

Table of Contents

Семинар 2. Арифметические операции над массивами и инструменты языка для управления потоком выполнения программы	1
Арифметика	1
Контроль потока выполнения.....	4
if ...elseif...else...end.....	4
short-circuit логические операции.....	5
switch...case {выбор 1}.{действие1}.case {выбор 2} {действие2} otherwise.{действие }..end.....	6
try.{код, который может дать ошибку}..catch Exception{переменная хранящая объект ошибки}.....{код, который выполнится, если в коде блока try есть ошибка}...end.....	6
Циклы.....	7
for...break...continue...end.....	7
while...break...continue...end.....	9
MATLAB - COLUMN ORIENTED LANGUAGE -> в памяти при хранении матрицы A, ее элемент A(i+1,j) находится ближе к A(i,j), чем A(i,j+1) => алгоритмы перебирают элементы матрицы по колонкам	10
Выводы по семинару 2.....	11

Семинар 2. Арифметические операции над массивами и инструменты языка для управления потоком выполнения программы

- Арифметика
- if-else
- Циклы

Арифметика

Арифметику удобней показывать на символьных переменных (symbolic toolbox), символьные массивы ведут себя также как численные

```
clearvars
type = "num"
```

```
type =
"num"
```

```
switch type
case "sym"
    a_col = sym('a',[3 1])
    b_row = sym('b',[1 4])
    M3x4 = sym('m',[3,4])
    W3x4 = sym('w',[3,4])
    B3x3 = sym('B',[3 3])
case "num"
    a_col = rand([3 1])
    b_row = rand([1 4])
    M3x4 = rand([3,4])
```

```
W3x4 = rand([3,4])
B3x3 = rand([3 3])
```

```
end
```

```
a_col = 3x1
    0.0083
    0.7259
    0.9886
b_row = 1x4
    0.1270    0.7694    0.0668    0.6352
M3x4 = 3x4
    0.0207    0.9955    0.8906    0.2522
    0.6898    0.1641    0.1856    0.9887
    0.5465    0.3847    0.7711    0.4701
W3x4 = 3x4
    0.2228    0.7552    0.4181    0.8048
    0.6079    0.2459    0.3249    0.3531
    0.7395    0.1896    0.7678    0.6114
B3x3 = 3x3
    0.1038    0.6444    0.4961
    0.5243    0.7566    0.9722
    0.4189    0.6723    0.5128
```

```
% СЛОЖЕНИЕ
```

```
a_col + a_col
```

```
ans = 3x1
    0.0167
    1.4518
    1.9772
```

```
plus(a_col,a_col)
```

```
ans = 3x1
    0.0167
    1.4518
    1.9772
```

```
a_col + b_row
```

```
ans = 3x4
    0.1354    0.7778    0.0751    0.6436
    0.8529    1.4953    0.7927    1.3611
    1.1156    1.7580    1.0554    1.6238
```

```
a_col.^b_row
```

```
ans = 3x4
    0.5444    0.0251    0.7262    0.0478
    0.9601    0.7815    0.9788    0.8159
    0.9985    0.9912    0.9992    0.9927
```

```
M3x4 + W3x4
```

```
ans = 3x4
    0.2435    1.7507    1.3087    1.0570
    1.2977    0.4100    0.5106    1.3418
    1.2860    0.5743    1.5389    1.0815
```

```
M3x4 + 3
```

```
ans = 3x4
    3.0207    3.9955    3.8906    3.2522
    3.6898    3.1641    3.1856    3.9887
    3.5465    3.3847    3.7711    3.4701
```

M3x4 + a_col

```
ans = 3x4
    0.0291    1.0038    0.8989    0.2605
    1.4157    0.8899    0.9115    1.7146
    1.5351    1.3733    1.7597    1.4587
```

M3x4 + b_row

```
ans = 3x4
    0.1478    1.7649    0.9574    0.8874
    0.8168    0.9335    0.2525    1.6240
    0.6735    1.1541    0.8379    1.1053
```

% УМНОЖЕНИЕ

2*M3x4

```
ans = 3x4
    0.0415    1.9910    1.7812    0.5044
    1.3796    0.3281    0.3713    1.9774
    1.0929    0.7694    1.5422    0.9401
```

M3x4*transpose(b_row) % ' - транспонирование матрицы

```
ans = 3x1
    0.9883
    0.8543
    0.7155
```

M3x4.*a_col

```
ans = 3x4
    0.0002    0.0083    0.0074    0.0021
    0.5007    0.1191    0.1348    0.7177
    0.5402    0.3803    0.7623    0.4647
```

B3x3^2

```
ans = 3x3
    0.5564    0.8879    0.9323
    0.8584    1.5639    1.4942
    0.6108    1.1234    1.1244
```

B3x3.^2

```
ans = 3x3
    0.0108    0.4152    0.2461
    0.2749    0.5725    0.9451
    0.1755    0.4520    0.2629
```

M3x4.^2 % поэлементные операции

```
ans = 3x4
    0.0004    0.9910    0.7931    0.0636
    0.4759    0.0269    0.0345    0.9776
    0.2986    0.1480    0.5946    0.2210
```

```
% ПРАВОЕ И ЛЕВОЕ ДЕЛЕНИЕ
```

```
mldivide(M3x4,a_col) % левое деление решает  $Ax = B \Rightarrow x = A \backslash B$ 
```

```
ans = 4x1
      0
    -1.6898
     1.7013
     0.6951
```

```
B3x3 \ a_col
```

```
ans = 3x1
     3.1326
     0.5846
    -1.3978
```

```
% mrdivide(M3x4,transpose(a_col)) % правое деление решает  $xA = B \Rightarrow x = A/B$ 
```

Операции над строками

```
clearvars
s_string = "string1"
```

```
s_string =
"string1"
```

```
"str" + s_string
```

```
ans =
"strstring1"
```

Контроль потока выполнения

if ...elseif...else...end

Ветвление кода

```
clearvars
a = rand()
```

```
a = 0.1863
```

```
if a>=0.9
    disp("a>=0.9")
elseif a>=0.5
    disp("0.5<=a<0.9")
elseif a>=0.2
    disp("0.2<=a<0.5")
else
    disp("a<0.2")
end
```

```
a<0.2
```

short-circuit логические операции

| и & - логическое сложение и умножение ведут себя также как + и .*, только для логических массивов

операторы || и && - могут работать только со скалярными логическими выражениями, ими можно пользоваться как if-else, но при этом в одну строчку

выполнение этого кода происходит последовательно, причем, компилятор в состоянии понять, что условие нарушено и выполнить только часть инструкций

Для примера рассмотрим условие удовлетворения одновременно большому числу условий:

(условие-1)&&(условие-2)&&(условие-3)... и т.д. - все эти условия вместе выполняются только если выполнены одновременно все условия

```
%ПРИМЕР - блок кода, который сравнивает случайную матрицу с числом, при этом  
%если в качестве аргумента задается не число, он выдает сообщение  
clearvars  
a="a"
```

```
a =  
"a"
```

```
try % ловим ошибку (об этой конструкции - позже)  
    rand(1000)>a  
catch ex  
    disp(ex.message)% нельзя сравнивать тип строка с числовым типом  
end
```

Comparison between double and string is not supported.

```
% Запись при помощи  
if ~ischar(a)&&~isstring(a)&&isscalar(a)&&all(rand(1000)>a,'all')  
    disp("a меньше нуля с вероятностью больше, чем "+ 1- 1/1000 + "!")  
else  
    disp("Неподходящий тип входного аргумента")  
end
```

Неподходящий тип входного аргумента

```
% Это эквивалентно записи через стандартный if-else:  
if ~ischar(a)  
    if ~isstring(a)  
        if isscalar(a)  
            if all(rand(1000)>a)  
                disp("a меньше нуля с вероятностью больше, чем "+ 1- 1/1000  
+ "!")  
            else  
                disp("Неподходящий тип входного аргумента")  
            end  
        else  
            disp("Неподходящий тип входного аргумента")  
        end  
    end  
else  
    disp("Неподходящий тип входного аргумента")  
end
```

```

        disp("Неподходящий тип входного аргумента")
    end
else
    disp("Неподходящий тип входного аргумента")
end
else
    disp("Неподходящий тип входного аргумента")
end
end

```

Неподходящий тип входного аргумента

```
%
```

**switch...case {выбор 1}.{действие1}.case {выбор 2} {действие2} otherwise.
{действие }..end**

```

clearvars
a = '2C00'; % UNICODE символ для Азь
index = randi(4,1) -1;
% char({число}) - преобразует номер из таблиц unicode в символ
ag = char(hex2dec(a) + index); % Смещаем на один индекс в таблице юникод
str = string(ag);
switch index
    case 0
        disp(str + " Азь")
    case 1
        disp(str + " Буки")
    case 2
        disp(str + " Веди")
    otherwise
        disp(str + " Глаголи")
end
end

```

Веди

try.{код, который может дать ошибку}..catch Exception{переменная хранящая объект ошибки}.....{код, который выполнется, если в коде блока try есть ошибка}...end

```

A = rand(5);
B = rand(6);
try
    A*B % код, который пытаемся выполнить
catch Ex
    disp(Ex.stack)
    disp("Сообщение ошибки:"+Ex.message)
    %rethrow(Ex) % перебрасывает ошибку выше по call-stack

```

```
end
```

```
file: 'C:\Users\user\AppData\Local\Temp\Editor_jjvhy\LiveEditorEvaluationHelperE1812138283.m'  
name: 'LiveEditorEvaluationHelperE1812138283'  
line: 106
```

Сообщение ошибки:Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix equals the number of rows in the second matrix.

Циклы

for...break...continue...end

```
clearvars  
for iii=5:-1:1  
    disp(iii)  
end
```

```
5  
4  
3  
2  
1
```

```
% break  
for iii=5:-1:1  
    disp(iii)  
    if iii==2  
        break % прерывает выполнение цикла  
    end  
end
```

```
5  
4  
3  
2
```

```
% continue  
for iii=5:-1:1  
    if iii>2  
        continue % переходит на следующую итерацию, не выполняя инструкции ниже  
    end  
    disp(iii)  
end
```

```
2  
1
```

Итерирование по элементам массива

% итерирование по коллекции

```
clearvars
```

```
%A = 1:5
```

```
A = rand(5,1)
```

```
A = 5x1
```

```
0.4144
```

```
0.5725
```

```
0.4519
```

```
0.7189
```

```
0.6546
```

```
%A = rand(5)
```

```
i=0;
```

```
for a = A
```

```
    i = i+1;
```

```
    disp("size(a) = " + join(string(size(a))," x "))
```

```
    disp("i="+ i + " a = " + join(string(a)))
```

```
end
```

```
size(a) = 5 x 1
```

```
i=1 a = 0.41445 0.57247 0.45186 0.71889 0.65458
```

```
for a = transpose(A)
```

```
    i = i+1;
```

```
    disp("size(a) = " + join(string(size(a))," x "))
```

```
    disp("i="+ i + " a = " + join(string(a)))
```

```
end
```

```
size(a) = 1 x 1
```

```
i=2 a = 0.41445
```

```
size(a) = 1 x 1
```

```
i=3 a = 0.57247
```

```
size(a) = 1 x 1
```

```
i=4 a = 0.45186
```

```
size(a) = 1 x 1
```

```
i=5 a = 0.71889
```

```
size(a) = 1 x 1
```

```
i=6 a = 0.65458
```

BATTLE† Итерирование с индексирование VS итерирование по коллекции:

% **timeit** - замеряет время вызова (вызывает много раз и усредняет)

```
clearvars
```

```
disp("Итерирование по индексам типа for i=1:numel(A) :")
```

Итерирование по индексам типа for i=1:numel(A) :


```
timeit(@indexwise_iter) % итерирование по индексам
```

```
ans = 4.8166e-04
```

```
disp("Итерирование по коллекции типа for a=A :")
```

```
Итерирование по коллекции типа for a=A :
```

```
timeit(@elementwise_iter) % итерирование по элементам
```

```
ans = 5.1722e-04
```

while...break...continue...end

Если нужно что-то делать до выполнения какого-то условия

```
clearvars  
iii=0
```

```
iii = 0
```

```
while true  
    iii=iii+1;  
    disp(iii)  
    if rand()>0.9  
        break  
    end  
end
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
11
```

```
12
```

```
13
```

```
14
```

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

MATLAB - COLUMN ORIENTED LANGUAGE -> в памяти при хранении матрицы A , ее элемент $A(i+1,j)$ находится ближе к $A(i,j)$, чем $A(i,j+1)$ => алгоритмы перебирают элементы матрицы по колонкам

BATTLE† Перебор столбцов vs перебор строк:

```
% timeit - замеряет время вызова (вызывает много раз и усредняет)
clearvars
disp("заполнение с перебором элементов вдоль строки A(i,j)=>A(i,j+1):")
```

заполнение с перебором элементов вдоль строки A(i,j)=>A(i,j+1):

```
timeit(@fill_by_row) % заполнение матрицы с перебором по строкам
```

```
ans = 0.1442
```

```
disp("заполнение с перебором элементов вдоль столбца A(i,j)=>A(i+1,j):")
```

заполнение с перебором элементов вдоль столбца A(i,j)=>A(i+1,j):

```
timeit(@fill_by_column) % заполнение матрицы с перебором по колонкам
```

```
ans = 0.0577
```

BATTLE† Сравнение скорости работы векторизованного метода и перебора массива циклами:

```
clearvars
M = rand(5000);
disp("Функция выполняется отдельно для каждого элемента матрицы:")
```

Функция выполняется отдельно для каждого элемента матрицы:

```
timeit(@()sin_in_circle(M)) % заполнение матрицы с перебором по строкам
```

```
ans = 0.5458
```

```
disp("Функция выполняется отдельно для каждого элемента матрицы, но отсутствует  
вложенный цикл (линейное индексирование:")
```

Функция выполняется отдельно для каждого элемента матрицы, но отсутствует вложенный цикл (линейное индексирование:

```
timeit(@()sin_in_circle_line_index(M)) % заполнение матрицы с перебором по строкам
```

```
ans = 0.3706
```

```
disp("Функция выполняется непосредственно для всей матрицы:")
```

Функция выполняется непосредственно для всей матрицы:

```
timeit(@()sin_direct(M)) % заполнение матрицы с перебором по колонкам
```

```
ans = 0.0891
```

Выводы по семинару 2.

1. Для ветвления потока выполнения: **if ...elseif...else...end** и **switch...case {выбор 1}. {действие1}.case {выбор 2} {действие2} otherwise.{действие }..end**
2. Для проверки условий, например, типа объекта удобно использовать **short-circuit** скалярные операции **&&** и **||**, которые позволяют сформулировать **if-else-end** в значительно более компактной форме
3. Для многократного выполнения однотипных операций есть **for...break...continue...end** и **while...break...continue...end**
4. МАТЛАБ - column-oriented язык, перебирать элементы массива лучше всего вдоль столбцов
5. Перебор элементов массива при помощи линейного индексирования быстрее вложенных циклов
6. Если есть возможность свести задачу к операциям непосредственно с матрицами (**векторизовать задачу**), это будет самым быстрым вариантом за счет встроенных операций.
7. Многие функции по умолчанию "воспринимают" матрицу как набор столбцов
8. В цикле "итерирование по коллекции" происходит по столбцам матрицы

```
function return_value = like_example(be_like_me)
    return_value = zeros(numel(be_like_me), 'like', be_like_me);
    return_value = return_value*be_like_me(:);
end
function A = fill_by_row()
    N = 5000;
    A = zeros(N);
    for iii=1:N % внешний цикл перебирает строки
        for jjj=1:N
            A(iii,jjj) = 5;
        end
    end
end
function A = fill_by_column()
    N = 5000;
    A = zeros(N);
    for jjj=1:N % внешний цикл перебирает колонки
        for iii=1:N
            A(iii,jjj) = 5;
        end
    end
end
function A=fill_by_column_no_memalloc()
    N = 5000;
    for jjj=1:N % внешний цикл перебирает колонки
        for iii=1:N
            A(iii,jjj) = 5;
        end
    end
end
function MAT=fill_by_column_reverse_order()
    N = 5000;
```

```

    for jjj=N:-1:1 % внешний цикл перебирает колонки
        for iii=N:-1:1
            MAT(iii,jjj) = 5;
        end
    end
end

function [r_str,r_ch] = gen_random_string(N)
    alphabeth = 'a':'y';
    n = numel(alphabeth);
    rand_inds = randi(n,[1,N]);
    r_ch = alphabeth(rand_inds);
    r_str = string(r_ch);
end

%% Сравнение операций, выполняемых непосредственно для всей матрицы и перебором
элементов матрицы
function A = sin_in_circle(A)
    N = size(A);
    for jjj=1:N(2) % внешний цикл перебирает колонки
        for iii=1:N(1)
            A(iii,jjj) = sin(A(iii,jjj));
        end
    end
end
function A = sin_direct(A)
    A = sin(A);
end
function A = sin_in_circle_line_index(A)
    N = numel(A);
    for iii=1:N
        A(iii) = sin(A(iii));
    end
end
end
%
function out = ALL(A)
    out = sum(A,'all');
end
% что быстрее итерирование по коллекции или итерирование с индексацией
function s = indexwise_iter() % индексирование по индексам
    A = rand(100000,1);
    s=0;
    for iii = 1:numel(A)
        s = s+ A(iii);
    end
end
function s = elementwise_iter()
    A = rand(100000,1);
    s=0;
    for a = transpose(A)

```

```

        s = s+ a;
    end
end
% Пример использования структур типа cell - функция с произвольным числом
% аргументов
function varar_fun(varargin)
    counter = 0;
    for arg = varargin
        counter = counter + 1;
        disp("arg" + counter);
        disp(arg{1})
    end
end

function folder = get_folder()
% текущая папка
folder = fileparts(matlab.desktop.editor.getActiveFilename);
end

```