

Операторы языка MATLAB

Формат оператора	Выполнение
<code>var=expr</code>	Оператор присваивания
<pre> if условие_1 операторы_1 elseif условие_2 операторы_2 elseif условие_3 операторы_3 else операторы end </pre>	Условный оператор. Если справедливо условие_1, то выполняются операторы_1, Если справедливо условие_2, то выполняются операторы_2, ... Если все указанные условия являются ложными, то выполняются операторы, расположенные между else и end .
<pre> switch expr case val1 операторы_1 case val2 операторы_2 otherwise операторы end </pre>	Переключатель по значению выражения expr .
<pre> for var=e1:e2:e3 операторы end </pre>	Цикл типа арифметической прогрессии, в котором переменная var при каждом повторении тела цикла изменяется от начального значения e1 с шагом e2 до конечного значения e3 .
<pre> while условие операторы end </pre>	Цикл с предусловием, выполняется до тех пор, пока истинно указанное условие.
<pre> try операторы_1 catch операторы_2 end </pre>	Попытка выполнить группу операторы_1. При условии, что в результате их выполнения возникает исключительная ситуация, управление передается группе операторы_2. Если ошибка не возникла, то группа операторы_2 не выполняется.
break	Досрочный выход из управляющих конструкций типа for , while , switch , try-catch
<pre> function [y1,y2,...]=f(x1,x2,...) return </pre>	Заголовок функции. x1,x2, ... - входные параметры, y1,y2, ... - выходные
	Досрочный выход из тела функции

- Условный оператор **if**. Варианты использования:

if <условие> <команды> end	if <условие> <команды> else <команды> end	if <условие> <команды> elseif <условие> <команды> else <команды> end	
--	--	--	--

«команды» выполняются, только если «условие» – верно.

Пример:

```
if a < 0
disp('a is negative');
end
```

При записи в одну строку после «if expression» используется «,»:

```
if a < 0, disp('a is negative'); end
```

if... else

При конструкциях **if... else** и **if... elseif** допускается множественный выбор:

```
if x < 0
    error('x is negative; sqrt(x) is imaginary');
else
    r = sqrt(x);
end
```

- **Конструкция switch**

Конструкции с переключателем полезны в случаях, когда тестовая переменная может принимать дискретные значения, представляющие собой целые числа или строковые переменные.

Синтаксис:

```
switch expression
    case value1,
        block of statements
    case value2,
        block of statements
    ...
    otherwise,
        block of statements
end
```

Пример 1:

```
color = '...'; % строковая переменная
switch color
    case 'red'
        disp('Color is red');
    case 'blue'
        disp('Color is blue');
    case 'green'
        disp('Color is green');
    otherwise
        disp('Color is not red, blue, or green');
end
```

Пример 2: % Ввод и вывод, оператор switch

```
input_num=input('Введите значение, -1,0,1 или другое ')
switch input_num
    case -1
        disp('минус один')
    case 0
        disp('ноль')
    case 1
        disp('плюс один')
    otherwise
        disp('другое значение')
end
```

- **Циклы**

Последовательность команд повторяется до тех пор, пока не будет выполнено *одно из условий*:

1. Обработаны все элементы вектора или матрицы;
2. Получен результат, который соответствует заданному критерию завершения.

Для циклов используются конструкции с `for` или `while`.

- **Циклы с оператором `for`**

Конструкции циклов с оператором **`for`** чаще всего используются в случаях, когда надо поэлементно обработать вектор или матрицу.

Синтаксис:

```
for index = expression
block of statements
end
```

Пример: *Сумма элементов вектора*

```
x = 1:5; % create a row vector
sumx = 0; % initialize the sum
for k = 1:length(x)
sumx = sumx + x(k);
end
```

Вариации цикла с оператором `for`

Пример: *цикл с шагом 2*

```
for k = 1:2:n
...
end
```

Пример: *цикл с отрицательным значением шага*

```
for k = n:-1:1
...
end
```

Пример: *цикл с нецелочисленным значением шага*

```
for x = 0:pi/15:pi
fprintf('%8.2f %8.5f\n', x, sin(x));
end
```

Пример: *% Формирование трехдиагональной матрицы*

```
n=4;
for i=1:n
    for j=1:n
        if i==j
            A(i,j)=2;
        elseif abs(i-j)==1
            A(i,j)=1;
        else
            A(i,j)=0;
        end
    end
end
A
```

- **Циклы с оператором `while`**

Циклы **`while`** чаще всего употребляются в случаях, когда операторы повторяются до тех пор, пока истинно указанное условие.

Синтаксис:

```
while expression
block of statements
end
```

«*block of statements*» выполняется до тех пор, пока истинно «*expression*».

Пример: обработка исключительных ситуаций

```
% Обработка ввода правильности имени открываемого файла
i=1;
while (i==1)
    try
        i=0;
        name_file=input('Введите имя файла (прерывание Ctrl+c): ','s');
        feval('type',name_file);
    catch
        disp('Файл не найден!!!');
        i=1;
    end
end
```

Замечание:

1. Функция **eval** выполняет команду, которую записывают в строку – аргумент функции **eval**.

Пример:

```
x=pi
y=eval('sin(x)+cos(x)')
x =    3.1416
y =   -1.0000
```

2. Имя функции можно записать в строку и выполнить с помощью программы feval()

Пример:

```
y=feval('cos',pi)
y =   -1
```

Пример: % Вычисление значений функции $\sin(x)$ через разложение в ряд Тейлора(1685-1731)
% (Маклорена(1698-1746))

$$f(x) = \sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

Рекуррентная формула:

$$r_n = r_{n-1} \cdot (-1) \cdot \frac{x^2}{(2n) \cdot (2n+1)}, \text{ где } r_0 = x.$$

Замечание: Радиус сходимости рассматриваемого ряда: $R = \infty$, так как $R = 1/g$, а

$$g = \lim_{n \rightarrow \infty} \left| \frac{r_n}{r_{n-1}} \right| = \lim_{n \rightarrow \infty} \left| \frac{x^{2n+1} \cdot (2n-1)!}{x^{2n-1} \cdot (2n+1)!} \right| = \lim_{n \rightarrow \infty} \left| \frac{x^2}{2n(2n+1)} \right| = 0$$

% Инициализация данных

```
x=input('Введите аргумент: x=');
eps=1e-6; r=x; s=x; n=0;
```

%цикл с предусловием

```
while (abs(r)>eps)
    n=n+1;
    r=r*(-1)*(x*x)/((2*n)*(2*n+1));
    s=s+r;
    fprintf(1,'n=%f\tr=%f\ts=%f\n',n,r,s);
end
fprintf(1,'sin(%f)=%f\n',x,sin(x));
```

Пример: Оценка \sqrt{x} методом Ньютона

Рекуррентная формула: $r_k = \frac{1}{2} \left(r_{k-1} + \frac{x}{r_{k-1}} \right)$.

```
r = ... % initialize
rold = ...
while abs(rold-r) > delta
```

```

        rold = r;
        r = 0.5*(rold + x/rold);
    end

```

Зачастую бывает полезно накладывать ограничения на количество выполняемых итераций в цикле с оператором **while**. Если рассмотреть предыдущий пример, то можно следующим образом улучшить код:

```

maxit = 25; %max number of iterations
it = 0;
while abs(rold-r) > delta & it<maxit
    rold = r;
    r = 0.5*(rold + x/rold);
    it = it + 1;
end

```

Выйти из цикла можно также с помощью команд **break** и **return**. Данные команды применимы как к циклам с **while**, так и к циклам с **for**.

Оператор **break** используется для выхода из цикла с **while** или с **for**. Выполнение продолжается с момента завершения цикла.

Оператор **return** используется для принудительного выхода из **функции**. При этом, любая последовательность команд, следующая за данным циклом в теле функции, игнорируются.

• Оператор **break**

Пример: Выход из цикла *while*

```

function k = breakDemo(n)
% Search a random vector to find index of the 1st element greater than 0.8.
%
% Вход: n = размер случайного вектора
%
% Выход: k = первый (наименьший) индекс в x, такой что x(k)>0.8
x = rand(1,n);
k = 1;
while k<=n
    if x(k)>0.8
        break
    end
    k = k + 1;
end
fprintf('x(k)=%f for k = %d n = %d\n',x(k),k,n);
% Что произойдёт, если цикл прервется, не найдя x(k)>0.8 ?

```

• Оператор **return**

Пример: Возврат из тела функции

```

function k = returnDemo(n)
% Search a random vector to find index of the 1st element greater than 0.8.
%
% Synopsis: k = returnDemo(n)
%
% Вход: n = размер случайного вектора
%
% Выход: k = первый (наименьший) индекс в x % такой что x(k)>0.8
x = rand(1,n);
k = 1;
while k<=n
    if x(k)>0.8
        return
    end
    k = k + 1;
end
% Что произойдёт, если цикл прервется, не найдя x(k)>0.8 ?

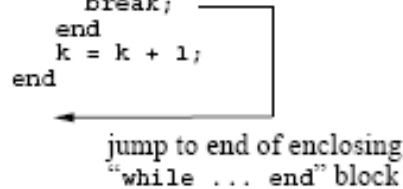
```

Сравнение операторов «break» и «return»

break используется для выхода из текущего цикла **while** или **for**.

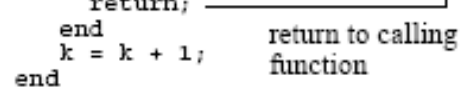
return используется для выхода из тела функции.

```
function k = demoBreak(n)
...
while k<=n
    if x(k)>0.8
        break;
    end
    k = k + 1;
end
    
```



jump to end of enclosing
"while ... end" block

```
function k = demoReturn(n)
...
while k<=n
    if x(k)>0.8
        return;
    end
    k = k + 1;
end
    
```



return to calling
function

Векторизация

Векторизация представляет собой использование векторных операций (выражений MATLAB) для обработки всех элементов вектора или матрицы. Правильно векторизованные выражения эквивалентны выполнению циклов по элементам соответствующих векторов или матриц. Векторизованное выражение является более компактным, при этом соответствующий код выполняется быстрее по сравнению с не векторизованным выражением.

Для векторизации кода:

- Там, где это применимо, используйте векторные операции вместо циклов;
- Предварительно выделяйте память для векторов и матриц.
- Используйте векторизованное индексирование и логические функции.

Замена циклов на векторные операции

Скалярный код

```
for k=1:length(x)
    y(k) = sin(x(k))
end
    
```

Векторизованный эквивалент

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

Выделение памяти

Следующий цикл увеличивает размер **s** при каждом проходе.

```
y = ... % some computation to define y
for j=1:length(y)
    if y(j)>0
        s(j) = sqrt(y(j));
    else
        s(j) = 0;
    end
end
    
```

Заранее выделите память, предопределив **s** перед тем, как присваивать значения элементам.

```
y = ... % some computation to define y
s = zeros(size(y));
for j=1:length(y)
    if y(j)>0
        s(j) = sqrt(y(j));
    end
end
    
```

Векторизованное индексирование и логические функции

Векторизация кода требует использования **индексирования массивов** и **логического индексирования**.

Индексирование массивов:

Используйте вектор или матрицу как «индекс» для другой матрицы:

```
>> x = sqrt(0:4:20)
x =
0 2.0000 2.8284 3.4641 4.0000 4.47210
>> i = [1 2 5];
>> y = x(i)
y =
0 2 4
```

Выражение **x(i)** выбирает элементы **x**, имеющие индексы из **i**. Выражение **y = x(i)** эквивалентно следующему:

```
k = 0;
for i = [1 2 5]
    k = k + 1;
    y(k) = x(i);
end
```

Логическое индексирование:

Используйте вектор или матрицу в качестве маски для выбора элементов другой матрицы:

```
>> x = sqrt(0:4:20)
x =
0 2.0000 2.8284 3.4641 4.0000 4.47210
>> j = find(rem(x,2)==0)
j =
1 2 5
>> z = x(j)
z =
0 2 4
```

Вектор **j** содержит те индексы из **x**, которые соответствуют элементам **x**, являющимся целыми числами.

Пример: Векторизация кода

Мы уже рассматривали пример с фрагментом кода, описывающим предварительное выделение памяти:

```
y = ... % some computation to define y
s = zeros(size(y));
for j=1:length(y)
    if y(j)>0
        s(j) = sqrt(y(j));
    end
end
```

На самом деле, цикл может быть полностью заменен на код, использующий логическое индексирование и индексирование массивов.

```
y = ... % some computation to define y
s = zeros(size(y));
i = find(y>0); % indices such that y(i)>0
s(y>0) = sqrt(y(y>0))
```

Можно еще больше сократить код:

```
y = ... % some computation to define y
s = zeros(size(y));
s(y>0) = sqrt(y(y>0))
```

Векторизованные операции копирования

Пример: Копирование строк (столбцов)

Скалярный код

```
[m,n] = size(A); % предполагаем что A и B(:,1) = A(:,1);
% B имеют одинаковое кол-во строк
for i=1:m
```

Векторизованный код

```
B(i,1) = A(i,1);
end
```

Пример: Копирование и транспонирование подматриц

Скалярный код

```
for j=2:3
    B(1,j) = A(j,3);
end
```

Векторизованный код

```
B(1,2:3) = A(2:3,3)'
```

Inline Function Objects

С введением элементов объектно-ориентированного программирования в MATLAB стало доступным использование встраиваемых функций, позволяющих большую гибкость программы.

Встраиваемая функция определяется с помощью функции **inline()** и представляет собой формулу вычисления выражения:

имя_функции = inline('формула', список_аргументов),

где формула – текстовая строка.

Пример:

```
fun=inline('sin(x)+cos(x)', 'x')
y=fun(pi)
```

Если во встраиваемой функции используется аргумент, который явно не описан в строке аргументов, но есть в рабочем пространстве, функция его не «видит».

```
x=5
```

```
y=10
```

```
% вариант с ошибкой
```

```
fun=inline('sin(x)+cos(y)', 'x')
```

```
z=fun(pi)
```

```
% Вариант без ошибки:
```

```
x=5
```

```
y=10
```

```
fun=inline('sin(x)+cos(y)', 'x', 'y')
```

```
z=fun(pi,2)
```

Пример: Вместо создания файла-функции

```
function y = myFun(x)
y = x.^2 - log(x);
```

можно использовать команду в файле-скрипте

```
myFun = inline('x.^2 - log(x)');
```

Оба определения myFun позволяют обращаться к ней:

```
z = myFun(3);
```

```
s = linspace(1,5);
```

```
t = myFun(s);
```

Но!!! inline лучше не использовать!

Анонимная функция использует указатель @:

имя_функции = @(список аргументов) формула

Анонимной функции доступны переменные рабочей среды, используемые в формуле. Но они являются константами, то есть имеют те значения, которые были при создании функции и далее не меняются.

Пример:

```
x=1
```

```
y=pi
```



```

c=5
fun=@(x) c+sin(x)+cos(y)
z=fun(pi)
c =10
z=fun(pi)

```

Использование команды **keyboard**

```

function r = quadroot(a,b,c)
% quadroot Roots of quadratic equation and demo of keyboard command
%
% Synopsis: r = quadroot(a,b,c)
%
% Input: a,b,c = coefficients of  $a*x^2 + b*x + c = 0$ 
%
% Output: r = column vector containing the real or complex roots
d = b^2 - 4*a*c;
if d<0
    fprintf('Warning in function QUADROOT:\n');
    fprintf('\tNegative discriminant\n\tType "return" to continue\n');
    keyboard;
end
q = -0.5*( b + sign(b)*sqrt(b^2 - 4*a*c) );
r = [q/a; c/q]; % store roots in a column vector

```

Техника рационального программирования в MATLAB

Пример 1:

```

clear all
tic
x=0:2*pi/10000:2*pi;
y=exp(-x.^2).*cos(x);
toc
tic
for i=1:10001
    c(i)=2*pi/10000*(i-1);
    d(i)=exp(-c(i)^2)*cos(c(i));
end
toc

```

Работа этой программы состоит из двух частей. Первая часть начинается с вызова функции `tic` – таймера, затем вычисляются значения двух массивов `x` и `y`, и вызывается функция `toc` – вывести на экран время в секундах, прошедшее с момента включения таймера.

Вторая часть функции начинается вновь с включения таймера. Затем в цикле `for` вычисляются значения массивов `c` и `d`. Содержимое массивов `c` и `d` полностью совпадает с содержимым массивов `x` и `y`. Обе части функции делают практически одно и тоже, но используя разные операторы.

Пример 2:

```

clear all
tic
x=rand(100000,1);
s=sum(x);
toc
tic
s=0;
for i=1:100000
    c(i)=rand(1);
    s=s+c(i);
end
toc

```

Задание:

Нахождение выборочного мат. ожидания, стандартного отклонения, коэффициента корреляции, значений однофакторной линейной регрессии. Данные хранятся в файле **gr10m.txt**, результаты выполнения программы записываются в файл **rez10m.txt**

Теория:

$$\begin{aligned}M(x) &= \frac{1}{n} \sum_{i=1}^n x_i, & M(y) &= \frac{1}{n} \sum_{i=1}^n y_i. \\D(x) &= \frac{1}{n} \sum_{i=1}^n (x_i - M(x))^2, & D(y) &= \frac{1}{n} \sum_{i=1}^n (y_i - M(y))^2 \\S(x) &= \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - M(x))^2}, & S(y) &= \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - M(y))^2}. \\R(x, y) &= \frac{\sum_{i=1}^n (x_i - M(x))(y_i - M(y))}{n \cdot S(x) \cdot S(y)} \\y_r &= M(y) + \frac{S(y)}{S(x)} \cdot R \cdot (x - M(x))\end{aligned}$$

В текущую директорию загрузим файл gr10m.txt с анализируемыми данными (если файл отсутствует, создадим его):

Программа нахождения статистических характеристик

Нахождение зависимости веса от роста

10

178 75

182 80

186 79

171 70

184 77

172 72

178 80

183 79

185 84

180 80

```
function regr()
% Программа нахождения статистических характеристик двух случайных величин:
% роста и веса. Программа находит математические ожидания, дисперсии,
% коэффициент корреляции и линейную регрессию - зависимость веса от роста

format short

fid=fopen('gr10m.txt','r+');

% считайте данные их файла с помощью функции fscanf
% в переменную n занесите количество объектов, в массив D - данные о росте и
весе. Далее из массива D в вектор-строку x передайте данные о росте, а в
вектор-строку y - данные о весе. Выведите на экран эти данные с заголовком
«Данные: рост и вес»
n=
D=
fclose(fid);
x=
y=
disp('Данные: рост и вес');
```

```

% Математические ожидания, дисперсии, ст. отклонения: Mx,My, Dx,Dy, Sx,Sy
Mx=
My=
Dx=
Dy=
Sx=
Sy=

% Коэффициент корреляции
r=

% Нахождение уравнения прямой  $a_0 + a_1 \cdot x$  в случае если значение r не меньше 0.5

% Вывод значений Mx,My,Dx,Dy,r в командное окно с 2 знаками после запятой
fprintf

% Вывод значений a0, a1 с 2 знаками после запятой

    yr=a0+a1*x;
% обновить массив D, добавив значения yr
    D=
% Вывести в командное окно таблицу результатов, значения с 2 десятич. знаками
    disp('Таблица результатов: [рост; вес; расчетный вес]');

% построить график пар (x,y) и (x,yr) с помощью функции plot

% Вывод результатов в файл rez10m.txt
% первая строка – заголовок «Программа нахождения зависимости веса от роста»
% вторая строка – заголовок «Статистические характеристики:»
% третья строка– результаты с двумя десятичными знаками
% 4-я строка – заголовок «Коэффициенты уравнения прямой:»
% 5-я строка – значения коэффициентов с двумя десятичными знаками
% 6-я строка – заголовок «Результаты (рост вес расчетный вес)»
% далее идет таблица с результатами, 2 знака после запятой, разделитель –
пробел
% в случае значения коэффициента корреляции < 0.5 первые три строки те же, в 4-
й строке сообщение «Коэффициент корреляции < 0.5»

```

Результаты программы, выведенные на экран

Программа нахождения статистических характеристик
Нахождение зависимости веса от роста

Данные: рост и вес

178	182	186	171	184	172	178	183	185	180
75	80	79	70	77	72	80	79	84	80

$M_x=179.90$

$M_y=77.60$

$D_x=242.90$

$D_y=158.40$

$r=0.82$

$a_0=-41.35$

$a_1=0.66$

Таблица результатов: [рост; вес; расчетный вес]

178.00	75.00	76.34
182.00	80.00	78.99
186.00	79.00	81.63
171.00	70.00	71.72
184.00	77.00	80.31
172.00	72.00	72.38
178.00	80.00	76.34
183.00	79.00	79.65
185.00	84.00	80.97
180.00	80.00	77.67

Результаты программы, выведенные в файл rez10m.txt

Просмотр файла:

type rez10m.txt

Программа нахождения зависимости веса от роста

Статистические характеристики:

$M_x=179.90$ $M_y=77.60$ $D_x=242.90$ $D_y=158.40$ $r=0.82$

Уравнение прямой:

$a_0=-41.35$ $a_1=0.66$

Результаты:

178.00	75.00	76.34
182.00	80.00	78.99
186.00	79.00	81.63
171.00	70.00	71.72
184.00	77.00	80.31
172.00	72.00	72.38
178.00	80.00	76.34
183.00	79.00	79.65
185.00	84.00	80.97
180.00	80.00	77.67