МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Набережночелнинский институт (филиал)

Кафедра Бизнес-информатики и математических методов в экономике

Имитационное моделирование экономических процессов. Основные приемы работы в среде MATLAB

Учебно-методическое пособие

Набережные Челны
2019 г.

УДК 510.872
ББК 22.171

Печатается по решению учебно-методической комиссии экономического отделения Набережночелнинского института (филиала) федерального государственного автономного образовательного учреждения высшего
Доктор физ.-мат. наук, профессор А.Г. Исавнин
образования «Казанский (Приволжский) федеральный университет», от «24» января 2019г. (протокол №5)

Рецензенты:
Доктор экономических наук, профессор А.Н. Макаров

Розенцвайг А.К., Шарипов Р.Ш. Имитационное моделирование экономических процессов. Основные приемы работы в среде MATLAB: учебно-методическое пособие / А.К. Розенцвайг, Р. Ш. Шарипов – Набережные Челны: Изд-во Набережночелнинского института КФУ, 2019. – 48 с.

Учебно-методическое пособие содержит последовательное изложение базовых понятий теории визуального моделирования экономических процессов в среде MATLAB. Основные приемы работы в среде MATLAB. Подробно изложены: рабочая среда MATLAB; встроенные элементарные функции; работа с массивами; работа с графикой; редактор М-файлов; программирование в MATLAB; optimization toolbox; statistics toolbox; обмен данными между MATLAB в Excel.

Учебно-методическое пособие предназначено для использования в учебном процессе студентами технических направлений в экономике и экономического отделения дневной, заочной и дистанционной форм обучения.

Оглавление

1. Введение

Пакет MATLAB был создан компанией MathWorks более десяти лет назад. В настоящее время MATLAB является мощным и универсальным средством решения задач, возникающих в различных областях человеческой деятельности. Спектр проблем, исследование которых может быть осуществлено при помощи MATLAB, охватывает: матричный анализ, обработку сигналов и изображений, задачи математической физики, оптимизационные задачи, обработку и визуализацию данных, работу с картографическими изображениями, нейронные сети, нечеткую логику и многое другое.

Огромным преимуществом MATLAB является открытость кода, что дает возможным опытным пользователям разбираться в запрограммированных алгоритмах и, при необходимости, изменять их.

MATLAB прекрасно интегрируется с Microsoft Word и Excel. Связь MATLAB и Word обеспечивает возможность написания в редакторе Word интерактивных документов, так называемых M-книг, основанных на специальном шаблоне. Данное средство прекрасно подходит для создания отчетов и учебных пособий, поскольку позволяет дополнить документ примерами и результатом расчетов.

Информация, хранящаяся в базах данных многих популярных форматов, может быть импортирована в MATLAB, нужным образом обработана и исследована при помощи функции MATLAB, а затем экспортирована в какую-либо другую базу данных.

Простой встроенный язык программирования позволяет легко создавать собственные алгоритмы. Простота языка программирования компенсируется огромным множеством функции MATLAB и ToolBox. Данное сочетание позволяет достаточно быстро разрабатывать эффективные программы, направленные на решение практически важных задач.

2. Рабочая среда MATLAB

При запуске системы по умолчанию открывается три окна: окно команд, рабочая область, команды.

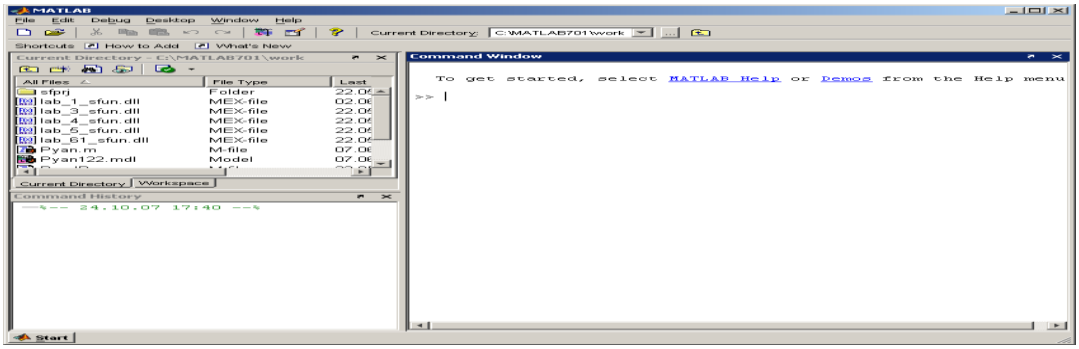

Рис. 1. Окно MATLAB

Командное окно, предназначено для работы с переменными, файлами и функциями системы MATLAB в режиме командной строки.

Для командного окна предусмотрен специальный набор настроек (рис. 2), которые управляют форматом вывода чисел, включают режим вывода на экран исполняемых операторов, регулируют размеры шрифта и цвета.

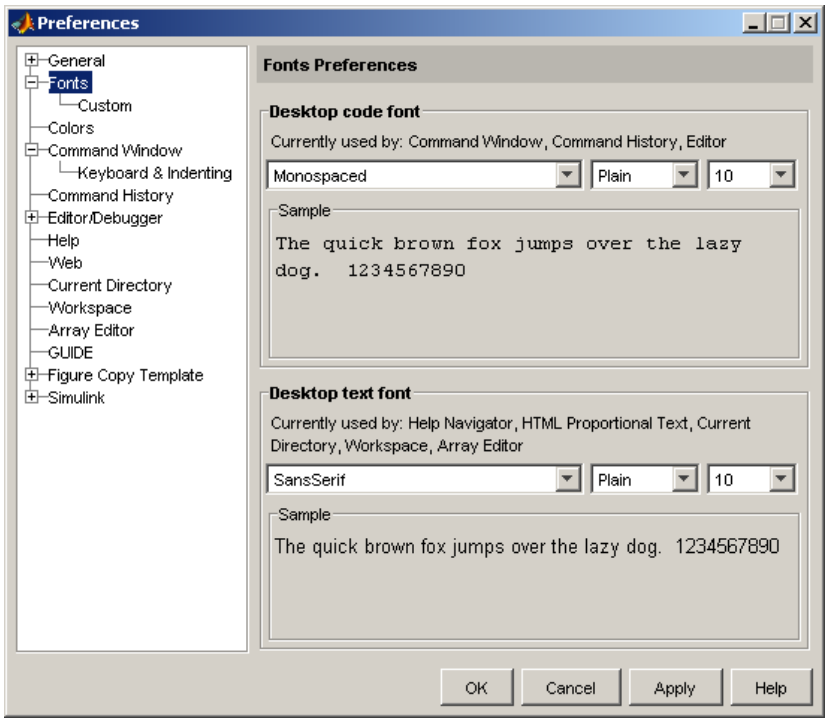

Рис. 2. Окно настроек MATLAB

Набор настроек «Fonts» позволяет управлять параметрами отображения текста. Числовой формат – формат вывода чисел в командное окно. Соответствующий список включает в себя следующие форматы вывода:

- short – краткая запись в формате с фиксированной запятой;
- long – длинная запись в формате с фиксированной запятой;
- short e – краткая запись в формате с плавающей запятой;
- long e – длинная запись в формате с плавающей запятой;
- short g – вторая форма краткой записи в формате с плавающей запятой;
- long g – вторая форма длинной записи в формате с плавающей запятой;
- hex – запись в виде шестнадцатеричного числа;
- bank – запись до сотых долей
- + - записывается только знак числа;
- rational – запись в виде рациональной дроби.

При работе с файлами в системе MATLAB текущий каталог исполняет роль точки отсчета. Любой файл, к которому нужно обратиться, должен размещаться либо в текущем каталоге, либо на пути доступа. Окно просмотра текущего каталога показано на рисунке 3.

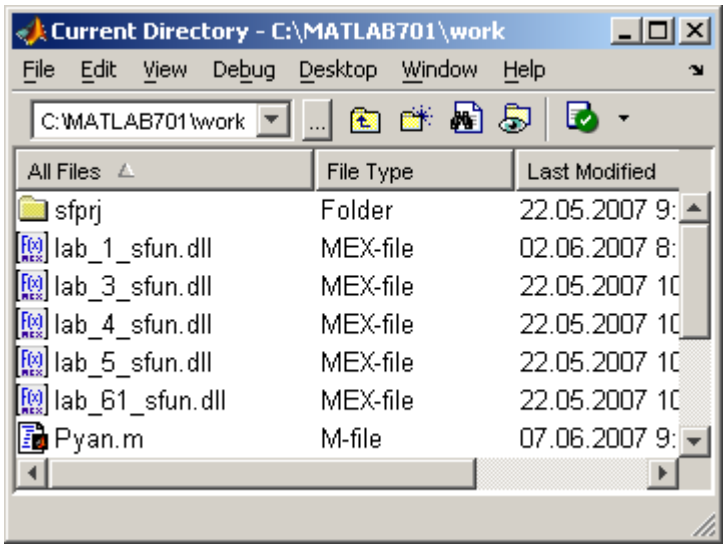

Рисунок 3 – Панель Текущий каталог

В состав MATLAB входят мощная подсистема справки и обширная демонстрационная система. Окно справочной системы делится на две области: область навигации и область просмотра. В Навигаторе помощи пользователь может указать интересующую тему, термин или функцию, а в области просмотра ознакомиться с найденной информацией.

3. Встроенные элементарные функции

3.1. Арифметические вычисления

Встроенные математические функции MATLAB позволяют находить значения различных выражений. MATLAB предоставляет возможность управления форматом вывода результата. Команды для вычисления выражений имеют вид, свойственный всем языкам программирования высокого уровня.

Если набрать в командной строке 1+2 и нажать <Enter>, то в результате в командном окне отображается следующее:

```
>> 1+2
ans =
3
>>
```

В результате MATLAB вычислил сумму **1+2**, затем записал результат в специальную переменную **ans** и вывел ее значение, равное 3, в командное окно.

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**.

```
>> ans/4.5
ans =
```

0.6667

>>

3.2. Элементарные функции

Встроенные элементарные функции включают тригонометрические, гиперболические, экспоненциальные и логарифмические функции, а также функции для работы с комплексными числами и для округления различными способами.

Тригонометрические, гиперболические и обратные к ним функции:

1. **Sin(x), cos(x), tan(x), cot(x)** – синус, косинус, тангенс и котангенс;
2. **Sec(x), csc(x)** – секанс, косеканс;
3. **Asin(x), acos(x), atanh(x), acoth(x)** – арксинус, арккосинус, арктангенс и арккотангенс;
4. **Asech(x), acsch(x)** - арксеканс, арккосеканс;
5. **Sinh(x), cosh(x), tanh(x), coth(x)** - гиперболические синус, косинус, тангенс и котангенс;
6. **Sech(x), csch(x)** – гиперболические секанс и косеканс;
7. **Asinh(x), acosh(x), atanh(x), acoth(x)** – гиперболические арксинус, арккосинус, арктангенс и арккотангенс;
8. **Asech(x), acsch(x)** – гиперболические арксеканс и арккосеканс.

Экспоненциальная функция, логарифмы, степенные функции:

1. **Exp(x)** – экспоненциальная функция;
2. **Log(x)** – натуральный логарифм;
3. **Log10(x)** – десятичный логарифм;
4. **Log2(x)** – логарифм по основанию 2;
5. **Pow2(x)** – возведение числа 2 в степень x;
6. **Sqrt(x)** – квадратный корень;

7. **Nextpow2(x)** – степень, в которую надо возвести число 2, чтобы получить ближайшее число (большое или равное аргументу x).

Функции для работы с комплексными числами:

1. **Complex(x,y)** – конструирует комплексное число по его действительной (x) и мнимой (y) части;

2. **Conj(x,y)** – возвращает комплексно-сопряженное число;

3. **Imag(x), real(x)** – возвращает мнимую и действительную часть комплексного числа x.

Округление и остаток от деления

1. **Fix(x)** – округление до ближайшего целого по направлению к нулю;

2. **Floor(x), ceil(x)** - округление до ближайшего целого по направлению к минус бесконечности или плюс бесконечности;

3. **Round(x)** – округление до ближайшего целого;

4. **Mod(x,y)** – остаток от целочисленного деления **x** на **y** (со знаком);

5. **Rem(x,y)** – остаток от целочисленного деления **x** на **y**;

6. **Sign(x)** – возвращает знак числа.

4. Работа с массивами

MATLAB является системой, которая предназначена для осуществления сложных операций с векторами, матрицами и полиномами. Под вектором понимается одномерный массив чисел, а под матрицей – двумерный массив. Начальные значения векторов можно вводить с клавиатуры поэлементно. Для этого в строке следует указать сначала имя вектора, потом поставить знак присваивания, далее – в квадратных скобках указать элементы вектора, разделенные пробелами или запятыми. Значения элементов матрицы вводят в квадратных скобках, по строкам, разделенным точкой с запятой.

Например:

>> V=[4.2 -5.3 -100]

V =

4.2000 -5.3000 -100.0000

>> M=[3 6 -3;7 7 2; -4 -2 2]

M =

3 6 -3

7 7 2

-4 -2 2

В MATLAB имеется несколько встроенных функций, которые позволяют формировать векторы и матрицы определенного вида.

Функции, формирующие векторы и матрицы:

1. **zeros(m,n) -** Создает матрицу размером **M x N** с нулевыми элементами;
2. **ones(m,n) -** Создает матрицу размером **M x N** с единичными элементами;
3. **eye(m,n) -** Создает единичную матрицу размером **M x N**, с единицами по главной диагонали;
4. **rand(m,n) -** Создает матрицу размером **M x N** из случайных чисел, равномерно с распределенными в диапазоне от 0 до 1;
5. **randn(m,n) -** Создает матрицу размером **M x N** из случайных чисел, распределенных по нормальному (гауссову) закону с нулевым математическим ожиданием и стандартным отклонением, равным 1;
6. **hadamard(n) -** Создает матрицу Адамара размером **N x N**
7. **hilb(n) -** Создает матрицу Гильберта размером **N x N;**
8. **invhilb(n) -** Создает обратную матрицу Гильберта размером **N x N;**
9. **pascal(n) -** Создает матрицу Паскаля размером **N x N;**

Предусмотрено несколько функций, которые позволяют формировать одну матрицу на основе заданных матрицы или вектора.

Функции, формирующие матрицы из заданных матриц и векторов:

1. **fliplr(A) -** Формирует матрицу, переставляя столбцы известной матрицы **A** относительно вертикальной оси, то есть меняя местами левую и правую стороны матрицы;
2. **flipud(A) -** Переставляет строки заданной матрицы **A** относительно горизонтальной оси, то есть меняя местами верхнюю и нижнюю стороны матрицы;
3. **rot90(A) -** Формирует матрицу путем «поворота» заданной матрицы **A** на 90^o против часовой стрелки;
4. **reshape(A,m,n) -** Образует матрицу размером **M x N**, выбирая из столбцов элементы заданной матрицы **A** и распределяя их по столбцам, каждый из которых содержит **M** элементов;
5. **tril(A) -**Образует нижнюю треугольную матрицу на основе матрицы **A** путем обнуления ее элементов выше главной диагонали;
6. **triu(M) -** Образует верхнюю треугольную матрицу на основе матрицы **A** путем обнуления ее элементов ниже главной диагонали;
7. **hankel(V) -** Образует квадратную матрицу Ганкеля, первый столбец которой совпадает с заданным вектором **V;**
8. **diag(A) -** Извлекает диагональ матрицы **A.**

Извлечение и вставка частей матриц осуществляется путем указания индекса строки и столбца матрицы. Например:

```
>> M
M =
 3    6   -3
 7    7    2
-4   -2    2
```

```
>> M(2,3)
ans =
2
```

Если нужно поместить в указанное место число, то необходимо выполнить следующее:

```
>> M
M =
    3     6    -3
    7     7     2
   -4    -2     2
>>

>> M(2,1)=300
M =
        3     6    -3
      300     7     2
       -4    -2     2
>>
```

Если требуется создать меньшую матрицу на основе большей, формируя ее путем извлечения из последней матрицы элементов ее нескольких строк и столбцов, или вставить меньшую матрицу таким образом, чтобы она стала определенной частью матрицы большего размера, то используется разделительный символ « **:** » (двоеточие) с указанием номера строки или столбца, или путем определения верхней и нижней границы. Например, необходимо создать вектор из третьего столбца некоторой матрицы A. Для этого нужно записать:

```
>> V=M(:,3)
V =
    -3
     2
     2
>>
```

Или, например необходимо из матрицы размером **3x4** образовать матрицу **B** размером **2x2**, которая состоит из элементов левого нижнего угла матрицы **M**:

```
>> B=M(2:3,1:2)
B =
   300     7
    -4    -2
>>
```

Основные действия над векторами представлены в таблице 1.

Таблица 1. Действия над векторами

Операция	**Пример использования**
Сложение векторов	**>> x=[2 5 1];** **>> y=[2 -2 7];** **>> z=x+y** **z =** **4 3 8**
Вычитание векторов	**>> x=[2 5 1];** **>> y=[2 -2 7];** **>> z=x-y**

	z = **0 7 -6**
Транспонирование вектора	**>> x=[2 5 1];** **>> z=x'** **z =** **2** **5** **1**
Умножение вектора на число	**>> x=[2 5 1];** **>> z=2*x** **z =** **4 10 2**
Умножение двух векторов	**>> x=[2 5 1];** **>> y=[2 -2 7];** **>> z=x'*y** **z =** **4 -4 14** **10 -10 35** **2 -2 7** **>> z=x*y'** **z =** **1**
Векторное произведение двух векторов	**>> x=[2 5 1];** **>> y=[2 -2 7];** **>> z=cross(x,y)** **z =** **37 -12 -14**

В языке MATLAB предусмотрено выполнение ряда операций, позволяющих преобразовать заданный вектор в другой вектор, имеющий такой же размер и тип. К таким операциям относятся, в частности, все операции,

осуществляемые с помощью элементарных математических функций одного аргумента (например, y=sin(x)). Кроме таких операций предусмотрено несколько операций поэлементного преобразования. Такие операции представлены в таблице 2.

Таблица 2. Поэлементное преобразование векторов

Операция	Назначение	Пример использования
+ (-)	Добавление числа к каждому элементу (вычитание числа из каждого элемента) вектора	>> **x=[2 5 1];** >> **z=x+2** **z =** **4 7 3** >>
.*	Поэлементное умножение векторов	>> **x=[2 5 1];** >> **z=3.*x** **z =** **6 15 3** >>
./	Поэлементное деление векторов	>> **z=3./x** **z =** **1.5000 0.6000 3.0000** >>
.	Поэле	>> **x=[2 5**

\	ментное деление векторов в обратном направлении	**1];** **1.6667 0.3333**	**>> z=3.\x** **z =** **0.6667** **>>** **>> x=[2 5** **1];** **>> z=3.^x** **z =** **9 243** **>> z=x.^3** **z =** **8 125** **>**
. ^	Поэлементное возведение в степень	**3** **1**	

Для поэлементного преобразования матрицы пригодны все алгебраические функции. Они формируют матрицу того же размера, что и исходная матрица, у которой каждый элемент вычисляется как значение указанной функции от соответствующего элемента заданной матрицы. Для матриц определены также операции поэлементного умножения матриц одинакового размера **(.*)**, поэлементного деления (**./** и **.**), поэлементного возведения в степень **(.^)**, а также прибавления к матрице числа (A+k или k+A, где A матрица, k-некоторое число. В результате будет получена матрица такого же размера, что и матрица A, все элементы которой будут изменены на величину k).

Операции над матрицами:

1. **A+B (A-B)** - Сложение (вычитание) матриц
2. **k*A (A*k)** - Умножение матрицы на число

3. **A'** - Транспонирование матрицы

4. **A*B (A*B)** - Умножение матрицы на матрицу

5. **inv(C)** - Обращение матрицы

6. **A^ n** - Возведение матрицы в целую степень

7. **/ (\ )** - Деление матриц слева направо (справа на лево)

8. Функция **exp(A)** формирует матрицу, значение каждого элемента которой равняется e в степени, равной соответствующему элементу матрицы A;

9. Функция **logm(A)** логарифмирует матрицу по натуральному основанию;

10. Функция **sqrtm(A)** вычисляет матрицу Y, такую, что Y*Y=A;

11. Функция **cond(A)** возвращает число обусловленности матрицы относительно операции обращения, которое равняется отношению максимального сингулярного числа матрицы к минимальному;

12. Функция **norm(v,p)** , где p — целое положительное число, — возвращает корень степени p из суммы абсолютных значений элементов вектора, возведенных в степень p. При p = 1 это может совпадать либо с первой нормой, либо с нормой неопределенности матриц. **Норма вектора** - скаляр, дающий представление о величине элементов **вектора**

13. Функция **norm(A,p)** вычисляет p - норму матрицы, где параметр **p** может принимать одно из следующих значений: 1,2 .Если аргумент **p** не указан, вычисляется 2-норма.

14. Функция **recond(A)** вычисляет величину, которая обратна значению числа обусловленности матрицы A относительно 1-нормы. Если матрица A хорошо обусловлена, то возвращаемое функцией значение близко к 1. если же она плохо обусловлена, оно близко к 0.

15. Функция **rank(A)** вычисляет ранг матрицы.

16. Функция **det(A)** вычисляет определитель квадратной матрицы.

17. Функция **trace(A)** вычисляет след матрицы A, равный сумме ее диагональных элементов.

18. Функция **null(A)** вычисляет ортонормированный базис матрицы A. Базис называется ортонормированным, если его векторы попарно ортогональны и равны единице.

19. Функция **orth(A)** возвращает ортонормированный базис матрицы A;

20. Функция **rref(A)** формирует треугольную матрицу, используя метод исключения Гаусса с частичным выбором ведущего элемента.

5. Работа с графикой

Одно из достоинств системы MATLAB — обилие средств графики, начиная от команд построения простых графиков функций одной переменной в декартовой системе координат и кончая комбинированными и презентационными графиками с элементами анимации, а также средствами проектирования графического пользовательского интерфейса (GUI). Особое внимание в системе уделено трехмерной графике с функциональной окраской отображаемых фигур и имитацией различных световых эффектов.

Основной функцией, обеспечивающей построение графиков на экране дисплея, является функция **plot**. Общая форма обращения к ней:

plot(x1,y1,s1,x2,y2,s2…),

где **x1,y1** – заданные векторы, элементами которых являются массивы значений аргумента **(x1)** и функции **(y1)**, отвечающие первой кривой графика; **x2, y2** – массивы значений аргумента и функции второй кривой и так далее; **s1, s2** – символьные переменные, способные содержать до трех символов, обозначающих тип линии, соединяющей отдельные точки графика, тип точки графика, цвет линии.

Графики выводятся в отдельное графическое окно, которое называют фигурой. Например, для того, чтобы вывести график функции $y = 3\sin(x + \frac{\pi}{3})$ для

значений аргумента от -3π до $+3\pi$ с шагом $\frac{\pi}{100}$ необходимо сначала сформировать массив значений аргумента **x**:

>> x = -3*pi : pi/100 : 3*pi;

Затем вычислить массив соответствующих значений функции:

>> y = 3*sin(x+pi/3);

После этого построить график зависимости y(x):

>> plot(x,y)

Для вывода координатной сетки необходимо добавить процедуру **grid**. Заголовок графика выводиться с помощью процедуры **title('текст').** Над графиком появиться текст, записанный между апострофами в скобках. Аналогично выводятся объяснения к графику, которые размещаются вдоль горизонтальной оси (функция **xlabel**) и вдоль вертикальной оси (функция **ylabel**).

Например, набрав последовательность ниже приведенных операторов можно получить график, представленный на рисунке 3:

>> x = -3*pi : pi/100 : 3*pi;
>> y = 3*sin(x+pi/3);
>> plot(x,y), grid;
>> title ('function y=3*sin(x+pi/3)');
>> xlabel ('x'); ylabel('y');

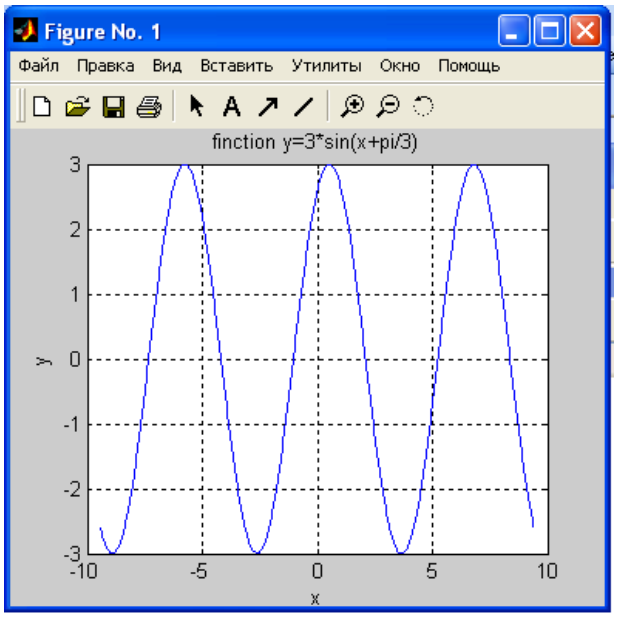

Рис. 3. График функции

Система MATLAB предоставляет возможность не указывать аргумент функции при построении графика. В этом случае в качестве аргумента система принимает номер элемента вектора, график которого строиться. Для построения графика вектора задают сам вектор, например **x=[1 2 3 4]** и функцию **plot(x).**

Вектор можно представить в виде столбцовой диаграммы с помощью функции **bar(x).**

Если функция задана своими значениями при дискретных значениях аргумента и неизвестно, как она может изменяться в промежутках между значениями аргумента, удобнее представлять ее график в виде отдельных вертикальных линий для любого из заданных значений аргумента. Это можно сделать, применяя процедуру **stem**.

Для построения графика гистограммы заданного вектора используется функция **hist(y,x)**, где **y** – вектор, гистограмму которого нужно построить; **x** – вектор, элементы которого определяют интервалы изменения первого вектора.

Процедура **comet(x,y)** строит график зависимости **y(x)** постепенно в виде траектории кометы. При этом «изображающая» точка на графике имеет вид маленькой кометы, которая плавно перемещается от одной точки к другой.

Такие графики удобно использовать при анализе характера изменения траектории во времени.

MATLAB имеет несколько функций, которые позволяют строить графики в логарифмическом масштабе.

Функции, строящие графики в логарифмическом масштабе:

1. **logspace(d1,d2,n)** - Формирует вектор строку, содержащую n равноотстоящих в логарифмическом масштабе друг от друга значений в диапазоне от 10^{d1} до 10^{d2}
2. **loglog** - Создает графики по обеим осям в логарифмическом масштабе
3. **semilogx** - Создает графики с логарифмическим масштабом по оси **x**
4. **semilogy** - Создает графики с логарифмическим масштабом по оси **y**

Высокоуровневая графическая подсистема MATLAB автоматически реализует трёхмерную графику без специальных усилий со стороны пользователя. Пусть в точке с координатами **x1,y1** вычислено значение функции **z=f(x,y)** и оно равно **z1**. В некоторой другой точке (то есть при другом значении аргументов) **x2,y2** вычисляют значение функции **z2**. Продолжая этот процесс, получают массив (набор) точек **(x1,y1,z1)**, **(x2,y2,z2), … (xN,yN,zN)** в количестве **N** штук, расположенных в трёхмерном пространстве. Специальные функции системы MATLAB проводят через эти точки гладкие поверхности и отображают их проекции на плоский дисплей компьютера.

Чаще всего точки аргументов расположены в области определения функции регулярно в виде прямоугольной сетки (то есть матрицы). Такая сетка точек порождает две матрицы одной и той же структуры: первая матрица содержит значения первых координат этих точек (**x** - координат), а вторая матрица содержит значения вторых координат (**y** - координат). Обозначим первую матрицу как **X**, а вторую - как **Y**. Есть ещё и третья матрица - матрица

значений функции **z=f(x,y)** при этих аргументах. Эту матрицу обозначим буквой **Z**.

Простейшей функцией построения графика функции двух переменных в системе MATLAB является функция:

plot3(X , Y , Z), где **X**, **Y** и **Z** - матрицы одинаковых размеров

В системе MATLAB имеется специальная функция для получения двумерных массивов **X** и **Y** по одномерным массивам **x, y**.

Пусть по оси x задан диапазон значений в виде вектора

u = -2 : 0.1 : 2 , а по оси y этот диапазон есть **v = -1 : 0.1 : 1 .**

Для получения матриц **X** и **Y**, представляющих первые и вторые координаты получающейся прямоугольной сетки точек используют специальную функцию системы MATLAB:

[X , Y] = meshgrid(u, v)

Эта функция получает на входе два одномерных массива (вектора), представляющие массивы точек на осях координат, и возвращает сразу два искомых двумерных массива. На прямоугольной сетке точек вычисляют значения функции, например функции **exp**:

Z = exp(- X.^2 - Y.^2)

Наконец, применяя описанную выше функцию **plot3**, получаем следующее изображение трёхмерного графика этой функции:

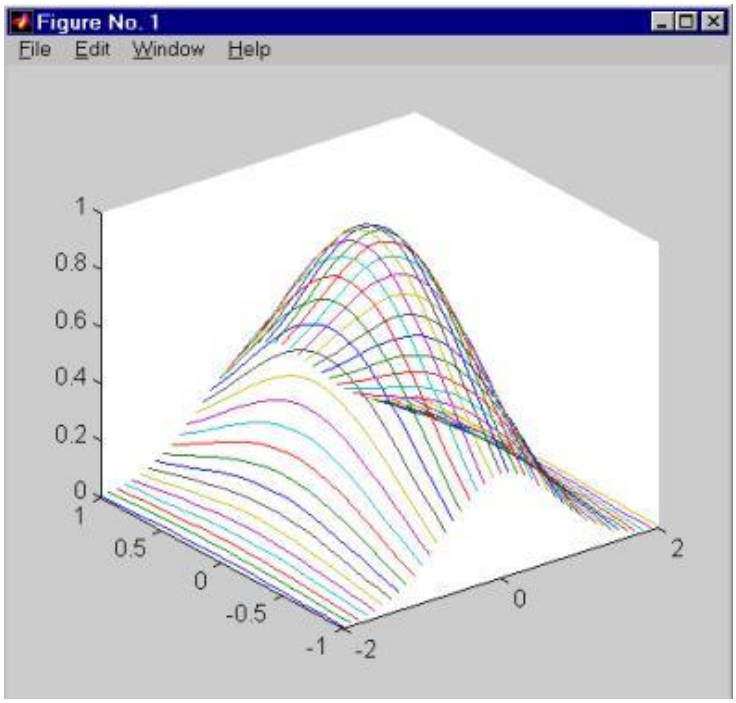

Рис 4. Использование функции plot3

Из этого рисунка видно, что функция **plot3** строит график в виде набора линий в пространстве, каждая из которых является сечением трёхмерной поверхности плоскостями, параллельными плоскости **yOz**. По-другому можно сказать, что каждая линия получается из отрезков прямых, соединяющих набор точек, координаты которых берутся из одинаковых столбцов матриц **X, Y** и **Z**. То есть, первая линия соответствует первым столбцам матриц **X, Y Z**; вторая линия - вторым столбцам этих матриц и так далее.

Помимо этой простейшей функции система MATLAB располагает ещё рядом функций, позволяющих добиваться большей реалистичности в изображении трёхмерных графиков. Это функции **mesh, surf** и **surfl**.

Функция **mesh** соединяет вычисленные соседние точки поверхности графика отрезками прямых и показывает в графическом окне системы MATLAB плоскую проекцию такого объёмного тела. Вместо ранее показанного при помощи функции **plot3** графика функции **exp(- X.^2 - Y.^2)** можно получить изображение, показанное на рис 5.

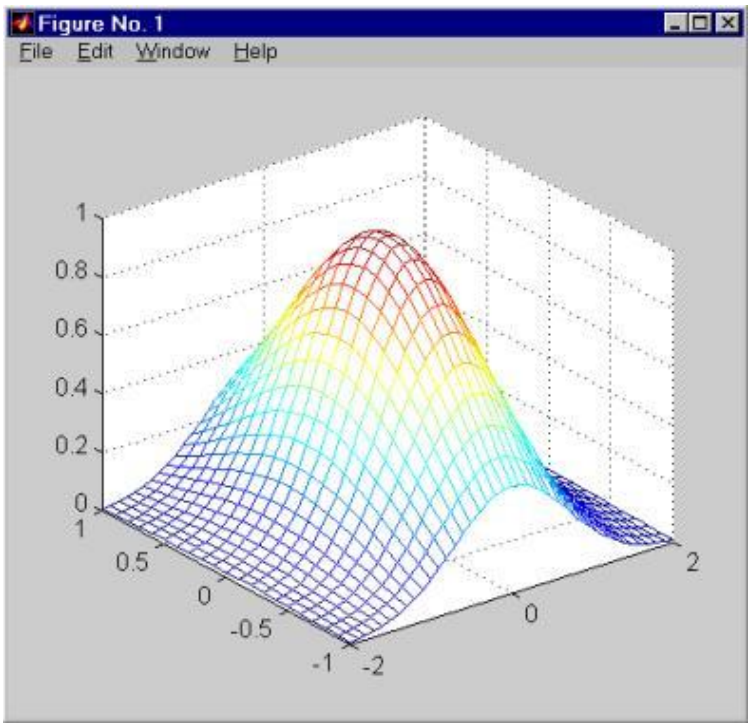

Рис 5. Использование функции mesh

Для лучшего восприятия «объёмности» изображения разные рёбра автоматически окрашиваются в разные цвета. Кроме того (в отличие от функции **plot3**) осуществляется удаление невидимых линий. Если нет необходимости скрывать задние линий, то можно ввести команду **hidden off** , после чего такие линии появятся на изображении. Более плотного изображения поверхности можно добиться, если вместо функции **mesh** применить функцию **surf(X, Y, Z).**

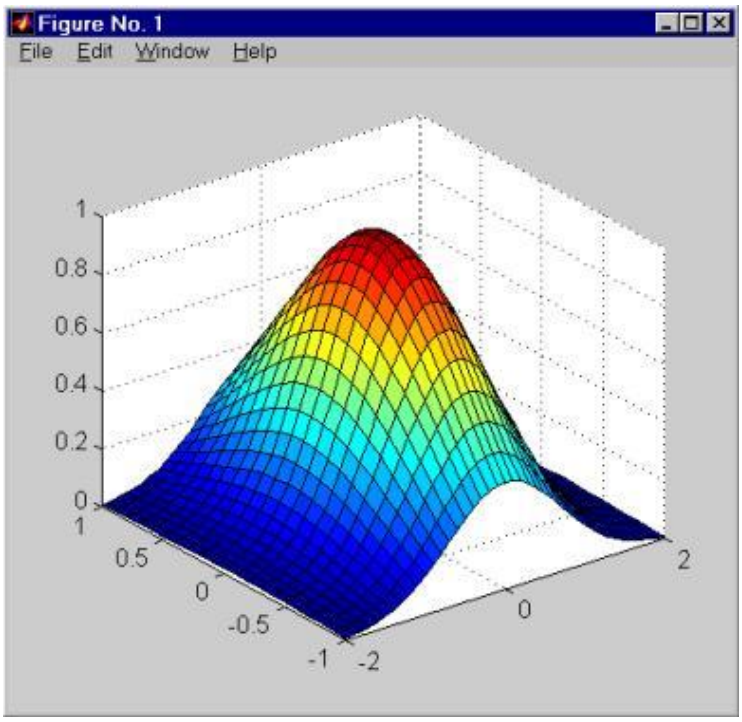

Рис 6. Использование функции surf

В результате получается изображение представляющее плотную (непрозрачную) сетчатую поверхность, причём отдельные ячейки (грани) этой сетчатой поверхности (плоские четырёхугольники) автоматически окрашиваются в разные цвета.

С помощью функции **surf** получаются хотя и искусственно раскрашенные, но весьма наглядные изображения. Если же необходимо добиться более естественных и объективных способов окрашивания поверхностей, то следует использовать функцию **surfl.**

Функция **surfl** трактует поверхность графика как материальную поверхность с определёнными физическими свойствами по отражению света. По умолчанию задаётся некоторый источник света, освещающий такую материальную поверхность, после чего рассчитываются траектории отражённых лучей, попадающих в объектив условной камеры. Изображение в такой камере и показывается в графическом окне системы MATLAB.

Так как разные материалы по-разному отражают падающие лучи, то можно подобрать некоторый материал, чтобы получить наилучшее (с точки зрения пользователя) изображение. В частности, можно использовать функцию:

colormap(copper)

С помощью которой для изображения графика выбирается набор цветов (по-английски - colormap), который характерен для света, отражающегося от медной поверхности (медь по-английски - copper). После этого применение функции

surfl(X, Y, Z) приводит к получению очень реалистически выглядящего и очень наглядного графика:

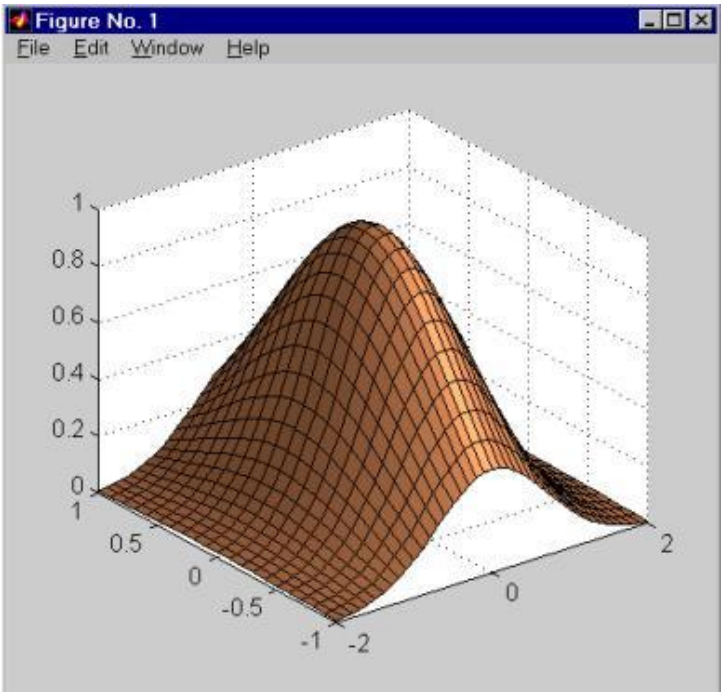

Рис7. Использование функции surf1

Можно с такого графика убрать чёрные линии, изображающие рёбра, а также добиться ещё более плавного перехода освещения поверхности, если выполнить команду **shading interp.** Означающую, что теперь цвет (освещённость) будет меняться даже внутри отдельных граней (ячеек). В итоге будет получаться совсем уж реальное изображение некоторой объёмной фигуры.

6. Редактор M-файлов

В MATLAB имеется редактор M-файлов, для запуска которого следует нажать кнопку New M-file на панели инструментов рабочей среды,

либо выбрать в меню File в пункте New подпункт M-file. На экране появляется окно редактора. Содержимое М-Файла аналогично командам, вводимым в рабочей области MATLAB:

Пример построения графика функции **y=exp(x)**:

```
x=[-1 : 0.01 : 1];
y=exp(x);
plot(x,y)
grid on
title('Экспоненциальная функция')
```

Для запуска программы или ее части есть несколько способов. Первый, самый простой — выделить операторы и выбрать в меню Text пункт Evaluate Selection (или нажать <F9>). Выделенные операторы выполняются последовательно, точно так же, как если бы они были набраны в командной строке.

После того, как программа сохранена в М-файле, для ее запуска можно использовать пункт Run меню Tools Debug, либо набрать в командной строке имя М-файла (без расширения) и нажать <Enter>, то есть выполнить, как команду MATLAB. При таких способах запуска программы следует учесть важное обстоятельство — путь к каталогу с М-файлом должен быть известен MATLAB.

Когда текущий каталог установлен, то все М-файлы, находящиеся в нем, могут быть запущены из командной строки, либо из редактора М-файлов. Все переменные файл-программы после ее запуска доступны в рабочей среде, т. е. являются глобальными. Убедиться в этом можно, выполнив команду **whos.** Более того, файл-программа может использовать переменные рабочей среды. Например, если была введена команда:

```
>> a=[0.1 0.4 0.3 1.9 3.3]
```

то файл-программа, содержащая строку **bar(a)**, построит столбцевую диаграмму вектора **a** (разумеется, если он не был переопределен в самой файл-программе).

Файл-функции с одним выходным аргументом

Файл-функции отличаются от файл-программ тем, что они могут иметь входные и выходные аргументы, а все переменные, определенные внутри файл-функции, являются локальными и не видны в рабочей среде. М-файл, содержащий файл-функцию, должен начинаться с заголовка, после него записываются операторы MATLAB. Заголовок состоит из слова **function**, списка выходных аргументов, имени файл-функции и списка входных аргументов. Аргументы в списках разделяются запятой:

```
function c=mysum(a,b)
c=a+b;
```

При сохранении М-файла MATLAB предлагает в качестве его имени название файл-функции. Имя М-файла обязательно должно совпадать с именем функции! Вызов файл-функции осуществляется следующим образом:

```
>> s=mysum(2,3)
s =
     5
```

При вызове файл-функции **mysum** произошли следующие события:

- входной аргумент **a** получил значение 2;
- входной аргумент **b** стал равен 3;
- сумма **a** и **b** записалась в выходной аргумент **c**;

- значение выходного аргумента **c** получила переменная **s** рабочей среды и результат вывелся в командное окно.

Оператор **c=a+b** в функции mysum завершен точкой с запятой для подавления вывода локальной переменной **c** в командное окно. Для просмотра значений локальных переменных при отладке файл-функций, очевидно, не следует подавлять вывод на экран значений требуемых переменных.

Практически все функции MATLAB являются файл-функциями и хранятся в одноименных М-файлах. Функция **sin** допускает два варианта вызова: **sin(x)** и **y=sin(x)**, в первом случае результат записывается в **ans**, а во втором — в переменную **y.** Приведенная в качестве примера функция **mysum** ведет себя точно так же. Более того, входными аргументами **mysum** могут быть массивы одинаковых размеров или массив и число.

Файл-функции с несколькими выходными аргументами

Список выходных аргументов в заголовке файл-функции заключается в квадратные скобки, сами аргументы отделяются запятой.

Пример. Вычислить значения корней квадратного уравнения по его заданным коэффициентам:

```
function [x1,x2]=quadeq(a,b,c)
D=b^2-4*a*c;
x1=(-b+sqrt(D))/(2*a);
x2=(-b-sqrt(D))/(2*a);
```

При вызове **quadeq** из командной строки необходимо использовать квадратные скобки для указания переменных, в которые будут занесены значения корней:

```
>> [r1,r2]=quadeq(1,3,2)
```

r1 =

-1

r2 =

-2

Файл-функцию **quadeq** можно вызвать без выходных аргументов, или только с одним выходным аргументом. В этом случае вернется только первый корень.

Файл-функция может и не иметь входных или выходных аргументов, заголовки таких файл-функций приведены ниже:

function noout(a,b), function [v,u]=noin, function noarg()

Умение писать собственные файл-функции и файл-программы необходимо как при программировании в MATLAB, так и при решении различных задач средствами MATLAB (в частности, поиска корней уравнений, интегрирования, оптимизации).

7. Программирование в MATLAB

Язык программирования MATLAB достаточно простой, он содержит основной набор конструкций: операторы ветвления и циклы. Простота языка программирования окупается огромным количеством встроенных функций, которые позволяют решать задачи из различных областей.

Цикл **for** используется для повторения операторов в случае, когда число повторений заранее известно. В цикле **for** используется счетчик цикла, его начальное значение, шаг и конечное значение указываются через двоеточие. Блок операторов, размещенный внутри цикла, должен заканчиваться словом **end**.

Пример использования цикла **for**.

Построить график функции $f(x,\beta) = e^{\beta \cdot x} \cdot \sin(x)$ на отрезке [-2;2], для значений параметра $\beta \in [-0.5, 0.5]$.

x = [-2 : 0.01 : 2];

```
for beta = -0.5 : 0.1 : 0.5
y = exp(beta*x).*sin(x);
plot(x,y)
hold on
end
hold off
```

Если шаг равен единице, то его указывать не обязательно. Например, для вычисления суммы

$$\sum_{k=1}^{10} \frac{x^k}{k!}$$

при различных значениях **x** потребуется файл-функция **sum10**, текст которой приведен ниже. Функция **sum10** может быть вызвана как от числа, так и от массива значений, благодаря применению поэлементных операций.

```
function s = sum10(x)
s = 0;
for k = 1 : 10
s = s+x.^k/factorial(k);
end
```

Цикл **for** подходит для повторения заданного числа определенных действий. В том случае, когда число повторов заранее неизвестно и определяется в ходе выполнения блока операторов, следует организовать цикл **while**. Цикл **while** работает, пока выполнено условие цикла. Файл-функция **negsum** (см. ниже) находит сумму всех первых отрицательных элементов вектора.

```
function s = negsum(x)
s = 0;
```

```
k = 1;
while x(k)<0
s = s+x(k);
k = k+1;
end
```

В качестве операторов отношения используются символы: >, < , >=, <=, == (равно), ~= (не равно). Файл-функция **negsum** имеет один недостаток: если все элементы массива — отрицательные числа, то **k** становится больше длины массива **x**, что приводит к ошибке, например:

```
>>b=[-2 -7 -1 -9 -2 -5 -4];
>>s=negsum(b)
??? Index exceeds matrix dimensions.
```

Кроме проверки значения **x(k)** следует позаботится о том, чтобы значение **k** не превосходило длины вектора **x**. Вход в цикл должен осуществляться только при одновременном выполнении условий **k<=length(x)** и **x(k)<0**, т. е. необходимо применить логический оператор « и », обозначаемый в MATLAB символом **&**: **k<=length(x) & x(k)<0**. Если первое из условий не выполняется, то второе условие проверяться не будет, именно поэтому выбран такой порядок операндов.

Логический оператор « или » обозначается символом вертикальной черты (**|**), а отрицание - при помощи тильды (**~**). Ниже приведены логические операции по мере убывания их приоритета:

- отрицание **~** ;
- операторы отношения **>, < , >=, <=, == , ~=** ;
- логическое « и » **&** ;
- логическое « или » **|** .

Для изменения порядка выполнения логических операторов используются круглые скобки.

Циклы могут быть вложены друг в друга. Например, для поиска суммы элементов матрицы, расположенных выше главной диагонали, следует использовать два цикла **for**, причем начальное значение счетчика внутреннего цикла зависит от текущего значения счетчика внешнего цикла (см. ниже).

```
function s=upsum(A)
[n m]=size(A);
s=0;
for i=1:n
for j=i+1:m
s=s+A(i,j);
end
end
```

Ветвление в ходе работы программы осуществляется при помощи конструкции **if-elseif-else**. Самый простой вариант ее использования (без **elseif** и **else**) реализован в файл-функции **possum** (см. ниже), которая предназначена для нахождения суммы всех положительных элементов вектора.

```
function s=possum(x)
s=0;
for k=1:length(x)
if x(k)>0
s=s+x(k);
end
end
```

Если ход программы должен изменяться в зависимости от нескольких условий, то следует использовать полную конструкцию **if-elseif-else**. Каждая из ветвей **elseif** в этом случае должна содержать условие выполнения блока операторов, размещенных после нее. Важно понимать, что условия

проверяются подряд, первое выполненное условие приводит к работе соответствующего блока, выходу из конструкции **if-elseif-else** и переходу к оператору, следующему за **end**. У последней ветви **else** не должно быть никакого условия. Операторы, находящиеся между **else** и **end**, работают в том случае, если все условия оказались невыполненными. Предположим, что требуется написать файл-функцию для вычисления кусочно-заданной функции:

$$f(x)=\begin{cases}1-e^{-1-x}, x<-1\\ x^2-x-2,-1\le x\le 2\\ 2-x, x>2\end{cases}$$

Первое условие **x<-1** проверяется в ветви **if**. Обратите внимание, что условие **–1<=x** не требуется включать в следующую ветвь **elseif** (см. ниже), поскольку в эту ветвь программа заходит, если предыдущее условие (**x<-1**) оказалось не выполнено. Условие **x>2** проверять не надо — если не выполнены два предыдущих условия, то **x** будет больше двух.

```
function f=pwf(x)
if x<-1
f=1-exp(-1-x);
elseif x<=2
f=x^2-x-2;
else
f=2-x;
end
```

Ход работы программы может определяться значением некоторой переменной (переключателя). Такой альтернативный способ ветвления программы основан на использовании оператора переключения **switch.** Переменная-переключатель помещается после **switch** через пробел. Оператор **switch** содержит блоки, начинающиеся со слова **case**, после каждого **case** записывается через пробел то значение переключателя, при котором выполняется данный блок. Последний блок начинается со слова **otherwise**, его

операторы работают в том случае, когда ни один из блоков **case** не был выполнен. Если хотя бы один из блоков **case** выполнен, то происходит выход из оператора **switch** и переход к оператору, следующему за end.

Предположим, что требуется найти количество единиц и минус единиц в заданном массиве и, кроме того, найти сумму всех элементов, отличных от единицы и минус единицы. Следует перебрать все элементы массива в цикле, причем в роли переменной-переключателя будет выступает текущий элемент массива. Пример ниже содержит файл-функцию, которая по заданному массиву возвращает число минус единиц в первом выходном аргументе, число единиц — во втором, а сумму — в третьем.

```
function [m,p,s]=mpsum(x)
m=0;
p=0;
s=0;
for i=1:length(x)
switch x(i)
case -1
m=m+1;
case 1
p=p+1;
otherwise
s=s+x(i);
end
end
```

Блок **case** может быть выполнен не только при одном определенном значении переключателя, но и в том случае, когда переключатель принимает одно из нескольких допустимых значений. В этом случае значения указываются после слова **case** в фигурных скобках через запятую, например: **case {1,2,3}**.

Досрочное завершение цикла **while** или **for** осуществляется при помощи оператора **break**. Пусть, например, требуется по заданному массиву **x** образовать новый массив **y** по правилу **y(k)=x(k+1)/x(k)** до первого нулевого элемента **x(k)**, т.е. до тех пор, пока имеет смысл операция деления. Номер первого нулевого элемента в массиве **x** заранее неизвестен, более того, в массиве **x** может и не быть нулей. Решение задачи состоит в последовательном вычислении элементов массива **y** и прекращении вычислений при обнаружении нулевого элемента в **x**. Файл-функция, приведенная ниже, демонстрирует работу оператора **break**.

```
function y=div(x)
for k=1:length(x)-1
if x(k)==0
break
end
y(k)=x(k+1)/x(k);
end
```

8. Optimization toolbox

В состав MATLAB входит ToolBox Optimization, предназначенный для решения линейных и нелинейных оптимизационных задач. Функции этого ToolBox реализуют основные алгоритмы оптимизации, причем понимание алгоритма позволяет пользователю настроить нужную функцию на эффективное решение поставленной задачи.

Линейное программирование

Задача линейного программирования состоит в нахождении вектора **x**, который минимизирует целевую функцию $f^T x$**.**

Функция **Linprog** предназначена для решения задач линейного программирования. Варианты использования функции:

1. **x = linprog(f,A,b)** находит **min f'*x** при условии, что **A*x <= b**.

2. **x = linprog(f,A,b,Aeq,beq)** решает указанные выше задачу при условии дополнительного выполнения ограничений в виде равенств **Aeq*x = beq**. Если нет неравенств, то устанавливается **A=[]** и **b=[]**.

3. **x = linprog(f,A,b,Aeq,beq,lb,ub)** определяет набор нижних и верхних границ для проектируемых переменных **x**, так что решение всегда находится в диапазоне **lb <= x <= ub**. Если нет неравенств, то устанавливается **Aeq=[] и beq=[]**.

4. **x = linprog(f,A,b,Aeq,beq,lb,ub,x0)** устанавливает начальную точку как **x0**. Эта опция имеет место только для средне-масштабного алгоритма **(options.LargeScale** равна '**off**'). Принимаемый по умолчанию крупно-масштабный алгоритм игнорирует любую стартовую точку.

5. **x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)** проводит оптимизацию с определенными в структурной опции параметрами оптимизации.

6. **[x,fval] = linprog(...)** возвращает значение целевой функции **fun** как решение от **x: fval = f'*x**.

7. **[x,lambda,exitflag] = linprog(...)** возвращает значение **exitflag**, которое содержит описание выходных условий.

8. **[x,lambda,exitflag,output] = linprog(...)** возвращает структурный выход с информацией об оптимизации

9. **[x,fval,exitflag,output,lambda] = linprog(...)** возвращает структурную lambda, чьи поля включают в себя множители Лагранжа как решение от **x.**

Пример: Найти такое **x**, что является минимумом от

$$f(x) = -5x_1 - 4x_2 - 6x_3$$

при условии, что

$$x_1 - x_2 + x_3 \le 20$$

$$3x_1 + 2x_2 + 4x_3 \le 42$$

$$3x_1 + 2x_2 \le 30$$

$$0 \le x_1, 0 \le x_2, 0 \le x_3$$

```
>> f=[-5;-4;-6]
>> A=[1 -1  1 ; 3 2 4 3 2 0];
>> b=[20; 42; 30];
>> lb = zeros(3,1);
>> [x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb);

x =
    0.0000
15.0000
3.0000



lambda.ineqlin =
    0
    1.5000
0.5000
lambda.lower=
    1.0000
0
0
```

Ненулевые элементы векторов в полях **lambda** указывают на активные ограничения при решении. В нашем случае, второе и третье ограничения в виде неравенств (в **lambda.ineqlin**) и первое нижнее граничное ограничение (в **lambda.lower**) являются активными ограничениями (т.е. решение находится на ограничительных условиях).

Квадратичное программирование

В задачах квадратичного программирования целевая функция имеет вид:

$$\frac{1}{2}x^T Hx + f^T x$$

Функция **quadprod** предназначена для решения задач квадратичного программирования. Интерфейс **quadprod** практически не отличается от **linprog,** за исключением того, что первыми двумя входными параметрами являются массив **h** и вектор-столбец **f**, соответствующие матрице **H** и вектору **f** целевой функции. Вместо матриц и векторов неиспользуемых ограничений задаются пустые массивы.

Нелинейное программирование

ToolBox Optimization позволяет решать ряд оптимизационных задач, в которых к линейным ограничениям добавляются нелинейные.

Задача нелинейного программирования представляет собой поиск минимума нелинейной задачи

$$\min_x f(x)$$

с ограничениями:

$$c(x) < 0;$$
$$ceq(x) = 0;$$
$$A * x \le b;$$
$$Aeq * x = beq;$$
$$lb < x < ub$$

при условии, что где **x, b, beq**, **lb** и **ub** - векторы, **A** и **Aeq** - матрицы, и **c(x)** и **ceq(x)** есть функции, **f(x)** - функция, которая возвращает скаляр. **f(x)**, **c(x)** и **ceq(x)** могут быть нелинейными функциями.

Функция **fmincon** находит минимум для скалярной функции нескольких переменных с ограничениями начиная с начального приближения. В общем случае, эта задача относится к нелинейной оптимизации с ограничениями или к нелинейному программированию. Варианты использования функции:

1. **x = fmincon(fun,x0,A,b)** начинает с точки **x0** и находит минимум от **x** для функции представленной как **fun** при условии выполнения линейных неравенств **A*x <= b. x0** может быть скаляром, вектором или матрицей.

2. **x = fmincon(fun,x0,A,b,Aeq,beq)** минимизирует **fun** при условии выполнения линейных равенств **Aeq*x = beq**, а так же **A*x <= b.** Устанавливается **A=[] и b=[]** в случае отсутствия неравенств.

3. **x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)** определяет набор нижних и верхних ограничений на конструируемые переменные **x** так, что решение всегда находится в диапазоне **lb <= x <= ub**. Устанавливается **Aeq=[] and beq=[]** в случае отсутствия равенств.

4. **x=fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)** подчиняет минимизацию определенных в **nonlcon fmincon** нелинейных неравенств **c(x)** или равенств **ceq(x)** такому оптимуму, что **c(x) <= 0 и ceq(x) = 0.** Устанавливается **lb=[]** и/илиr **ub=[]** в случае отсутствия ограничений.

5. **x=mincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)** проводит минимизацию с оптимизационными параметрами, определенными в структурной опции.

6. **x=fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)** передает зависящие от типа задачи параметры непосредственно в функции **fun** и **nonlcon**. Передает пустые матрицы заменители для **A, b, Aeq, beq, lb, ub, nonlcon** и **options** в случае, если эти аргументы не являются необходимыми.

7. **[x,fval]=fmincon(...**) возвращает значение целевой функции **fun** как решение от **x.**

8. **[x,fval,exitflag]=fmincon(...)** возвращает значение **exitflag**, которое содержит описание выходных условий **fmincon**.

9. **[x,fval,exitflag,output]=fmincon(...)** возвращает структурный выход с информацией об оптимизации.

10. **[x,fval,exitflag,output,lambda]=fmincon(...)** возвращает структурную **lambda** с полями, содержащими множители Лагранжа в виде решения от **x.**

11.**[x,fval,exitflag,output,lambda,grad]=fmincon(...)** возвращает значение градиента от **fun** в виде решения от **x**.

12.**[x,fval,exitflag,output,lambda,grad,hessian]= fmincon(...)** возвращает значения матрицы Гессе от **fun** в виде решения от **x**.

Целочисленное линейное программирование

Для решения задачи целочисленного линейного программирования используется функция **bintprog**. Варианты использования функции:

1. **x = bintprog(f)** решает задачу целочисленного программирования **min f'*x**

2. **x = bintprog(f, A, b)** решает задачу целочисленного программирования **min f'*x** при условии, что **A*x <= b**

3. **x = bintprog(f, A, b, Aeq, beq)** решает предыдущую задачу при дополнительных условиях типа равенств **Aeq*x = beq**

4. **x = bintprog(f, A, b, Aeq, beq, x0)** - устанавливает начальную точку поиска в **x0**. Если точка **x0** находится в недопустимой области, то команда **bintprog** принимает произвольную начальную точку.

5. **x = bintprog(f, A, b, Aeq, Beq, x0, options) -** при оптимизации используется принимаемая по умолчанию опция из структуры **option**s, которую можно задать с помощью функции **optimset**.

6. **[x, fval] = bintprog(...) -** возвращает **fval** как значение целевой функции в точке **x.**

7. **[x,fval, exitflag] = bintprog(...) -** возвращает параметр **exitflag** с описанием выходных условий команды **bintprog**.

8. **[x, fval, exitflag, output] = bintprog(...)** возвращает структуру **output**, которая содержит информацию о данных результатах оптимизации.

Выходные параметры **exitflag** и **output** имеет ряд особенностей:

1. **exitflag** - некое целое число, идентифицирующее причину остановки алгоритма. Возможны следующие значения параметра:

- **1** - Функция сошлась к некому решению x.
- **0** - Число итераций превысило значение options.MaxIter.

- **-2** - Данная задача не имеет решения.
- **-4** - Число перебранных узлов превышает значение options.MaxNodes.
- **-5** - Время перебора превышает значение options.MaxTime.
- **-6** - Число итераций решателя LP для некого узла при решении задачи LP-релаксации превысило значение options.MaxRLP.

2. **output** - Структура с информацией о результатах оптимизации. Поле данной структуры имеет вид.

- Iterations - Число выполненных итераций.
- Nodes - Число узлов перебора.
- Time - Превышение времени работы алгоритма.
- Algorithm - Используемый алгоритм.
- Message- Причина остановки работы алгоритма.

9. Statistics toolbox

Расчет функции распределения плотности вероятности

Функция распределения плотности вероятности дискретного закона представляет собой ряд распределения, ставящий в соответствие значения дискретной случайной величины и вероятности появления этих значений. Например, для биномиального закона с параметрами: вероятностью удачного исхода при одном опыте 0,5 и числом испытаний 10, ряд распределения **P** для значений случайной величины **k**, числа удачных исходов в серии опытов, примет следующий вид:

```
>> p=0.5;
>> n=5;
>> k=0:1:n;
>> P=binopdf(k,n,p);
>> [k' P']

ans =
```

0 0.0313

1.0000 0.1562

2.0000 0.3125

3.0000 0.3125

4.0000 0.1562

5.0000 0.0313

Значения функций распределения плотности вероятности для нормального, равномерного, экспоненциального и бета законов рассчитываются следующим образом:

```
>>x=0:0.01:2;
>>f1=normpdf(x,1,1/3);
>>f2=betapdf(x,2,3);
>>f3=exppdf(x,1);
>>f4=unifpdf(x,0.5,1.5);
>> plot(x,f1,'r-',x,f2,'b.-',x,f3,'g--',x,f4,'m--')
```

Расчет функции распределения

Функция распределения вероятностей, или интегральный закон распределения вероятностей, показывает вероятность попадания значений случайной величины **X** в интервал $(-\infty; x]$:

$F(x) = P(X \le x) = \int_{-\infty}^{x} f(t)dt$, где **f** - функция распределения плотности вероятности, **X** - одномерная случайная величина, ***x*** - граничное значение одномерной случайной величины, **F** - интегральный закон распределения случайной величины.

Для биномиального закона с параметрами: вероятностью удачного исхода при одном опыте 0,5 и числом испытаний 10, функция распределения **P** для значений случайной величины $(-\infty; k]$, числа удачных исходов в серии опытов, примет следующий вид:

```
>>p=0.5;
>>n=10;
>>k=0:1:n;
>>P=binocdf(k,n,p);
>>[k'P']
ans =
        0         0.0010
   1.0000         0.0107
   2.0000         0.0547
   3.0000         0.1719
   4.0000         0.3770
   5.0000         0.6230
   6.0000         0.8281
   7.0000         0.9453
   8.0000         0.9893
   9.0000         0.9990
  10.0000         1.0000
```

Значения функций распределения вероятности для нормального, равномерного, экспоненциального и бета законов вычисляются следующим образом:

```
>>x=0:0.01:2;
>>f1=normcdf(x,1,1/3);
>>f2=betacdf(x,2,3);
>>f3=expcdf(x,1);
>>f4=unifcdf(x,0.5,1.5);
```

10. Обмен данными между MATLAB и Excel.

Среда MATLAB допускает достаточно простое интегрирование с Ms Word и Ms Excel. Редактор Word используется для написания интерактивных документов, так называемых М-книг, которые позволяют наглядно оформить расчеты в MATLAB в документе Word. Обработка данных существенно облегчается при сочетании работы в MATLAB и Excel.

Excel Link Позволяет использовать Microsoft Excel, как процессор ввода-вывода MATLAB. Для этого достаточно установить в Excel как **add-in** функцию поставляемый Math Works файл **excllinkxla**. В Excel нужно набрать Сервис > Надстройки > Обзор, выбрать файл в каталоге \MATLABrl2\toolbox\exlink и установить его. Теперь при каждом запуске Excel появится командное окно MATLAB, а панель управления Excel дополнится кнопками **getmatrix**, **putmatrix**, **evalstring**. Для закрытия MATLAB из Excel достаточно набрать **=MLC1ose()** в любой ячейке Excel. Для открытия после выполнения этой команды нужно либо щелкнуть мышью на одной из кнопок **getmatrix, putmatrix, evalstring**, либо набрать в Excel Сервис > Макрос >Выполнить **mat! abi ni t**. Выделив мышью диапазон ячеек Excel, вы можете щелкнуть на **getmatrix** и набрать имя переменной MATLAB. Матрица появится в Excel. Заполнив числами диапазон ячеек Excel, вы можете выделить этот диапазон, щелкнуть на **putmatrix** и ввести имя переменной MATLAB. Работа, таким образом, интуитивно понятна. В отличие от MATLAB Excel Link не чувствителен к регистру: **I** и **i, J** и **j** равноценны.

Функции Excel Link:

1. **MLPutMatrix** - служит для помещения данных из ячеек листа Excel в массив рабочей среды MATLAB;

2. **MlEvalString** - при помощи этой функции производится обращение из Excel к командам MATLAB;

3. **MlDeleteMatrix** – при помощи этой функции производится удаление массива;

4. **MlPutVar** - предназначена для экспорта значения переменной в рабочую среду MATLAB;

5. **MLGetVar** –осуществляет импорт данных из рабочей среды в переменную процедуры;

6. **MATLABinit, MLautoStart, MlClose,MLOpen** - обеспечивают согласованную работу MATLAB и Excel. Подробная информация о данных функциях содержится в справочной системе по Excel Link.

11. Литература

1. Иглин С.П. Математические расчеты на базе Matlab. - СПб.: БХВ-Петербург. 2007 - 640 с.

2. Алексеев Е.Р., Чеснокова О.В. MATLAB 7 – М.: НТ-Пресс. 2006 - 464 с.

3. http://www.matclub.ru

4. http://www.radiomaster.ru/cad/matlab/index.php

5. http://www.matlab.exponenta.ru

Розенцвайг А.К., Шарипов Р.Ш.

Имитационное моделирование экономических процессов.
Основные приемы работы в среде MATLAB

Учебно-методическое пособие

Подписано в печать 22.04.2019.
Формат 60x84/16. Печать ризографическая.
Бумага офсетная. Гарнитура «Times New Roman».
Усл.п.л. 3 Уч.-изд. л. 2.88
Тираж 100 экз. Заказ № 1249

Отпечатано в Издательско-полиграфическом центре
Набережночелнинского института
Казанского (Приволжского) федерального университета

423810, г. Набережные Челны, Новый город, пр.Мира, 68/19
тел./факс (8552) 39-65-99 e-mail: ic-nchi-kpfu@mail.ru