МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. НИЗАМИ

Набиулина Л.М., Абдурахманова Ш.А.

ПРИКЛАДНЫЕ МАТЕМАТИЧЕСКИЕ ПРОГРАММНЫЕ ПАКЕТЫ

Учебное пособие

Ташкент-2016

Данное учебное пособие предназначено для обучения студентов высших педагогических образовательных учреждений по направлениям бакалавриата 5111000-Профессиональное образование (5330200 – Информатика и информационные технологии и 5111000-Профессиональное образование (5330400 – Компьютерная графика и дизайн) учебной дисциплине «Прикладные математические программные пакеты». В нем представлены основы работы с пакетами MATHCAD и MATLAB. Возможности MATHCAD и MATLAB иллюстрируются на типовых вычислительных задачах. Приводятся примеры решения задач с подробными комментариями. Реализация заданий на компьютере позволит студентам приобрести навыки работы с данными математическими пакетами и эффективно использовать их в учебном процессе.

Рекомендовано студентам педагогических университетов и институтов, а также учителям академических лицеев и профессиональных колледжей

Рецензеты: Ташпулатова Н.Б. Кандидат технических наук, доцент кафедры «Основы информатики» при ТУИТ

Алибеков С.А. Кандидат физико-математических наук, старший преподаватель кафедры «Информационные технологии» ТГПУ им. Низами

Оглавление

Введение

Еще за долго до объявления независимости с первых дней своего руководства республикой Президент Ислом Абдуганиевич Каримов проявлял особую заботу о научной молодежи. Он отмечал: «Каждый из нас должен отдавать себе отчет в том, что Узбекистан сегодня - это составная часть мирового пространства…»[1, с.10]. Мировые тенденции развития образования направлены на всестороннее развитие человека путем вовлечения его в целесообразную самостоятельную учебно-познавательную деятельность.

Задачей информационно-образовательной технологии как науки является выявления совокупности закономерностей с целью определения использования на практике наиболее эффективных, последовательных образовательных действий, требующих меньших затрат времени, материальных и интеллектуальных ресурсов для достижения какого-либо результата.

Математические и научно-технические расчеты являются важной сферой применения персональных компьютеров. Часто они выполняются с помощью программ, написанных на языке высокого уровня, например Delphi или C++. Сегодня эту работу нередко выполняет обычный пользователь ПК. Для этого он вынужден изучать языки программирования и многочисленные, подчас весьма тонкие капризные численные методы математических расчетов. Нередко при этом из под руки способного физика, химика или инженера выходят далёкие от совершенства программы.

Это не вполне нормальное положение может изменить к лучшему применение интегрированных программных систем автоматизации математических расчетов (MathCAD, MatLab и др.).

Программы MATHCAD и MATLAB позволяют моделировать в электронном документе научно-технические расчеты в форме, близкой к общепринятому в математике виду.

Применение MATHCAD и MATLAB в учебном процессе способствуют повышению эффективности работы студентов при изучении таких учебных дисциплин, как «Математическое и компьютерное моделирование», а также при выполнении курсовых и дипломных работ.

Чрезвычайная простота интерфейса MATHCAD и MATLAB сделала их одним из самых популярных математических пакетов.

В учебном пособии излагаются средства системы MATHCAD и MATLAB, приводится ряд примеров по следующим темам:

- вычисление сложных математических выражений, записанных в естественной форме;
- табулирование функций одной и двух переменных;
- построение двух- и трехмерных графиков;
- операции с векторами и матрицами;
- символьные преобразования.

В учебном пособии приведены также элементы численных методов и примеры их реализации в MATHCAD и MATLAB, а именно:

- решение уравнений и систем уравнений;
- вычисление интегралов;
- решение дифференциальных уравнений.

Цель учебного пособия – научить студентов быстро и легко решать в среде MATHCAD и MATLAB простые математические задачи вычислительного характера. Кроме того, к каждой теме прилагается рабочий документ MATHCAD и MATLAB, в котором приведены примеры. Предполагается, что студенты выполняют их в первую очередь, и только освоив эти примеры и получив те же результаты, приступают к выполнению своих индивидуальных заданий.

Структура программы MATLAB

1. Структура программы пакета MATLAB
2. Простые переменные и основные типы данных в MATLAB
3. Арифметические операции с простыми переменными

MATLAB — язык программирования и система научных и инженерных расчетов, построенная на основе интерпретатора этого языка. MATLAB, сокращение от «Matrix Laboratory», предназначен в первую очередь для выполнения алгоритмов, использующих векторы и матрицы. Оболочка MATLAB состоит из командной строки, текстового редактора со встроенным отладчиком и окнами со списком файлов, списком видимых переменных и с историей введенных команд.

MATLAB имеет большое число пакетов (toolboxes) — как собственных, так и распространяемых независимыми разработчиками часто на условиях открытого кода. В MATLAB включен Simulink — визуальный редактор для моделирования динамических систем.

При запуске системы MATLAB появляется основное окно, изображенная на рисунке 1.

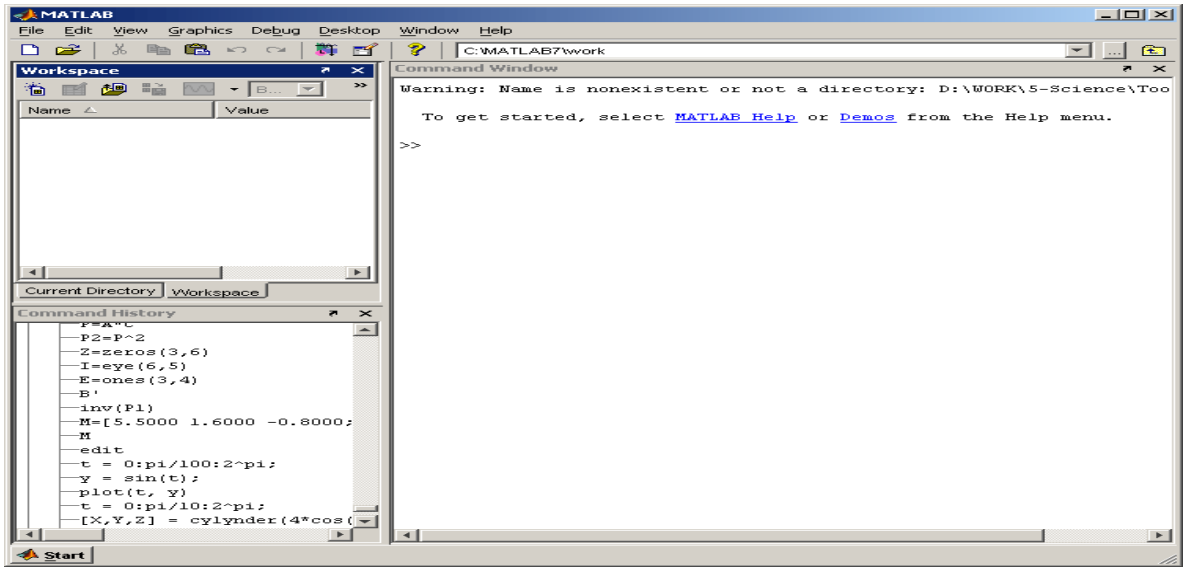


Рис. 1. Основное окно системы MATLAB

Рабочая среда MATLAB содержит следующие элементы:

- панель инструментов с кнопками и раскрывающимся списком;
- окно с вкладками Launch Pad и Workspace, из которого можно получить доступ к различным модулям ToolBox и к содержимому рабочей среды;
- окно с вкладками **Command History** и **Current Directory**, предназначенное для просмотра и повторного вызова ранее введенных команд, а также для установки текущего каталога;
- командное окно, в котором находится приглашение к вводу » и мигающий вертикальный курсор;
- строку состояния.

Если в рабочей среде MATLAB отсутствуют некоторые окна, приведенные на рисунке, то следует в меню View выбрать соответствующие пункты: Command Window, Command History, Current Directory, Workspase, Launch Pad.

Команды следует набирать в командном окне. Символ », обозначающий приглашение к вводу командной строки, набирать не нужно. Для просмотра рабочей области удобно использовать полосы скроллинга или клавиши Home, End, для перемещения влево или вправо, и PageUp, PageDown для перемещения вверх или вниз. Если вдруг после перемещения по рабочей области командного окна пропала командная строка с мигающим курсором, просто нажмите Enter.

Важно помнить, что набор любой команды или выражения должен заканчиваться нажатием на Enter, для того, чтобы программа MATLAB выполнила эту команду или вычислила выражение.

Наберите в командной строке 1+2 и нажмите Enter. В результате в командном окне MATLAB отображается следующее:

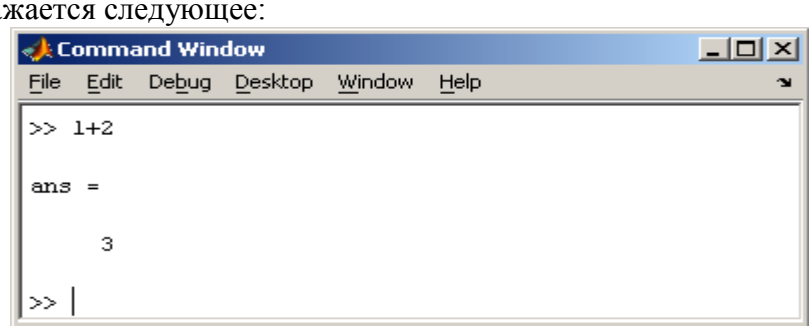


Рис. 2. Запись примера на листе вычислений системе MATLAB

Сначала она вычислила сумму 1+2, затем записала результат в специальную переменную ans и вывела ее значение, равное 3, в командное окно. Ниже ответа расположена командная строка с мигающим курсором, обозначающая, что MATLAB готов к дальнейшим вычислениям. Можно набирать в командной строке новые выражения и находить их значения. Если требуется продолжить работу с предыдущим выражением, например, вычислить (1+2)/4.5, то проще всего воспользоваться уже имеющимся результатом, который хранится в переменной ans. Наберите ans/4.5 (при вводе десятичных дробей используется точка) и нажмите Enter, получается

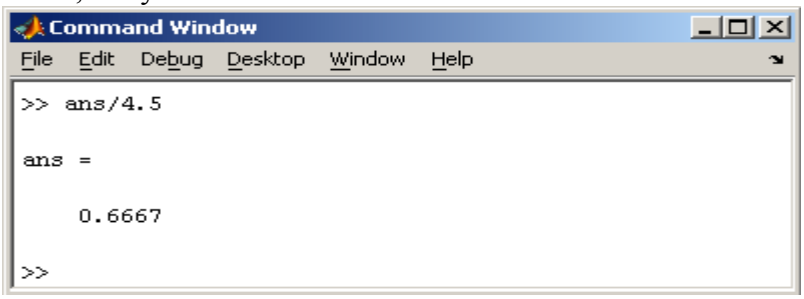


Рис. 3. Запись примера на листе вычислений в системе MATLAB

Как правило, каждая программа в MATLAB представляет собой функцию и начинается с ключевого слова function, за которым через пробел следует ее название. Например,

```
function Lab1
a = 5;
b = 2;
c = a*b;
```

Данная программа заключена в функции с именем Lab1 и вычисляет произведение двух переменных a и b. При сохранении программы в m-файл рекомендуется указывать имя файла, совпадающее с именем функции, т.е. в данном случае – Lab1.

Следует отметить, что в одном m-файле можно задавать множество дополнительных функций. Для этого достаточно написать в конце листинга основной программы еще одно ключевое слово function и задать ее имя, например,

```
functionLab1
a=5;
b=2;
c=a*b;
out_c(c);              - вызов функции out_c()

function out_c(arg_c)   - определение функции out_c()
disp(arg_c);
```

Обратите внимание, что функцию out_c() можно вызывать в основной программе до ее определения. Это особенность языка MATLAB, позволяющая не беспокоиться программисту о последовательности задания функций. В приведенном примере функция out_c() имеет один входной параметр с именем arg_c, который выводится на экран (в командное окно MATLAB) с помощью встроенной функции disp(). В итоге, при выполнении приведенной программы в командном окне MATLAB будет отображено значение переменной c.

Дополнительные функции можно оформлять и в отдельных m-файлах. Например, если есть необходимость какую-либо функцию описать в одном m-файле, а вызывать ее в другом, то это можно реализовать следующим образом.

1-й файл (Lab1.m)

```
function Lab1
a = 5;
b = 2;
c = square(a,b);      - вызов функции square()
out_c(c);             - вызов функции out_c()

function out_c(arg_c)   - определение функции out_c()
disp(arg_c);
```

2-й файл (square.m)

```
function res=square(a, b)
res = a*b;
```

При выполнении функции Lab1 система MATLAB вызовет функцию square из файла square.m. Это будет сделано автоматически, т.к. встроенные функции языка MATLAB определены также и вызываются из файлов, имена которых, как правило, соответствуют именам вызываемых функций. Обратите также внимание на то, что функция square() не только принимает два аргумента a и b, но и возвращает их произведение с помощью переменной res. Представленный синтаксис следует использовать всякий раз, когда требуется возвратить результат вычислений основной программе. В четвертой главе данного пособия более подробно изложены конструкции вызова функций для реализации разнообразных алгоритмов.

2. Создание программы, как правило, начинается с определения переменных и способа представления данных. Следовательно, чтобы правильно организовать описание

данных программы, необходимо знать как задавать переменные в MATLAB и какие виды переменных возможны.

Самый простой и наиболее распространенный тип данных – это число. В MATLAB число хранится в переменной, которая имеет некоторое уникальное имя, например,

```
a = 5;
```

задает переменную с именем a и присваивает ей значение 5. По умолчанию переменная a является вещественной (тип double), т.е. может принимать дробные значения, например,

```
a = -7.8;
```

задает значение переменной a равное -7,8. Изменить тип переменной можно, указав тип присваиваемого числа с помощью соответствующего ключевого слова, например,

```
a = int16(5);
```

выполнит присваивание числа 5 как целочисленного 16-битового значения. В результате выполнения такой операции тип переменной a будет соответствовать int16.

Типы данных, доступные в MATLAB, представлены в табл. 1.

Таблица 1. Основные типы данных в MATLAB

double	вещественный, 64 бит
single	вещественный, 32 бит
int8	знаковый целочисленный, 8 бит
int16	знаковый целочисленный, 16 бит
int32	знаковый целочисленный, 32 бит
int64	знаковый целочисленный, 64 бит
uint8	беззнаковый целочисленный, 8 бит
uint16	беззнаковый целочисленный, 16 бит
uint32	беззнаковый целочисленный, 32 бит
uint64	беззнаковый целочисленный, 64 бит

По умолчанию используется тип double, который имеет наибольшую точность представления вещественного числа и является потому универсальным типом. Однако, если необходимо экономить память ЭВМ, то можно указывать самостоятельно желаемый тип.

Последнее, что следует знать при задании переменных – это правило определения их имен. В MATLAB имена переменных могут задаваться только латинскими буквами, цифрами и символом ‘_’. Причем, первый символ в имени должен соответствовать букве латинского алфавита. Также следует отметить, что имена

```
arg = 1;
Arg = 2;
ARG = 3;
```

это три разных имени, т.е. три разные переменные со значениями 1, 2 и 3 соответственно. Данный пример показывает, что MATLAB различает регистр в именах переменных.

При программировании лучше всего задавать осмысленные имена переменных, по которым можно было бы понять какие данные они представляют. Это позволяет избежать путаницы при построении больших программ.

3.Рассмотрим базовые арифметические операции над простыми переменными в MATLAB. Пусть заданы две переменные a и b. Тогда операции сложения, вычитания, умножения и деления запишутся так:

```
c = a+b;  - сложение
c = a-b;  - вычитание
c = a*b; - умножение
c = a/b;   - деление
```

Дополнительно, в MATLAB определена операция возведения в степень, которая записывается так:

c = a^2; - возведение переменной a в квадрат

c = a^0.5; - извлечение квадратного корня из переменной a

Приоритет арифметических операций * и / выше операций + и - , т.е. сначала выполняется умножение и деление, а затем, сложение и вычитание. Операция возведения в степень имеет наивысший приоритет. Для изменения приоритетов арифметических операций используются круглые скобки, как и в обычной математике, например,

c = 7+2*2; - значение c = 28

c = (7+2)*2; - значение c = 18

В практике программирования есть несколько устоявшихся подходов использования арифметических операций. Например, если необходимо увеличить значение переменной arg на 1, то этого можно добиться следующим образом:

arg = arg + 1;

При этом совершенно не важно чему равна переменная arg, в любом случае ее значение будет увеличено на 1. Аналогичным образом можно выполнять увеличение или уменьшение значения переменной на любую величину.

Другим устоявшимся приемом программирования является обмен значений между двумя переменными. Например, заданы переменные arg_a и arg_b и необходимо произвести обмен данными между ними. Это достигается следующим образом:

```
temp = arg_a;
arg_a = arg_b;
arg_b = temp;
```

Здесь temp – это временная переменная, необходимая для сохранения значения переменной arg_a, т.к. оно затирается во второй строчке значением arg_b. Поэтому в третьей строке переменной arg_b присваивается сохраненное в temp значение переменной arg_a.

Если в результате выполнения вычислений появляется комплексное число, то MATLAB автоматически будет оперировать с такими числами в соответствии с арифметикой комплексных чисел. Например, при извлечении квадратного корня из -1, получим следующий результат:

c = (-1)^0.5; - c = 0.0000 + 1.0000i

Здесь i – зарезервированное имя мнимой единицы и представленная запись обозначает комплексное число с нулевой действительной частью и единичной мнимой. В MATLAB в качестве зарезервированного имени мнимой единицы также используется буква j.

Для того, чтобы задать комплексную переменную достаточно указать значения их действительной и мнимой частей как показано ниже

c = 6 + 5i; - комплексное число

и с заданными комплексными переменными также можно выполнять описанные выше арифметические операции.

При работе с комплексными числами существуют две специальные функции:

real(x) – взятие действительной части комплексного числа x;

imag(x) – взятие мнимой части комплексного числа x;

abs(x) – вычисление абсолютного значения комплексного числа x;

conj(x) – вычисление комплексно-сопряженного числа x;

angle(x) – вычисление аргумента комплексного числа x;

Контрольные вопросы:

1. Интерфейс программы MATLAB?
2. Назовите основные типы данных в MATLAB?

3. Какие арифметические операции можно произвести над простыми переменными в MATLAB?
4. Структура программы MATLAB?

Структура программы MATHCAD

1. Возможности программы MATHCAD
2. Простые переменные и основные типы данных в MATHCAD
3. Арифметические операции с простыми переменными

1.MATHCAD - система визуальных математических расчетов. Основная идея MATHCAD состоит в том, что вычисляемые выражения записываются в визуальной форме, максимально приближенной к математической записи, привычной для человека. Используется принцип *WYSIWYG (What You See Is What You Get* - «что видите, то и получаете»).

Возможности системы:

1. *Числовые расчеты* со скалярами, матрицами и векторами (матрицами из одного столбца). Возможны расчеты с использованием комплексных чисел.
2. *Аналитические преобразования*: интегрирование, дифференцирование, вычисление пределов, сумм и произведений рядов, упрощение, преобразования Лапласа и Фурье и др.
3. *Определение законов вычисления элементов матриц*, что позволяет реализовать итерационные вычисления, в том числе по рекуррентным формулам.
4. Работа со *стандартными функциями*: интерполяция, экстраполяция, численное интегрирование, матричные функции и др.
5. *Определение* своих *функций*.
6. *Построение* двумерных и трехмерных *графиков* различных видов.
7. *Решение систем* линейных и нелинейных уравнений.
8. *Решение оптимизационных задач* вида: найти значения переменных, при которых функция принимает минимальное или максимальное значение.
9. *Решение дифференциальных уравнений* (обыкновенные дифференциальные уравнения и системы уравнений; уравнения Пуассона и Лапласа).
10. *Элементы программирования*.

При запуске системы MATHCAD появляется основное окно, которое обычно содержит три панели (стандартную, форматирования текста, математическую) и рабочую область с автоматически созданным листом вычислений.

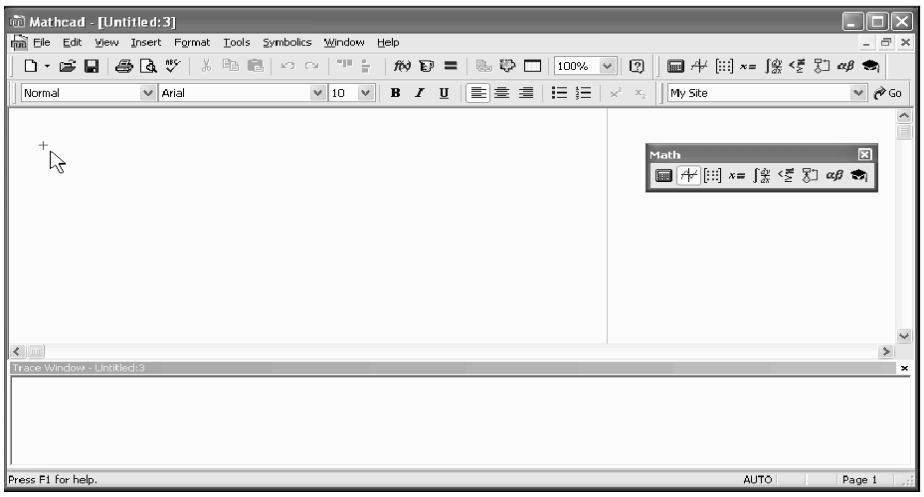


Рис. 4. Оновное окно системы MATHCAD

На математической панели находятся кнопки, при нажатии на которые открываются дополнительные панели с шаблонами ввода различных выражений: *Calculator* (знаки некоторых основных функций и операций), *Calculus* (шаблоны операций интегрирования, дифференциорования, пределов и других), *Evaluation* (операторы присваивания и вычисления), *Graph* (графики), *Greek* (символы греческого алфавита), *Matrix* (операции векторного и скалярного произведений, транспонирования, векторной суммы, вычисления определителя матрицы), *Programming* (элементы программирования), *Boolean* (логические операции), *Symbolic* (различного рода аналитические преобразования). Включать и отключать панели также можно из верхнего меню в разделе *View\Toolbars* или *Просмотр\Панели* (здесь и далее при указании разделов меню будут даваться варианты наименований для русифицированной и англоязычной версий системы).

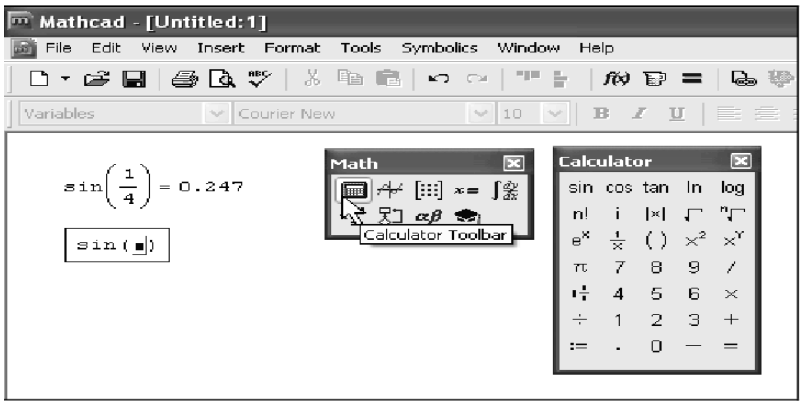


Рис. 5. Запись примера на листе вычислений в системе MATHCAD

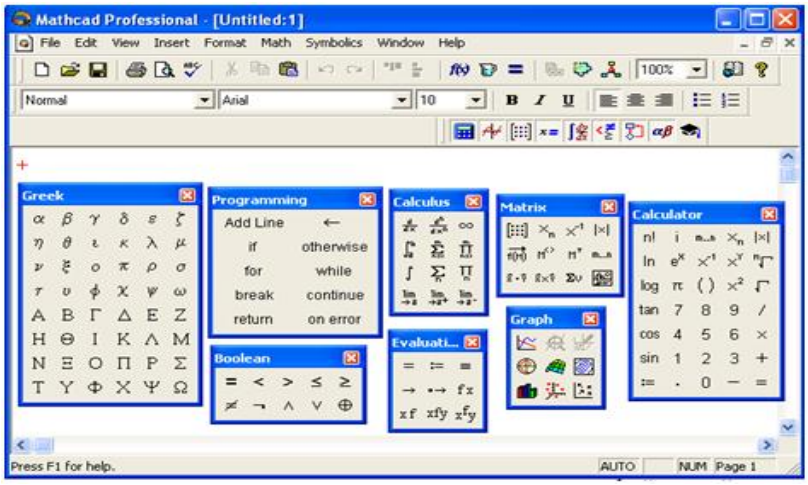


Рис. 6. Математическая панель в системе MATHCAD

На лист вычислений должны записываться все выражения и формулы, с которыми идет работа, также можно вставлять текст и графические изображения.

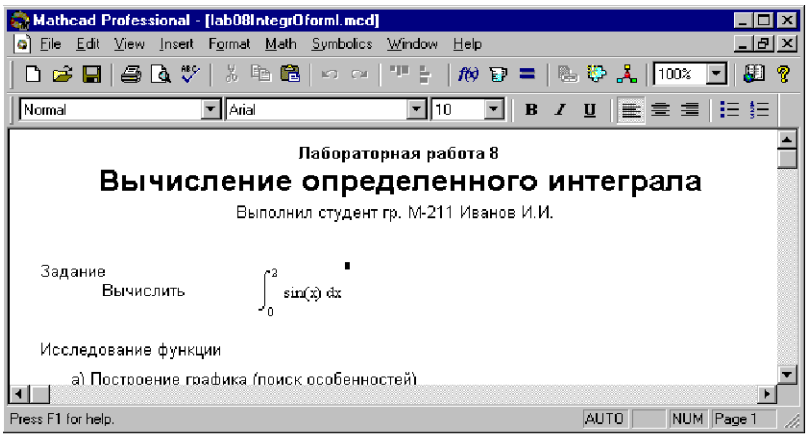


Рис. 7. Вычисление определенного интеграла в системе MATHCAD

MATHCAD может одновременно работать с несколькими листами. Создание нового листа осуществляется из верхнего меню - *File\New* или *Файл\Новый*. Загрузить уже имеющийся лист - *File\Open* или *Файл\Открыть*. Сохранить созданный лист - *File\Save* или *Файл\Сохранить*. Переключение между загруженными листами - комбинация клавиш *Ctrl-F6*.

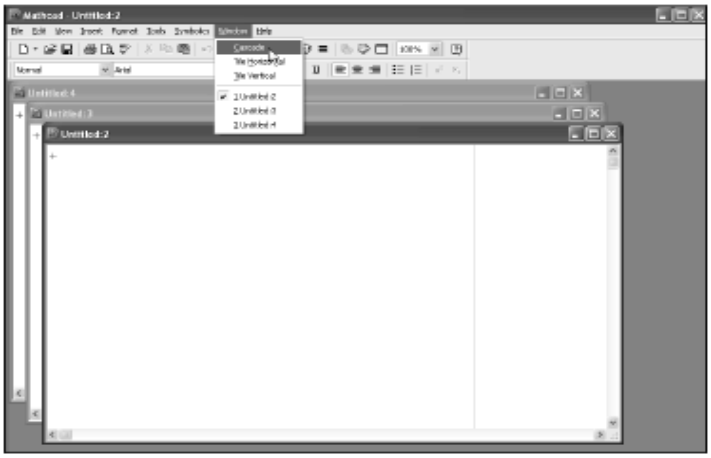


Рис. 8. Запись формул и выражений на лист вычислений

2. Любая формула может быть записана в любом месте листа. Необходимо щелкнуть мышью в предполагаемой точке ввода формулы, там должен появиться указатель ввода - крест. Ввод обычно производится как с клавиатуры, так и с помощью мыши. С клавиатуры всегда вводятся числа и имена переменных. Знаки арифметических операций и названия функций обычно также вводятся с клавиатуры.

Таблица 2. **Ввод некоторых типовых выражений и шаблонов MATHCAD с клавиатуры**

Требуемое действие	Вид выражения на листе MATHCAD	Последовательность нажатий клавиш
Умножение F на X	$F \cdot X$	F*X
Деление F на X	$\frac{F}{X}$	F/X
Возведение F в степень A	F^A	F^A
Квадратный корень из y	$\sqrt{y}$	\y
Ввод комплексного числа. Необходимо обратить	1+2j	1+2j

внимание на то, что при вводе комплексной части не следует записывать операцию умножения. В приведенном примере просто последовательно нажимаются клавиши «2» и «j»		
Модуль z. Здесь и далее в фигурных скобках будут перечисляться клавиши, которые необходимо нажимать одновременно. В данном случае после нажатия клавиши с буквой «z» необходимо одновременно нажать клавиши «Shift» и «\»	$\lvert z \rvert$	z {Shift \}
Сопряженное к z комплексное число	$\bar{z}$	z {Shift "}
Числовое вычисление выражения 1+2	1+2=	1+2=
Аналитическое вычисление выражения A. Результатом такого вычисления является не число, а формула (например, результат вычисления интеграла)	$A \rightarrow$	A {Ctrl .}
Присваивание переменной A значения 5	A:=5	A:5
Ввод логического равенства. Здесь прямых вычилений не происходит, лишь устанавливаем тот факт, что одно выражение равно другому выражению. Обычно такой оператор используется при записи уравнений, решаемых в блоке Given-Find. Иногда используется для ввода поясняющих (не вычисляемых) формул, например F = ma	A=x+y	A {Ctrl =} x+y
Определение последовательности F в диапазоне от 10 до 15 с шагом 1. F = 10,11,12,13,14,15	F:=10..15	F : 1 ; 10
Определение последовательности F в диапазоне от 2 до 10 с шагом 2. F = 2,4,6,8,10	F:=2,4..10	F : 2 , 4 ; 10
i-й элемент вектора F	F_i	F [i
Элемент матрицы F с индексами (i, j)	$F_{i,j}$	F [i , j
Ввод переменной с вспомогательным индексом. Здесь индекс - только часть имени	F_{max}	F.max
Транспонирование матрицы. Не следует путать его с возведением в степень	A^T	A {Ctrl 1}
Поэлементное умножение матриц. Здесь клавиша «пробел» используется для смены уровня ввода (будет разъяснено далее)	$\overline{A \cdot B}$	A*B пробел {Ctrl -}
Ввод матрицы или вектора. После нажатия *Ctrl-M* откроется окно, в котором необходимо указать количество строк (в пункте Rows) и столбцов (в пункте Columns)	$\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}$	{Ctrl M}
Корень n-й степени из y	$\sqrt[\bullet]{y}$	{Ctrl \} y

Первая производная	$\frac{d}{d\blacksquare}\blacksquare$	{Shift /}
Символ производной в записи начальных условий к дифференциальным уравнениям. Используется в блоке Given-Odesolve	x’(t)	x {Ctrl F7} (t)
Двусторонний предел	$\lim_{\blacksquare\to\blacksquare}\blacksquare$	{Ctrl L}
Неопределенный интеграл	$\int \blacksquare\, d\blacksquare$	{Ctrl I}
Греческие символы. Вводится буква латинского алфавита (обычно произносится так же, как первая буква названия греческого символа) и нажимается *Ctrl-G*	$\alpha\ \beta\ \delta\ \varepsilon\ \phi\ \gamma\ \varphi\ \lambda$ $\mu\ \pi\ \rho\ \tau\ \omega\ \psi$	a b d e f g j l m p r t w y {Ctrl G}

Здесь перечислены лишь основные шаблоны. *Совершенно излишне* запоминать все приведенные комбинации клавиш, поскольку эти и остальные шаблоны доступны из панелей, открывающихся при нажатии различных кнопок математической панели. Достаточно щелкнуть мышью по шаблону - и он появится на листе в месте, отмеченном указателем ввода.

В последних пяти примерах таблицы и во многих других стандартных выражениях после нажатия указанной последовательности клавиш необходимо ввести данные в места, отмеченные небольшими черными квадратами, называемыми *позициями ввода*. Чтобы ввести данные в такую позицию, необходимо сначала либо щелкнуть по ней мышью, либо передвинуть в данную позицию курсор, пользуясь клавишами управления курсором. В любом случае необходимо добиться того, чтобы *курсор подчеркивал только данную позицию* и затем вводить данные.

Во время ввода или редактирования формулы MATHCAD обводит ее рамкой (рис. 9).

$$\frac{a+b}{2}+x$$

Рис. 9. Формула на листе MATHCAD в рамке ввода

Наиболее частая ошибка заключается в том, что в процессе ввода формулы пользователь неосторожно выходит из указанной рамки ввода (при этом рамка исчезает и в рабочей области появляется указатель ввода - крест) и все равно продолжает ввод. При этом на листе вместо одной обычно оказываются две формулы, каждая из которых - лишь часть требуемой. Проблема в том, что визуально эти две формулы могут находиться близко друг к другу и выглядеть как единое целое, но это не так. Единственный способ проверить целостность формулы - щелкнуть по ней мышью. Если все правильно, то в рамке ввода окажется вся формула.

При наборе выражения можно последовательно отменять произведенные действия, нажимая *Ctrl-Z*. Однако следует учесть, что в младших версиях MATHCAD (до MATHCAD 11) отмена возможна только *до выхода из рамки ввода*. Закончить ввод формулы можно, либо нажав *Enter* или *Tab*, либо щелкнув мышью где-нибудь вне рамки ввода формулы.
Если при вводе выражения произошла какая-либо ошибка, то ошибочный фрагмент будет автоматически выделен красным цветом.

Для *удаления формулы* достаточно выделить ее (например, щелкнув мышью) и нажать *Ctrl-D*. Для *перемещения формулы* необходимо выделить ее и, ухватившись мышью за

появившуюся рамку ввода, переместить. Можно одновременно перемещать блок формул, который необходимо предварительно выделить методом протягивания.

3. Смена уровня ввода формул

При вводе формул часто возникает необходимость применить какую-либо операцию не к одному элементу выражения, а к некоторой его части из нескольких элементов. Например, необходимо ввести выражение вида $\frac{a+b}{2}$. После ввода последовательности «a+b» формула приобретет вид $a+b$. Курсор подчеркивает символ «b», поэтому если сразу ввести символ деления и затем цифру, то будет введено выражение $a+\frac{b}{2}$. Поэтому после ввода «a+b» следует перевести курсор на более низкий уровень, нажав клавишу «пробел». Тогда выражение принимает вид $a+b$, и если далее ввести символ деления и цифру, то будет получена требуемая формула.

Смена уровня ввода с помощью клавиши «пробел» наиболее часто требуется при наборе выражений со степенями, например $A^{-1}B$.

Примеры простых вычислений

$$\int_a^b \exp(x)\,dx \rightarrow \exp(b) - \exp(a)$$

$$\frac{5-\sqrt{3}}{2} = 1.634$$

$$\frac{1+2j}{1-2j} = -0.6 + 0.8i$$

Рис. 10. Фрагмент листа MATHCAD с вычисленными выражениями

На рис. 10 приведены примеры аналитического и числового вычислений. Левая часть, включая знаки вычисления, вводится пользователем, правая часть - результат, вычисленный системой MATHCAD.

Помимо интегралов система позволяет аналитически вычислять выражения, содержащие производные различных порядков, суммы и произведения рядов, односторонние и двусторонние пределы. В разделе *Symbolics* верхнего меню также доступны многие операции аналитического преобразования выражений (упрощение, свертка, разложение в ряд, преобразования Лапласа и Фурье и др.)

Вставка текста и рисунков

Чтобы *вставить текст*, необходимо щелкнуть мышью в той точке листа, где необходим текст, и выполнить команду верхнего меню *Insert\Text Region* или *Вставка\Текстовый регион*. Появится рамка, в которую можно ввести текст. При этом иногда возникает необходимость сменить шрифт для правильного отображения русских букв - это можно сделать с помощью списка шрифтов на панели форматирования.

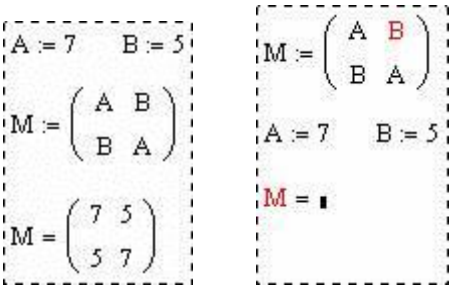
Вставку рисунка проще всего осуществить через буфер обмена. Предварительно необходимо поместить изображение в буфер (например, с помощью графического редактора Paint), затем вернуться в MATHCAD и выполнить пункт верхнего меню *Edit\Paste* или *Правка\Вставить*.

Вычисления с переменными

Переменные могут использоваться в выражениях MATHCAD на тех же правах, что и числовые константы. Имена переменных могут включать латинские и греческие буквы,

цифры, знаки подчеркивания и процента, а также вспомогательный индекс. Строчные и прописные буквы различаются, поэтому, например, x и X - две различные переменные.

Чтобы *определить переменную*, достаточно просто присвоить ей значение. Есть *строгое правило порядка записи переменных и выражений с переменными*: если в некотором выражении используется переменная, то эта переменная должна быть определена на листе MATHCAD либо выше выражения, либо в той же строчке, но левее (рис. 11). Несоблюдение данного правила - одна из наиболее частых ошибок.



а) *б)*

Рис. 11. Примеры правильной (а) и неправильной (б) последовательностей расположения формул с переменными

Значения переменных можно изменить в любой момент, отредактировав соответствующую формулу. При этом обычно автоматически пересчитываются все формулы, которые прямо или опосредованно зависят от данной переменной. Если же этого не происходит, то необходимо включить автовычисление с помощью верхнего меню MATHCAD (пункт *Math\Automatic calculation* или *Математика\Автовычисление*).

Пример матричных вычислений: решение системы линейных алгебраических уравнений методом обратной матрицы

Допустим, что необходимо решить следующую систему уравнений:

$$\begin{cases} 2x + y = 4; \\ -x + y = 1. \end{cases}$$

Воспользуемся *матричным методом*, когда решение находится по формуле $X = A^{-1}B$, где A - матрица коэффициентов при переменных системы; B - вектор свободных членов. Поскольку в MATHCAD нет понятия вектора, используется матрица из одного столбца. На листе MATHCAD (рис. 12) необходимо определить эти матрицы, записать формулу для X и подсчитать X, применив оператор числового вычисления «=».

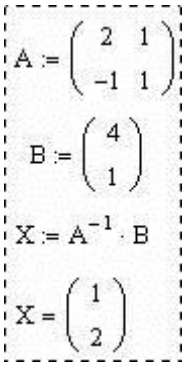


Рис. 12. Вид листа MATHCAD при решении данной задачи

Контрольные вопросы:

1.Общие принципы MathCAD и описание интерфейса.

2. Как записываются формулы и выражения в листе вычислений?

3. Как вводятся формулы?

4. Типы данных MathCAD?
5. Переменные в MathCAD?

Основные математические операции в MATLAB и типы данных

1. Основные математические функции MATLAB
2. Операции над матрицами и векторами
3. Структуры в MATLAB
4. Ячейки в MATLAB

1. MATLAB содержит в себе все распространенные математические функции, которые доступны по их имени при реализации алгоритмов. Например, функция sqrt() позволяет вычислять квадрат числа и может быть использована в программе следующим образом:

```
x = 2;
y = 4;
d = sqrt(x^2+y^2);  -вычисление евклидового расстояния
```

Аналогичным образом вызываются и все другие математические функции, представленные в табл. 3.

Таблица 3. Основные математические функции MATLAB

sqrt(x)	вычисление квадратного корня
exp(x)	возведение в степень числа e
pow2(x)	возведение в степень числа 2
log(x)	вычисление натурального логарифма
log10(x)	вычисление десятичного логарифма
log2(x)	вычисление логарифма по основанию 2
sin(x)	синус угла x, заданного в радианах
cos(x)	косинус угла x, заданного в радианах
tan(x)	тангенс угла x, заданного в радианах
cot(x)	котангенс угла x, заданного в радианах
asin(x)	арксинус
acos(x)	арккосинус
atan(x)	арктангенс
pi	число пи
round(x)	округление до ближайшего целого
fix(x)	усечение дробной части числа
floor(x)	округление до меньшего целого
ceil(x)	округление до большего целого
mod(x)	остаток от деления с учётом знака
sign(x)	знак числа
factor(x)	разложение числа на простые множители
isprime(x)	истинно, если число простое
rand	генерация псевдослучайного числа с равномерным законом распределения
randn	генерация псевдослучайного числа с нормальным законом распределения
abs(x)	вычисление модуля числа

Почти все элементарные функции допускают вычисления и с комплексными аргументами. Например:

res = sin(2+3i)*atan(4i)/(1 — 6i); - res = -1.8009 — 1.9190i или

exp(i*x) = cos(x)+i*sin(x);

Ниже показан пример задания вектора с именем a, и содержащий значения 1, 2, 3, 4:

a = [1 2 3 4]; - вектор-строка

Для доступа к тому или иному элементу вектора используется следующая конструкция языка:

disp(a(1)); - отображение значения 1-го элемента вектора

disp(a(2)); - отображение значения 2-го элемента вектора

disp(a(3)); - отображение значения 3-го элемента вектора

disp(a(4)); - отображение значения 4-го элемента вектора

т.е. нужно указать имя вектора и в круглых скобках написать номер индекса элемента, с которым предполагается работать. Например, для изменения значения 2-го элемента массива на 10 достаточно записать

a(2) = 10; - изменение значения 2-го элемента на 10

Часто возникает необходимость определения общего числа элементов в векторе, т.е. определения его размера. Это можно сделать, воспользовавшись функцией length() следующим образом:

N = length(a); - (N=4) число элементов массива a

Если требуется задать вектор-столбец, то это можно сделать так

a = [1; 2; 3; 4]; - вектор-столбец

или так

b = [1 2 3 4]'; - вектор-столбец

при этом доступ к элементам векторов осуществляется также как и для векторов-строк.

Следует отметить, что векторы можно составлять не только из отдельных чисел или переменных, но и из векторов. Например, следующий фрагмент программы показывает, как можно создавать один вектор на основе другого:

a = [1 2 3 4]; - начальный вектор a = [1 2 3 4]

b = [a 5 6]; - второй вектор b = [1 2 3 4 5 6]

Здесь вектор b состоит из шести элементов и создан на основе вектора a. Используя этот прием, можно осуществлять увеличение размера векторов в процессе работы программы:

a = [a 5]; - увеличение вектора a на один элемент

Недостатком описанного способа задания (инициализации) векторов является сложность определения векторов больших размеров, состоящих, например, из 100 или 1000 элементов. Чтобы решить данную задачу, в MATLAB существуют функции инициализации векторов нулями, единицами или случайными значениями:

a1 = zeros(1, 100); - вектор-строка, 100 элементов с- нулевыми значениями

a2 = zeros(100, 1); - вектор-столбец, 100 элементов с- нулевыми значениями

a3 = ones(1, 1000); - вектор-строка, 1000 элементов с- единичными значениями

a4 = ones(1000, 1); - вектор-столбец, 1000 элементов с - единичными значениями

a5 = rand(1000, 1); - вектор-столбец, 1000 элементов со- случайными значениями

Матрицы в MATLAB задаются аналогично векторам с той лишь разницей, что указываются обе размерности. Приведем пример инициализации единичной матрицы размером 3x3:

E = [1 0 0; 0 1 0; 0 01]; - единичная матрица 3x3

или

E = [1 0 0
0 1 0
0 0 1]; - единичная матрица 3x3

Аналогичным образом можно задавать любые другие матрицы, а также использовать приведенные выше функции zeros(), ones() и rand(), например:
A1 = zeros(10,10); - нулевая матрица 10x10 элементов
или
A2 = zeros(10); - нулевая матрица 10x10 элементов
A3 = ones(5); - матрица 5x5, состоящая из единиц
A4 = rand(100); - матрица 100x100, из случайных чисел

Для доступа к элементам матрицы применяется такой же синтаксис как и для векторов, но с указанием строки и столбца где находится требуемый элемент:
A = [1 2 3;4 5 6;7 8 9]; - матрица 3x3
disp(A(2,1)); - вывод на экран элемента, стоящего во второй строке первого столбца, т.е. 4
disp(A(1,2)); - вывод на экран элемента, стоящего в первой строке второго столбца, т.е. 2
Также возможны операции выделения указанной части матрицы, например:
B1 = A(:,1); - B1 = [1; 4; 7] – выделение первого столбца
B2 = A(2,:); - B2 = [1 2 3] – выделение первой строки
B3 = A(1:2,2:3); - B3 = [2 3; 5 6] – выделение первых двух строк и 2-го и 3-го столбцов матрицы A.

Размерность любой матрицы или вектора в MATLAB можно определить с помощью функции size(), которая возвращает число строк и столбцов переменной, указанной в качестве аргумента:
a = 5; - переменная a
A = [1 2 3]; - вектор-строка
B = [1 2 3; 4 5 6]; - матрица 2x3
size(a) - 1x1
size(A) - 1x3
size(B) - 2x3

2. В системе MATLAB достаточно просто выполняются математические операции над матрицами и векторами. Рассмотрим сначала простые операции сложения и умножения матриц и векторов. Пусть даны два вектора
a = [1 2 3 4 5]; - вектор-строка
b = [1; 1; 1; 1; 1]; - вектор-столбец
тогда умножение этих двух векторов можно записать так
c = a*b; - c=1+2+3+4+5=16
d = b*a; - d – матрица 5x5 элементов

В соответствии с операциями над векторами, умножение вектор-строки на вектор-столбец дает число, а умножение вектор-столбца на вектор-строку дает двумерную матрицу, что и является результатом вычислений в приведенном примере, т.е.

$$c = \sum_{i=1}^{5} a_i b_i = 1+2+3+4+5+6 = 16,$$

$$d = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}.$$

Сложение и вычитание двух векторов записывается так

a1 = [1 2 3 4 5];
a2 = [5 4 3 2 1];
c = a1+a2; - c = [1+5, 2+4, 3+3, 4+2, 5+1];
c = a2-a1; - c = [5-1, 4-2, 3-3, 2-4, 1-5];

Следует обратить внимание, что операции сложения и вычитания можно выполнять между двумя векторами-столбцами или двумя векторами-строками. Иначе MATLAB выдаст сообщение об ошибке, т.к. разнотипные векторы складывать нельзя. Так обстоит дело со всеми недопустимыми арифметическими операциями: в случае невозможности их вычисления система MATLAB сообщит об ошибке и выполнение программы будет завершено на соответствующей строке.

Аналогичным образом выполняются операции умножения и сложения между матрицами:
A = [1 2 3; 4 5 6; 7 8 9];
B = ones(3);
C = A+B; - сложение двух матриц одинакового размера
D = A+5; - сложение матрицы и числа
E = A*B; - умножение матрицы A на B
F = B*A; - умножение матрицы B на A
G = 5*A; - умножение матрицы на число

Операции вычисления обратной матрицы, а также транспонирования матриц и векторов, записываются следующим образом:
a = [1 1 1]; - вектор-строка
b = a'; - вектор-столбец, образованный транспонированием вектора-строки a.
A = [1 2 3; 4 5 6; 7 8 9]; - матрица 3x3 элемента
B = a*A; - B = [12 15 18] – вектор-строка
C = A*b; - C = [6; 15; 24] – вектор-столбец
D = a*A*a'; - D = 45 – число, сумма эл-ов матрицы A
E = A'; - E – транспонированная матрица A
F = inv(A); - F – обратная матрица A
G = A^-1; - G – обратная матрица A

Из приведенного примера видно, что операция транспонирования матриц и векторов обозначается символом ‘ (апостроф), который ставится после имени вектора или матрицы. Вычисление обратной матрицы можно делать путем вызова функции inv() или возводя матрицу в степень -1. Результат в обоих случаях будет одинаковым, а два способа вычисления сделано для удобства использования при реализации различных алгоритмов.

Если в процессе вычислений требуется поэлементно умножить, разделить или возвести в степень элементы вектора или матрицы, то для этого используются операторы:
.* - поэлементное умножение;
./ и .\ - поэлементные деления;
.^ - поэлементное возведение в степень.
Рассмотрим работу данных операторов на следующем примере.
a = [1 2 3]; - вектор-строка
b = [3 2 1]; - вектор-строка
c = a.*b; - c = [3 4 3]
A = ones(3); - матрица 3x3, состоящая из единиц
B = [1 2 3;4 5 6; 7 8 9]; - матрица 3x3
C = A.*B; - матрица 3x3, состоящая из $\left\| c_{ij} = a_{ij} b_{ij} \right\|_{i,j=\overline{1,3}}$
D = A./B; - матрица 3x3, состоящая из $\left\| d_{ij} = a_{ij} / b_{ij} \right\|_{i,j=\overline{1,3}}$

E = A.\B; - матрица 3x3, состоящая из $\left\| e_{ij} = b_{ij} / a_{ij} \right\|_{i,j=\overline{1,3}}$
F = A.^2; - возведение элементов матрицы A в квадрат

Рассмотрим несколько функций полезных при работе с векторами и матрицами.

Для поиска максимального значения элемента вектора используется стандартная функция max(), которая возвращает найденное максимальное значение элемента и его позицию (индекс):

```
a = [1 6 3 4];
[v, i] = max(a);          - v = 6, i = 2;
```

или

```
v = max(a);               - v = 6;
```

Приведенный пример показывает два разных способа вызова функции max(). В первом случае определяется и максимальное значение элемента и его индекс в векторе, а во втором – только максимальное значение элемента.

В случае с матрицами, данная функция определяет максимальные значения, стоящие в столбцах, как показано ниже в примере:

```
A = [4 3 5; 6 7 2; 3 1 8];
[V, I] = max(A);    - V=[6 7 8], I = [2 2 3]
V = max(A);         - V=[6 7 8]
```

Полный синтаксис функции max() можно узнать, набрав в командном окне MATLAB команду

help <название функции>

Аналогичным образом работает функция min(), которая определяет минимальное значение элемента вектора или матрицы и его индекс.

Другой полезной функцией работы с матрицами и векторами является функция sum(), которая вычисляет сумму значений элементов вектора или столбцов матрицы:

```
a = [3 5 4 2 1];
s = sum(a);   - s = 3+5+4+2+1=15
A = [4 3 5; 6 7 2; 3 1 8];
S1 = sum(A);  - S1=[13 11 15]
S2 = sum(sum(A));  - S2=39
```

При вычислении суммы S2 сначала вычисляется сумма значений элементов матрицы A по столбцам, а затем, по строкам. В результате, переменная S2 содержит сумму значений всех элементов матрицы A.

Для сортировки значений элементов вектора или матрицы по возрастанию или убыванию используется функция sort() следующим образом:

```
a = [3 5 4 2 1];
b1 = sort(a);                - b1=[1 2 3 4 5]
b2 = sort(a, ‘descend’);   - b2=[5 4 3 2 1]
b3 = sort(a, ‘ascend’);     - b3=[1 2 3 4 5]
```

для матриц

```
A = [4 3 5; 6 7 2; 3 1 8];
B1 = sort(A);      - B1=[3 1 2
                   -    4 3 5
                   -    6 7 8]
B2 = sort(A, ‘descend’);     - B2=[6 7 8
                    -     4 3 5
                    -     3 1 2]
```

Во многих практических задачах часто требуется найти определенный элемент в векторе или матрице. Это можно выполнить с помощью стандартной функции find(), которая

в качестве аргумента принимает условие, в соответствии с которым и находятся требуемые элементы, например:

```
a = [3 5 4 2 1];
b1 = find(a == 2);        - b1 = 4 – индекс элемента 2
b2 = find(a ~= 2);        - b2 = [1 2 3 5] – индексы без 2
b3 = find(a > 3);         - b3 = [2 3]
```

В приведенном примере символ ‘= =’ означает проверку на равенство, а символ ‘~=’ выполняет проверку на неравенство значений элементов вектора a. Более подробно об этих операторах будет описано в разделе условные операторы.

Еще одной полезной функцией работы с векторами и матрицами является функция mean() для вычисления среднего арифметического значения, которая работает следующим образом:

```
a = [3 5 4 2 1];
m = mean(a);              - m = 3
A = [4 3 5; 6 7 2; 3 1 8];
M1 = mean(A);             - M1 = [4.333 3.667 5.000]
M2 = mean(mean(A));       - M2 = 4.333
```

3. При разработке программ важным является выбор эффективного способа представления данных. Во многих случаях недостаточно объявить простую переменную или массив, а нужна более гибкая форма представления данных. Таким элементом может быть структура, которая позволяет включать в себя разные типы данных и даже другие структуры. Структуры задаются следующим образом:

```
S = struct('field1',VALUES1,'field2',VALUES2,...);
```

где field1 – название первого поля структуры; VALUES1 – переменная первого поля структуры, и т.д.

Приведем пример, в котором использование структуры позволяет эффективно представить данные. Таким примером будет инвентарный перечень книг, в котором для каждой книги необходимо указывать ее наименование, автора и год издания. Причем количество книг может быть разным, но будем полгать, что не более 100. Для хранения информации об одной книге будем использовать структуру, которая задается в MATLAB с помощью ключевого слова struct следующим образом:

```
S = struct('title','','author','','year',0);
```

В итоге задается структура с тремя полями: title, author и year. Каждое поле имеет свой тип данных и значение.

Для того, чтобы записать в эту структуру конкретные значения используется оператор ‘.’ (точка) для доступа к тому или иному полю структуры:

```
S.title = 'Евгений Онегин';
S.author = 'Пушкин';
S.year = 2000;
```

и таким образом, переменная S хранит информацию о выбранной книге.

Однако по условиям задачи необходимо осуществлять запись не по одной, а по 100 книгам. В этом случае целесообразно использовать вектор структур lib, который можно задать следующим образом:

```
lib(100,1) = struct('title','','author','','year',0);
```

и записывать информацию о книгах так:

```
lib(1).title = 'Евгений Онегин';
lib(1).author = 'Пушкин';
lib(1).year = 2000;
```

Данный пример показывает удобство хранения информации по книгам. Графически массив структур можно представить в виде таблицы, в которой роль столбцов играют поля, а роль строк элементы массива структур (рис. 13).

При работе со структурами полезными являются следующие функции:

isstruct(S) – возвращает истину, если аргумент структура

isfield(S, 'name') – возвращает истину, если имеется такое поле

fieldnames(S) – возвращает массив строк с именами всех полей

которые позволяют программно определить всю необходимую информацию о той или иной структуре и корректно выполнять обработку ее полей.

	название	автор	год издания
lib[1]	lib[1].title	lib[1].author	lib[1].year
lib[2]	lib[2].title	lib[2].author	lib[2].year
lib[3]	lib[3].title	lib[3].author	lib[3].year
⋮			
lib[100]	lib[100].title	lib[100].author	lib[100].year

Рис. 13. Графическое представление массива структур хранения информации по 100 книгам

4. Ячейки также как и структуры могут содержать разные типы данных, объединенные одной переменной, но в отличии от вектора структур, вектор ячеек может менять тип данных в каждом элементе. Таким образом, вектор ячеек является универсальным контейнером – его элементы могут содержать любые типы и структуры данных, с которыми работает MATLAB – векторы чисел любой размерности, строки, векторы структур и другие (вложенные) векторы ячеек.

Методы создания вектора ячеек похожи на методы создания вектора структур. Как и в случае структур, векторы ячеек могут быть созданы либо путём последовательного присваивания значений отдельным элементам массива, либо созданы целиком при помощи специальной функции cell(). Однако в любом случае важно различать ячейку (элемент вектора ячеек) и её содержимое. Ячейка – это содержимое плюс некоторая оболочка (служебная структура данных) вокруг этого содержимого, позволяющая хранить в ячейке произвольные типы данных любого размера.

Приведем пример создания вектора ячеек хранения разных типов данных.

```
book = struct('title','Онегин','author','Пушкин','year',2000);
MyCell(1)={book};
MyCell(2)={'Пушкин'};
MyCell(3)={2000};
```

Здесь задан вектор ячеек MyCell с тремя элементами. Первый элемент соответствует структуре, второй – строке, а третий – числу. В этом и заключается особенность организации данных с помощью ячеек: у каждого элемента свой тип данных.

Для обращения к содержимому той или иной ячейки используются фигурные скобки, внутри которых ставится индекс элемента с которым предполагается работа:

```
MyCell{1}
```

выведет на экран

```
    title: 'Евгений Онегин'
   author: 'Пушкин'
     year: 2000
```

Если же используются круглые скобки, то будет возвращена структура данных вместо отдельных значений, например

```
MyCell(1)
```

выведет

```
[1x1 struct]
```

Для того чтобы задать вектор или матрицу ячеек с пустыми (неопределенными) значениями, используется функция cell() как показано ниже.

```
MyCellArray = cell(2, 2);
```

задается матрица размером 2x2. Данную инициализацию целесообразно выполнять когда нужно определить большой вектор или матрицу ячеек и в цикле задавать их значения. В этом случае MATLAB сразу создает массивы нужных размеров, в результате чего повышается скорость выполнения программ.

В заключении рассмотрим возможность программирования функции с произвольным числом аргументов благодаря использованию ячеек. Для этого в качестве аргумента функции указывается ключевое слово varargin, которое интерпретируется внутри функции как вектор ячеек с переданными аргументами:

```
function len = SumSquare( varargin )
n= length( varargin );
len = 0;
for k = 1 : n
      len = len + varargin{ k }(1)^2 +varargin{ k }(2)^2;
end
```

Данная функция вычисляет сумму квадратов чисел, которые передаются ей следующим образом:

```
SumSquare( [ 1 2], [3 4] )  - ответ 30
SumSquare( [ 1 2], [3 4], [ 5 6] ) - ответ 91
```

Таким образом, благодаря использованию ячеек функции SumSquare() можно передавать произвольное число двумерных векторов.

Контрольные вопросы:

1. Какие математические функции существует в программе MATLAB?
2. Как выполняются математические операции в программе MATLAB?
3. Что такое структуры в программе MATLAB?
4. Какие типы данных содержатся в ячейках?
5. Как решают уравнения с одной переменной в программе MATLAB?
6. Порядок выполнения действий над матрицами и векторами в программе MATLAB?

Основные математические операции в MATHCAD и типы данных

1. Некоторые стандартные функции MATHCAD
2. Операции над матрицами и векторами
3. Аппроксимация, интерполяция и экстраполяция

1.Рассмотрим некоторые стандартные функции системы MATHCAD. Введем специальные обозначения для аргументов функций. Пусть первый символ имени аргумента обозначает его тип:

M - квадратная матрица;

V - вектор (матрица из одного столбца);

A - произвольная матрица;
S - симметричная матрица;
G - произвольная матрица или число;
X - вектор или число;
Z - комплексная матрица или число;
z - комплексное число;
прочие символы - скалярные величины.

Экспоненциальные и логарифмические функции

exp(X) - экспонента от X;
ln(X) - натуральный логарифм от X;
log(X) - десятичный логарифм от X;
log(X,b) - логарифм от X по основанию b.

Гиперболические и тригонометрические (прямые и обратные) функции

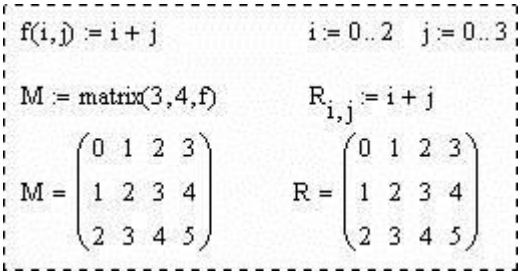
sin(X), cos(X), tan(X), cot(X), sec(X), csc(X) - соответственно синус, косинус, тангенс, котангенс, секанс, косеканс от X, причем аргументы указываются в *радианах*;
sinh(X), cosh(X), tanh(X), coth(X), sech(X), csch(X) - аналогичные гиперболические функции;
asin(z), acos(z), atan(z), acot(z), asec(z), acsc(z) - соответственно арксинус, арккосинус, арктангенс, арккотангенс, арксеканс, арккосеканс от z.

Функции для работы с комплексными числами

Re(Z), Im(Z) - соответственно вещественная и мнимая части комплексного числа Z;
arg(z) - аргумент комплексного числа z (в радианах).

Матричные функции

length(V) - возвращает число элементов вектора V;
cols(A) - возвращает число столбцов матрицы A;
rows(A) - возвращает число строк матрицы A;
matrix(m,n,f) - матрица размером mxn, значения элементов матрицы определяются f - функцией f(i,j) от двух переменных (номера строки и номера столбца). Эта функция должна быть предварительно определена пользователем;



identity(n) - единичная матрица $n \times n$;
tr(M) - след матрицы M (сумма элементов главной диагонали);
rank(A) - ранг матрицы M;
norme(M) - эвклидова норма матрицы M, то есть корень квадратный из суммы квадратов всех элементов;
eigenvals(M) - вектор, элементы которого являются собственными числами матрицы M;
eigenvecs(M) - матрица, состоящая из нормализованных собственных векторов матрицы M;
cholesky(S) - возвращает нижнетреугольную матрицу L - результат разложения Холецкого вида $L \cdot L^T = S$;
lu(M) - возвращает матрицу размера $n \times 3n$, состоящую из трех соединенных матриц P, L, U, являющихся результатом LU-разложения вида $P \cdot M = L \cdot U$.

2. Пример вычислений с матричными функциями: нахождение собственного числа путем решения матричного уравнения $\det(M - \lambda E) = 0$ и с помощью функции eigenvals.

$$M := \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \lambda := 0$$

Given

$$|M - \lambda \cdot \text{identity}(\text{cols}(M))| = 0$$

$$\text{Find}(\lambda) = -1$$

$$\text{eigenvals}(M) = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

Элементы статистического анализа данных

gmean(G1,G2,G3…) - среднее геометрическое аргументов;
mean(G1,G2,G3…) - среднее арифметическое аргументов;
var(G1,G2,G3…) - дисперсия;
stdev(G1,G2,G3…) - среднеквадратичное отклонение.
Дискретные преобразования
fft(V1), ifft(V2) - прямое и обратное быстрые преобразования Фурье над вещественными данными. V1 - вектор из 2^m элементов, V2 - вектор из $1 + 2^{m-1}$ элементов, m>2;
cfft(A), icfft(A) - прямое и обратное преобразования Фурье над вещественными и комплексными векторами и матрицами;
wave(V), iwave(V) - прямое и обратное вейвлет-преобразования, V - вектор из 2^m элементов, m - целое число.
3. Аппроксимация, интерполяция и экстраполяция

Аппроксимация - поиск функции, которая с заданной степенью точности описывает исходные данные.

Интерполяция - определение наиболее правдоподобных промежуточных значений в интервале между известными значениями (подбор гладкой кривой, проходящей через заданные точки или максимально близко к ним).

Экстраполяция - определение наиболее правдоподобных последующих значений на основании анализа предыдущих значений (предсказание дальнейшего поведения неизвестной функции).

Применяются следующие функции MATHCAD:
regress(VX,VY,k) - возвращает вектор данных, используемый для поиска интерполирующего полинома $\left(a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k\right)$ порядка k. Полином должен описывать данные, состоящие из упорядоченных значений аргумента (VX) и соответствующих значений неизвестной функции (VY), то есть график полинома должен проходить через все точки, заданные координатами (VX,VY), или максимально близко к этим точкам;

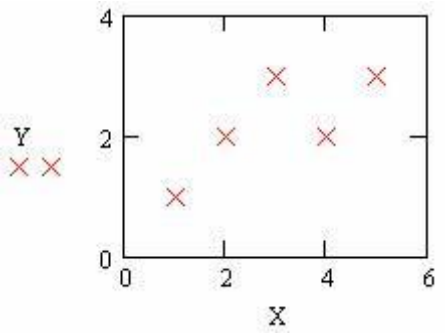
interp(VS,VX,VY,x) - возвращает интерполированное значение неизвестной функции при значении аргумента x. VS - вектор значений, который вернула функция regress. VX,VY - те же данные, что и для regress. Функции interp и regress используются в паре;

predict(V,m,n) - возвращает вектор из n предсказанных значений на основании анализа m предыдущих значений из вектора V. Предполагается, что значения функции в векторе V были получены при значениях аргумента, взятых последовательно, с одинаковым шагом. Используется алгоритм линейной предикции. Наиболее целесообразно использовать predict для предсказания значений по данным, в которых отмечены колебания.

Для интерполяции система MATHCAD использует подход, основанный на применении *метода наименьших квадратов.*

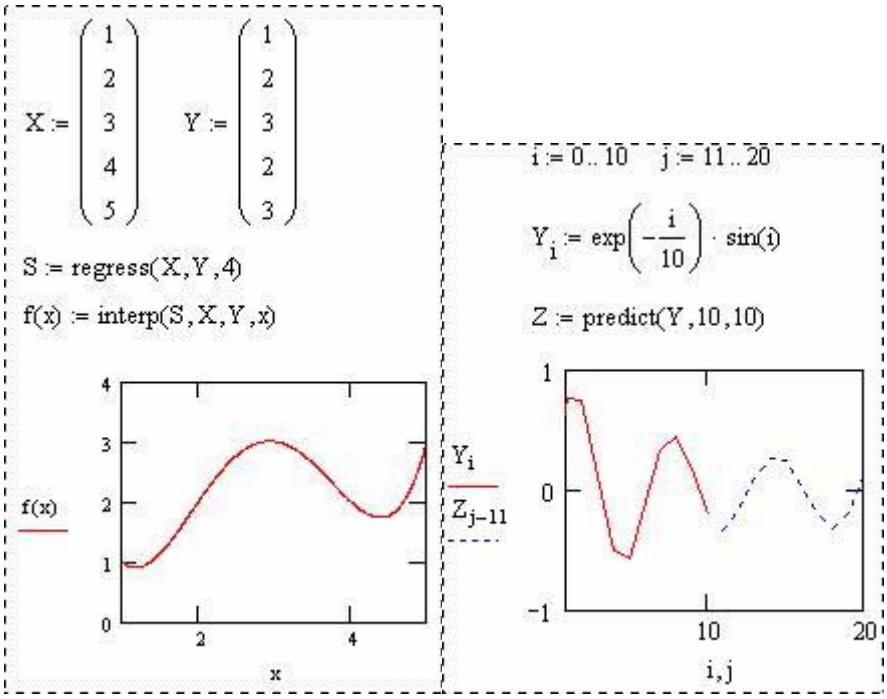
Примеры интерполяции и экстраполяции:

1. Пусть заданы координаты пяти точек (1; 1), (2; 2), (3; 3), (4; 2), (5; 3), представляющих результаты измерения значений некоторой неизвестной функции при различных значениях x. Необходимо подобрать интерполирующую функцию (гладкую кривую), проходящую через заданные точки.



2. Дана функция $y(i) = e^{-i/10}*\sin(i)$. Известны значения данной функции при i = 0, 1, …, 10. Основываясь на десяти последних значениях, необходимо предсказать последующие десять значений.

Решения показаны на рис.14.



а) б)

Рис. 14. Решения в MATHCAD первой (а) и второй (б) задач

Нахождение корней полинома

polyroots(V) - возвращает вектор, содержащий все корни полинома $a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k$, заданного вектором-столбцом коэффициентов

$$V = \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_k \end{pmatrix}.$$

Прочие функции

max(G1,G2,…) - максимальное значение среди аргументов;
min(G1,G2,…) - минимальное значение среди аргументов;
if(a,b,c) - возвращает b, если $a \neq 0$, иначе возвращает c;
sign(a) - возвращает –1, 0 или 1 в зависимости от знака числа a.
На рис.15 показан пример применения функции if.

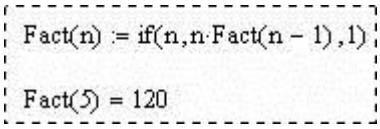


Рис. 15. Функция, вычисляющая факториал

Контрольные вопросы:

1. Какие математические функции существует в программе MATHCAD?
2. Как выполняются математические операции в программе MATHCAD?
3. Порядок выполнения действий над матрицами и векторами в программе MATHCAD?
4. Выполнение вычислений основных математических функций в программе MATHCAD?
5. Аппроксимация, интерполяция и экстраполяция?

Условные операторы и циклы в MATLAB

1. Условный оператор if
2. Условный оператор switch
3. Оператор цикла while
4. Оператор цикла for

Вторым шагом создания полноценных программ на языке MATLAB является изучение операторов ветвления и циклов. С их помощью можно реализовывать логику выполнения математических алгоритмов и создавать повторяющиеся (итерационные, рекуррентные) вычисления.

1. Для того чтобы иметь возможность реализовать логику в программе используются условные операторы. Умозрительно эти операторы можно представить в виде узловых пунктов, достигая которых программа делает выбор по какому из возможных направлений двигаться дальше. Например, требуется определить, содержит ли некоторая переменная arg

положительное или отрицательное число и вывести соответствующее сообщение на экран. Для этого можно воспользоваться оператором if (если), который и выполняет подобные проверки.

В самом простом случае синтаксис данного оператора if имеет вид:

if <выражение>
<операторы>
end

Если значение параметра «выражение» соответствует значению «истинно», то выполняется оператор, иначе он пропускается программой. Следует отметить, что «выражение» является условным выражением, в котором выполняется проверка некоторого условия. В табл. 4 представлены варианты простых логических выражений оператора if.

Таблица 4. Простые логические выражения

if a < b	Истинно, если переменная a меньше переменной b и ложно в противном случае.
if a > b	Истинно, если переменная a больше переменной b и ложно в противном случае.
if a == b	Истинно, если переменная a равна переменной b и ложно в противном случае.
if a <= b	Истинно, если переменная a меньше либо равна переменной b и ложно в противном случае.
if a >= b	Истинно, если переменная a больше либо равна переменной b и ложно в противном случае.
if a ~= b	Истинно, если переменная a не равна переменной b и ложно в противном случае.

Ниже представлен пример реализации функции sign(), которая возвращает +1, если число больше нуля, -1 – если число меньше нуля и 0, если число равно нулю:

```
function my_sign
x = 5;
if x > 0
disp(1);
end
if x < 0
disp(-1);
end
if x == 0
disp(0);
end
```

Анализ приведенного примера показывает, что все эти три условия являются взаимоисключающими, т.е. при срабатывании одного из них нет необходимости проверять другие. Реализация именно такой логики позволит увеличить скорость выполнения программы. Этого можно добиться путем использования конструкции

```
if <выражение>
<операторы1>      - выполняются, если истинно условие
else
<операторы2>      - выполняются, если условие ложно
end
```

Тогда приведенный выше пример можно записать следующим образом:

```
function my_sign
x = 5;
if x > 0
disp(1);
else
if x < 0
disp(-1);
else
disp(0);
end
end
```

В данной программе сначала выполняется проверка на положительность переменной x, и если это так, то на экран выводится значение 1, а все другие условия игнорируются. Если же первое условие оказалось ложным, то выполнение программы переходит по else (иначе) на второе условие, где выполняется проверка переменной x на отрицательность, и в случае истинности условия, на экран выводится значение -1. Если оба условия оказались ложными, то выводится значение 0.

Приведенный выше пример можно записать в более простой форме, используя еще одну конструкцию оператора if языка MATLAB:

```
if <выражение1>
<операторы1>       - выполняются, если истинно выражение1
elseif <выражение2>
<операторы2>       - выполняются, если истинно выражение2
...
elseif <выражениеN>
<операторыN>       - выполняются, если истинно выражениеN
end
```

и записывается следующим образом:

```
function my_sign
x = 5;
if x > 0
    disp(1);       - выполняется, если x > 0
elseif x < 0
    disp(-1);      - выполняется, если x < 0
else
    disp(0);       - выполняется, если x = 0
end
```

С помощью условного оператора if можно выполнять проверку более сложных (составных) условий. Например, необходимо определить: попадает ли переменная x в диапазон значений от 0 до 2? Это можно реализовать одновременной проверкой сразу двух условий: x >= 0 и x <=2. Если эти оба условия истинны, то x попадает в диапазон от 0 до 2.

Для реализации составных условий в MATLAB используются логические операторы:

& - логическое И
| - логическое ИЛИ
~ - логическое НЕ

Рассмотрим пример использования составных условий. Пусть требуется проверить попадание переменной x в диапазон от 0 до 2. Программа запишется следующим образом:

```
function my_if
x = 1;
if x >= 0 & x <= 2
```

```
  disp('x принадлежит диапазону от 0 до 2');
else
  disp('x  не принадлежит диапазону от 0 до 2');
end
```

Во втором примере выполним проверку на не принадлежность переменной x диапазону от 0 до 2. Это достигается срабатыванием одного из двух условий: x < 0 или x > 2:

```
function my_if
x = 1;
if x < 0 | x > 2
  disp('x не принадлежит диапазону от 0 до 2');
else
  disp('x  принадлежит диапазону от 0 до 2');
end
```

Используя логические операторы И, ИЛИ, НЕ, можно создавать разнообразные составные условия. Например, можно сделать проверку, что переменная x попадает в диапазон от -5 до 5, но не принадлежит диапазону от 0 до 1. Очевидно, это можно реализовать следующим образом:

```
function my_if
x = 1;
if (x >= -5 & x <= 5) & (x < 0 | x > 1)
  disp('x принадлежит [-5, 5], но не входит в [0, 1]');
else
  disp('x или не входит в [-5, 5] или в [0, 1]');
end
```

Обратите внимание, что при сложном составном условии были использованы круглые скобки. Дело в том, что приоритет операции И выше приоритета операции ИЛИ, и если бы не было круглых скобок, то условие выглядело бы так: (x >= -5 и x <= 5 и x < 0) или x > 1. Очевидно, что такая проверка давала бы другой результат от ожидаемого.

Круглые скобки в программировании используются для изменения приоритетов выполнения операторов. Подобно арифметическим операторам, логические также могут быть изменены по желанию программиста. Благодаря использованию круглых скобок, сначала выполняется проверка внутри них, а, затем, за их пределами. Именно поэтому в приведенном выше примере они необходимы для достижения требуемого результата.

Приоритет логических операций следующий:

НЕ (~) – самый высокий приоритет;

И (&) – средний приоритет;

ИЛИ (|) – самый низкий приоритет.

2. В некоторых задачах программирования требуется выполнять проверку на равенство некоторой переменной константным значениям. Например, нужно преобразовать малые буквы в заглавные. В этом случае необходимо произвести проверку текущего символа со всеми возможными буквами алфавита и при равенстве с одной из них, заменить ее на заглавную. Для решения таких задач удобнее пользоваться условным оператором switch, который имеет следующий синтаксис:

```
switch expr
        case case_expr,
          <операторы1>
        case {case_expr1, case_expr2, case_expr3,...}
          <операторы2>
        ...
        otherwise,
```

<операторы>
end

Здесь expr – переменная, значение которой проверяется на равенство тем или иным константам; case_expr – константы, с которым сравнивается значение переменной; otherwise – ключевое слово, для выполнения операторов, при всех ложных условиях.

Приведем пример работы данного оператора для преобразования малых букв латинского алфавита в заглавные.

```
function upper_symbol
ch='c';
switch ch
   case 'a', ch='A';
   case 'b', ch='B';
   case 'c', ch='C';
   case 'd', ch='D';
   case 'e', ch='E';
  ...
   case 'z', ch='Z';
end

disp(ch);
```

В данной программе задается символьная переменная ch со значением c. Затем, с помощью оператора switch проверяется ее значение со всеми возможными малыми буквами латинского алфавита от a до z. Как только одно из условий сработало, оператор switch завершает свою работу и выполнение программы переходит на функцию disp(), которая отображает значение переменной ch на экран.

3.Язык программирования MATLAB имеет два оператора цикла: while и for. С их помощью, например, выполняется программирование рекуррентных алгоритмов, подсчета суммы ряда, перебора элементов массива и многое другое.

В самом простом случае цикл в программе организуется с помощью оператора while, который имеет следующий синтаксис:

while <условие>
  <операторы>
end

Здесь <условие> означает условное выражение подобное тому, которое применяется в операторе if, и цикл while работает до тех пор, пока это условие истинно.

Следует обратить внимание на то, что если условие будет ложным до начала выполнения цикла, то операторы, входящие в цикл, не будут выполнены ни разу.

Приведем пример работы цикла while для подсчета суммы ряда $S = \sum_{i=1}^{20} i$:

```
function sum_i
S = 0;              - начальное значение суммы
i=1;                - счетчик суммы
while i <= 20       - цикл (работает пока i <= 20)
   S=S+i;           - подсчитывается сумма
   i=i+1;           - увеличивается счетчик на 1
end                 - конец цикла
disp(S);            - отображение суммы 210 на экране
```

Теперь усложним задачу и будем подсчитывать сумму ряда $S = \sum_{i=1}^{20} i$, пока $S \le 20$. Здесь в операторе цикла получается два условия: либо счетчик по i доходит до 20, либо значение суммы S превысит 20. Данную логику можно реализовать с помощью составного условного выражения в операторе цикла while:

```
function sum_i
S = 0;              - начальное значение суммы
i=1;                - счетчик суммы
while i <= 20 & S <= 20 - цикл (работает пока i<=10 и S<=20
    S=S+i;          - подсчитывается сумма
    i=i+1;          - увеличивается счетчик на 1
end                 - конец цикла
disp(S);            - отображение суммы 21 на экране
```

Приведенный пример показывает возможность использования составных условий в цикле while. В общем случае в качестве условного выражения можно записывать такие же условия, что и в условном операторе if.

Работу любого оператора цикла, в том числе и while, можно принудительно завершить с помощью оператора break. Например, предыдущую программу можно переписать следующим образом с использованием оператора break:

```
function sum_i
S = 0;              - начальное значение суммы
i=1;                - счетчик суммы
while i <= 20       - цикл (работает пока i<=10
    S=S+i;          - подсчитывается сумма
    i=i+1;          - увеличивается счетчик на 1
   if S > 20        - если S > 20,
      break;        - то цикл завершается
    end
end                 - конец цикла
disp(S);            - отображение суммы 21 на экране
```

В данном примере второе условие завершения цикла, когда S будет больше 20, записано в самом цикле и с помощью оператора break осуществляется выход из цикла на функцию disp(), стоящую сразу после цикла while.

Второй оператор управления выполнением цикла continue позволяет пропускать выполнение фрагмента программы, стоящий после него. Например, требуется подсчитать сумму элементов массива

```
a = [1 2 3 4 5 6 7 8 9];
```

исключая элемент с индексом 5. Такую программу можно записать следующим образом:

```
function sum_array

S = 0;                  - начальное значение суммы
a = [1 2 3 4 5 6 7 8 9]; - массив
i=0;                    - счетчик индексов массива
while i < length(a)     - цикл (работает пока i меньше длины массива a)
    i=i+1;              - увеличивается счетчик индексов на 1
   if i == 5            - если индекс равен 5
      continue;         - то его не подсчитываем
    end
```

```
    S=S+a(i);          - подсчитывается сумма элементов
end                    - конец цикла
disp(S);               - отображение суммы 40 на экране
```

Следует отметить, что в данной программе увеличение индекса массива i происходит до проверки условия. Это сделано для того, чтобы значение индекса увеличивалось на 1 на каждой итерации работы цикла. Если увеличение счетчика i записать как в предыдущих примерах, т.е. после подсчета суммы, то из-за оператора continue его значение остановилось бы на 5 и цикл while работал бы «вечно».

4.Часто при организации цикла требуется перебирать значение счетчика в заданном диапазоне значений и с заданным шагом изменения. Например, чтобы перебрать элементы вектора (массива), нужно организовать счетчик от 1 до N с шагом 1, где N – число элементов вектора. Чтобы вычислить сумму ряда, также задается счетчик от a до b с требуемым шагом изменения step. И так далее. В связи с тем, что подобные задачи часто встречаются в практике программирования, для их реализации был предложен свой оператор цикла for, который позволяет проще и нагляднее реализовывать цикл со счетчиком.

Синтаксис оператора цикла for имеет следующий вид:

```
for <счетчик> = <начальное значение>:<шаг>:<конечное значение>
    <операторы цикла>
end
```

Рассмотрим работу данного цикла на примере реализации алгоритма поиска максимального значения элемента в векторе:

```
function search_max
a = [3 6 5 3 6 9 5 3 1 0];
m = a(1);              - текущее максимальное значение
for i=1:length(a)      - цикл от 1 до конца вектора с шагом 1 (по умолчанию)
   if m < a(i)         - если a(i) > m,
      m = a(i);        - то m = a(i)
   end
end                    - конец цикла for
disp(m);
```

В данном примере цикл for задает счетчик i и меняет его значение от 1 до 10 с шагом 1. Обратите внимание, что если величина шага не указывается явно, то он берется по умолчанию равным 1.

В следующем примере рассмотрим реализацию алгоритма смещения элементов вектора вправо, т.е. предпоследний элемент ставится на место последнего, следующий на место предпоследнего, и т.д. до первого элемента:

```
function queue
a = [3 6 5 3 6 9 5 3 1 0];
disp(a);
for i=length(a):-1:2   - цикл от 10 до 2 с шагом -1
    a(i)=a(i-1);       - смещаем элементы вектора a
end                    - конец цикла for
disp(a);
```

Результат работы программы

```
3 6 5 3 6 9 5 3 1 0
3 3 6 5 3 6 9 5 3 1
```

Приведенный пример показывает, что для реализации цикла со счетчиком от большего значения к меньшему, нужно явно указывать шаг, в данном случае, -1. Если этого не сделать, то цикл сразу завершит свою работу и программа будет работать некорректно.

Рассмотрим работу оператора цикла for на примере моделирования случайной последовательности с законом изменения

$$x_i = rx_{i-1} + \xi_i,$$

где r - коэффициент от -1 до 1; ξ_i - нормальная случайная величина с нулевым математическим ожиданием и дисперсией

$$\sigma_\xi^2 = \sigma_x^2(1-r^2),$$

где σ_x^2 - дисперсия моделируемого случайного процесса. При этом первый отсчет x_1 моделируется как нормальная случайная величина с нулевым математическим ожиданием и дисперсией σ_x^2. Программа моделирования имеет следующий вид:

```
function modeling_x
r = 0.95;                    - коэффициент модели
N = 100;                     - число моделируемых точек
ex = 100;                    - дисперсия процесса
et = ex*(1-r^2);             - дисперсия случайной добавки ξi
x = zeros(N,1);              - инициализация вектора x
x(1) = sqrt(ex)*randn;       - моделирование 1-го отсчета
for i=2:N                    - цикл от 2 до N
    x(i)=r*x(i-1)+sqrt(et)*randn;        - моделирование СП
end                          - конец цикла
plot(x);                     - отображение СП в виде графика
```

При выполнении данной программы будет показана реализация смоделированной случайной последовательности $\mathbf{x}$.

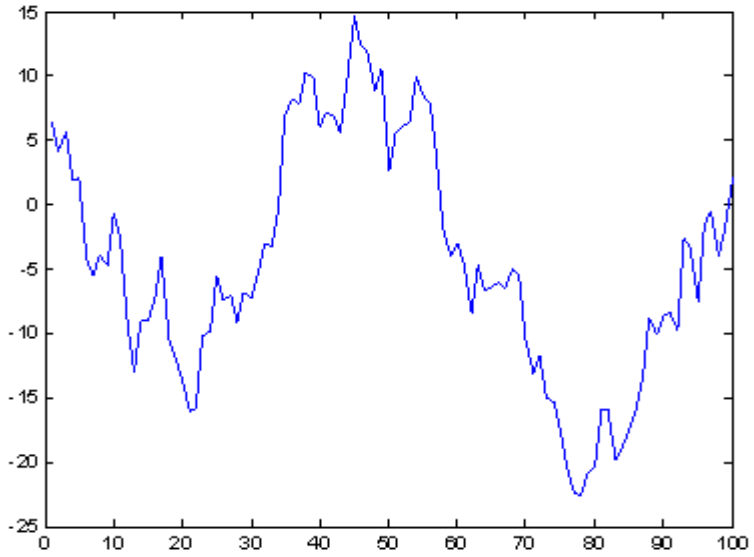


Рис. 16. Результат моделирования случайной последовательности.

Работа программы начинается с определения переменных r, σ_x^2 (в программе переменная ex) и N для реализации указанной модели. Затем вычисляется дисперсия $\sigma_\xi^2 = \sigma_x^2(1-r^2)$ и моделируется первый отсчет случайного процесса с помощью функции randn. Функция randn выполняет генерацию нормальных случайных величин с нулевым средним и единичной дисперсией. Чтобы сгенерировать случайную величину с дисперсией

σ_x^2 достаточно случайную величину с единичной дисперсией умножить на $\sqrt{\sigma_x^2} = \sigma_x$, т.к. дисперсия – это средний квадрат случайной величины относительно математического ожидания. В результате имеем программную строчку

x(1) = sqrt(ex)*randn;

Затем, реализуется цикл for со счетчиком i от 2 до N с шагом 1. Внутри цикла выполняется моделирование оставшихся N-1 отсчетов случайного процесса в соответствии с приведенной выше формулой. В последней строчке программы записана функция plot(), которая выводит смоделированную последовательность на экран в виде графика. Более подробно работа с выводом графиков на экран будет рассмотрена в следующей главе.

Контрольные вопросы:

1. Что такое оператор if
2. Что такое.условный оператор switch
3. Что такое оператор цикла while
4. Что такое оператор цикла for

Способы решения в программе MATH CAD

1. Решение систем уравнений с помощью блока Given-Find
2. Работа с последовательностями. Законы вычисления элементов матриц
3. Способы решения дифференциальных уравнений

1. Решение систем уравнений MATHCAD осуществляет численными методами. При этом должно быть задано некоторое начальное приближение для тех переменных, значение которых необходимо найти. Основываясь на этих начальных данных, MATHCAD будет последовательно уточнять решение до тех пор, пока не подберет наиболее точные значения. Проблемы возникают, когда нелинейная система имеет несколько решений. За один раз MATHCAD находит только одно решение, которое обычно более близко к заданному начальному приближению. Поэтому в таких случаях необходимо решать систему несколько раз с различными начальными приближениями.

Решающий блок состоит из нескольких компонент, следующих на листе *в строго определенном порядке*:

1. Начальное приближение (присваивание начальных значений переменным).
2. Директива *Given*, которую необходимо набрать с клавиатуры.
3. Уравнения, которые необходимо решить. Уравнения вводятся в обычной математической форме, но вместо простого знака равенства «=» используется оператор логического равенства (вводится путем нажатия *Ctrl*-=).
4. Обращение к функции *Find*. Аргументами функции являются имена переменных, относительно которых решается система. Функция возвращает вектор значений, где первый элемент соответствует первой переменной в списке аргументов, второй элемент - второй переменной и так далее.

Пример. Решим систему нелинейных уравнений:

$$\begin{cases} x^2 - x = 2; \\ x - 0{,}5y = 0. \end{cases}$$

Данная система имеет два решения. Найдем одно из них (рис. 17) с начальным приближением x = 0; y = 0.

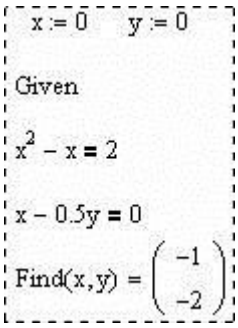


Рис. 17. Решение системы в MATHCAD

Последняя запись - вектор (-1; -2) есть значение, которое вернула функция Find, то есть одно из решений системы. Найти второе решение можно, если взять другое начальное приближение x = 2; y = 2. Тогда функция Find вернет вектор (2; 4).

Начиная с MATHCAD 2000 существует возможность *одновременно найти несколько решений*. Для этого система уравнений и начальные приближения должны быть переписаны в векторной форме (рис. 18). Каждая переменная будет вектором, содержащим столько компонент, сколько решений находится. В системе изменения коснутся преимущественно членов с перемножением переменных. Допустим, что в уравнении присутствует выражение x*x. Если $x = (x_1; x_2)$ - вектор, то $x \cdot x = x_1 \cdot x_1 + x_2 \cdot x_2$. Нам же необходим результат поэлементного перемножения $(x_1 \cdot x_1; x_2 \cdot x_2)$. Для этого существует специальная операция, записываемая как $\overrightarrow{(x \cdot x)}$.

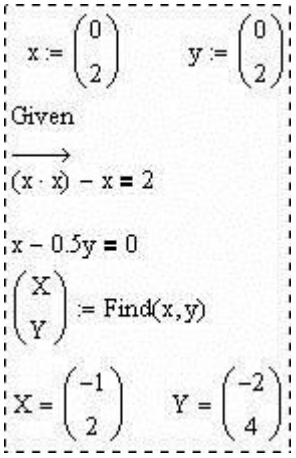


Рис. 18. Пример одновременного поиска нескольких решений

Изменения коснулись и части получения результата. В данном случае функция Find вернет вектор из двух элементов, которые мы обозначили как X и Y. Каждый из этих элементов есть вектор значений x или y для решений. Соответственно первое решение - (-1; -2); второе решение - (2; 4).

Аналитическое решение линейных и нелинейных систем уравнений

Данное решение используется для получения решений в общем виде. Обычно при этом система уравнений записывается только с использованием буквенных обозначений переменных, без конкретных чисел. Для получения аналитического решения (рис. 19, 20) используется оператор аналитического вычисления «$\rightarrow$» вместо оператора числового вычисления «=».

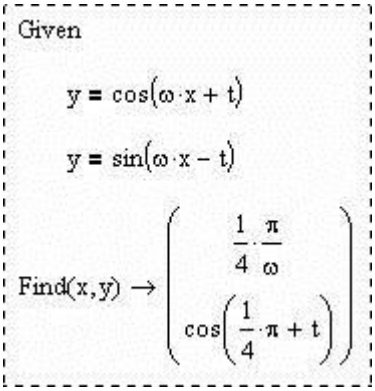


Рис. 19. Пример аналитического решения нелинейной системы

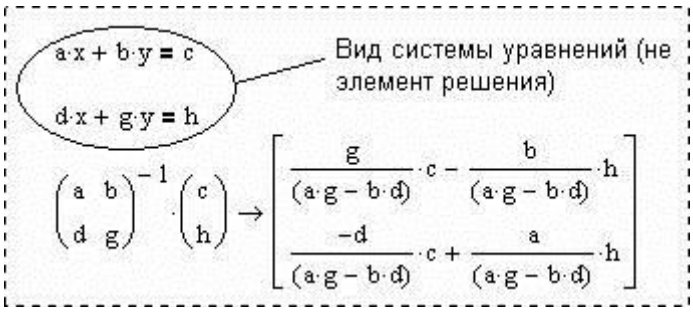


Рис. 20. Пример аналитического решения линейной системы

Следует обратить внимание, что здесь при решении системы нелинейных уравнений в блоке Given-Find уже нет необходимости указывать начальные приближения, поскольку решение идет не численными, а символьными методами (используется ядро математической системы Maple).

2. MATHCAD позволяет задавать значения элементов матриц не только по отдельности, но и путем определения общих законов вычисления элементов. При этом возникает необходимость определить границы действия законов, то есть диапазоны изменения индексов матриц. Для этого широко используются последовательности.

Замечание: в MATHCAD нумерация элементов векторов и матриц начинается с нуля.

*Пример***.** Ряд чисел Фибоначчи определяется по следующим правилам:

$$F_1 = 1;\ F_2 = 1;$$

$$F_i = F_{i-1} + F_{i-2};\ i>2.$$

Данный ряд бесконечен. Но мы не можем определить в MATHCAD бесконечные последовательности, возможно лишь нахождение части ряда. Поэтому вместо указания i>2

необходимо задать конкретный диапазон изменения индекса i, например нас могут интересовать члены ряда со третьего по восьмой. Решение показано на рис. 21.

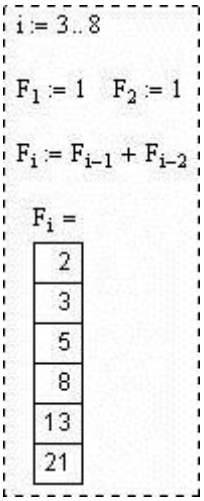


Рис. 21. Вычисление части ряда чисел Фибоначчи в MATHCAD

Здесь сначала определен диапазон изменения индекса i, затем заданы значения двух первых членов ряда и записана общая формула для вычисления члена с индексом i. Последняя часть (столбец значений) есть часть ряда F_3 .. F_8, вычисленная системой. Такой порядок записи выражений объясняется тем, что MATHCAD вычисляет формулы последовательно, в порядке «слева направо и сверху вниз». При вычислении члена F_3 по заданной рекуррентной формуле системе уже должны быть известны реальные значения F_1 и F_2. Поэтому они должны быть записаны до этой формулы.

Пример. Рассмотрим численное интегрирование явным методом Эйлера.

3. Сначала кратко рассмотрим суть метода. Пусть дано дифференциальное уравнение в общем виде:

$$\frac{dx}{dt} = F(x,t) \quad ; \; x(t_0) = x_0;$$

$$t \in [t_0; T].$$

Необходимо проинтегрировать данное уравнение. Если функция F(x,t) достаточно сложна, то такое уравнение может не иметь очевидного аналитического решения. В таких случаях необходимо использовать приближенные численные методы, к которым и относится метод Эйлера.

Прежде всего необходимо заменить производную каким-либо приближенным соотношением. Воспользуемся определением производной:

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t}.$$

К сожалению, компьютеры не могут непосредственно работать с бесконечно малыми величинами. Поэтому перейдем к конечным величинам:

$$\frac{dx}{dt} \approx \frac{\Delta x}{\Delta t}.$$

Такая простая формула будет тем точнее, чем меньше Δt. В любом случае в решение вносится погрешность, которая будет оказывать более или менее существенное влияние на соответствие результата точному решению. Рассмотрение этого вопроса выходит за рамки данного пособия. В первом приближении можно рекомендовать выбирать Δt в диапазоне $10^{-1} \div 10^{-8}$.

Далее вводится дискретное время:

$$t_{i+1} = t_i + \Delta t;\ i = 0..N,$$

где Δt - шаг интегрирования по времени; $N = T/\Delta t$ - номер конечного шага интегрирования. Функцию F(x,t) можно вычислять как в точке (x_i,t_i), так и в точке (x_{i+1},t_{i+1}) - это не влияет на точность метода, хотя может влиять на характер роста погрешности. Первый вариант соответствует явному методу Эйлера, второй - неявному.

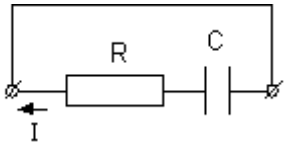
Получаем выражение

$$\frac{\Delta x}{\Delta t} = \frac{x_{i+1} - x_i}{\Delta t} = F(x_i, t_i),$$

из которого следует *расчетная формула* явного метода Эйлера

$$x_{i+1} = x_i + \Delta t \cdot F(x_i, t_i).$$

Рассчитаем переходный процесс в электрической цепи:



Учитывая, что здесь сумма напряжений на резисторе и конденсаторе равна нулю, а ток, протекающий через резистор, равен току через конденсатор, то есть $I = C\frac{dU_c}{dt}$, можно вывести дифференциальное уравнение

$$\frac{dU_c}{dt} = -\frac{1}{RC}U_c.$$

Пусть R = 1, C = 1 с начальным условием $U_c(0) = 1$. Введем дискретное время:

$$t_0 = 0;$$

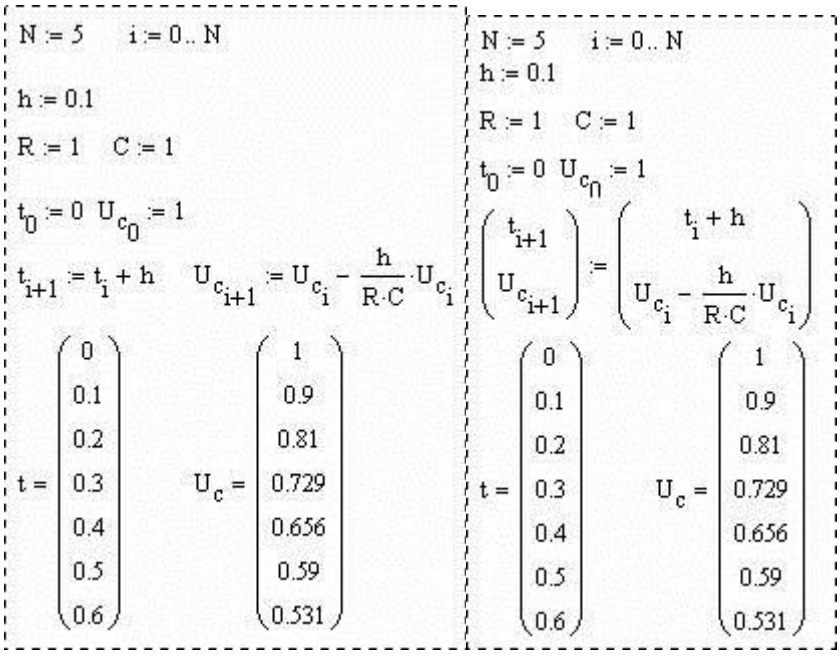
$$t_{i+1} = t_i + h;\ i = 0..N,$$

где h - шаг интегрирования по времени; N - номер конечного шага интегрирования. Воспользуемся расчетной формулой явного метода Эйлера

$$\frac{(U_c)_{i+1} - (U_c)_i}{h} = -\frac{1}{RC}(U_c)_i; (U_c)_0 = 1;$$

$$(U_c)_{i+1} = (U_c)_i - \frac{h}{RC}(U_c)_i.$$

Предположим, что нас интересуют пять итераций с шагом по времени 0,1. Решение можно записать в двух формах (рис. 22).



а) б)

Рис. 22. Две формы записи решения в MATHCAD

Форма (а) более проста, но не позволяет записывать решения систем дифференциальных уравнений, правые части которых являются *взаимозависимыми*. Вторая форма (б) более универсальна.

Полученный вектор U_c содержит значения $U_c(t_i)$ для заданных дискретных моментов времени. Можно легко убедиться, что полученное численное решение похоже на точное аналитическое $U_C(t) = e^{-\frac{1}{RC}t}$. Добавим следующие выражения:

$$R_i := e^{\frac{-1}{R \cdot C} \cdot t_i}$$

$$R = \begin{pmatrix} 1 \\ 0.905 \\ 0.819 \\ 0.741 \\ 0.67 \\ 0.607 \end{pmatrix}$$

и сравним полученные значения R (точное решение) и U_C. В MATHCAD существуют специальные стандартные средства интегрирования дифференциальных уравнений (функция rkfixed, блок Given-Odesolve), которые будут рассмотрены далее.

Контрольные вопросы:

1. Что такое блок Given-Find?
2. Как вычесляются элементы матриц?
3. Вычисления в MathCAD. Доступ к встроенным функциям. Решения систем алгебраических уравнений?
4. Расскажите о способе решения уравнений с одной переменной в программе MATH CAD?
5. Расскажите о способе решения систем уравнений в программе MATH CAD?
6. Расскажите о способе решения диф. уравнений?

Графические возможности программы MATLAB

1. Работа с графиками в MATLAB
2. Функция plot
3. Оформление графиков

1. MATLAB предоставляет богатый инструментарий по визуализации данных. Используя внутренний язык, можно выводить двумерные и трехмерные графики в декартовых и полярных координатах, выполнять отображение изображений с разной глубиной цвета и разными цветовыми картами, создавать простую анимацию результатов моделирования в процессе вычислений и многое другое.

Рассмотрение возможностей MATLAB по визуализации данных начнем с двумерных графиков, которые обычно строятся с помощью функции plot(). Множество вариантов работы данной функции лучше всего рассмотреть на конкретных примерах.

Предположим, что требуется вывести график функции синуса в диапазоне от 0 до π. Для этого зададим вектор (множество) точек по оси Ox, в которых будут отображаться значения функции синуса:

```
x = 0:0.01:pi;
```

В результате получится вектор столбец со множеством значений от 0 до π и с шагом 0,01. Затем, вычислим множество значений функции синуса в этих точках:

```
y = sin(x);
```

и выведем результат на экран

```
plot(x,y);
```

В результате получим график, представленный на рис. 23.

Представленная запись функции plot() показывает, что сначала записывается аргумент со множеством точек оси Ox, а затем, аргумент со множеством точек оси Oy. Зная эти значения, функция plot() имеет возможность построить точки на плоскости и линейно их интерполировать для придания непрерывного вида графика.

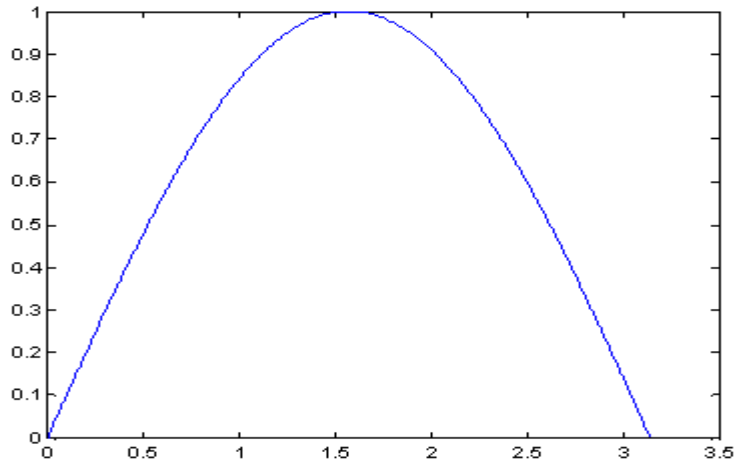


Рис. 23. Отображение функции синуса с помощью функции plot().

Функцию plot() можно записать и с одним аргументом x или y:

```
plot(x);
plot(y);
```

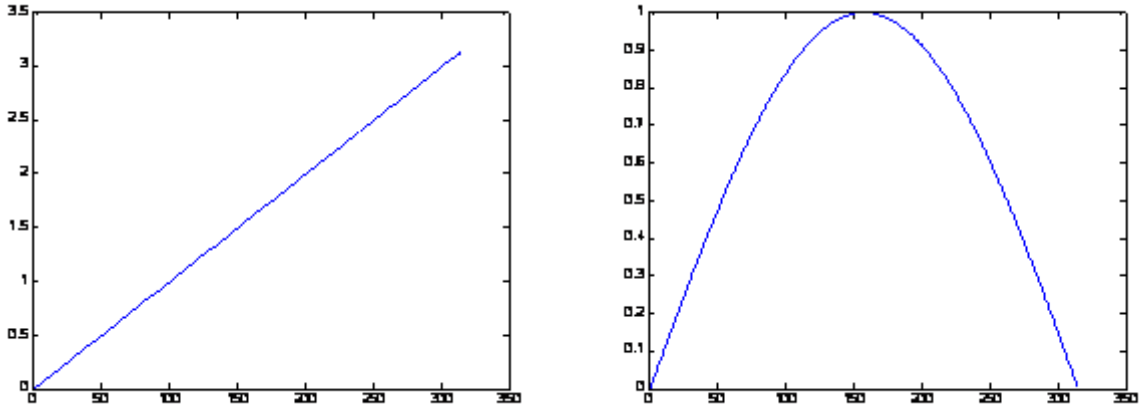
в результате получим два разных графика, представленные на рис. 16.

Анализ рис. 23 показывает, что в случае одного аргумента функция plot() отображает множество точек по оси Oy, а по оси Ox происходит автоматическая генерация множества точек с единичным шагом. Следовательно, для простой визуализации вектора в виде двумерного графика достаточно воспользоваться функцией plot() с одним аргументом.

Для построения нескольких графиков в одних и тех же координатных осях, функция plot() записывается следующим образом:

```
x = 0:0.01:pi;
y1 = sin(x);
y2 = cos(x);
plot(x,y1,x,y2);
```

Результат работы данного фрагмента программы представлен на рис. 24.



а) б)

Рис. 24. Результаты работы функции plot() с одним аргументом: а – plot(x); б – plot(y).

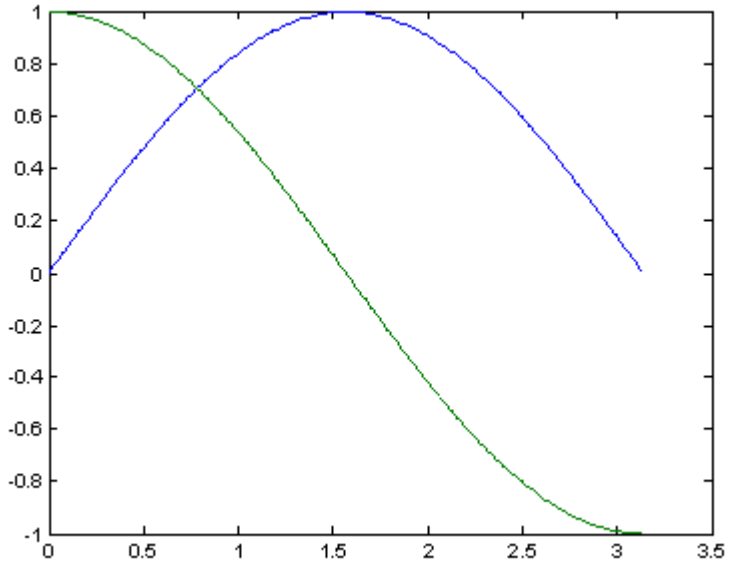


Рис. 25. Отображение двух графиков в одних координатных осях.

Аналогичным образом можно построить два графика, используя один аргумент функции plot(). Предположим, что есть два вектора значений

y1 = sin(x);

y2 = cos(x);

которые требуется отобразить на экране. Для этого объединим их в двумерную матрицу

$$[y1'\,y2'] = \begin{vmatrix} y1_1 & y2_1 \\ y1_2 & y2_2 \\ \dots & \dots \\ y1_N & y2_N \end{vmatrix},$$

в которой столбцы составлены из векторов y1 и y2 соответственно. Такая матрица будет отображена функцией

plot([y1’ y2’]); - апострофы переводят вектор-строку- в вектор-столбец в виде двух графиков (рис. 26).

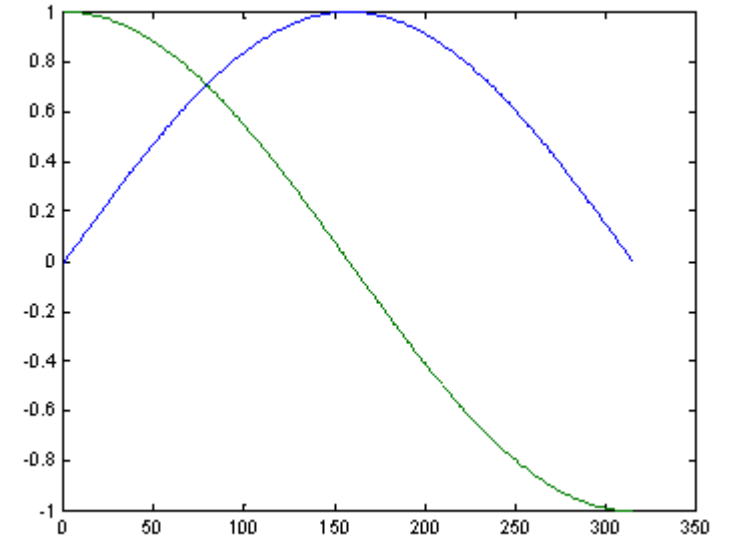


Рис. 26. Отображение двумерной матрицы в виде двух графиков.

2. Два вектора в одних осях можно отобразить только в том случае, если их размерности совпадают. Когда же выполняется работа с векторами разных размерностей, то они либо должны быть приведены друг к другу по числу элементов, либо отображены на разных графиках. Отобразить графики в разных координатных осях можно несколькими способами. В самом простом случае можно создать два графических окна и в них отобразить нужные графики. Это делается следующим образом:

```
x1 = 0:0.01:2*pi;
y1 = sin(x1);

x2 = 0:0.01:pi;
y2 = cos(x2);

plot(x1, y1);       - рисование первого графика
figure;             - создание 2-го графического окна
plot(x2, y2);       - рисование 2-го графика во 2-м окне
```

Функция figure, используемая в данной программе, создает новое графическое окно и делает его активным. Функция plot(), вызываемая сразу после функции figure, отобразит график в текущем активном графическом окне. В результате на экране будут показаны два окна с двумя графиками.

Неудобство работы приведенного фрагмента программы заключается в том, что повторный вызов функции figure отобразит на экране еще одно новое окно и если программа будет выполнена дважды, то на экране окажется три графических окна, но только в двух из них будут актуальные данные. В этом случае было бы лучше построить программу так, чтобы на экране всегда отображалось два окна с нужными графиками. Этого можно достичь, если при вызове функции figure в качестве аргумента указывать номер графического окна, которое необходимо создать или сделать активным, если оно уже создано. Таким образом, вышеприведенную программу можно записать так.

```
x1 = 0:0.01:2*pi;
y1 = sin(x1);

x2 = 0:0.01:pi;
y2 = cos(x2);

figure(1);          -создание окна с номером 1
plot(x1, y1);       - рисование первого графика
figure(2);          - создание графического окна с номером 2
plot(x2, y2);       - рисование 2-го графика во 2-м окне
```

При выполнении данной программы на экране всегда будут отображены только два графических окна с номерами 1 и 2, и в них показаны графики функций синуса и косинуса соответственно.

В некоторых случаях большего удобства представления информации можно достичь, отображая два графика в одном графическом окне. Это достигается путем использования функции subplot(), имеющая следующий синтаксис:

subplot(<число строк>, <число столбцов>, <номер координатной оси>)

Рассмотрим пример отображения двух графиков друг под другом вышеприведенных функций синуса и косинуса.

```
x1 = 0:0.01:2*pi;
y1 = sin(x1);

x2 = 0:0.01:pi;
```

y2 = cos(x2);

figure(1);
subplot(2,1,1); - делим окно на 2 строки и один столбец
plot(x1,y1); - отображение первого графика
subplot(2,1,2); - строим 2-ю координатную ось
plot(x2,y2); - отображаем 2-й график в новых осях

Результат работы программы показан на рис. 27.

Аналогичным образом можно выводить два и более графиков в столбец, в виде таблицы и т.п. Кроме того, можно указывать точные координаты расположения графика в графическом окне. Для этого используется параметр position в функции subplot():
subplot('position', [left bottom width height]);
где left – смещение от левой стороны окна; bottom – смещение от нижней стороны окна; width, height – ширина и высота графика в окне. Все эти переменные изменяются в пределах от 0 до 1.

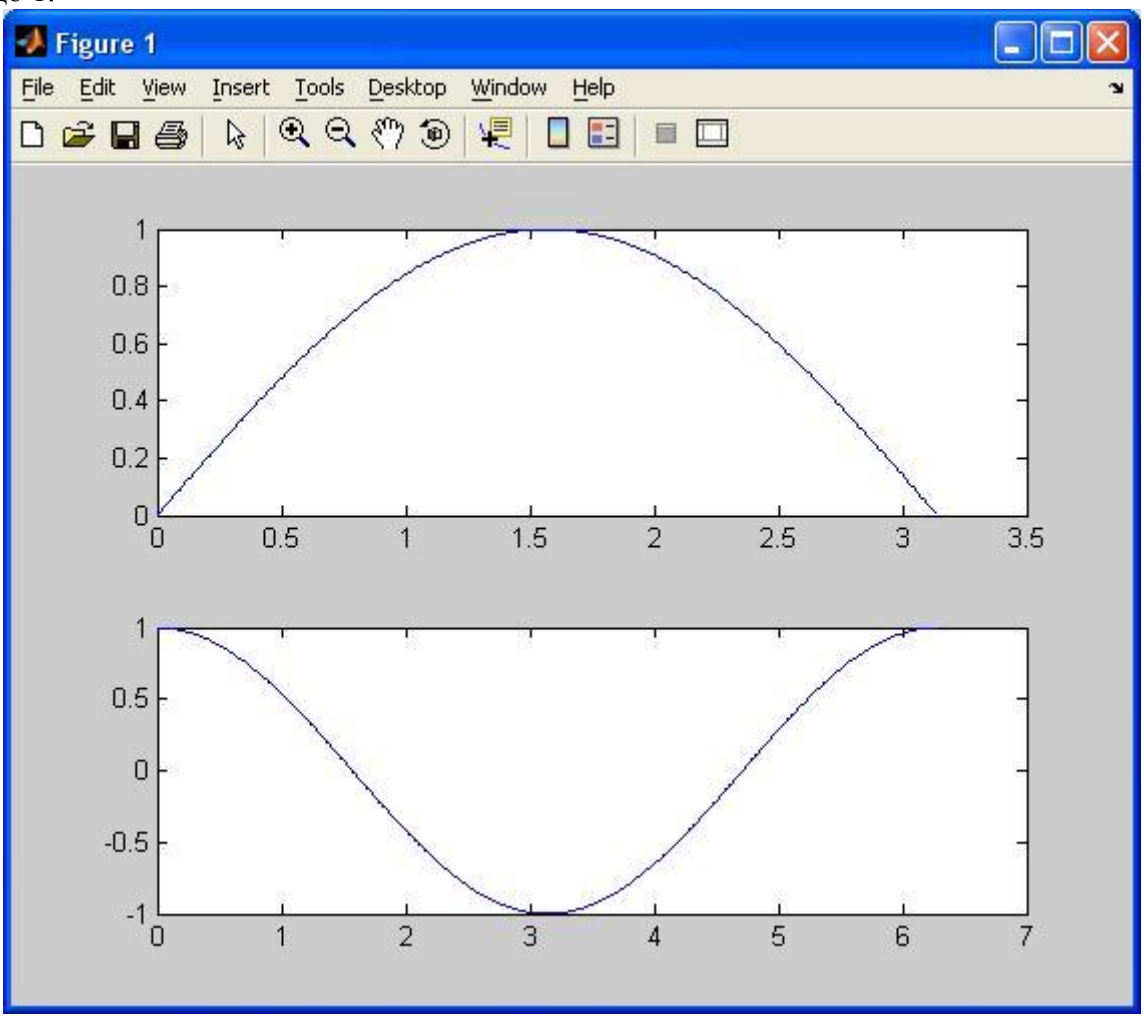


Рис. 27. Пример работы функции subplot.

Ниже представлен фрагмент программы отображения графика функции синуса в центре графического окна. Результат работы показан на рис. 28.
x1 = 0:0.01:2*pi;
y1 = sin(x1);

subplot(‘position’, [0.33 0.33 0.33 0.33]);
plot(x1,y1);

В данном примере функция subplot() смещает график на треть от левой и нижней границ окна и рисует график с шириной и высотой в треть графического окна. В результате, получается эффект рисования функции синуса по центру основного окна.

Таким образом, используя параметр position можно произвольно размещать графические элементы в плоскости окна.

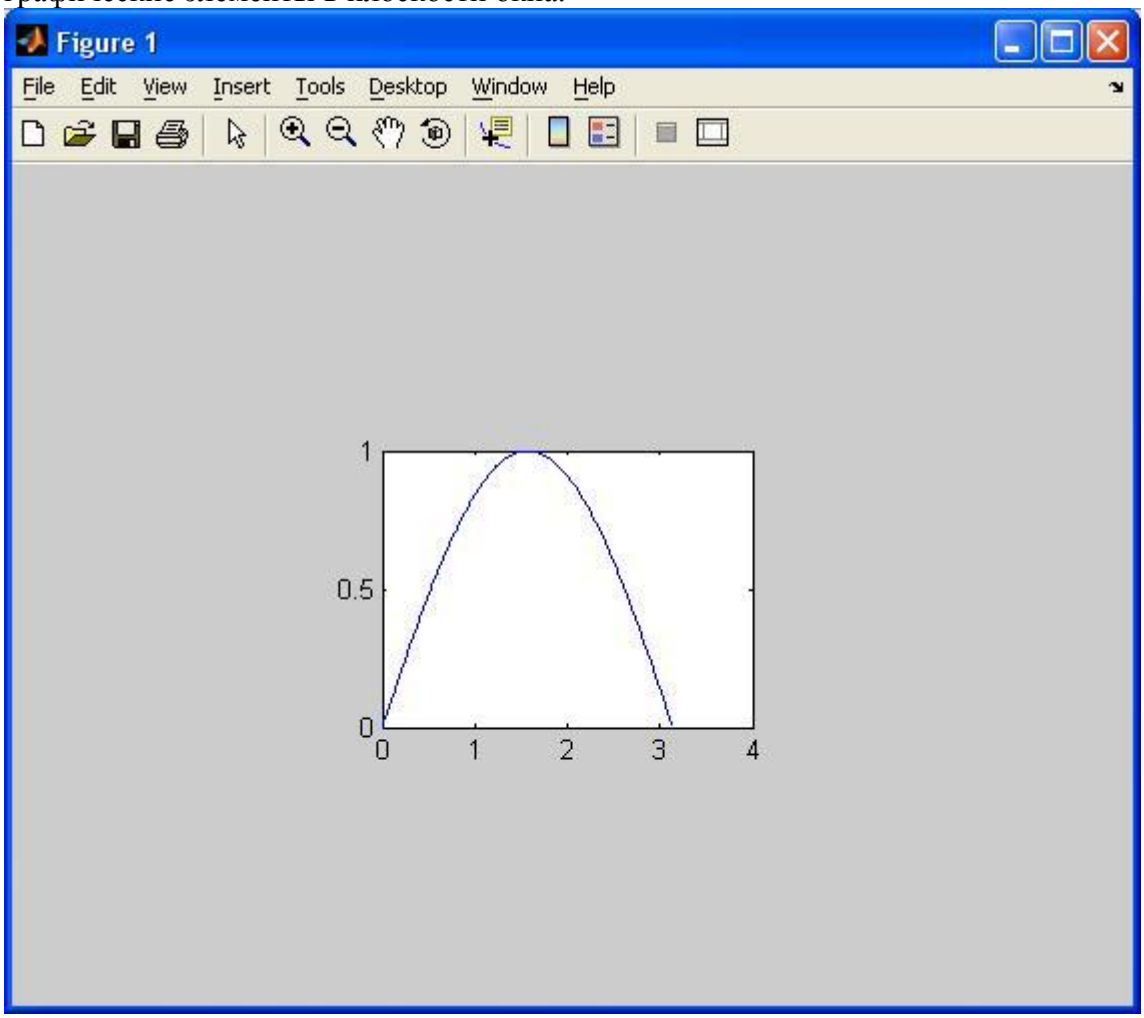


Рис. 28. Пример работы функции subplot с параметром position.

3. Пакет MATLAB позволяет отображать графики с разным цветом и типом линий, показывать или скрывать сетку на графике, выполнять подпись осей и графика в целом, создавать легенду и многое другое. В данном параграфе рассмотрим наиболее важные функции, позволяющие делать такие оформления на примере двумерных графиков.

Функция plot() позволяет менять цвет и тип отображаемой линии. Для этого, используются дополнительные параметры, которые записываются следующим образом:
plot(<x>, <y>, <’цвет линии, тип линии, маркер точек’>);

Обратите внимание, что третий параметр записывается в апострофах и имеет обозначения, приведенные в таблицах 5-7. Маркеры, указанные ниже записываются подряд друг за другом, например,
‘ko’ – на графике отображает черными кружками точки графика,
‘ko-‘ – рисует график черной линией и проставляет точки в виде кружков.

Табл. 5. Обозначение цвета линии графика

Маркер	Цвет линии
c	голубой
m	фиолетовый
y	желтый
r	красный
g	зеленый
b	синий
w	белый
k	черный

Табл. 6. Обозначение типа линии графика

Маркер	Цвет линии
-	непрерывная
--	штриховая
:	пунктирная
-.	штрих-пунктирная

Табл. 7. Обозначение типа точек графика

Маркер	Цвет линии
.	точка
+	плюс
*	звездочка
o	кружок
x	крестик

Ниже показаны примеры записи функции plot() с разным набором маркеров.

```
x = 0:0.1:2*pi;
y = sin(x);

subplot(2,2,1); plot(x,y,'r-');
subplot(2,2,2); plot(x,y,'r-',x,y,'ko');
subplot(2,2,3); plot(y,'b--');
subplot(2,2,4); plot(y,'b--+');
```

Результат работы фрагмента программы приведен на рис. 21. Представленный пример показывает, каким образом можно комбинировать маркеры для достижения требуемого результата. А на рис. 21 наглядно видно к каким визуальным эффектам приводят разные маркеры, используемые в программе. Следует особо отметить, что в четвертой строчке программы по сути отображаются два графика: первый рисуется красным цветом и непрерывной линией, а второй черными кружками заданных точек графика. Остальные варианты записи маркеров очевидны.

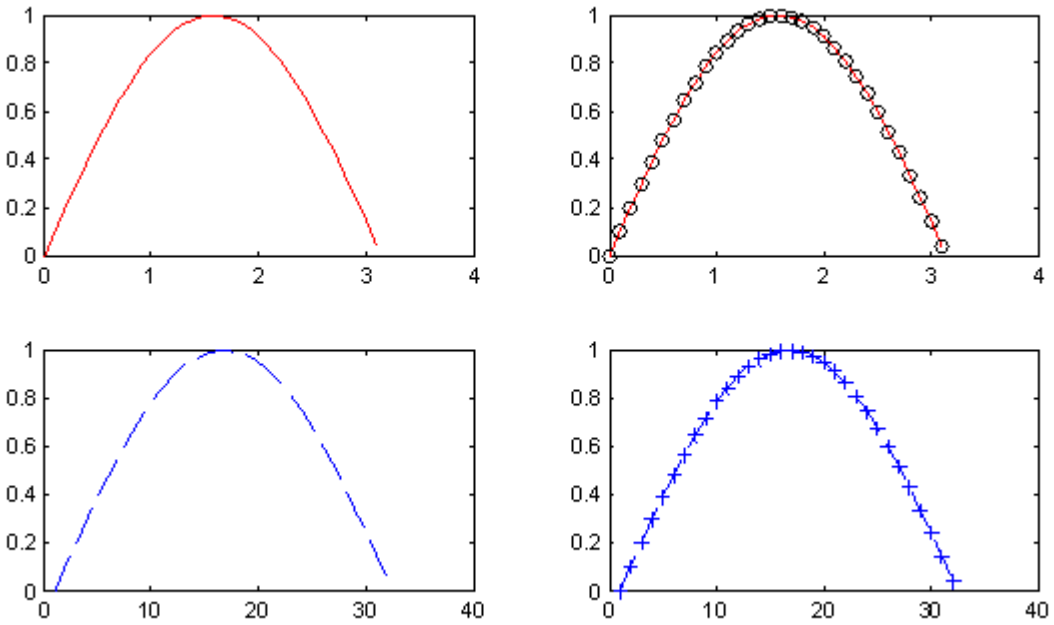


Рис.29. Примеры отображения графиков с разными типами маркеров

Из примеров рис. 29 видно, что масштаб графиков по оси Ox несколько больше реальных значений. Дело в том, что система MATLAB автоматически масштабирует систему координат для полного представления данных. Однако такая автоматическая настройка не всегда может удовлетворять интересам пользователя. Иногда требуется выделить отдельный фрагмент графика и только его показать целиком. Для этого используется функция axis() языка MATLAB, которая имеет следующий синтаксис:

axis([xmin, xmax, ymin, ymax]),

где название указанных параметров говорят сами за себя.

Воспользуемся данной функцией для отображения графика функции синуса в пределах от 0 до 2π:

```
x = 0:0.1:2*pi;
y = sin(x);

subplot(1,2,1);
plot(x,y);
axis([0 2*pi -1 1]);

subplot(1,2,2);
plot(x,y);
axis([0 pi 0 1]);
```

Из результата работы программы (рис. 30) видно, что несмотря на то, что функция синуса задана в диапазоне от 0 до 2π, с помощью функции axis() можно отобразить как весь график, так и его фрагмент в пределах от 0 до π.

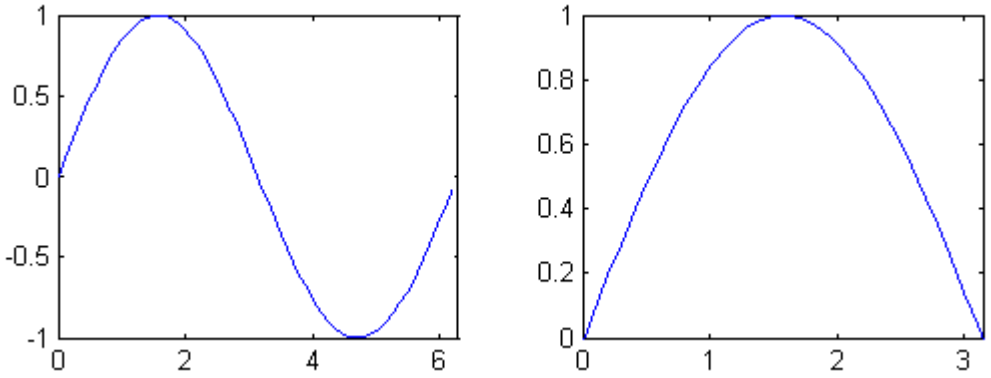


Рис. 30. Пример работы функции axis()

А также рассмотрим возможности создания подписей графиков, осей и отображения сетки на графике. Для этого используются функции языка MATLAB, перечисленные в табл. 8.

Таблица 8. Функции оформления графиков

Название	Описание
grid [on, off]	Включает/выключает сетку на графике
title(‘заголовок графика’)	Создает надпись заголовка графика
xlabel(‘подпись оси Ox’)	Создает подпись оси Ox
ylabel(‘подпись оси Oy’)	Создает подпись оси Oy
text(x,y,’текст’)	Создает текстовую надпись в координатах (x,y).

Рассмотрим работу данных функций в следующем примере:

```
x = 0:0.1:2*pi;
y = sin(x);

plot(x,y);
axis([0 2*pi -1 1]);
grid on;
title('The graphic of sin(x) function');
xlabel('The coordinate of Ox');
ylabel('The coordinate of Oy');
text(3.05,0.16,'\leftarrow sin(x)');
```

Из результата работы данной программы, представленного на рис. 31, видно каким образом работают функции создания подписей на графике, а также отображение сетки графика.

Таким образом, используя описанный набор функций и параметров, можно достичь желаемого способа оформления графиков в системе MATLAB.

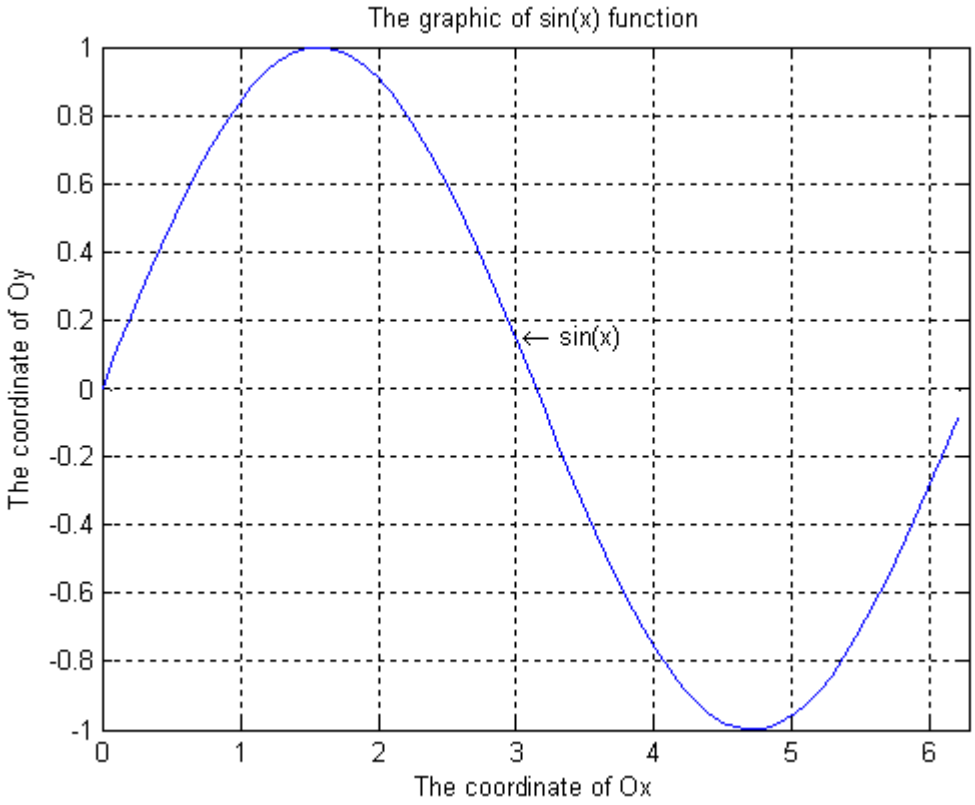


Рис. 31. Пример работы функций оформления графика

Контрольные вопросы:

1. Расскажите о графических возможностях в программе MATLAB?
2. Для чего нужна функция plot?
3. Как можно оформлять графики?
4. Как можно отобразить два вектора на одних осях?

Работа с файлами в MATLAB

1. Функции save и load
2. Функции fwrite и fread
3. Функции fscanf и fprintf
4. Функции imread и imwrite

1. Создание программ часто предполагает сохранение результатов расчетов в файлы для их дальнейшего анализа, обработки, хранения и т.п. В связи с этим в MATLAB реализованы различные функции по работе с файлами, содержащие данные в самых разных форматах. В этой главе рассмотрим наиболее полезные функции для сохранения и загрузки результатов работы алгоритмов из файлов.

В самом простом случае для сохранения и последующей загрузки каких-либо данных в MATLAB предусмотрены две функции

save <имя файла> <имена переменных> - сохранение данных
load <имя файла> <имена переменных> - загрузка данных

Функция save позволяет сохранять произвольные переменные программы в файл, который будет (по умолчанию) располагаться в рабочем каталоге (обычно поддиректория work) и иметь расширение mat. Соответственно функция load позволяет загрузить из указанного mat-файла ранее сохраненные переменные. Ниже представлен пример использования данных функций:

```
function save_load
x = ones(5);
y = 5;
s = 'hello';

save params x y s;
x = zeros(5);
y = 0;
s = '';

load params x y s;
disp(x);
disp(y);
disp(s);
```

В данной программе сначала выполняется инициализация переменных x, y, s, затем, они сохраняются в файл params.mat, заменяются другими значениями и после загрузки отображаются на экране. При выполнении этой программы на экране будут показаны те же значения переменных, которые они принимали в самом начале. Таким образом демонстрируется работа функций save и load.

Следует обратить внимание, что функция load позволяет загружать из mat-файла не все, а только указанные программистом переменные, например

load params x; - загружает только значение переменной x
load params y; - загружает только значение переменной y
load params x s; - загружает значения переменных x и s

2. Недостатком рассмотренных функций save и load является то, что они работают с определенными форматами файлов (обычно mat-файлы) и не позволяют загружать или сохранять данные в других форматах. Между тем бывает необходимость загружать информацию, например, из бинарных файлов, созданных другими программными продуктами для дальнейшей обработки результатов в MATLAB. С этой целью были разработаны функции

fwrite(<идентификатор файла>, <переменная>, <тип данных>);

и

<переменная>=fread(<идентификатор файла>);
<переменная>=fread(<идентификатор файла>, <размер>);
<переменная>=fread(<идентификатор файла>, <размер>, <точность>);

Здесь <идентификатор файла> - это указатель на файл, с которым предполагается работать. Чтобы получить идентификатор, используется функция

<идентификатор файла> = fopen(<имя файла>,<режим работы>);

где параметр <режим работы> может принимать значения, приведенные в табл. 9.

Таблица 9. Режимы работы с файлами в MATLAB

параметр <режим работы>	описание
'r'	чтение
'w'	запись (стирает предыдущее содержимое файла)

'a'	добавление (создает файл, если его нет)
'r+'	чтение и запись (не создает файл, если его нет)
'w+'	чтение и запись (очищает прежнее содержимое или создает файл, если его нет)
'a+'	чтение и добавление (создает файл, если его нет)
'b'	дополнительный параметр, означающий работу с бинарными файлами, например, ‘wb’, ‘rb’ ‘rb+’, ‘ab’ и т.п.

Если функция fopen() по каким-либо причинам не может корректно открыть файл, то она возвращает значение -1. Ниже представлен фрагмент программы записи и считывания данных из бинарного файла:

```
A = [1 2 3 4 5];

fid = fopen('my_file.dat', 'wb');     - открытие файла на запись
if fid == -1                          - проверка корректности открытия
   error('File is not opened');
end

fwrite(fid, A, 'double');             - запись матрицы в файл (40 байт)
fclose(fid);                          - закрытие файла

fid = fopen('my_file.dat', 'rb');     - открытие файла на чтение
if fid == -1                          - проверка корректности открытия
   error('File is not opened');
end

B = fread(fid, 5, 'double');          - чтение 5 значений double
disp(B);                              - отображение на экране
fclose(fid);                          - закрытие файла
```

В результате работы данной программы в рабочем каталоге будет создан файл my_file.dat размером 40 байт, в котором будут содержаться 5 значений типа double, записанных в виде последовательности байт (по 8 байт на каждое значение). Функция fread() считывает последовательно сохраненные байты и автоматически преобразовывает их к типу double, т.е. каждые 8 байт интерпретируются как одно значение типа double.

В приведенном примере явно указывалось число элементов (пять) для считывания из файла. Однако часто общее количество элементов бывает наперед неизвестным, либо оно меняется в процессе работы программы. В этом случае было бы лучше считывать данные из файла до тех пор, пока не будет достигнут его конец. В MATLAB существует функция для проверки достижения конца файла

feof(<идентификатор файла>)

которая возвращает 1 при достижении конца файла и 0 в других случаях. Перепишем программу для считывания произвольного числа элементов типа double из входного файла.

```
fid = fopen('my_file.dat', 'rb');  - открытие файла на чтение
if fid == -1
   error('File is not opened');
end

B=0;                               - инициализация переменной
cnt=1;                             - инициализация счетчика
while ~feof(fid)                   - цикл, пока не достигнут конец файла
```

```
   [V,N] = fread(fid, 1, 'double');  -считывание одного значения double (V содержит значение элемента, N – число считанных элементов)
   if N > 0          - если элемент был прочитан успешно, то
      B(cnt)=V;   - формируем вектор-строку из значений V
      cnt=cnt+1;  - увеличиваем счетчик на 1
   end
end
disp(B);             - отображение результата на экран
fclose(fid);         - закрытие файла
```

В данной программе динамически формируется вектор-строка по мере считывания элементов из входного файла. MATLAB автоматически увеличивает размерность векторов, если индекс следующего элемента на 1 больше максимального. Однако на такую процедуру тратится много машинного времени и программа начинает работать заметно медленнее, чем если бы размерность вектора B с самого начала была определена равным 5 элементам, например, так

```
B = zeros(5,1);
```

Следует также отметить, что функция fread() записана с двумя выходными параметрами V и N. Первый параметр содержит значение считанного элемента, а второй – число считанных элементов. В данном случае значение N будет равно 1 каждый раз при корректном считывании информации из файла, и 0 при считывании служебного символа EOF, означающий конец файла. Приведенная ниже проверка позволяет корректно сформировать вектор значений B.

С помощью функций fwrite() и fread() можно сохранять и строковые данные. Например, пусть дана строка

```
str = 'Hello MATLAB';
```

которую требуется сохранить в файл. В этом случае функция fwrite() будет иметь следующую запись:

```
fwrite(fid, str, 'int16');
```

Здесь используется тип int16, т.к. при работе с русскими буквами система MATLAB использует двухбайтовое представление каждого символа. Ниже представлена программа записи и чтения строковых данных, используя функции fwrite() и fread():

```
fid = fopen('my_file.dat', 'wb');
if fid == -1
   error('File is not opened');
end

str='Привет MATLAB';                  - строка для записи
fwrite(fid, str, 'int16');            - запись в файл
fclose(fid);

fid = fopen('my_file.dat', 'rb');
if fid == -1
   error('File is not opened');
end

B='';                                  - инициализация строки
cnt=1;
while ~feof(fid)
   [V,N] = fread(fid, 1, 'int16=>char');  - чтение текущего символа и преобразование его в тип
```

```
char
   if N > 0
      B(cnt)=V;
      cnt=cnt+1;
   end
end
disp(B);                                  - отображение строки на экране
fclose(fid);
```

Результат выполнения программы будет иметь вид Привет MATLAB

3. Описанные выше функции работы с файлами позволяют записывать и считывать информацию по байтам, которые затем требуется правильно интерпретировать для преобразования их в числа или строки. В то же время выходными результатами многих программ являются текстовые файлы, в которых явным образом записаны те или числа или текст. Например, при экспорте данных из MS Excel можно получить файл формата

174500,1.63820,1.63840,1.63660,1.63750,288
180000,1.63740,1.63950,1.63660,1.63820,361
181500,1.63830,1.63850,1.63680,1.63740,223
183000,1.63720,1.64030,1.63720,1.64020,220

где числа записаны в столбик и разделены запятой.

Прочитать такой файл побайтно, а затем интерпретировать полученные данные довольно трудоемкая задача, поэтому для этих целей были специально разработаны функции чтения

[value, count] = fscanf(fid, format, size)

и записи

count = fprintf(fid, format, a,b,...)

таких данных в файл. Здесь value – результат считывания данных из файла; count – число прочитанных (записанных) данных; fid – указатель на файл; format – формат чтения (записи) данных; size – максимальное число считываемых данных; a,b,.. – переменные для записи в файл.

Приведем пример чтения данных из файла, приведенного выше с помощью функции fscanf():

```
function fscanf_ex

fid = fopen('my_excel.dat', 'r');
if fid == -1
   error('File is not opened');
end

S = fscanf(fid, '-d,-f,-f,-f,-f,-d');
fclose(fid);
```

Здесь форматная строка состоит из спецификаторов

-d – работа с целочисленными значениями;

-f – работа с вещественными значениями

и записана в виде '-d,-f,-f,-f,-f,-d'. Это означает, что сначала должно быть прочитано целочисленное значение из файла, затем, через запятую должно читаться второе вещественное значение, затем третье и так далее до последнего целочисленного значения. Полный список возможных спецификаторов приведен в табл. 9.

В результате работы программы переменная S будет представлять собой вектор-столбец, состоящий из 24 элементов:

S = [174500 1,6382 1,6384 1,6366 1,6375 288 180000 1,6374 1,6395 1,6366 1,6382 361 181500 1,6383 1,6385 1,6368 1,6374 223 183000 1,6372 1,6403 1,6372 1,6402 220]';

Несмотря на то, что данные были корректно считаны из файла, они из таблицы были преобразованы в вектор-столбец, что не соответствует исходному формату представления данных. Чтобы сохранить верный формат данных, функцию fscanf() в приведенном примере следует записать так:

S = fscanf(fid, '-d,-f,-f,-f,-f,-d', [6 4]);

Тогда на выходе получится матрица S размером в 6 столбцов и 4 строки с соответствующими числовыми значениями.

Таблица 10. Список основных спецификаторов для функций fscanf() и fprintf()

Спецификатор	Описание
-d	целочисленные значения
-f	вещественные значения
-s	строковые данные
-c	символьные данные
-u	беззнаковые целые значения

Для записи данных в текстовый файл в заданном формате используется функция fprintf(). Ниже представлен пример записи матрицы чисел

180000 1.28210 1.28240 1.28100 1.28120 490
190000 1.28100 1.28150 1.27980 1.28070 444
200000 1.28050 1.28080 1.27980 1.28000 399
210000 1.27990 1.28020 1.27880 1.27970 408
220000 1.27980 1.28060 1.27880 1.28030 437
230000 1.28040 1.28170 1.28020 1.28130 419
000000 1.28140 1.28140 1.28010 1.28100 294
010000 1.28080 1.28190 1.28030 1.28180 458
020000 1.28190 1.28210 1.28080 1.28140 384
030000 1.28130 1.28170 1.28080 1.28140 313

в файл, в котором числовые значения должны разделяться точкой с запятой. Будем также предполагать, что данная матрица хранится в переменной Y.

```
function fprintf_ex

fid = fopen('my_excel.txt', 'w');
if fid == -1
   error('File is not opened');
end

fprintf(fid, '-6d;-.4f;-.4f;-.4f;-.4f;-d\r\n', Y');
fclose(fid);
```

Следует отметить, что в функции fprintf() переменная Y имеет знак транспонирования ‘, т.к. данные в файл записываются по столбцам матрицы. Кроме того, перед спецификаторами стоят числа, которые указывают сколько значащих цифр числа должно быть записано в файл. Например, спецификатор -6d говорит о том, что целые числа должны иметь 6 значащих цифр, а спецификатор -.4f означает, что после запятой будет отображено только 4 цифры. Наконец, в форматной строке были использованы управляющие символы

\r – возврат каретки;

\n – переход на новую строку

которые необходимы для формирования строк в выходном файле. В итоге, содержимое файла будет иметь вид:

180000;1.2821;1.2824;1.2810;1.2812;490
190000;1.2810;1.2815;1.2798;1.2807;444
200000;1.2805;1.2808;1.2798;1.2800;399
210000;1.2799;1.2802;1.2788;1.2797;408
220000;1.2798;1.2806;1.2788;1.2803;437
230000;1.2804;1.2817;1.2802;1.2813;419
0;1.2814;1.2814;1.2801;1.2810;294
10000;1.2808;1.2819;1.2803;1.2818;458
20000;1.2819;1.2821;1.2808;1.2814;384
30000;1.2813;1.2817;1.2808;1.2814;313

С помощью функции fprintf() можно записать значения двух и более переменных разного формата. Например, для записи числа и строки можно воспользоваться следующей записью:

```
str = 'Hello';
y = 10;
count = fprintf(fid, '-d\r\n-s\r\n', y, str);
```

и содержимое файла будет иметь вид:

```
10
Hello
```

Таким образом можно осуществлять запись разнородных данных в файл в требуемом формате.

4. При работе с файлами изображений, представленных в форматах bmp, png, gif, jpeg, tif и т.д., используются функции чтения

```
[X, map] = imread(filename, fmt)
```

и записи

```
imwrite(X, map, filename, fmt)
```

Здесь X – матрица точек изображения; map – цветовая карта изображения; filename – путь к файлу; fmt – графический формат файла изображения.

Работа функции imread() была подробно рассмотрена в п. 3.4. Ниже приведен пример ее использования для загрузки растрового изображения

```
[A, map]=imread('1024.bmp','bmp');
```

где A – матрица размером 1024x1024xN точек; map – цветовая карта загруженного изображения. Значение N показывает число байт, расходуемых на представление точки изображения. Например, если изображение представляется в формате RGB с 24 бит/пиксел, то N=3. Если же загружается изображение с 256 градациями серого (8 бит/пиксел), то N=1.

После обработки изображение A можно обратно сохранить в файл, используя следующую запись:

```
imwrite(A, map, 'out_img.bmp', 'bmp');
```

В результате в рабочем каталоге MATLAB будет сохранено изображение в формате bmp с исходной цветовой картой. Однако следует отметить, что если загруженное изображение A было преобразовано, например, в формат double

```
A = double(A);
```

то непосредственная запись такой матрицы как изображение невозможно. Дело в том, что значения матрицы A должны соответствовать целым числам в диапазоне от 0 до 255, т.е. являться байтовыми числами. Этого можно добиться преобразованием типов при записи изображения в файл следующим образом:

```
imwrite(uint8(A), map, 'out_img.bmp', 'bmp');
```

Здесь uint8 – беззнаковый целый тип в 8 бит.

В качестве переменной map можно указывать любые другие цветовые карты (hot, hsv, gray, pink, cool, bone copper) отличные от исходной. Например, для записи изображения в 256 градациях серого можно записать
imwrite(uint8(A), gray(256), 'out_img.bmp', 'bmp');
При этом матрица A должна иметь размерность MxNx1, т.е. один байт на пиксел.

Контрольные вопросы:
1. Для чего предназначены функции save и load?
2. Для чего предназначены функции fwrite и fread?
3. Для чего предназначены функции fscanf и fprintf?
4. Для чего предназначены функции imread и imwrite?

Программирование функций в MATLAB

1. Порядок определения и вызова функций
2. Область видимости переменных

1. Часто при программировании приходится много раз повторять одни и те же вычисления, например, определение модуля числа или расчет евклидового расстояния между точками и т.п. Для реализации таких повторяющихся вычислений целесообразно создавать функции и вызывать их по мере необходимости.

Синтаксис для определения собственных функций в MATLAB имеет следующий вид:
function [RetVal1, RetVal2,…] = FunctionName(arg1, arg2,…)
<тело функции>
где RetVal1, RetVal2,... – набор возвращаемых значений функцией (результаты работы); arg1, arg2,... – набор входных аргументов; тело функции – набор операторов (программа), которые выполняются при вызове функции.

Рассмотрим пример реализации функции для вычисления евклидового расстояния:
function length = euqlid(x1, y1, x2, y2)
length = sqrt((x1-x2)^2+(y1-y2)^2);

Продемонстрируем возможность возвращения нескольких параметров на примере вычисления ширины и высоты прямоугольника, заданного координатами левого верхнего угла (x1,y1) и правого нижнего (x2,y2):
function [width, height] = RectangleHW(x1,y1,x2,y2)
width = abs(x1-x2);
height = abs(y1-y2);

Данную функцию можно записать еще и с таким набором параметров:
function [width, height] = RectangleHW(P1, P2)
width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
где P1 и P2 – векторы (массивы) размером в 2 элемента и описывают точку в двумерном пространстве. В этом случае при вызове функции, значения координат точек можно передавать таким образом:
[W, H] = RectangleHW([0 0], [10 20]);

Если же программист сделает ошибку и при вызове функции передаст неверный размер вектора, например, так
[W, H] = RectangleHW(0, [10 20]);

то выполнение функции завершится с ошибкой и выполнение всего алгоритма остановится. Чтобы избежать этой ситуации MATLAB позволяет проводить проверку корректности переданных аргументов и корректно завершать работу функции без остановки работы всего алгоритма. Следующий пример записи функции демонстрирует работу такой проверки:

```
function [width, height] = RectangleHW(P1, P2)
if length(P1) < 2 | length(P2) < 2
    error( 'Bad 1st or 2nd parameter' );
end

width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
```

При выполнении данной функции с неверными параметрами, функция выдаст сообщение об ошибке в командное окно MATLAB, но программа продолжит свою работу.

Предложенная проверка осуществляет контроль за корректностью переданных аргументов. Однако важной является также проверка числа переданных входных аргументов и числа возвращаемых значений функцией. Например, если вместо двух аргументов, был передан только один, то функция ошибочно завершит свою работу. Аналогично, если функция ожидает возврата трех аргументов, в то время как она определена лишь для двух, то также возникнет ошибочная ситуация.

Для проверки числа переданных аргументов и числа ожидающих возвращаемых значений используются переменные nargin и nargout. Ниже приведен пример функции, использующей проверку корректности числа входных и выходных аргументов.

```
function [width, height] = RectangleHW(P1, P2)
if nargin ~= 2
   error( 'Bad number of parameters' );
end
if nargout ~= 2
   error(  'Must be 2 return values' );
end
if length(P1) < 2 | length(P2) < 2
   error( 'Bad 1st or 2nd parameter' );
end

width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
```

При этом проверки корректности параметров функции будут срабатывать в следующих ситуациях:

```
[W, H] = RectangleHW([0 0]);      - Bad number of parameters
[W, H, V] = RectangleHW([0 0], [10 20]); - Must be 2 return
                                - values
[W, H] = RectangleHW(0, [10 20]); - Bad 1st or 2nd parameter
```

2. Следует отметить, что переменные, объявленные внутри функций, имеют область видимости только в пределах функции, и за ее пределами уже не доступны (не видны). Следующий пример программы демонстрирует механизм области видимости имен переменных в MATLAB:

```
function MyFunc
x = 10;
disp(x);
MyFunc2();

function MyFunc2()
disp(x);
```

В результате на экране будет отображено

```
10
??? Undefined function or variable 'x'.
```

Этот пример показывает, что переменная с именем x, объявленная в функции MyFunc, не доступна в функции MyFunc2. Это сделано с расчетом, чтобы переменные в разных функциях не влияли друг на друга даже если они имеют одни и те же имена. Однако в некоторых случаях требуется, чтобы переменная была видна за пределами функции, в которой объявлена. Это достигается путем обращения к переменной как к глобальной с помощью ключевого слова global, за которым следует имя глобальной переменной. Перепишем пример, представленный выше с использованием глобальной переменной:

```
function MyFunc
x = 10;
disp(x);
MyFunc2();

function MyFunc2()
global x;
disp(x);
```

Обратите внимание, что ключевое слово global написано в функции MyFunc2 и говорит о том, что переменная x уже объявлена ранее и нужно ее использовать внутри текущей функции.

Контрольные вопросы:

1. Что такое порядок определения функций?
2. Что такое порядок вызова функций?
3. Что такое область видимости переменных?
4. Программирование функций в программе MATLAB?

Графические возможности программы MATHCAD

1. Двухмерные графики в декартовой системе координат
2. Кривые на плоскости, заданные параметрически.
3. Графики в трехмерном пространстве.

1. MATHCAD позволяет легко строить двух- и трехмерные гистограммы, двухмерные графики в декартовых и полярных координатах, трехмерные графики поверхностей, линии уровня поверхностей, изображения векторных полей, пространственные кривые.

Существует три способа построения графиков в системе MATHCAD:

- можно воспользоваться позицией Главного меню Insert, выбрав команду Graph и в раскрывающемся списке - тип графика;
- выбрать тип графика на наборной панели Graph, которая включается кнопкой на панели Math;

– воспользоваться быстрыми клавишами (они предусмотрены не для всех типов графиков).

Рассмотрим более подробно команды меню Insert->Graph (слева изображены соответствующие кнопки наборной панели Math):

 X-Y Plot (X-Y Зависимость) клавиша [@]

Служит доля построения графика функции y=f(x) в виде связанных друг с другом пар координат (xi,yi) при заданном промежутке изменения для i.

 Polar Plot (Полярные координаты)клавиши [Ctrl+7]

Служит для построения графика функции r(q), заданной в полярных координатах, где полярный радиус r зависит от полярного угла q.

 Surface Plot (Поверхности) клавиши [Ctrl+2]

Служит для представления функции z=f(x,y) в виде поверхности в трехмерном пространстве. При этом должны быть заданы векторы значений xi и yj), а также определена матрица вида Ai,j=f(xi,yj). Имя матрицы A указывается при заполнении рамки-шаблона. С помощью этой команды можно строить параметрические графики.

 Contour Plot (Контурный график)

Строит диаграмму линий уровня функции вида z=f(x,y), т.е. отображает точки, в которых данная функция принимает фиксированное значение z=const.

 3D Scatter Plot (3D Точечный)

Служит для точечного представления матрицы значений Ai,j или отображения значений функции z=f(x,y) в заданных точках. Эта команда может также использоваться для построения пространственных кривых.

 3D Bar Plot (3D Диаграммы)

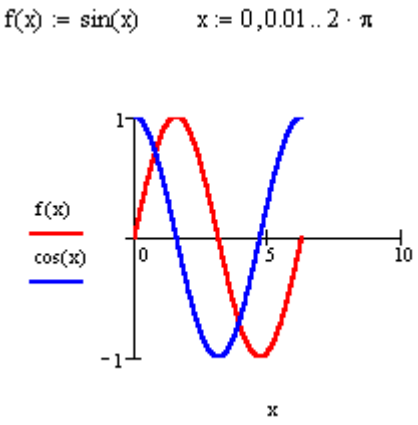
Служит для представления матрицы значений Ai,j или отображения значений функции z=f(x,y) в виде трехмерной столбчатой диаграммы.

 Vector Field Plot (Поле векторов)

Служит для представления двухмерных векторных полей V=(Vx, Vy). При этом компоненты векторного поля Vx и Vy должны быть представлены в виде матриц. При помощи этой команды можно построить поле градиента финкции f(x,y).

3D Plot Wizard (вызов мастера для быстрого построения 3-хмерного графика) При выборе этой команды возникает ряд всплывающих окон, в которых требуется выбрать параметры построения трехмерного графика (задаются тип трехмерного графика, стиль его изображения, цветовая гамма). График по умолчанию строится на промежутке от -5 до +5 (по обеим переменным).

График функции y=f(x).



При выполнении команды Inset -> Graph -> Plot в документ помещается рамка-шаблон с двумя незаполненными ячейками для построения графика. (Клавиша [@]).

В ячейке, расположенной под осью абсцисс, указывается независимая переменная x. Ее следует определить заранее как переменную, принимающую значения из промежутка (ранжированная переменая).

Рис. 32. График функции y=f(x).

В ячейке рядом с осью ординат необходимо задать функцию f(x), график которой мы хотим построить. Если эта функция была определена заранее, то в ячейку достаточно ввести f(x), в противном случае следует ввести изображаемую функцию в явном виде (например, cos(x)).

После ввода x и f(x) в графической области появятся еще четыре ячейки, которые не обязательно заполнять. MATHCAD автоматически находит подходящие значения для xmin, xmax, ymin, ymax. Если же предлагаемые MATHCAD значения вас не устраивают, вы можете задать свои.

В MATHCAD существует возможность строить график функции, не задавая предварительно промежуток изменения независимой переменной. По умолчанию этот промежуток принимается равным [-10, 10].

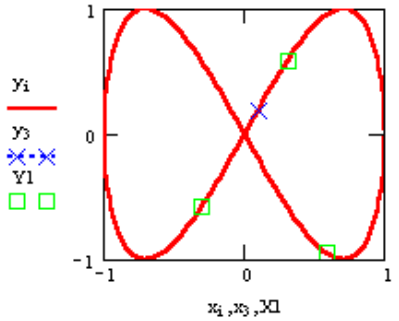
Для представления на одной диаграмме графиков нескольких функций необходимо выделить ячейку рядом с осью ординат и через запятую ввести вторую функцию. По умолчанию график этой функции будет представлен пунктирной линией другого цвета.

2. Уравнения x=f(t), y=y(t) , где f(t) и y(t) непрерывны при t из (a, b), устанавливающие зависимость декартовых координат (x,y) точки плоскости от значения параметра t , определяют на плоскости кривую, заданную в параметрической форме.

$N := 200 \qquad i := 0..N$

$x_i := \sin\left(2 \cdot \pi \cdot \frac{i}{N}\right) \qquad y_i := \sin\left(4 \cdot \pi \cdot \frac{i}{N}\right)$

$X1 := \begin{pmatrix} x_{10} \\ x_{80} \\ x_{190} \end{pmatrix} \qquad Y1 := \begin{pmatrix} y_{10} \\ y_{80} \\ y_{190} \end{pmatrix}$



В случае построения параметрически заданной кривой, вместо независимой переменной x под осью абсцисс необходимо задать индексированую переменную xi. А рядом с осью ординат необходимо соответственно указать yi.

Для нанесения на график функции отдельных точек, их координаты указываются через запятую под осью абсцисс и слева от оси ординат. Если требуется вывести множество точек, то можно сформировать два вектора, один из которых содержит абсциссы точек, а другой - их ординаты. В этом случае на графике в соответствующих ячейках указываются только имена векторов.

Рис. 33. Уравнения x=f(t), y=y(t).

Если вас не устраивает внешний вид построенных графиков, вы можете его изменить, выделив график (выполнив на нем щелчок, так, чтобы вокруг него появилась рамка) и воспользовавшись командойFormat -> Graph -> X-Y Plot, или, выполнив на графике щелчок правой кнопкой мыши и выбрав команду Format из выпадающего контекстного меню (можно выполнить также двойной щелчок левой кнопкой мыши). В результате на экране появится диалоговое окно Formatting Currently Selected X-Y Plot, позволяющее изменить вид графика.

Данное диалоговое окно содержит несколько вкладок: X-Y Axes (форматирование осей), Traces (тип линий графиков), Labels (подписи), Defaults (по умолчанию).

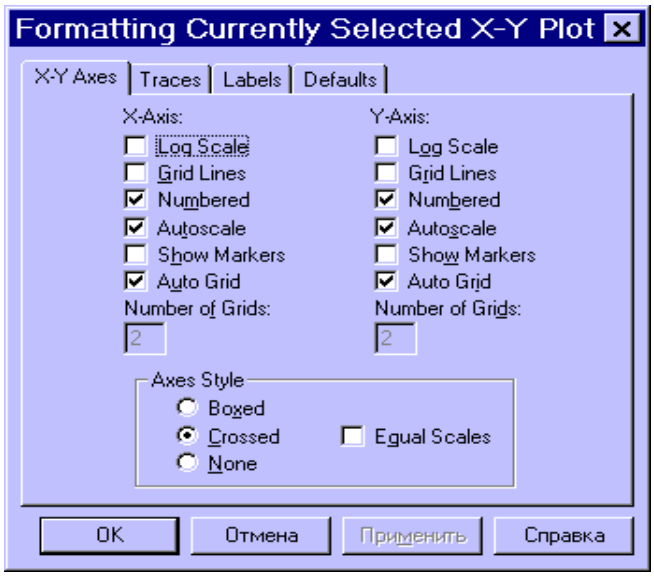
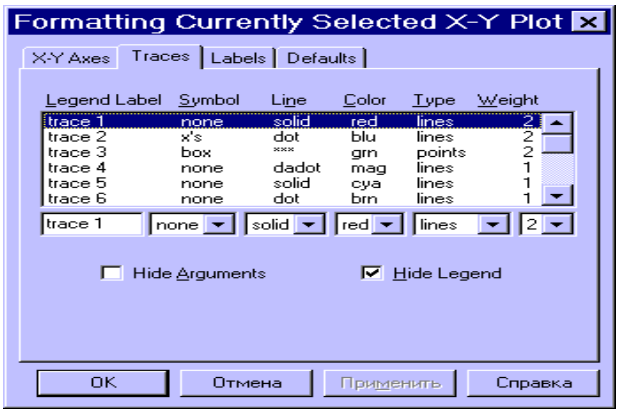


Рис. 34. Окно позволяющая изменить вид графика.

Первая из вкладок позволяет форматировать оси координат:

- **Log Scale** (Логарифмическая шкала) - задает логарифмические оси, в этом случае границы графика должны задаваться положительными числами.
- **Grid Lines** (Вспомогательные линии) - Задает отображение сетки из параллельных осям линий.
- **Numbered** (Нумерация) - Задает отображение подписи к маркировкам на осях.
- **Autoscale** (Автомасштаб) - Задает автоматическое нахождение подходящих границ для осей. Но если вы сами зададите в соответствующих ячейках минимальные и максимальные значения xmin, xmax, ymin, ymax, именно эти значения будут использоваться для определения границ графика.

- **Show Markers** (Показать метки) - Если установить эту опцию, то в графической области появятся четыре дополнительные ячейки для создания красных линий маркировки, соответствующих двум специальным значениям x и двум специальным значениям y.
- **Auto Grid** (Автосетка) - При установке этой опции число линий сетки определяет MathCAD.
- **Axes Style** (Вид осей) - Группа кнопок этой области позволяет выбрать следующие варианты представления осей:
- **Boxed** (ограниченная область)
- **Crossed** (пересечение) - оси пересекаются в точке с координатами (0.0),
- **None** (без границ). Флажок **Equal Scales** (равный масштаб) позволяет задать одинаковый масштаб для обеих осей.



Форматирование оси графика можно произвести, выполнив на ней двойной щелчок. Для изменения типа линий графиков необходимо активизировать вкладку Traces (След)

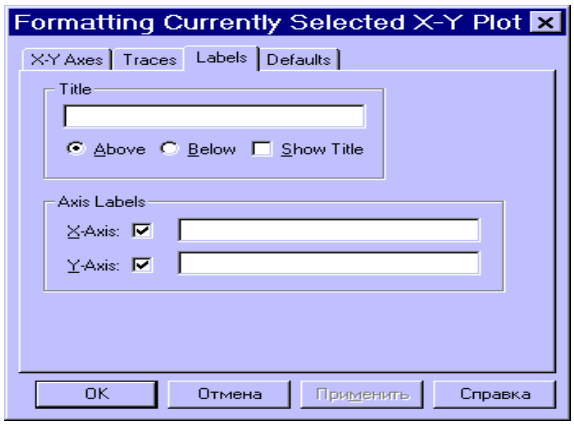
Рис. 35. Окно форматирование оси графика.

- **Legend Lable** (Имя в легенде) - Каждой кривой модно поставить в соответствие некоторый текст, называемый легендой. Легенда отображается в нижней части графической области, а рядом с каждой легендой отображается тип линии соответствующей кривой.
- **Symbol** (Символ) - Позволяет выбрать символ для каждой точки кривой (плюс, крестик, кружок и др.)
- **Line** (Линия) - Можно выбрать один из следующих типов линий: *solid* (сплошная), *dash* (штриховая), *dot* (точечная) или *dadot* (штрихпунктирная). Это поле списка доступно в случае, если в поле **Type** (Тип) выбран элемент *lines*/
- **Color** (Цвет) - Задается цвет представления кривой на экране.
- **Type (Тип)** - Позволяет выбрать один из семи видов графика: в виде кривых (), в виде столбцов () и т. п. Специальным видом графика является тип (погрешность, расхождение), представляющий собой разность двух заданных функций. Величина шага независимой переменной определяет расстояние между отдельными столбцами, ступенями или линиями погрешностей на диаграмме.
- **Weight (Вес)** - Позволяет задавать толщину линий графика.

В нижней части вкладки Traces расположены опции:

- **Hide Arguments** (Скрыть аргументы) - Эта опция по умолчанию отключена. В этом случае под именем функции рядом с осью ординат указывается текущий тип линий. Если установить данную опцию, указание типа линий исчезнет.

– **Hide Legend** (Скрыть легенду) - По умолчанию легенда не отобрадается. Есло вы хототе отобразить под графиком текст легенды, его необходимо перед этим ввести в поле Legend Lable (Имя в легенде) и подтвердить ввод, выполнив щелчок на кнопке Применить.



Вкладка Labels (Метки) позволяет ввести заголовок графика и подписи для осей.

Рис. 36. Вкладка Labels

В подменю **Graph** (Графика) меню **Format** (Формат) содержатся кроме прочих следующие команды:

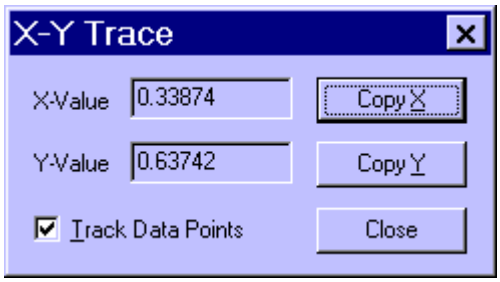


Рис. 37. Вкладка Trace

Trace (След) - При перемещении в области графика указателя мыши при нажатой левой кнопке в полях **X-Value** (значение X) **Y-Value** (значение Y) диалогового окна **X-Y Trace** отображаются координаты точки, на которую указывает курсор. Если установлена опция **Track Data Points** (След точек данных), то курсор-крестик будет перемещаться вдоль графика функции и вы сможете считывать текущее значение аргумента *x* и соответствующее значение функции *y=f(x)*. Координаты текущей точки можно скопировать в буфер при помощи кнопок **Copy X** (Копировать X) **Copy Y** (Копировать Y).

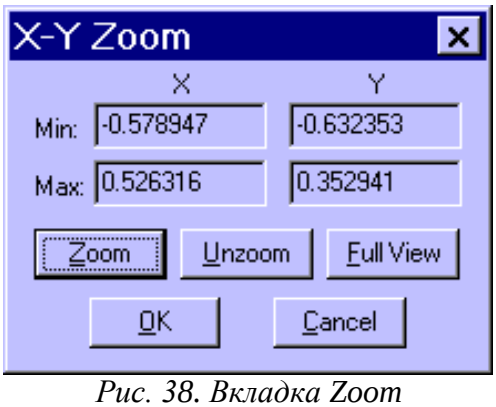


Рис. 38. Вкладка Zoom

Zoom (Изменение масштаба) - При помощи этой команды можно увеличить фрагмент графика, предварительно выделив его протаскиванием мышки с нажатой левой клавишей. После отпускания клавиши координаты углов выделенной области будут отображены в полях окна **X-Y Zoom**. При помощи кнопки **Zoom** (Масштаб +) фрагмент можно увеличить, при помощи кнопки **Unzoom** (Масштаб -) отменить выделение фрагмента, а при помощи кнопки **Full View** (Обзор) - восстановить первоначальный вид графика. Если вы увеличили фрагмент графика, то при щечке на кнопке **OK** в документе будет отображаться только этот фрагмент.

3.В MathCAD так же можно построить и отформатировать график в полярных координатах. Для его построения надо дать команду Insert > Graph > Polar Plot (Вставка > График > Полярные координаты). Для построения простейшего трехмерного графика, необходимо задать матрицу значений. Отобразить эту матрицу можно в виде поверхности – Insert > Graph > Surface Plot (Вставка > График > Поверхность), столбчатой диаграммы – Insert > Graph > 3D Bar Plot (Вставка > График > Столбчатая диаграмма) или линий уровня – Insert > Graph > Contour Plot (Вставка > График > Линии уровня).

Для отображения векторного поля при помощи команды Insert > Graph > Vector Field Plot (Вставка > График > Поле векторов) значения матрицы должны быть комплексными. В этом случае в каждой точке графика отображается вектор с координатами, равными действительной и мнимой частям элемента матрицы. Во всех этих случаях после создания области графика необходимо указать вместо заполнителя имя матрицы, содержащей необходимые значения. Для построения параметрического точечного графика командой Insert > Graph > 3D Scatter Plot (Вставка > График > Точки в пространстве) необходимо задать три вектора с одинаковым числом элементов, которые соответствуют x-, y- и z-координатам точек, отображаемых на графике. В области графика эти три вектора указываются внутри скобок через запятую.

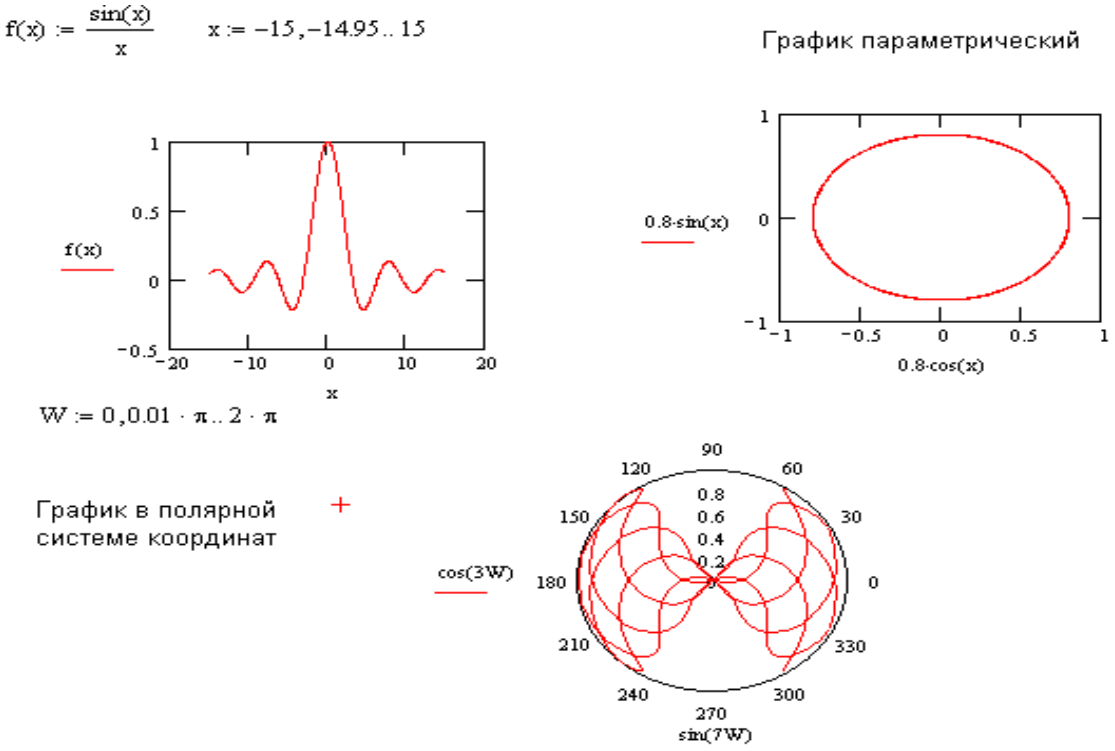


Рис. 39.Отображения графика в полярных координатах.

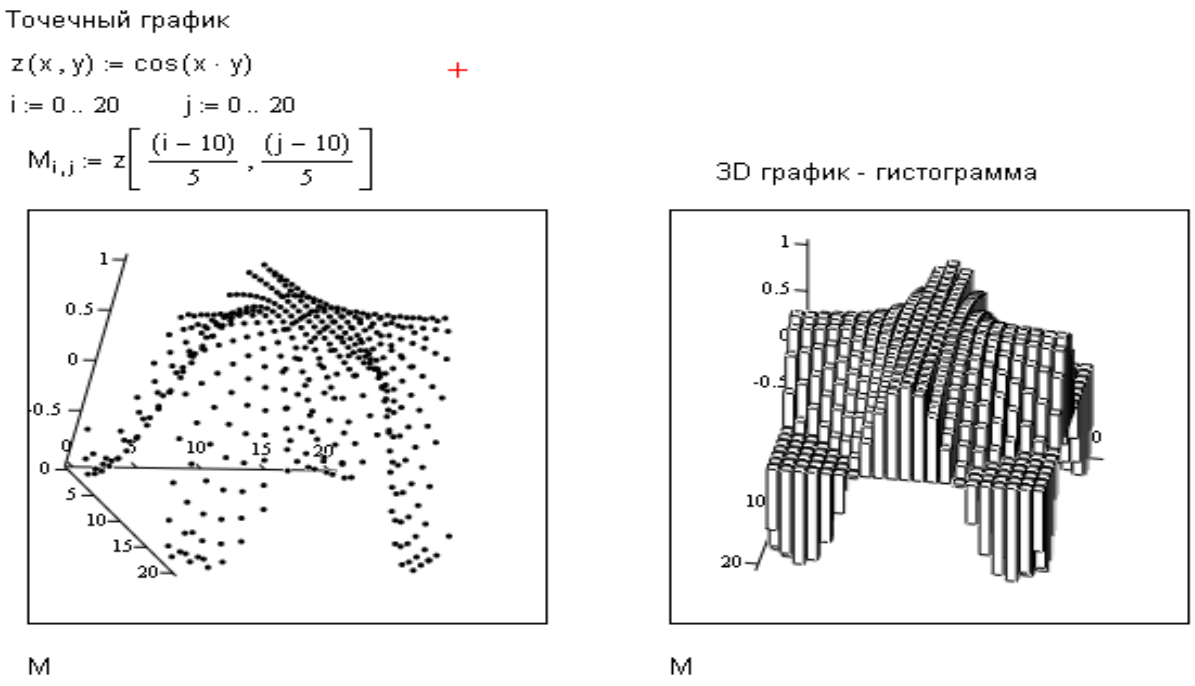


Рис. 40 .Отображения точечного графика.

Аналогичным образом можно построить поверхность, заданную параметрически. Для этого надо задать три матрицы, содержащие, соответственно, *x*-, *y*- и z-координаты точек поверхности. Теперь надо дать команду построения поверхности Insert > Graph >Surface Rot (Вставка > График > Поверхность) и указать в области графика эти три матрицы в скобках и через запятую. Таким образом можно построить практически любую криволинейную поверхность, в том числе с самопересечениями.

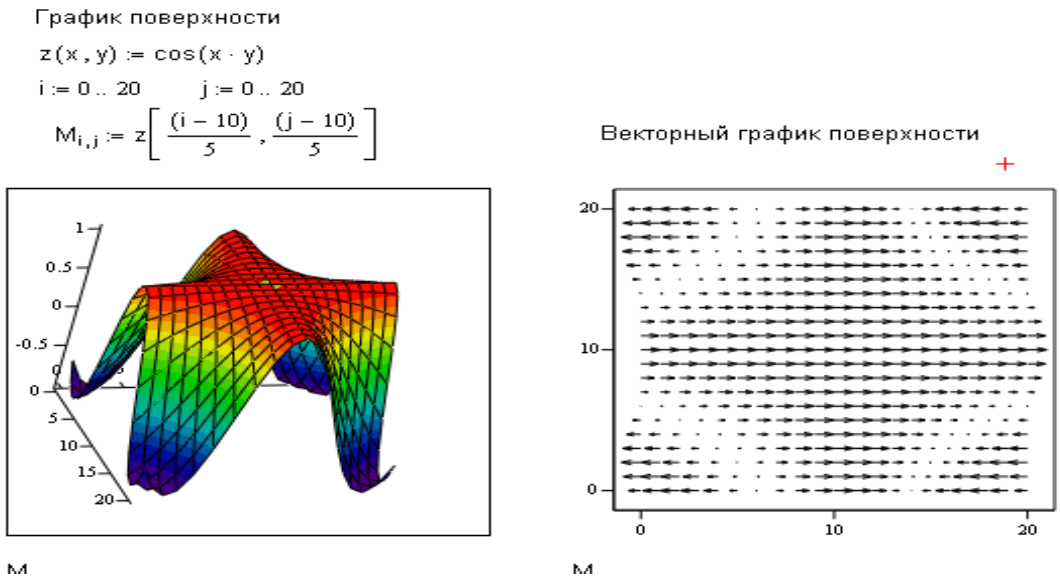


Рис. 41. Отображения поверхности графика.

Контрольные вопросы:

1. Способы построения графиков в системе MATHCAD?
2. Как можно построить кривые заданные параметрически?
3. Особенности трехмерной графики в программе MATHCAD?
4. Какие другие графические возможности программы MATHCAD вам известны?

Список литературы

1. Каримов И.А. Мировой финансово-экономический кризис, пути и меры по его преодолению в условиях Узбекистана.-Ташкент: Узбекистан, 2009.-48с.
2. Закон Республики Узбекистан “Об образование”//Халқ таълими.-Тошкент, 1997.-№5.-с.4-16
3. Ф.Б.Бадалов. Оптималлаш назарияси ва математик программалаштириш. -Т.:Ўқитувчи, 1989.
4. K.Safoeva. Matematik programmalsh. O’quv qo’llanma.T.:UAJBHT, 2004.
5. Т.Х., Ҳолматов Н.И. Тойлоқов. Амалий математика, дастурлаш ва компьютернинг дастурий таъминоти.Т.:“Мехнат”, 2000 й.
6. U.Yuldashev, F.Sharipxodjayeva. Sonli usullar.Qo’llanma. T.: [s.n], 2012
7. Матросов А. Решения задачи математики и механики системе Maple-6. Санкт-Петергбург . 2000
8. Савотченко С.Е., Кузьмичева Т.Г. Методы решения мате-матических задач в Maple. : Учебное пособие - Белгород: Изд. Белаудит, 2001. - 116 с
9. Алексеев Е.Р., Чеснокова О.В. Решение задач вычислитель-ной математики в пакетах Mathcad12, Mathlab 7, Maple 9. 2007
10. Очков В.Ф. "Советы пользователям Mathcad". (Второй выпуск, советы 100-...)
11. Макаров Е. Инженерные расчеты в Mathcad. Изд. Питер. М. 2003г.
12. Плис А.И., Силвина Н.А. Mathcad 2000: Математический практикум для экономистов и инженеров: Учеб.пособие. –М. Финансы и статистика, 2000г.
13. Макаров Е. Г. Инженерные расчеты в Mathcad. Учебный курс. СПб.: Питер, 2003.
14. Mathcad 2001 - что нового. КомпьютерПресс, 4'2001
15. Соловьева Е. Б. Машинное моделирование РЭУ. Математическое моделирование линейных аналоговых цепей в прграммной среде MatLAB. Специальность 200700/ Е. Б. Соловьева, А. И. Солонина, А. И. Солонина. -СПб., 1998.
16. Гаспарян О.Н. MATLAB. Учебное пособие. ГУИА. 2005. – 143 с.
17. Гультяев А. Визуальное моделирование в среде Matlab. Учебный курс. – СПб.: Питер, 2000.
18. Дьяконов В., Круглое В. Математические пакеты расширения Matlab. Специальный справочник. – СПб.: Питер, 2001.
19. Егоренков Д. Л., Фрадков А. Л., Харламов В. Ю. Основы математического моделирования. – СПб.: БГТУ. 1996.
20. Половко А., Бутусов П. Интерполяция. Методика и компьютерные технологии их реализации. – СПб.: БХВ-Петербург, 2004.
21. Потемкин В. Г. Инструментальные Средства Mallab 5_t. – М.: Диалог-МИФИ, 2000.
22. Потемкин В. Система инженерных и научных расчетов Matlab5_r.–М.: Диалог-МИФИ, 1999.
23. Солонина А.И.,Арбузов С.М. Цифровая обработка сигналов . Моделирование в MATLAB. – СПб.:БХВ – Петербург, 2008. – 816 с.:ил. –(Учебное пособие).