



**ПРИКЛАДНЫЕ ЗАДАЧИ
АНАЛИЗА ДАННЫХ
(лекция-1)**

**MATRIX LABORATORY
(эффективное программирование)**

Дьяконов А.Г.

**Московский государственный университет
имени М.В. Ломоносова (Москва, Россия)**

Спецкурс для студентов старших курсов

Языки и платформы для анализа «малых данных»

- **MATLAB (m-язык)**
- **R (S + randomForest, gbm, ...)**
- **Python (+ numpy, scipy, pandas, sklearn, ...)**
 - **остальное (Java, SAS, SQL, C/C++/C#)**

Каждый язык для своего!

| R | MATLAB |
|--|--|
| для анализа данных - загрузка csv-файлов - датафреймы | для матричных вычислений - всё – матрицы - эффективное матричные вычисления |
| * – поэлементное умножение $x = \text{solve}(A, b)$ | * – матричное умножение $x = A \backslash b$ |

MATLAB (компании «The MathWorks, Inc.»)

- + наиболее простой**
- + идеален переход с C/C++**
- + чистая линейная алгебра**
- + идеален для прототипирования**
- + хорошая справочная система**
- + прекрасный пошаговый отладчик**
- + на синтаксисе Матлаба много чего основано (numpy)**

- платный**
- нет/мало библиотек (для АД)**
- приходится писать с нуля**
- нет портирования сложных данных (уже есть)**

Зачем MATLAB дата-майнеру

из личного опыта... процент использования Матлаба:



 – можно заменить другим средством,  – нельзя

The screenshot displays the MATLAB R2012b environment. The Editor window shows a script named `sc_loaddata4.m` with the following code:

```
7 - testid_A = []; % номера пользователей
8
9 - while 1
10 -     s = fgetl(ffile);
11 -     if ~ischar(s), break, end;
12
13 -     t = find(s==' ');
14 -     id = str2num(s(1:t-1)); % число до двоеточия
15 -     j = str2num(s(t+1:end)); % числа после
```

The Command Window shows the execution of the following commands:

```
>> a = [1 2; 3 4]

a =

     1     2
     3     4

>> diag(diag(a))

fx
```

The Workspace window displays the following variables:

| Name | Value | Min | Max |
|------|------------|-----|-----|
| a | [1 2; 3 4] | 1 | 4 |
| ans | [1 0; 0 4] | 0 | 4 |

The Command History window shows the following commands:

```
clear
a = [1 2; 3 4]
diag(diag(a))
```

Основные окна

Command Window – вводятся команды (стандартные приёмы типа ↑)

Editor – пишутся m-скрипты (команда edit)

Command History – история команд

Workspace – переменные в памяти

Help – помощь

Current Directory – текущая директория (тут ищутся скрипты)

Figure – рисунки

Profiler – профайлер

MatLab – это интерпретатор!

Стандартные переменные

i
j
pi
Inf
eps
realmax
realmin
ans
NaN

```
i = 2
i =
      2
clear i
i
ans =
      0 + 1.0000i
-1/0
ans =
      -Inf

[1,2] .* [3 4]
ans =
      3      8
```

clear – очистка памяти

; – конец команды, не выводить результат

Чувствительность к регистру!

Теоретико-множественные операции

```
>> A = [1,3,5,3]; B = [4,2,2,3];
```

```
>> [A, B]
```

```
ans =  
     1     3     5     3     4     2     2     3
```

```
setdiff(A, B)  
intersect(A, B)  
setxor(A, B)  
issorted(ans)  
sort(B)
```

```
>> union(A, B)
```

```
ans =  
     1     2     3     4     5 порядок!
```

```
>> ismember(2, B)
```

```
ans =  
     1
```

```
>> ismember(A, B)
```

```
ans =  
     0     1     0     1 ЛОГИЧЕСКИЙ МАССИВ
```

```
>> unique(B)
```

```
ans =  
     2     3     4
```


Порождение матриц

```
>> A = [1,2; 3,4]
```

```
A =  
     1     2  
     3     4
```

```
>> zeros(3)
```

```
ans =  
     0     0     0  
     0     0     0  
     0     0     0
```

```
>> B = fix(10*rand([2 3]))
```

```
B =  
     2     9     1  
     5     9     9
```

операции поэлементно!

```
>> a = 1:5
```

```
a =  
     1     2     3     4     5
```

```
>> a = 1:2:5
```

```
a =  
     1     3     5
```

```
>> A = ones([2 2 2 1]);
```

```
>> size(A)
```

```
ans =  
     2     2     2
```

```
>> reshape(A, [2 4])
```

```
ans =  
     1     1     1     1  
     1     1     1     1
```

Операции с матрицами

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

```
>> A(:)'
```

```
ans =  
     1     3     2     4
```

```
>> sum(A)
```

```
ans =  
     4     6
```

```
>> sum(A, 2)
```

```
ans =  
     3  
     7
```

```
>> sum(A(:))
```

```
ans =  
    10
```

отличие от суммы:

```
>> max(A)
```

```
ans =  
     3     4
```

```
>> max(A, [], 2)
```

```
ans =  
     2  
     4
```

```
>> max(A, 2)
```

```
ans =  
     2     2  
     3     4
```

```
>> min(A,A')
```

```
ans =  
     1     2  
     2     4
```

Операции выполняются по столбцам!

Конкатенация матриц

```
>> A = [ones(2); eye(2)]
```

```
A =
```

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 0 | |
| 0 | 1 | |

```
>> B = [ones(2), eye(2)]
```

```
B =
```

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |



```
>> C = cat(3, ones(2), eye(2))
```

```
C(:, :, 1) =
```

| | |
|---|---|
| 1 | 1 |
| 1 | 1 |

```
C(:, :, 2) =
```

| | |
|---|---|
| 1 | 0 |
| 0 | 1 |



Индексация элементов

С единицы! Строка – столбец

| | | | |
|-------|-------|-------|-------|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |

| | | | |
|---|---|---|----|
| 1 | 4 | 7 | 10 |
| 2 | 5 | 8 | 11 |
| 3 | 6 | 9 | 12 |

```
>> [i,j] = ind2sub([3 4], [3 7 12])
```

```
i =
     3     1     3
j =
     1     3     4
```

```
>> sub2ind([3,4], 1, 3)
```

```
ans =
     7
```

```
A(1, 2) % элемент
```

```
A(1, 2, 1, 1) % он же
```

```
A(1:2, [2 4]) % подматрица
```

| | | | |
|-------|-------|-------|-------|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |

Индексация элементов

`A(:,3) = []` % удаление

| | | | |
|-------|-------|-------|-------|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |

end – очень удобно

`A(end, 1)` % последняя строка

| | | | |
|-------|-------|-------|-------|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |

`A(:, 2:3) = A(:, [3 2])`
% перестановка столбцов

`>> A = [], A(end+2) = 2`

`A =`
`[]`
`A =`
`0 2`

`>> B(2,2,2) = 3`

`B(:,:,1) =`
`0 0`
`0 0`
`B(:,:,2) =`
`0 0`
`0 3`

Чего нет – `A(:, -i)` **Вместо этого** `A(:, [1:i-1 i+1:end])`

Приведение размеров

```
>> A = reshape(1:9, 3, 3)
```

```
A =  
    1    4    7  
    2    5    8  
    3    6    9
```

```
>> A(:, end) = [1 2 3]
```

```
A =  
    1    4    1  
    2    5    2  
    3    6    3
```

```
>> A(end, :) = 5
```

```
A =  
    1    4    1  
    2    5    2  
    5    5    5
```

```
>> A(1:8) = rand([2 4])
```

```
A =  
    0.8147    0.9134    0.2785  
    0.9058    0.6324    0.5469  
    0.1270    0.0975    5.0000
```

Что нельзя:

```
A + [1,2]
```

Размерность матриц

```
>> A = rand([2 1 3 4 1]);
```

```
>> size(A)
```

```
ans =
```

```
     2     1     3     4
```

```
>> A = squeeze(A);
```

```
>> size(A)
```

```
ans =
```

```
     2     3     4
```

```
>> [rows, cols, ~] = size(A)
```

```
rows =
```

```
     2
```

```
cols =
```

```
     3
```

Удобное устранение фиктивных размерностей.

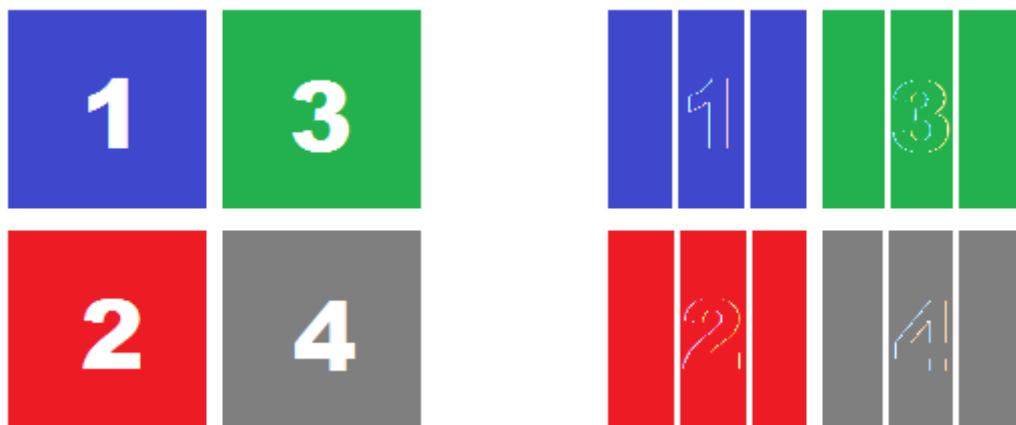
Так помечаются ненужные переменные.

Фокусы с размерностями

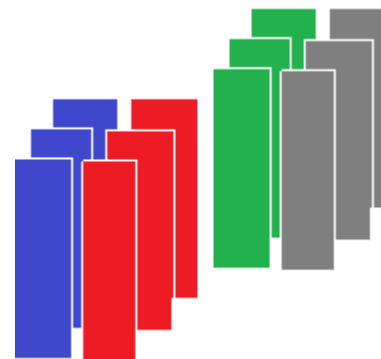
```
>> A = [1 1; 1 1];
>> B = 2*A; C = 3*A; D = 4*A;
```



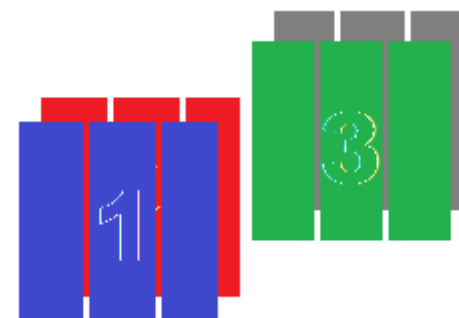
```
>> X = [A, 3*A; 2*A 4*A]
```



```
>> Y = reshape(X, [2 2 2 2])
```



```
>> Y = permute(Y, [1 3 2 4]);
```



```
>> Y = reshape(Y, [2 2 4])
```



Операции с матрицами

```
>> A = [1 2 3; 4 5 6]
```

```
A =  
     1     2     3  
     4     5     6
```

```
>> circshift(A, [0 -1])
```

```
ans =  
     2     3     1  
     5     6     4
```

```
>> flipud(A)
```

```
ans =  
     4     5     6  
     1     2     3
```

```
>> rot90(A)
```

```
ans =  
     3     6  
     2     5  
     1     4
```

```
>> diag(diag(A))
```

```
ans =  
     1     0  
     0     5
```

```
>> triu(A)
```

```
ans =  
     1     2     3  
     0     5     6
```

```
>> cumsum(A)
```

```
ans =  
     1     2     3  
     5     7     9
```

```
>> diff(A)
```

```
ans =  
     3     3     3
```

Работа с файлами

Стандартный `import` – очень неэффективное средство

```
>> save b.mat a
>> load b.mat
>> save b.txt a -ascii
```

**Внутренний
формат: mat-
файлы**

```
>> f = fopen ('st.txt');
>> a = fscanf(f, '%d:%d %d/%d', [4 100])'
```

```
a =
```

```
    12     1     5     7
    13     6     3     2
```

```
12:01 5/7
```

```
13:06 3/2
```

```
>> fclose(f);
```

```
>> g = fopen('st2.txt','w');
```

```
>> fprintf(g, '%2.2d:%2.2d %2.2f/%2.2d\n', a);
```

```
>> fclose(g);
```

```
12:01 5.00/07
```

```
13:06 3.00/02
```

В Матлабе придётся писать ввод-вывод...

Разреженные матрицы

хранятся только ненулевые элементы (по столбцам)

действия как с обычными матрицами

```
>> S = [0 0 1; 2 0 3; 0 0 0];
```

```
>> S1 = sparse(S)
```

```
S1 =
      (2,1)      2
      (1,3)      1
      (2,3)      3
```

```
>> S2 = full(S1)
```

```
S2 =
      0      0      1
      2      0      3
      0      0      0
```

```
>> whos
```

| Name | Size | Bytes | Class |
|-------------------|------|-------|---------------|
| Attributes | | | |
| S | 3x3 | 72 | double |
| S1 | 3x3 | 80 | double sparse |
| S2 | 3x3 | 72 | double |

```
>> i = [1 2 3 4 5];
```

```
>> j = [1 1 2 2 1];
```

```
>> v = (1:5).^2;
```

```
>> sparse(i,j,v)
```

```
ans =
      (1,1)      1
      (2,1)      4
      (5,1)     25
      (3,2)      9
      (4,2)     16
```

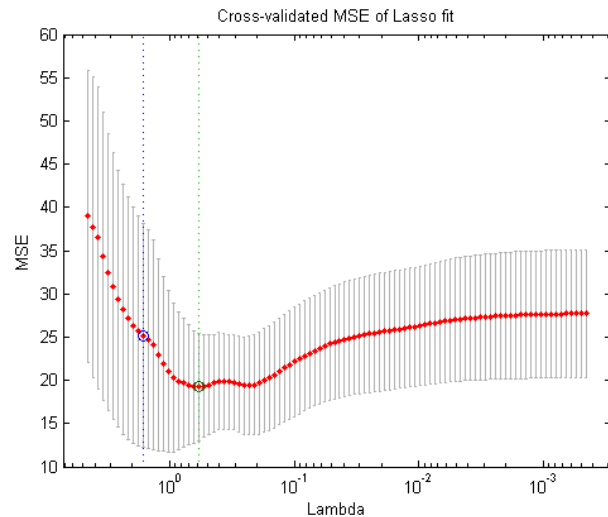
```
speye([2 3])
```

```
spy(S)
```

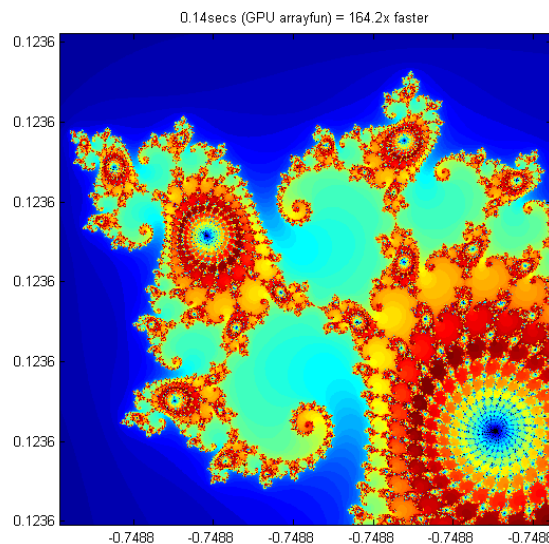
```
S = spalloc(2,3,2)
```

Графика

Пожалуй, максимально удобная для анализа



```
>> clf; % очистить рисунок
>> plot (x,y);
>> title('plot'); % название
>> xlabel('x'); % метки осей
>> ylabel('y');
>> figure % создание нового окна
>> area(x, y); % закрашенный график
>> grid on; % нарисовать сетку
>> figure(1); % вывод в первое
>> hold on % не стирать предыдущее
>> legend('g2', 'x') % легенда
```



bar
scatter
plot

Функция meshgrid

```
[X,Y] = meshgrid(x,y);
```

```
x = (x1, x2, . . . , xm)
```

```
y = (y1, y2, . . . , yn)
```

X=

| | | | |
|-------|-------|-----|-------|
| x_1 | x_2 | ... | x_m |
| x_1 | x_2 | ... | x_m |
| ... | ... | ... | ... |
| x_1 | x_2 | ... | x_m |

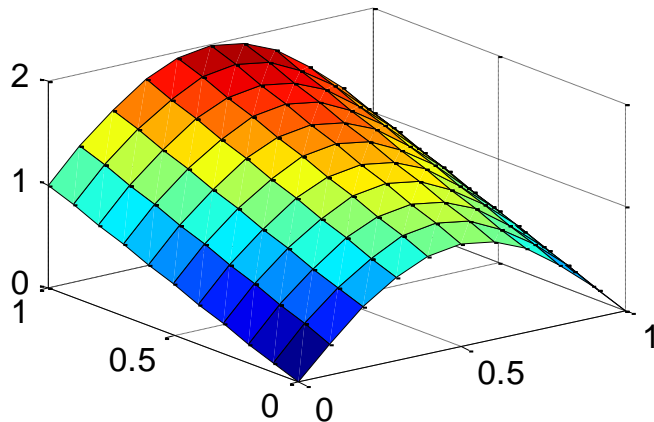
Y=

| | | | |
|-------|-------|-----|-------|
| y_1 | y_1 | ... | y_1 |
| y_2 | y_2 | ... | y_2 |
| ... | ... | ... | ... |
| y_n | y_n | ... | y_n |

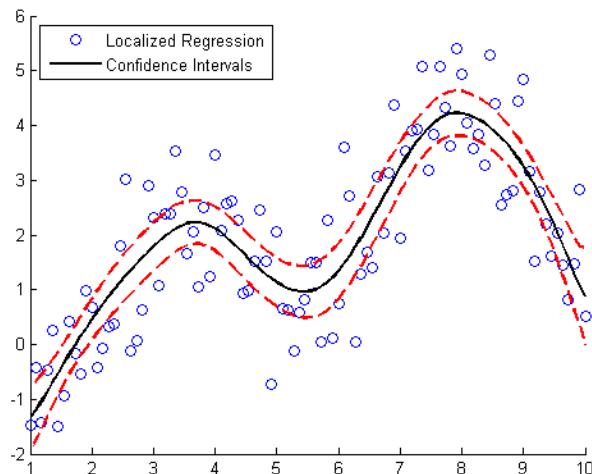
**Вычисление функции во всех точках
вида (x_i, y_j) теперь упрощается:**

```
Z = sin(X.*Y) + abs(X-Y);
```

Графика



```
>> x = 0:0.1:1;  
>> y = linspace(0,1,11); % или y=0:0.1:1  
>> [X,Y] = meshgrid(x,y);  
                                % или [Y,X] = ndgrid(y,x)  
>> Z = sin(X*pi) + Y;  
>> surf(x, y, Z);
```



**Можно сохранять в формате fig,
копировать и вставлять**

**Совет: сохраняйте данные, а не
графики
(+ m-скрипты для получения
графиков)**

Синтаксис очень простой!

```
prms = zeros(10,1);
count = 1;
current = 1;
while true
    current = current + 1;
    if ~isprime(current)
        continue
    end
    % current - простое число
    prms(count) = current;
    if count == 10
        break;
    end
    count = count + 1;
end
prms
```

```
prms =      2      3      5      7      11      13      17
19      23      29
```

Синтаксис

отрицание – ~

или – ||, поэлементное или – | (аналогично &)

Вообще-то циклы стараются не использовать...

```
switch isempty(A)
  case 0
    'непустой'
  case 1
    'пустой'
  otherwise
    '?'
end
```


Логические массивы

одно из средств отказа от циклов

```
>> A = [1 5 3; 4 2 6]
```

```
A =  
    1    5    3  
    4    2    6
```

```
>> L = A>2
```

```
L =  
    0    1    1  
    1    0    1
```

```
>> A(L)'
```

```
ans =  
    4    5    3    6
```

```
>> A(L) = 0
```

```
A =  
    1    0    0  
    0    2    0
```

```
>> find(A>0)
```

```
ans =  
    1  
    4
```

```
>> [i j] = find(A>0)
```

```
i =          j=  
    1          1  
    2          2
```

Стараться обходиться без find

```
>> +(A>0) Зачем + ?
```

Многие функции определены над **double**

Что делает команда

```
+(~~a) ?
```

Логические массивы

`any, all, isprime, logical, true()`

m-файлы

% MPAR сравнение сигналов в 2х метриках

```
function [p1 p2] = mpar(x, y)
x = mmean(x);
y = mmean(y);
p1 = sum(abs(x-y));
p2 = sqrt(sum((x-y).^2));
```

% MMEAN вычесть среднее

```
function x = mmean(x)
x = x - mean(x);
```

```
%{
код для иллюстрации
вложенных функций
%}
```

```
function f = myfunc(a,b,c)
a = a + c; % new
b(1) = b(1) + 1; % new!!!
f = a(c>0) + b(c>0);
```

Переменные a, b меняются, поэтому под них будут созданы новые области памяти, а под переменную c – нет.

Структуры / Анонимные функции

```
>> a.a = 1
a =
    a: 1
```

```
>> a.b = 2
a =
    a: 1
    b: 2
```

```
>> a.c.d = 3
a =
    a: 1
    b: 2
    c: [1x1 struct]
```

```
f = @(x,y) max(sort(x)-sort(y),0)
A = [1 2 3; 4 5 6]
bsxfun(f, A, [2 2]')
```

```
f =
    @(x,y)max(sort(x)-sort(y),0)
```

```
A =
     1     2     3
     4     5     6
```

```
ans =
     0     0     1
     2     3     4
```



Анонимные функции медленные, но упрощают жизнь

```
a = 10; b = 20;  
swap = @(a,b)deal(b,a)  
[a,b] = swap(a,b)
```

```
size1 = @(x)size(x,1)  
rows = size1(zeros(4,5))
```

```
iseven = @(x) ~logical(rem(x,2))
```

```
myExpr = @(x) low < x < high;  
inResult = myExpr(pi)  
outResult = myExpr(17)  
inResult =  
    1  
outResult =  
    1
```

**Поменять две переменные
значениями**

Первая размерность

Чётность

Но тут можно проще...

В чём ошибка?

Массивы ячеек

```
>> c = cell([2 2])
```

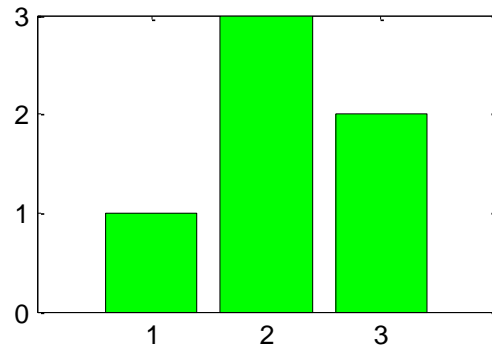
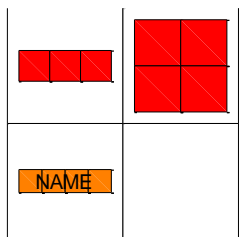
```
c =
     []     []
     []     []
```

```
>> c{1, 1} = [1, 2, 3];
```

```
>> c{1, 2} = eye([2, 2]);
```

```
>> c{2, 1} = 'NAME';
```

```
>> cellplot(c)
```



```
>> arg = {[1 3 2], 'g'}
```

```
arg =
     [1x3 double]     'g'
```

```
>> bar(arg{:})
```

```
>> X = [1 2 3; 4 5 6; 7 8 9];
```

```
>> a = {[1 3], ':'}
```

```
a =
     [1x2 double]     ':'
```

```
>> a{:}
```

```
ans =
     1     3
ans =
     :
```

```
>> X(a{:})
```

Массивы ячеек

+ используются для разных трюков

```
min_and_max = @(x) cellfun(@(f) f(x), {@min, @max});  
y = randi(10, 1, 10)  
just_values      = min_and_max(y)  
[~, just_indices] = min_and_max(y)  
[extrema, indices] = min_and_max(y)
```

```
y =  
    5    10     8    10     7     1     9    10     7     8
```

```
just_values =  
     1    10
```

```
just_indices =  
     6     2
```

```
extrema =  
     1    10
```

```
indices =  
     6     2
```

Строки

```
>> s = '2.34:-4:UN';
>> s(s==':') = ' '
s =
2.34 -4 UN
>> s = strrep(s, 'UN', 'NaN')
s =
2.34 -4 NaN
>> str2num(s)
ans =
    2.3400   -4.0000    NaN
```

```
>> files = dir('*.mat');
>> for I = 1:length(files)
>> thefile = files(i).name
>> % действия с файлом
>> end;

>> for i = 1:9
>> thefile =
        sprintf('mat%d.mat', k);
>> data = load(thefile);
>> end;
```

Задача о универсальной декартовой степени

```
X = [1 5 7];
n = 3;
eval(['[ ' sprintf('A%d ',1:n) ' ] =
        ndgrid( ' repmat('X, ',1,n-1) 'X );']);
[ A1 A2 A3 ] = ndgrid( X, X, X );
eval(['A = [ ' sprintf('A%d(:) ',1:n) ' ]']);
A = [A1(:) A2(:) A3(:) ]
```

```
findstr, strncmp, [s1, s2]
```


Строки

Регулярные выражения

```
mystring = 'ПЗАД - лучший. Люблю Матлаб! А ты?';  
splitstring = regexp(mystring, '(?<=[!.?])\s', 'split');  
disp(splitstring)
```

Вот зачем нужны массивы ячеек

```
'ПЗАД - лучший.'      'Люблю Матлаб!'      'А ты?'
```

Что плохо

Работа с датами

```
>> x = datenum('2012-10-02')
```

```
x =
```

```
735144
```

```
>> datestr(x+1, 'dd/mm/yy')
```

```
ans =
```

```
03/10/12
```

Типы

Между типами есть конвертация

Поэлементные функции

arrayfun, cellfun, spfun, structfun

```
>> a = {'one', 'two', 'tree', [1 2; 3 4]}  
a =  
    'one'    'two'    'tree'    [2x2 double]
```

```
>> cellfun(@length, a)  
ans =  
     3     3     4     2
```

NaN

1. Результат деления на ноль

2. Нет данных

3. Для отделения рядов при выводе графиков

Некоторые функции работают с нанами, некоторые – нет

```
>> any([NaN 0 1])  
ans =  
    1
```

```
>> all([NaN 0 1])  
ans =  
    0
```

```
>> max([NaN 0 1 2])  
ans =  
    2
```

```
>> x = 0/0;  
>> x==x  
ans =  
    0
```

```
>> min([NaN 0 1 2])  
ans =  
    0
```

```
>> mean([NaN 0 1 2])  
ans =  
NaN
```

```
>> nanmean([NaN 0 1 2])  
ans =  
    1
```

Переменная не всегда равна себе!

Эффективное программирование

```
E = []; % надо E = zeros([1 1000]);  
for i=1:1000  
    E(i) = sin(i);  
end;
```

Выделять память заранее

```
data2 = [];  
data3 = [];  
for idx = 1:100  
    data1(idx) = fetchData();  
    data2(end+1) = fetchSomeOtherData();  
    data3 = [ data3 fetchYetMoreData() ];  
end
```

**Пример: как может
увеличиваться массив**

```
function data = unnecessaryPreallocation  
    data = zeros(1,100);  
    data = fetchAllData();  
end
```

**Ошибка новичка:
не всегда надо выделять
память**

Эффективное программирование

```
A = rand(10000);  
B = A';
```

```
tic; C = A+B; toc;
```

```
tic;  
for i=1:10000  
    for j=1:10000  
        C(i,j) = A(i,j)+B(i,j);  
    end;  
end;  
toc;
```

```
Elapsed time is 0.147535 seconds.
```

```
Elapsed time is 9.747009 seconds.
```

**векторные / матричные
вычисления**

Векторизируйте!

Эффективное программирование

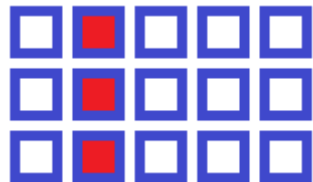
Логические массивы

```
s = 0;
tic;
for i=1:10000
    for j=1:10000
        if (A(i,j)>0.5)
            s = s+1;
        end;
    end;
end;
toc;
s
Elapsed time is 2.083207 seconds.
s =
    50000460

>> tic; sum(A(:)>0.5), toc;
ans =
    50000460
Elapsed time is 0.433364 seconds.
```

Эффективное программирование

```
for i=1:1000
    A(:,i) = func(i) % ...
end;
```



Где в анализе данных это пригодится?

```
x = A\b % не inv(A)*b
x = x(x>0) % не x(find(x>0))
```

Не вызывать лишних функций

Экспериментируйте! Для новых данных – новые законы.

Эффективное программирование

```
>> x = round(rand(1, 1e7));  
>> tic; y = 1./x; toc  
% половина делений на ноль  
Elapsed time is 1.540780  
seconds.
```

```
>> x = x + 1;  
% теперь делений на 0 не будет  
>> tic; y = 1./x; toc  
Elapsed time is 0.329759  
seconds.
```

```
>> a = rand([1 100000]);  
tic; a = a(a>0.5); toc;  
Elapsed time is 0.001209 /  
0.001159 / 0.001159 sec.  
>> a = rand([1 100000]);  
>> tic; a(a<=0.5) = []; toc;  
Elapsed time is 0.001919 /  
/ 0.001872 / 0.001827 sec.
```

Нет тормозным данным!

Типичный приём:

```
x = x / (sqrt(sum(x.^2)) + 0.0001)
```

Всегда есть несколько вариантов!

Эффективное программирование

Используйте bsxfun вместо repmat

Три способа нормировки

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

```
A =
```

```
    1    2
    3    4
```

```
>> A./repmat(sum(A,2),1,2)
```

```
>> bsxfun(@rdivide,A,sum(A,2))
```

```
>> sparse(1:2,1:2,1./sum(A,2))*A
```

```
ans =
```

```
    0.3333    0.6667
    0.4286    0.5714
```

```
n = 10000;
```

```
A = rand(n);
```

```
tic; A./repmat(sum(A,2),1,n); toc
```

```
tic; bsxfun(@rdivide,A,sum(A,2));
```

```
toc
```

```
tic; sparse(1:n,1:n,1./sum(A,2))*A;
```

```
toc
```

```
Elapsed time is 0.359943 sec.
```

```
Elapsed time is 0.234170 sec.
```

```
Elapsed time is 0.394007 sec.
```

```
A = sparse(rand(10000)>0.5);
```

```
tic; A./repmat(sum(A,2),1,n); toc
```

```
tic; bsxfun(@rdivide,A,sum(A,2));
```

```
toc
```

```
tic; sparse(1:n,1:n,1./sum(A,2))*A;
```

```
toc
```

```
Elapsed time is 4.010275 sec.
```

```
Elapsed time is 1.148956 sec.
```

```
Elapsed time is 0.707605 sec.
```

Где в анализе данных это пригодится?

Решение обычного уравнения

```
A = rand([10000 1000]);  
b = rand([10000 1]);  
tic; x = pinv(A)*b; toc  
tic; x = inv(A'*A)*A'*b; toc  
tic; x = inv(A'*A)*(A'*b); toc  
tic; x = (A'*A)\A'*b; toc  
tic; x = (A'*A)\(A'*b); toc  
tic; x = A\b; toc
```

**Всё зависит от расстановки
скобок...**

```
Elapsed time is 2.104123 seconds.  
Elapsed time is 0.439358 seconds.  
Elapsed time is 0.248658 seconds.  
Elapsed time is 0.547900 seconds.  
Elapsed time is 0.192129 seconds.  
Elapsed time is 1.847785 seconds.
```

Где в анализе данных это пригодится?

Ещё советы

- **Используйте функции, а не скрипты**
- **Не меняйте тип переменной**
- **Не балуйтесь разреженными матрицами**
- **Используйте профайлер**
- **Доверяйте анализатору кода**
- **Не используйте `eval`, `evalin`, `andassignin`**
- **Переменная должна появиться слева**
- **Не используйте глобальные переменные**

Когда нестандартные решения

```
>> X(X(:,1)==className,:)

>> X = sortrows(X, 1); % сортировка строк
>> [classes indxF] = unique(X(:,1), 'first');
% классы и их начала
>> [classes indxL] = unique(X(:,1), 'last');
% последние эл-ты классов
>> ic = find(classes==className, 1, 'first');
% номер нашего класса
>> X(indxF(ic):indxL(ic), :) % вывод объектов
```

Часто лучше один раз отсортировать, чем использовать логические массивы

| 1 | 2 | 3 | 4 |
|---|-----|---|----|
| 0 | 0.3 | 3 | 45 |
| 0 | 0.2 | 2 | 23 |
| 0 | 0.3 | 5 | 34 |
| 0 | 0 | 1 | 54 |
| 1 | 1.1 | 2 | 44 |
| 1 | 0.5 | 3 | 6 |
| 1 | 0.7 | 1 | 18 |
| 2 | 1 | 2 | 20 |
| 2 | 1 | 5 | 36 |

Специфика вычислений

```
>> ((1e-15 + 1e-15) + 1e30) - 1e30
```

```
ans =  
      0
```

```
>> (0.3 - 0.1*3)
```

```
ans =  
-5.5511e-017
```

```
>> X = [1.1:0.01:10];
```

```
>> find(S==1.13)
```

```
ans =  
Empty matrix: 1-by-0
```

**Не проверяйте на равенство
вещественные числа!**

Фрагмент кода – оформление

```
for iFile = 1:nFiles
    for jPosition = 1:nPositions

        fileName = getname(); % Имя функции - маленькими буквами
        fileSize = filesize(); % Не надо fsz
        if (checkTiffFormat()) % Не checkTIFFFormat!
            tableNo = % определение номера таблицы
                % ...
            for i = 1:MAX_ITERATIONS % Константа заглавными буквами
                Segment.length = % Название структуры заглавными
                    % Не надо Segment.segmentlength
            end;
        end; % if (checkTiffFormat())

    end; % for jPosition = 1:nPositions
end; % iFile = 1:nFiles

weightedPopulation = (doctorWeight * nDoctors) + ...
                    (lawyerWeight * nLawyers) + ...
                    (chiefWeight * nChiefs);
```

Big Data

- 1. 64-bit Computing. 500 Гб!!!**
- 2. Memory Mapped Variables.** Функция `memmapfile` – работаем с диском, а не памятью.
- 3. Disk Variables.** Функция `matfile` – обращение прямо к mat-файлам.
- 4. Intrinsic Multicore Math.** Многие функции многопоточковые.
- 5. GPU Computing.** Функции в `Parallel Computing Toolbox`
- 6. Parallel Computing.** `Parallel Computing Toolbox/MATLAB Distributed Computing Server` (на кластерах)
- 7. Cloud Computing.** `MATLAB Distributed Computing Server on Amazon's Elastic Computing Cloud (EC2)`
- 8. Distributed Arrays.** `Parallel Computing Toolbox & MATLAB Distributed Computing Server.`
- 9. Streaming Algorithms.**
- 10. Image Block Processing.** Функция `blockproc` – работа с большими изображениями.

Что читать

1. Помощь

2. Полезные блоги

Loren on the Art of MATLAB - MATLAB Central Blogs
blogs.mathworks.com/loren/

3. Учебное пособие

Дьяконов А. Г. Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (практикум на эвм кафедры математических методов прогнозирования). — МАКСПресс, 2010. — 278 с