индексный указатель), Search (Поиск), Demo (Демонстрационные файлы) и Favorites (Первоочередные справочные документы), которые реализуют различные режимы работы со справочной системой.

При активизации вкладки Contents осуществляется вывод в виде древовидной структуры списка всех установленных или выбранных с помощью

фильтра Product Filter приложений, а также связанных с ними справочных документов и демонстрационных файлов. Работа с областью навигации при активной вкладке Contents аналогична работе с областью навигации в стандартном приложении Explorer операционной системы Windows. Объекты древовидной структуры различаются с помощью пиктограмм. При активизации объекта в области навигации изменяется содержание области просмотра, в которую выводится соответствующая справочная информация.

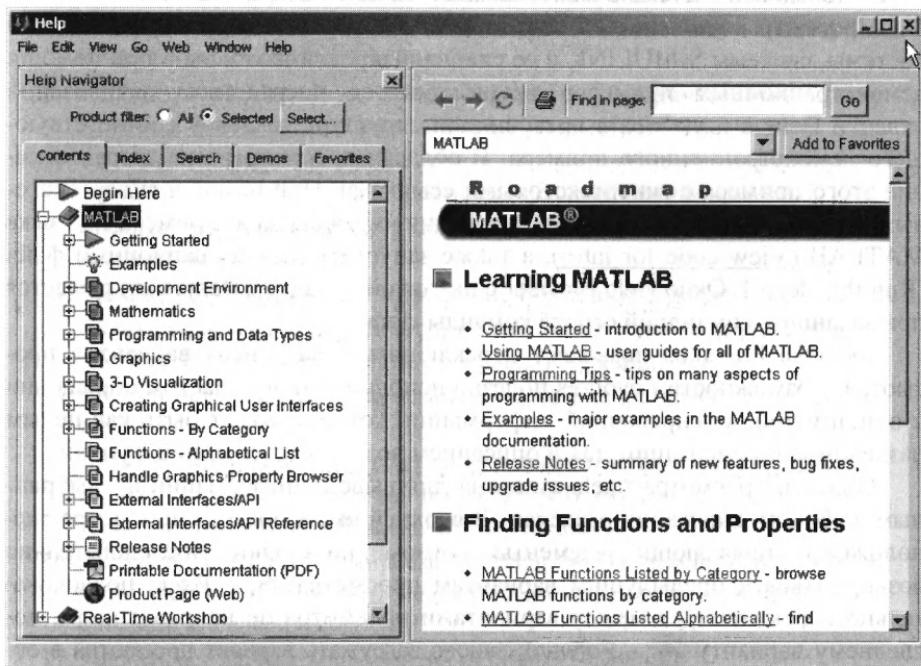


Рис. П-1.20. Окно Help Browser

Активизация вкладки Index переводит область навигации в новый вид представления информации. При активной вкладке Index в области навигации в алфавитном порядке располагаются ключевые слова документации по системе MATLAB. При этом область навигации содержит ключевые слова, начинающиеся на одну и ту же букву. Для смены начальной буквы необходимо ввести в поле Search index for: соответствующую букву или имя искомого слова. Ввод текста в указанное поле сопровождается обновлением содержимого области навигации. Следовательно, возможности индексного указателя Index можно использовать только в тех случаях, когда точно известно имя искомого ключевого слова или его начальные буквы.

Поиск фрагментов текста внутри документации осуществляется с помощью вкладки **Search**. При активизации этой вкладки становятся доступными два поля: **Search type** и **Search for**. Первое поле позволяет управлять типом поиска. Возможны четыре варианта поиска: **Full text** (весь текст), **Document Titles** (заголовки документов), **Function Name** (имена функций) и **Online Knowledge Base** (база знаний сайта фирмы). Второе поле предназначено для ввода искомого фрагмента текста. Кнопка **Tip** служит для получения справки по работе с поисковой системой. Поиск начинается после щелчка на кнопке **Go**.

Просмотр возможностей системы MATLAB, пакетов расширения этой системы, системы SIMULINK и ее специализированных блоков при помощи демонстрационных примеров осуществляется посредством активизации вкладки **Demos** и выбора в древовидной структуре разделов соответствующего демонстрационного примера. В области просмотра выводится описание этого примера с гипертекстовыми ссылками. При помощи гипертекстовых ссылок можно просмотреть внутреннюю реализацию примера на языке MATLAB ([View code for intro](#)), а также запустить демонстрационный файл ([Run this demo](#)). Окно **Help Browser** с активной вкладкой **Demos** открывается при задании в командной строке команды **demo**.

Вкладка **Favorites** позволяет переключиться на список вариантов просмотра. Возможности **Favorites** полезно использовать при частом обращении к одной и той же справочной информации, которая может быть как целым разделом в документации, так и описанием возможностей одной функции.

Область просмотра предназначена для вывода информации по выбранным в области навигации разделам. В верхней части области просмотра расположены управляющие элементы, которые позволяют: последовательно возвращаться к предыдущим вариантам просмотра – **Back**; последовательно переключаться между вариантами просмотра по направлению к последнему варианту – **Forward**; заново загружать вариант просмотра в область просмотра – **Reload**; распечатывать информацию, выведенную в область просмотра, – **Print Page**; осуществлять поиск введенного в поле **Find in page**: фрагмента текста в области просмотра; переключаться в произвольном порядке между последними вариантами просмотра в рабочей области с помощью списка и добавлять текущий вариант просмотра в список вариантов просмотра **Add Favorites**.

Упражнение 14. Работа с окном просмотра справочной информации.

1. Открыть окно **Help Browser** и сделать активной вкладку **Contents**.
2. Установить фильтр поиска только в MATLAB. (В области навигации активизировать управляющий элемент **Selected**. Нажать на кнопку **Select...** В появившемся диалоговом окне установить «» только напротив имени MATLAB и закрыть окно нажатием на кнопку **OK**).

3. Последовательно перемещаясь по древовидной структуре разделов (MATLAB → Development Environment → Starting and Quitting MATLAB → Starting MATLAB), сделать активным раздел (Starting MATLAB on Windows Platforms).
4. Просмотреть справочную информацию по разделу в области просмотра.
5. Перейти в раздел MATLAB → Development Environment → Using Desktop → What the Desktop Is.
6. Добавить этот вариант просмотра к списку Favorites.
7. Перейти в раздел MATLAB → Development Environment → Using the Desktop → Common Desktop Features.
8. Используя размещенную в окне просмотра гипертекстовую ссылку Keyboard Shortcuts, отобразить в окне просмотра справочную информацию по сочетаниям клавиш, применяемым в системе MATLAB.
9. Добавить текущий вариант просмотра к списку Favorites.
10. Сделать текущей вкладку Index, в поле Search index for: ввести or и нажать на кнопку Go.
11. Просмотреть описание по запрошенной логической операции.
12. Активизировать вкладку Search; в поле Search Type задать тип поиска Full Text; в поле Search ввести or и нажать на кнопку Go.
13. Просмотреть в области просмотра несколько разделов, выведенных в область навигации.
14. Сменить тип поиска на Function Name и просмотреть результаты поиска.
15. Перейти на вкладку Favorites и просмотреть последовательно все разделы, которые были добавлены в этот список.
16. Активизировать вариант просмотра What the Desktop Is; в поле Find in page: ввести desktop и, последовательно нажимая на кнопку Go, просмотреть все совпадения.
17. Удалить все варианты просмотра из списка Favorites.
18. Закрыть окно Help Browser.

Окно запуска приложений (Launch Pad) и кнопка Start

Окно запуска приложений и кнопка Start, представленные на рис. П-1.21 и рис. П-1.22 соответственно, обеспечивают доступ ко всем установленным приложениям и пакетам расширения системы MATLAB, системе моделирования SIMULINK, библиотекам блоков расширения системы SIMULINK (Blocksets), инструментальным средствам рабочего стола (DeskTop Tools), страницам сайта фирмы–разработчика системы MATLAB (Web), установкам системы MATLAB (Preferences...), системе помощи (Help) и демонстрационным примерам (Demos). Работа с окном запуска приложений аналогична работе с панелью обозревателя стандартного файлового менеджера Проводник операционной системы Windows. Работа с кнопкой

операционной системы Windows. Работа с кнопкой Start аналогична работе с кнопкой Пуск (Start) в операционной системе Windows.

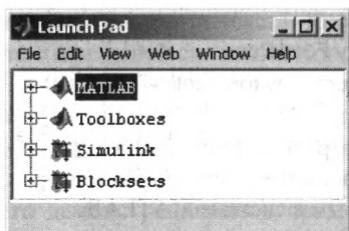


Рис. П-1.21. Окно Launch Pad

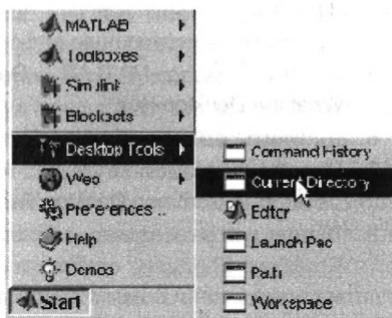


Рис. П-1.22. Главное меню системы MATLAB

Упражнение 15. Просмотр демонстрационных файлов.

1. Используя кнопку Start, открыть окно Help Browser на вкладке Demos.
2. Просмотреть демонстрационные файлы из раздела Desktop Environment.
3. Закрыть окно Help Browser.

Упражнение 16. Работа с системой поиска.

Самостоятельно освойте принципы работы с утилитой поиска Find, диалоговое окно которой представлено на рис. П-1.23.

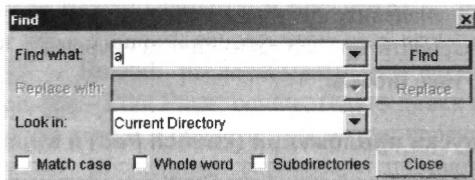


Рис. П-1.23. Утилита поиска Find

Практикум 3. КОМАНДНОЕ ОКНО

Цель: Закрепление навыков работы с командным окном и знакомство с принципами работы со скалярными величинами, векторами и матрицами в MATLAB.

Последовательность выполнения практикума

Во время выполнения практикума необходимо выполнить программы lab3_1, lab3_2 и lab3_3. Программы могут работать в двух режимах: обучения и тестирования. Сначала следует пройти все программы в режиме обучения. Перед загрузкой первой программы необходимо выключить сохранение команд, введенных в командное окно, в файле истории команд (File → Preferences → Command History → Don't save history file (Saving)). После завершения режима обучения следует перезапустить систему MATLAB и пройти все программы в режиме тестирования. Результаты тестирования необходимо сообщить преподавателю. Для более эффективной работы с командным окном можно использовать сочетания клавиш, которые представлены в табл. П-3.1.

Таблица П-3.1

Сочетания клавиш при работе с командным окном

Сочетание клавиш (Emacs)	Описание
↑ (Ctrl+P)	Вызов предыдущей команды из буфера команд
↓ (Ctrl+N)	Вызов следующей команды из буфера команд
← (Ctrl+B)	Перемещение курсора на один символ влево
→ (Ctrl+F)	Перемещение курсора на один символ вправо
Ctrl+←	Перемещение курсора на одно слово влево
Ctrl+→	Перемещение курсора на одно слово вправо
Home (Ctrl+A)	Перемещение курсора к началу текущей строки
End (Ctrl+E)	Перемещение курсора к концу текущей строки
Ctrl+Home	Перемещение курсора на первую строку
Ctrl+End	Перемещение курсора на последнюю строку
Esc (Ctrl+U)	Очистка текущей командной строки
Delete (Ctrl+D)	Удаление символа, на который указывает курсор
Backspace (Ctrl+H)	Удаление символа слева от курсора
Shift+Home	Выделение фрагмента строки от текущего положения курсора до начала строки
Shift+End	Выделение фрагмента строки от текущего положения курсора до конца строки

Перед запуском первой программы необходимо письменно ответить на два вопроса, полученных от преподавателя.

Контрольные вопросы

1. Что такое компьютерная математика?
2. Какие существуют классы систем компьютерной математики?
3. Перечислите основные системы компьютерной математики для каждого класса.
4. Какова структура универсальных систем компьютерной математики?
5. Состав системы MATLAB.
6. История создания системы MATLAB.
7. Рабочий стол системы MATLAB, его инструментальные средства и их назначение.
8. Импорт и экспорт данных в системе MATLAB: последовательность действий и форматы файлов.
9. Назначение командного окна.
10. Принципы работы с окном просмотра рабочей области.
11. Функциональные возможности окна истории команд.
12. Приемы работы с окном просмотра текущей директории.
13. Особенности работы с редактором данных.
14. Управляющие элементы окна просмотра справочной информации.
15. Назначение окна запуска приложений и главного меню системы MATLAB.
16. Форматы вывода числовых данных в командное окно и окно редактора данных.

Примечания

Так как файлы программ даются в открытом виде, то преподаватели могут вносить изменения, которые учитывают текущий уровень студентов, а также время, отводимое на проведение практикума.

Предложенные программы не являются универсальными, а соответствуют определенному классу заданий. Создание высококлассной обучающей программы, работающей в окружении системы MATLAB, является одной из тем курсовой работы (проекта). Основное внимание при реализации обучающих программ следует обращать на написание анализатора команд. Реализованный в программе анализатор не учитывает всех возможностей (например, вызов функции, размещенной в доступном системе MATLAB файле) и, следовательно, является основным сдерживающим фактором для создания универсальной обучающей программы.

Во время знакомства с элементами программирования в системе MATLAB можно обращаться к файлу программы и смотреть на примерах, как реализованы интересующие конструкции.

Практикум 4. КЛАССЫ ДАННЫХ В СИСТЕМЕ MATLAB

Цель: Получение практических навыков работы с классами данных в системе MATLAB.

Порядок выполнения практикума

Практикум состоит из пяти заданий. В первом задании необходимо выполнить ряд действий над данными арифметического класса, во втором – логического класса, в третьем – символьного класса, в четвертом – класса массив структуры и в пятом – класса массив ячеек. По итогам выполнения практикума нужно оформить отчет.

Задание 1. Согласно варианту (табл. П-4.1), задайте действительные переменные x , y и z и вычислите вещественные функции a и b в командном окне.

Порядок выполнения задания:

Заданные функции: $a = \ln\left(y^{\sqrt{|x|}}\right)\left(x - \frac{y}{2}\right)$, $b = \operatorname{tg}^2(\operatorname{arctg} z)$.

Значения переменных: $x = -15.246$; $y = 4.642$; $z = 0.201$.

1. Ввести переменные в командной строке:

```
>> x=-15.246; y=4.642; z=0.201;
```

2. Просмотреть значения переменных с помощью Array Editor.

3. Вычислить функции a и b :

```
>> a=log(y^(sqrt(abs(x))))*(x-y/2), b=tan(atan(z))^2  
a =  
-105.3  
b =  
0.040401
```

4. Удалить переменные x , y , z , a и b из рабочей области.

Таблица П-4.1

Индивидуальные варианты к заданию 1

№	Функции	Значения переменных
1	$a = 2^{-x} \sqrt{x + \sqrt[4]{ y }}$, $b = \sqrt{e^{\sin x}}$	$x = 3.381$; $y = 1.625$; $z = 0.512$.
2	$a = y^{\sqrt{ x }} + \operatorname{ch}^3(y-3)$, $b = \frac{y\left(\operatorname{arctg} z - \frac{\pi}{6}\right)}{ x + \frac{1}{y^2 + 1}}$	$x = -6.251$; $y = 0.827$; $z = 25.001$.

№	Функции	Значения переменных
3	$a = 2^{(y^x)} + (3^x)^y, b = \frac{ x+y \left(1 + \frac{\sin z}{x+y} \right)}{e^{ x+y } + \frac{x}{2}}$	$x = 3.251; y = 0.325; z = 5.466.$
4	$a = \frac{\sqrt{ x-1 } + \sqrt{ y }}{1 + \frac{x^2}{2} + \frac{y^2}{4}}, b = x(\arctg z + e^{-(x+3)})$	$x = -0.622; y = -3.379; z = 0.546.$
5	$a = \sqrt[4]{y + \sqrt{x-1}}, b = x-y (\sin^2 z + \operatorname{tg} z)$	$x = 17.622; y = 10.365; z = 0.808.$
6	$a = \frac{y^{(x+1)}}{\sqrt[3]{ y-2 } + 3}, b = (x+1)^{-\frac{1}{\sin z}}$	$x = 1.625; y = 15.400; z = 0.808.$
7	$a = \frac{x^{y+1} + e^{y-1}}{1+x y-\operatorname{tg} z }, b = 1 + y-x + \frac{ y-x ^3}{3}$	$x = 2.444; y = 0.869; z = -0.166.$
8	$a = 1 + x + \frac{x}{2!} + \frac{x}{3!} + \frac{x}{4!}, b = x(\sin \arctg z + \cos^2 y)$	$x = 0.355; y = 0.025; z = 32.005.$
9	$a = (1+y) \frac{x + \frac{y}{x+4}}{y^{x-2} + \frac{1}{x^2+4}}, b = \frac{1+\operatorname{ch}(y-2)}{\frac{x}{2} + \sin^2 z}$	$x = 3.258; y = 4.005; z = -0.666.$
10	$a = y + \frac{x}{y + \frac{x^2}{y + \frac{x^3}{y}}}, b = \left(1 + \operatorname{tg} \frac{27}{21} \right)^{\sqrt{ y +6}}$	$x = 0.100; y = -8.750; z = 0.765.$
11	$a = \lg \left(\sqrt{e^{x-y}} + x^{ y } + z \right), b = x - \frac{x^3}{3} - \frac{x^5}{5}$	$x = 1.542; y = -3.201; z = 80.005.$
12	$a = \frac{2 \cos(x - \pi/6)}{1/2 + \sin^2 y}, b = 1 + \frac{y^2}{3.8 + z^2/5}$	$x = 1.426; y = 1.220; z = 3.527.$
13	$a = \frac{1 + \operatorname{sh}(x+y)}{\left x - \frac{2y}{1+x^2y^2} \right }, b = e^{ x-y } (\operatorname{tg}^2 z + 1)^x$	$x = -4.500; y = 0.750; z = 0.845.$

Окончание табл. П-4.1

№	Функции	Значения переменных
14	$a = \frac{1 + \operatorname{sh}(x+y)}{\left x - \frac{2y}{1+x^2y^2} \right }, b = \cos^2(\operatorname{arctg} l/z)$	$x = 3.741;$ $y = -0.825; z = 0.160.$
15	$a = \cos x + \cos y ^{1+2\sin^2 y}, b = 1+z+\frac{z^2}{2}+\frac{z^3}{3}+\frac{z^4}{4}$	$x = 0.400;$ $y = -0.866; z = -0.475.$
16	$a = \ln\left(y^{\sqrt{ x }}\right)\left(x - \frac{y}{2}\right), b = \operatorname{tg}^2(\operatorname{arcsin} z)$	$x = -15.246;$ $y = 4.642; z = 0.201.$
17	$a = \sqrt[3]{10\left(\sqrt[3]{x} + x^{y+2}\right)}, b = (\operatorname{arcsin} z)^2 + x+y $	$x = 16.550;$ $y = -2.776; z = 0.281.$
18	$a = e^{ x-y } + x-y ^{x+y}, b = \frac{(\operatorname{arctg} z + \operatorname{arctg} x)}{y}$	$x = -2.235;$ $y = -0.825; z = 15.299.$
19	$a = x ^{y/x} - \sqrt{y x }, b = (y-x) \frac{y - \frac{z}{(y-x)}}{1 + (y-x)^2}$	$x = -1.725;$ $y = 18.252; z = -3.988.$
20	$a = \frac{x + \frac{y}{(5+\sqrt{x})}}{ y-z + \sqrt{x}}, b = e^{z-1} + \arcsin\left(\frac{y}{x}\right)$	$x = 47.885;$ $y = -8.782; z = 3.238.$
21	$a = y^x + \sqrt{ x + y }, b = z + \frac{z^2}{x + \frac{y}{(x+z)}}$	$x = -0.856;$ $y = 1.258; z = -0.456.$
22	$a = \sqrt[3]{x + \sqrt[4]{ y }}, b = \sqrt{ y } e^{-(y+z/2)}$	$x = 37.157;$ $y = -12.575; z = 1.945.$
23	$a = \frac{1}{2} \left(x^{\frac{ y-x }{2}} - \sqrt[3]{\frac{x-1}{ y +1}} \right), b = \lg(z^{1/3} + y^{1/2} + 2)$	$x = 3.925;$ $y = 2.582; z = 9.238.$
24	$a = (2+y^2) \frac{x + \frac{y}{7.89}}{y^2 + \frac{1}{(1+y^2)}}, b = \left(\sin^2(\operatorname{arctg} z) + \frac{x}{y} \right)$	$x = 2.591;$ $y = -2.572; z = 0.871.$
25	$a = y^x + (x + y)^{7/9}, b = \cos\left(2x \arcsin \frac{z}{y}\right)$	$x = 0.991;$ $y = 1.007; z = 0.675.$

Задание 2. Построить логическую область в графическом окне. Варианты заданий изображены на рис. П-4.2.

Порядок выполнения задания:

Рассматривается логическая область из примера 2.2.

- Сформировать два массива, которые соответствуют осям координат и содержат один миллион случайных значений в диапазоне от -2 до 2:

$$>> x=2-4*rand(1,1000000); \text{ \% ось } x$$

$$>> y=2-4*rand(1,1000000); \text{ \% ось } y$$
- Определить логический вектор l, размерность которого равна размерности массивов координат. Если точка попадает в область, то для соответствующих координат элемент логического вектора равен единице. В противном случае (точка вне логической области) значение логического элемента равно нулю.

```
>> l=(x.^2+y.^2)>=1) & (abs(x)<=1) & (abs(y)<=1);
```

- Вывод результатов в графическое окно:

а) построение логической области черным цветом:

```
>> plot(x(l),y(l),'.k')
```

б) включение отображения координатной сетки:

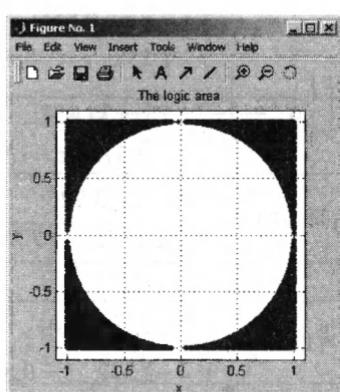
```
>> grid on
```

в) установка одинакового масштаба и границ для координатных осей:

```
>> axis equal, axis([-1.1 1.1 -1.1 1.1])
```

г) обозначение координатных осей и ввод заголовка:

```
>> xlabel('x'), ylabel('y'), title('The logic area')
```



Результат выполнения команд приведен на рис. П-4.1.

Параметром (индексом) каждого координатного вектора является логический вектор l. Это приводит к тому, что функция `plot()` автоматически принимает решение об отображении точки. Отображается только та точка, индексы которой имеют значение «true», т.е. точка, которая попала в заданную область.

4. Удалить переменные из рабочей области и закрыть графическое окно.

Рис. П-4.1. Логическая область

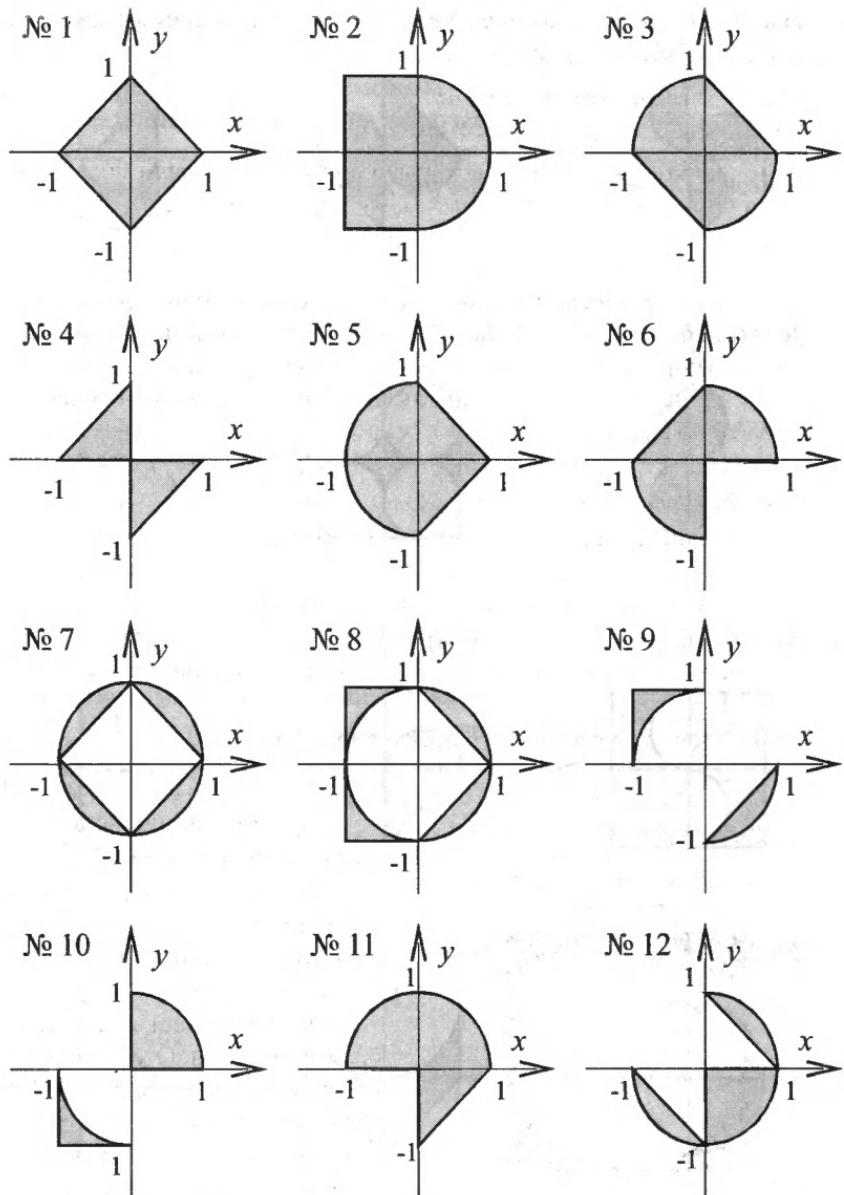


Рис. П-4.2. Варианты к заданию 2

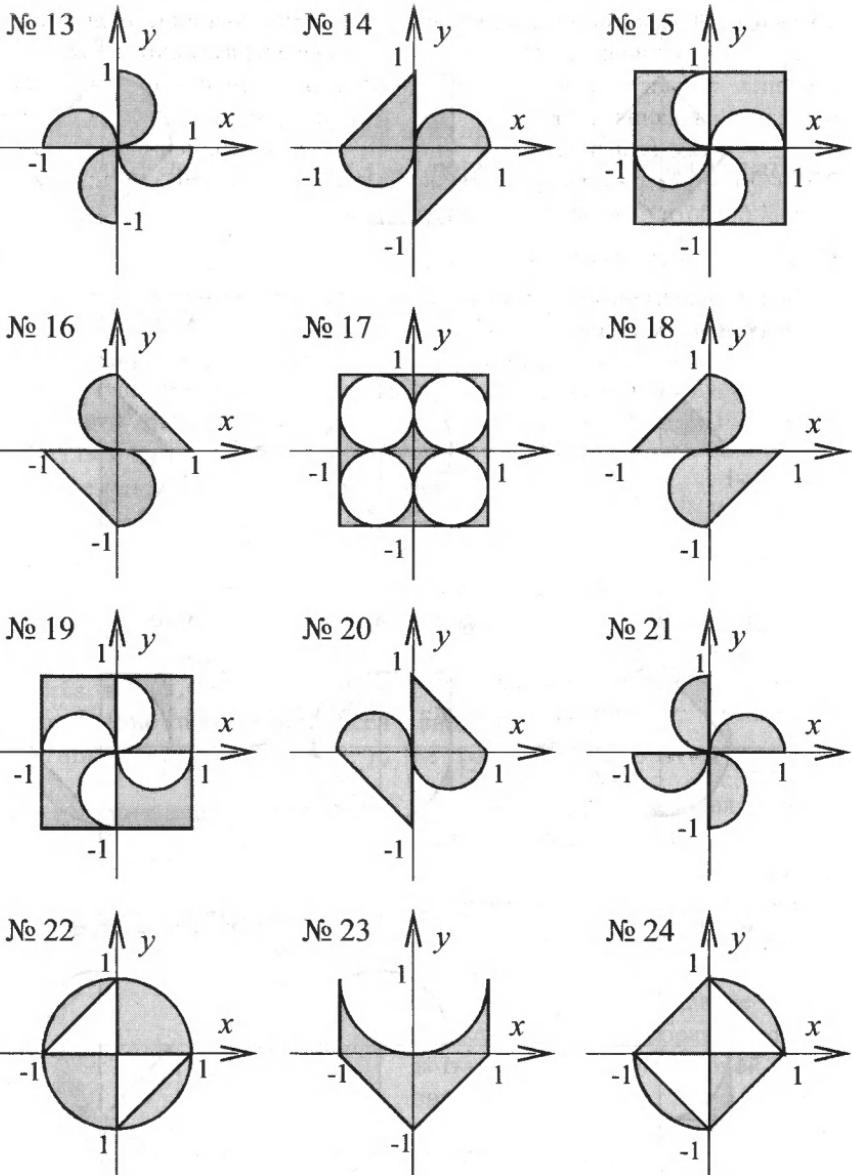


Рис. П-4.2. Варианты к заданию 2 (Продолжение)

Задание 3. Сформировать два символьных массива и вычислить функции, которые в них записаны. Первый массив содержит фамилию, имя и отчество студента, а также значения переменных и аналитическую запись функций из первого задания. Второй массив включает фамилию, имя и отчество студента, а также команды для формирования двух переменных, состоящих из 200 000 равномерно распределенных случайных значений, и описание логической области из второго задания.

Порядок выполнения задания:

1. Задать переменную символьного класса str1, которая содержит данные из первого задания:

```
>> str1 = char('Иванов', 'Сергей', 'Александрович');
>> str1 = char(str1, 'x=-15.246; y=4.642; z=0.201;');
>> str1 = char(str1, 'a=log(y^(sqrt(abs(x))))*(x-y/2)');
>> str1 = char(str1, 'b=tan(atan(z))^2')
str1 =
Иванов
Сергей
Александрович
x=-15.246; y=4.642; z=0.201;
a=log(y^(sqrt(abs(x))))*(x-y/2)
b=tan(atan(z))^2
```

2. Вывести в командное окно последовательно Ф.И.О. студента на первой строке и результаты вычислений заданных функций – на последующих строках:

```
>> [deblank(str1(1,:)), ' ', deblank(str1(2,:)), ...
' ', deblank(str1(3,:))], eval(str1(4,:)), ...
eval(str1(5,:)), eval(str1(6,:))
ans =
Иванов Сергей Александрович
a =
-105.3
b =
0.040401
```

3. Удалить переменные x и y из рабочей области с помощью функции clear():

```
>> clear('x', 'y')
```

4. Задать переменную символьного класса str2, которая содержит данные из второго задания:

```
>> str2 = char('Иванов Сергей Александрович');
>> str2 = char(str2, 'x=2-4*rand(1,200000);');
>> str2 = char(str2, 'y=2-4*rand(1,200000);');
>> str2=char(str2,'l=((x.^2+y.^2)>=1)&(abs(x)<=1)&(abs(y)<=1);')
```

```

str2 =
Иванов Сергей Александрович
x=2-4*rand(1,200000);
y=2-4*rand(1,200000);
l=((x.^2+y.^2)>=1) & (abs(x)<=1) & (abs(y)<=1);

```

5. Вычислить логическую переменную l:

```
>> eval([str2(2,1:end), str2(3,:), str2(4,:)])
```

6. Построить логическую область в графическом окне (см. предыдущее задание). В качестве заголовка вывести аналитическое описание логической области.
7. Удалить переменные из рабочей области, используя окно просмотра рабочей области.

Задание 4. Создать переменные student и stud, которые содержат в соответствующих полях фамилии, имена и отчества студентов, а также их оценки за первый семестр. При формировании переменной student использовать матричную организацию массива структуры, а при создании stud – поэлементную. Отсортировать поля по алфавиту и вычислить среднюю оценку каждого студента за семестр. При заполнении полей взять свои данные и данные предыдущего и следующего студентов по вариантам заданий. После выполнения задания удалить созданные переменные.

Порядок выполнения задания показан на примерах 2.13, 2.14 и 2.15.

Задание 5. Создать массив ячеек 1×4 . В первой ячейке расположить на отдельной строке фамилию, имя и отчество писателя. Во второй ячейке сохранить даты рождения и смерти. В третьей ячейке на отдельной строке – произведения из табл. П-4.2 и в четвертой ячейке – логический массив. В одномерном логическом массиве значение 1 («true») указывает на то, что соответствующее произведение принадлежит этому писателю. В заключение сформировать предложение: (Фамилия Имя Отчество писателя) является автором следующих произведений: (перечислить произведения).

Порядок выполнения задания показан на примерах 2.23 и 2.26.

Таблица П-4.2

Индивидуальные варианты к заданию 5

Вариант 1	Вариант 2
Федор Михайлович Достоевский (11.11.1821 – 09.02.1881) «Идиот», «Бесы», «Игрок», «Преступление и наказание» и «Мцыри»	Александр Сергеевич Пушкин (06.06.1799 – 10.02.1837) «Руслан и Людмила», «Евгений Онегин», «Бородино», «Медный всадник» и «Нос»

Продолжение табл. П-4.2

Вариант 3 Иван Сергеевич Тургенев (09.11.1818 – 03.09.1883) «Накануне», «Записки охотника», «Отцы и дети», «Рудин» и «Игрок»	Вариант 4 Михаил Юрьевич Лермонтов (15.10.1814 – 27.07.1841) «Кобзарь», «Бородино», «Мцыри», «Родина» и «Демон»
Вариант 5 Николай Алексеевич Некрасов (10.12.1821 – 08.01.1878) «Кому на Руси жить хорошо», «Русские женщины», «Мороз, Красный нос», «Коробейники» и «Калистрат»	Вариант 6 Алексей Васильевич Кольцов (15.10.1809 – 10.11.1842) «Разуверение», «Кольцо», «Косарь», «Грусть девушки» и «Мцыри»
Вариант 7 Александр Иванович Куприн (07.09.1870 – 25.08.1938) «Гамбринус», «Юнкера», «Олеся», «Яма» и «Тихий Дон»	Вариант 8 Лев Николаевич Толстой (09.09.1828 – 20.11.1910) «Война и мир», «Воскресение», «Анна Каренина», «Исповедь» и «Рудин»
Вариант 9 Александр Николаевич Островский (12.04.1823 – 14.06.1886) «Беспринадница», «Бедность не погоня», «Горячее сердце», «Гроза», «Лес» и «Без вины виноватые»	Вариант 10 Александр Александрович Блок (28.11.1880 – 07.08.1921) «Возмездие», «Двенадцать», «Интеллигентия и Революция», «Родина» и «Калистрат»
Вариант 11 Владимир Владимирович Маяковский (19.07.1893 – 14.04.1930) «Облако в штанах», «Город», «Левый марш», «Бесы» и «Секрет молодости»	Вариант 12 Михаил Александрович Шолохов (24.05.1905 – 21.02.1984) «Судьба человека», «Поднятая целина», «Они сражались за Родину», «Тихий Дон» и «Демон»
Вариант 13 Антон Павлович Чехов (29.01.1860 – 15.07.1904) «Палата № 6», «Дама с собачкой», «Человек в футляре», «Мужики» и «Кольцо»	Вариант 14 Алексей Николаевич Толстой (10.01.1883 – 23.02.1945) «Аэлита», «Детство Никиты», «Петр Первый», «Хождение по мукам» и «Гиперболоид инженера Гарина»
Вариант 15 Тарас Григорьевич Шевченко (09.03.1814 – 10.03.1861) «Цари», «Кобзарь», «Сон», «Ведьма» и «Гроза»	Вариант 16 Иван Андреевич Крылов (13.02.1769 – 21.11.1844) «Пирог», «Ворона и лисица», «Орапул», «Накануне» и «Волк на пасарне»

Вариант 17 Сергей Александрович Есенин (03.10.1895 – 28.12.1925) «Русь», «Русь советская», «Русь уходящая», «Письмо к матери» и «Бесы»	Вариант 18 Иван Алексеевич Бунин (22.10.1870 – 08.11.1953) «Деревня», «Суходол», «Жизнь Арсеньева», «Господин из Сан-Франциско» и «Левый марш»
Вариант 19 Алексей Константинович Толстой (05.09.1817 – 10.10.1875) «Упырь», «Илья Муромец», «Садко», «Князь серебряный» и «Двенадцать»	Вариант 20 Борис Леонидович Пастернак (10.02.1890 – 30.05.1960) «Доктор Живаго», «Девятьсот пятый год», «Лейтенант Шмидт», «Сестра моя – жизнь», «Конармия»
Вариант 21 Андрей Платонович Платонов (01.09.1899 – 05.01.1951) «Электрификация», «Усомнившийся Макар», «Котлован», «Сокровенный человек» и «Палата №6»	Вариант 22 Владимир Владимирович Набоков (24.04.1899 – 12.07.1977) «Машенька», «Защита Лужина», «Лолита», «Приглашение на казнь» и «Русь советская»
Вариант 23 Гавриил Романович Державин (14.07.1743 – 20.07.1816) «Ключ», «Бог», «На взятие Измаила», «Изображение Фелицы» и «Котлован»	Вариант 24 Лев Абрамович Кассиль (10.07.1905 – 21.06.1970) «Швамбрания», «Вратарь республики», «Твои защитники», «Ранний восход» и «Упырь»
Вариант 25 Михаил Афанасьевич Булгаков (15.05.1891 – 10.03.1940) «Белая гвардия», «Мастер и Маргарита», «Дни Турбиных», «Бег» и «Господин из Сан-Франциско»	

Контрольные вопросы

1. Классы данных.
2. Арифметический класс данных.
3. Логический класс данных.
4. Приоритеты операций.
5. Символьный класс данных.
6. Преобразование класса переменной.
7. Арифметические, логические и символьные выражения.
8. Массив структуры.
9. Массив ячеек.

Практикум 5. РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Цель: Приобретение навыков решения задач линейной алгебры с использованием системы MATLAB.

Порядок выполнения работы

1. Ответить на три теоретических вопроса – по одному из каждого раздела: векторная алгебра, операции с матрицами и решение систем линейных алгебраических уравнений. При ответе на вопрос привести пример в MATLAB.
2. Используя теорему Кронекера–Капелли, исследовать СЛАУ на совместность.
3. Решить СЛАУ следующими способами:
 - а) матричным методом;
 - б) методом Крамера;
 - в) методом Гаусса с частичным выбором главного элемента;
 - г) с помощью оператора \.
4. Выполнить визуализацию решения заданной системы уравнений методом Гаусса–Жордана с помощью функции `rrefmovie()`.

Метод Гаусса–Жордана

Метод Гаусса–Жордана является одной из модификаций метода Гаусса, в котором исходная матрица коэффициентов при неизвестных последовательно приводится к единичной матрице, а на месте столбца свободных членов в расширенной матрице в результате располагается решение СЛАУ:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array} \right) \Rightarrow \left(\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & x_1 \\ 0 & 1 & \cdots & 0 & x_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & x_n \end{array} \right),$$

где a_{ij} , $i = \overline{1, n}$, $j = \overline{1, n}$ – коэффициенты системы, b_i – свободные члены, x_j – неизвестные.



Мари Энмон Камиль Жордан (*Marie Ennemond Camille Jordan*) родился 5 января 1838 года в Лионе (Франция). Работы Жордана посвящены алгебре, теории функций и кристаллографии. Ему принадлежит трехтомный курс анализа (1882–1887). В 1885 году стал членом-корреспондентом Петербургской академии наук. Ввел в употребление понятие «нормальная форма матрицы». Умер 21 января 1922 года в Париже. В методе Гаусса–Жордана под фамилией Жордана понимается не Камиль Жордан, а Уильям Жордан (*Wilhelm Jordan*, 1842–1899), который первым применил этот метод.

Пример П-5.1. Решить с помощью метода Гаусса–Жордана СЛАУ

$$\begin{cases} 3 \cdot x_1 + 2 \cdot x_2 - x_3 = 4, \\ 2 \cdot x_1 - x_2 + 3 \cdot x_3 = 9, \\ x_1 - 2 \cdot x_2 + 2 \cdot x_3 = 3. \end{cases}$$

Решение:

```
>> A=[3 2 -1; 2 -1 3; 1 -2 2]; B=[4; 9; 3]; AB = [A B];
>> rref(AB)
ans =
    1         0         0         1
    0         1         0         2
    0         0         1         3
```

В результате исходная расширенная матрица приводится к *ступенчатой форме* (reduced row echelon form).

В системе MATLAB присутствует стандартная функция `rrefmovie()`, которая показывает по шагам процесс нахождения решения СЛАУ с помощью метода Гаусса–Жордана.

Пример П-5.2. Визуализировать процесс решения СЛАУ из предыдущего примера с помощью функции `rrefmovie()`.

Решение:

```
>> rrefmovie(AB)
Original matrix
A =
    3         2         -1         4
    2        -1          3         9
    1        -2          2         3
Press any key to continue. . .
pivot = A(1,1)
A =
    1        2/3        -1/3        4/3
    2        -1          3          9
    1        -2          2          3
Press any key to continue. . .
eliminate in column 1
A =
    1        2/3        -1/3        4/3
    2        -1          3          9
    1        -2          2          3
Press any key to continue. . .
A =
    1        2/3        -1/3        4/3
    0       -7/3        11/3      19/3
    1        -2          2          3
```

A =

1	2/3	-1/3	4/3
0	-7/3	11/3	19/3
0	-8/3	7/3	5/3

Press any key to continue. . .

swap rows 2 and 3

A =

1	2/3	-1/3	4/3
0	-8/3	7/3	5/3
0	-7/3	11/3	19/3

Press any key to continue. . .

pivot = A(2,2)

A =

1	2/3	-1/3	4/3
0	1	-7/8	-5/8
0	-7/3	11/3	19/3

Press any key to continue. . .

eliminate in column 2

A =

1	2/3	-1/3	4/3
0	1	-7/8	-5/8
0	-7/3	11/3	19/3

Press any key to continue. . .

A =

1	0	1/4	7/4
0	1	-7/8	-5/8
0	-7/3	11/3	19/3

A =

1	0	1/4	7/4
0	1	-7/8	-5/8
0	0	13/8	39/8

Press any key to continue. . .

pivot = A(3,3)

A =

1	0	1/4	7/4
0	1	-7/8	-5/8
0	0	1	3

Press any key to continue. . .

eliminate in column 3

A =

1	0	1/4	7/4
0	1	-7/8	-5/8
0	0	1	3

Press any key to continue. . .

A =

$$\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & -7/8 & -5/8 \\ 0 & 0 & 1 & 3 \end{matrix}$$

A =

$$\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{matrix}$$

Метод Гаусса–Жордана широко используется при нахождении обратной матрицы. В этом случае на месте столбца свободных членов стоит единичная матрица, которая преобразуется в обратную матрицу.

Описание остальных методов решения СЛАУ, а также примеры их использования представлены в разд. 4.2.

Индивидуальные задания

Варианты индивидуальных заданий приведены в табл. П-5.1.

Таблица П-5.1

Варианты заданий

№	Задание	№	Задание
1	$\begin{cases} x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 = 11, \\ 2 \cdot x_1 + 3 \cdot x_2 + 4 \cdot x_3 + x_4 = 12, \\ 3 \cdot x_1 + 4 \cdot x_2 + x_3 + 2 \cdot x_4 = 13, \\ 4 \cdot x_1 + x_2 + 2 \cdot x_3 + 3 \cdot x_4 = 14. \end{cases}$	2	$\begin{cases} x_1 + x_2 - 6 \cdot x_3 - 4 \cdot x_4 = 6, \\ 3 \cdot x_1 - x_2 - 6 \cdot x_3 - 4 \cdot x_4 = 2, \\ 2 \cdot x_1 + 3 \cdot x_2 + 9 \cdot x_3 + 2 \cdot x_4 = 6, \\ 3 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 8 \cdot x_4 = -7. \end{cases}$
3	$\begin{cases} 2 \cdot x_1 - x_2 + x_3 - x_4 = 1, \\ 2 \cdot x_1 - x_2 - 3 \cdot x_4 = 2, \\ 3 \cdot x_1 - x_3 + x_4 = -3, \\ 2 \cdot x_1 + 2 \cdot x_2 - 2 \cdot x_3 + 5 \cdot x_4 = -6 \end{cases}$	4	$\begin{cases} x_1 + 2 \cdot x_2 + 3 \cdot x_3 - 2 \cdot x_4 = 6, \\ 2 \cdot x_1 - x_2 - 2 \cdot x_3 - 3 \cdot x_4 = 8, \\ 3 \cdot x_1 + 2 \cdot x_2 - x_3 + 2 \cdot x_4 = 4, \\ 2 \cdot x_1 - 3 \cdot x_2 + 2 \cdot x_3 + x_4 = -8. \end{cases}$
5	$\begin{cases} x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 = 5, \\ 2 \cdot x_1 + x_2 + 2 \cdot x_3 + 3 \cdot x_4 = 1, \\ 3 \cdot x_1 + 2 \cdot x_2 + x_3 + 2 \cdot x_4 = 1, \\ 4 \cdot x_1 + 3 \cdot x_2 + 2 \cdot x_3 + x_4 = -5 \end{cases}$	6	$\begin{cases} x_2 - 3 \cdot x_3 + 4 \cdot x_4 = -5, \\ x_1 - 2 \cdot x_3 + 3 \cdot x_4 = -4, \\ 3 \cdot x_1 + 2 \cdot x_2 - 5 \cdot x_4 = 12, \\ 4 \cdot x_1 + 3 \cdot x_2 - 5 \cdot x_3 = 5. \end{cases}$

Продолжение табл. П-5.1

№	Задание	№	Задание
7	$\begin{cases} 2 \cdot x_1 - 2 \cdot x_2 + 3 \cdot x_3 - x_4 = 1, \\ 2 \cdot x_1 - x_2 - 3 \cdot x_4 = 2, \\ 3 \cdot x_1 - x_3 + x_4 = -3, \\ 2 \cdot x_1 + 2 \cdot x_2 - 2 \cdot x_3 + 5 \cdot x_4 = -6. \end{cases}$	8	$\begin{cases} 2 \cdot x_1 - x_2 + x_3 - x_4 = 1, \\ 2 \cdot x_1 - x_2 - 3 \cdot x_4 = 5, \\ 3 \cdot x_1 - x_2 + x_3 + x_4 = -3, \\ x_1 + 2 \cdot x_2 - 4 \cdot x_3 + 5 \cdot x_4 = -6. \end{cases}$
9	$\begin{cases} 7 \cdot x_1 - 5 \cdot x_2 + x_3 - 4 \cdot x_4 = 8, \\ -3 \cdot x_1 - 2 \cdot x_2 + x_3 + 2 \cdot x_4 = -3, \\ 3 \cdot x_1 - 4 \cdot x_2 + x_3 - x_4 = 1, \\ -x_1 + x_3 - 5 \cdot x_4 = 1. \end{cases}$	10	$\begin{cases} x_1 + 2 \cdot x_2 - 3 \cdot x_3 + 5 \cdot x_4 = 1, \\ 4 \cdot x_1 - x_2 + 5 \cdot x_3 - 2 \cdot x_4 = 1, \\ 3 \cdot x_1 + 5 \cdot x_2 - 3 \cdot x_3 - 2 \cdot x_4 = -1, \\ x_1 + 2 \cdot x_2 - x_3 - x_4 = -1. \end{cases}$
11	$\begin{cases} 2 \cdot x_1 + 5 \cdot x_2 + 4 \cdot x_3 + x_4 = 2, \\ x_1 + 3 \cdot x_2 + 2 \cdot x_3 + x_4 = -4, \\ 2 \cdot x_1 + 10 \cdot x_2 + 9 \cdot x_3 + 9 \cdot x_4 = -4, \\ 3 \cdot x_1 + 8 \cdot x_2 + 9 \cdot x_3 + 2 \cdot x_4 = 1. \end{cases}$	12	$\begin{cases} 3 \cdot x_1 - x_2 + x_4 = 13, \\ 3 \cdot x_1 - 8 \cdot x_2 + 5 \cdot x_3 + x_4 = -23, \\ 4 \cdot x_1 - 7 \cdot x_2 + 14 \cdot x_3 + 5 \cdot x_4 = -5, \\ x_1 + 2 \cdot x_2 - 3 \cdot x_3 - x_4 = 15. \end{cases}$
13	$\begin{cases} x_1 - 7 \cdot x_2 + 7 \cdot x_3 - x_4 = 1, \\ 2 \cdot x_1 + x_2 + x_4 = 2, \\ 3 \cdot x_1 - x_2 + 2 \cdot x_3 + x_4 = 2, \\ 2 \cdot x_1 + 2 \cdot x_2 + 5 \cdot x_3 + x_4 = 1. \end{cases}$	14	$\begin{cases} 3 \cdot x_1 - 7 \cdot x_2 + 7 \cdot x_3 + 2 \cdot x_4 = -22, \\ x_1 - 8 \cdot x_2 + 10 \cdot x_3 + 3 \cdot x_4 = -35, \\ 4 \cdot x_1 - 7 \cdot x_2 + 14 \cdot x_3 + 5 \cdot x_4 = -48, \\ x_1 + 2 \cdot x_2 - 3 \cdot x_3 - x_4 = 12. \end{cases}$
15	$\begin{cases} 4 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 + x_4 = 3, \\ 2 \cdot x_1 + x_2 + 3 \cdot x_3 + x_4 = -1, \\ 6 \cdot x_1 + 5 \cdot x_2 + x_3 + 2 \cdot x_4 = -4, \\ 3 \cdot x_1 + x_2 + 2 \cdot x_3 + 3 \cdot x_4 = -2. \end{cases}$	16	$\begin{cases} 5 \cdot x_1 + x_2 + x_3 + x_4 = 1, \\ 2 \cdot x_1 + 5 \cdot x_2 + 2 \cdot x_3 + 2 \cdot x_4 = 2, \\ 3 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 3 \cdot x_4 = 3, \\ 4 \cdot x_1 + 4 \cdot x_2 + 4 \cdot x_3 + 5 \cdot x_4 = 4. \end{cases}$
17	$\begin{cases} -x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 = -10, \\ -x_1 - 3 \cdot x_2 - x_3 - 3 \cdot x_4 = -7, \\ -x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 3 \cdot x_4 = 3, \\ 5 \cdot x_1 + 4 \cdot x_2 + 4 \cdot x_3 + 5 \cdot x_4 = 4. \end{cases}$	18	$\begin{cases} 5 \cdot x_1 + 4 \cdot x_2 + 3 \cdot x_3 + 2 \cdot x_4 = 13, \\ 4 \cdot x_1 + 3 \cdot x_2 + 2 \cdot x_3 + 5 \cdot x_4 = 11, \\ 3 \cdot x_1 + 2 \cdot x_2 + 5 \cdot x_3 + 4 \cdot x_4 = 10, \\ 2 \cdot x_1 + 5 \cdot x_2 + 4 \cdot x_3 + 3 \cdot x_4 = 12. \end{cases}$

№	Задание	№	Задание
19	$\begin{cases} 2 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 4 \cdot x_4 = 2, \\ 5 \cdot x_1 + 6 \cdot x_2 + 4 \cdot x_3 + 5 \cdot x_4 = -1, \\ 4 \cdot x_1 - x_2 - 8 \cdot x_3 + 2 \cdot x_4 = 2, \\ 2 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 2 \cdot x_4 = -2. \end{cases}$	20	$\begin{cases} x_1 - x_3 - 2 \cdot x_4 = -3, \\ -3 \cdot x_1 - 4 \cdot x_2 - 4 \cdot x_3 - 3 \cdot x_4 = -2, \\ -4 \cdot x_1 - 3 \cdot x_2 - 2 \cdot x_3 - x_4 = -1, \\ -5 \cdot x_1 + 3 \cdot x_2 + 2 \cdot x_3 + x_4 = 0. \end{cases}$
21	$\begin{cases} 2 \cdot x_1 + x_2 - 2 \cdot x_3 - 3 \cdot x_4 = 5, \\ -5 \cdot x_1 - 3 \cdot x_2 + x_3 + 5 \cdot x_4 = 3, \\ 2 \cdot x_1 - 5 \cdot x_2 - x_3 + 5 \cdot x_4 = 5, \\ 4 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 - x_4 = 1. \end{cases}$	22	$\begin{cases} -7 \cdot x_1 + 8 \cdot x_2 - 9 \cdot x_3 + 10 \cdot x_4 = -1, \\ 8 \cdot x_1 - 9 \cdot x_2 + 10 \cdot x_3 - 7 \cdot x_4 = 2, \\ -9 \cdot x_1 + 10 \cdot x_2 - 7 \cdot x_3 + 8 \cdot x_4 = -3, \\ 10 \cdot x_1 - 7 \cdot x_2 + 8 \cdot x_3 - 9 \cdot x_4 = 4. \end{cases}$
23	$\begin{cases} -5 \cdot x_1 + x_2 + x_3 + 2 \cdot x_4 = 1, \\ x_1 + 4 \cdot x_2 + 5 \cdot x_3 + 8 \cdot x_4 = 5, \\ 6 \cdot x_1 + 4 \cdot x_2 + 4 \cdot x_3 + 6 \cdot x_4 = -1, \\ -x_1 + x_2 + x_3 - x_4 = 1. \end{cases}$	24	$\begin{cases} 3 \cdot x_1 - 7 \cdot x_2 + 7 \cdot x_3 + 2 \cdot x_4 = -22, \\ x_1 - 8 \cdot x_2 + 9 \cdot x_3 + 2 \cdot x_4 = -35, \\ 4 \cdot x_1 - 7 \cdot x_2 + 14 \cdot x_3 + 5 \cdot x_4 = -48, \\ x_1 + 2 \cdot x_2 - 3 \cdot x_3 - x_4 = 12. \end{cases}$
25	$\begin{cases} 3 \cdot x_1 - 7 \cdot x_2 + 7 \cdot x_3 + 2 \cdot x_4 = 8, \\ x_1 - 8 \cdot x_2 + 9 \cdot x_3 + 2 \cdot x_4 = 3, \\ 4 \cdot x_1 - 2 \cdot x_2 + 3 \cdot x_3 + x_4 = 17, \\ 5 \cdot x_1 - 17 \cdot x_2 + x_3 - 2 \cdot x_4 = -24. \end{cases}$		

Решения заданных СЛАУ представлены в табл. П-5.2.

Таблица П-5.2

Решения СЛАУ

№		№		№		№		№	
1	$X = \begin{vmatrix} 2 \\ 1 \\ 1 \\ 1 \end{vmatrix}$	2	$X = \begin{vmatrix} 0 \\ 2 \\ 1/3 \\ -3/2 \end{vmatrix}$	3	$X = \begin{vmatrix} 0 \\ 2 \\ 5/3 \\ -4/3 \end{vmatrix}$	4	$X = \begin{vmatrix} 1 \\ 2 \\ -1 \\ -2 \end{vmatrix}$	5	$X = \begin{vmatrix} -2 \\ 2 \\ -3 \\ 3 \end{vmatrix}$

Окончание табл. П-5.2

№		№		№		№		№	
6	$X = \begin{vmatrix} 1 \\ 2 \\ 1 \\ -1 \end{vmatrix}$	7	$X = \begin{vmatrix} -4/21 \\ 34/21 \\ 23/21 \\ -4/3 \end{vmatrix}$	8	$X = \begin{vmatrix} 4/3 \\ 17/3 \\ 4/3 \\ -8/3 \end{vmatrix}$	9	$X = \begin{vmatrix} -19 \\ -35 \\ -98 \\ -16 \end{vmatrix}$	10	$X = \begin{vmatrix} 8/5 \\ -11/5 \\ -8/5 \\ -1/5 \end{vmatrix}$
11	$X = \begin{vmatrix} 50 \\ -23 \\ 1 \\ 13 \end{vmatrix}$	12	$X = \begin{vmatrix} 6 \\ 11/2 \\ 1/2 \\ 1/2 \end{vmatrix}$	13	$X = \begin{vmatrix} -38 \\ -16 \\ 3 \\ 94 \end{vmatrix}$	14	$X = \begin{vmatrix} 1 \\ 0 \\ -3 \\ -2 \end{vmatrix}$	15	$X = \begin{vmatrix} 13/3 \\ -14/3 \\ -2/3 \\ -3 \end{vmatrix}$
16	$X = \begin{vmatrix} 3/89 \\ 8/89 \\ 18/89 \\ 48/89 \end{vmatrix}$	17	$X = \begin{vmatrix} 2/3 \\ 25/3 \\ -2/3 \\ -6 \end{vmatrix}$	18	$X = \begin{vmatrix} 15/14 \\ 37/28 \\ 4/7 \\ 9/28 \end{vmatrix}$	19	$X = \begin{vmatrix} -7/5 \\ -2/5 \\ -2/5 \\ 2 \end{vmatrix}$	20	$X = \begin{vmatrix} 1/9 \\ 4/3 \\ -10/3 \\ 29/9 \end{vmatrix}$
21	$X = \begin{vmatrix} -6 \\ -50 \\ 28 \\ -41 \end{vmatrix}$	22	$X = \begin{vmatrix} 14/17 \\ 3/17 \\ -3/17 \\ 3/17 \end{vmatrix}$	23	$X = \begin{vmatrix} -3/14 \\ -5 \\ 11/2 \\ -2/7 \end{vmatrix}$	24	$X = \begin{vmatrix} 1 \\ -1 \\ -6 \\ 5 \end{vmatrix}$	25	$X = \begin{vmatrix} 4 \\ 3 \\ 3 \\ -2 \end{vmatrix}$

Контрольные вопросы

- Что такое вектор? Задание вектора и обращение к элементам вектора в системе MATLAB.
- Линейные операции над векторами и их свойства.
- Как определяется длина вектора? Определить длину вектора $\bar{a} = (1;5;2)$.
- Скалярное произведение векторов в координатной форме и его свойства.
- Как определяется угол между векторами? Вычислить угол между векторами $a = (2;1;3)$ и $b = (4;0;1)$.
- Направляющие косинусы вектора.
- Векторное произведение двух векторов и его свойства.
- Смешанное произведение трех векторов и его свойства.
- Внешнее произведение.
- Норма вектора.
- Определение расстояния между векторами.

12. Что такое матрица?
13. Виды специальных матриц.
14. Какие матрицы называются равными?
15. Каковы линейные операции над матрицами и их свойства?
16. Что такое произведение матриц?
17. Что такое транспонирование матрицы и транспонирование матрицы с комплексным сопряжением?
18. Что такое определитель матрицы и каковы его свойства? Привести примеры вычисления определителя и рассмотреть его свойства с матрицей третьего порядка в системе MATLAB.
19. Что такое обратная матрица? Привести формулу ее вычисления.
20. Что такое норма матрицы? Какими свойствами обладает норма матрицы и какие виды нормы матрицы существуют?
21. Дать определение ранга матрицы. Каковы его свойства?
22. Как записать СЛАУ в матричном виде?
23. Как решить матричное уравнение? Пример решения матричного уравнения в MATLAB для СЛАУ с тремя неизвестными.
24. Формулы Крамера. Пример использования формул Крамера в MATLAB для решения СЛАУ с тремя неизвестными.
25. Метод Гаусса с частичным выбором главного элемента. Привести пример реализации метода Гаусса с частичным выбором главного элемента в MATLAB для СЛАУ с тремя неизвестными.
26. Метод Гаусса с полным выбором главного элемента. Привести пример реализации метода в MATLAB для СЛАУ с тремя неизвестными.
27. Опишите LU-разложение и приведите пример в MATLAB для СЛАУ с тремя неизвестными.
28. Метод Гаусса–Жордана. Приведите пример решения СЛАУ с тремя неизвестными методом Гаусса–Жордана.
29. Какие методы решения СЛАУ реализованы в MATLAB? Используя один из методов решения СЛАУ в MATLAB, привести пример для решения СЛАУ с тремя неизвестными.
30. Теорема Кронекера–Капели. Привести пример использования теоремы Кронекера–Капелли в MATLAB для СЛАУ с тремя неизвестными.
31. Какова последовательность действий системы MATLAB при решении СЛАУ с помощью оператора «/»? Диагностические сообщения при решении СЛАУ.

Практикумы 6, 7. ГРАФИЧЕСКИЕ СРЕДСТВА MATLAB

Цель: Знакомство с графическими возможностями системы MATLAB.

Дескрипторная графика

Для визуализации графической информации в системе MATLAB используются графические объекты. Создание и отображение графических объектов основано на принципах дескрипторной графики (Handle Graphics). Иерархическая структура объектов дескрипторной графики представлена на рис. П-6.1.

Иерархическая структура объектов дескрипторной графики, состоящая из четырех уровней (0, 1, 2 и 3), позволяет определить взаимосвязь в терминах «родитель – потомок» между графическими объектами.

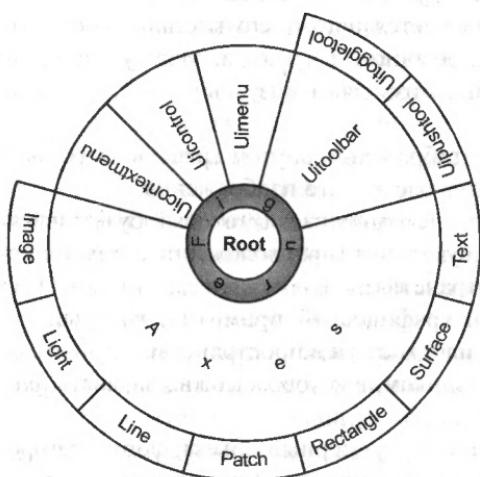
Корневой объект **Root**, расположенный на нулевом уровне иерархии и представляющий собой экран монитора, создается во время запуска MATLAB и существует в течение всего сеанса работы с системой. Корневой объект не имеет родительских объектов.

Объект **Figure** является отдельным окном на экране монитора, где отображается графическая информация. Несмотря на

Рис. П-6.1. Иерархическая структура объектов дескрипторной графики

то что система MATLAB позволяет создавать произвольное число графических окон, только одно из них является текущим. Команды по созданию, управлению видимостью и удалению объектов, являющихся «потомками» графического окна, будут применяться для текущего на данный момент объекта **Figure**. Все графические окна являются «потомками» корневого объекта.

В пределах одного графического окна можно создавать несколько объектов координатного пространства (**Axes**). Как и в случае графических окон, только один из этих объектов является текущим. Следовательно, все команды по созданию объектов, являющихся потомками координатного пространства, будут применяться для текущего объекта **Axes**. Графическое окно является «родителем» для координатного пространства. Внутри координатного про-



странства можно создавать несколько однотипных объектов. Координатные оси объекта *Axes* используются для размещения «потомков» в пределах координатного пространства.

Графические объекты *Uicontextmenu*, *Uicontrol* и *Uimenu* являются независимыми от координатного пространства объектами и позволяют создавать в пределах графического окна элементы управления интерфейса пользователя, запускающие соответствующие функции при их активизации. Как и для объекта *Axes*, «родителем» для них является графическое окно.

Графический объект *UIToolbar* представляет собой панель инструментов, на которой могут размещаться объекты, являющиеся кнопками быстрого доступа к командам. Кнопки могут быть двух типов – *UIpushbutton* и *UITogglebutton*. Основное отличие между ними заключается в том, что кнопки *UITogglebutton* могут находиться в двух состояниях: обычном и нажатом. Между кнопками можно создавать разделители, которые позволяют визуально отделить одну группу кнопок от другой.

Объект *Image* используется для отображения внутри графического окна в пределах координатного пространства растрового изображения.

Объект *Light* позволяет определить источник света, который будет действовать на все поверхности или многоугольники, расположенные внутри соответствующего координатного пространства.

Объект *Line* представляет собой графический примитив, который используется для построения линии на плоскости и в пространстве.

Объект *Patch* является многоугольником, в котором можно задавать цвет поверхности и ребер.

Объект *Rectangle* представляет собой двухмерный объект, форма которого может изменяться от эллипса до прямоугольника. Наиболее полезно использовать этот объект при создании блок-схем.

Объект *Surface* используется для построения поверхностей в координатном пространстве. Поверхности строятся путем соединения соседних точек, координаты которых содержатся в трехмерном массиве. Первая размерность соответствует значениям координат точек по оси *x*, вторая размерность – по оси *y* и третья размерность – по оси *z*.

Объект *Text* является строкой символов. Этот объект используется для вывода текста внутри графического окна.

Таким образом, чтобы изобразить поверхность, сначала необходимо создать графическое окно, а затем в нем создать координатное пространство, в котором будет отображаться заданная поверхность. В том случае, если при создании «потомка» (поверхности) родительский объект (координатное пространство) или объекты (графическое окно и координатное пространство) не существуют, то система MATLAB сначала создаст соответствующие родительские объекты, а уже затем сформирует необходимый объект.

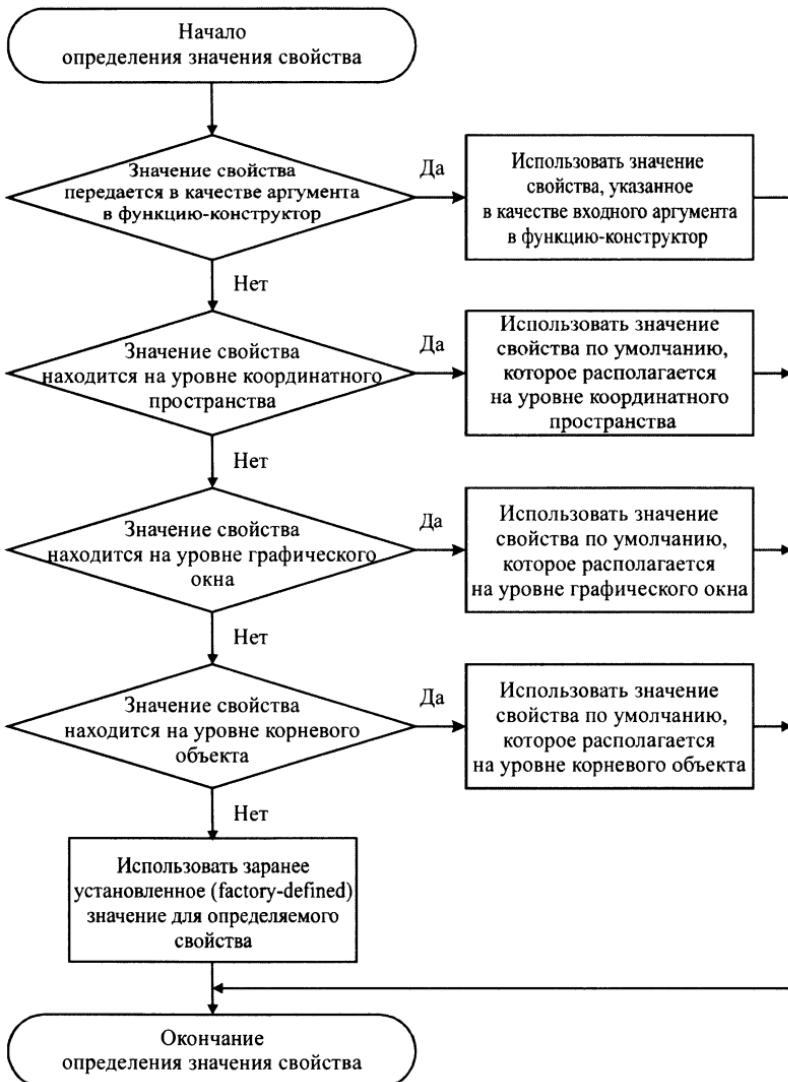
Все объекты дескрипторной графики создаются при помощи *функций-конструкторов*, имена которых совпадают с именами создаваемых объектов. После создания соответствующего объекта функция-конструктор возвращает число, которое является *дескриптором* этого объекта, т.е. ссылкой на этот объект. Для последующего обращения к объекту следует при его создании присвоить переменной возвращаемое число. Корневой объект всегда имеет дескриптор, равный нулевому значению.

Объекты дескрипторной графики состоят из свойств (properties), которые определяют внешний вид объекта (например, размер графического окна, масштаб осей координатного пространства, толщина линии, размер шрифта текстовой надписи и т.д.), и функций (callbacks), в которых описывается реакция на соответствующие события (например, нажатие на кнопку, перемещение указателя мыши, удаление объекта и т.д.).

Во время создания объекта дескрипторной графики система MATLAB осуществляет поиск значений по умолчанию для всех свойств, начиная с текущего объекта, и продолжает поиск в направлении родительских объектов до тех пор, пока не дойдет до значений, встроенных в систему MATLAB (так называемые заводские установки – factory-defined values). На рис. П-6.2 изображена последовательность действий, которые выполняет система MATLAB при задании значений свойствам объектов.

Чем ближе к корневому объекту определяется свойство со значением по умолчанию, тем более широкую область видимости оно имеет. Если на уровне координатного пространства задать значение по умолчанию свойства, соответствующего толщине линии (DefaultLineLineWidth), равным двум пунктам, то все линии, которые будут создаваться впоследствии в пределах этого координатного пространства, будут иметь толщину два пункта. Однако на толщину линий, создаваемых внутри другого координатного пространства, значение по умолчанию свойства этого уровня влияния не оказывает. Для изменения толщины линий, выводимых во всех координатных пространствах графического окна, следует задать необходимое значение по умолчанию соответствующему свойству на уровне графического окна. Однако на толщину линий, создаваемых в пределах другого графического окна, значение по умолчанию свойства этого уровня влияния не оказывает. С целью изменения толщины линий, создаваемых в координатных пространствах различных графических окон, следует задать необходимое значение по умолчанию соответствующему свойству на уровне корневого объекта. Несмотря на широкую область видимости свойств со значениями по умолчанию, которые располагаются на верхних уровнях иерархии, они имеют более низкий приоритет по сравнению с соответствующими свойствами на низких уровнях иерархии. Следовательно, если присвоено значение по умолчанию свойству на уровне координатного пространства, то система MATLAB при

создании графического объекта не будет просматривать значения по умолчанию соответствующих свойств на уровнях графического окна и корневого объекта.



Р и с . П-6.2. Последовательность шагов при задании значения свойству объекта дескрипторной графики

Имена свойств, которые содержат значения по умолчанию, формируются следующим образом: Default + имя объекта + имя свойства объекта. Например, DefaultFigureColor – имя свойства, которое содержит значение по умолчанию для цвета фона графических окон. Значения для свойств объектов дескрипторной графики задаются с помощью функции set(), а считаются с помощью функции get(). Например, при задании в командном окне команды >> set(0, 'DefaultFigureColor', 'w') цвет фона всех графических окон, создаваемых впоследствии в течение сеанса работы с MATLAB со значениями по умолчанию, будет белым. Удалить значение по умолчанию можно с помощью задания значения 'remove'. Например, при задании следующей команды >> set(0, 'DefaultFigureColor', 'remove') будет удалено значение по умолчанию из свойства DefaultFigureColor. Цвет фона создаваемого графического окна в данном случае будет определяться с помощью значения свойства factoryFigureColor, которое равно [0 0 0], т.е. цвет будет черным.

Просмотреть все предопределенные значения свойств можно с помощью команды >> get(0, 'factory'). Определенные значения по умолчанию для корневого объекта можно просмотреть с помощью задания команды >> get(0, 'Default').

Во время запуска MATLAB выполняется файл matlabrc.m, в котором на уровне корневого объекта задаются значения по умолчанию для свойств, определяющих положение графического окна на экране (DefaultFigurePosition), наличие или отсутствие панелей инструментов в графическом окне (DefaultFigureToolBar) и т.д. Задание пользовательских значений по умолчанию для свойств на уровне корневого объекта, создание приветствующих сообщений, загрузка начальных данных, а также ряд других действий, выполняемых при инициализации системы, можно реализовать с помощью файла startup.m, к которому обращается система после выполнения файла matlabrc.m.

Упражнение 1. Создать модальное диалоговое окно во время запуска MATLAB, в котором отображается сообщение «Вы вошли в мир Mat^{rix} Lab^{oratory}» с картинкой из файла ..\MATLAB6p5\bin\win32\grfwnd.ico. В строке заголовка диалогового окна вывести надпись «Привет!». Создание диалогового окна реализовать в файле startup.m и сохранить его в директории ..\MATLAB6p5\work.

Модальным называется такое диалоговое окно, которое держит фокус до тех пор, пока оно не будет закрыто. В данном случае, пока выведенное диалоговое окно не будет закрыто, любое другое окно системы MATLAB нельзя будет сделать активным.

Система MATLAB позволяет отображать на экране ряд простых диалоговых окон общего назначения, в которых могут находиться сообщения различного рода. Наряду с сообщениями в диалоговом окне могут располагаться картинки. В табл. П-6.1 представлены все типы диалоговых окон, в которых выводятся сообщения и соответствующие им картинки.

Диалоговое окно общего назначения с выводом сообщения и произвольной картинки создается при помощи функции `msgbox(message, title, 'custom', iconData, iconCmap, createMode)`.

Таблица П-6.1

Типы сообщений и соответствующие им картинки

Тип	пользовательский	ошибка	подсказка	предупреждение
Опция	'none'	'custom'	'error'	'help'
Картинка	нет	произвольная		

Первый входной аргумент `message` содержит сообщение, которое выводится в диалоговое окно. Этот аргумент должен быть строкового класса. Если необходимо создать сообщение, состоящее из нескольких строк, то следует использовать массив строковых ячеек. Второй аргумент `title` содержит название диалогового окна. Этот аргумент также должен быть строкового класса. Третий аргумент – определяет тип диалогового окна. Все возможные варианты значений третьего аргумента представлены в табл. П-6.1. В том случае, если выбирается пользовательский тип диалогового окна с выводом графического изображения, необходимо задать четвертый (`iconData`) и пятый (`iconCmap`) параметры, которые определяют изображение картинки и ее цветовую палитру соответственно. Шестой аргумент (`createMode`) позволяет задавать режим диалогового окна и включать интерпретатор TeX для вывода сообщений.

При создании диалогового окна общего назначения с помощью функции `msgbox()` происходит обращение к функции `dialog()`, которая создает новое графическое окно и устанавливает значения соответствующих свойств таким образом, чтобы созданное графическое окно было похоже на диалоговое окно. Дополнительную информацию по диалоговым окнам системы MATLAB можно получить с помощью команд `>> help msgbox` или `>> help dialog`.

Создайте с помощью любого текстового редактора (например, Notepad) файл `startup.m` и сохраните его в директории `..\MATLAB\work`. Содержание файла `startup.m` представлено на листинге П-6.1. После создания файла загрузите MATLAB.

Листинг П.-6.1

```
[iconData, iconCmap] = imread('grfwnd','ico');
createMode.WindowStyle = 'modal';
createMode.Interpreter = 'tex';
message = ['    Вы вошли в мир'; ' \bf Mat^{\rm rix} Lab^{\rm oratory}'];
msgbox(message, 'Привет!', 'custom', iconData, iconCmap, createMode)
clear message iconData iconCmap createMode
```

Результаты команд, помещенных в файл startup.m, изображены на рис. П-6.3.

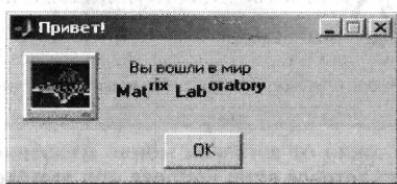


Рис. П-6.3. Диалоговое окно при входе в MATLAB

Упражнение 2. Создать диалоговое окно вопроса при завершении сеанса работы с системой MATLAB, в котором вывести вопрос: «Вы действительно хотите покинуть мир *Mat^{rix} Lab^{oratory}?*» – и две кнопки «Да» и «Нет» с активной кнопкой «Нет».

После задания команды quit система MATLAB производит поиск файла с именем finish.m. В том случае, если файл найден, последовательно выполняются все команды, расположенные в этом файле.

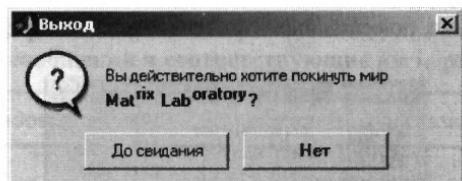
Для создания модального диалогового окна вопроса следует использовать функцию questdlg(Question,Title,Button1,Button2,Options). Первые четыре аргумента являются переменными строкового класса, а пятый представляет собой структуру, состоящую из двух полей – Default и Interpreter, которые принимают значения строкового класса. Первый входной аргумент содержит вопрос, второй – название диалогового окна, третий и четвертый – названия первой и второй кнопки соответственно. В поле Default находится имя кнопки, активной по умолчанию. Нажатие на клавишу Enter будет соответствовать нажатию на эту кнопку. Поле Interpreter позволяет использовать команды системы TeX.

Создайте с помощью любого текстового редактора файл finish.m и сохраните его в директории ..\MATLAB\work. Содержание файла finish.m представлено на листинге П-6.2. Диалоговое окно, появляющееся при выходе из MATLAB, представлено на рис. П-6.4.

```

quest = {'Вы действительно хотите покинуть мир';
'`bfMat^{rix} Lab^{oratory}?'};
Options.Default = 'Нет'; Options.Interpreter = 'tex'; l = false;
l = strcmp(questdlg(quest,'Выход','До свидания','Нет',Options),
'Dо свидания');
if l ~= true, quit cancel, end

```



Р и с . П-6.4. Диалоговое окно вопроса при выходе из MATLAB

В файл finish.m следует помещать команды, которые выполняют завершающие операции с данными, расположенными в рабочей области (например, сохранение данных в файле).

Упражнение 3. С помощью файла startup.m задать размеры графического окна и его положение на экране, которые будут использоваться при построении всех графических окон по умолчанию. Высота графического окна равна 350 точкам, а ширина – 450 точкам. Графическое окно должно выводиться в центре экрана. С помощью команд, задаваемых в командной строке, создать два графических окна. Первому графическому окну присвоить имя fig1, второму – fig2. Сделать текущим окно fig1. В строке заголовка текущего графического окна вывести надпись – «Первое окно». Сделать текущим окно fig2. Изменить цвет фона текущего графического окна на черный.

В созданный в первом упражнении файл с именем startup.m перед командой clear (последней строкой) добавьте следующие строки:

```

screen = get(0,'Screensize');
width = 450; height = 350;
position=[fix((screen(3)-width)/2),...
fix((screen(4)-height)/2),width,height];
set(0,'DefaultFigurePosition', position);
clear screen position width height

```

Первая команда записывает в переменную screen разрешение экрана. Если разрешение экрана равно 1024×768, то переменная screen будет

равна [1 1 1024 768]. В переменной `width` содержится ширина графического окна, а в переменной `height` – его высота. Переменная `position` содержит координаты графического окна относительно экрана монитора.

В командную строку введите команду `>> quit` и в появившемся диалоговом окне нажмите на кнопку «До свидания». Заново загрузите систему MATLAB.

В командную строку введите следующие команды:

```
>> fig1 = figure
fig1 =
    1
>> fig2 = figure
fig2 =
    2
```

В результате задания двух команд в центре экрана были созданы два графических окна, дескрипторы которых `fig1` и `fig2` соответственно.

Текущим графическим окном является то окно, которое было активным последним, в данном случае – `fig2`. Активизация графического окна происходит либо щелчком мыши в области этого окна или его выбором из панели задач, либо с помощью задания команды `figure(handle)`, где `handle` – имя дескриптора окна. Для изменения надписи в строке заголовка первого окна и цвета фона второго окна введите в командное окно следующие команды:

```
>> figure(fig1)
>> set(gcf, 'Name', 'Первое окно')
>> figure(fig2)
>> set(gcf, 'Color', 'w')
```

После выполнения примеров удалите созданные файлы (`startup.m` и `finish.m`) с помощью инструментального средства `Current Directory`.

Графическое окно

При решении математических задач графические окна используются в основном для визуализации линий, поверхностей в координатном пространстве и их обозначений в пределах графического окна. Один из вариантов графического окна изображен на рис. П-6.5.

Графическое окно, представленное на рис. П-6.5, состоит из строки заголовка, строки главного меню, стандартной панели инструментов, панели инструментов «Камера», а также рабочей области, где отображается графическая информация.

Строка главного меню состоит по умолчанию из семи пунктов: `File`, `Edit`, `View`, `Insert`, `Tools`, `Window` и `Help`. Пункт меню `Параметры` был добавлен с помощью функции-конструктора `uimenu()`.

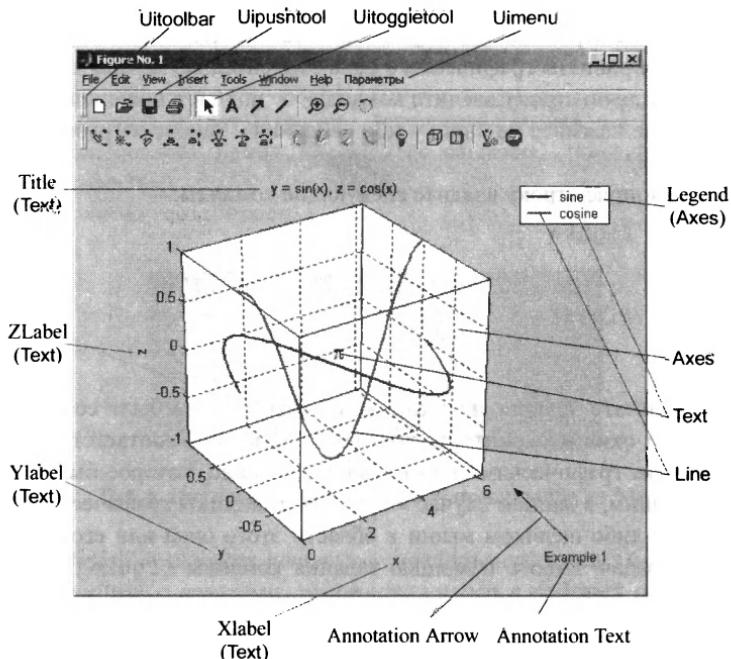


Рис. П-6.5. Графическое окно

В пункте меню **File** содержатся 11 команд, разбитых на четыре группы (рис. П-6.6, а). Команда **New Figure** создает новое графическое окно. Команда **Open...** открывает одноименное диалоговое окно, где можно выбрать для открытия файл, содержащий графическое окно. Команда **Close** закрывает графическое окно. Следующая группа команд позволяет сохранить графическое окно в файле. Команды **Save** и **Save As...** используются для сохранения графического окна в файле с расширением fig. Команда **Export...** используется для сохранения графического окна в файлах различного формата в виде векторной (emf, eps или ai) или растровой (bmp, tif, jpg, png и т.д.) графики. Команда **Preferences...** открывает одноименное диалоговое окно для изменения настроек системы MATLAB. Заключительная группа команд предназначена для настройки и запуска процесса печати содержимого графического окна. Команда **Page Setup...** открывает диалоговое окно, где можно задать размер листа бумаги, который будет использоваться для печати (например, А4 (210×297 мм)), размер и положение содержимого графического окна на листе бумаги, а также ряд других параметров, влияющих на цвет изображения. Команда **Print Setup...** открывает одноименное диалоговое окно, в кото-

ром осуществляется выбор принтера и задание его свойств для печати. Команда **Print Preview...** создает окно, в котором отображается примерный вид твердой копии. Команда **Print...** вызывает одноименное диалоговое окно, с помощью которого запускается процесс печати.

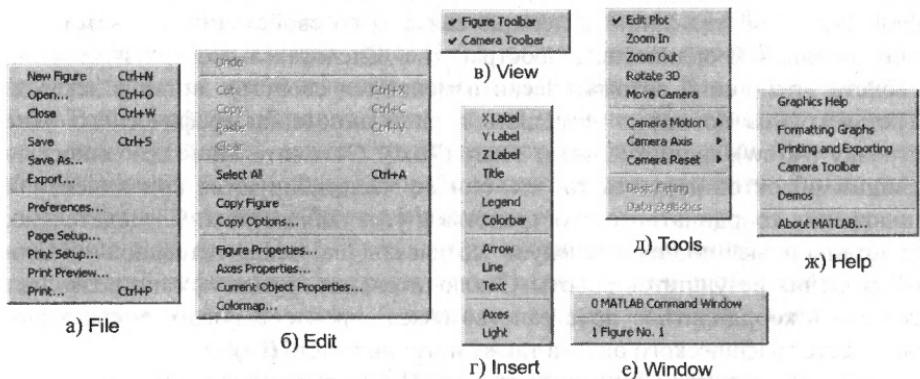


Рис. П-6.6. Пункты главного меню графического окна

Команды меню **Edit**, показанного на рис. П-6.6, б, позволяют редактировать объекты графического окна. Первые шесть команд совпадают по назначению с соответствующими командами меню **Edit** рабочего стола, но применяются к графическим объектам. Команда **Copy Figure** копирует рабочую область графического окна в буфер обмена в формате, который задается с помощью диалогового окна **Preferences** (**Figure Copy Template – Copy Options**), вызываемого командой **Copy Options...**. Следующие три команды – **Figure Properties...**, **Axes Properties...** и **Current Object Properties...** – вызывают Редактор свойств (**Property Editor**) и настраивают его для редактирования значений свойств графического окна, координатного пространства и текущего объекта соответственно. Последняя команда (**Colormap...**) вызывает Редактор цветовой палитры (**Colormap Editor**), с помощью которого можно выбрать и настроить цветовую палитру графического окна.

С помощью команд меню **View** (рис. П-6.6, в) можно включать или отключать видимость стандартной панели инструментов (**Figure Toolbar**) и панели инструментов «Камера» (**Camera Toolbar**). Пункты меню работают как переключатели. Если панель инструментов присутствует в графическом окне, то напротив соответствующего пункта находится «».

Команды меню **Insert** (рис. П-6.6, г) предназначены для создания объектов в рабочем поле графического окна. Команды этого меню разделены на четыре группы. В первой группе находятся команды, которые позволяют вставить обозначения оси абсцисс (**XLabel**), оси ординат (**YLabel**) и оси

аппликат (*ZLabel*), а также заголовок (*Title*) в текущее координатное пространство. Следующая группа команд позволяет вставить в графическое окно шкалу цветовой палитры (*Colorbar*) и в текущее координатное пространство – обозначение выведенных линий (*Legend*). После создания этих объектов их можно перемещать в пределах графического окна и изменять их свойства. Свойства линий в легенде связаны со свойствами соответствующих линий в координатном пространстве. Следовательно, при изменении свойств этих линий автоматически изменяются свойства линий в легенде. Третья группа позволяет помещать в слой аннотации графического окна стрелку (*Arrow*), линию (*Line*) и текст (*Text*). Отличительной особенностью данных объектов является то, что они не масштабируются при изменении положения координатного пространства и угла наблюдения. Следовательно, аннотационные подписи следует создавать на заключительной стадии оформления результатов работы. Заключительная группа команд позволяет создавать координатное пространство (*Axes*) произвольного размера в любом месте графического окна, а также источник света (*Light*).

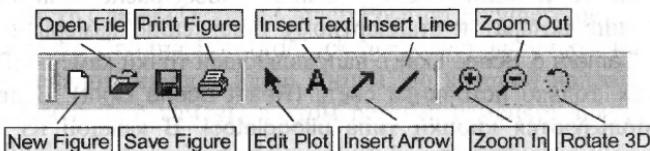
Меню *Tools*, представленное на рис. П-6.6, д, содержит команды, которые позволяют осуществлять интерактивное редактирование объектов, находящихся в графическом окне. Команда *Edit Plot* служит для включения и отключения режима редактирования. При включенном режиме редактирования напротив команды появляется «». При активизации режима редактирования можно выделять графические объекты и изменять их свойства с помощью команд контекстно-зависимого меню. Команды *Zoom In* и *Zoom Out* включают режимы интерактивного увеличения и уменьшения области координатного пространства соответственно. Двукратное увеличение или уменьшение области координатного пространства осуществляется с помощью щелчка основной (левой) кнопки мыши. Противоположное действие можно выполнить с помощью щелчка дополнительной кнопки мыши (правой). Двойной щелчок любой кнопкой мыши возвращает границы координатных осей в исходное состояние. Помимо двукратного увеличения можно увеличить произвольную область координатного пространства, охватив ее в прямоугольник выбора. Включить режим увеличения только по одной оси абсцисс или ординат для двухмерного графика можно с помощью задания в командном окне команд `>> zoom xon` или `>> zoom yon` соответственно. Команда *Rotate 3D* изменяет точку наблюдения, т.е. значения углов азимута и высоты. Во время изменения точки наблюдения в нижнем левом углу графического окна отображаются текущие значения этих углов. Команда *Move Camera* и следующая группа команд: *Camera Motion*, *Camera Axis* и *Camera Reset*, открывающих соответствующие подменю, предназначены для управления положением и направлением камеры. Определение координат точки наблюдения с помощью камеры снимает ограничения, которые присущи

стандартному способу с помощью двух углов. Использование камеры позволяет не только изменять углы наблюдения, но и управлять расстоянием от точки наблюдения до координатного пространства. Последние две команды – **Basic Fitting** и **Data Statistics** – открывают одноименные диалоговые окна для построения слаживающих кривых до 10-го порядка и отображения статистических данных (математического ожидания, дисперсии и т.д.) соответственно.

Команды меню **Window** (рис. П-6.6, е) позволяют переключаться между созданными графическими окнами (1 Figure No 1) и окном команд (0 MATLAB Command Window). В том случае, если число созданных графических окон превышает девять, то появляется дополнительный пункт меню **More Windows...**, с помощью которого можно вызвать диалоговое окно выбора графического окна (**Choose a window**).

Меню **Help** (рис. П-6.6, ж) содержит команды, обеспечивающие доступ к справочной информации по графическим возможностям системы MATLAB, демонстрационным примерам, а также по информационному диалоговому окну о системе MATLAB.

Стандартная панель инструментов, представленная на рис. П-6.7, содержит ряд разделенных на три группы кнопок, которые обеспечивают наряду с «горячими» клавишами быстрый доступ к соответствующим командам главного меню.



Р и с . П-6.7. Стандартная панель инструментов

В стандартной панели инструментов, являющейся объектом `uifToolbar`, размещаются кнопки двух типов – `uipushtool` и `uitoggletool`. Кнопки `uitoggletool` соответствуют тем командам меню, которые являются переключателями из одного состояния в другое, т.е. напротив них устанавливается или снимается «✓». Кнопками типа `uitoggletool` являются кнопки `Edit Plot`, `Insert Text`, `Insert Arrow`, `Insert Line`, `Zoom In`, `Zoom Out` и `Rotate 3D`. Утопленная кнопка обозначает включенный соответствующий режим. Так как в стандартной панели инструментов только одна кнопка может находиться в утопленном состоянии, то при переключении с одного режима работы графического окна на другой (в том числе и с помощью меню) кнопка, соответствующая старому режиму работы, переходит в обычное состояние, а кнопка, соответствующая новому режиму работы, – в утопленное состояние.

Панель инструментов «Камера», представленная на рис. П-6.8, позволяет выполнять ряд операций по изменению точки наблюдения в интерактивном режиме. Панель инструментов состоит из ряда кнопок, которые разделены на пять групп. К первой группе относится восемь кнопок, которые включают соответствующие режимы движения камеры (Orbit Camera) и источника света сцены (Orbit Scene Light). Те же самые функции доступны и с помощью команд меню Tools → Camera Motion. Второй группе принадлежат четыре кнопки, активизирующие соответствующую координатную ось, относительно которой будет происходить изменение положения камеры, либо сбрасывающие все ограничения движения камеры относительно осей. Включить режим работы камеры относительно одной оси или сбросить эти ограничения можно также с помощью команд, расположенных в меню Tools → Camera Axis. Третья группа состоит из одной кнопки Toggle Scene Light, которая включает или отключает источник света сцены. Четвертая группа, состоящая из двух кнопок – Orthographic Projection и Perspective Projection, позволяет установить соответствующий вид проекции текущего координатного пространства. Последняя группа содержит две кнопки – Reset Camera and Scene Light и Stop Camera/Light Motion. Нажатие на первую кнопку возвращает текущее координатное пространство к стандартному трехмерному виду, принятому в MATLAB. Нажатие на вторую кнопку останавливает движение камеры или источника света сцены. В меню Tools → Camera Reset располагаются команды, осуществляющие возврат к стандартному положению камеры и источника света (Reset Camera & Scene Light), направляющей точки камеры (Reset Target Point), а также только источника света (Reset Scene Light). В двух первых группах используются кнопки типа uitoggletool. В каждой из этих групп только одна кнопка может быть утопленной. В остальных группах используются кнопки типа uipushtool.

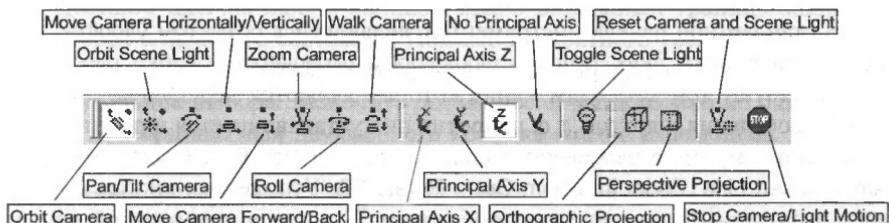


Рис. П-6.8. Панель инструментов «Камера»

Кнопка Orbit Camera предназначена для включения или отключения режима вращения камеры относительно целевой точки (CameraTarget), определяющей направление объектива камеры. Вращение камеры может быть

произвольным либо относительно одной из осей x , y или z . При вращении камеры изменяется значение свойства `CameraPosition` объекта `Axes`.

Кнопка `Orbit Scene Light` позволяет включать или отключать режим вращения источника света сцены, связанного с камерой. По умолчанию источник света сцены прикреплен к верхнему правому углу камеры. Изменение положения источника света сцены связано с изменением расстояния от источника до камеры. При движении источника света сцены другие источники света, созданные с помощью функции-конструктора `light()`, остаются неподвижными. Источники света могут применяться только к поверхностям или объектам типа `Patch`, увеличивая реалистичность изображения.

Кнопка `Pan/Tilt Camera` служит для включения или отключения режима изменения положения точки, на которую направлен объектив камеры. Положение камеры при этом остается прежним. Изменение положения целевой точки происходит с помощью вращения относительно центра камеры вокруг выбранной оси или произвольным образом. Свойство `CameraTarget` объекта `Axes` содержит значения координат точки, на которую направлен объектив камеры.

Кнопка `Move Camera Horizontally/Vertically` предназначена для включения или отключения режима горизонтального или/и вертикального перемещения сцены. Под сценой понимается текущее координатное пространство и камера с источником света сцены. При перемещении сцены изменяются значения свойств `CameraTarget` и `CameraPosition` текущего объекта `Axes`.

Кнопка `Move Camera Forward/Back` позволяет включать или отключать режим перемещения камеры вдоль линии, по которой направлен объектив камеры. Перемещение камеры в этом режиме изменяет значение свойства `CameraPosition`.

Кнопка `Zoom Camera` служит для включения и отключения режима изменения угла обзора текущего координатного пространства. Изменение угла обзора связано со свойством `CameraViewAngle` и не влияет на положение камеры. Значения угла обзора текущего координатного пространства задаются в градусах.

Кнопка `Roll Camera` предназначена для включения или отключения режима вращения камеры относительно оси, по которой направлен объектив камеры. При вращении камеры вокруг оси, определяющей направление объектива, изменяется значение свойства `CameraUpVector`.

Кнопка `Walk Camera` позволяет включать или отключать режим перемещения камеры вдоль выбранной координатной оси или оси, по которой направлен объектив камеры, с сохранением расстояния между камерой и точкой, на которую направлен ее объектив. При перемещении камеры изменяются свойства `CameraPosition` и `CameraTarget` текущего координатного пространства.

Упражнение 4. Создать в центре координатного пространства сферу единичного радиуса. Переместить камеру внутрь созданной сферы и просмотреть всю сферу изнутри.

В командном окне задать команду >> sphere(40). В появившемся графическом окне сделать видимой панель инструментов «Камера». Включить режим перемещения камеры вдоль оси, по которой направлен ее объектив, с одновременным перемещением направляющей точки. Переместить камеру внутрь сферы. Активизировать режим изменения угла обзора и охватить как можно больше просматриваемой области. Переключиться на режим изменения положения направляющей точки. Изменить положение направляющей точки таким образом, чтобы включить режим автоматического обновления положения направляющей точки. Нажать на основную кнопку мыши и, не отпуская ее, резко переместить курсор мыши в сторону желаемого направления вращения. При завершении перемещения отпустить кнопку мыши.

Помимо главного меню и двух панелей инструментов графическое окно включает ряд контекстно-зависимых меню, которые вызываются щелчком дополнительной кнопки мыши на выделенном объекте. С помощью команд контекстно-зависимых меню, представленных на рис. П-6.9, можно осуществлять операции с буфером обмена (Cut, Copy и Paste), удалять выбранный объект (Clear), изменять значения некоторых свойств объекта Line (Line Width, Line Style и Color), показывать или скрывать легенду координатного пространства (Show Legend или Hide Legend), включать или отключать режим свободного перемещения координатного пространства (Unlock Axes Position или Lock Axes Position), изменять размер (Font Size), начертание (Font Style), цвет (Color...) и объект Text (String...), а также вызывать редактор свойств (Properties...) с режимом редактирования текущего объекта.

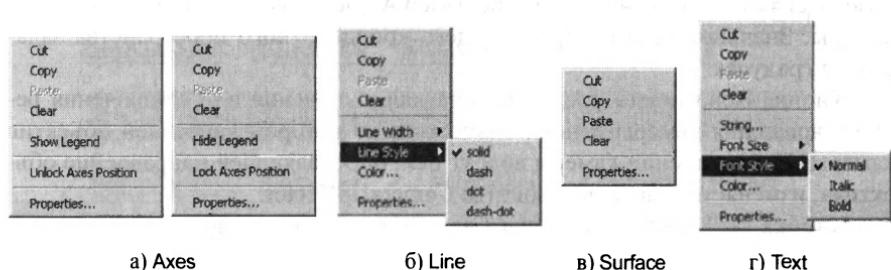


Рис. П-6.9. Контекстно-зависимые меню графического окна

Редактор свойств

Редактор свойств предназначен для быстрого изменения значений наиболее часто настраиваемых свойств текущего графического объекта с одновременным просмотром результатов изменений в графическом окне. Окно редактора свойств, представленное на рис. П-6.10, может быть вызвано различными способами:

- с помощью команды `>> propedit(имя_дескриптора);`
`>> x=[0:0.1:2*pi]; h=line(x,zeros(size(x)),cos(x)); propedit(h)`
- используя команды **Figure Properties...**, **Axes Properties...** и **Current Axes Properties...** меню **Edit**;
- двойным щелчком основной кнопкой мыши на графическом объекте в режиме редактирования;
- с помощью команды **Properties...** контекстно-зависимого меню текущего объекта в режиме редактирования.

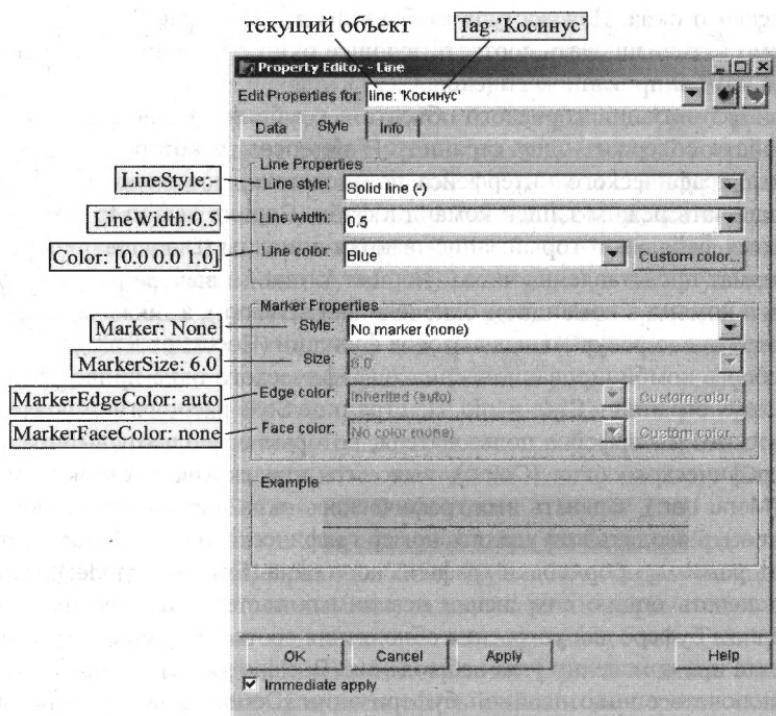


Рис. П-6.10. Окно редактора свойств

Окно редактора свойств состоит из комбинированного списка, нескольких страниц, переключение между которыми осуществляется с помощью щелчка на соответствующей вкладке, ряда кнопок для подтверждения или отмены сделанных изменений и вызова справочной информации, а также переключателя, позволяющего немедленно просматривать сделанные изменения.

Вид окна редактора свойств зависит от выбранного с помощью комбинированного списка *Edit Properties for:* графического объекта. В комбинированном списке имена объектов представлены с помощью древовидной структуры. Имя объекта состоит из названия объекта, за которым следует двоеточие и значения дескриптора объекта или значения свойства *Tag*. На самом верхнем уровне располагается корневой объект (*root: 0*). Под корневым объектом с отступом в один символ идут имена созданных графических окон (например, *figure: 1* или *figure:'Графическое окно'*). Так как только одно графическое окно может быть активным, то в списке с соответствующим количеством отступов отображаются имена объектов только текущего графического окна. Для доступа к объектам другого графического окна необходимо активизировать соответствующее окно с помощью щелчка на его имени в комбинированном списке.

При активизации корневого объекта в комбинированном списке редактор свойств содержит одну страницу *Preferences*, в которой размещаются элементы графического интерфейса пользователя, позволяющие: включать или отключать режим записи команд в файл (*Capture screen to diary file*), задавать имя файла, в который записываются команды (*Diary file name:*), выбирать формат представления чисел (*Number format:*) и вывода результатов выполнения команд в командном окне (*Number spacing:*), а также задавать максимальное число рекурсивных вызовов функций (*Recursion limit:*).

Выбор в комбинированном списке графического окна приводит к появлению двух страниц – *Style* и *Info*. На странице *Style* располагаются элементы графического интерфейса пользователя, которые позволяют: выбирать цвет фона графического окна (*Color:*), выводить или скрывать строку главного меню (*Menu bar:*), задавать имя графического окна в строке заголовка (*Window name:*), выводить или удалять номер графического окна (*Figure number:*), задавать режимы прорисовки графического окна (*Renderer mode:*), включать или отключать опцию сохранения невидимых частей графических объектов в отдельном буфере для ускорения обновления соответствующего графического объекта при изменении угла наблюдения (*Backing store:*), а также включать или выключать опцию двойной буферизации (*Double buffering:*) при анимации сцены в режиме прорисовки экрана *Painters*. В режиме *ZBuffer* всегда используется опция двойной буферизации, а в режиме *OpenGL*, напротив, эта опция никогда не используется. На странице *Info* располагаются элементы,

с помощью которых можно задавать значение свойству Tag, которое отображается в имени объекта в комбинированном списке, включать или выключать видимость графического окна (Visibility:), а также переключать окно редактора свойств в режим редактирования «родителя» (Edit parent...) или «потомков» (Edit Children...). В данном случае под «родителем» понимается корневой объект, а под «потомками» – все координатные пространства (в том числе и легенды). Вид страницы Info является стандартным для всех объектов, где она встречается.

При выборе из комбинированного списка объекта Axes окно редактора свойств будет содержать восемь страниц: X, Y, Z, Style, Aspect, Lights, Viewpoint и Info. На страницах X, Y и Z располагаются управляющие элементы, которые служат для независимого изменения свойств соответствующих координатных осей. Страница Style содержит элементы, позволяющие: изменять заголовок координатного пространства (Title:), выводить или убирать обрамляющий параллелепипед (Axes box), задавать положение координатной сетки и засечек относительно выведенных в координатное пространство объектов (Ticks and grids: Bottom or Top), цвет фона координатного пространства (Background:), толщину координатных осей и обрамляющего параллелепипеда (Axes line width:), а также атрибуты шрифта (Tick font attributes), который используется для обозначения значений координатных осей. На странице Aspect располагаются элементы, с помощью которых можно задавать масштаб координатных осей и ребер, обрамляющего параллелепипеда. Страница Lights предназначена для создания нового источника света (Add a new light), определения положения (X position, Y position и Z position) выделенного в списке Handle источника света, его цвета (Color:) и направления лучей (Infinite distance:), а также удаления выделенного источника света (Delete selected light). Управляющие элементы страницы Viewpoint позволяют задавать вид проекции (Projection: Orthographic or Perspective), положение точки наблюдения (View) в угловых координатах азимута (Azimuth) и высоты (Elevation), а также значения свойств, которые связаны камерой (Camera Properties). Существует возможность возврата к положениям точки наблюдения, заданным по умолчанию для двухмерного (Set default 2-D view) и трехмерного (Set default 3-D view) вида.

При активизации объекта Line окно редактора свойств, изображенное на рис. П-5.10, содержит три страницы: Data, Style и Info. С помощью элементов, расположенных на странице Data, задаются имена переменных, в которых содержатся значения линии по соответствующим осям. Размер переменных должна быть одинаковой. На странице Style находятся элементы, с помощью которых можно изменить стиль линии и маркеров в вычисленных точках (рис. П-5.10).

Выбор в комбинированном списке объекта Surface приводит к появлению в окне редактора свойств шести страниц: Data, Face, Marker/Edges,

Transparency, Lighting и **Info**. С помощью элементов первой страницы можно изменить имена переменных, в которых содержатся массивы значений координат узловых точек поверхности по соответствующим осям, а также задать цвет узловых точек. Переменные должны иметь одинаковую размерность. При задании цвета можно использовать трехмерный массив для цвета каждой точки либо установить цветовую палитру (**Colormaps:**) и включить опцию масштабирования цвета поверхности к цветам выбранной палитры (**Scale to figure colormap**). На странице **Face** располагаются элементы, которые позволяют изменить следующие свойства поверхности: цвет (**Color:**), освещенность (**Lighting:**) и прозрачность (**Alpha:**). В центральной части страницы располагаются пять кнопок, которые активизируют следующие виды поверхности: **Set wireframe (hide)** – каркасная модель поверхности с удаленными скрытыми линиями (после выполнения функций `mesh()` и `hidden('on')`); **Set wireframe** – каркасная модель поверхности с показом скрытых линий (после выполнения функций `mesh()` и `hidden('off')`); **Set point cloud (hide)** – поверхность изображается белым цветом с маркерами в виде точек в узловых точках поверхности (`FaceColor: [1.0, 1.0, 1.0]`, `LineStyle: 'none'` и `Marker: '.'`); **Set point cloud** – поверхность изображается только с помощью маркеров в местах узловых точек поверхности (свойствам `FaceColor` и `LineStyle` присвоено значение «`None`», а свойству `Marker` – «`.`») и **Set solid** – стандартный вид поверхности после выполнения функции `surf()`. Страница **Marker/Edges** содержит элементы управления, которые позволяют изменить тип (**Style:**), размер (**Size:**) и цвет (**EdgeColor:** и **FaceColor:**) маркеров, а также свойства линий (**Edge Properties**), соединяющих узловые точки поверхности. На странице **Transparency** располагаются элементы, которые позволяют изменять прозрачность поверхности. Элементы на странице **Lighting** служат для изменения освещенности поверхности.

При выборе объекта **Text** окно редактора свойств содержит три страницы: **Text**, **Style** и **Info**. С помощью элементов первой страницы можно вводить текст, а также выбирать шрифт (**Font name**) и задавать его параметры. Параметрами шрифта являются: насыщенность (**Font weight:**), угол наклона (**Font angle:**), размер (**Font size:**), единицы измерения размера (**Font units:**) и цвет (**Color:**). С помощью элементов, расположенных в верхней части страницы **Style**, можно задавать цвет фона текстового объекта (**Background color:**), цвет обрамляющего текстовый объект прямоугольника (**Edge color:**), стиль линии прямоугольника (**Line style:**), ее ширину (**Line width:**) и расстояние до текста (**Margin:**). Элементы, расположенные в нижней части страницы **Style**, позволяют включать или отключать интерпретатор издательской системы **LaTeX** (**Use LaTeX interpreter**), размещать текстовый объект относительно координатного пространства (**Position:**), вращать текстовый объект (**Rotation:**), а также выравнивать его по горизонтали (**Horizontal:**) и по вертикали (**Vertical:**).

Упражнение 5. Создать в графическом окне поверхность, представляющую собой функцию двух переменных. С помощью редактора свойств изменить внешний вид поверхности и заголовка координатного пространства.

В командном окне задать команду `>> peaks(40)`. Активизировать редактор свойств и сделать текущим объект **Surface**. Перейти на страницу **Face**. С помощью присутствующих на странице **Face** элементов изменить внешний вид поверхности. Перейти на страницу **Marker/Edges** и с помощью элементов, расположенных на этой странице, изменить внешний вид поверхности. После каждого изменения просмотреть результаты в графическом окне. Перейти на страницу **Info** и ввести описание поверхности **Peaks(40)**. С помощью комбинированного списка активизировать объект **Axes**. Перейти на страницу **Style** и нажать на кнопку **Properties...**, расположенную напротив текстового поля **Title**. Редактор свойств переключится в режим редактирования текстового объекта. С помощью элементов на странице **Style** создать рамку черного цвета толщиной 2 пункта вокруг текста и отстоящую от него на 10 точек.

Самостоятельная работа

Самостоятельная работа состоит из двух заданий. При выполнении первого задания создать две координатные плоскости. В нечетных вариантах плоскости располагаются вертикально, в четных – горизонтально. В первой плоскости определить прямоугольную систему координат, а во второй – полярную. Построить заданную линию в двух системах координат, а также вывести координатные оси. Обозначить с помощью кругового маркера и текстового объекта начало координат. Изменить свойства всех созданных графических объектов (текстовых обозначений, линий, координатных плоскостей или графических окон) с помощью редактора свойств. В строке заголовка графического окна убрать его номер и вывести номер варианта и название заданной линии (например, Вариант 1: *Лемниската Бернулли*). Вызвести в графическое окно легенду и изменить цвет линии. Исследовать влияние значений переменных, входящих в уравнение линии, на вид линии.

При выполнении второго задания свойства объектов графического окна задаются с помощью команд, вводимых в командном окне, а наиболее подходящая точка обзора задается с помощью инструментов панели **Camera**. Вызвести цветовую шкалу в графическое окно и координатные оси внутри координатного пространства с помощью прямых линий синего цвета толщиной 1 пункт. При построении поверхности скрыть линии, соединяющие узловые точки поверхности, и задать плавный переход между цветами палитры. Фон координатного пространства совпадает с фоном графического окна. Значения

вычисленных параметров вывести в заголовке координатного пространства, используя функцию `num2str()`.

При обозначении координатных осей и заголовка координатного пространства использовать команды системы верстки LaTeX.

Исследовать влияния коэффициентов на вид поверхности с помощью инструментального средства *Command History*, а цветовой палитры на цвета поверхности – с помощью редактора цветовой палитры (*Colormap Editor*).

Примечание. При выполнении поворота вокруг соответствующих осей и перемещении поверхности в пространстве необходимо надлежащим образом использовать следующие команды:

```
XYZ(:,:,1) = X + x0; XYZ(:,:,2) = Y + y0; XYZ(:,:,3) = Z + z0;  
XYZ = permute(XYZ,[3 2 1]);  
for page=1:size(XYZ,3)  
    XYZ(:,:,:,page) = Rx*Ry*XYZ(:,:,:,page);  
    XYZ(:,:,:,page) = Rz*XYZ(:,:,:,page); % поворот вокруг оси вращения  
end  
XYZ = ipermute(XYZ,[3 2 1]);  
X = XYZ(:,:,:1); Y = XYZ(:,:,:2); Z = XYZ(:,:,:3);
```

В приведенных командах использованы следующие обозначения: x_0, y_0, z_0 – координаты центра поверхности; Rx, Ry, Rz – матрицы поворота относительно соответствующих осей. Функции `permute()` и `ipermute()` использовались для прямого и соответственно обратного изменения последовательности следования строк, столбцов и страниц в трехмерном массиве `XYZ`.

Вариант 1

1. Построить лемнискату Бернулли, параметрические уравнения которой имеют вид $x = a \cdot (t + t^3)/(t^4 + 1)$, $y = a \cdot (t - t^3)/(t^4 + 1)$, где $t \in [-\infty; \infty]$.
2. Найти координаты точки пересечения P прямой линии L и плоскости Q . Параметрическое уравнение прямой линии L : $x = 2t + 3$, $y = t - 2$, $z = t + 3$. Параметр t изменяется в диапазоне $[0; 2]$ с шагом 0.1. Общее уравнение плоскости Q : $2x - 6y + 13z + 1 = 0$. Изобразить в графическом окне плоскость Q , прямую линию L синего цвета толщиной 2 пункта и точку пересечения P в виде кругового маркера красного цвета размером 6 пунктов. Выбрать стандартную палитру `gray`. Исследовать влияние коэффициентов общего уравнения и значений координат направляющего вектора линии на значения координат точки пересечения.

Вариант 2

1. Построить четырехлепестковую розу, заданную уравнением в полярных координатах: $r = a \sin 2\varphi$, где $\varphi \in [0; 2\pi]$.

2. Построить цилиндрическую поверхность, направляющей которой является *трехлепестковая роза* ($r = a \sin 3\varphi$). Длина образующей поверхности равна 2. Образующая параллельна оси z . Центр цилиндрической поверхности находится в точке с координатами $(0.5; 0.5; -0.25)$. Повернуть поверхность относительно оси y на 45° . Задать стандартную цветовую палитру *summer*. Исследовать влияние значений коэффициента a , диапазона изменения угла φ , а также цветовой палитры на вид поверхности.

Вариант 3

- Построить *улитку Паскаля*, уравнение которой в полярной системе координат имеет вид $r = b + a \cos \varphi$.
- Определить угол между двумя прямыми линиями — L_1 и L_2 . Прямая линия L_1 задана с помощью общих уравнений: $x - y + 2z - 3 = 0$ и $2x + y - z + 2 = 0$, где значения координат x и y лежат в диапазоне $[-2; 2]$. Прямая линия L_2 определяется параметрическими уравнениями: $x = 2 - t$, $y = -t$ и $z = -1 - 2t$, где $t \in [0; 2]$. В графическом окне изобразить две линии, а также их направляющие векторы единичной длины. Линию L_1 изобразить с помощью двух плоскостей — Q_1 и Q_2 . Место пересечения обозначить красной линией. Цвет одной плоскости — синий, а для другой использовать цвета палитры *summer*. Линию L_2 изобразить черным цветом. Толщина этой линии составляет 2 пункта. Исследовать влияние коэффициентов общих уравнений прямой линии L_1 и координат направляющего вектора прямой линии L_2 на угол между двумя прямыми.

Вариант 4

- Построить *полукубическую параболу* (*параболу Нейля*), параметрические уравнения которой: $x = t^2$, $y = t^3$.
- Изобразить в координатном пространстве три точки: $A(-1; 2; 1)$, $B(-2; -3; 1.5)$, $C(2; 1; -1)$ и плоскость Q , заданную уравнением в отрезках: $x/2 + y/3 - z = 1$. Значения координат x и y узловых точек плоскости располагаются в диапазоне $[-3; 3]$. Вычислить расстояния между геометрическими объектами. Определить точку, которая ближе всех расположена к плоскости Q , и обозначить ее с помощью кругового маркера красного цвета. Остальные точки обозначить квадратными маркерами черного цвета. Окрасить плоскость в цвета палитры *hot*. Исследовать влияние значений коэффициентов уравнения на положение плоскости в пространстве и на расстояния до точек.

Вариант 5

- Построить *спираль Архимеда*, уравнение которой в полярных координатах: $r = a\varphi$.

2. Построить в координатном пространстве две плоскости – Q_1 и Q_2 . Плоскость Q_1 задана с помощью общего уравнения $x - y - z - 0.75 = 0$, а плоскость Q_2 – с помощью точки $M(1;2;-1)$ и вектора $V = (1;-1;1.5)$, который перпендикулярен плоскости. Окрасить плоскость Q_2 в зеленый цвет, а плоскость Q_1 – в цвета палитры cool. Определить угол между плоскостями. Отобразить в центре координатного пространства нормали к поверхностям. Исследовать влияние коэффициентов уравнений на положение соответствующей плоскости и на значение угла между плоскостями.

Вариант 6

1. Построить конхоиду Никомеда, уравнение которой в полярных координатах: $r = a / \cos \varphi \pm d$. Угол φ изменяется в диапазоне $[-1.25; 1.25]$ с шагом 0.1.
2. Построить в координатном пространстве прямую линию L , черного цвета и толщиной 2 пункта, и плоскость Q , используя цветовую палитру winter. Линия L задана параметрическими уравнениями: $x = 2 - 5t$, $y = -0.5 + 2t$, $z = 1 - 1.5t$, где $t \in [-3; 3]$. Плоскость Q задана с помощью точки $M(-1; 2; 2)$ и вектора $V = (1; -2; 1.5)$, который перпендикулярен плоскости. Построить направляющий вектор прямой линии и вектор нормали к плоскости, используя красный и зеленый цвет соответственно. Исследовать влияние значений коэффициентов уравнений и координат вектора нормали к плоскости Q на положение соответствующих геометрических объектов и на значение угла между ними.

Вариант 7

1. Построить инволюту окружности, уравнения которой в прямоугольных координатах: $x = r \cos \theta + r \theta \sin \theta$, $y = r \sin \theta - r \theta \cos \theta$, где r – радиус окружности, θ – угол $\in [0; 2\pi]$.
2. Построить цилиндрическую поверхность, направляющей которой является улитка Паскаля ($r = b + a \cos \varphi$, где $b = a = 1$, $\varphi \in [0; 2\pi]$). Длина образующей поверхности равна 4. Направляющая строится в плоскости Oxy . Поверхность повернута относительно оси z на 45° , а относительно оси x – на 30° . Задать стандартную цветовую палитру bone. Исследовать влияние значений коэффициента a , диапазона изменения угла φ и цветовой палитры на вид поверхности.

Вариант 8

1. Построить кохлеоиду, уравнение которой в полярных координатах: $r = a \sin \varphi / \varphi$, где $\varphi \in [0; \infty)$.
2. Построить поверхность вращения, образованную с помощью вращения правой части нефроиды, определенной в плоскости Oyz (параметрические уравнения линии): $x = 0$, $y = 3a \cos t - a \cos 3t$, $z = 3a \sin t - a \sin 3t$, где $a = 1$.

и $t \in [-\pi/2; \pi/2]$), вокруг оси z . Повернуть поверхность относительно оси x на 30° . Задать цветовую палитру `hot`. Исследовать влияние значений коэффициента a , значений параметра t и цветовой палитры на вид поверхности.

Вариант 9

- Построить локон Аньези. Уравнение в прямоугольных координатах: $y = a^3 / (a^2 + x^2)$.
- Построить в координатном пространстве эллипсоид, со значениями полуосей $a = 1, b = 2, c = 0.5$. Повернуть эллипсоид относительно оси z на 45° , а относительно оси x – на 30° . Переместить центр эллипса в точку $O(0.5; -0.25; 0.25)$. Задать цветовую палитру `copper`. Исследовать влияние значений полуосей и цветовой палитры на вид эллипса.

Вариант 10

- Построить цепную линию, заданную уравнением в прямоугольных координатах: $y = a \operatorname{ch}(x/a) = a(e^{x/a} + e^{-x/a})/2$.
- Построить однополостный гиперболоид с мнимой осью $a = 2$ и действительными полуосами $b = 1.5$ и $c = 2$. Высота однополостного гиперболоида $h = 3$. Центр поверхности расположен в точке $O(0.25; -0.25; 0.5)$. Повернуть поверхность относительно оси u на 45° . Задать цветовую палитру `prism`. Исследовать влияние значений полуосей, угла поворота относительно оси u и цветовой палитры на вид поверхности в координатном пространстве.

Вариант 11

- Построить гиперболическую спираль, заданную уравнением в полярных координатах: $r = a/\theta$.
- Построить двуполостный гиперболоид, заданный с помощью канонического уравнения $x^2/a^2 - y^2/b^2 + z^2/c^2 = -1$. Величины $a = 1, b = 2, c = 1$ и $h \in [-2-b; -b] \cup [b; b+2]$. Повернуть поверхность относительно оси x на 45° . Задать цветовую палитру `colordcube`. Исследовать влияние значений величин a, b и c , угла поворота относительно оси x и цветовой палитры на вид поверхности в координатном пространстве.

Вариант 12

- Построить трактису, заданную с помощью параметрических уравнений: $x = a(\cos t + \ln \operatorname{tg}(t/2)), y = a \sin t, t \in (0; \infty)$.
- Построить эллиптический параболоид, заданный с помощью канонического уравнения $y^2/p + z^2/q = 2x$, где $p = 3$ и $q = 1$. Высота параболоида $h = 3$. Повернуть поверхность относительно оси z на 60° . Задать цветовую палитру

pink. Исследовать влияние значений коэффициентов p и q , а также высоты параболоида и цветовой палитры на вид поверхности.

Вариант 13

1. Построить спираль Кейли. Уравнение в полярных координатах имеет вид $r = a / \cos^3(\varphi / 3)$.
2. Построить гиперболический параболоид, заданный с помощью канонического уравнения $y^2/p - z^2/q = 2x$, где $p = 1.5$ и $q = 3$. Высота параболоида $h = 8$. Задать цветовую палитру hsv и ориентировать цвета поверхности по оси x . Исследовать влияние значений коэффициентов p и q , а также высоты параболоида и цветовой палитры на вид поверхности.

Вариант 14

1. Построить логарифмическую спираль, уравнение которой в полярных координатах: $r = ae^{b\cdot\varphi}$.
2. Построить конус второго порядка, заданный с помощью канонического уравнения $x^2/a^2 - y^2/b^2 + z^2/c^2 = 0$, где $a = 1$, $b = 2$, $c = 1.5$. Высота конуса $h = 5$. Задать цветовую палитру jet и ориентировать цвета поверхности по оси y . Повернуть поверхность относительно оси x на 45° . Исследовать влияние значений коэффициентов уравнения и палитры цветов на вид поверхности.

Вариант 15

1. Построить дельтоид, заданный с помощью параметрических уравнений $x = 2a \cos t + a \cos 2t$, $y = 2a \sin t - a \sin 2t$, $t \in [0; 2\pi]$.
2. Построить поверхность вращения, образованную с помощью вращения линии L вокруг оси y . Линия задана в плоскости Oxy с помощью уравнения в полярных координатах: $r = a \cos 2\varphi$, где $a = 1$ и $\varphi \in [-\pi/2; \pi/2]$. Повернуть поверхность оси z на 45° . Задать цветовую палитру colorcube. Исследовать влияние значений коэффициента a , диапазона изменения полярного угла φ и цветовой палитры на вид поверхности.

Вариант 16

1. Построить нефроиду, заданную с помощью параметрических уравнений $x = 3a \cos t - a \cos 3t$, $y = 3a \sin t - a \sin 3t$, $t \in [0; 2\pi]$.
2. Построить однополостный гиперболоид, заданный с помощью канонического уравнения $x^2/a^2 - y^2/b^2 + z^2/c^2 = 1$, где $a = 2$, $b = 1.5$ и $c = 3$. Высота гиперболоида $h = 5$. Повернуть поверхность относительно оси z на 30° . Задать цветовую палитру autumn. Ориентировать цвета поверхности по оси y . Исследовать влияние значений коэффициентов уравнения и палитры цветов на вид поверхности.

Вариант 17

- Построить строфоиду, заданную с помощью параметрических уравнений $x=2at^2/(t^2+1)$, $y=at\cdot(t^2-1)/(t^2+1)$, $t \in [-2; 2]$, $a > 0$.
- Построить двуполостный гиперболоид, заданный с помощью канонического уравнения $-x^2/a^2 + y^2/b^2 + z^2/c^2 = -1$, где $a = 1.5$, $b = 2.5$, $c = 1$ и $h \in [-3-a; -a] \cup [a; a+3]$. Повернуть поверхность относительно оси z на угол 60° . Задать цветовую палитру `flag`. Ориентировать цвета поверхности по оси x . Исследовать влияние значений коэффициентов уравнения и палитры цветов на вид поверхности.

Вариант 18

- Построить циссоиду Диоклеса, уравнение которой в полярной системе координат: $r = 2a \sin^2 \varphi / \cos \varphi$, $\varphi \in [-\pi/4; \pi/4]$.
- Построить эллиптический параболоид, заданный с помощью канонического уравнения $x^2/p + z^2/q = 2y$, где $p = 1$ и $q = 2$. Высота параболоида $h = 4$. Повернуть поверхность относительно оси x на 30° . Задать цветовую палитру `bone`. Ориентировать цвета поверхности по оси y . Исследовать влияние значений коэффициентов p и q , а также высоты параболоида и цветовой палитры на вид поверхности.

Вариант 19

- Построить спираль Ферма, заданную с помощью уравнения в полярных координатах $r = a\sqrt{\varphi}$, $\varphi \in [0; 4\pi]$.
- Построить гиперболический параболоид, заданный с помощью канонического уравнения $-x^2/p + z^2/q = 2y$, где $p = 3$ и $q = 2$. Высота параболоида $h = 10$. Задать цветовую палитру `winter` и ориентировать цвета поверхности по оси y . Исследовать влияние значений коэффициентов p и q , а также высоты параболоида и цветовой палитры на вид поверхности.

Вариант 20

- Построить спираль Галилея, уравнение в полярных координатах которой $r = a \cdot \varphi^2$, $\varphi \in [0; 2\pi]$.
- Построить конус второго порядка, заданный с помощью канонического уравнения $-x^2/a^2 + y^2/b^2 + z^2/c^2 = 0$, где $a = 2$, $b = 1.5$, $c = 3$. Высота конуса $h = 4$. Задать цветовую палитру `spring` и ориентировать цвета поверхности по оси x . Повернуть поверхность относительно оси z на 60° . Исследовать влияние значений коэффициентов уравнения и палитры цветов на вид поверхности.

Вариант 21

1. Построить лист щавеля, заданный с помощью уравнения в полярных координатах $r = 4(1 + \cos 3\phi) + 4\sin^2 3\phi$, $\phi \in [0; 2\pi]$.
2. Построить в координатном пространстве эллипсоид, заданный с помощью канонического уравнения $x^2/a^2 + y^2/b^2 + z^2/c^2 = 1$, где $a = 3$, $b = 1.5$, $c = 1$. Повернуть поверхность относительно оси y на 25° . Задать стандартную цветовую палитру `cool` и ориентировать цвета по оси y . Переместить центр эллипса в точку $O(1; -1; 0)$. Исследовать влияние значений полусей и цветовой палитры на вид эллипса.

Вариант 22

1. Построить Декартов лист, заданный с помощью параметрических уравнений $x = 3at^2/(t^3 + 1)$, $y = 3at^2/(t^3 + 1)$, $t \in [-6\pi; -2] \cup [-0.3; 6\pi]$.
2. Определить взаимное расположение прямых линий L_1 и L_2 в пространстве. Прямая линия L_1 задана с помощью общих уравнений $x - y + 3z - 1 = 0$ и $-x - y - 2z + 0.5 = 0$. Прямая линия L_2 задана с помощью параметрических уравнений $x = 2.5 - 2.5 \cdot t$, $y = 1.5 + 0.5 \cdot t$ и $z = t$, где $t \in [0; 2]$. Одну плоскость окрасить в зеленый цвет, а другую в цвета палитры `cool`. Прямую линию L_1 обозначить черным цветом и толщиной 2 пункта. Прямую линию L_2 вывести красным цветом и толщиной 2 пункта. Исследовать влияние коэффициентов уравнений на взаимное расположение прямых линий.

Вариант 23

1. Построить трехлепестковую розу, уравнение которой в полярных координатах $r = a \sin 3\phi$, $\phi \in [0; 2\pi]$.
2. Построить цилиндрическую поверхность, направляющей которой является дельтоид ($x = 2a \cos t + a \cos 2t$, $y = 2a \sin t - a \sin 2t$, где $a = 1$, $t \in [-0; 2\pi]$). Длина образующей поверхности равна 5. Повернуть поверхность относительно оси x на 30° . Задать цветовую палитру `copper` и ориентировать цвета поверхности по оси x . Исследовать влияние значений коэффициента a , параметра t и палитры цветов на вид поверхности.

Вариант 24

1. Построить трисектрису Маклорена, заданную с помощью уравнения в полярных координатах $r = a / \cos(\phi/3)$, $\phi = (-3\pi/2; 3\pi/2)$.
2. Построить в координатном пространстве куб, вершины которого располагаются в точках $V_1(1; 1; -1)$, $V_2(1; -1; -1)$, $V_3(-1; 1; 1)$, $V_4(1; 1; 1)$, $V_5(-1; 1; -1)$, $V_6(-1; -1; -1)$, $V_7(-1; -1; 1)$ и $V_8(-1; 1; 1)$. Грань 1, образованную последовательным соединением вершин V_1 , V_2 , V_3 и V_4 , окрасить в красный цвет. Грань 2 (V_5 , V_1 , V_4 , V_8) окрасить в зеленый цвет, грань 3 (V_5 , V_6 , V_7 , V_8) –

в синий цвет, грань 4 (V_6, V_2, V_3, V_7) – в желтый цвет, грань 5 (V_8, V_4, V_3, V_7) – в голубой цвет и грань 6 (V_5, V_1, V_2, V_6) – в малиновый цвет. Повернуть куб относительно оси x на 30° , относительно оси y – на 45° и относительно оси z – на 60° . Исследовать влияние значений углов поворота на положение куба в пространстве.

Вариант 25

- Построить *трилистник*, заданный с помощью уравнения в полярных координатах $r = 4(1 + \cos 3\varphi) - 4 \sin^2 3\varphi$, $\varphi \in [0; 2\pi]$.
- Построить тор, у которого радиус врачающегося круга $r = 1$, а расстояние от центра круга до оси вращения $a = 3$. Ось вращения совпадает с осью x . Переместить тор на расстояние $d = 2$ по направлению $n = (1; -1; 1)$. Повернуть тор относительно оси z на 30° . Задать цветовую палитру *autumn* и ориентировать цвета палитры по оси y . Исследовать влияние значений r и a , а также цветовой палитры на вид тора.

Контрольные вопросы

При ответе на вопрос привести пример, используя систему MATLAB.

- Координатные системы на плоскости.
- Линии первого порядка. Общее уравнение прямой линии.
- Уравнение прямой линии с угловым коэффициентом.
- Уравнение прямой линии, проходящей через данную точку в заданном направлении.
- Уравнение прямой линии, проходящей через две точки.
- Уравнение прямой линии, проходящей через заданную точку перпендикулярно заданному вектору.
- Полярное уравнение прямой линии.
- Параметрический способ задания прямой линии на плоскости.
- Линии второго порядка. Каноническое уравнение окружности.
- Линии второго порядка. Каноническое уравнение эллипса.
- Линии второго порядка. Каноническое уравнение гиперболы.
- Линии второго порядка. Каноническое уравнение параболы.
- Системы координат в пространстве. Преобразование координат между прямоугольной, цилиндрической и сферической системой координат.
- Трансформация системы координат в пространстве. Матрицы поворота системы координат.
- Общее уравнение плоскости.
- Уравнение плоскости, проходящей через три заданные точки.
- Уравнение плоскости в отрезках.
- Уравнение плоскости, проходящей через точку перпендикулярно данному вектору.

19. Общие уравнения прямой линии.
20. Параметрические уравнения прямой линии.
21. Канонические уравнения прямой линии.
22. Уравнение прямой линии в пространстве, проходящей через две точки.
23. Определение угла между двумя плоскостями.
24. Вычисление расстояния от точки до плоскости.
25. Определение угла между прямыми линиями.
26. Определение взаимного расположения прямых линий в пространстве.
27. Вычисление угла между прямой линией и плоскостью.
28. Определение точки пересечения прямой линии и плоскости.
29. Цилиндрические поверхности.
30. Поверхности вращения.
31. Поверхности второго порядка. Каноническое уравнение эллипсоида.
32. Поверхности второго порядка. Каноническое уравнение однополостного гиперболоида.
33. Поверхности второго порядка. Каноническое уравнение двуполостного гиперболоида.
34. Поверхности второго порядка. Каноническое уравнение эллиптического параболоида.
35. Поверхности второго порядка. Каноническое уравнение гиперболического параболоида.
36. Поверхности второго порядка. Конус второго порядка.
37. Объекты дескрипторной графики. Функции-конструкторы графических объектов.
38. Последовательность задания значений свойствам графических объектов.
39. Назначение графического окна. Главное меню.
40. Назначение графического окна. Панели инструментов.
41. Назначение редактора свойств.
42. Функции `set()`, `get()` и `findobj()`.
43. Создание текстового объекта. Формирование команды системы LaTeX.
44. Функции построения линий на координатной плоскости.
45. Варианты выравнивания текстового объекта.
46. Функции построения поверхности в координатном пространстве.
47. Стандартные палитры цветов графического окна.
48. Задание прозрачности. Какие объекты могут быть полупрозрачными?
49. Создание координатного пространства. Обозначение координатных осей и координатного пространства. Вывод координатной сетки и включение режима добавления объектов.
50. Режимы прорисовки графического окна.

Практикум 8. ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО EDITOR/DEBUGGER

Цель: Знакомство с инструментальным средством Editor/Debugger, а также приобретение навыков работы со структурами выбора и повторения языка MATLAB.

Система MATLAB включает инструментальное средство Editor/Debugger (Редактор), которое применяется для выполнения операций по созданию, редактированию и отладке m-файлов. Редактор обладает развитыми средствами графического интерфейса пользователя, которые позволяют сократить время на создание окончательной версии работающей программы.

Редактор может работать в двух режимах: в режиме создания и редактирования текстовых файлов, а также в режиме создания и редактирования m-файлов с возможностью их отладки. Для активизации Редактора в первом режиме работы его необходимо запустить как автономное приложение. Работа с Редактором в этом режиме мало чем отличается от работы со стандартными текстовыми редакторами (например, Блокнот). Так как в этом режиме работы нельзя проверить работоспособность созданной на языке MATLAB программы, то он практически не используется при разработке m-файлов. Редактор во втором режиме работы можно запустить только из системы MATLAB. Возможные способы активизации Редактора из системы MATLAB приведены в табл. П-8.1.

Таблица П-8.1

Возможные варианты активизации Редактора из MATLAB

Создание нового m-файла	Открытие уже существующего m-файла
Используя главное меню рабочего стола системы MATLAB: File → New → M-File	Используя главное меню рабочего стола системы MATLAB: File → Open... В появившемся диалоговом окне выбрать необходимый файл с расширением m
С помощью кнопки New M-File , расположенной на стандартной панели инструментов рабочего стола системы MATLAB	С помощью кнопки Open File , находящейся на стандартной панели инструментов рабочего стола системы MATLAB
С помощью стандартной функции edit() , вызываемой из командной строки. Например, edit	С помощью стандартной функции edit() , вызываемой из командной строки с указанием имени файла. Например, edit example 6 4
Используя команду Create M-File контекстно-зависимого меню Command History	Используя команду Open контекстно- зависимого меню инструментального средства Current Directory

На рис. П-8.1 представлен Редактор в режиме создания и редактирования m-файлов с возможностью их отладки, в котором открыт файл example_6_4.

```
% Программирование для ввода массива положительных чисел
1 answer = input('Введите массив положительных чисел: ');
2
3 tic % измерение таймера
4 a = isprime(answer(:)); % логический массив
5 if any(a) % проверка введенного массива
6 s = sum(a); % подсчет суммы элементов
7 % формирование строки сообщения
8 string = ['Вы ввели ' num2str(s) ' простых чисел:'];
9 disp(string) % вывод строки сообщения
10 answer(a') % вывод простых чисел
11 else
12 % формирование строки сообщения
13 string = ['Введенный массив не содержит простых чисел'];
14 disp(string) % вывод строки сообщения
15 end % окончание структуры импорта
16 toc % измерение таймера и вывод вычислительного времени
```

Рис. П-8.1. Инструментальное средство Editor/Debugger

Редактор состоит из строки заголовка, строки главного меню, панели инструментов, рабочей области, в которой располагается текст файла, поля номеров строк, поля контрольных точек (точек останова) и строки состояния.

В строке заголовка отображается полное имя файла с учетом пути доступа.

В строке главного меню расположены девять пунктов меню, при активизации которых появляется соответствующее всплывающее меню. Набор команд в меню File, View, Web, Window и Help совпадает с одноименными меню рабочего стола MATLAB. Остальные выпадающие меню изображены на рис. П-8.2.

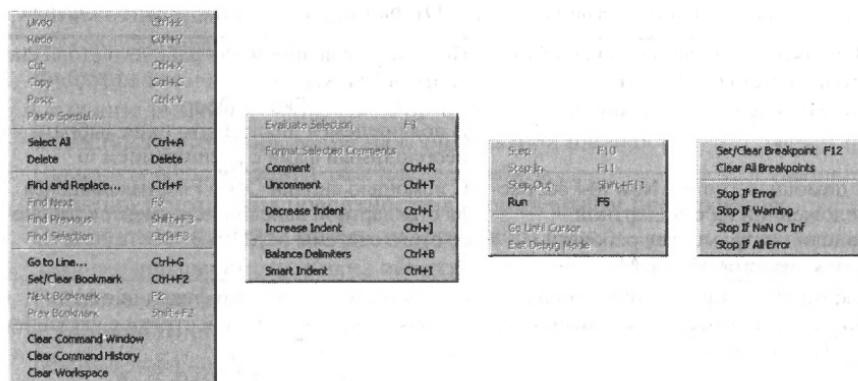


Рис. П-8.2. Всплывающие меню Editor/Debugger

Меню **Edit** содержит команды, которые разделены на шесть групп. Команды первой группы – **Undo** и **Redo** – используются для последовательной отмены или восстановления ранее выполненных действий соответственно. Команды второй группы применяются при работе с буфером обмена и позволяют вырезать (**Cut**) или копировать (**Copy**) выделенную часть текста в буфер, вставлять (**Paste**) содержимое буфера в текущую позицию, на которую указывает текстовый курсор, а также импортировать данные (**Paste Special...**) из буфера обмена согласно выбранному формату. Третью группу образуют две команды – **Select All** и **Delete**. Первая команда этой группы используется для выделения всего содержимого рабочей области, а вторая – для удаления выделенного текста или текущего символа, на который указывает текстовый курсор, в случае отсутствия выделения. Четвертая группа команд (**Find and Replace...**, **Find Next**, **Find Previous** и **Find Selection**) применяется для вызова диалогового окна **Find&Replace**, а также поиска в файле выделенного текста или указанного в нем фрагмента текста в обоих направлениях. Пятая группа команд применяется для быстрого перемещения по тексту. Первая команда в этой группе **Go to Line...** вызывает одноименное диалоговое окно, где указывается номер строки, в начало которой необходимо переместить текстовый курсор. Следующая команда **Set/Clear Bookmark** позволяет установить или сбросить закладку. Закладка представляет собой указатель на строку и отображается в виде прямоугольника напротив строки. Следующие две команды (**Next Bookmark** и **Previous Bookmark**) применяются для перемещения между закладками в направлении конца и начала файла соответственно. Заключительная группа команд совпадает с соответствующей группой команд в меню **Edit** рабочего стола системы MATLAB.

Меню **Text** используется для работы с текстом и включает команды, разделенные на четыре группы. Первая группа состоит из одной команды (**Evaluate Selection**), которая передает выделенный фрагмент текста ядру MATLAB для вычисления результата. Вторая группа команд предназначена для работы с комментариями. Команда **Format Selected Comment** форматирует выделенные комментарии согласно заданной максимальной длине строки. Если комментарий превышает эту длину, то он переносится на новую строку с добавлением символа % в ее начало. Команда **Comment** используется для вставки символа % в начало выделенных строк для их комментирования. При выполнении команды **Uncomment** происходит удаление одного символа % из выделенных строк. Третья группа команд позволяет увеличивать (**Increase Indent**) или уменьшать (**Decrease Indent**) отступ в выделенных строках. Использование отступов улучшает читабельность программы. Если отсутствуют выделенные строки, то команды **Comment**, **Uncomment**, **Increase Indent** и **Decrease Indent** применяются к текущей строке, в которой расположен текстовый курсор. Последняя группа состоит из двух команд – **Balance Delimiters** (Симметричные разделители) и **Smart Indent** (Автоматический отступ). Первая команда применяется для выделения

текста внутри симметричных разделителей ([], (), {}), между которыми располагается курсор. В том случае, если одного из разделителей нет, то раздается звуковой сигнал. Вторая команда применяется для автоматического задания отступов внутри выделенного текста.

Меню Debug используется при отладке программы и состоит из двух групп команд. Первую группу образуют четыре команды: Step, Step In, Step Out и Run (Continue). Первые три команды активизируются после запуска текущего файла на исполнение и доступны только в режиме отладки. При задании первой команды выполняются инструкции, находящиеся в текущей строке. Текущая строка во время отладки обозначается стрелкой зеленого цвета. После выполнения всех инструкций стрелка смещается на следующую исполняемую строку. Исполняемыми строками называются строки, содержащие инструкции языка MATLAB. Исполняемые строки обозначаются знаком «→» в поле контрольных точек. Кроме исполняемых строк в файле могут присутствовать пустые строки, а также строки, содержащие комментарии и заголовки функций. Отличием второй команды от первой является то, что при выполнении инструкций происходит передача управления в вызываемую функцию или сценарий. При выполнении перехода в вызываемую функцию или сценарий напротив строки, где расположен вызов, появляется белая стрелка, которая обозначает текущую строку вызывающей функции или сценария. Третья команда служит для возврата в точку вызова с полным выполнением оставшихся инструкций текущей функции или сценария. После возврата в вызывающий модуль управление передается на следующую инструкцию. Если эта инструкция находится на следующей строке, то стрелка автоматически смещается на нее. Четвертая команда запускает текущий файл на исполнение и используется в режиме отладки для продолжения выполнения инструкций до ближайшей контрольной точки. Вторая группа доступна только в режиме отладки и состоит из двух команд – Go Until Cursor и Exit Debug Mode. Первая команда применяется для продолжения выполнения инструкций до строки, в которойложен текстовый курсор. Вторая команда используется для выхода из режима отладки и перехода в режим редактирования.

Меню Breakpoint используется для работы с контрольными точками и состоит из двух групп. В первую группу входят команды Set/Clear Breakpoint и Clear All Breakpoints. Первая команда применяется для установки и снятия контрольной точки в текущей строке, вторая команда удаляет все контрольные точки, установленные в файле. Контрольные точки применяются для перехода в режим отладки. Активные контрольные точки изображаются в виде красного круга. Если контрольная точка отображается серым цветом, то она неактивная. Причиной перехода контрольных точек в неактивное состояние может служить наличие ошибки в файле или внесение изменений после ее установки. Для активизации контрольной точки необходимо сохранить файл. В том случае, если в файле имеется синтаксическая ошибка, то после сохранения файла

появляется диалоговое окно, в котором указывается причина ошибки и ее местоположение. Если в файле присутствует несколько синтаксических ошибок, то в диалоговом окне отображается информация, относящаяся только к первой ошибке. Вторую группу образуют переключатели Stop If Error, Stop If Warning, Stop If NaN Or Inf и Stop If All Error, которые используются для прекращения выполнения инструкций при возникновении ошибки (за исключением ошибок в блоках `try`), предупреждения, появления результата `Nan` или `Inf` и ошибки в любых блоках соответственно. В том случае, если опция активна, то напротив нее стоит «✓».

Во время отладки можно просматривать текущие значения переменных прямо в Редакторе. Для просмотра значения переменной необходимо установить курсор мыши над переменной и через некоторое время появится всплывающая подсказка, содержащая ее значение. Если массив значений переменной не помещается во всплывающей подсказке, то для просмотра всех значений следует воспользоваться инструментальным средством `Array Editor`. В ряде случаев отладку удобно проводить, когда инструментальные средства `Array Editor` и `Editor/Debugger` располагаются внутри рабочего стола системы MATLAB.

Упражнение 1. Работа с командами главного меню Editor/Debugger.

1. Создать файл `example_6_4` из примера 6.4 (`File → New → M-File`).
2. Поставить точку останова (`Set/Clear Breakpoints`) и две закладки (`Set/Clear Bookmarks`), как показано на рис. П-8.1.
3. Переместить текстовый курсор ко второй закладке, а затем к первой (`Next` и `Previous Bookmarks`).
4. Определить, сколько раз встречается в файле слово `string` (`Find&Replace`).
5. Заменить в файле `example_6_4` все слова `string` на `str` (`Find&Replace`).
6. Сохранить файл под именем `example_6_4_a` (`Save As...`).
7. Перейти в режим отладки (`Run`).
8. После ввода необходимых чисел в командном окне выполнить один шаг (`Step`) и просмотреть значение переменной `a`.
9. С помощью пошагового выполнения с заходом в вызываемую функцию перейти в функцию `num2str()` (`Step In`).
10. Вернуться обратно в сценарий `example_6_4` с помощью команды `Step Out`.
11. Завершить выполнение сценария (`Continue`).
12. Удалить контрольную точку (`Clear All Breakpoints`).
13. Удалить две закладки (`Set/Clear Bookmarks`).
14. Установить текстовый курсор внутри слова `num2str` и выбрать команду `Balance Delimiter`.
15. Закрыть файл `example_6_4_a` (`Close example_6_4_a`).

Панель инструментов Редактора, представленная на рис. П-8.3, содержит кнопки быстрого доступа к часто используемым командам главного меню. Инструментальная панель разделена на семь групп. В первой группе располагаются кнопки, которые позволяют выполнять файловые операции: создать новый m-файл (New M-File), открыть уже созданный файл (Open file) и сохранить текущий файл (Save). Вторую группу образуют кнопки, которые связаны с командами обмена информацией между Редактором и буфером (Cut, Copy и Paste), а также с командами последовательной отмены действий (Undo) и их восстановления (Redo). Третья группа состоит из одной кнопки, при нажатии

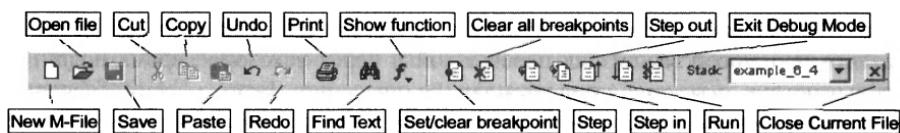


Рис. П-8.3. Инструментальная панель Editor/Debugger

на которую открывается диалоговое окно Печать (Print). С помощью этого диалогового окна можно задавать параметры печати и отправлять содержимое рабочей области Редактора на печать. Четвертая группа состоит из двух кнопок – Find Text и Show function. При нажатии на первую кнопку появляется диалоговое окно Find&Replace, с помощью которого осуществляется поиск заданного фрагмента текста в текущем файле, а также его замена в случае необходимости. Вторая кнопка является выпадающим списком, в котором содержатся все имена функций, находящиеся в текущем файле. Кнопки пятой группы (Set/clear breakpoint и Clear all breakpoints) служат для установки и удаления контрольных точек в текущей строке или во всем файле. Кнопки Step, Step In, Step Out, Run/Continue и Exit Debug Mode, которые образуют шестую группу, применяются при отладке программы, а также для входа и выхода из режима отладки. Последняя группа состоит из комбинированного списка, который позволяет переключаться между рабочими областями выполняемых в текущий момент функций. В результате переключения в инструментальном средстве Workspace Browser отображается рабочая область выбранной функции. В правой части панели инструментов располагается кнопка Close Current File, при нажатии на которую происходит закрытие текущего файла.

Под панелью инструментов располагается рабочая область, в которой размещается содержимое файла. Редактор позволяет работать с несколькими файлами одновременно. Переключение между файлами осуществляется с помощью щелчка на соответствующей вкладке, находящейся под рабочей областью.

Слева от рабочей области располагается поле номеров строк. Редактор позволяет изменять ширину этого поля. Максимальное число разрядов номера строки равно девяти. Номера строк позволяют быстро найти необходимую строку, в которой содержится ошибка.

В нижней части Редактора располагается строка состояния. Страна состояния служит для отображения имени функции, внутри которой находится текстовый курсор, а также номеров строки (Ln) и колонки (Col) его текущей позиции.

Управление 2. Работа с командами панели инструментов Редактора.

1. Создать сценарий example_6_6 из примера 6.6.
2. Установить контрольные точки, как показано на рис. П-8.4.
3. Перейти в режим отладки (Run) и ввести массив чисел.
4. Выполнить инструкцию, расположенную во второй строке, и просмотреть значение переменной *a*.
5. Выполнить инструкции до следующей контрольной точки (Continue) и просмотреть значения переменных *s* и *str* в случае наличия простых чисел во введенном массиве.
6. Выполнить инструкции до следующей контрольной точки с заходом в вызываемые функции (Step In) при наличии простых чисел в вызываемом массиве. После передачи управления в вызываемые функции необходимо полностью выполнить их и вернуться в точку вызова (Step Out).
7. Просмотреть значение переменной *string* и выполнить оставшиеся инструкции в сценарии.

Кроме рассмотренных элементов графического интерфейса пользователя Редактор имеет контекстно-зависимое меню, представленное на рис. П-8.5. Контекстно-зависимое меню появляется при щелчке правой кнопки мыши в рабочей области. В меню содержатся наиболее часто используемые команды при работе с Редактором. Команды разбиты на четыре группы. Первая группа команд становится активной при выделении фрагмента текста. Первая команда этой группы передает выделенный фрагмент ядру системы MATLAB для вычисления результата. Вторая команда служит для открытия файла, содержащего выделенную функцию или сценарий, в отдельной рабочей области Редактора. Третья команда активизирует инструментальное средство Help Browser и отображает в нем справочную информацию по выделенной функции. Вторая группа команд позволяет обмениваться информацией с буфером обмена. Третья группа предназначена для работы с комментариями и отступами. Последняя группа служит для установки и снятия контрольных точек, а также выполнения программы до строки, в которой размещается текстовый курсор. При вызове контекстно-зависимого меню происходит перемещение текстового курсора в позицию, на которую указывал до щелчка курсор мыши.

Система MATLAB позволяет изменять установки Редактора, принятые по умолчанию. Установки можно изменять с помощью диалогового окна Preferences, представленного на рис. П-8.6 и появляющегося при задании одноименной команды из меню File.

```

1 answer = input('Введите массив положительных чисел: ');
2 if a = isprime(answer(:)); % логический массив
3 if any(a) % проверка наличия простых чисел в массиве
4 s = sum(a); % вычисление кол-ва простых чисел
5 str = int2str(s); % преобразование числа в строку
6 d = size(str,2); % определение числа разрядов
7 % значение правого десятичного разряда в числе
8 lsd = str2num(str(d));
9 % формирование сообщения о количестве простых чисел в массиве
10 if ((d>1)&&(str2num(str(d-1))==1)) || ~((lsd>0)&(lsd<5))
11 string = ['Вы ввели ' num2str(s) ' простых чисел'];
12 elseif lsd == 1
13 string = ['Вы ввели ' num2str(s) ' простое число'];
14 else
15 string = ['Вы ввели ' num2str(s) ' простых числа'];
16 end
17 disp(string) % вывод информации о количестве простых чисел
18 answer(a') % вывод простых чисел
19 else
20 disp('Во введенном массиве нет простых чисел')
21 end

```

The screenshot shows the MATLAB Editor window with the file `example_6_6.m` open. The code performs the following steps:
1. Prompts the user to enter a vector of positive integers.
2. Checks if the input is a logical array.
3. Checks if there are any prime numbers in the array.
4. Calculates the count of prime numbers.
5. Converts the count to a string.
6. Determines the number of digits in the string.
7. Gets the rightmost digit of the number.
8. Formats a message based on the count and the digit.
9. Displays the message.
10. Handles the case where there is exactly one prime number.
11. Handles the case where there are no prime numbers.
12. Handles the case where there is exactly one digit.
13. Handles the case where there are multiple digits.
14. Handles the case where there is exactly one digit.
15. Handles the case where there are multiple digits.
16. Handles the case where there is exactly one digit.
17. Displays the message.
18. Returns the prime numbers.
19. Handles the case where there are no prime numbers.
20. Displays the message.
21. Handles the case where there are no prime numbers.

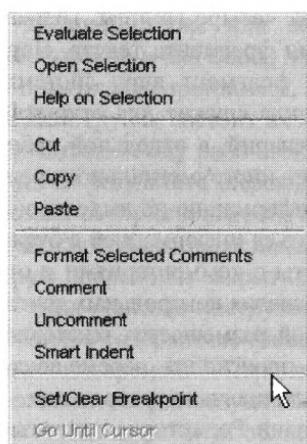
Рис. П-8.4. Окно Редактора с активным файлом `example_6_6.m`

Рис. П-8.5. Контекстно-зависимое меню

Диалоговое окно Preferences позволяет:

- выбирать внешний редактор для создания m-файлов (Text Edit) или использовать стандартное инструментальное средство Editor/Debugger (MATLAB editor);
- задавать максимальное количество строк в выпадающем меню File, отводимых под имена последних открываемых в Редакторе файлов (Most recently used files list, Number of entries);
- изменять максимальную ширину комментария (Max width) и включать автоматический перенос комментариев на новую строку при достижении установленной ширины (Autowrap comments);
- управлять автоматическим открытием файлов во время отладки (Automatically open files when debugging);

- управлять автоматическим открытием файлов во время загрузки новой сессии, которые были открыты в предыдущей сессии во время ее завершения (*On restart reopen files from previous MATLAB session*);
- назначать имя шрифта, его размер и начертание или использовать шрифт рабочего стола (*Font*);
- задавать цвет текста и рабочей области, а также задавать цвет и включать выделение цветом ключевых слов языка MATLAB, комментариев, строк, неправильно заданных строк, команд операционной системы и ошибок (*Colors*);

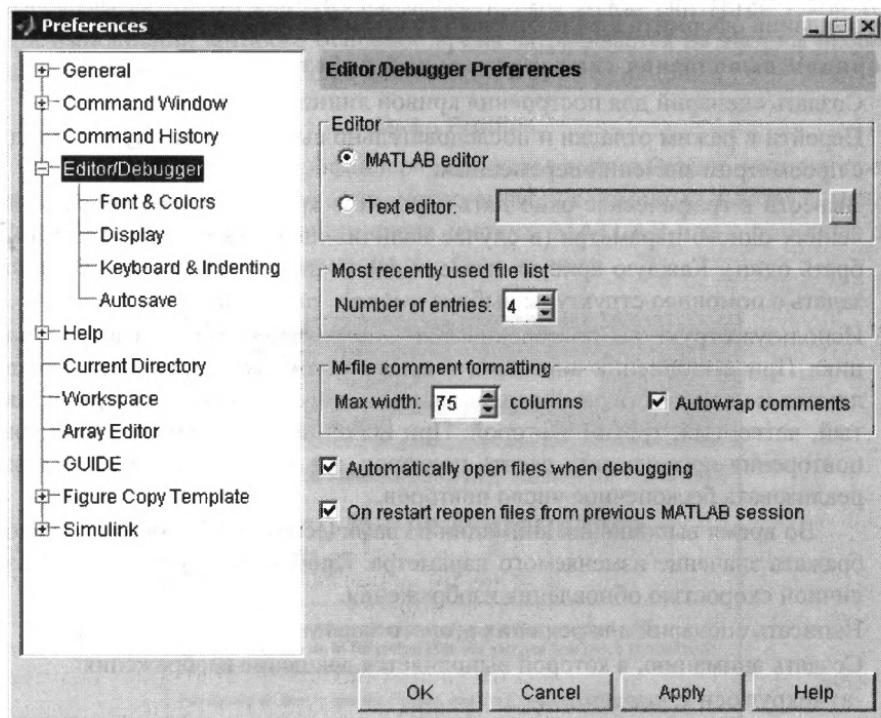


Рис . П-8.6. Диалоговое окно Preferences для задания установок Редактора

- выбирать способ открытия файлов в одном окне Редактора, где переключение между файлами выполняется с помощью вкладок, или в индивидуальных окнах (*Open files in editor*);
- включать отображение панели инструментов (*Show toolbar*), поля номеров строк (*Show line numbers*) и всплывающих подсказок со значениями переменных в режиме редактирования (*Enable datatips in edit mode*);

- задавать сочетания клавиш, которые приняты по умолчанию при работе с операционными системами Windows или Unix, устанавливать размер отступов и табуляции, а также включать настройки, определяющие соответствие числа открывающихся и числу закрывающихся скобок (Keyboard&Indenting);
- управлять режимом автоматического сохранения файлов (Autosave).

Самостоятельная работа

Самостоятельная работа состоит из двух заданий. Задания необходимо взять из предыдущего практикума согласно своему варианту. Решение каждого задания оформить в виде отдельного сценария.

Порядок выполнения самостоятельной работы:

1. Создать сценарий для построения кривой линии из первого задания.
2. Перейти в режим отладки и последовательно выполнить каждую команду с просмотром значений переменных.
3. Вывести в графическое окно пять вариантов кривой при различных значениях одного параметра (в случае наличия нескольких параметров выбрать один). Каждую кривую вывести своим цветом. Цвет кривой линии задать с помощью структуры выбора switch case end.
4. Используя структуры повторения for end и while end, создать анимацию. При выполнении анимации в графическом окне циклически появляются и исчезают первый вариант линии, второй, третий, четвертый, пятый, четвертый, третий и второй. При использовании первой структуры повторения организовать десять повторов, а в случае второй структуры реализовать бесконечное число повторов.

Во время выполнения анимации в графическом окне необходимо отображать значение изменяемого параметра. Просмотреть анимацию с различной скоростью обновления изображения.

5. Написать сценарий для решения второго задания.
6. Создать анимацию, в которой выполняется вращение изображения:
 - а) вокруг оси x;
 - б) вокруг оси y;
 - в) вокруг оси z;
 - г) последовательно один оборот вокруг оси z, затем один оборот вокруг оси y и, наконец, один оборот вокруг оси x.

Для поворота изображения использовать матрицы поворота. Во время поворота в графическом окне отображать текущие значения углов поворота напротив соответствующих осей. При повороте изображения на угол, кратный 45° относительно исходного положения, изменять цвет соответствующего текстового изображения.

Практикум 9.

ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО PROFILER

Цель: Знакомство с инструментальным средством Profiler, а также приобретение навыков создания высокоэффективного кода на языке MATLAB.

В состав системы MATLAB входит инструментальное средство Profiler (Профайлер), которое является подпрограммой протоколирования и позволяет оценить время выполнения отдельных функций, входящих в программу. Использование Профайлера при разработке программ позволяет создавать высокоэффективный код. Высокоэффективным кодом считается такой код, при выполнении которого основное время затрачивается на вызовы небольшого числа встроенных в ядро системы MATLAB функций.

Окно Profiler

Результаты работы Профайлера выводятся в окно Profiler, которое представлено на рис. П-9.1. Это окно можно вызвать с помощью выбора команды Profiler из меню View рабочего стола системы MATLAB или с помощью задания в командном окне команды profile viewer.

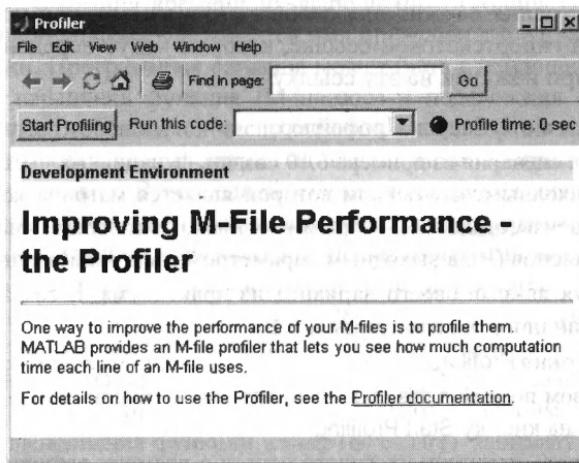


Рис. П-9.1. Инструментальное средство Profiler

Окно Profiler состоит из строки заголовка, строки главного меню, панели инструментов, рабочей области и строки состояния.

Главное меню Профайлера состоит из шести пунктов: File, Edit, View, Web, Window и Help. Команды всех пунктов главного меню совпадают с соответствующими командами главного меню рабочего стола системы MATLAB.

Под главным меню размещается панель инструментов, которая позволяет:

- перемещаться между страницами рабочей области (Back и Forward);
- повторно загружать текущую страницу рабочей области (Reload);
- открывать начальную или итоговую страницу отчета о времени выполнения команд с начала запуска Профайлером до его остановки (Profile Summary);
- выводить на печать текущую страницу (Print);
- осуществлять поиск введенного в поле фрагмента текста в пределах текущей страницы.

Под панелью инструментов располагается рабочее поле, которое состоит из поля управляющих элементов и страницы отчета о времени выполнения инструкций. В поле управляющих элементов размещаются кнопка запуска и остановки Профайлером, комбинированный список, а также индикатор состояния Профайлером. Комбинированный список используется для вызова сценария или функции, время выполнения которых необходимо просмотреть. Комбинированный список позволяет задавать команду в текстовом поле, а также выбирать ранее заданные команды из выпадающего списка. Индикатор используется для визуальной оценки времени работы Профайлером. Во время его работы индикатор окрашен в зеленый цвет.

Строка состояния служит для отображения команд при позиционировании курсора на гипертекстовой ссылке, которые будут переданы ядру системы MATLAB при нажатии на эту ссылку.

Упражнение 1. Запуск Профайлером.

1. На основе сценария из примера 6.10 создать функцию `example_6_10()`, первым входным параметром которой является матрица коэффициентов при неизвестных (A), вторым входным параметром – матрица свободных членов (B), а выходным параметром – матрица неизвестных x .
2. Используя данные своего варианта из практикума 4, создать с помощью командного окна матрицы A и B .
3. Открыть окно Profiler.
4. В текстовом поле Run this code написать имя функции `example_6_10()` и нажать на кнопку Start Profiling.
5. Просмотреть страницу итогового отчета о времени выполнения функции `example_6_10()`.

После окончания работы Профайлером на странице рабочего поля отображается итоговая информация о времени выполнения всех файлов, которые были вызваны во время его работы. Под файлами в данном случае понимается любая программная единица (сценарий, первичная функция и подфункция), которая не была реализована в ядре системы MATLAB. Итоговый отчет о времени выполнения функции, созданной в первом упражнении, представлен на рис. П-9.2.

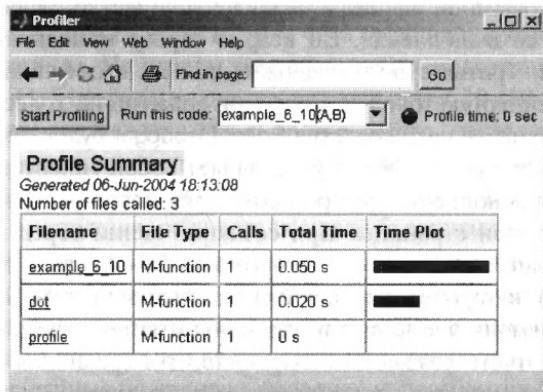


Рис. П-9.2. Страница итогового отчета о работе Профайлера

В итоговый отчет входит дата его создания, число вызванных файлов, а также таблица, содержащая более полную информацию о вызванных программных единицах. Вызываемые программные единицы размещаются в таблице в порядке убывания времени их выполнения. Таблица состоит из пяти колонок. В первой колонке указывается имя программной единицы, во второй колонке – ее тип. Программная единица может быть следующего типа: сценарий (M-script), первичная функция (M-function) и подфункция (M-subfunction). В третьей колонке отображается число вызовов, а в четвертой колонке – числовое значение времени, затраченного на выполнение программной единицы с учетом числа вызовов. В последней колонке приводится диаграмма относительного времени выполнения. Максимальная ширина диаграммы соответствует самой продолжительной по времени выполнения программной единице.

Профилирование файлов осуществлялось на компьютере с процессором PII-400 и оперативной памятью объемом 128 Mb. Данные, полученные при выполнении практикума, могут отличаться от данных, представленных на рисунках. При задании команды `example_6_10(A,B)` было вызвано три файла – `example_6_10`, `dot` и `profile`, в которых размещаются одноименные функции. Полное время выполнения функции `example_6_10()` занимает 50 мс. Во время ее выполнения вызывается функция `dot()`, на которую затрачивается 20 мс. Так как окончание профилирования сопровождается вызовом файла `profile`, то он помещен в итоговую таблицу.

С целью просмотра полного отчета о времени выполнения каждой инструкции функции `example_6_10()` необходимо щелкнуть на гипертекстовой ссылке, содержащей ее имя. Страница полного отчета разделена на три секции.

Первая секция, изображенная на рис. П-9.3, представляет собой заголовок страницы. Заголовок состоит из четырех строк. В первой строке указы-

вается имя программной единицы, а также количество ее вызовов и время, затраченное на ее выполнение. Во второй строке выводится дата и время создания отчета. Третья строка содержит полное имя файла с учетом пути доступа в виде гипертекстовой ссылки. При нажатии на эту ссылку в рабочей области инstrumentального средства Editor/Debugger будет открыт соответствующий файл. Четвертая строка представляет собой гипертекстовую ссылку на команду, при выполнении которой создается новое окно, куда копируется содержимое текущей страницы. При создании копии перед первой строкой заголовка появляется новая строка (*Links are disabled because this is a static copy of a profile report*), которая информирует пользователя о том, что все внешние динамические ссылки удалены, так как данная копия отчета является статической. Копию отчета создают, как правило, для сравнения производительности функции при различных вариантах ее реализации.

example_6_10 (1 call, 0.050 sec)
Generated 06-Jun-2004 18:24:57
M-function in file C:\MATLAB6p5\work\example_6_10.m
[Copy to new window for comparing multiple runs]

Рис. П-9.3. Заголовок страницы полного отчета

Во вторую секцию, изображенную на рис. П-9.4, входит список программных единиц, которые вызывают анализируемую программную единицу (Parents). Список функций может быть скрыт (hide) либо представлен:

- в виде таблицы (table), в которой отображаются имя вызывающей программной единицы, ее тип и число вызовов;
- в виде списка (list), в котором перечисляются только имена вызывающих программных единиц (сокращенный вариант табличного вида).

В рассматриваемом примере нет вызывающих программных единиц.

После списка вызывающих программных единиц следует список ссылок на строки, выполнение которых занимает большую часть всего времени выполнения текущей программной единицы (Lines where the most time was spent). Список может быть скрыт (hide), представлен в виде таблицы (table) или в виде списка гипертекстовых ссылок на соответствующие строки (list). Таблица состоит из шести столбцов. В первом столбце (Line Number) выводятся номера строк. Во втором столбце (Code) отображаются начальные символы строк. В третьем столбце (Calls) размещается информация о количестве обращений к инструкциям, расположенным в соответствующих строках. В четвертом столбце (Total Time) в секундах выводится время выполнения инструкций с учетом числа обращений. В пятом столбце (% Time) отображается в процентах относительное время выполнения инструкций. Заключительный шестой столбец (Time Plot) содержит диаграммы времени выполнения. В рассматриваемом примере

наибольшее время (20 мс) расходуется на выполнение инструкций в 29-й строке. Время их выполнения составляет 40% от общего времени выполнения программной единицы. На втором месте находятся инструкции, располагающиеся в строке 11. На их выполнение затрачивается 20 (10 + 10) миллисекунд, что также составляет 40%. Третье место занимают инструкции в строке 20, которые выполняются 3 раза. Время их суммарной работы равно 10 миллисекундам, что составляет 20% от общего времени. Четвертое место (0.1%) занимает инструкция, осуществляющая объединение массивов и расположенная в 3-й строке. На пятом месте (0.1%) находится инструкция из строки 7, при выполнении которой происходит создание одномерного массива, состоящего из единиц. Выполнение остальных строк занимает 0% от общего времени. Последняя строка в таблице содержит итоговую информацию. Приведенные в таблице данные по времени выполнения программной единицы являются приближенными данными, полученными после округления, что приводит к ошибкам округления. Например, в рассматриваемом случае суммарное относительное время выполнения инструкций равно 100.2%.

Parents (calling functions) [table list hide]					
Lines where the most time was spent [table list hide]					
Line Number	Code	Calls	Total Time	% Time	Time Plot
29	x(row,:) = (AB(row,col_co...)	1	0.020 s	40.0%	
11	[absmax, index] = max(abs(...	2	0.020 s	40.0%	
23	elseif row == row_count-1...	2	0.010 s	20.0%	
3	AB = [A B]; % объединение...	1	0.000 s	0.1%	
7	row_ones = ones(1,col_cou...	1	0.000 s	0.1%	
All other lines			0 s	0%	
Totals			0.050 s	100%	

Children (called functions) [table list hide]					
Filename	File Type	Calls	Total Time	% Time	Time Plot
dot	M-function	1	0.020 s	40.0%	
All other time (built-ins, math, etc.)			0.030 s	60.0%	
Totals			0.050 s	100%	

Рис. П-9.4. Вторая секция полного отчета

После списка ссылок на строки, выполнение которых занимает большую часть времени, располагается информация о времени выполнения вызываемых программных единиц (*Children*). Эта информация может быть скрыта либо представлена в виде таблицы или списка. Список, являющийся сокращенной формой таблицы, состоит из имен вызываемых программных единиц, представляющих собой гипертекстовые ссылки, за которыми в круглых скобках указано относительное время их выполнения. После нажатия на гипертекстовую ссылку на странице отчета появится детальная информация о времени выполнения соответствующей программной единицы. Таблица состоит из шести столбцов. В первом столбце размещается имя программной единицы, во втором – ее тип, в третьем – число вызовов, в четвертом – время выполнения в секундах, в пятом – относительное время выполнения в процентах и в шестом – диаграмма времени выполнения. В рассматриваемом случае информация о вызываемых программных единицах представлена на рис. П-9.4 в виде таблицы. В таблице содержится информация о времени выполнения функции `dot()`, находящейся в библиотеке специальных функций в директории ...\\MATLAB бр5\\toolbox\\MATLAB \\specfun. Время, затрачиваемое на ее выполнение, равно 20 мс, что составляет 40% от общего времени выполнения. Остальное время (30 мс) расходуется на выполнение функций, встроенных в ядро системы MATLAB.

Последняя секция страницы полного отчета, представленная на рис. П-9.5, содержит листинг анализируемой программной единицы. В листинге отображается ее код, а также время выполнения строк (`time`), число обращений к строкам (`calls`), результат применения ускорителя (`acc`) и номера строк (`line`).

С целью облегчения анализа программной единицы используются цветовые обозначения символов и их фона. Цветовые обозначения применяются ко всем символам строки. Различают три цвета символов: зеленый, черный и серый. Зеленый цвет используется для обозначения комментария, если он занимает всю строку (например, строка 9 – % прямой ход). Черным цветом обозначается строка, которая содержит исполняемые инструкции (например, строка 3 – `AB = [A B]; % объединение массивов`). Черным цветом обозначается и комментарий, который располагается после исполняемых инструкций на той же строке. Серым цветом окрашиваются символы строки, которая не содержит исполняемых инструкций (например, строка 1 – заголовок функции), либо эта строка является продолжением предыдущей строки, на которой располагаются исполняемые инструкции (например, строка 16, являющаяся продолжением строки 15). Информацию о наличии исполняемых инструкций в строке можно использовать при отладке программной единицы.

Наряду с цветом символов применяется цветовое обозначение фона строки, в которой располагаются исполняемые инструкции. Цвет фона строки

зависит от вида анализируемой характеристики на выходе. Выбор основной характеристики осуществляется с помощью щелчка на соответствующей гипертекстовой ссылке, располагающейся в верхней части секции. Существует пять вариантов выбора: время выполнения (Time), число обращений к строке (Number of Calls), выполненные строки (Coverage), применение ускорителя (Acceleration) и отсутствие основной выходной характеристики (No Color), при которой все строки отображаются без цветового фона.

File listing
 Color highlight code according to [Time | Number of Calls | Coverage | Acceleration | No Color]

```

time calls acc line
1 function [x] = example_6_10(A,B)
2 % Решение СЛАУ методом Гаусса с частичным выбором главного э
0.000 1 .   3 AB = [A B]; % объединение массивов
0.000 1 .   4 row_count = size(AB,1); % число строк = числу неизвестных
0.000 1 .   5 col_count_A = size(A,2); % число столбцов в матрице A
0.000 1 .   6 col_count_B = size(B,2); % число столбцов в матрице B
0.000 1 .   7 row_ones = ones(1,col_count_B); % строка единиц
0.000 1 .   8 x = zeros(col_count_A,col_count_B); % выделение памяти под x
0.000 1 .   9 % прямой ход
0.020 1 x   10 for col = 1:col_count_A-1 % шаг равен +1
2 x   11 [absmax,index] = max(abs(AB(:,end,col)));
2 .   12 if index > 1 % перестановка строк
1 .   13 AB([index+col-1 col],:) = AB([col index+col-1],:);
1 .   14 end
2 .   15 AB(:,col+1:end,col:end) = AB(:,col+1:end,col:end)-...
16 (AB(:,col+1:end,col)./AB(:,col,col)).*AB(:,col+1:end);
2 .   17 end
1 .   18 % обратный ход
1 .   19 for row = row_count:-1:1
3 .   20 if row == row_count
21 % Вычисление последнего неизвестного
1 .   22 x(row,:)=AB(row,col_count_A+1:end)./AB(row,row);
0.010 2 .   23 elseif row == row_count-1
24 % Вычисление предпоследнего неизвестного
1 .   25 x(row,:)=(AB(row,col_count_A+1:end)-...
26 AB(row, row+1).*x(row+1,:))./AB(row, row);
1 .   27 else
28 % Вычисление остальных неизвестных
0.020 1 x   29 x(row,:)=(AB(row,col_count_A+1:end)-...
30 dot((AB(row, row+1:col_count_A)).*row_ones),x(row+1:end,
31 ./AB(row, row));
1 .   32 end
3 .   33 end

```

Рис. П-9.5. Третья секция полного отчета

В том случае, если основной характеристикой является время выполнения, то исполняемые строки окрашиваются в розовый цвет. Интенсивность цвета зависит от времени их выполнения. В рассматриваемом примере строки

11 и 29 имеют самую большую интенсивность, так как на выполнение находящихся в них инструкций затрачивается наибольшее время. В колонке time выводится с точностью до третьего знака после десятичной точки время выполнения исполняемых инструкций. В некоторых случаях в этой колонке отсутствует значение времени выполнения исполняемых инструкций, которые вызывались во время работы программной единицы. Это свидетельствует о том, что время их выполнения настолько мало, что даже не было вычислено Профайлером.

Если основной характеристикой служит число обращений к строке, то в качестве фона используется голубой цвет. Интенсивность голубого цвета зависит от числа обращений к строке. В рассматриваемом примере строки 20 и 33 будут окрашены в голубой цвет наибольшей интенсивности. Эти строки соответствуют началу и концу структуры с тройным выбором if elseif else end. Так как во время выполнения анализируемой программной единицы каждая ветвь была пройдена по одному разу, то условие первой ветви было вычислено три раза, а второй – два. В данном случае нет принципиальной разницы, инструкции какой из ветвей использовать первыми.

При выборе характеристики Coverage строки, исполняемые инструкции которых использовались во время выполнения анализируемой программной единицы, окрашиваются в голубой цвет одной интенсивности. В рассматриваемом примере все строки, в которых содержатся исполняемые инструкции, окрашиваются в голубой цвет, что свидетельствует об эффективном использовании программной единицы.

В случае выбора характеристики Accelerator применяется розовый цвет одной интенсивности. В розовый цвет окрашиваются только те строки, исполняемые инструкции которых были оптимизированы с применением механизма ускорения выполнения инструкций. Наряду с цветовым выделением в колонке acc напротив соответствующей строки появляется «». Если механизм ускорения выполнения инструкций не был применен, то напротив этой строки вместо точки выводится символ «x». При щелчке на этом символе появляется диалоговое окно Acceleration information, где выводится информация о причинах, по которым механизм ускорения выполнения инструкций не может быть применен к этой строке. В рассматриваемом примере при щелчке на символе «», расположенному напротив строки 29, появляется диалоговое окно, изображенное на рис. П-9.6. В этом окне отображается причина, по которой строка 29 не может быть оптимизирована. В этой строке выполняется вызов функции `dot()`, которая является внешней по отношению к ядру системы MATLAB. При появлении сообщений подобного рода оптимизацию кода следует проводить самостоятельно. Если возможно, вызов внешней функции следует заменить вызовом или последовательностью из нескольких вызовов встроенных в ядро MATLAB функций.

В рассматриваемом примере скалярное произведение соответствующих векторов можно вычислить с помощью операции умножения их элементов и суммирования полученных произведений. Операция умножения может быть реализована с помощью оператора «.*», а суммирование полученных произведений – с помощью функции `sum()`.

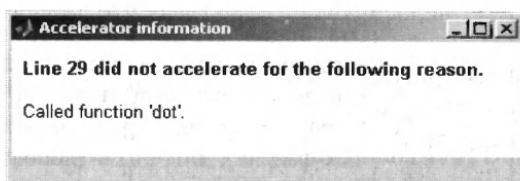


Рис . П-9.6. Диалоговое окно Accelerator information

Упражнение 2. Работа с окном Профайлером.

1. Создать четыре копии полного отчета о времени выполнения файла `example_6_10.m`. Во всех копиях данные второй секции представлены в виде таблицы. В первой копии основной характеристикой при анализе листинга является время выполнения, во второй копии – число обращений к строкам кода, в третьей копии – выполненные исполняемые строки и в четвертой копии – применение оптимизатора выполнения инструкций к строке.
2. Сделать вывод на основе полученных копий.
3. Заменить вызов функции `dot()` на оператор «.*» и вызов функции `sum()`.
4. Запустить Профайлер и просмотреть результаты выполнения файла `example_6_10.m`.
5. Сделать выводы на основе полученных результатов.

Работа с Профайлером из командной строки

В окно Profiler выводится часть результатов выполнения функции `profile()`. Наиболее полную информацию о результатах профилирования можно получить с помощью вызова функции `profile()` из командной строки. Вызов функции `profile()` осуществляется в командном формате с добавлением опций. Возможные варианты вызова функции представлены в табл. П-9.1.

Таблица П-9.1

Вариант вызова	Опция	Описание
profile on		Выполняется запуск профилирования с предварительным удалением предыдущей информации
	-detail level	Эта опция определяет набор функций, для которых собирается статистика. Существует три возможных варианта этой опции: <code>mfunc</code> (внешние функции, располагающиеся в <code>m-</code> и <code>tex-</code> файлах), <code>builtin</code> (дополнительно анализируются встроенные функции) и <code>operator</code> (дополнительно анализируются встроенные операторы)
	-history	Фиксируется последовательность вызовов функций согласно их набору
profile off		Окончание профилирования
profile report		Окончание профилирования, создание отчета в формате HTML и вывод его в установленном по умолчанию Web-браузере
	имя файла	Сохраняет отчет с указанным именем в текущей директории
profile plot		Окончание профилирования и вывод результатов в графическое окно в виде диаграмм
profile resume		Запуск Профайлером с сохранением результатов, полученных при предыдущем запуске
profile clear		Удаление собранных во время работы Профайлером статистических данных
profile viewer		Открытие окна Profiler
profile status		Получение информации о текущих параметрах профилирования
profile info		Прекращение профилирования и вывод в командное окно результатов

При работе с Профайлером из командной строки необходимо выполнить следующие шаги:

- 1) активизировать профилирование;
- 2) выполнить `m`-файл, профиль которого необходимо получить;
- 3) остановить профилирование и вывести результаты для анализа.

Результаты профилирования можно выводить в Web-браузер либо в командное окно. При создании отчета в формате HTML автоматически загружается установленный по умолчанию Web-браузер и в нем открывается страница итоговой информации (**Summary**). На рис. П-9.7 представлены итоговые данные по профилированию файла example_6_10 с учетом изменений, сделанных в упражнении 2.

Страница итоговой информации состоит из:

- заголовка страницы (**MATLAB Profile Report: Summary**);
- даты и времени создания отчета (*Report generated*);
- общего времени профилирования (Total recorded time);
- числа вызванных встроенных функций (Number of Builtin-functions);
- числа вызванных m-функций (Number of M-functions);
- точности измерения времени (Clock precision);
- тактовой частоты компьютера (Clock Speed);
- списка функций, представленного в виде таблицы.

В зависимости от установленного для профилирования набора функций на странице итогового отчета могут появляться строки, соответствующие количеству вызовов встроенных функций (Builtin-functions), первичных функций (M-functions), подфункций (M-subfunctions), а также функций смешанного программирования (MEX-functions).

В рассматриваемом примере общее время профилирования составляет 40 мс, количество вызванных встроенных файлов – 9, число вызванных m-файлов – 2, точность измерения – 0.2 мкс, тактовая частота используемого компьютера – 400 МГц.

Таблица состоит из восьми столбцов. В первом столбце в виде гипертекстовой ссылки отображается имя функции (Name). Во втором и третьем столбцах располагается соответственно абсолютное и относительное время (Time), затраченное на выполнение функции с учетом вызываемых в ней функций. В четвертом столбце находится количество вызовов функции (Calls). В пятом столбце приведено время одного вызова функции (Time/call). В шестом и седьмом столбцах содержится абсолютное и относительное время выполнения функции (Self time) без учета времени вызываемых в ней функций соответственно. В последнем столбце указывается ее местоположение (Location). Если функция является внешней, то указывается полный путь к файлу, в котором она находится. Полный путь представляет собой гипертекстовую ссылку, при нажатии на которую активизируется инструментальное средство Editor/Debugger, где отображается содержимое файла.

MATLAB Profile Report: Summary

Report generated 09-Jun-2004 22:09:23

Total recorded time 0.04 s
 Number of Built-in-functions 9
 Number of M-functions 2
 Clock precision 0.0000002 s
 Clock Speed 400 Mhz

Function List

Name	Time	Calls	Time/call	Self time	Location
example_6_10	0.0400000	100.0%	1.000000000	0.0400000	100.0% C:/MATLAB6p5/work/example_6_10.m
callstats	0.0000000	0.0%	2.000000000	0.0000000	0.0% Built-in function
profile	0.0000000	0.0%	1.000000000	0.0000000	0.0% C:/MATLAB6p5/toolbox/matlab/general/profile.m
whos	0.0000000	0.0%	1.000000000	0.0000000	0.0% Built-in function
*	0.0000000	0.0%	1.000000000	0.0000000	0.0% Built-in function
ctranspose	0.0000000	0.0%	1.000000000	0.0000000	0.0% Built-in function
max	0.0000000	0.0%	2.000000000	0.0000000	0.0% Built-in function
abs	0.0000000	0.0%	2.000000000	0.0000000	0.0% Built-in function
end	0.0000000	0.0%	2.000000000	0.0000000	0.0% Built-in function
-	0.0000000	0.0%	1.000000000	0.0000000	0.0% Built-in function
display	0.0000000	0.0%	2.000000000	0.0000000	0.0% Built-in function

Рис. П-9.7. Итоговая информация о профилировании файла example_6_10.m

Имена функций в таблице являются гипертекстовыми ссылками, при нажатии на которые открывается страница полного отчета о времени выполнения функций (**Function Details**) на описании соответствующей функции. При нажатии на имени example_6_10 появляется полный отчет о времени выполнения этой функции, представленный на рис. П-9.8. В полном отчете после общей информации выводятся строки кода, на выполнение которых затрачивается время. Затрачиваемое на выполнение время выводится с точностью до седьмого знака после десятичной точки, что дает более полную информацию по сравнению с результатом, выводимым в окно Profiler.

В том случае, если профилирование осуществлялось с фиксацией последовательности вызовов функций, то в созданном отчете содержится третья страница (**Function Call History**), на которой отображается последовательность вызовов функций.

MATLAB Profile Report: Function Details

example_6_10 C:/MATLAB6p5/work/example_6_10.m
 Time 0 0400000 s (100.0%)
 Calls 1
 Selftime 0 0400000 s (100.0%)

Function:	Time	Calls	Time/call
example_6_10	0.0400000	1	0.0400000

Parent functions:
none

Child functions:
none

100% of the total time in this function was spent on the following lines:

```

2: % Решение СЛАУ методом Гаусса с частичным выбором главного элемента
0.0000513 0: 3: AB = [A B]; % объединение массивов
0.0000009 0: 4: row_count = size(AB,1); % число строк = числу неизвестных
0.0000005 0: 5: col_count_A = size(A,2); % число столбцов в матрице A
0.0000005 0: 6: col_count_B = size(B,2); % число столбцов в матрице B
0.0000258 0: 7: row_ones = ones(1,col_count_B); % строка единиц
0.0000160 0: 8: x = zeros(col_count_A,col_count_B); % выделение памяти под x
9: % прямой ход
10: for col = 1:col_count_A-1 % шаг равен +1
0.0100000 25: 11: [absmax,index] = max(abs(AB(:,end,col)));
12: if index > 1 % перестановка строк

16: (AB(:,col+1:end,col) ./ AB(:,col,col)) *AB(:,col:end);
0.0200000 50: 17: end
18: % обратный ход

26: AB(row,:)*x(row+1,:)) ./ AB(row,row);
0.0000002 0: 27: else
28: % Вычисление остальных неизвестных
0.0100000 25: 29: x(row,:) = (AB(row,col_count_A+1:end) - ...
30: sum((AB(row,:)*x(row+1:col_count_A)) *x(row+1:end,:))) ...

```

Рис. П-9.8. Страница полного отчета о времени выполнения функции example_6_10

MATLAB Profile Report: Function Call History

display

example_6_10 (C:/MATLAB6p5/work/example_6_10.m)

=

end

abs

max

end

abs

max

ctranspose

*

whos

display

profile (C:/MATLAB6p5/toolbox/matlab/general/profile.m)

callstats

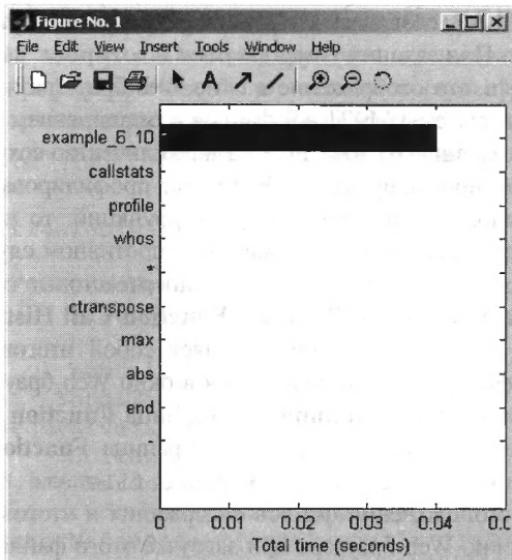
callstats

**Р и с . П-9.9. Последовательность вызовов функций
при выполнении файла example_6_10.m**

Информация, представленная на странице **Function Call History**, позволяет определить, какие функции вызываются внутри структур повторения. Уменьшение их числа и/или их замена на встроенные в ядро функции позволяет сократить время выполнения основной программной единицы.

В рассматриваемом примере замена функции `dot()` на оператор `.*` и встроенную функцию `sum()` привела к увеличению быстродействия на 25%.

Наряду с представлением результатов профилирования в текстовом виде, система MATLAB позволяет отображать результаты в графическом виде. С целью просмотра результатов в графическом окне необходимо задать команду `>> profile plot`. После ее выполнения появится графическое окно, изображенное на рис. П-9.10. В графическом окне в виде диаграммы показано общее время выполнения отдельных функций в секундах.



Р и с . П-9.10. Общее время выполнения функций

Упражнение 3. Профилирование с помощью командной строки.

1. Включить профилирование с анализом встроенных функций, первичных функций, подфункций и функций смешанного программирования.
2. Выполнить функцию `example_6_10()`. В качестве входных данных использовать данные из предыдущих упражнений. При вызове функции из командного окна подавить вывод результата.
3. Создать отчет в формате HTML.
4. Просмотреть созданные страницы отчета и сделать вывод на основе полученных данных.
5. Включить профилирование с анализом операторов, встроенных функций, первичных функций, подфункций и функций смешанного программирования, а также с сохранением последовательности вызовов функций.
6. Повторно вызвать функцию `example_6_10()`.
7. Создать отчет в формате HTML.
8. Сделать вывод на базе полученных данных о последовательности вызовов функций.
9. Отобразить общее время выполнения функции в графическом окне в виде диаграммы.
10. Сравнить полученные результаты с результатами предыдущего упражнения.

Система MATLAB позволяет сохранять результаты отчета, созданного из командной строки. При задании команды `profile report filename` происходит создание отчета, его отображение в окне Web-браузера и сохранение в текущей директории системы MATLAB файлов с расширением `html`, имена которых начинаются с заданного слова `filename`. Количество сохраняемых файлов зависит от режима профилирования. Если при профилировании осуществлялась запись последовательности вызываемых функций, то информация о результатах отчета сохраняется в пяти файлах, в противном случае – в четырех. Файл `filename_contents.html` содержит гипертекстовые ссылки на страницы отчета **Summary**, **Function Details** и **Function Call History**. Содержимое файла `filename_summary.html` представляет собой итоговую информацию о результатах профилирования и копируется в окно Web-браузера при обращении к гипертекстовой ссылке **Summary**. Страница **Function Details** размещается в файле `filename_details.html`, а страница **Function Call History** – в файле `filename_history.html`. В файле `filename.html` содержится информация о местоположении файлов содержания и итогового отчета, которые выводятся в окно Web-браузера при загрузке этого файла. Он загружается в окно Web-браузера по умолчанию во время вывода отчета после его создания.

Система MATLAB позволяет просматривать результаты профилирования не только в окнах **Profiler** и Web-браузера, но и в командном окне. Перед выводом результатов работы Профайлер в командное окно необходимо создать переменную, в которой они будут располагаться. Переменная с результатами профилирования создается в результате вызова функции `profile()` с входным строковым параметром `info`. Например, команда `>> res = profile('info')` создает в основной рабочей области системы MATLAB переменную `res`, которая содержит результаты профилирования, представленные в виде массива структуры. Первый уровень массива структуры содержит пять полей: `FunctionTable`, `FunctionHistory`, `ClockPrecision`, `Name`, `ClockSpeed`. В результате выполнения следующей последовательности команд в командном окне отобразится первый уровень массива структуры, содержащий результаты профилирования:

```
>> profile on -detail builtin -history, example_6_10(A,B);...
res = profile('info')
res =
    FunctionTable: [8x1 struct]
    FunctionHistory: [2x24 double]
    ClockPrecision: 2.331002331002331e-007
        Name: 'MATLAB'
    ClockSpeed: 400
```

Первое поле – `FunctionTable` – является массивом структуры, размер которого совпадает с числом вызванных функций. В этом массиве структуры

содержится информация о каждой вызываемой во время профилирования функции. Функции заносятся в массив структуры последовательно при их вызове. Во время профилирования было вызвано восемь функций. Функция `example_6_10()` располагается во втором элементе массива структуры. Следовательно, для просмотра профиля этой функции необходимо задать следующую команду:

```
>> res.FunctionTable(2)
ans =
    FunctionName: 'example_6_10'
    FileName: 'C:\MATLAB6p5\work\example_6_10.m'
        Type: 'M-function'
    NumCalls: 1
    TotalTime: 0.039999999999996
    TotalRecursiveTime: 0.039999999999996
        Children: [4x1 struct]
        Parents: [0x1 struct]
    ExecutedLines: [22x4 double]
    AcceleratorMessages: {[1x168 char] [1x93 char]}
```

Массив структуры `FunctionTable` состоит из десяти полей. Поле `FunctionName` содержит имя вызываемой функции (`example_6_10`). В поле `FileName` располагается вместе с путем доступа имя файла, в котором находится описание вызываемой функции (`C:\MATLAB6p5\work\example_6_10.m`). Информация о типе функции размещается в поле `Type` (`M-function`). Поле `NumCalls` содержит данные о количестве вызовов (1). В поле `TotalTime` размещается общее время выполнения функции с учетом времени выполнения вызываемых из нее функций (0.0399999999996). Данные о времени выполнения функции с учетом ее рекурсивных вызовов находятся в поле `TotalRecursiveTime` (0.0399999999996). В следующих двух полях – `Children` и `Parents`, которые являются массивами структур, располагаются данные о вызываемых и вызывающих функциях соответственно. В рассматриваемом примере при выполнении функции `example_6_10()` вызываются четыре функции.

С целью просмотра информации о первой вызываемой функции необходимо задать следующую команду:

```
>> res.FunctionTable(2).Children(1)
ans =
    Index: 6
    NumCalls: 1
    TotalTime: 0
```

Первое поле – `Index` – содержит номер элемента массива структуры `FunctionTable`, в котором размещается полное описание профиля этой функции (6). В поле `NumCalls` размещается количество вызовов функции (1).

Информация о полном времени выполнения этой функции хранится в поле TotalTime(0).

Поле ExecutedLines является массивом двойной точности, число строк которого совпадает с числом строк кода. В первом столбце располагается номер строки, во втором – число обращений к этой строке во время выполнения функции, в третьем – время выполнения этой строки и в четвертом – номер элемента массива, в котором содержится сообщение оптимизатора кода выполнения. Сообщения оптимизатора кода выполнения располагаются в поле AcceleratorMessages в виде массива ячеек. В рассматриваемом примере поле AcceleratorMessages состоит из двух ячеек.

С целью просмотра содержимого второй ячейки необходимо задать следующую команду:

```
>> res.FunctionTable(2).AcceleratorMessages{2}
ans =
Called function 'max' with a number of inputs (1) or outputs (2)
that prevented acceleration.
```

Оптимизатор кода выполнения не может оптимизировать вызов функции, встроенной в ядро max(), так как ее вызов сопровождается одним входным параметром или двумя выходными. Вызов функции max() оптимизируется при наличии двух входных параметров и одного выходного.

Второе поле – FunctionHistory – верхнего уровня иерархии представляет собой двухмерный массив двойной точности, который содержит информацию о последовательности вызываемых функций. Это поле заполняется только в том случае, если Профайлер запускался с опцией history. Двухмерный массив состоит из двух строк. Вторая строка содержит номера вызываемых функций, которые соответствуют номерам элементов массива FunctionTable. Первая строка состоит из нулей и единиц. Ноль соответствует началу выполнения функции, а единица – окончанию.

С целью просмотра содержимого поля FunctionHistory необходимо задать следующую команду:

```
>> res.FunctionHistory
ans =
Columns 1 through 16
 0   1   0   0   1   0   1   0   1   0   1   0   1   0   1   0
 1   1   2   3   3   4   4   4   5   5   3   3   4   4   4   5   5   6
Columns 17 through 24
 1   1   0   0   1   0   1   1
 6   2   7   8   8   8   8   7
```

Следовательно, при выполнении функции example_6_10() последовательно вызываются функции 3 – end(), 4 – abs(), 5 – max() и 6 – ctranspose().

Первые три внутренние функции вызываются последовательно 2 раза, что соответствует их выполнению внутри структуры повторения. Группа из трех функций располагается в строке 11. Перед вызовом функции `max()` вычисляется значение ее входного параметра. При вычислении входного параметра последовательно вызываются функции `end()` и `abs()`. В результате вызова функции `end()` определяется число строк расширенной матрицы `AB`. Полученное значение используется при формировании массива данных, который передается функции `abs()` для вычисления абсолютных значений его элементов. В рассматриваемом примере число строк и столбцов расширенной матрицы может быть определено после ее формирования. Число строк расширенной матрицы совпадает с числом строк матрицы `A`, а число столбцов равно сумме столбцов матрицы `A` и матрицы `B` (`col_count = col_count_A + col_count_B`). Следовательно, вместо вызовов функции `end()`, где определяются номера максимальной строки или столбца расширенной матрицы, можно использовать значения соответствующих переменных. Функции, которые выполняются перед вызовом функции `example_6_10()`, а также после окончания ее работы, являются сервисными функциями. Сервисные функции предназначены для запуска и останова фоновых программ, фиксирующих время выполнения функций, а также для сбора полученных результатов.

Третье поле – `ClockPrecision` – верхнего уровня иерархии содержит данные о точности, с которой фиксировалось время выполнения функций. В рассматриваемом примере точность равнялась `2.331002331002331e-007` сек.

В четвертом поле – `Name` – указывается имя системы, в которой выполнялось профилирование. В данном случае в этом поле содержится строковое значение MATLAB.

В пятом поле – `ClockSpeed` – располагается значение в мегагерцах тактовой частоты процессора. Так как в данном случае использовался процессор PII-400, то поле содержит значение 400.

Упражнение 4. Просмотр результатов профилирования в командном окне.

1. В описании функции `example_6_10()` в местах, где встречается определение числа строк и столбцов расширенной матрицы `AB`, заменить вызовы функции `end()` на обращение к соответствующим переменным.
2. Выполнить профилирование функции `example_6_10()` с фиксацией времени выполнения всех видов функций и операторов, а также с записью последовательности вызовов функций.
3. Просмотреть результаты профилирования.
4. Определить, какие данные из полученного массива структуры используются при создании каждой страницы отчета в формате HTML и в окне Profiler.
5. Выводы записать в тетрадь.

Самостоятельная работа

Самостоятельная работа заключается в создании профиля сценария первого задания из практикумов 5 и 6, а также в повышении эффективности написанного кода.

Порядок выполнения самостоятельной работы:

1. Открыть сценарий, в котором осуществляется построение заданной линии.
2. Выполнить профилирование сценария с помощью окна Profiler.
3. Проанализировать полученные данные.
4. Если возможно, внести изменения, которые сократят время выполнения сценария. Объяснить сделанные изменения или их отсутствие. При внесении изменений сохранять копии предыдущих отчетов.
5. Закрыть окно Profiler.
6. Осуществить операцию получения профиля сценария из командной строки. Последовательно выполнить шесть вариантов профилирования. Четные варианты профилирования отличаются от нечетных наличием информации о последовательности вызываемых функций. В первом варианте необходимо фиксировать время выполнения только внешних функций, в третьем варианте – наряду с внешними функциями фиксировать время выполнения встроенных функций и в пятом варианте – дополнительно фиксировать время выполнения операторов. По каждому из вариантов сделать вывод на основе созданного соответствующего отчета.
7. Сохранить в виде html файлов шестой вариант отчета.
8. Выполнить профилирование сценария и просмотр результатов с помощью командной строки.
9. В тетради оформить отчет о выполнении практикума, в который включить отчет, созданный во время работы с окном Profiler и сгенерированный в формате HTML.

Практикум 10. ФУНКЦИИ

Цель: Закрепление навыков по работе с функциями.

Последовательность выполнения практикума

Практикум состоит из пяти индивидуальных заданий.

Первые два задания заключаются в письменном ответе на два контрольных вопроса, полученных от преподавателя. Список контрольных вопросов приведен в конце практикума.

При выполнении третьего задания необходимо создать функцию с фиксированным числом входных и выходных формальных параметров. Все формальные параметры являются обязательными. При ошибочном количестве и классе фактических параметров во время вызова функции в командное окно необходимо выводить соответствующее сообщение об ошибке. (В качестве типового примера при создании функции можно использовать описание функции, приведенное в листинге 6.15.)

Четвертое задание заключается в разработке функции, которая имеет обязательные и дополнительные формальные параметры. (При выполнении задания в качестве типовых примеров можно использовать примеры 6.17 и 6.18.)

Пятое задание посвящено созданию функции, которая позволяет работать с произвольным числом входных формальных параметров. (В качестве типовых примеров рекомендуется использовать примеры 6.19 и 6.20.)

При выполнении индивидуальных заданий уравнения соответствующих кривых и поверхностей, а также все необходимые данные взять из практикумов 6 и 7.

По итогам выполнения практикума необходимо подготовить отчет, который должен содержать письменные ответы на контрольные вопросы, описания, вызовы и результаты выполнения соответствующих функций.

Вариант 1

3. Создать функцию, вычисляющую значения декартовых координат лемнискаты Бернуlli при изменении параметра t . Входными формальными параметрами в функцию являются параметр t и коэффициент a , выходными параметрами – декартовые координаты x и y .
4. Разработать функцию, определяющую координаты точки пересечения прямой линии L и плоскости Q . Обязательными входными формальными параметрами являются координаты направляющего вектора линии (первый входной параметр) и координаты начальной точки (второй входной параметр). Дополнительными входными формальными параметрами служат коэффициенты $A(2)$, $B(-6)$, $C(13)$ и $D(1)$ общего уравнения

ния плоскости Q . В скобках указаны значения по умолчанию. Общее число входных параметров в функцию – шесть.

5. Создать функцию, которая выполняет построение лемнискаты Бернулли в графическом окне. С помощью входных параметров передается необходимый вид линии (цвет, толщина, тип и т.д.).

Вариант 2

3. Создать функцию, вычисляющую расстояние от точки четырехлепестковой розы до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой осуществляется построение цилиндрической поверхности с направляющей в виде трехлепестковой розы. Обязательными входными параметрами служат коэффициент a и угол φ (вектор), дополнительными параметрами – длина образующей поверхности $L(2)$ и угол поворота $\alpha(30^\circ)$ относительно оси x в градусах. В скобках указаны значения по умолчанию.
5. Создать функцию, которая выполняет построение в графическом окне цилиндрической поверхности из задания 4. С помощью входных параметров передается необходимый вид поверхности (наличие или отсутствие линий, соединяющих узловые точки, прозрачность и т.д.).

Вариант 3

3. Создать функцию, вычисляющую расстояние от точки улитки Паскаля до полюса. Входными параметрами функции являются угол φ и коэффициенты a и b , выходным параметром – расстояние r .
4. Разработать функцию, определяющую угол между линиями L_1 и L_2 . Первый входной параметр является обязательным и задает координаты линии L_1 . Второй параметр, являющийся дополнительным, определяет линию L_2 . Так как линия L_1 задается с помощью общего уравнения, то первый параметр является матрицей 4×2 , где первый столбец содержит коэффициенты первой плоскости, а второй столбец – коэффициенты второй плоскости. Линия L_2 задается с помощью параметрических уравнений, и, следовательно, второй входной параметр представляет собой матрицу 3×2 , в которой первой столбец содержит координаты направляющего вектора, а второй столбец – координаты начальной точки. Значение по умолчанию для второго параметра: $[-1, 2; -1, 0; -2, -1]$.
5. Создать функцию, которая выполняет построение в графическом окне улитки Паскаля. С помощью входных параметров передается необходимый вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 4

3. Создать функцию, определяющую значения декартовых координат *парabolы Нейля* на основе значения параметра t . Входным параметром функции является параметр t , выходными параметрами – значения декартовых координат x и y .
4. Разработать функцию, при выполнении которой осуществляется вычисление расстояния между плоскостью Q , заданной с помощью уравнения в отрезках, и тремя точками. Первый обязательный входной параметр содержит координаты точек пересечения плоскости с соответствующими координатными осями. Остальные три входных параметра являются дополнительными и служат для передачи значений координат соответствующей точки. По умолчанию координаты точек имеют следующие значения: $A(-1;2;1)$, $B(-2;-3;1.5)$ и $C(2;1;-1)$.
5. Создать функцию, при выполнении которой определяются расстояния между плоскостью Q и точками. Плоскость Q задана с помощью уравнения в отрезках. Первый входной параметр содержит координаты точек пересечения плоскости Q с соответствующими осями. Каждый последующий входной параметр содержит координаты одной точки. Выходом является вектор-столбец, значения элементов которого равны расстоянию от плоскости Q до соответствующей точки.

Вариант 5

3. Создать функцию, вычисляющую расстояние от точки *спирали Архимеда* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, осуществляющую построение в графическом окне плоскости Q_1 , которая задана с помощью общего уравнения. Функция принимает четыре входных значения и не возвращает ни одного значения. Функция имеет один обязательный входной параметр, соответствующий коэффициенту A в общем уравнении плоскости, и три дополнительных входных параметра, которые соответствуют остальным коэффициентам. Значения коэффициентов по умолчанию имеют вид $B = -1$, $C = -1$ и $D = -0.75$.
5. Создать функцию, которая выполняет построение в графическом окне плоскости Q_2 , заданной с помощью точки и вектора, перпендикулярного плоскости. Первый входной параметр содержит координаты точки, второй параметр – координаты вектора, остальные входные параметры определяют внешний вид координатного пространства (цвет осей, наличие или отсутствие координатной сетки и т.д.). Выходные параметры у функции отсутствуют.

Вариант 6

3. Создать функцию, вычисляющую расстояние от точки *конхоиды Никомеда* до полюса. Входными параметрами функции являются угол φ и коэффициенты a и d , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение в графическом окне прямой линии L , заданной с помощью параметрических уравнений, и плоскости Q , заданной с помощью точки и вектора, перпендикулярного плоскости. Обязательный входной параметр содержит информацию о линии, а дополнительный входной параметр – информацию о поверхности. Обязательный входной параметр является матрицей 3×2 , первый столбец которой содержит значения координат направляющего вектора, а второй столбец – значения координат начальной точки. Дополнительный параметр также является матрицей 3×2 , первый столбец которой содержит значения координат точки, а второй столбец – значения координат вектора, перпендикулярного плоскости. Значение по умолчанию для дополнительного параметра равно $[-1, 1; 2, -2; 2, 1.5]$.
5. Создать функцию, которая выполняет построение в графическом окне плоскости Q , заданной с помощью точки и вектора, перпендикулярного плоскости. Первый входной параметр содержит координаты точки, второй параметр – координаты вектора, остальные входные параметры определяют внешний вид графического окна (текст в строке заголовка, размеры, цветовая палитра и т.д.). Выходные параметры у функции отсутствуют.

Вариант 7

3. Создать функцию, определяющую значения декартовых координат точки *инволюты окружности*. Входными параметрами функции являются угол θ и радиус окружности r , выходными параметрами – значения координат x и y .
4. Разработать функцию, при выполнении которой происходит построение цилиндрической поверхности с направляющей в виде *улитки Паскаля*. Обязательными входными параметрами являются коэффициенты a и b , а также угол φ , которые входят в полярное уравнение улитки Паскаля. Дополнительным параметром является длина образующей поверхности, которая по умолчанию равна 4.
5. Создать функцию, которая выполняет построение цилиндрической поверхности из предыдущего задания в графическом окне. Первый обязательный параметр содержит все необходимые данные для построения поверхности, все остальные входные данные определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 8

3. Создать функцию, вычисляющую расстояние от точки *кохлеоиды* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой в графическом окне происходит построение поверхности вращения. Поверхность вращения образуется с помощью вращения правой части нефроиды, определенной в плоскости Oyz , вокруг оси z . Функция принимает два входных параметра – обязательный и дополнительный. Обязательный параметр соответствует коэффициенту a . Дополнительный параметр используется для передачи в функцию значений углов поворота вокруг соответствующих осей. По умолчанию углы поворота имеют следующие значения: $a = 30^\circ$, $\beta = 0^\circ$ и $\gamma = 0^\circ$.
5. Создать функцию, которая выполняет построение поверхности вращения из предыдущего задания в графическом окне. Первый обязательный параметр содержит все необходимые данные для построения поверхности, все остальные входные данные определяют внешний вид координатного пространства (наличие координатной сетки, цвет и т.д.).

Вариант 9

3. Создать функцию, с помощью которой можно определить значение координаты *у локона Аньези*. Входными параметрами функции являются координата x и коэффициент a , выходным параметром – координата y .
4. Разработать функцию, при выполнении которой в графическом окне происходит построение *эллипсоида*. Функция принимает три входных параметра – два обязательных и один дополнительный. Первый обязательный параметр содержит координаты центра эллипса, а второй обязательный параметр – значения полуосей. Дополнительный параметр используется для передачи в функцию значений углов поворота эллипса вокруг соответствующих осей. По умолчанию углы поворота имеют следующие значения: $a = 30^\circ$, $\beta = 0^\circ$ и $\gamma = 45^\circ$.
5. Создать функцию, которая выполняет построение *эллипсоида* в графическом окне. Первый обязательный параметр содержит все необходимые данные для построения поверхности, все остальные входные данные определяют внешний вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 10

3. Создать функцию, определяющую значение координаты *у цепной линии*. Входными параметрами функции являются координата x и коэффициент a , выходным параметром – координата y .

4. Разработать функцию, при выполнении которой происходит построение *однополостного гиперболоида*. Функция принимает три входных параметра, первые два из которых являются обязательными, а третий – дополнительным. Первый параметр содержит координаты центра поверхности, второй параметр – значения полуосей и высоту гиперболоида, третий параметр – значения углов вращения в градусах вокруг соответствующих осей. Третий параметр по умолчанию равен $[0, 45, 0]$, что соответствует повороту вокруг оси u на 45° .
5. Создать функцию, которая выполняет построение *однополостного гиперболоида* в графическом окне. Входные параметры в функцию определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 11

3. Создать функцию, вычисляющую расстояние от точки *гиперболической спирали* до полюса. Входными параметрами функции являются угол θ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение *двуполостного гиперболоида*. Функция принимает три входных параметра, первые два из которых являются обязательными, а третий – дополнительным. Первый параметр содержит значения величин a , b и c , второй параметр – высоту каждой полости гиперболоида, третий параметр – значения углов вращения в градусах вокруг соответствующих осей. Третий параметр по умолчанию равен $[45, 0, 0]$, что соответствует повороту вокруг оси x на 45° .
5. Создать функцию, которая выполняет построение *двуполостного гиперболоида* в графическом окне. Входные параметры в функцию определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 12

3. Создать функцию, с помощью которой можно определить значения декартовых координат x и y произвольной точки *трактризы*. Входными параметрами функции являются угол θ и радиус окружности r , выходными параметрами – значения координат x и y .
4. Разработать функцию, при выполнении которой происходит построение *эллиптического параболоида*. Функция принимает два входных параметра. Первый параметр содержит значения величин p и q , а также высоты параболоида, второй дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно $[0, 0]$.

тельного параметра по умолчанию равно $[0, 0, 60]$, что соответствует повороту вокруг оси z на 60° .

5. Создать функцию, которая выполняет построение *эллиптического параболоида* в графическом окне. Входные параметры в функцию определяют внешний вид координатного пространства (наличие координатной сетки, цвет и т.д.).

Вариант 13

3. Создать функцию, вычисляющую расстояние от точки *спирали Кейли* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение *гиперболического параболоида*. Функция принимает два входных параметра. Первый параметр содержит значения величин p и q , а также высоты параболоида, второй дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно $[0, 0, 0]$.
5. Создать функцию, которая выполняет построение *гиперболического параболоида*. Входные параметры в функцию определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 14

3. Создать функцию, вычисляющую расстояние от точки *логарифмической спирали* до полюса. Входными параметрами функции являются угол φ и коэффициенты a и b , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение *конуса второго порядка*. Функция принимает два входных параметра. Первый параметр содержит значения величин a , b и c , а также высоты конуса h , второй дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно $[45, 0, 0]$.
5. Создать функцию, которая выполняет построение *конуса второго порядка*. Входные параметры в функцию определяют внешний вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 15

3. Создать функцию, с помощью которой можно определить значения декартовых координат x и y произвольной точки *дельтоида*. Входными параметрами функции являются параметр t и коэффициент a , выходными параметрами – значения координат x и y .

4. Разработать функцию, при выполнении которой в графическом окне происходит построение поверхности вращения. Поверхность вращения образуется с помощью вращения линии L , определенной в плоскости Oxy , вокруг оси y . Линия L задана с помощью полярного уравнения $r = a \cos 2\varphi$, где $\varphi \in [-\pi/2; \pi/2]$. Функция принимает два входных параметра – обязательный и дополнительный. Обязательный параметр соответствует коэффициенту a . Дополнительный параметр используется для передачи в функцию значений углов поворота вокруг соответствующих осей. По умолчанию углы поворота имеют следующие значения: $\alpha = 0^\circ$, $\beta = 0^\circ$ и $\gamma = 45^\circ$.
5. Создать функцию, которая выполняет построение поверхности вращения из предыдущего задания в графическом окне. Первый обязательный параметр содержит все необходимые данные для построения поверхности, все остальные входные данные определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 16

3. Создать функцию, с помощью которой можно определить значения декартовых координат x и y произвольной точки *нефроиды*. Входными параметрами функции являются параметр t и коэффициент a , выходными параметрами – значения координат x и y .
4. Разработать функцию, при выполнении которой происходит построение однополостного гиперболоида. Функция принимает три входных параметра. Первый параметр является обязательным, а второй и третий параметры – дополнительными. Первый параметр содержит координаты центра поверхности, второй параметр – значения полуосей и высоту гиперболоида, третий параметр – значения углов вращения в градусах вокруг соответствующих осей. Второй параметр по умолчанию равен $[2, 1.5, 3, 5]$, что соответствует $a = 2$, $b = 1.5$, $c = 3$ и $h = 5$. Третий параметр по умолчанию равен $[0, 0, 30]$, что соответствует повороту вокруг оси z на 30° .
5. Создать функцию, которая выполняет построение однополостного гиперболоида в графическом окне. Входные параметры в функцию определяют внешний вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 17

3. Создать функцию, с помощью которой можно определить значения декартовых координат x и y произвольной точки *строфоиды*. Входными параметрами функции являются параметр t и коэффициент a , выходными параметрами – значения координат x и y .

-
4. Разработать функцию, при выполнении которой происходит построение *двуполостного гиперболоида*. Функция принимает три входных параметра. Первый параметр является обязательным, а второй и третий – дополнительными. Первый параметр содержит значения величин a , b и c , второй параметр – высоту каждой полости гиперболоида, третий параметр – значения углов вращения в градусах вокруг соответствующих осей. Второй параметр по умолчанию равен [3, 3]. Третий параметр по умолчанию равен [0, 0, 60], что соответствует повороту вокруг оси z на 60° .
 5. Создать функцию, которая выполняет построение *двуполостного гиперболоида* в графическом окне. Входные параметры в функцию определяют внешний вид координатного пространства (наличие координатной сетки, цвет и т.д.).

Вариант 18

3. Создать функцию, вычисляющую расстояние от произвольной точки *циклоиды Диоклеса* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение *эллиптического параболоида*. Функция принимает два входных параметра. Первый параметр содержит значения величин p и q , а также высоты параболоида, второй дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно [30, 0, 0], что соответствует повороту вокруг оси x на 30° .
5. Создать функцию, которая выполняет построение *эллиптического параболоида* в графическом окне. Входные параметры в функцию определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 19

3. Создать функцию, вычисляющую расстояние от произвольной точки *спирали Ферма* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение *гиперболического параболоида*. Функция принимает три входных параметра. Первый параметр содержит значения величин p и q , второй параметр – значение высоты параболоида, третий дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно [0, 30, 0], что соответствует повороту поверхности вокруг оси u на 30° .

5. Создать функцию, которая выполняет построение гиперболического параболоида в графическом окне. Входные параметры в функцию определяют внешний вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 20

3. Создать функцию, вычисляющую расстояние от произвольной точки спирали Галилея до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение конуса второго порядка. Функция принимает три входных параметра. Первый параметр содержит значения величин a , b и c , второй – значение высоты конуса h , третий дополнительный параметр – значения углов вращения в градусах вокруг соответствующих осей. Значение дополнительного параметра по умолчанию равно $[0, 0, 60]$, что соответствует повороту поверхности вокруг оси z на угол, равный 60° .
5. Создать функцию, которая выполняет построение конуса второго порядка в графическом окне. Входные параметры в функцию определяют внешний вид координатного пространства (наличие координатной сетки, цвет и т.д.).

Вариант 21

3. Создать функцию, вычисляющую расстояние от произвольной точки листа щавеля до полюса. Входным параметром функции является угол φ , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой в графическом окне происходит построение эллипсоида. Функция принимает три входных параметра – один обязательный и два дополнительных. Обязательный параметр содержит координаты центра эллипсоида. Первый дополнительный параметр определяет значения полуосей, а второй – значения углов поворота относительно соответствующих осей. Значения полуосей по умолчанию: $a = 3$, $b = 1.5$ и $c = 1$. Углы поворота по умолчанию: $\alpha = 0^\circ$, $\beta = 25^\circ$ и $\gamma = 0^\circ$.
5. Создать функцию, которая выполняет построение эллипсоида в графическом окне. Входные параметры в функцию определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Вариант 22

3. Создать функцию, с помощью которой можно определить значения декартовых координат x и y произвольной точки декартова листа. Вход-

ными параметрами функции являются параметр t и коэффициент a , выходными параметрами – значения координат x и y .

4. Разработать функцию, при выполнении которой происходит построение прямой линии в пространстве. Функция имеет три входных параметра. Первый обязательный входной параметр соответствует параметру t в параметрических уравнениях линии L . Второй и третий параметры, которые являются дополнительными, содержат значения координат направляющего вектора и начальной точки соответственно. По умолчанию значения координат начальной точки – $(2.5; 1.5; 0)$, координат направляющего вектора – $(-2.5; 0.5; 1)$.
5. Создать функцию, которая выполняет построение прямой линии из предыдущего задания в графическом окне. Входные параметры определяют внешний вид графического окна (текст в строке заголовка, размеры и т.д.).

Вариант 23

3. Создать функцию, вычисляющую расстояние от произвольной точки *трехлепестковой розы* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение цилиндрической поверхности с направляющей в виде *дельтоида*. Обязательными входными параметрами являются коэффициент a и параметр t , которые входят в параметрические уравнения дельтоида. Дополнительным параметром является длина образующей поверхности, которая по умолчанию равна 5.
5. Создать функцию, которая выполняет построение цилиндрической поверхности из предыдущего задания в графическом окне. Входные параметры определяют внешний вид координатного пространства (наличие сетки, цвет и т.д.).

Вариант 24

3. Создать функцию, вычисляющую расстояние от произвольной точки *трисектрисы Маклорена* до полюса. Входными параметрами функции являются угол φ и коэффициент a , выходным параметром – расстояние r .
4. Разработать функцию, при выполнении которой происходит построение куба в координатном пространстве. Функция принимает два параметра. Первый параметр содержит координаты вершин куба, а второй и третий параметры, которые являются дополнительными, – данные для цвета граней и углов поворота соответственно. По умолчанию грань 1 окрашивается в красный цвет, грань 2 – в зеленый, грань 3 – в синий, грань 4 – в желтый, грань 5 – в голубой и грань 6 – в малиновый цвет. Значения углов поворота по умолчанию: $\alpha = 30^\circ$, $\beta = 45^\circ$ и $\gamma = 60^\circ$.

5. Создать функцию, которая выполняет построение куба в графическом окне. Входные параметры определяют внешний вид графического окна.

Вариант 25

3. Создать функцию, вычисляющую расстояние от произвольной точки *трилистника* до полюса. Входным параметром функции является угол φ , выходным параметром – расстояние r .
4. Разработать функцию, во время выполнения которой происходит построение *эллиптического тора* в координатном пространстве графического окна. Функция принимает три параметра. Первый входной параметр служит для передачи в функцию координат центра эллиптического тора. Второй и третий параметры являются дополнительными. Второй параметр содержит информацию о значениях полуосей эллипса (a и b) и радиусе направляющей окружности (r), третий параметр – значения углов поворота (α , β и γ) относительно соответствующих осей. Значения по умолчанию для дополнительных параметров: $a = 1$, $b = 1$, $r = 3$, $\alpha = \beta = 0^\circ$ и $\gamma = 30^\circ$.
5. Создать функцию, которая выполняет построение *эллиптического тора* в графическом окне. Входные параметры определяют внешний вид поверхности (наличие сетки, соединяющей узловые точки, прозрачность и т.д.).

Контрольные вопросы

При ответе на вопрос необходимо привести пример.

1. Стили и языки программирования.
2. Виды m-файлов.
3. Особенности структурного программирования.
4. Структура единственного выбора if end.
5. Структура двойного выбора if else end.
6. Структура тройного выбора if elseif else end.
7. Структура множественного выбора switch case end.
8. Структура повторения for end.
9. Структура повторения while end.
10. Функции break() и continue().
11. Описание и вызов функции.
12. Классы памяти и области видимости переменных.
13. Параметры функции.
14. Инициализация входных формальных параметров.
15. Обязательные и дополнительные входные формальные параметры.
16. Задание произвольного числа входных формальных параметров.
17. Задание произвольного числа выходных формальных параметров.

Практикум 11. ГЕОМЕТРИЧЕСКИЕ ФРАКТАЛЫ

Цель: Приобретение навыков рекурсивного решения задач и знакомство с геометрическими фракталами.

Фракталы

Понятие *фрактал* было впервые введено в 1975 г. членом совета исследовательского центра фирмы IBM Бенуа Мандельбротом. Слово «фрактал» (*fractal*) образовано от латинского слова *fractus*, что означает *дробный*. Определение фрактала, данное Мандельбротом, звучит следующим образом: «Фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому». В 1980 г. было получено графическое изображение множества, представленное на рис. П-11.1, которое вследствии получило имя Мандельброта. Рождение фрактальной геометрии принято связывать с выходом в 1982 году книги Бенуа Мандельброта «The Fractal Geometry of Nature» («Фрактальная геометрия природы»). В этой книге систематизированы труды многих известных математиков, работавших в период 1875–1925 гг., включая Кантора, Пеано, Пуанкаре, Серпинского, Хаусдорфа.

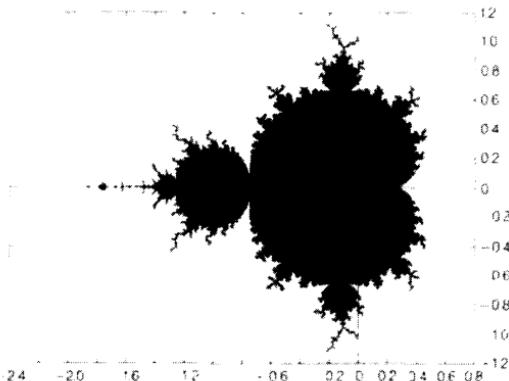


Рис. П-11.1. Множество Мандельброта

В настоящее время фракталы наиболее широко применяются в машинной графике для создания искусственных форм облаков, рельефов местностей и поверхности моря, сжатия изображений, исследования поведения нелинейных динамических систем, формирования декоративных изображений на плоскости и в пространстве и др. [44, 45, 56, 59]. Фракталы, получаемые с помощью электронных вычислительных машин, принято делить на три группы: алгебраические, геометрические и стохастические [12]. Первые две группы являются детерминированными фракталами, а последняя группа относится к недетерминированным.

Алгебраические фракталы являются самой большой группой фракталов и получаются с помощью нелинейных процессов в п-мерных пространствах. Множество Мандельброта является одной из разновидностей алгебраических фракталов. Более подробную информацию об алгебраических фракталах можно найти в [45].

Стохастические фракталы получаются при внесении в итерационный процесс помех с желаемыми стохастическими параметрами. Эта группа фракталов применяется при моделировании рельефа местности, поверхности моря и формы облаков. Информацию о создании стохастических фракталов можно посмотреть в [44].

Геометрические фракталы строятся на основе исходной фигуры путем ее дробления и выполнения аффинных (поворот и/или перемещение) и логических преобразований (объединение или исключение). При построении геометрических фракталов можно использовать как рекурсивный, так и итерационный метод. Как правило, рекурсивный метод проще и нагляднее в программировании, но требует значительных вычислительных ресурсов по сравнению с итерационным методом.

Независимо от метода построения фракталов у них есть одно общее свойство, которое называется *фрактальной размерностью* D . Фрактальная размерность определяется с помощью следующей формулы:

$$D = \log_{k_i} k_N,$$

где $k_N = \lim_{i \rightarrow \infty} \left(\frac{N_{i+1}}{N_i} \right)$ – предельный коэффициент увеличения числа элементов,

$k_i = \frac{\delta_i}{\delta_{i+1}}$ – коэффициент уменьшения разрешения, i – шаг; N_i – количество элем-

ментов на i шаге, N_{i+1} – количество элементов на $i+1$ шаге, δ_i – минимальный размер элемента на i шаге, δ_{i+1} – минимальный размер элемента на $i+1$ шаге.

В том случае, если фрактальная размерность совпадает с числом степеней свободы (1 – линейный элемент, 2 – поверхностный элемент и 3 – объемный элемент), то фрактал обладает постоянной длиной, площадью или объемом соответственно. Таким образом, фрактальная размерность не обязательно должна быть дробной.

Если размерность больше числа степеней свободы (например, $D > 2$ в случае плоскостного геометрического фрактала), то длина, площадь или объем соответственно неограниченно возрастают. Такие фракталы называются *аддитивными*.

Если размерность меньше числа степеней свободы (например, $D < 1$ в случае линейного геометрического фрактала), то длина, площадь или объем стремятся к нулю. Фракталы этого класса называются *субтрактивными*.

Самостоятельная работа

1. Создать m-файл, в котором размещаются две функции. Первичная функция принимает в качестве входного параметра число, которое соответствует максимальной глубине рекурсивных вызовов. Во время выполнения первичной функции создается графическое окно, оформляется координатная плоскость и происходит вызов рекурсивной функции. Подфункция является рекурсивной функцией, которая выполняет построение геометрического фрактала согласно индивидуальному варианту. В том случае, если первичная функция вызывается без входного параметра, то он по умолчанию равен двум. Процедуру построения геометрического фрактала следует выполнять до тех пор, пока не будет достигнута необходимая глубина рекурсивных вызовов или текущая длина линий, из которых формируется необходимая фигура, не будет меньше минимально допустимой длины L_{MIN} . Значение минимально допустимой длины следует хранить в глобальной переменной L_MIN , которая удаляется из памяти во время окончания работы первичной функции.
2. Создать профили функций при различных значениях входного параметра.
3. Создать отчет. В отчет необходимо поместить задание на практикум, описание первичной функции и рекурсивной подфункции, построенные геометрические фракталы первого, второго и третьего порядков, профили функций при построении геометрических фракталов соответствующих порядков и ответ на контрольный вопрос, полученный от преподавателя.

Вариант 1. Построить треугольник (салфетку) Серпинского, представленный на рис. П-11.2. Исходной фигурой является равносторонний треугольник ABC, который разбивается на четыре одинаковых треугольника. Центральный из полученных треугольников отбрасывается. Каждый из оставшихся треугольников вновь разбивается на четыре равных треугольника и т.д. Длина стороны исходного треугольника $L = 1$. Минимальная длина $L_{MIN} = 0.025$.

Вариант 2. Построить квадрат (ковер) Серпинского, представленный на рис. П-11.3. Исходной фигурой является квадрат ABCD, который разбивается на девять одинаковых квадратов. Центральный из полученных девяти квадратов отбрасывается. Каждый из восьми оставшихся квадратов вновь разбивается на девять равных квадратов и т.д. Координаты точек исходного квадрата: A(-1;-1), B(1;-1), C(1;1) и D(-1;1). Минимальная длина стороны треугольника $L_{MIN} = 0.025$.

Вариант 3. Построить линию Серпинского, представленную на рис. П-11.4. Исходной фигурой является «крест» со сторонами A, B, C и D. Каждая сторона исходной фигуры состоит из трех последовательных линий, соединенных между собой под углом 120° . Длина каждой линии $L = 1$. Соседние стороны соединяются между собой отрезками a (A; B), b (B; C),

c (C; D) и *d* (D; A), длина которых равна L . При образовании следующего уровня линии Серпинского одна из сторон заменяется на четыре соответствующие последовательные стороны исходной фигуры. Например, сторона A заменяется на $A + B + D + A$.

Вариант 4. Построить модифицированную линию Серпинского, изображенную на рис. П-11.5. Исходной фигурой является «крест» со сторонами A, B, C и D. Каждая из сторон состоит из четырех последовательных линий, соединенных между собой под углом 90° . Длина каждой линии $L = 1$. Соседние стороны соединяются между собой горизонтальными или вертикальными отрезками *a* (A; B), *b* (B; C), *c* (C; D) и *d* (D; A), длина которых равна L . При создании модифицированной линии Серпинского следующего порядка одна из сторон заменяется на пять сторон исходной линии. Например, сторона C заменяется на $C + D + A' + B + C$, где A' – соответствует обратной последовательности действий, используемой при построении стороны A.

Вариант 5. Построить вторую модифицированную линию Серпинского, изображенную на рис. П-11.6. Исходной фигурой является «крест» со сторонами A, B, C и D. Каждая из сторон состоит из четырех последовательных линий, соединенных между собой под углом 90° . Длина каждой линии $L = 1$. Соседние стороны соединяются между собой горизонтальными или вертикальными отрезками *a* (A; B), *b* (B; C), *c* (C; D) и *d* (D; A), длина которых равна L . При создании второй модифицированной линии Серпинского следующего порядка одна из сторон заменяется на пять сторон исходной линии. Например, сторона C заменяется на $C + D + C + B + C$.

Вариант 6. Построить линию, названную «драконом» Хартера–Хейтуэя, изображенную на рис. П-11.7. Исходная линия состоит из двух отрезков: AB и BC, соединенных между собой под углом 90° . Длина отрезка $L = 1$. При построении линии следующего уровня каждый отрезок заменяется на два отрезка, соединенных между собой под углом 90° . Например, отрезок AB заменяется на два отрезка: AD и DB, где $\angle ADB = 90^\circ$. Минимально допустимая длина отрезков, образующих линию, – $L_{MIN} = 0.05$.

Вариант 7. Построить линию Пеано, показанную на рис. П-11.8. Исходная линия представляет собой отрезок AB, длина которого $L = 1$. При построении следующего уровня исходный отрезок заменяется на девять отрезков: AC, CD, DE, EF, FC, CH, HG, GF и FB, длина которых в 3 раза меньше длины исходной линии. Минимально допустимая длина линии $L_{MIN} = 0.025$.

Вариант 8. Построить модифицированную линию Пеано, представленную на рис. П-11.9. Исходная линия является отрезком AB, длина которого $L = 1$. При построении следующего уровня исходный отрезок заменяется на восемь отрезков: AC, CD, DE, EF, FG, GH, HC и FB, длина которых в 3 раза меньше длины исходной линии. Минимально допустимая длина линии $L_{MIN} = 0.025$.

Вариант 9. Построить замкнутую кривую Гилберта, изображенную на рис. П-11.10. Под замкнутой кривой Гилберта понимается линия, состоящая из двух кривых Гилберта, которые являются зеркальными отражениями друг друга относительно вертикальной оси. Кривая Гилберта первого порядка состоит из трех отрезков: АВ, ВС и СД, длина которых $L = 1$. Кривая Гилберта второго порядка состоит из четырех кривых Гилберта первого порядка, соединенных между собой, как показано на рис. П-11.10. Кривая Гилберта третьего порядка состоит из четырех кривых Гилберта второго порядка и т.д.

Вариант 10. Построить «снежинку» Коха, показанную на рис. П-11.11. Базисом «снежинки» является равносторонний треугольник АВО, длина стороны которого $L = 1$. При построении «снежинки» следующего уровня сторона треугольника разбивается на три отрезка (АС, СД и ДВ) и вместо среднего отрезка выполняется построение двух отрезков (СЕ и ЕД), которые являются боковыми сторонами равностороннего треугольника СЕД. Следовательно, сторона исходного треугольника заменяется на четыре линии, длина которых в 3 раза меньше ее длины. Минимальная длина линии $L_{\min} = 0.01$.

Вариант 11. Построить «снежинку» Коха, показанную на рис. П-11.12. Базисом «снежинки» является квадрат АВОО', длина стороны которого $L = 1$. При построении «снежинки» следующего уровня сторона квадрата разбивается на три отрезка (АС, СД и ДВ) и вместо среднего отрезка выполняется построение трех отрезков (СЕ, ЕФ и ФД), которые являются сторонами квадрата СЕФД. Следовательно, сторона исходного квадрата заменяется на пять линий, длина которых в 3 раза меньше ее длины. Минимальная длина линии $L_{\min} = 0.01$.

Вариант 12. Построить геометрический фрактал, представленный на рис. П-11.13. Исходной фигурой является окружность с радиусом $R = 1$. При создании следующего уровня вокруг окружности строятся четыре окружности (А, В, С и D), радиус которых уменьшается согласно золотому сечению $R_2 / R_1 = (\sqrt{5} - 1) / 2$. Минимальный радиус окружности $R_{\min} = 0.05$.

Вариант 13. Построить модифицированную «снежинку» Коха, изображенную на рис. П-11.14. Базисом модифицированной «снежинки» является равносторонний треугольник АВО, длина стороны которого $L = 1$. При построении модифицированной «снежинки» следующего уровня сторона треугольника разбивается на три отрезка (АС, СД и ДВ). Средний отрезок является основанием нового равностороннего треугольника, вершина которого находится за пределами исходного треугольника. Минимальная длина стороны треугольника $L_{\min} = 0.025$.

Вариант 14. Построить модифицированную «снежинку» Коха, изображенную на рис. П-11.15. Базисом модифицированной «снежинки» является квадрат АВОО', длина стороны которого $L = 1$. При построении мани-

фицированной «снежинки» следующего уровня сторона квадрата разбивается на три отрезка (AC, CD и DB). Средний отрезок (CD) является стороной нового квадрата, который располагается за пределами исходного квадрата. Минимальная длина стороны квадрата $L_{MIN} = 0.025$.

Вариант 15. Построить геометрический фрактал, показанный на рис. П-11.16. Исходной фигурой является пятиугольник ABDEF, длины сторон которого $L = 1$. При создании следующего уровня на сторонах исходного пятиугольника строятся новые пятиугольники – A'B'D'E'F', длины сторон которых в 3 раза меньше длины исходной для пятиугольника стороны. Минимальная длина стороны пятиугольника $L_{MIN} = 0.025$.

Вариант 16. Построить геометрический фрактал, изображенный на рис. П-11.17. Исходной фигурой является шестиугольник ABDEFG, длины сторон которого $L = 1$. При создании следующего уровня на сторонах исходного шестиугольника строятся новые шестиугольники – A'B'D'E'F'G', длины сторон которых в 3 раза меньше длины исходной для шестиугольника стороны. Минимальная длина стороны шестиугольника $L_{MIN} = 0.025$.

Вариант 17. Построить геометрический фрактал, который представлен на рис. П-11.18. Исходной фигурой является «звезда», состоящая из пяти линий (AC, BC, DC, EC и FC), соединяющихся в точке C. Длина этих линий $L = 1$. Угловое расстояние между соседними линиями составляет 72° . При построении следующего уровня в точках A, B, D, E и F создаются новые «звезды», длина линий которых в 3 раза меньше длины линий текущего уровня. Минимальная длина линий $L_{MIN} = 0.025$.

Вариант 18. Построить геометрический фрактал, который изображен на рис. П-11.19. Исходной фигурой является «звезда», состоящая из восьми линий (AC, BC, DC, EC, FC, GC, HC и IC), соединяющихся в точке C. Длина этих линий $L = 1$. Угловое расстояние между соседними линиями составляет 45° . При построении следующего уровня в точках A, B, D, E, F, G, H и I создаются новые «звезды», длина линий которых в 3 раза меньше длины линий текущего уровня. Минимальная длина линий $L_{MIN} = 0.025$.

Вариант 19. Построить «фрактальную звезду», показанную на рис. П-11.20. Исходной фигурой является квадрат ABCD, сторона которого $L = 1$. При переходе к следующему уровню в угловых точках A, B, C и D создаются новые квадраты, центры которых совпадают с соответствующими точками. Длина стороны нового квадрата в 2 раза меньше стороны квадрата текущего уровня. После создания всех квадратов нового уровня они объединяются с квадратами предыдущих уровней. Минимальная длина стороны создаваемых квадратов $L_{MIN} = 0.05$.

Вариант 20. Построить геометрический фрактал, изображенный на рис. П-11.21. Геометрический фрактал представляет собой «фрактальную

звезды», созданную с помощью квадратов без их объединения (см. вариант 19). Исходной фигурой является квадрат ABCD, в угловых точках которого создаются новые квадраты, площадь которых в 4 раза меньше площади исходного квадрата. Центры новых квадратов располагаются в соответствующих угловых точках исходного квадрата, длина стороны которого $L = 1$. Минимальная длина стороны создаваемых квадратов $L_{MIN} = 0.05$.

Вариант 21. Построить геометрический фрактал, который изображен на рис. П-11.22. Геометрический фрактал представляет собой модификацию «фрактальной звезды», созданную с помощью квадратов без их объединения (см. варианты 19 и 20). В данном случае угловые точки исходного квадрата ABCD, в которых создаются новые квадраты, являются не центрами новых квадратов, а одной из их угловых точек. Центр создаваемого квадрата и исходного должны лежать на одной прямой с их общей точкой. Длина стороны исходного квадрата $L = 1$. Длины сторон квадратов следующего уровня в 2 раза меньше, чем длины сторон квадратов текущего уровня. Минимальная длина сторон $L_{MIN} = 0.05$.

Вариант 22. Построить геометрический фрактал, который представлен на рис. П-11.23. В основе этого фрактала лежит равносторонний треугольник ABC, длина стороны которого $L = 1$. При переходе к следующему уровню необходимо в угловых точках исходного треугольника построить новые треугольники таким образом, чтобы их вершины совпадали с соответствующими угловыми точками и центр каждого треугольника был расположен на одной линии с центром исходного треугольника и их общей точкой. Длина сторон треугольников при переходе от одного уровня к другому уменьшается пропорционально золотому сечению ($L_{i+1}/L_i = (\sqrt{5}-1)/2$). Минимальная длина стороны треугольника $L_{MIN} = 0.05$.

Вариант 23. Построить геометрический фрактал, который изображен на рис. П-11.24. Геометрический фрактал является «цветком», построенным с помощью пятиугольников. Основанием фрактала служит пятиугольник ABCDE. При создании следующего уровня в угловых точках исходного пятиугольника строятся новые пятиугольники, длины сторон которых в 2.5 раза меньше, чем длины сторон текущего пятиугольника. Длины сторон исходного пятиугольника $L = 1$. Центры каждого пятиугольника нового уровня лежат на одной прямой с центром пятиугольника предыдущего уровня и общей для них точкой. Минимальная длина стороны пятиугольника $L_{MIN} = 0.015$.

Вариант 24. Построить геометрический фрактал, который представлен на рис. П-11.25. Геометрический фрактал является «цветком», состоящим из шестиугольников различных площадей. На нулевом уровне геометрического фрактала располагается шестиугольник ABCDEF, в угловых точках которого строятся шестиугольники следующего уровня. Центры каждого из шестиугольников следующего уровня лежат на одной прямой

с центром шестиугольника текущего уровня и общей для них точкой. Длины стороны исходного шестиугольника $L = 1$. Длины сторон шестиугольника следующего уровня в 2.7325 раза меньше, чем длины сторон шестиугольника текущего уровня. Минимальная длина стороны $L_{MIN} = 0.01$.

Вариант 25. Построить геометрический фрактал, который представлен на рис. П-11.26. Основой геометрического фрактала является квадрат ABCD. При создании следующего уровня вокруг квадрата, согласно рис. П-11.26, создаются восемь квадратов. Длина стороны квадрата следующего уровня в 3.1 раза меньше длины стороны квадрата текущего уровня. Центры квадратов следующего уровня располагаются относительно центра квадрата текущего уровня на расстоянии $d = 1.25 \times L$, где L – длина стороны квадрата текущего уровня. Длина стороны квадрата ABCD $L_0=1$. Минимальная длина стороны квадрата $L_{MIN} = 0.01$.

Контрольные вопросы

- Что такое рекурсия? Достоинства и недостатки использования рекурсивных функций при решении задач. При ответе необходимо привести пример.
- Базовая задача, рекурсивный спуск и подъем. В качестве примера рассмотрите рекурсивную функцию вычисления определителя.
- Рекурсивный и итерационный подходы к решению задачи.
- Что такое фрактал? Какие существуют группы фракталов? Привести примеры использования фракталов.
- Свойства геометрического фрактала.

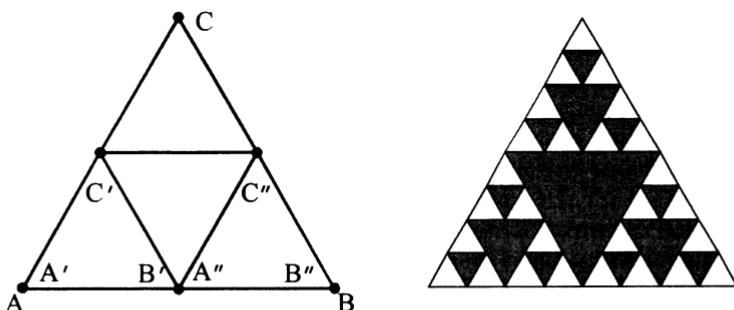


Рис. П-11.2. Треугольник Серпинского

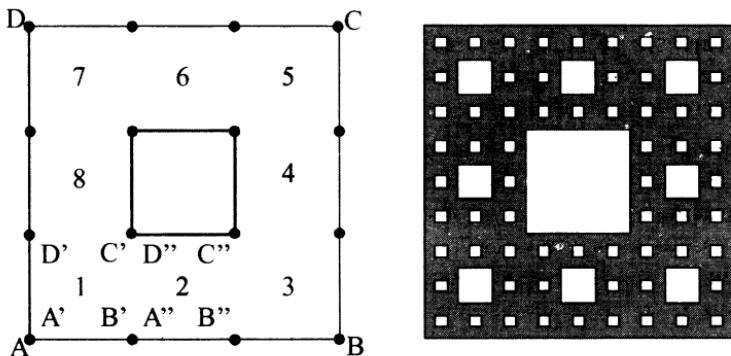


Рис. П-11.3. Ковер Серпинского

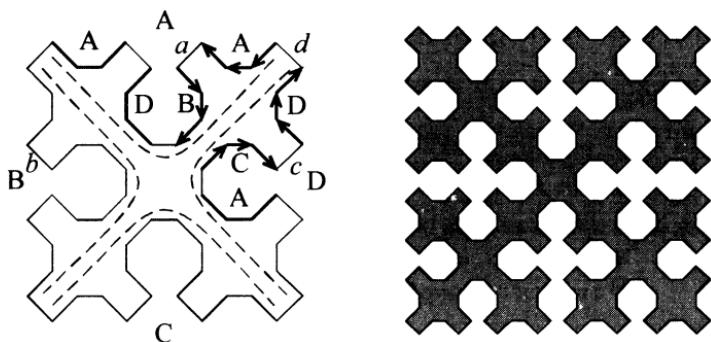


Рис. П-11.4. Линия Серпинского

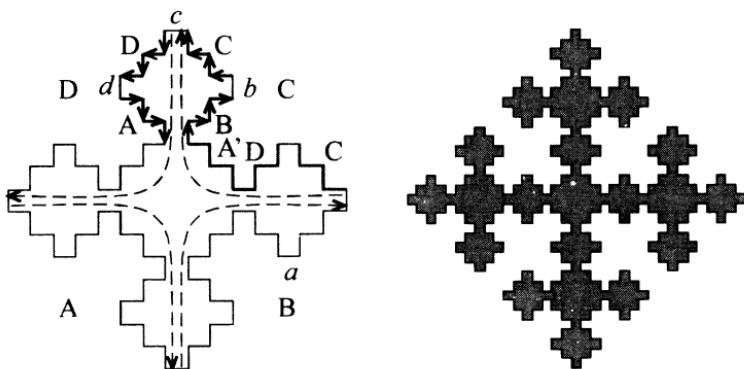


Рис. П-11.5. Модифицированная линия Серпинского

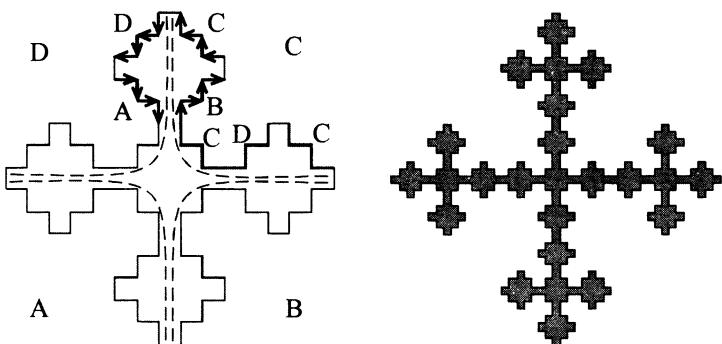


Рис. П-11.6. Вторая модифицированная линия Серпинского

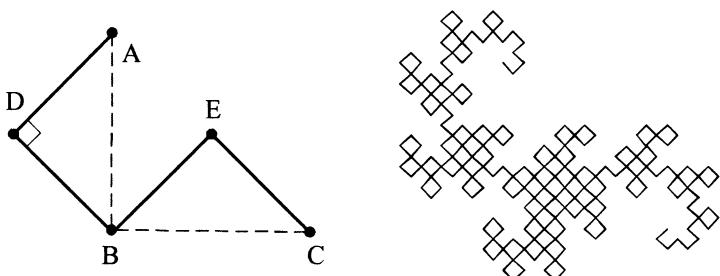


Рис. П-11.7. «Дракон» Хартера–Хейттэя

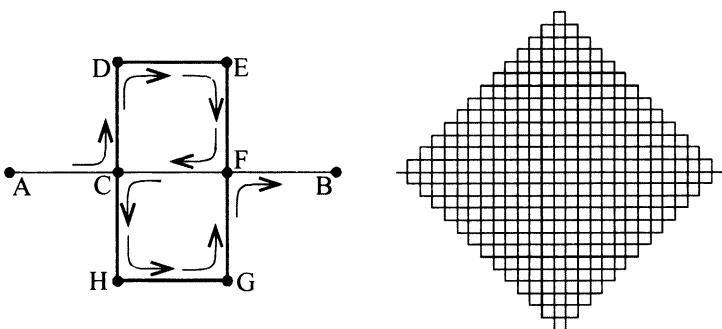


Рис. П-11.8. Линия Пеано

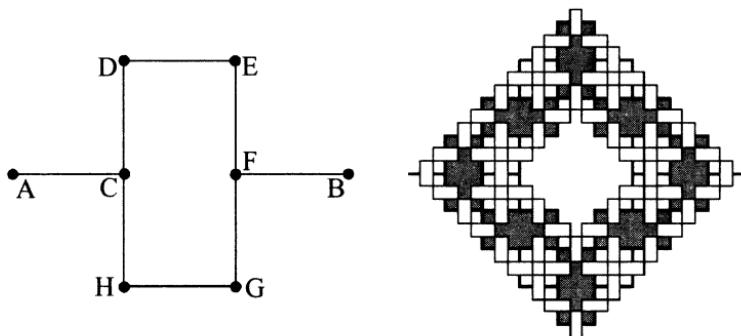


Рис. П-11.9. Модифицированная линия Пеано

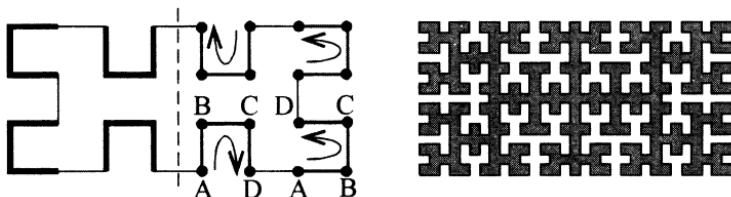


Рис. П-11.10. Замкнутая кривая Гилберта

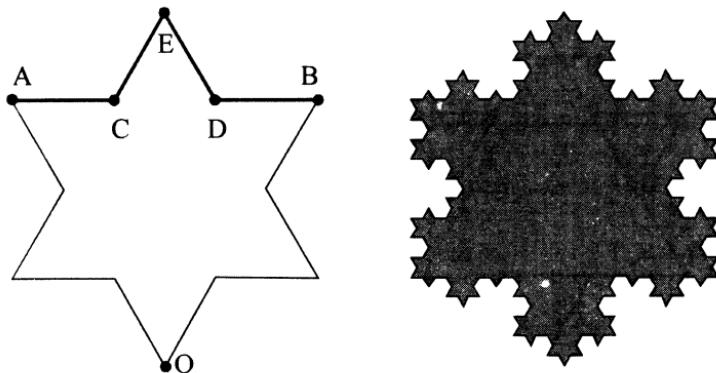


Рис. П-11.11. «Снежинка» Коха с основанием в виде треугольника

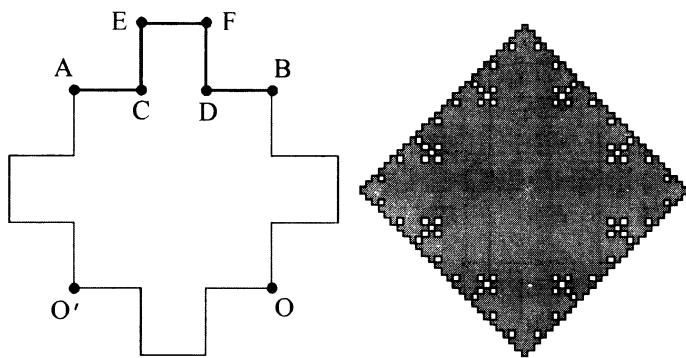


Рис. П-11.12. «Снежинка» Коха с основанием в виде квадрата

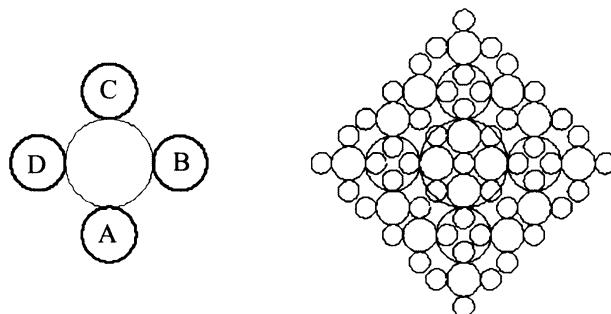


Рис. П-11.13. Окружности

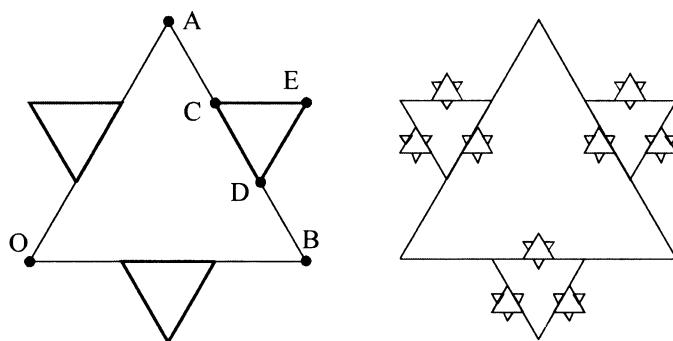


Рис. П-11.14. Модификация «снежинки» Коха с основанием в виде треугольника

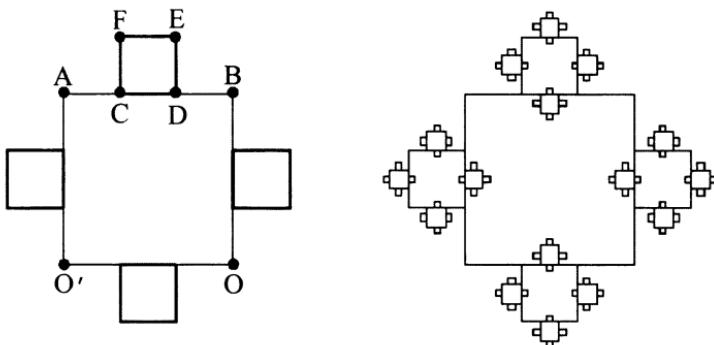


Рис. П-11.15. Модификация «снежинки» Коха
с основанием в виде квадрата

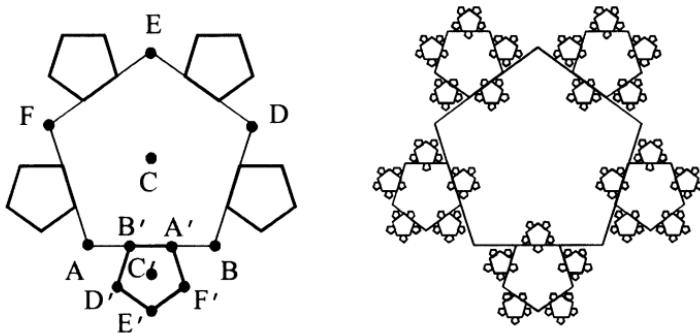


Рис. П-11.16. Пятиугольники

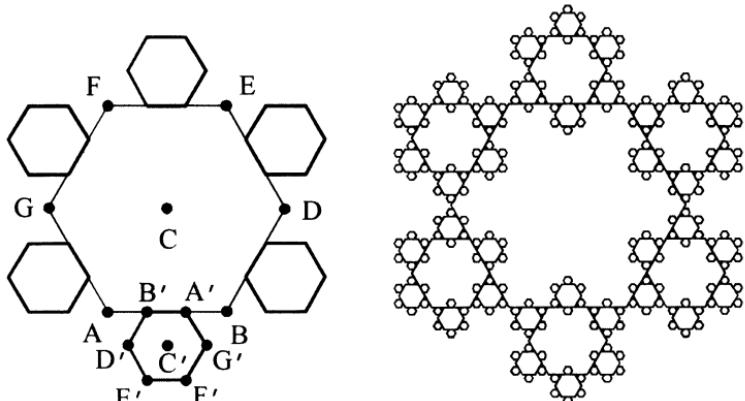


Рис. П-11.17. Шестиугольники

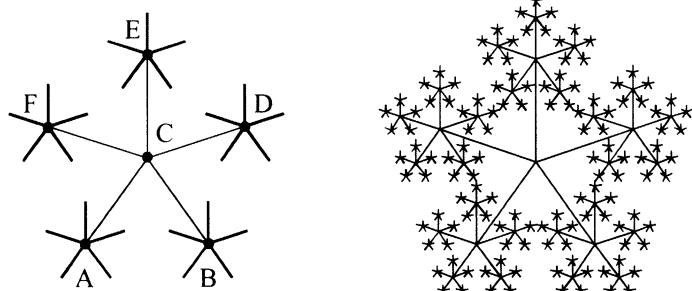


Рис. П-11.18. Пятиконечная «звезда» в виде линий

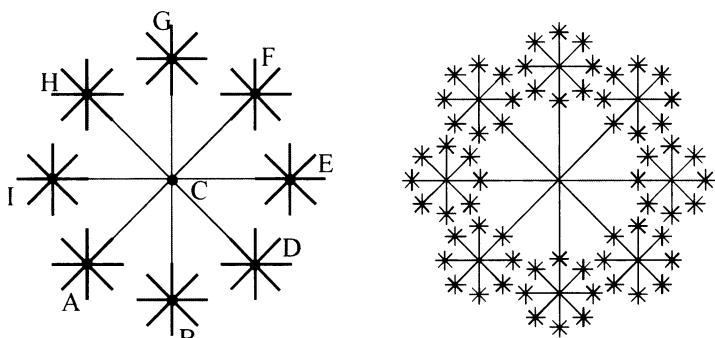


Рис. П-11.19. «Снежинка»

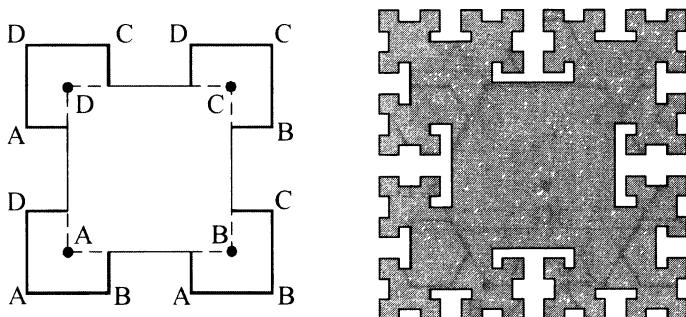


Рис. П-11.20. «Фрактальная звезда»

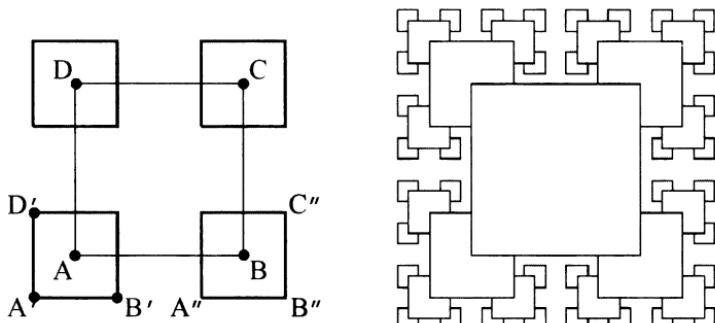


Рис. П-11.21. «Фрактальная звезда» в виде квадратов

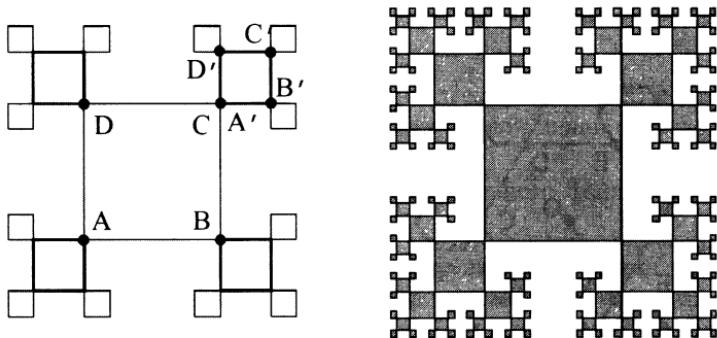


Рис. П-11.22. Модификация «фрактальной звезды» в виде квадратов

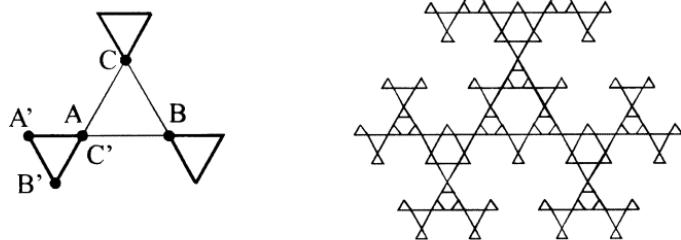


Рис. П-11.23. «Цветок» из треугольников

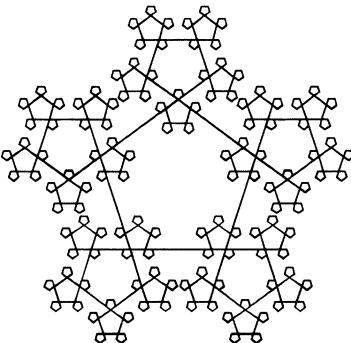
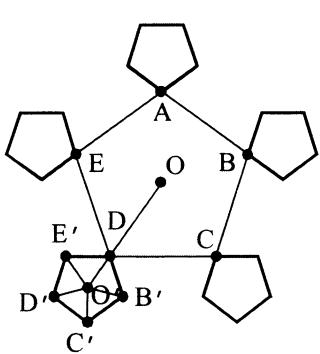


Рис. П-11.24. «Цветок» из пятиугольников

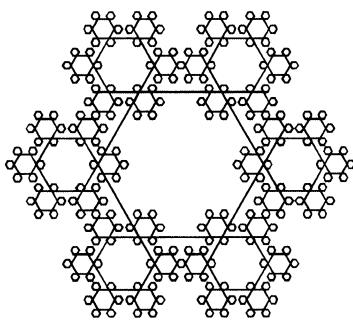
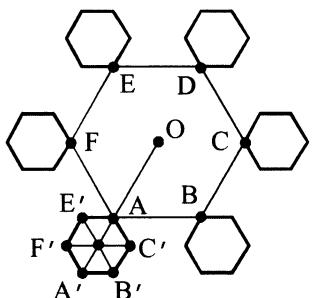


Рис. П-11.25. «Цветок» из шестиугольников

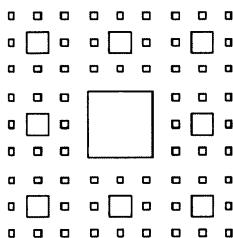
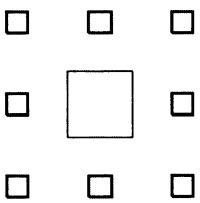


Рис. П-11.26. Квадраты

Практикум 12. ФАЙЛЫ

Цель: Приобретение навыков работы с текстовыми и двоичными файлами в системе MATLAB.

Порядок выполнения практикума

В данном практикуме необходимо выполнить три задания.

Варианты индивидуальных заданий для первого и второго заданий необходимо взять из практикумов 6, 7 согласно своему варианту.

Первое задание:

- разработать функцию для формирования текстового файла, содержащего форматированную таблицу исходных данных для построения кривой линии в декартовой системе координат;
- написать функцию для форматированного ввода данных из созданного файла и построения графика.

Второе задание:

- разработать функцию для формирования двоичного файла, содержащего таблицу исходных данных для построения кривой линии в полярной системе координат;
- написать функцию для ввода данных из созданного файла и построения графика.

Третье задание связано с разработкой функции обработки данных или созданием текстового либо двоичного файлов в зависимости от номера варианта. Индивидуальные варианты третьего задания приведены в конце практикума.

По итогам выполнения практикума необходимо создать отчет, в котором должны находиться задания к практикуму, листинги соответствующих функций, содержимое созданных текстовых файлов и ответ на вопрос, полученный от преподавателя.

Типовые примеры

Пример П-12.1. Разработать функцию, при выполнении которой создается текстовый файл, содержащий таблицу значений аргумента $x \in [-\pi; \pi]$ с шагом $\Delta x = \pi/25$ и значений функции $y = \cos x$.

Решение:

Листинг П-12.1

```
| function cosxtbl( xleft, xright, xstep, filename )  
| %COSXTBL - формирование в текстовом файле таблицы косинусов  
| % Таблица формируется на отрезке [XLEFT;XRIGHT]  
| % с шагом XSTEP
```

Продолжение листинга П-12.1

```
% filename - строка, содержащая имя создаваемого текстового файла

% проверка количества фактических параметров
if nargin ~= 4
    error( 'Функция должна иметь четыре фактических параметра!' )
end

% проверка принадлежности имени файла к символьному классу
if ~ischar( filename )
    error('Параметр "filename" должен принадлежать к классу CHAR!')
end

% открытие файла на запись
fid = fopen( filename, 'wt' );
if fid == -1
    error( 'Некорректное создание файла!' )
end

% наименования столбцов создаваемой таблицы
nametbl = 'Таблица функции y=cos(x)';
namex = 'X';
namecosx = 'COS(X)';

% формирование "шапки" таблицы
fprintf( fid, '-----\n' );
fprintf( fid, '|%26s |\\n', nametbl );
fprintf( fid, '-----\n' );
fprintf( fid, '| %6s | %8s |\\n', namex, namecosx );
fprintf( fid, '-----\n' );

% вычисление данных для заполнения таблицы в виде столбцов
% для формирования столбца применена операция транспонирования
x = xleft:xstep:xright'; cosx = cos( x );

% вывод данных в таблицу файла (данные формируются как матрица из
% двух столбцов)
fprintf( fid, '|%10.4f |%10.4f |\\n', [x; cosx] );

% завершение формирования таблицы
fprintf( fid, '-----\n' );
```

Окончание листинга П-12.1

```
% закрытие файла (обязательная операция для корректного сохранения данных)
fclose( fid );
```

Для корректной работы разрабатываемой функции необходимо выполнить проверку правильности передачи параметров. В данном случае проверяется количество передаваемых параметров и принадлежность имени файла к символьному классу данных. Если количество параметров не равно 4 (*if nargin ~= 4*) или имя файла не является строкой (*if ~ischar(filename)*), то порождается сообщение об ошибке и дальнейшее выполнение прекращается (это прекращение автоматически выполняет функция *error()*). Первая проверка не является обязательной, так как MATLAB автоматически проверяет корректность обращения к функции с фиксированным числом параметров.

При открытии файла (несмотря на то, что файл создается) также необходимо выполнить проверку (*if fid == -1*). Дело в том, что имя файла может содержать указание на несуществующие носители или директории.

Данные для заполнения таблицы формируются в виде двух столбцов, для чего они объединяются в матрицу. Полученная матрица и выводится в файл.

Обязательной при работе с файлами является операция закрытия файла. Если создаваемый файл после вывода данных не закрыть, то либо вся информация, либо часть ее будет потеряна. Это связано с тем, что между логическим и физическим файлом имеется буфер обмена. Информация вначале помещается в буфер обмена и только после его заполнения – в файл. При закрытии файла, даже если буфер обмена не заполнен, информация из него помещается в файл. Но если файл не закрыть, то информация из незаполненного буфера обмена в него не передается. Это и является причиной потери информации. В рассматриваемом примере если файл не закрыть, то по завершении он остается пустым.

Обращение к разработанной функции можно осуществить из командного окна или файла сценария. Последовательность команд в командном окне имеет вид

```
>> xleft = -pi;
>> xright = pi;
>> xstep = pi/25;
>> filename = 'tcosx.txt';
>> cosxtbl( xleft, xright, xstep, filename )
```

В результате выполнения приведенных выше команд будет создан файл следующего вида (часть информации не приводится):

Таблица функции $y=\cos(x)$		
X	COS(X)	
-3.1416	-1.0000	
-3.0159	-0.9921	
...	...	
3.0159	-0.9921	
3.1416	-1.0000	

Пример П-12.2. Разработать функцию, которая считывает текстовый файл, созданный в примере П-12.1. После прочтения из файла данных отобразить их в графическом окне.

Решение:

Листинг П-12.2

```
function [ x, y ] = tbl2num( filename )
%TBL2NUM - преобразование таблицы в массивы чисел
% с использованием форматированного ввода данных из файла
% filename - имя файла, который содержит таблицу данных
% x, y      - считанные из таблицы данные
%              (представляются в виде векторов)

% Проверка принадлежности имени файла к символьному классу
if ~ischar( filename )
    error( 'Ошибка: имя файла должно быть строкой!' )
end

% Существующий файл открывается для чтения данных
fid = fopen( filename, 'rt' );

% Проверка корректности открытия файла
if fid ==-1 % файл открыт некорректно
    error( 'Ошибка при открытии файла!' )
end

% Определение количества строк в файле
strcount = 0;
while ~feof( fid )
```

Окончание листинга П-12.2

```
strcount = strcount + 1;
fgetl( fid );
end

% Закрываем файл
fclose( fid );
% И открываем его вновь, чтобы маркер записи установить в начало файла
fid = fopen( filename, 'rt' );

% rownumber - число строк, которые необходимо считать из таблицы,
% на 6 меньше, чем число строк в файле (5 - "шапка" файла
% и 1 - последнее подчеркивание)
rownumber = strcount - 6;

% Пропуск первых пяти строк файла (чтение "шапки" таблицы)
for k = 1:5
    fgetl( fid );
end

% Считывание данных из таблицы
[num, count] = fscanf( fid, '%*s %f %*s %f %*s', [2 rownumber] );

if count ~= 2*rownumber
    error( 'Ошибка при чтении данных из файла!' );
else
    % В случае корректного чтения разбиваем прочитанные данные на два
    % массива: абсцисс и ординат; это и есть возвращаемые параметры
    % функции
    x = num(1,:);
    y = num(2,:);
end

% Закрытие файла
fclose( fid );
```

При форматированном вводе для чтения чисел из созданного ранее файла можно использовать формат `%g` вместо `%f`. Символ `«*»` в формате чтения обеспечивает пропуск символьных данных (символов `|`), которые делают таблицу наглядной. Эти данные прочитываются при обращении к файлу, но возвращаемые результаты (`[num, count]`) не помещаются (игнорируются).

Реализовать обращение к разработанной функции и осуществить построение графика можно посредством следующих команд:

```
>> filename = 'tcosx.txt';
>> [x y] = tbl2num( filename );
>> plot(x,y), grid on
>> xlabel('x'), ylabel('y'), title('y = cos(x)')
```

В результате выполнения будет построен график функции $y = \cos x$, изображенный на рис. П-12.1.

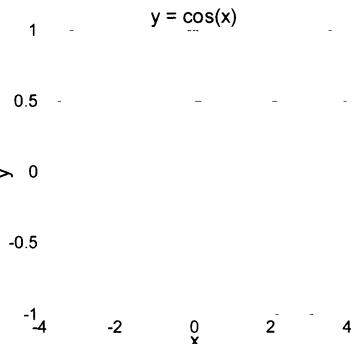


Рис. П-12.1. График функции $y = \cos x$

Пример П-12.3. Разработать функцию, при выполнении которой создается двоичный файл, содержащий таблицу значений аргумента $x \in [-\pi; \pi]$ с шагом $\Delta x = \pi/25$ и значений функции $y = \cos x$.

Решение:

Листинг П-12.3

```
function cosxtbl_bin( xleft, xright, xstep, filename )
%COSXTBL_BIN - формирование в двоичном файле таблицы косинусов
% Таблица формируется на отрезке [XLEFT;XRIGHT]
% с шагом XSTEP
% filename - строка, содержащая имя создаваемого двоичного файла

% проверка количества фактических параметров
if nargin ~= 4
    error( 'Функция должна иметь четыре фактических параметра!' )
end

% проверка принадлежности имени файла к символьному классу
if ~ischar( filename )
```

Окончание листинга П-12.3

```

error('Параметр "filename" должен принадлежать к классу CHAR!')
end

% создание и открытие двоичного файла на запись
fid = fopen( filename, 'w' );
if fid == -1
    error( 'Некорректное создание файла!' )
end

% вычисление данных для заполнения таблицы в виде столбцов
x = xleft:xstep:xright'; % для формирования столбца применена
% операция транспонирования
cosx = cos( x );

% запись полученных данных в двоичный файл
fwrite( fid, [x; cosx], 'single' );

% закрытие файла
fclose( fid );

```

Обращение к разработанной функции можно осуществить из командного окна. Последовательность команд в командном окне имеет вид

```

>> xleft = -pi; xright = pi; xstep = pi/25;
>> filename = 'tcosx.dat';
>> cosxtbl_bin( xleft, xright, xstep, filename )

```

Пример П-12.4. Разработать функцию, которая считывает двоичный файл, созданный в примере П-12.3. После прочтения из файла данных отобразить их в графическом окне.

Решение:

Листинг П-12.4

```

function [ x, y ] = tbl2num_bin( filename )
%TBL2NUM - преобразование таблицы в массивы чисел
%           с использованием ввода данных из двоичного файла
% filename - имя файла, который содержит таблицу данных
% x, y     - считанные из таблицы данные

% Проверка принадлежности имени файла к символьному классу
if ~ischar( filename )

```

```
    error( 'Ошибка: имя файла должно быть строкой!' )
end

% Существующий файл открывается для чтения данных
fid = fopen( filename, 'rt' );

% Проверка корректности открытия файла
if fid == -1 % файл открыт некорректно
    error( 'Ошибка при открытии файла!' )
end

% Открываем двоичный файл (при этом маркер записи устанавливается
% в начало файла перед первой записью)
fid = fopen( filename, 'r' );
status = 0; x = []; y = [];

% Задаем имя класса, в котором сохранены данные в файле
class_name = 'single';
% и определяем размер этого класса в байтах
class_size = 4;

% Считывание данных из файла сначала в массив x
while ( ~feof( fid ) ) & ( status ~= -1 )
    % Читаем одно значение абсциссы и добавляем его в массив
    x = [x fread( fid, 1, class_name )];
    % и переводим маркер через одно к следующему значению абсциссы
    status = fseek( fid, class_size, 'cof' );
end

rewind( fid ); % Возвращаем маркер в исходное положение

% Теперь считывание данных в массив ординат
while ~feof( fid )
    % переводим маркер через одно к следующему значению ординаты
    status = fseek( fid, class_size, 'cof' );
    % и читаем одно значение ординаты и добавляем его в массив
    if status ~= -1
        y = [y fread( fid, 1, class_name )];
    else
        break
    end
```

Окончание листинга П-12.4

```

end

% Закрытие файла
fclose( fid );

```

При изучении функции чтения данных из двоичного файла следует обратить внимание на заголовки и тела циклов `while end`. Различие в них объясняется тем, что в файле данных значения абсцисс и ординат чередуются: сначала абсцисса, затем ордината, затем опять абсцисса и т.д. Следовательно, при чтении абсцисс (вектор x) необходимо сначала прочесть значение, а затем передвинуть маркер записи, а при чтении ординат – наоборот.

Построить график на основе данных, находящихся в двоичном файле, можно с помощью следующих команд:

```

>> filename = 'tcosx.dat';
>> [x y] = tbl2num_bin( filename );
>> plot( x, y ), grid on
>> xlabel('x'), ylabel('y'), title('y = cos(x)')

```

В результате вышеприведенных команд будет создан график, который изображен на рис. П-12.1.

Пример П-12.5. Разработать функцию, которая заменяет в текстовом файле все фрагменты «aaa» на «bbb». Текстовый файл имеет следующее содержание:

```

aaa fffff ff ggg aaa hh
qqq eee aaa aaa fff
nnnnn aaa mmmmm
ddddd fffff hhhh jjjjj
sssss aaa aaa rrrrr

```

Существование решения данной задачи состоит в том, чтобы последовательно из исходного файла читать строки, находить в них фрагменты «aaa» и заменять их новым фрагментом: «bbb». Новые строки записывать во временный файл. Если же искомых фрагментов в прочитанной строке нет, то ее записывать во временный файл в исходном состоянии. В конечном счете созданный временный файл скопировать в исходный (заменив в нем, таким образом, информацию), после чего временный файл удалить.

Решение:

Для определения факта вхождения некоторого фрагмента (`substring`) в строку `string` используется функция `findstr(string, substring)`. Результатом выполнения данной функции является вектор, значения ко-

торого равны номерам позиций, начиная с которых искомый фрагмент входит в данную строку. Например, для первой строки исходного файла вектор будет иметь значения 1 и 18 (именно в этих позициях в первой строке располагается первый символ фрагмента «aaa»). Замену старого фрагмента новым необходимо осуществлять с конца строки. В противном случае, если длина заменяющего фрагмента не равна длине заменяемого, индексация внутри строки будет нарушена и замена окажется некорректной.

Листинг П-12.5

```
function change( filename, strold, strnew )
%CHANGE - функция замены во всех строках файла FILENAME
%  всех подстрок STROLD на новые подстроки STRNEW

% Проверка принадлежности входных параметров к классу CHAR
if ~ischar( filename )
    error('Ошибка задания параметра: FILENAME не является строкой!')
end
if ~ischar( strold )
    error('Ошибка задания параметра: STROLD не является строкой!')
end
if ~ischar( strnew )
    error('Ошибка задания параметра: STRNEW не является строкой!')
end

% Открытие файла на чтение
fid_r = fopen( filename, 'rt' );
% Проверка корректности открытия файла
if fid_r == -1
    error('Указано имя несуществующего файла!')
end
% Создаем временный файл для записи модифицированного состояния
fid_w = fopen( 'temp.txt', 'wt' );

while ~feof( fid_r )
    str = fgetl( fid_r ); % Последовательно читаем из файла строки
    % Если длина прочитанной строки не меньше заменяемого фрагмента
    if length( str ) >= length( strold )
        % Определяем вхождение фрагмента в строку
        pos = findstr( strold, str );
        end
        % Если фрагменты в строке есть, то заменяем все эти фрагменты
```

Окончание листинга П-12.5

```

if length( pos ) > 0
    for k = length( pos ) : -1 : 1
        str = [str(1:(pos(k)-1)) strnew ...
                str((pos(k)+length(strold)):length(str))];
    end
    fprintf(fid_w, '%s\n', str)
else
    fprintf(fid_w, '%s\n', str)
end
end

fclose( fid_r ); fclose( fid_w ); % Закрываем оба файла
% Копируем содержимое временного файла в исходный
copyfile('temp.txt', filename)
delete('temp.txt') % удаляем временный файл

```

Обращение к данной функции можно оформить, как и в предыдущих примерах, в виде последовательности команд:

```

>> filename = 'f.dat';
>> strold = 'aaa'; strnew = 'bbb';
>> change( filename, strold, strnew );

```

В результате выполнения программы файл изменит свое содержимое на следующее:

```

bbb ffffff ff ggg bbb hh
qqq eee bbb bbb fff
nnnnn bbb mmmmm
ddddd fffff hhhh jjjjj
sssss bbb bbb rrrrr

```

Индивидуальные варианты

Вариант 1. В текстовом файле содержится непустая последовательность слов, содержащих от 1 до 8 букв. Соседние слова разделены запятой, за последним словом следует точка. Разработать функцию `getword()`, которая извлекает из файла все слова, отличные от последнего слова.

Вариант 2. В текстовом файле содержится непустая последовательность слов, содержащих от 1 до 8 букв. Соседние слова разделены запятой, за последним словом следует точка. Разработать функцию `getminword()`, которая извлекает из файла слово наименьшей длины (или все из них, если таких слов несколько).

Вариант 3. В двоичном файле содержатся числа типа double. Среди этих чисел встречаются группы, образующие возрастающие последовательности. Разработать функцию `maxrow()`, определяющую количество элементов в наиболее длинной последовательности и извлекающую из файла эту последовательность.

Вариант 4. Даны два двоичных файла с числами типа double. Известно, что числа в каждом файле расположены в неубывающем порядке. Разработать функцию `files2one()`, объединяющую эти файлы в один, так, чтобы числа в нем также были расположены в неубывающем порядке.

Вариант 5. Разработать функцию `triangle()`, формирующую текстовый файл из 9 строк, в первой из которых содержится символ '1', во второй – два символа '2', в третьей – три символа '3' и т.д.

Вариант 6. Разработать функцию `line40()`, которая считывает из некоторого текстового файла символы до первой точки и записывает их (без точки) в новый текстовый файл, формируя в нем строки по 40 символов (точка в новый файл не записывается, в последней строке может быть менее 40 символов).

Вариант 7. Разработать функцию `empty()`, которая подсчитывает количество пустых строк в некотором текстовом файле и находит непустую строку минимальной длины.

Вариант 8. Разработать функцию `copynotempty()`, которая переписывает содержимое одного текстового файла в другой, но без пустых строк.

Вариант 9. Некоторый текстовый файл разбит на строки, длина каждой из которых не превосходит 80 символов. Разработать функцию `transform()`, которая, дополняя строки файла пробелами справа, формирует новый текстовый файл, все строки которого имеют длину 80 символов.

Вариант 10. В текстовом файле записана непустая последовательность чисел, разделенных пробелами. Разработать функцию `minpositive()` для нахождения минимального из положительных чисел этого файла.

Вариант 11. В текстовом файле содержится последовательность целых чисел, разделенных пробелами. Разработать функцию `positive()`, записывающую в новый текстовый файл все положительные числа из данного файла.

Вариант 12. Разработать функцию `print()`, которая содержитимо данного файла выводит в командное окно, вставляя в начало каждой строки ее порядковый номер (четырехзначное число с последующим пробелом, например 0012).

Вариант 13. Разработать функцию `tri_pascal()`, выводящую в текстовый файл первые 10 строк треугольника Паскаля в следующем виде:

```

1
1 1
1 2 1
...
1 9 ... 126 126 ... 9 1

```

Вариант 14. Разработать функцию `tri_pascal()`, выводящую в текстовый файл первые 10 строк треугольника Паскаля в следующем виде:

```

      1
      1   1
      1   2   1
      ...
1   9   ...   126   126   ...   9   1

```

Вариант 15. Имеется текстовый файл. Разработать функцию `book()`, которая, игнорируя исходное деление этого файла на строки, переформатирует его, разбивая на строки так, чтобы каждая строка оканчивалась точкой либо содержала ровно 60 символов, если среди них нет точки.

Вариант 16. Имеется текстовый файл. Разработать функцию `short()`, которая выводит в командное окно первую из самых коротких его строк.

Вариант 17. Разработать функцию `sintan()`, формирующую в текстовом файле таблицу значений функций $\sin(x)$ и $\tg(x)$ на отрезке $[0;3]$ с шагом 0.1. Каждая строка содержит три числа: значение аргумента x , значение $\sin(x)$ и значение $\tg(x)$. Значение x выводить с одной цифрой после десятичной точки, значение $\sin(x)$ – с четырьмя, значение $\tg(x)$ – в экспоненциальной форме.

Вариант 18. Имеется двоичный файл, содержащий целые числа. Разработать функцию `middle()`, определяющую среднеарифметическое данных чисел и создающую новый файл, в который переписывает числа из данного файла, вычитая из каждого среднеарифметическое (центрирует данные).

Вариант 19. Имеется некоторый текстовый файл. Разработать функцию `backcopy()`, переписывающую содержимое этого файла в новый файл, размещая при этом строки в обратном порядке.

Вариант 20. Имеется двоичный файл, содержащий целые числа. Разработать функцию `odd()`, переписывающую числа из данного файла в два других текстовых файла в один – нечетные числа, в другой – четные.

Вариант 21. В текстовом файле хранится заранее подготовленная «картинка» (составлена из символов: например, 1 – изображение, 0 – фон). Разработать функцию `invert()`, во время выполнения которой создается негативное изображение «картинки».

Вариант 22. В текстовом файле хранится заранее подготовленная «картинка» (составлена из символов: например, 1 – изображение, 0 – фон).

Создать функцию `rotate90()`, которая поворачивает «изображение» на 90° по часовой стрелке.

Вариант 23. Имеется некоторый текстовый файл, содержащий информацию. Разработать функцию `change(old, new)`, при выполнении которой происходит замена всех символов `old`, переданных в качестве первого входного фактического параметра, на соответствующие символы `new`, переданные в качестве второго входного фактического параметра.

Вариант 24. Имеется некоторый текстовый файл. Создать функцию `code_text()`, которая переписывает информацию из одного текстового файла в другой, зашифровывая ее. Ключ шифра задать самостоятельно (исходный файл содержит информацию только из латинских букв).

Вариант 25. Существует некоторый текстовый файл, в котором помимо прочих находятся строки, содержащие символы #D. Разработать функцию `delete_lines()`, которая удаляет строки, содержащиеся между ними.

Контрольные вопросы

1. Какие существуют типы файлов?
2. Какова последовательность работы с файлами?
3. Структура файла. Метки начала и конца файла. Маркер текущей позиции файла.
4. Основные операции при работе с текстовыми файлами.
5. Форматированный вывод текстовых данных.
6. Форматированный ввод текстовых данных.
7. Основные операции при работе с двоичными файлами.
8. Каким образом осуществляется доступ к данным, хранящимся в двоичных файлах?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Абстракция данных и решение задач на С++. Стены и зеркала. Изд. 3: Пер. с англ. М.: Издательский дом «Вильямс», 2003. – 848 с.: ил.
2. Аладьев В.З., Ваганов В.А., Хунт Ю.Я., Шишаков М.Л. Автоматизированное рабочее место математика. Таллинн–Минск–Москва: Российская академия естественных наук, 1999. – 608 с.
3. Анализ данных на компьютере. Изд. 3, перераб. и доп. / Под ред. В.Э. Фигурнова. М.: ИНФРА-М, 2003. – 544 с.
4. Андриевский Б.Р., Фрадков А.Л. Элементы математического моделирования в программных средах MATLAB 5 и Scilab. СПб.: Наука, 2001. – 286 с.: ил.
5. Ануфриев И.Е. Самоучитель MATLAB 5.3/6.x. СПб.: БХВ-Петербург, 2002. – 736 с.: ил.
6. Баврин И.И., Матросов В.Л. Высшая математика: Учеб. для студ. высш. учеб. заведений. М.: Гуманит. изд. центр ВЛАДОС, 2002. – 400 с.: ил.
7. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы: Учебное пособие. М.: Наука. Гл. ред. физ.-мат. лит., 1987. – 600 с.
8. Боровиков В. STATISTICA «Искусство анализа данных на компьютере: Для профессионалов». Изд. 2. СПб.: Питер. 2003. – 688 с.: ил.
9. Бронштейн И.Н., Семеняев К.А. Справочник по математике для инженеров и учащихся вузов. Изд. 13, испр. М.: Наука. Гл. ред. физ.-мат. лит., 1986. – 544 с.
10. Бююль А., Цёфель П. SPSS: искусство обработки информации: Анализ статистических данных и восстановление скрытых закономерностей: Пер. с нем. СПб.: ООО «ДиаСофт», 2002. – 608 с.
11. Васильев А.Н. Maple 8: Самоучитель. М.: Издательский дом «Вильямс», 2003. – 352 с.: ил.
12. Витолин Д. Применение фракталов в машинной графике. Computer-World-Россия. 1995. №15. С. 11.
13. Говорухин В., Цибулин В. Компьютер в математическом исследовании: Учебный курс. СПб.: Питер, 2001. – 624 с.: ил.
14. Городняя Л.В. Основы функционального программирования: Курс лекций: Учебное пособие. М.: ИНТУИТ.РУ «Интернет-университет Информационных технологий», 2004. – 208 с.
15. Гультьяев А. Визуальное моделирование в среде MATLAB: Учебный курс. СПб.: Питер, 2000. – 432 с.: ил.
16. Гурский Д.А. Вычисления в MathCad. Минск: Новое знание, 2003. – 814 с.: ил.

17. Дантеманн Д., Мишел Д., Тейлор Д. Программирование в среде Delphi: Пер. с англ. / Джейфф Дантеманн, Джим Мишел, Дон Тейлор. Киев: НИПФ «ДиаСофт Лтд.», 1995. – 608 с.
18. Дейтел Х., Дейтел П. Как программировать на С++: Пер. с англ. М.: ЗАО «Издательство БИНОМ», 1998. – 1024 с.: ил.
19. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения: Пер. с англ. М.: Мир, 2001. – 430 с.: ил.
20. Дьяконов В.П. Справочник по применению системы DERIVE. М.: Наука. ФИЗМАТЛИТ, 1996. – 144 с.
21. Дьяконов В.П. Mathematica 4.0: Учебный курс. СПб.: Питер, 2000. – 656 с.
22. Дьяконов В.П. Компьютерная математика. Теория и практика. М.: Нолидж, 2001. – 1296 с.: ил.
23. Дьяконов В.П., Круглов В. Математические пакеты расширения MATLAB: Специальный справочник. СПб.: Питер, 2001.
24. Дьяконов В.П. Системы компьютерной алгебры DERIVE: Самоучитель и руководство пользователя. М.: Солон-Р, 2002. – 320 с.
25. Йодан Э. Структурное проектирование и конструирование программ: Пер. с англ. М.: Мир, 1979. – 415 с.
26. Каханер Д., Моулер К., Нэш С. Численные методы и программное обеспечение: Пер. с англ. Изд. 2, стереотип. М.: Мир, 2001. – 575 с.
27. Кнут Д. Искусство программирования. Т. 1: Основные алгоритмы. Изд. 3: Пер. с англ. М.: Издательский дом «Вильямс», 2001. – 720 с.: ил.
28. Кнут Д. Искусство программирования. Т. 2: Получисленные алгоритмы. Изд. 3: Пер. с англ. М.: Издательский дом «Вильямс», 2003. – 832 с.: ил.
29. Кнут Д. Искусство программирования. Т. 3: Сортировка и поиск. Изд. 2: Пер. с англ. М.: Издательский дом «Вильямс», 2003. – 832 с.: ил.
30. Кнут Д. Все про Тех.: Пер. с англ. М.: Издательский дом «Вильямс», 2003. – 560 с.: ил. (Парал. тит. англ.).
31. Кондрашев В.Е., Королев С.Б. MATLAB как система программирования научно-технических расчетов. М.: Мир; Институт стратегической стабильности Минатома РФ, 2002. – 350 с.: ил.
32. Кудрявцев Е.М. MathCad 2000 Pro. М.: ДМК Пресс, 2001. – 576 с.: ил.
33. Кулаичев А.П. Методы и средства анализа данных в среде Windows STADIA 6.0. М.: Информатика и компьютеры, 1998. – 270 с.
34. Кэнту M. Delphi 4 для профессионалов. СПб: Питер, 1999. – 1120 с.: ил.
35. Лисичкин В.Т., Соловейчик И.Л. Математика: Учебное пособие для техникумов. М.: Высш. шк., 1991. – 480 с.: ил.

36. Лобанова О.В. Практикум по решению задач в математической системе DERIVE: Учебное пособие. М.: Финансы и статистика, 1999. – 544 с.: ил.
37. Львовский С.М. Набор и верстка в системе LaTeX. Изд. 3, испр. и доп. М.: МЦНМО, 2003. – 448 с.
38. Мартынов Н.Н. Введение в MATLAB 6. М.: КУДИЦ-ОБРАЗ, 2002. – 352 с.
39. Матросов А. Maple 6. Решение задач высшей математики и механики. СПб.: БХВ-Петербург, 2001. – 528 с.: ил.
40. Монастырский Л.Ф., Румянцев Д.Г. Путь программиста: опыт создания личности. М.: Издательский дом ИНФРА-М, 2000. – 864 с.: ил.
41. Мэтьюз Д., Финк К. Численные методы. Использование MATLAB. Изд. 3: Пер. с англ. М.: Издательский дом «Вильямс», 2001. – 720 с.: ил.
42. Немнюгин М.А., Стесик О.Л. Современный Фортран: Самоучитель. СПб.: БХВ-Петербург, 2004. – 496 с.: ил.
43. Непейвода Н.Н., Скопин И.Н. Основания программирования. М.; Ижевск: Институт компьютерных исследований, 2003. – 868 с.
44. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. СПб.: БХВ-Петербург, 2003. – 560 с.: ил.
45. Пайтген Х.О., Рихтер П.Х. Красота фракталов. Образы комплексных динамических систем: Пер. с англ. М.: Мир, 1993. – 176 с.: ил.
46. Перминов О.Н. Введение в язык Ада. М.: Радио и связь, 1991. – 288 с.: ил.
47. Пильщиков В.Н. Программирование на языке ассемблера IBM PC. М.: ДИАЛОГ-МИФИ, 1999. – 288 с.
48. Письменный Д.Т. Конспект лекций по высшей математике. Ч. 1. Изд. 2, испр. М.: Айрис-пресс, 2003. – 288 с.: ил.
49. Плис А.И., Сливина Н.А. MathCAD 2000. Математический практикум. М.: Финансы и статистика, 2000.
50. Потемкин В.Г. Система MATLAB: Справочное пособие. М.: ДИАЛОГ-МИФИ, 1997. – 350 с.
51. Потемкин В.Г. MATLAB 6: среда проектирования инженерных приложений. М: ДИАЛОГ-МИФИ, 2003. – 448 с.
52. Потемкин В.Г. Вычисления в среде MATLAB. М.: ДИАЛОГ-МИФИ, 2004. – 720 с.
53. Пратт Т. Языки программирования. Разработка и реализация: Пер. с англ. М.: Мир, 1979. – 574 с.
54. Райс Дж. Матричные вычисления и математическое обеспечение: Пер. с англ. М.: Мир, 1984. – 264 с.: ил.

55. Тюрин Ю.Н., Макаров А.А. Анализ данных на компьютере / Под ред. В.Э. Фигурнова. Изд. 3, перер. и доп. М.: ИНФРА-М, 2003. – 544 с.: ил.
56. Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии: Учебное пособие. М.: Триумф, 2003. – 320 с.: ил.
57. Фаддеев Д.К., Соминский И.С. Сборник задач по высшей алгебре. М.: Наука, 1968.
58. Чен К., Джисбин П., Ирвинг А. MATLAB в математических исследованиях: Пер. с англ. М.: Мир, 2001. – 346 с.: ил.
59. Delphi. Готовые алгоритмы / Род Стивенс / Пер. с англ. П.А. Мерещука. Изд. 2, стер. М.: ДМК Пресс; СПб.: Питер, 2004. – 384 с.: ил.
60. Programming Guide. Maple V Release 4. M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter. 1996. – 383 p.
61. The Maple Handbook. Maple V Release 4. Darren Redfern, 1996. – 495 p.
62. The Student Edition of MATLAB: version 5, user's guide / The MathWorks, Inc.: by Duane Hanselman and Bruce Littlefield, 1997. – 429 p.
63. Using MatLab Version 6, July 2002.

ССЫЛКИ НА САЙТЫ

64. <http://www.borland.com>
65. <http://www.bricklin.com>
66. <http://www.corel.com>
67. <http://www.derive-europe.com>
68. <http://www.exponenta.ru>
69. <http://www.flosim.com>
70. <http://www.frankston.com>
71. <http://www.insightful.com>
72. <http://www.lotus.com>
73. <http://www.maplesoft.com>
74. <http://www.mathsoft.com>
75. <http://www.mathworks.com>
76. <http://www.mscsoftware.com>
77. <http://www.microsoft.com>
78. <http://www.softline.ru>
79. <http://www.spss.com>
80. <http://www.statsoft.com>
81. <http://www.wolfram.com>

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

Абсцисса точки 109, 146
Азимут 151
Алгебраическое дополнение 85
Алгоритм
 Евклида 230
 z-buffer 159
Анимация 223–226
Аппликата точки 146

Б

Базис 55, 56
Базовая задача 273, 274, 276
Блок-схема 210
Буль Джордж 32
Буфер ввода-вывода 284

В

Вектор
 вектор-столбец 57, 73
 вектор-строка 57, 73
 длина вектора 55, 60, 61
 единичный 55
 равные векторы 55
 коллинеарный 55
 компланарный 55
 линейные операции 58–60
 неизвестных 92
 норма 68, 69
 нулевой 55
 перестановок 47
 свободных членов 92
Векторное произведение 64–66
Внешнее произведение 67, 68
Выравнивание текста 141
Выражение
 арифметическое 33

логическое 33
символьное 38, 39

Г

Гипербола
 асимптоты 132
 основной прямоугольник 132
 равносторонняя 132
 уравнение
 каноническое 131
 параметрические 133
 полярное 133
 фокусное расстояние 131
 эксцентриситет 132
Гиперболическая спираль 395, 436
Гиперболический параболоид
 192, 396, 397, 437, 439
Графическое окно
 главное меню 379–383
 контекстно-зависимое меню 386
 панель инструментов «Камера»
 384–386
 стандартная панель инструментов
 383, 384
Греческие буквы 119

Д

Декарт Рене 56
Декомпозиция задачи 201, 202, 236
Дельтоид 396, 437
Дескриптор 373
Дескрипторная графика 371–379
Детерминант 85
Двуполостный гиперболоид
 189, 190, 395, 397, 436, 439
Дизъюнкция 30
Длина вектора 55, 73
«Дракон» Хартера–Хэйтуса 446, 452

Е

Единицы измерения 140

Ж

Жордан Камиль 363

Жордан Уильям 363

И

Идентификатор файла 284

Иерархические массивы структур 41

Иерархическая структура объектов

дескрипторной графики 371

Импорт данных 324–326

Инволюта окружности 394, 434

Индексация

содержимого 49

ячеек 49

Инструментальное средство

Array Editor 320, 336–339

Command History 320, 339–341

Command Window 320, 329–333

Current Directory Brower

320, 341–345

Editor/Debugger 320, 401–410

Help Brower 320, 346–349

Launch Pad 320, 349, 350

Profiler 320, 411–430

Workspace Brower 320, 333–336

Интерпретатор 201

К

Карта прозрачности 188

Квадрат Серпинского 445, 451

Класс

арифметические 25–28

вещественный 26

двойной точности 26

логический 29–32

массив структуры 39–48

массив ячеек 48–53

одинарной точности 26

памяти 246, 247

символьный 33–39

целочисленный 25, 26

Команды системы LaTeX 116, 117, 119

Комментарий 208, 242

Компилятор 201

Компьютерная математика 15

Конхоида Никомеда 394, 434

Конъюнкция 30

Координатная сетка 113, 114

Координаты

вектора 56

точки 56, 113

Кохлеоида 394, 435

Крамер Габриель 96

Л

Лемниската Бернулли 392, 431

Линия

второго порядка 126

Гилберта 447, 453

Пеано 446, 452, 453

параметр 124

первого порядка 113

тип 111

Серпинского 445, 446, 451, 452

угловой коэффициент 114

цвет 111

Лист

Декартов 398, 440

щавеля 398, 440

Логическое индексирование 207,
208, 236

Локон Аньези 395, 435

М

Магический квадрат 79, 80

Максимальное натуральное число 30

Маркер

текущей позиции в файле 285

- типа 110, 111, 126
размера 110, 111, 126
цвета 110, 111, 126
- Массив**
абсцисса 110
нулевой размерности 43
ордината 110
строки 34
- Матрица**
верхняя треугольная 77, 78
вычисление определителя 86
диагональная 77
единичная 77
квадратная 73
коэффициентов системы 92
невырожденная 87, 88
нижняя треугольная 77, 78
норма 91, 92
обратная 88, 89
определение 73
определитель 85–87
перестановок 104, 105
поворота 149
произведение 81, 83–85
равные матрицы 75
ранг 89, 90
расширенная 93
симметричная 78
скалярная 77
след 80
ступенчатая форма 364
сумма 81
транспонирование 76
трапециевидная 78
эквивалентные матрицы 90
элементарные преобразования 82, 83
- Меню**
главное
графического окна 379–383
рабочего стола 319, 321
редактора т-файлов 402–405
контекстно-зависимое
- Редактора т-файлов 407, 408
командного окна 333
окна истории команд 339, 340
окна рабочей области 334, 335
окна редактора данных 338
окна текущей директории 342–344
- Метка**
конца строки 286, 289
конца файла 285
начала файла 285
- Метод**
Гаусса 97–101
Гаусса с частичным выбором главного элемента 101–103
Гаусса–Жордана 363–366
итерационный 93
матричный 94, 95
приближенный 93
прорисовки 159
точный 93
LU-разложение 104, 105
- Минор 85
Модальное диалоговое окно 375
Морган Огастес де 31
- Н**
- Направляющие косинусы 61, 62
Нефроида 394, 396, 435, 438
- Норма**
вектора 68, 69
матрицы 91, 92
- Нормаль** 121
- Нумерация**
элементов в массиве 75
областей графического окна 118
ячеек 50
- О**
- Область видимости 249
Обусловленность 103

- Объект**
- графическое окно 140, 371
 - источник света 372
 - кнопка быстрого доступа 372
 - координатное пространство 136, 371
 - корневой 144, 371
 - линия 372
 - многоугольник 149–151, 372
 - панель инструментов 372
 - поверхность 153, 372
 - прямоугольник 134, 135, 372
 - растровое изображение 197, 198, 372
 - родительский 371, 372
 - текст 140, 372
- Однополостный гиперболоид** 186–188, 395, 396, 436, 438
- Окно**
- графическое 379, 380
 - диалоговое 375–378
 - запуска инструментальных средств 350
 - импорта данных 325
 - истории команд 339
 - командное 330
 - помощи 347
 - Профайлера 411
 - рабочей области 334
 - Редактора данных 337
 - текущей директории 342
- Окружность**
- каноническое уравнение 127
 - концентрическая 111
 - параметрические уравнение 127
- Октант** 56
- Операции**
- аддитивные 32
 - логические
 - короткие 31
 - побитовые 30
 - поэлементные 30
 - сравнения 29
 - мультиплекативные 32
- приоритет** 32, 33
- унарные** 32
- Определитель** 85
- Организация структуры**
- матричная 40
 - поэлементная 40
- Ордината точки** 109, 146
- Ориентация текста** 140, 141
- Опт** 56
- Ось**
- автоматический режим 141
 - большая 129
 - действительная 132
 - диапазон 115
 - засечки 142
 - значения 142
 - масштаб 109, 145
 - малая 129
 - минимая 132
 - направление 145
 - обозначение 114
 - перенос 112, 148
 - поворот 112, 148
 - полярная 109
 - ручной режим 141
 - фиксация диапазона 141
 - цвет 142
- Остаток от деления** 28
- Отчет о профиле файла итоговый** 413–419
- Ошибка округления** 102, 103
- П**
- Пакеты расширения** 21–24
- Палитра цветов** 154, 155
- Панель инструментов**
- графического окна
 - «Камера» 384
 - стандартная 383
 - рабочего стола 328
 - редактора т-файлов 406
 - окна рабочей области 334
 - окна редактора данных 337

- окна текущей директории 341, 342
- Парабола
- директриса 136
 - каноническое уравнение 136
 - Нейля 393, 433
 - фокус 137
- Параметры функции
- передача
 - по значению 253
 - по ссылке 253
 - позиционная связь 252
 - фактические,
 - входные 253
 - выходные 253
 - символические 253
 - формальные
 - входные 252
 - выходные 252
 - дополнительные 254, 258, 259
 - значение по умолчанию 256
 - инициализация 253, 254
 - обязательные 258, 259
 - произвольное число 260, 264
- Паскаль Блез 79
- Пауза 231
- Переменная
- арифметических классов 26
 - глобальная 247
 - класс памяти 246, 247
 - класса массив структуры 41
 - логического класса 29
 - локальная 247
 - локальная сохраняемая 247, 248
 - область видимости 246, 247
 - символьного класса 33
- Поверхность
- вращения 180
 - второго порядка 183
 - первого порядка 152
 - цилиндрическая 178
- Полярный радиус 110
- Полярный угол 110
- Поток 285
- Преобразование
- класса переменных 35,
 - координат 112, 148
 - системы уравнений
 - замещение 98, 101
 - исключение 98, 101
 - масштабирование 98, 101
 - перестановка 98, 101
- Приоритет операций 32, 33
- Произведение
- векторное 64–66
 - внешнее 67, 68
 - матриц 83–85
 - скалярное 62–64
 - смешанное 66, 67
- Прямые линии в пространстве 162
- Псевдокод 246
- P**
- Рабочая область
- глобальная 248
 - основная 247
 - функции 247
- Рабочий стол 319
- Редактор свойств 387–391
- Рекурсия
- подъем 274, 275, 279
 - спуск 274, 275, 277
 - хвостовая 279
- Решение системы уравнений 93
- Роза
- трехлепестковая 393, 398, 432, 441
 - четырехлепестковая 392, 432
- C**
- Свойства
- векторного произведения 65, 66
 - линейных операций
 - над векторами 59, 60
 - линейных операций
 - над матрицами 81, 82
 - нормы вектора 68

- определителя матрицы 86, 87
ранга матрицы 90
скалярного произведения 63, 64
смешанного произведения 67
Сетка поверхности 153, 154
Система
аналитических расчетов 15, 16
координат
полярная 109, 110
прямоугольная 56
сферическая 146, 147
трансформация 148–151
цилиндрическая 146, 147
матричные 15, 16
счисления 232, 238
специальных расчетов 15, 16
статистических расчетов 15, 16
универсальные 15, 17
линейных уравнений
неопределенная 93
несовместная 93
однородная 93
определенная 93
совместная 93, 94
численных расчетов 15
Скалярное произведение 62–64
Смешанное произведение 66, 67
«Снежинка» Коха 447, 453, 454
Сортировка имен полей 46, 47
Специальные символы 119
Сpirаль
Архимеда 393, 433
Галилея 397, 440
Кэйли 396, 437
логарифмическая 396, 437
Ферма 397, 439
Стиль программирования
логический 201, 202
неструктурный 201
объектно-ориентированный
201, 202
структурный 201, 202
функциональный 201, 202
- Столбец
неизвестных 92
свободных членов 92
Строка
быстрой помощи 240
заголовка функции 239
помощи 240
состояния 328, 329, 407
Строфоида 397, 438
Структура
выбора 206
единственного выбора
if end 206–211
двойного выбора
if else end 211–214
тройного выбора
if elseif else end
214–218
 массив структуры 39
множественного выбора switch
case end 218–221
повторения 206
повторения for end 222–228
повторения while end 228–231
следования 206
Сумма по модулю 2 30
Схема вычислений
итеративная 272
рекурсивная 272
Сценарий 204
- Т**
- Таблица символов ASCII 35, 36
Табличный процессор 15
Текст
выравнивание 141
ориентация 141
Теорема Кронекера–Капелли 94
Тор 182, 183
Точка
абсцисса 109, 146
аппликата 146

обзора 151
ордината 109, 146
пересечения линии и плоскости 176, 177
начало координат 109, 146
Трактиса 395, 436
Транслятор 201
Треугольник Серпинского 445, 450
Трилистник 399, 442
Трисектриса Маклорена 398, 441

У

Угол
азимута 151
возвышения 151
между плоскостями 171
между прямыми линиями 174
между прямой линией и плоскостью 175
полярный 110
Улитка Паскаля 393, 394, 432, 434
Уравнение
астроиды 142
гиперболического параболоида
каноническое 192
параметрические 194
гиперболы 131, 133
двуполостного гиперболоида
каноническое 189
параметрические 190
конуса второго порядка 196
однополостного гиперболоида
каноническое 186
параметрические 187
окружности 127
параболы 136
плоскости
в отрезках 157
общее 152
проходящей через точку, перпендикулярно заданному вектору 160

проходящей через три заданные точки 156
прямой линии
в отрезках 118
общее 113
параметрические 124
полярное 122, 123
с угловым коэффициентом 114
проходящей через точку
в заданном направлении 115
проходящей через точку
перпендикулярно заданному вектору 121
проходящей через две точки 117
эллипса 128, 129
эллипсоида
каноническое 184
параметрические 185
эллиптического параболоида 190

Ф

Файл
двоичный 303
дисковый 283
доступ 286, 287
закрытие файла 284, 287
запись в файл 284, 293, 304
логический 284
последовательного доступа 284
произвольного доступа 309
открытие файла 284, 286, 304
текстовый 285
физический 284
 чтение из файла 284, 299, 304
Факториал числа 273–275
Фокус
гиперболы 131
параболы 136
эллипса 128
Формат числовых данных 331, 332
Формулы Крамера 95–97

Фрактал 443, 444
Фрактальная размерность 444
Функции
 арифметические 27
 вызов функции 242
 гиперболические 27
 главная 270
 логарифмические 27
 округления 28
 описание функции 237
 подфункции 270–272
 реквизиты 240
 сравнения 29
 степенные 27, 28
 тело 241
 тригонометрические 27
рекурсивные 272

Ц

Цепная линия 395, 435
Цикл
 с параметром 222
 с условием 228
 бесконечный 228
Циклоида 124–126
Цилиндр 178
Циссоида Диоклеса 397, 439

Ш

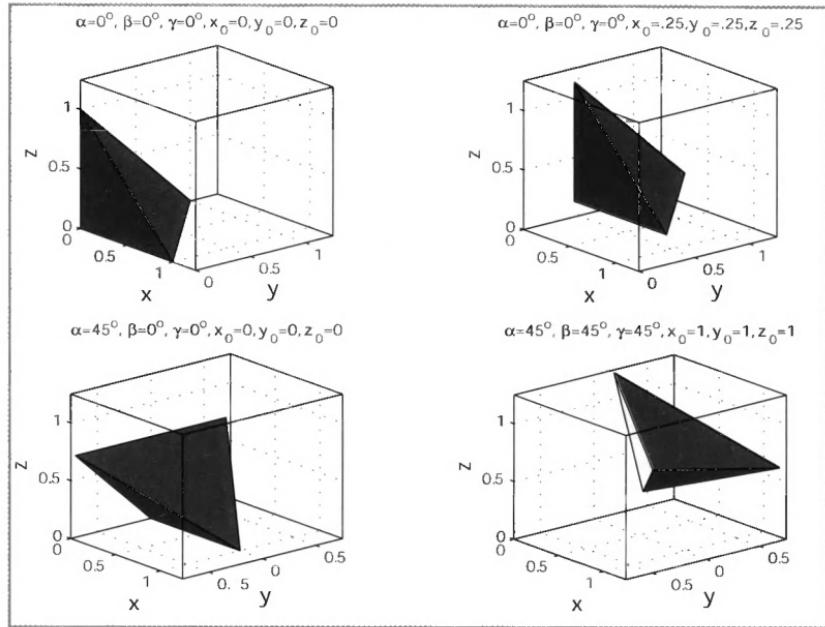
Шаг приращения 222
Шкала цветовой палитры 157
Шрифты 119

Э

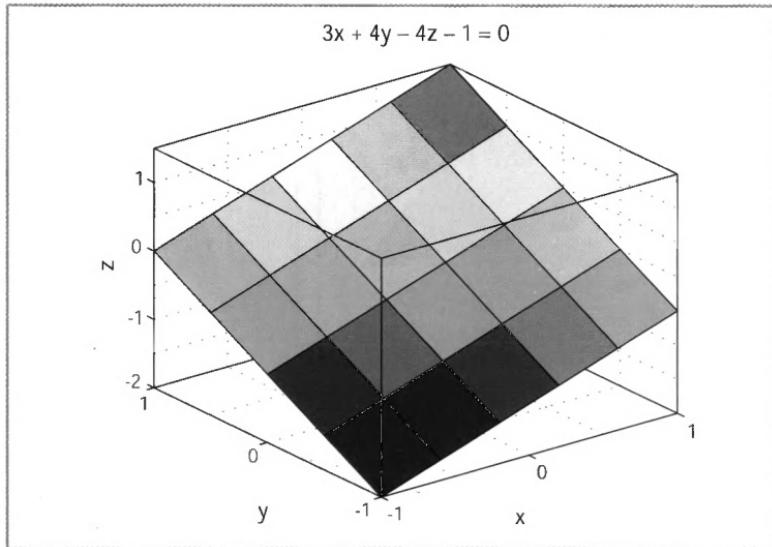
Эллипс
 большая ось 129
 уравнение
 каноническое 128
 параметрические 129
 малая ось 129
 эксцентриситет 129
Эллипсоид 184–186, 395, 398, 435, 440
Эллиптический параболоид
 192–194, 395, 397, 436, 439

Я

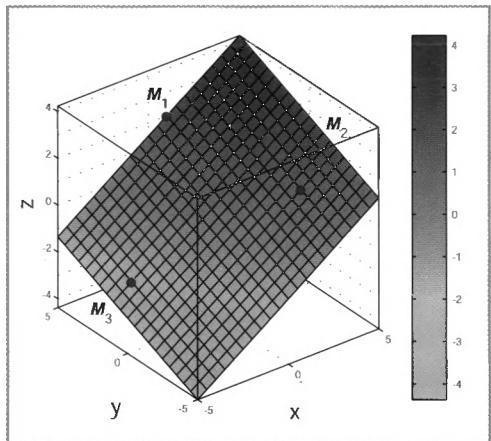
Ядро 18
Язык программирования 201
Ячейка
 доступ 50
 создание 49
 удаление 51, 52



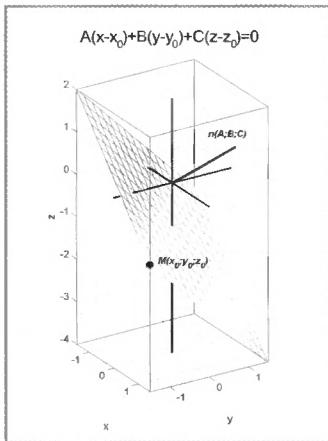
Ил. 1. Преобразование фигуры в пространстве



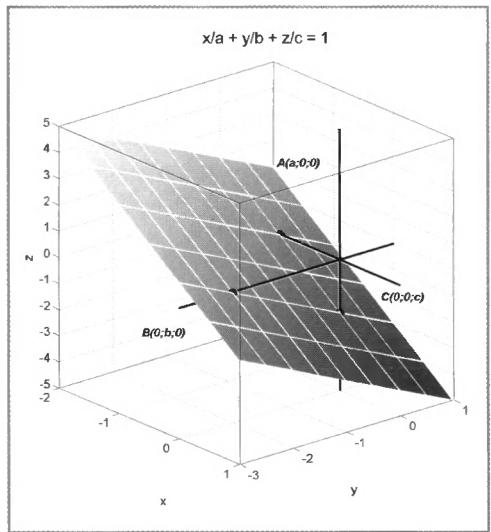
Ил. 2. График плоскости, заданной общим уравнением



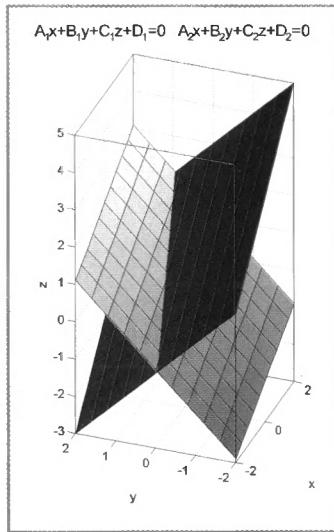
Ил. 3. Плоскость, заданная тремя точками



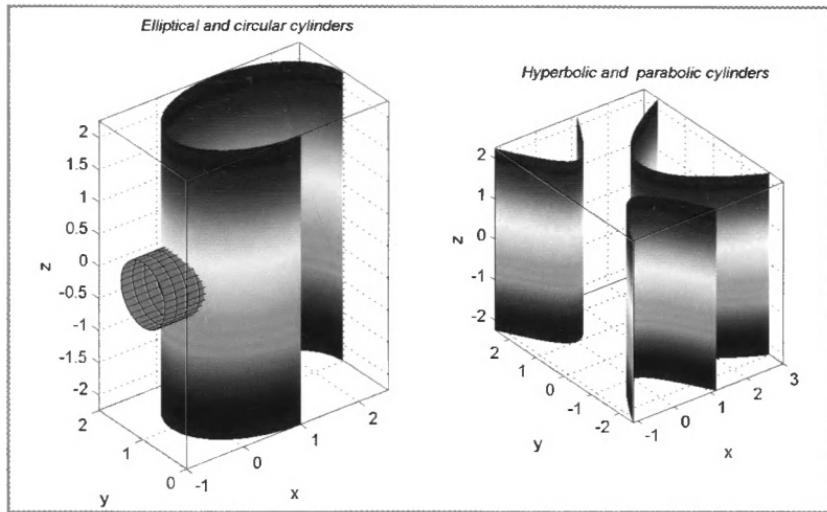
Ил. 5. Плоскость, заданная точкой и нормалью



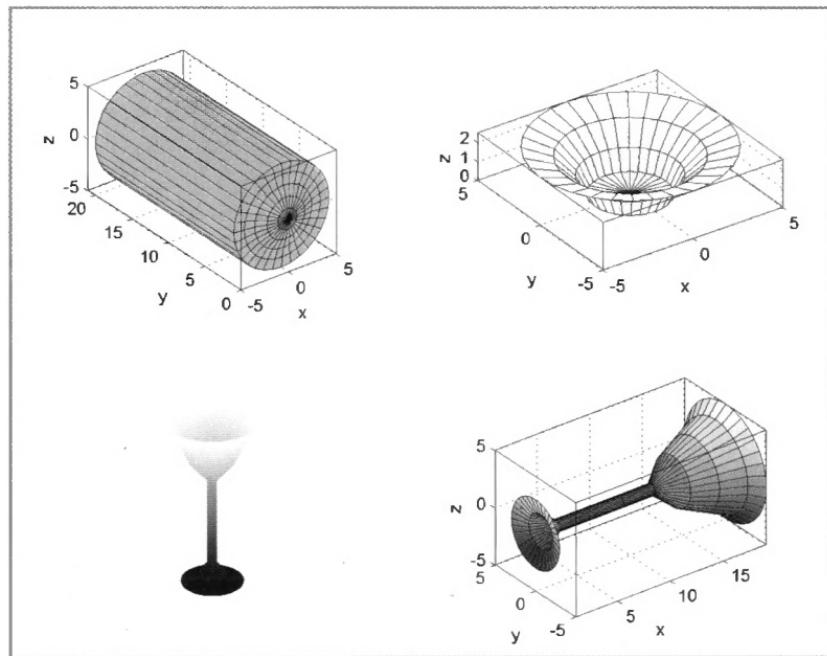
Ил. 4. Плоскость, заданная в отрезках



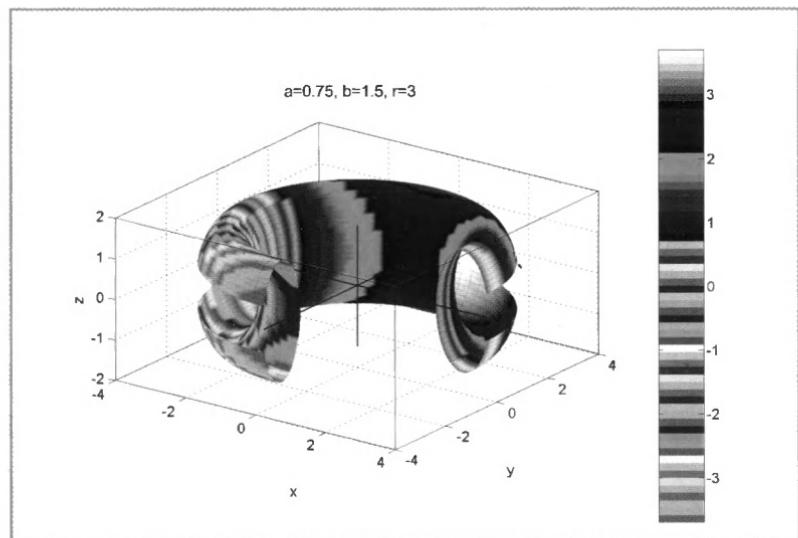
Ил. 6. Визуализация линий, заданной общими уравнениями



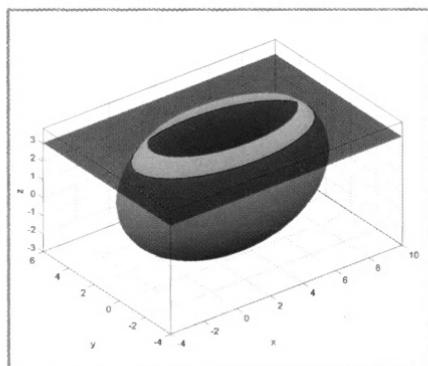
Ил. 7. Цилиндры второго порядка



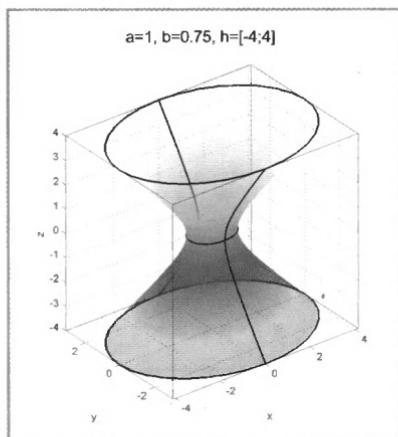
Ил. 8. Поверхности вращения



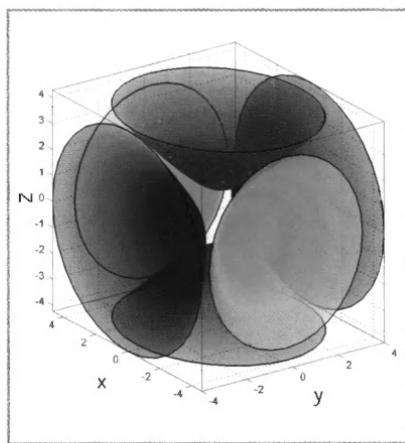
Ил. 9. Эллиптический тор



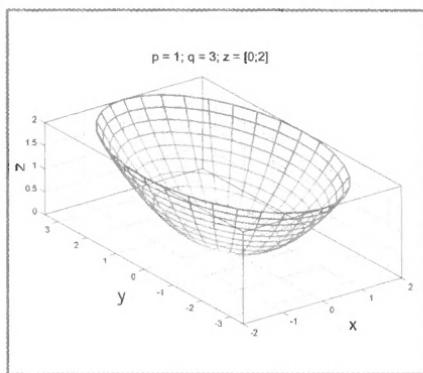
Ил. 10. Эллипсоид



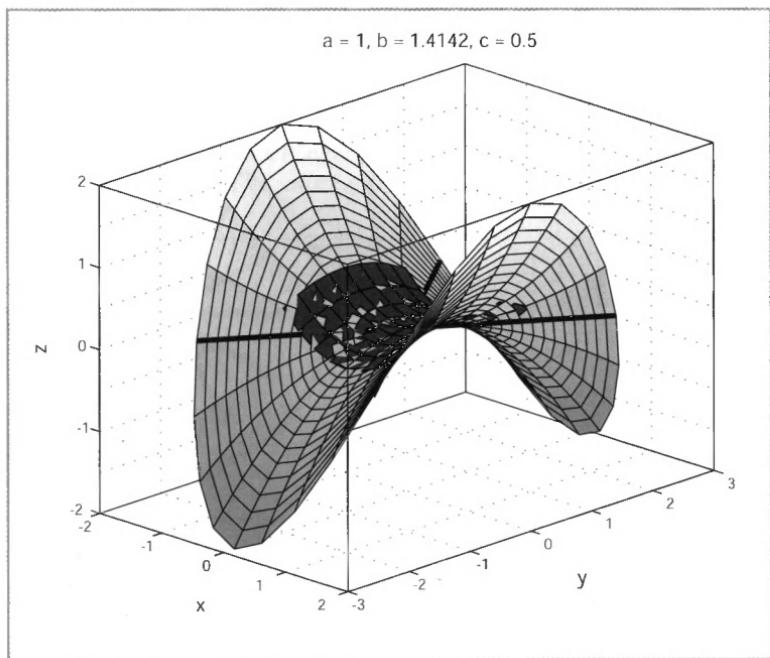
Ил. 11. Однополосный гиперболоид



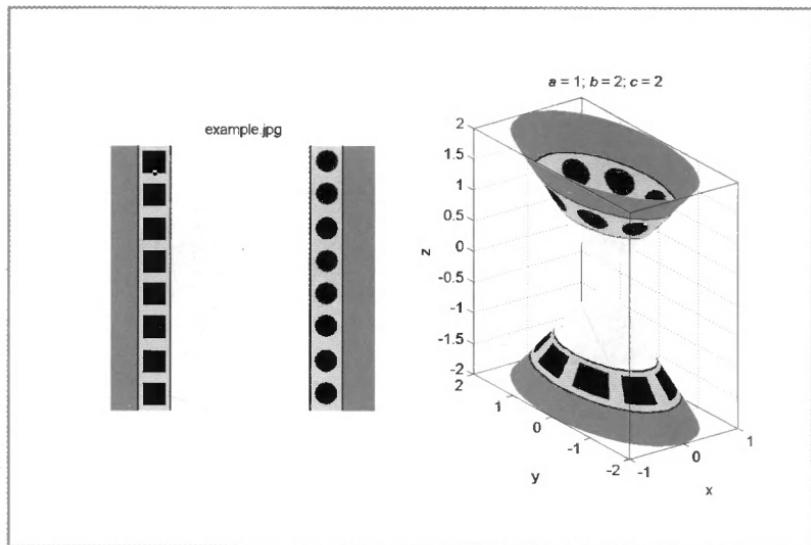
Ил. 12. Двуполосные гиперболоиды



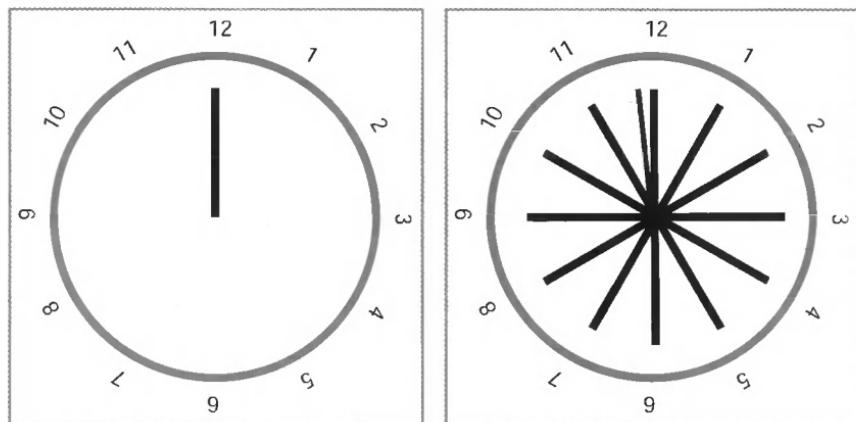
Ил. 13. Эллиптический параболоид



Ил. 14. Гиперболический параболоид



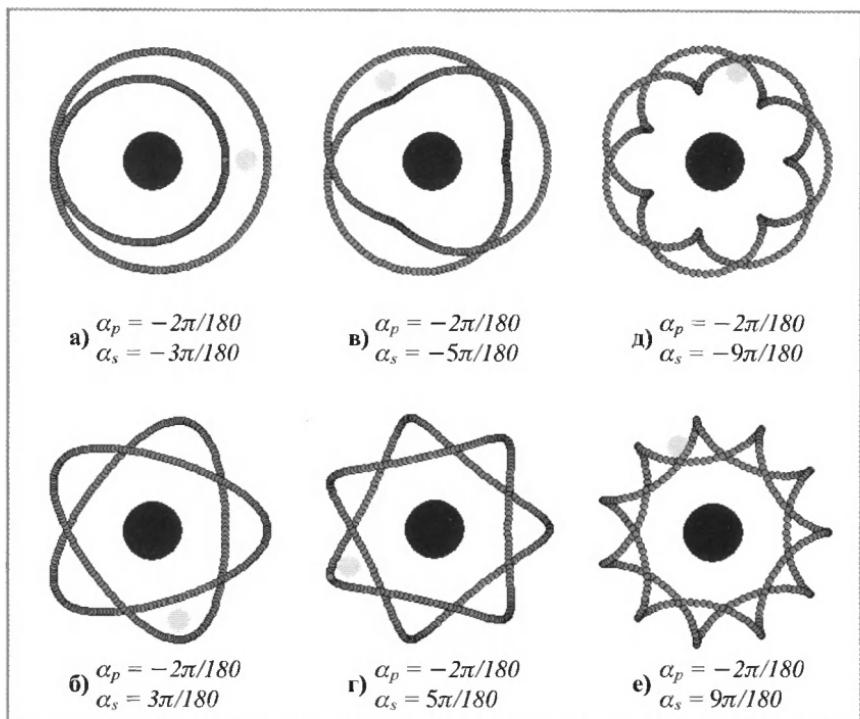
Ил. 15. Конус второго порядка



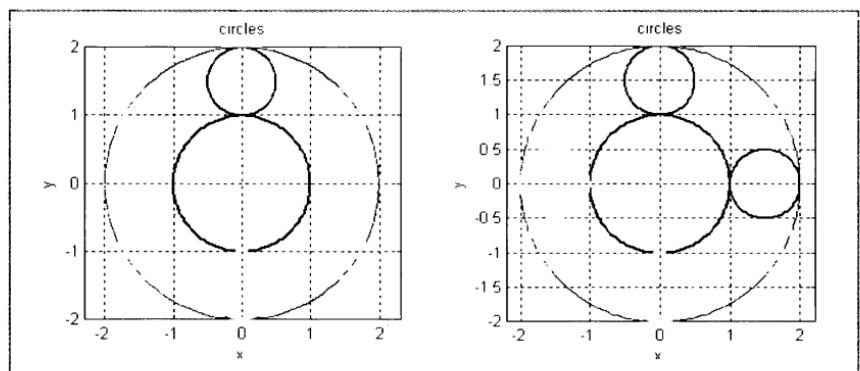
а) начальный момент анимации

б) заключительный момент анимации

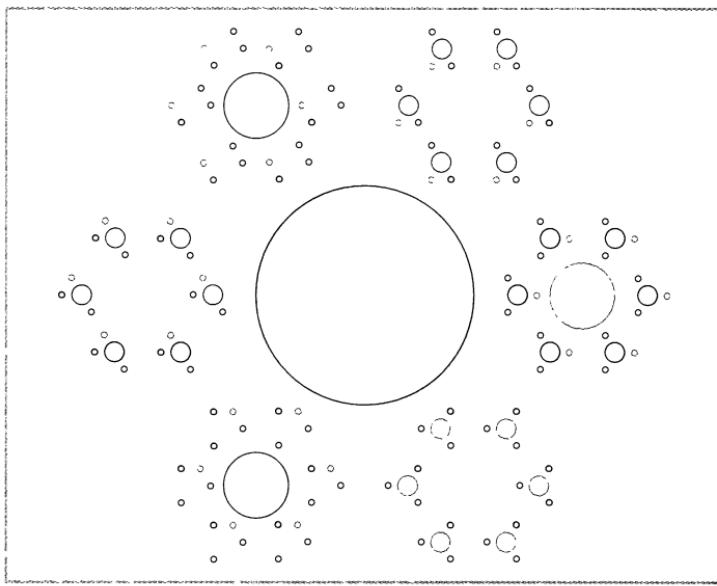
Ил. 16. Стрелочные часы



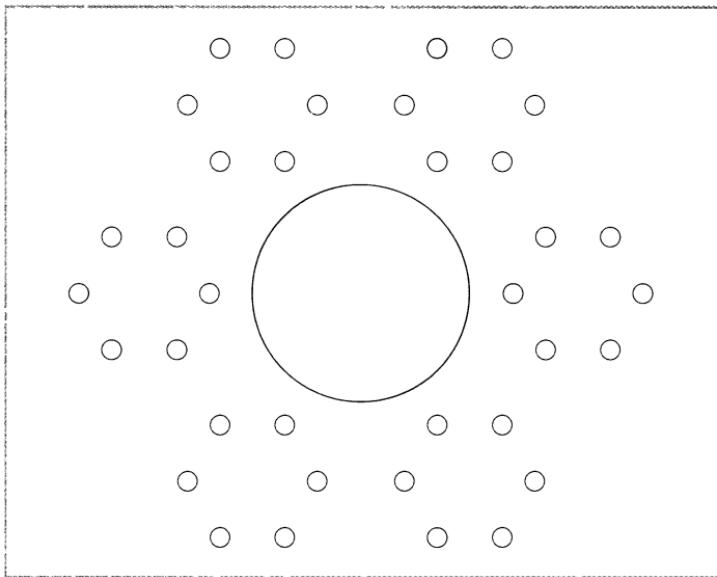
Ил. 17. Траектории обращения спутников вокруг планет, вращающихся вокруг солнца с разными углами поворота спутников



Ил. 18. Результаты выполнения функции *circles()* с переменным числом входных параметров



**Ил. 19. Использование рекурсивной схемы
при построении фрактального изображения**



**Ил. 20. Использование итеративной схемы
при построении фрактального изображения**

КРИВИЛЁВ Александр Владимирович

**ОСНОВЫ КОМПЬЮТЕРНОЙ МАТЕМАТИКИ
С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ MATLAB**

Подписано в печать 20.12.2004. Формат 60x90 $\frac{1}{16}$. Бумага офсетная.
Гарнитура Таймс. Печать офсетная. Усл. печ. л. 31.
Тираж 5000 экз. Заказ № 0416920.

Издательство «Лекс-Книга»:
119454, г. Москва, ул. Лобачевского, д. 92;
корп. 4, офис 6;
тел./факс: 789-34-06
E-mail: mail@lexkniga.ru
www.lexkniga.ru

По вопросам закупки обращаться в «Издательскую книготорговую группу «Лекс-Книга».
Оптовые поставки. Телефон 8(095)7893403. E-mail: mail@lexkniga.ru
Розничные продажи. Адрес магазина: Москва, ул. Лобачевского д. 92, кор. 4 офис № 6.
Телефон 8(095)7893406. Интернет-магазин: www.lexkniga.ru.
Возможность курьерской доставки по Москве,
отправление почтой заказанных и оплаченных книг в любую точку мира.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в ОАО «Ярославский полиграфкомбинат»
150049, г. Ярославль, ул. Свободы, 97.





за покупками в softline®

лицензионное программное обеспечение...

Экономия денежных средств. Выбор оптимальной схемы лицензирования ПО позволяет экономить до 50% ресурсов, выделяемых на ПО.

Надежность и признание вендоров. SoftLine[®] работает на рынке ПО более 10 лет и является надежным поставщиком, которого выбирают крупнейшие компании России. Обновление технологий и ПО — бесконечный процесс. Это и определяет долгосрочное взаимодействие с нашими клиентами. SoftLine[®] является авторизованным партнером более 150 компаний-производителей ПО, включая высшую ступень партнерства с такими компаниями, как Microsoft, VERITAS, Symantec, Citrix, WRQ, Macromedia и др.

Лучшая информационная поддержка при принятии решения. Информационная поддержка каждого клиента — наша гордость. Только работая с SoftLine[®], Вы можете получать каталог программного обеспечения, имеете возможность посещать семинары компании, а также «попробовать» выбранное Вами решение в Центре Решений. И все это абсолютно бесплатно!

SoftLine[®] — широкий спектр услуг и стратегическое партнерство с клиентами. SoftLine[®] предлагает широкий спектр услуг по поставке лицензионного ПО, технической поддержке, консалтингу и сертифицированному обучению в Учебном Центре SoftLine[®]. Работая с одним поставщиком услуг, Вы экономите средства и повышаете качество услуг.

Позвоните менеджерам отдела продаж по тел.: +7(095)232-00-23. Возможен бесплатный выезд менеджера компании для разработки программ корпоративного лицензирования!

softline решения для бизнеса

программное обеспечение — лицензирование, обучение, консалтинг

+ 7 (095) 232-00-23

www.softline.ru

©2004 SoftLine Int. Все права защищены. SoftLine, логотип SoftLine являются торговыми марками SoftLine и зарегистрированы в России и других странах. Другие компании и названия продуктов являются торговыми марками, представленными их владельцами.



каждый вторник

**зачем каждый вторник более
130 специалистов ведущих компаний
собираются в softline®?**

SoftLine продолжает цикл специализированных БЕСПЛАТНЫХ семинаров по программному обеспечению ведущих производителей ПО, интересы которых компания предоставляет в России. Это лидеры мирового рынка: Microsoft, Symantec, VERITAS, Citrix, CheckPoint, WRQ, ABBYY, «Лаборатория Касперского» и многие другие.

Квалифицированные докладчики. На семинарах SoftLine® выступают ведущие российские и западные специалисты компаний-разработчиков программного обеспечения, а также представители компаний, которые имеют практический опыт внедрения и использования программных продуктов. В качестве специальных гостей на семинарах выступают признанные эксперты в предметной области, определенной темой семинара.

Наши темы: сетевые операционные системы: установка, администрирование и upgrade; безопасность сетей: обеспечение сохранности данных, резервное копирование, антивирусная защита, подключение к Интернету и другие решения; средства системного администрирования: удаление администрирование, установка программ и др.; средства разработки: обзор визуальных сред разработки ПО, средства отладки и развертывания ПО, компоненты разработчика; лицензирование ПО.

Если вы определились с удобной датой семинаров, вы можете сообщить об этом нам в свободной форме на адрес seminars@softline.ru, зарегистрироваться на сайте www.softline.ru/seminars или позвонить по телефону +7 (095) 232-00-23.

softline

программное обеспечение – лицензирование, обучение, консалтинг

+ 7 (095) 232-00-23

www.softline.ru

©2004 Softline Int. Все права защищены. SoftLine, логотип SoftLine являются торговыми марками SoftLine и зарегистрированы в России и других странах. Другие компании и названия продуктов являются торговыми марками, представленными их владельцами.

