Table of Contents

Семинар 1. Базовые типы МАТЛАБ....1
Тип double - основной числовой тип 2
Тип logial - основной логический тип logical - 2 бита....3
Другие типы (об их существовании можно не знать до того как понадобится писать данные в бинарные файлы или работать с портами в бинарном режиме):....3
Массивы: вектора, матрицы и многомерные массивы чисел и логических элементов 3
Индексирование....4
Символьные типы char и string....7
Базовые операции над массивами....8
Выводы семинара 1....8
....9
Семинар 2. Арифметические операции над массивами и инструменты языка для управления потоком выполнения программы 9
Арифметика 9
Контроль потока выполнения....10
if ...elseif...else...end....10
short-circuit логические операции....10
switch...case {выбор 1}.{действие1}.case {выбор 2} {действие2} otherwise.{действие }..end....11
try.{код, который может дать ошибку}..cath Exception{переменная хранящая объект ошибки}.....{код, который выполнется, если в коде блока try есть ошибка}...end....12
Циклы....12
for...break...continue...end....12
while...break...continue...end....13
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 14
Выводы по семинару 2....14
Семинар 3. Контейнеры для работы с разнотипными данными....15
Дополнительные примеры к предыдущему семинару....15
Распараллеливание вычислений при помощи parfor....end....15
Контейнеры (встроенные типы для хранения разнородных данных)....17
тип cell....17
тип struct....18
Выводы по семинару 3....18
Семинар 4. Контейнеры для работы с разнотипными данными....19
тип table....20
тип containers.Map....20
тип dictionary (рекомендуется вместо containers.Map)....20
Итерирование по коллекциям....21
Не хватает коллекции уникальных элементов типа множество....22
Вариант 1 методы матлаб для работы с массивами как с множествами:....22
Вариант 2: использовать богатый арсенал java (collections)....22

Семинар 1. Базовые типы МАТЛАБ

- Типы double, logical
- Массивы, размерность, индексирование

Тип double - основной числовой тип

Число с плавающей точкой x , хранится в виде:

$$x = (-1)^s \cdot (1 + f) \cdot 2^e$$

где:

- *s* определяет знак
- *f* - мантисса, для которой 0 ≤ *f* < 1.
- *e* - экспонента.

s, *f*, и *e* определяются конечным числом бит в памяти

Тип **double** требует 64 бита, которые распределены, как показано в таблице.

Bits	Width	Usage
63	1	хранит знак, 0 положительный, 1 отрицательный
62 to 52	11	Хранит экспоненту, смещенную на 1023
51 to 0	52	Хранит мантиссу

```
class(1) % функция class(...) возвращает тип объекта
class(1.0)
1==1.0 % эквивалентно, то есть даже те числа, которые пишутся как целые матлабом
% воспринимаются как числа с плавающей точкой (если никакого типа не
% задано)
1e3 % можно записывать в e-notation

 num2hex(1) % представляет число в шестнадцатеричной системе
num2hex(1.0)
num2hex(1.0000000000000001)
num2hex(1.0000000000000002)
num2hex(realmin)
1.0000000000000001==1
1.0000000000000002==1

eps % минимальная разница между числами с плавающей точкой
realmin % минимальное значение
realmax
a = realmax
b = a+eps
a-b
```

Особые числа inf и NaN

```
num2hex(Inf)
num2hex(NaN)
%NaN==NaN
Inf==Inf
bit_nan = num2hex(NaN)
bit_nan(end)='f'
hex2num(bit_nan)
fit_inf = num2hex(inf)
fit_inf(1)='7'
hex2num(fit_inf)
```

Тип logial - основной логический тип `logical - 2 бита`

```
a = true
b= false
```

Другие типы (об их существовании можно не знать до того как понадобится писать данные в бинарные файлы или работать с портами в бинарном режиме):

- `single` - 32-бита
- `int8`, `int16`, `int32`, `int64`
- `uint8`, `uint16`, `uint32`, `uint64`
- `char - 8 бит`

Массивы: вектора, матрицы и многомерные массивы чисел и логических элементов

Массив - конструкция для хранения данных одного типа.

Заполнения массивов вручную

```
clearvars % удаляет все переменные из памяти
a_row = [1,2,3,4] %  - строка
a_col = [1;2;3;4] %  - столбец
a_row2 = [1 2 3 4]; %  - строка
a_lin = linspace(1,10,100); % фиксировано количество элементов (возвращает строку)
a_range = 1:0.01:100; % фиксированный шаг
a_range2 = 1:10; % если шаг не указанЮ будет единица
a_mat = [1,2,3,4;5,6,7,8] % матрица 2x4
```

Размерность массива

```
clearvars
a_scalar = 5 % массив размером (1,1)
a_row = [1,2,3,4]; %  - строка
a_col = [1;2;3;4]; %  - столбец
```

```
a_mat = [1,2,3,4;5,6,7,8]; % матрица 2x4
size(a_row)
size(a_col)
numel(a_mat)
size(a_mat)
size(a_mat,2) % количество столбцов
ndims(a_mat) % количество размерностей больших единицы
empty_mat = []
size(empty_mat)
whos empty_mat % функция whos выдает информацию о переменных
```

Индексирование

Прямое индексирование.

```
clearvars
a_rand = rand(10); % функция создает массив случайных чисел размером 10x10
a_rand(1,1) % (строка,столбец) декартово индексирование
a_rand(100) % линейное (MATLAB - COLUMN ORIENTED LANGUAGE!)
a_scalar = 5 % массив размером (1,1)
a_scalar(1) % ВСЕ МАССИВ!
```

Индексирование при помощи "сахара": символы "**:**" и "**end**"

```
clearvars
a_rand = rand(10);
col1 = a_rand(:,2)% возвращает второй толбец
row1 = a_rand(1,:)% возвращает первую строку
a_mat = [1,2,3,4;5,6,7,8]
col = a_mat(:) % превращает матрицу в вектор-колонку
submat = a_mat(2:end,:)
```

Если при работе программы происходит изменение размера массива, то это сильно замедляет вычисления, чтобы этого избежать нужно резервировать память

Способы резервировать память

```
clearvars
a_zeros = zeros(10);
a_ones = ones(10,'double');
a_nan = NaN(10);% not-a-number
a_zeros5x5x10 = zeros([5 5 10]);
i_complex = 1i; % мнимая единица
a_complex = zeros(10,'like',i_complex); % удобно, когда в процессе работы не
известно какого типа будут данные  (пример - в функциях)
a_uint8 = zeros(10,'uint8');% можно указывать тип в явном виде
a_logical = false(10);% матрица 10x10 false
a_rand = rand(10);% матрица 10x10
```

BATTLE† - соревнование по скорости

timeit() - позволяет замерить время работы функции без входных аргументов

значок "@" - создает указатель на функцию (что это такое обсудим когда-нибудь потом), в данном случае нужен чтобы превратить функцию с аргументом в функцию без аргумента

BATTLE† Сравнение скорости заполнения матриц

```
clearvars
disp("Заполнение NaN'ами:")
t1 = timeit(@()NaN(1000)) % заполнение NaN
disp("Заполнение нулями:")
t2 = timeit(@()zeros(1000)) % заполнение нулями
disp("Заполнение inf'ами:")
t3 = timeit(@()inf(1000)) % заполнение инфами
disp("Заполнение единицами:")
t4 = timeit(@()ones(1000)) % заполнение единицами
```

BATTLE†Влияние преаллокации памяти на скорость вычислений

```
clearvars
disp("заполнение без преаллокации памяти:")
timeit(@fill_by_column_no_memalloc) % no memory allocation
disp("заполнение с преаллокацией памяти:")
timeit(@fill_by_column) % with memory allocation
```

Заполнение матрицы из массива

```
clearvars
A = zeros(10);% memory allocation
a = linspace(1,100,100);
A(:) = a
```

Индексирование при помощи массива

```
clearvars
a_col = [1;2;3;4]; %  - столбец
r_index = randi(4,100);% функция возвращает 100 случайных целых числа в интервале
от 1 до 4
a_permuted = a_col(r_index) % матрица из случаных перестановок элементов массива
a_col
```

Пример - вытаскивание диагонали:

```
clearvars
a_rand = rand(10); % генерим матрицу
```

```
col_size = size(a_rand,1); % число элементов в одном столбце
a_diagonal = a_rand(1:col_size+1:end); % индексирование при помощи диапазона с шагом
```

Удаление элемента при помощи индексирования

```
clearvars
a = 1:5;
a(2) = [] % изменяет матрицу in-place!
a(2:4)=[]; %???
```

Индексирование многомерных массивов

```
a_rand_3d = rand([10,10,10]);% трехмерная матрица
a_rand_3d(2,3,5) % элемент второй строки третьего столбца пятой страницы
size(a_rand_3d(:,2,:))
size(a_rand_3d(:))
```

Создание массива при индексировании:

```
clearvars
a_mat(10,10)=10
```

BATTLE†Преаллокация памяти путем обратного индексирования:

```
disp("заполнение без преаллокации памяти:")
timeit(@fill_by_column_no_memalloc) % no memory allocation
disp("заполнение с преаллокацией памяти:")
timeit(@fill_by_column) % with memory allocation
disp("заполнение с преаллокацией памяти путем обратного индексирования:")
timeit(@fill_by_column_reverse_order)
```

Таким образом: A(10:-1:1)=0, быстрее, чем A(1:1:10), так как в первом случае, преаллокация происходит один раз на первой итерации!

Но A = zeros([10,1]), все равно быстрее

Конкатенация матриц и векторов

```
clearvars
a = 1:5 % создаем вектор-строку
b = [a,10] % [.,.]  функции horzcat
b2 = horzcat(a,10)
c = [a,a]
v_col = [a(:),a(:)]
v_row = [a(:);a(:)]% [.;.]  функции vertcat
v_row2 = [a;a]
v_mat = [rand([5 2]),rand([5 2])]
```

```
v_mat2 = [rand([5 2]);rand([5 2])]
```

Символьные типы char и string

```
clearvars
s_char = 'asfzassgfagag'; % ведут себя как массивы символов
s_string = "asfzassgfagag";% ведут себя как массивы групп символов
s_char(10)
%s_string(10)
s_char_pen = char(10000);
s_char_big = char(rand([1 10000]));
s_string_big = string(s_char_big);
onechar = 'a';
onestring = "a";
whos one*
whos s_*
a_range_char = 'a':'z';
a_range_string = arrayfun(@(x)string(x),a_range_char);% создаем массив строк
"a"<="b"
'a'<='b'
s_char<s_char
```

Пустые массивы

```
clearvars
empty_char = ''
fake_empty_string = ""
whos empty_char
whos fake_empty_string
real_empty_string = strings(0,0)
whos real_empty_string
```

BATTLE† Что быстрее строки или массивы символов?

```
[r_str,r_ch] = gen_random_string(1e6)% функция генерит случаную последовательность
1e5 символов
disp("Поиск и замена паттерна в строке:")
timeit(@()replace(r_str,"a","I"))
disp("Поиск и замена паттерна в массиве символов:")
timeit(@()replace(r_ch,'a','I'))
```

Конкатенация символьных типов

```
clearvars
s_char = 'a':'z'
[s_char,s_char]
[s_char;s_char]
s_string = arrayfun(@(x)string(x),s_char);% создаем массив строк
```

```
[s_string;s_string]
```

Базовые операции над массивами

Операции сравнения и логическое индексирование

```
clearvars
a = 1:10 % вектор-строка
% сравнение массива и скаляра
flag1 = a>3 % логический массив
sum(flag1) % самый быстрый способ посчитать количество элементов массива,
удовлетворящих условию
b = randi(10,[1 10]) % вектор-строка случаных элементов
flag2 = a==b % проверяет элементы
A_mat = rand(10)
flag_mat = A_mat>0.5
sum(flag_mat,"all") % суммирование по всем элементам
A_mat(flag_mat) % возвращает вектор-столбец
col_vec = randi(10,[5,1])
row_vec = randi(10,[1,3])
flag_col_row = col_vec>row_vec % ???
flag_row_col = row_vec>col_vec % ???
```

Как выбрать из массива элементы, удовлетворяющие неравенству:

```
clearvars
 a = rand(10,1)
 flag1 = a>0.8
 flag2 = a<0.2
 % логический плюс
 flag_union = flag1 | flag2
 %
 flag_intersection = a>=0.2 & a<=0.5
 a(flag_union) % элементы, которые либо <0.2, либо >0.8
 a(flag_intersection) % элементы лежащие внутри [0.2,0.8]

 % эквивалентно неравенству
```

Выводы семинара 1.

1. Базовый матлаб - double - число с плавающей точкой
2. Базовый объект - массив.
3. Массивы поддерживают два основных метода работы a= M(i) - получить элемент массива (get_index), M(i)=a - задать элемент массива (set_index)
4. Массивы поддерживают линейное индексирование (один индекс) и декартово (число индексов равно размерности массива).

5. Можно индексировать при помощи массивов индексов и при помощи логических массивов
6. Метод set_index позволяет изменять размер массива в процессе выполнения программы, однако это медленно, поэтому нужно делать преаллокацию
7. Самый быстрый способ преаллокации - функция zeros(), можно делать преаллокацию обратным индексированием
8. Есть два базовых типа символьных объектов: 'char' - массив символов и "string" - массив групп символов, первый ест меньше памяти, второй немного быстрее

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 = "sym"
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
% СЛОЖЕНИЕ
a_col + a_col
plus(a_col,a_col)
a_col + b_row
a_col.^b_row
M3x4 + W3x4
```

```
M3x4 + 3
M3x4 + a_col
M3x4 + b_row
% УМНОЖЕНИЕ
2*M3x4
M3x4*transpose(b_row) % ' - транспонирование матрицы
M3x4.*a_col
B3x3^2
B3x3.^2
M3x4.^2 % поэлементные операции
% ПРАВОЕ И ЛЕВОЕ ДЕЛЕНИЕ
mldivide(M3x4,a_col) % левое деление решает Ax = B => x = A\B
B3x3\a_col
% mrdivide(M3x4,transpose(a_col)) % правое деление решает xA = B => x = A/B
```

Операции над строками

```
clearvars
s_string = "string1"
"str" + s_string
```

Контроль потока выполнения

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

Ветвление кода

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

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 случаную матрицу с числом, при этом
%если в качестве аргумента задается не число, он выдает сообщение
clearvars
a="a"
try % ловим ошибку (об этой контрукции - позже)
    rand(1000)>a
catch ex
    disp(ex.message)% нельзя сравнивать тип строка с числовым типом
end

% 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
```

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

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

```
clearvars
a = '2C00'; % UNICODE символ для Азъ
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 + " Азъ")
    case 1
        disp(str + " Буки")
    case 2
        disp(str + " Веди")
    otherwise
        disp(str + " Глаголи")
end
```

try.{код, который может дать ошибку}..cath 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
```

Циклы

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

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

    if iii>2
```

```
        continue % переходит на следующую итерацию, не выполняя инструкции ниже
    end
    disp(iii)
end
```

Итерирование по элементам массива

```
% итерирование по коллекции
clearvars
%A = 1:5
A = rand(5,1)
%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
for a = transpose(A)
    i = i+1;
    disp("size(a) = " + join(string(size(a))," x "))
    disp("i="+ i + " a = "+ join(string(a)))

end
```

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) :")
timeit(@indexwise_iter) % итерирование по индексам
disp("Итерирование по коллекции типа for a=A :")
timeit(@elementwise_iter) % итерирование по элементам
```

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

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

```
clearvars
iii=0
while true
    iii=iii+1;
    disp(iii)
    if rand()>0.9
        break
    end
```

```
end
```

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):")
timeit(@fill_by_row) % заполнение матрицы с перебором по строкам
disp("заполнение с перебором элементов вдоль столбца A(i,j)=>A(i+1,j):")
timeit(@fill_by_column) % заполнение матрицы с перебором по колонкам
```

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

```
clearvars
M = rand(5000);
disp("Функция выполняется отдельно для каждого элемента матрицы:")
timeit(@()sin_in_circle(M)) % заполнение матрицы с перебором по строкам
disp("Функция выполняется отдельно для каждого элемента матрицы, но отсутсвует
вложенный цикл (линейное индексирование:")
timeit(@()sin_in_circle_line_index(M)) % заполнение матрицы с перебором по строкам
disp("Функция выполняется непосредственно для всей матрицы:")
timeit(@()sin_direct(M)) % заполнение матрицы с перебором по колонкам
```

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

Семинар 3. Контейнеры для работы с разнотипными данными

- Несколько примеров итерирования по коллекции
- Распараллеливание циклов
- Объекты для хранения данных разнородных типов
- Ячейки
- Структуры
- Таблицы
- Множества
- Словари
- Использование контейнеров java

Дополнительные примеры к предыдущему семинару

Многие функции по умолчанию "воспринимают" матрицу как вектор набор столбцов

```
clearvars
M3x4 = sym('m',[3,4])
sumM = sum(M3x4)
size(sumM) %- вектор-строка сумм элементов в столбцах
prod(M3x4) %
 % norm(M3x4)
Mlog = rand(3,6)>0.5
all(Mlog)
all(Mlog,'all')
any(Mlog)
```

В цикле "итерирование по коллекции" происходит по столбцам матрицы

```
clearvars
a_col = sym('a',[3 1])
b_row = sym('b',[1 4])
M3x4 = sym('m',[3,4])
for a=M3x4
    a % - это столбец матрицы
end
A = a_col*b_row;
B = sym([]);
for b = b_row
    B = [B,a_col*b];
end
A
B
```

Распараллеливание вычислений при помощи parfor....end

Представляю себе так:

параллельно запускаются столько матлабов сколько итераций в цикле, на каждом матлабе создается полная копия "внешних" по отношению к телу цикла данных (в данном случае - это вектор cConst , матрица A, n). Каждый работник не знает о существовании остальных, но знает номер итерации. Порядок выполнения итераций - произвольный.

*Если происходит распараллеливание на процессах, то примерно так и происходит в реальности, однако количество работников определяется при конфигурации профиля

С учетом этого цикл должен работать так, чтобы работники не стукались лбами.

основные правила:

1. Вложенные parfor - игнорируются (причем они могут быть где-то дальше по ветке кода)
2. Если необходимо в циклах производить "побочные" действия, такие как изменения элементов массива необходимо делать это так, чтобы не могло возникнуть ситуации, когда несколько работников одновременно меняют один и тот же набор эелементов
3. *(пункт для порядка) Если в цикле происходят изменения объекта типа "handle", то эти изменения не будут доступны после выполнения цикла, так как каждый работник будет менять свою копию объекта

```
clearvars
cur_pool = gcp;
n = 100;
A = zeros(n);
cConst = randn(n,1);
tic
ticBytes;
parfor i=1:n
    Z = rand(n);
    if (norm(Z)/n) > 0.5
        continue
    end
    Z = cConst.*Z;
    % f = A(i+1) - обращение к другому элементу
    A(:,i) = svds(Z,n);
    % break- нельзя!
    % continue  - можно
end
A
toc
tocBytes
wh = whos("A","cConst");
sum(arrayfun(@(x)x.bytes,wh))*cur_pool.NumWorkers
cConst = randn(n,1);
A = zeros(n);
tic
for i=1:n
    Z = rand(n);
    if (norm(Z)/n) > 0.5
```

```
        continue
    end
    Z = cConst.*Z;
    A(:,i) = svds(Z,n);
end
toc
A
```

Контейнеры (встроенные типы для хранения разнородных данных)

тип cell

Для хранения разнотипных данных в структуры с числовыми индексами

```
clearvars
array_of_cell = {'1',"2",3;@peaks,rand(10),figure(10)} %
array_of_cell(1) % возращает тип cell
array_of_cell{1} % возвращает содержимое ячейки
array_of_cell{2,1}() % вызываем указатель на функцию, который хранится в ячейке
% array_of_cell(1){1} так нельзя
whos array_of_cell
empty_cell = {};
whos empty_cell
empty_cell{1} = true
whos empty_cell
```

Массив ячеек

```
clearvars
array_of_cell = {'1',"2",3;@peaks,rand(10),figure(10)} %
array_of_cell_of_cell = {array_of_cell;array_of_cell} % заворачивает cell в cell =>
массив ячеек массивов ячеек
array_of_cell_concat = [array_of_cell;array_of_cell] % матрица ячеек
array_of_cell_concat{2,1}() % - вызываем содержимое ячейки (а там - указатель на
функцию peaks)
```

```
clearvars
folder = get_folder()
full_file = fullfile(folder,"tbl.xls")
cell_1 = cell(21,6);
cell_1(2:end,:) = num2cell(rand([20 6]));
cell_1(1,:) = {"а" "ё" "а" "а" "и" "л" };
writecell(cell_1,full_file);
tbl1 = readcell(full_file)
help("readcell")
```

Пример использования "cell" - splat аргументов функции

```
% Пример  - splat аргументов функции
A = rand(1,5);
varar_fun(A(:))
C = num2cell(A);
varar_fun(C{:})
```

тип struct

Для хранения разнородных данных по имени

```
clearvars
struct_scalar = struct("field1",rand(10),"field2",'Содержимое ячейки 2',"field3",
[], "peaks_handle",@peaks) % стурктура с полями
struct_scalar.field1 % получение данных из поля по названию поля
names_of_fields = fieldnames(struct_scalar) % возвращает имена полей
struct_scalar.(names_of_fields{2}) % имена можно задавать в виде символов,но это
несколько медленнее
struct_scalar.peaks_handle()
```

Массив структур

```
clearvars
struct_scalar = struct("field1",rand(),"field2",'Содержимое ячейки 2',"field3",[],
"peaks_handle",@peaks) % стурктура с полями
struct_array_empty(10) = struct_scalar % резервирование памяти путем обратного
индексирования  - для структур довольно удобно
% так как массив структур имеет все те же имена полей
struct_array_filled(10:-1:1) = struct_scalar
isfield(struct_scalar,"a")
struct_array_filled(3).field4 = "f" % новое поле добавилось ко всем элементам
struct_array_filled(end).field1=rand();
struct_array_filled(1).field1=1;
a = struct_array_filled(:).field1 % возвращает первый элемент
a(10:-1:1)=struct_array_filled(:).field1 % тут не получилось
a= [struct_array_filled(:).field1] % list comprehention

% varar_fun - показывает свои аргументы по одному
varar_fun("первый аргумент", pi)
varar_fun(struct_array_filled(:).field1)  % можно использовать аналогично {:} для
передачи большого числа аргументов по одному (splat -функция)
```

Выводы по семинару 3.

1. Если есть "parallel computing toolbox", то можно распараллеливать вычисления при помощи **parfor...continue....end.** Основной принцип расботы с параллельными циклами - нобходимо

помнить, что итерации цикла выполняются независимо друг от друга и нельзя чтобы в коде цикла могла возникнуть ситуация, когда параллельные процессы конкурируют за один и тот же элемент массива

2. Для хранения разнотипных данных в объектах с индексированием по типу массивов можно пользоваться массивом ячеек, тип **cell,** для которого существуют два способа индексирования (i,j) - возвращает элемент массива с типом **cell**, {i,j} - вытаскивает содержимое ячейки. Индексирование с круглыми скобками позволяет делать все те же операции с массивом ячеек, что и с массивом чисел.
3. Тип **cell** используется для передаче функции произвольного числа аргументов, для передачи аргументов по одному можно использовать **splat** - функцию **{:}**
4. Тип **struct** - для хранения разнотипных данных в массиве с "индексированием" при помощи имени. Может также использоваться для передаче большого числа аргументом по одному

Семинар 4. Контейнеры для работы с разнотипными данными

- Таблицы
- Множества
- Словари
- Использование контейнеров java

Небольшой пример по использованию структур и функции **{:}** из предыдущего семинара

Конструктор объекта типа **struct** имеет следующую форму:

struct("field1_name",value1,...."fieldN_name",valueN)

Для конструкирования структуры с заданными именами полей можно использовать возможности "splat" - функции ячеек.

Для примера создадим стурктуру с именами полей от 'a' до 'y' и значениями в этих полях от 1 до 25

```
clearvars
field_names = arrayfun(@string,'a':'y');
n = numel(field_names);
Name_Values_cell = cell(1,2*n); % создаем пустой массив ячеек,
% в него будут поочередно добавлены имена полей и их значения

% заплоняем имена полей
Name_Values_cell(1:2:end) = cellstr(field_names); % cellstr - преобразует массив
string в ячейку массивов char
Name_Values_cell
% заполняем содержимое ячеек
Name_Values_cell(2:2:end) = num2cell(1:n); %
Name_Values_cell
st = struct(Name_Values_cell{:}) % конструктор структур поддерживает произвольное
число аргументов
```

```
st.y
```

тип table

Для хранения данных в табличках с именами столбцов

```
clearvars
folder = get_folder()
full_file = fullfile(folder,"tbl.xls")
cell_1 = cell(21,6);
cell_1(2:end,:) = num2cell(rand([20 6]));
cell_1(1,:) = {"a" "ё" "а" "а" "и" "л" };
writecell(cell_1,full_file);
tbl1 = readtable(full_file);
% можно читать таблички из эксель или текстовых файлов с разделителем,
например, .csv
% readtable - высокоуровневая читалка с очень большим набором функций
help("readtable")

% функции, которые работают с объектами типа таблица
methods(tbl1)

tbl1.Properties.VariableNames = {'name' 'a1' 'a2' 'a3' 'a4' 'a5'}
```

```
mean_val = mean(tbl1(:,2:end)) % среднее значение
standard_deviation=std(tbl1(:,2:end)) % среднее значение
summary(tbl1)
bar(tbl1.name,tbl1.a1)
stackedplot(tbl1)
%tbl2 = table([1:4]',ones(4,3,2),eye(4,2))  - элементы разной размерности
%работают но не отображаюстя в LiveScript
```

тип containers.Map

Для хранения разнородных данных по имени (ключу)

```
clearvars
M = containers.Map('KeyType','char','ValueType','double')
M("a")=10
methods(M)
```

тип dictionary (рекомендуется вместо containers.Map)

Для хранения и получения данных по "ключу"

```
clearvars
```

```
d =  dictionary(["sin" "cos" "tan"],{@sin, @cos, @tan})
d("sin")
d1 = d("sin")
d1{1}(pi)
d("cot") = {@cot}

d2 = dictionary({[false true true] [true false true] [true true false]},{@sin,
@cos, @tan})
% можно в качестве ключей использовать массивы
fun = d2({[true true false]})
fun{1}(pi)
```

Итерирование по коллекциям

Циклы могут перебирать элементы коллекций (но только родных джавовских не могут)

```
% итерирование по ячейкам
A_cell = {1,2,3}
for a = A_cell
    class(a)
    disp(a{1})
end


% итерирование по структурам
A_struct(3) = struct('f1',3);
A_struct(1).f1 = 1;A_struct(2).f1 = 2;

for st = A_struct
    disp("st(i)=" + st.f1)
end

%итерирование по словарям

A_dict = dictionary(["a" "b" "c"],[1 2 3])
try
    for d = A_dict
        d
    end
catch Ex
    "ss"
end
i = 0;
for key = keys(A_dict)'
    i = i+1
    disp("key => value:  " + key+"=>"+ A_dict(key))
end
A_dict(key)
```

Не хватает коллекции уникальных элементов типа множество

Вариант 1 методы матлаб для работы с массивами как с множествами:

```
clearvars
A = ["c"  "b" "a" "c"];
B = ["a" "d"];
setdiff(A,B) % Элементы множества A не содержащиеся в множестве B
setdiff(A,B,'stable') % чтобы сохранить изначальный порядок элементов в массиве
intersect(A,B) % Пересечение двух множеств
unique(A)
```

Вариант 2: использовать богатый арсенал java (collections)

Матлаб имеет "встроенный" java 8

```
methodsview(java.util.HashSet) % Графическая оболочка для java документации
```

Пример нахождения пересечения двух множеств

```
import java.util.HashSet % java.util.* - импортирует все коллекции из джавы
jA = HashSet; % вызываем конструктор для джава объекта
jB = HashSet; % вызываем конструктор для джава объекта
methods(jA)

A = ["a" "b" "c"];
B = ["a" "d"];

for iii = A
    add(jA,iii) % добавляем элемент в множество jA
end
for iii = B
    add(jB,iii) % добавляем элемент в множество jB
end
unionAB = clone(jA) % клонирует объект (метод java)
unionAB.addAll(jB) % функция добавляет элементы множества jB в множество  unionAB

jA
unionAB

intersectionAB = clone(jA)
intersectionAB.retainAll(jB)

jA
intersectionAB
```

Вместо типа dictionary можно использовать java.util.HashMap, это возможно будет работать быстрей (скорее всего)

```
methodsview(java.util.HashMap)
```

```
import java.util.HashMap % можно использовать вместо словарей
jMap = java.util.HashMap; % создается объект java с которым напрямую можно работать
из матлаб
methods(jMap)
jMap.put("a",figure(1));
jMap.put("b",figure(2));
keySet(jMap)
jMap.get("b") % вытакскиваем число
jMap
jArrayObj = jMap.values().toArray()
methods(jArrayObj)
f_array= arrayfun(@(i)jArrayObj(i), 1:jMap.size(),"UniformOutput",false);
f_array{1}
```

```
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)
        alfabeth = 'a':'y';
        n = numel(alfabeth);
        rand_inds = randi(n,[1,N]);
        r_ch = alfabeth(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
%
function out = ALL(A)
    out = sum(A,'all');
end
% что быстрей 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 исопльзования 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
```