ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В КВАНТОВОЙ ФИЗИКЕ
НАЧАЛЬНОЕ ЗНАКОМСТВО С MATLAB

`demo` обзор возможностей MatLab
Справочная информация о любой функции вызывается командой `help <имя функции>`

Простейшие вычисления

Операторы

`+` сложение
`-` вычитание
`*` умножение
`/` деление
`^` возведение в степень
`.'` транспонирование матрицы
`'` транспонирование с комплексным сопряжением

Функции

Справка об элементарных функциях: `help elfun`
Справка о специальных функциях: `help specfun`
Справка о матричных функциях: `help elmat`

`abs` модуль
`sqrt` квадратный корень
`log` натуральный логарифм
`sin` синус
`cos` косинус
`tan` тангенс
`sinh` гиперболический синус
`fix` округление до ближайшего целого по направлению к нулю

Полезные константы

`pi` число пи
`i` мнимая единица
`j` мнимая единица
`eps` относительная точность вычислений с плавающей запятой
`realmin` наименьшее число с плавающей запятой, 2^{-1022}
`realmax` наибольшее число с плавающей запятой, $(2-\text{eps})2^{1023}$
`Inf` бесконечность (`1/0 = Inf`)
`NaN` not a number («не число»), например, `0/0 = NaN`

Пример. Вычислить выражение $e^{-2.5}(\ln 11.3)^{0.3} - \sqrt{\frac{\sin(2.45\pi)+\cos(3.78\pi)}{\text{tg}(3.3)}}$.

Решение.
```
exp(-2.5)*log(11.3)^0.3-sqrt((sin(2.45*pi)+cos(3.78*pi))...
/tan(3.3))
```
Результат:
```
ans = -3.2105
```
Троеточие `...` – символ продолжения строки; по умолчанию результат вычислений заносится в переменную с именем `ans`.

Переменные

Переменные образуются при помощи оператора присваивания `=`, например,
```
w = 3.54;
```

Строчные и прописные буквы различаются, например, переменные `t1` и `T1` будут двумя разными переменными. **Точка с запятой в конце строки** используется для подавления вывода результата вычислений на экран. **Комментарии** в MatLab вводятся командой `%`, например,
```
c = a + b % вычисление суммы векторов a и b
d = sum(a.*b) % вычисление скалярного произведения векторов a и b
```

Массивы

Вектор-столбцы и вектор-строки

```
a = [1.3; 5.4; 6.9] % вектор-столбец a
b = [7.1; 3.5; 8.2] % вектор-столбец b
c = a + b % сумма a и b

a = [1.3 5.4 6.9] % вектор-строка a
b = [7.1 3.5 8.2] % вектор-строка b
c = a + b % сумма a и b

size (c) % размер массива c
```

`d = sin(a)` – вычисление значений функции sin **для всех элементов массива a**

Обращение к элементам вектора

```
v = [1 2 6 4 8 34];
v(4) % обращение к четвертому элементу вектора v
v(2) = 100 % присвоение второму элементу нового значения
v1 = [5 4 3 2 1];
v2 = [v v1] % формирование нового вектора из двух векторов-строк
v(2:4) % обращение к элементам вектора v со второго по четвертый
v1(3:end) % обращение к элементам вектора v1 с третьего по последний
v3 = [v(2:4) v1(1:4)] % формирование нового вектора из частей v и v1
ind = [4 2 5];
v4 = v(ind) % формирование вектора при помощи индексации
```

Применение функций обработки данных к векторам

```
v = [5 2 4 3 1];
p = prod(v) % перемножение элементов вектора
s = sum(v) % сумма элементов вектора
l = length(v) % длина вектора
mean(v) % среднее арифметическое элементов вектора
% mean(v) = sum(v)/length(v)
M = max(v) % максимальный элемент вектора v
m = min(v) % минимальный элемент вектора v
[m, k] = min(v) % m – минимальный элемент v, k – его место в v
R = sort(v) % сортировка элементов v
[R, ind] = sort(v) % сортировка с выводом массива соответствия
```

Поэлементные операции с векторами

```
v1 = [2 -3 4 1];
v2 = [7 5 -6 9];
u = v1.*v2 % поэлементное умножение векторов одинаковой длины
p = v1.^2 % поэлементное возведение в степень вектора v1
P = v1.^v2
d = v1./v2
s = v1 + 4
q = v2/2
```

`Q = v1*v2` – умножение вектора на вектор, в данном случае **приведет к ошибке**, так как вектор-строки можно умножать лишь на вектор-столбцы соответствующей длины

```
V2 = v2' % транспонирование v2, получается вектор-столбец V2
Q = v1*V2 % теперь матричное умножение возможно
```

Построение таблицы значений функции

Пример. Построить таблицу значений функции $f(x) = \sin x \cos^2 3x$ на отрезке от 0 до 2π с шагом 0.01.

`x = [0 : 0.01 : 2*pi]` – создание вектор-строки, содержащей координаты заданных точек; операция вида `a:b:c` задает массив элементов от `a` до `c` с шагом `b`.

`y = sin(x).*cos(3*x).^2` – создание вектор-строки, содержащей значения функции f(x) в точках `x`. **Обратите внимание на поэлементные операции!**

Построение графиков функции одной переменной

Вывод отображения функции в виде графика состоит из следующих этапов:
1. Задание вектора значений аргумента x

```
x = [0 : 0.01 : 2*pi];
```

2. Вычисление вектора y значений функции y(x)

```
y = sin(x).*cos(3*x).^2;
```

3. Вызов команды `plot` для построения графика

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

Матрицы

Создание специальных матриц

`A = zeros(3,4)` – создание нулевой матрицы
`I = eye(4)` – создание единичной матрицы
`E = ones(2,6)` – создание матрицы из единиц
`R = rand(3)` – создание квадратной матрицы, заполненной случайными числами, равномерно распределенными между 0 и 1
`Q = rand(3,4)` – создание матрицы размером 3x4, заполненной случайными числами, распределенными по нормальному закону

Функция `diag` служит для выделения диагонали матрицы в вектор:
`d = diag(R)` – главная диагональ
`d1 = diag(R,-1)` – диагональ, расположенная на одну позицию вниз от главной
Функция `diag` также формирует диагональную матрицу из вектора:

```
D = diag(d)
D1 = diag(d,2)
```

Матрицы можно создавать непосредственно в командной строке:

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

Операции с матрицами

```
B = [4 3 -1; 2 7 0; -5 1 2]
E = ones(3)
A = B + E % сложение матриц
A = B - E % вычитание матриц
A = 3*B % умножение матрицы на число
A = B^2 % возведение матрицы в степень
A = B.^2 % возведение в степень каждого элемента матрицы B
```

Выделение части матрицы

```
B = rand(4)
P = B(2:3, 2:3)
P = B(2,:) % выделение второй строки
P = B(:,3) % выделение третьего столбца
P = B(2, 2:end) % элементы второй строки со второго по последний
```

Удаление строк и столбцов

```
P = B
size(P) % размер массива P
P(1,:) = [] % удаление первой строки
P(:,3) = [] % удаление третьего столбца
size(P) % размер массива P изменился
B(:, 2:3) % удаление столбцов со второго по третий
```

Визуализация матриц

Проанализируйте команды, используемые при построении матрицы G, и посмотрите на результат:

```
G = 2*eye(7)+diag(ones(1,6),1)+diag(ones(1,6),-1)
```

Портрет матрицы – маркерами отмечаются ненулевые элементы; внизу указывается общее число ненулевых элементов:

```
spy(G)
```

Графики функций двух переменных

Построение графика включает два предварительных этапа:
1. Разбиение области определения прямоугольной сеткой.
2. Вычисление значений функции в узлах сетки и запись их в матрицу.

Пример. Построить график функции $f(x,y) = x^2 + y^2$ в квадрате $[0,1]\times[0,1]$.

Генерация массива сетки на квадрате $[0,1]\times[0,1]$ с шагом 0.2:

```
[X,Y] = meshgrid(0:0.2:1, 0:0.2:1)
```

Вычисление значений функции в узлах сетки:

```
Z = X.^2 + Y.^2
```

Построение графика в виде каркасной поверхности:

```
mesh(X, Y, Z)
surf(X, Y, Z) % каркасная поверхность с заливкой
shading interp % плавная заливка
```

Для более точного построения графика можно выбрать меньший шаг сетки.

M-файлы

Самым удобным способом выполнения команд MatLab является использование M-файлов, в которых можно набирать команды, выполнять их все сразу или частями, сохранять в файле и использовать в дальнейшем. Для работы с M-файлами предназначен редактор M-файлов. Он запускается из меню: File→New→M-file.

Наберите в редакторе команды, приводящие к построению двух графиков в одном окне:

```
x = [0:0.1:7];
f = exp(-x);
subplot(1,2,1);
plot(x, f);
g = sin(x);
subplot(1,2,2);
plot(x,g);
```

Сохраните файл под именем, например, mydemo.m в каталоге, предложенном по умолчанию.
Для выполнения всех команд файла выберите пункт Run в меню.
Замените функцию `g` на `cos(x)`, сохраните файл и запустите все команды снова.

Типы M-файлов

M-файлы бывают двух типов: файл-программы (script M-files) и файл-функции (function M-files).
Файл-программа только что была создана для построения графика двух функций. Все переменные, объявленные в файл-программе, становятся доступными в рабочей среде после ее выполнения. Запуск файл-программы удобно выполнять из командной строки, при этом в качестве команды используется имя M-файла. Набрав в командной строке `mydemo`, можно вновь запустить созданный M-файл. После ввода команды `mydemo` MatLab выполняет следующие действия:
1. Проверяет, является ли введенная команда именем какой-либо из переменных, определенных в рабочей среде. Если это так, выводится значение этой переменной.
2. Если такой переменной нет, производится поиск введенной команды среди встроенных функций. Если введенная команда является встроенной функцией, происходит ее выполнение.
3. Если такой встроенной функции нет, то производится поиск M-файла с названием команды и расширением m. Поиск начинается с текущего каталога, если M-файл в нем не найден, просматриваются каталоги, установленные в путях поиска.

Установка путей поиска производится из меню: File→Set Path.

Файл-функции

Пример. Написать M-файл для вычисления значений функции $f(x) = e^{-x}\sqrt{\dfrac{x^2+1}{x^4+0.1}}$.

Решение. В редакторе M-файлов необходимо создать следующий файл:

```
function f = myfun(x)
f = exp(-x).*sqrt((x.^2+1)./(x.^4+0.1));
```

Здесь `myfun` – имя функции; x – входной аргумент; f – выходной аргумент – значение функции; поэлементные операции в выражении для функции использованы для того, чтобы можно было обращаться к функции с входным аргументом, являющимся вектором.

Далее файл следует сохранить под тем же именем `myfun`, что и имя функции (оно будет предложено по умолчанию).

Выполните далее в командной строке следующие команды:

```
x = [0:0.05:4]
y = myfun(x)
plot(x, y)
```

Пример. Написать М-файл для вычисления суммы и произведения двух квадратных матриц.
Решение. В этой задаче два входных и два выходных аргумента. М-файл будет, например, таким:

```
function [S, P] = sum_prod(A,B)
% A,B – входные матрицы
% S – сумма матриц
% P – произведение матриц
S = A + B;
P = A*B;
```

Сохраните файл под именем `sum_prod.m` и выполните далее в командной строке следующие команды:

```
I = eye(5)
E = ones(5)
[A1,A2] = sum_prod(I,E)
```

Операторы цикла

Цикл for

```
for count = start:step:final
       команды MatLab
end
```

Здесь `count` – переменная цикла, `start` – ее начальное значение, `final` – конечное значение, `step` – шаг, на который увеличивается `count` при каждом следующем заходе в цикл. Цикл заканчивается, как только значение `count` становится больше `final`. Переменная цикла может принимать вещественные значения любого знака.

Цикл while

```
while условие цикла
       команды MatLab
end
```

Цикл выполняется до тех пор, пока выполняется `условие цикла`. Для задания условий используются операции отношения и логические операторы:

`==`	равно
`<`	меньше
`>`	больше
`<=`	меньше или равно
`>=`	больше или равно
`~=`	не равно
`&`	логическое «И»
`|`	логическое «ИЛИ»
`~`	отрицание

Например, условие (x<3 и k=4) запишется так: `(x<3)&(k==4)`.

Прерывание цикла осуществляется оператором `break`, при этом происходит выполнение операторов, следующий за оператором `end`, обозначающим окончание текущего цикла. В случае вложенных циклов `break` осуществляет выход из внутреннего цикла.

Оператор ветвления if

```
if условие
       команды MatLab
end
```

Если условие верно, выполняются команды между `if` и `end`, если условие неверно, происходит переход к командам, расположенным после `end`.

Более общая форма оператора if:

```
if условие
       команды MatLab
elseif условие
       команды MatLab
else
       команды MatLab
end
```