

CS 2750 Machine Learning

Matlab Tutorial

Content

- Slides prepared by Jeongmin Lee
- based on Matlab tutorial file by Milos Hauskrecht:
<http://people.cs.pitt.edu/~milos/courses/cs2750/Tutorial/>

Outline

- Part 1. Basics of Matlab
- Part 2. Input / Output
- Part 3. Operations
- Part 4. Matrix functions
- Part 5. Special topics

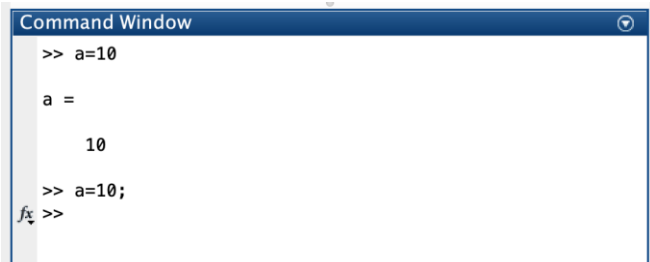
Part 1. Basics of Matlab

- Variable assignment
- IF/ELSE
- Loops
- Pause
- Case
- Script
- Function
- Help

Variable assignment

- `a = 10`
- `a = 10;`

- Assignment with semi colon (;) suppress printing of the result



```

Command Window
>> a=10

a =

    10

>> a=10;
fx >>
  
```

IF/ELSE control

```

if a==4 % equality condition
    b=a^2; % a squared
else
    if a~=10 % not equal
        b=a;
    else
        b=-a;
    end
end
  
```

- If / else / end
- Use indentation to represent a scope

Loops (for)

```
for i=1:5
    i
end
```

- for {var}={start:end index}
 - {var}
- end
- Use indentation

```
Command Window
>> for i=1:5
        i
    end
j=4
i =
     1
i =
     2
i =
     3
i =
     4
i =
     5
```

Loops (while)

```
while j>0
    j=j-1
end
```

```
Command Window
>> while j>0
        j=j-1
    end
j =
     3
j =
     2
j =
     1
j =
     0
```

Case

```
x=input('Value of x') % ask the
input
switch x
    case {2,4}
        'X is even' %%% prints the string
    case {1,3} % Braces!!!
        'X is odd'
    case 0
        'X is zero'
    otherwise
        'Out of range'
end
```

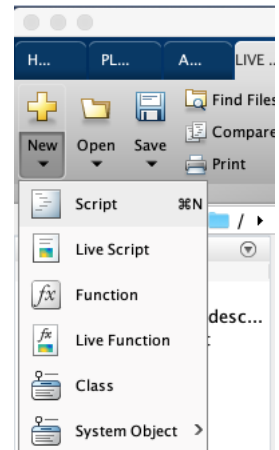
```
>> x=input('Value of x') % asks for the input
switch x
    case {2,4}
        'X is even' %%% prints the string
    case {1,3} % Braces!!!
        'X is odd'
    case 0
        'X is zero'
    otherwise
        'Out of range'
end
Value of x2
x =
    2
ans =
    'X is even'
```

Scripts

- Any syntactically correct sequence of Matlab commands
- It can be executed by specifying the name of the script (files of type xxx.m)

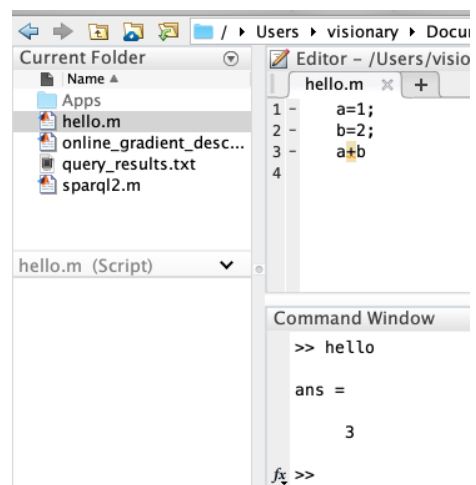
Scripts

- You can create your own new script
 - New -> Script
 - Then, write your own code
 - And save it with “*.m” extension



Scripts

- You can execute any script by calling the name of the script file
 - It should be in the current folder

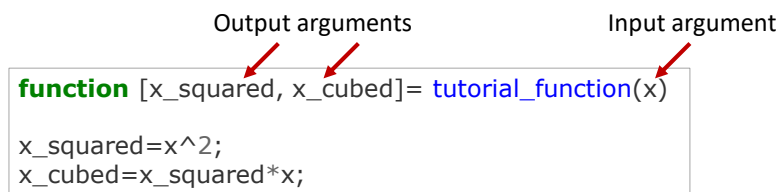


Function

- Function has a specific syntax and should be called
- It can have more than one input and output arguments
- Normally, arguments are transferred by value

Output arguments Input argument

```
function [x_squared, x_cubed]= tutorial_function(x)  
x_squared=x^2;  
x_cubed=x_squared*x;
```

A diagram illustrating the syntax of a MATLAB function. The text 'function [x_squared, x_cubed]= tutorial_function(x)' is shown. Above the first two output arguments, 'x_squared' and 'x_cubed', is the label 'Output arguments'. Above the input argument 'x' is the label 'Input argument'. Red arrows point from these labels to the corresponding parts of the function signature.

Function

- Example
 - Save following content as a new file tutorial_function.m

```
function [x_squared, x_cubed]= tutorial_function(x)  
x_squared=x^2;  
x_cubed=x_squared*x;
```

Function

- Example
 - You can call the function as follows:

```
Command Window
>> [x_sq,x_cube]=tutorial_function(x)

x_sq =

     1

x_cube =

     1
```

Help

- You can get reference about any function with `>> help function_name`

```
Command Window
>> help str2num
str2num Convert character array or string scalar to numeric array
X = str2num(S) converts a character array or string scalar
representation of a matrix of numbers to a numeric matrix. For example,

    S = ['1 2'      str2num(S) => [1 2;3 4]
         '3 4']

The numbers in S should be text representations of a numeric values.
Each number may contain digits, a decimal point, a leading + or - sign,
an 'e' or 'd' preceding a power of 10 scale factor, and an 'i' or 'j' for
a complex unit.

If S does not represent a valid number or matrix, str2num(S) returns
the empty matrix. [X,OK]=str2num(S) returns OK=0 if the conversion fails.

CAUTION: str2num uses EVAL to convert the input argument, so side
effects can occur if S contains calls to functions. Use
STR2DOUBLE to avoid such side effects or when S contains a single
number.

Also spaces can be significant. For instance, str2num('1+2i') and
str2num('1 + 2i') produce x = 1+2i while str2num('1 +2i') produces
x = [1 2i]. These problems are also avoided when you use STR2DOUBLE.

See also str2double, num2str, hex2num, char.

Reference page for str2num
Other functions named str2num
```


Help

- Even a function we defined, we can see its contents
>> **help tutorial_function**

```
Command Window
>> help tutorial_function
tutorial_function is a function.
[x_squared, x_cubed] = tutorial_function(x)
```

Part 2. Inputs and Outputs

- Get user input
- Save results
- Delete all variables
- Load results

Get user input

- `>> x = input('Explanation for user:');`
- Will get user input and assigns it to variable x

Command Window

```
>> x = input('Explanation for user:');  
Explanation for user:100  
>> x  
  
x =  
  
    100
```

Get user input

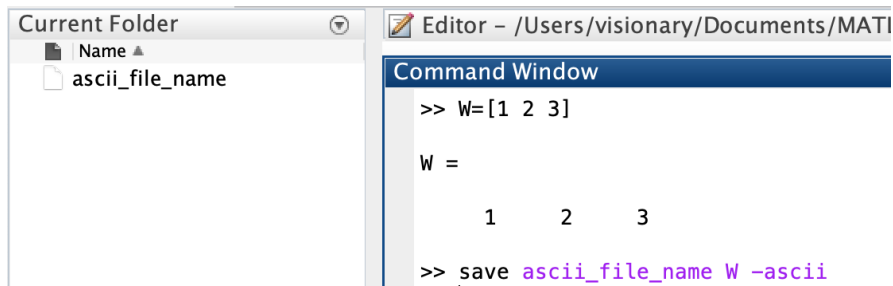
- When you need a **string** as input, add `'s'`
- `>> y = input('Gimme a string:','s');`

Command Window

```
>> y = input('Gimme a string: ','s');  
Gimme a string: hello world  
>> y  
  
y =  
  
    'hello world'
```

Save current results (=variable)

- You can save results to an ascii file
- It (ascii) means, you can read the file from any text editor
- (Lets say W is a variable)
- >> **save** ascii_file_name W **-ascii**



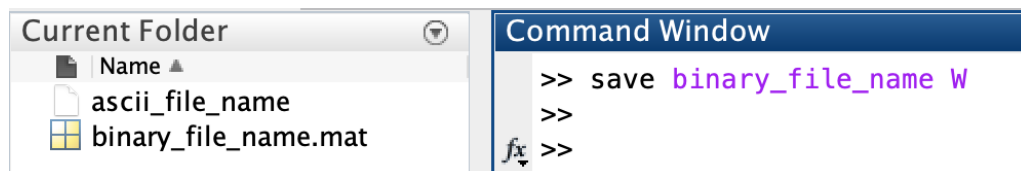
```

Current Folder
├── Name ▲
│   └── ascii_file_name
└── Editor - /Users/visionary/Documents/MATI

Command Window
>> W=[1 2 3]
W =
     1     2     3
>> save ascii_file_name W -ascii
  
```

Save current results (=variable)

- You can also save results into a binary file
- Binary is a Matlab-specific form
- >> **save** binary_file_name **W**



```

Current Folder
├── Name ▲
│   ├── ascii_file_name
│   └── binary_file_name.mat
└── Editor - /Users/visionary/Documents/MATI

Command Window
>> save binary_file_name W
>>
fx >>
  
```

Save current results (=variable)

- To save ALL variables, write only a file name
- >> **save** binary_file_name1

Delete variables

- To delete a variable in the working memory:
- >> **clear** variable_name

- To delete ALL variables in the working memory:
- >> **clear all**

Load variables

- Load all variables from a binary file
- `>> load binary_file_name`

- Load a specific variable from a binary file
- `>> load binary_file_name W`

Load variables

- Load from an ASCII file
- (note that when a data is not in matrix form, you need to write your own program using scanf functions)
- `>> load ascii_file_name`
- `>> W=ascii_file_name`

Part 3. Operations

- Scalar op
- Vector op
- Matrix op
- Vector and constant op
- Matrix and vector

Scalar operations

```
x=13; y=7;
```

```
x-y
```

```
x+y
```

```
x*y
```

```
x/y
```

- Should be straightforward

Command Window

```
>> x=13; y=7;
```

```
x-y
```

```
x+y
```

```
x*y
```

```
x/y
```

```
ans =
```

```
6
```

```
ans =
```

```
20
```

```
ans =
```

```
91
```

```
ans =
```

```
1.8571
```

Scalar operations

```
mod(x,y)
x=12.5
ceil(x)
floor(x)
round(x)
sin(x)
cos(y)
```

- Mod: modulus

Command Window

```
>> mod(x,y)
x=12.5
ceil(x)
floor(x)
round(x)

ans =

     6

x =

    12.5000

ans =

    13

ans =

    12

ans =

    13
```

Vector operations

Creation

```
row_vector=[1 2 3 4]
column_vector=[4 5 6 7]'
column_vector2=[4; 5; 6; 7]
```

- ' : transpose

Command Window

```
>> row_vector=[1 2 3 4]

row_vector =

     1     2     3     4
```

Command Window

```
>> column_vector=[4 5 6 7]'

column_vector =

     4
     5
     6
     7
```

Command Window

```
>> column_vector2=[4; 5; 6; 7]

column_vector2 =

     4
     5
     6
     7
```

Vector operations

Length : gives number of elements in a vector

Size : gives dimensionality

```
>> length(row_vector)
>> length(column_vector)

>> size(row_vector)
>> size(column_vector)
```

```
Command Window
>> length(row_vector)
ans =
     4

>> length(column_vector)
ans =
     4
```

```
Command Window
>> size(row_vector)
ans =
     1     4

>> size(column_vector)
ans =
     4     1
```

Vector operations

Inner product (row * column vector) :

$vec1 * vec2$

- They should have the same length

```
>> d = row_vector * column_vector
```

```
Command Window
>> d = row_vector * column_vector

d =

     60
```


Matrix operations

Creation

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

```
B=[1 2
-1 -2
2 3]
```

```
Command Window
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

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

```
Command Window
>> B=[1 2
-1 -2
2 3]

B =

     1     2
    -1    -2
     2     3
```

Matrix operations

Size

```
size(B) % full size
```

```
size(A,1) % num of rows
```

```
size(A,2) % num of cols
```

```
Command Window
>> size(B)

ans =

     3     2

>> size(B,1)

ans =

     3

>> size(B,2)

ans =

     2
```

Matrix operations

Multiplication

- Matrix multiplication is possible only when $\text{size}(A,2) == \text{size}(B,1)$
- Result is a matrix $\text{size}(A,1) \times \text{size}(B,2)$ A' # of cols == B 's # of rows

```
C=A*B
```

Command Window

```
>> C=A*B
```

```
C =
```

```
     5     7  
    11    16  
    17    25
```

Matrix operations

Select 3rd column of a matrix

```
A3=A(:,3)
```

Command Window

```
>> A3=A(:,3)
```

```
A3 =
```

```
     3  
     6  
     9
```

Matrix operations

Select the first row

```
A1=A(1,:)
```

Command Window

```
>> A1=A(1,:)
```

```
A1 =
```

```
    1    2    3
```

Matrix operations

Select a submatrix

```
AAAA=A(1:2,1:2)
```

Command Window

```
>> AAAA=A(1:2,1:2)
```

```
AAAA =
```

```
    1    2  
    4    5
```

Element-wise op.

Element-wise multiplication

```
D=[1 2 3; 4 5 6]
```

```
E=[1 2 3  
4 5 6]
```

```
F=D.*E
```

Command Window

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

```
D =
```

```
    1    2    3  
    4    5    6
```

```
>> E=[1 2 3  
4 5 6]
```

```
E =
```

```
    1    2    3  
    4    5    6
```

```
>>
```

```
>> F=D.*E
```

```
F =
```

```
    1    4    9  
   16   25   36
```

Element-wise op.

Element-wise division of vectors

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

```
b=[1 3 5 9]
```

```
c=a./b
```

Command Window

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

```
a =
```

```
    1    2    3    4
```

```
>>
```

```
>> b=[1 3 5 9]
```

```
b =
```

```
    1    3    5    9
```

```
>>
```

```
>> c=a./b
```

```
c =
```

```
  1.0000  0.6667  0.6000  0.4444
```

Element-wise op.

Element-wise square

```
D.^2
```

Command Window

```
>> D
D =
     1     2     3
     4     5     6

>> D.^2
ans =
     1     4     9
    16    25    36
```

Vector and constant

v is vector, c is scalar

```
v=[1 2 3 4]
```

```
c=2
```

Command Window

```
>> v=[1 2 3 4]
c=2

v =
     1     2     3     4

c =
     2
```

Vector and constant

When constant is added/subtracted to the vector, it is added / subtracted from **EACH element**

V-C
V+C

Command Window

```
>> v-c
ans =
    -1     0     1     2

>> v+c
ans =
     3     4     5     6
```

Vector and constant

Constant can simply multiply or divide a vector

v/c
v*c

Command Window

```
>> v/c
ans =
    0.5000    1.0000    1.5000    2.0000

>> v*c
ans =
     2     4     6     8
```

Vector and constant

However only $c./v$ is defined and gives $[c/v(1)...c/v(\text{length}(v))]$

```
c./v
```

Command Window

```
>> c/v
Error using ./
Matrix dimensions must agree.

>> c./v
ans =
    2.0000    1.0000    0.6667    0.5000
```

Matrix and vector

For this purpose, we define a **vector** as **2D matrix** with one **dimension** equal to **1**.

```
x=[1 2 3]'
% remember ' means
transpose
```

Command Window

```
>> x=[1 2 3]'
x =
     1
     2
     3

>> size(x)
ans =
     3     1
```

Matrix and vector

Multiplication

```
x=[1 2 3]'
A=[1 -2 3; 0 1 2; 0 0 1]

b=A*x
```

```
Command Window
>> x=[1 2 3]'
A=[1 -2 3; 0 1 2; 0 0 1]

b=A*x

x =

     1
     2
     3

A =

     1    -2     3
     0     1     2
     0     0     1

b =

     6
     8
     3
```

Matrix and vector

Matrix inversion or how to solve a linear system of equations

```
x_solved=inv(A)*b : equation Ax=b; solved x = inv(A)b
```

```
Command Window
>> x_solved=inv(A)*b

x_solved =

     1
     2
     3
```


Matrix and vector

Solving a system of linear equations in another way

`x_solved_another_way=A\b` (\ : backslash)

Command Window

```
>> x_solved_another_way=A\b

x_solved_another_way =

     1
     2
     3
```

Matrix and vector

The application of matrix computation for [linear regression](#)

```
x = rand(1,100)
xDumb = [ones(1,100); x]
A = [1  2]
y = A * xDumb + randn(1,100) * 0.1;
A_solved = y / xDumb
```

- bias and slope parameters of a linear model
- $y=1+2x+\text{random_noise}$
- performs linear regression, least squares solution of overdetermined system!

Matrix and vector

The application of matrix computation for [linear regression](#)

```
x = rand(1,100)
xDumb = [ones(1,100); x]
A = [1 2]
y = A * xDumb + randn(1,100) * 0.1;

A_solved = y / xDumb
```

Command Window

```
>> A_solved = y / xDumb

A_solved =

    1.0123    1.9787
```

Part 4. Matrix functions

- Square roots of elements of a matrix
- Aggregate functions
- Special matrices
- Transform matrices
- Multidimensional arrays
- Structures

Square roots

Square roots of each elements of a matrix

```
F = [ 1  4  9 ; 16 25 36]
```

```
F_ = sqrt(F)
```

Command Window

```
>> F = [ 1  4  9 ; 16 25 36]
```

```
F_ = sqrt(F)
```

```
F =
```

```
    1    4    9
   16   25   36
```

```
F_ =
```

```
    1    2    3
    4    5    6
```

Aggregate: Sum

Aggregate elements by summing operation

```
A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

Command Window

```
>> A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

```
A =
```

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

Aggregate: Sum

Summation performed by **first** index (sums of columns)

```
A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

```
sum(A)
```

Command Window

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

```
ans =
```

```
    15    18    21    24
```

```
A =
```

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

Aggregate: Sum

Summation performed by **second** index (sums of rows)

```
A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

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

Command Window

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

```
ans =
```

```
    10  
    26  
    42
```

```
A =
```

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

Aggregate: Product

Product performed by **second** index (product of rows)

```
A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

```
product(A, 2)
```

Command Window

```
>> prod(A,2)
```

```
ans =
```

```
    24
   1680
  11880
```

```
A =
```

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

Aggregate: Product

Product of the whole matrix

```
A=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
```

```
product(product(A))
```

Command Window

```
>> prod(prod(A))
```

```
ans =
```

```
479001600
```

```
A =
```

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

Aggregate: Means and Standard Deviations

Mean by **first** index (mean of columns)

```
mean_A_column=mean(A)
```

Command Window

```
>> mean_A_column=mean(A)
```

```
mean_A_column =
```

```
5    6    7    8
```

A =

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

Aggregate: Means and Standard Deviations

Mean by **second** index (mean of rows)

```
mean_A_row=mean(A,2)
```

Command Window

```
>> mean_A_row=mean(A,2)
```

```
mean_A_row =
```

```
2.5000
```

```
6.5000
```

```
10.5000
```

A =

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

Aggregate: Means and Standard Deviations

Standard deviation by **first** index (s.d. of columns)

```
std_A_column=std(A)
```

Command Window

```
>> std_A_column=std(A)
```

```
std_A_column =
```

```
4 4 4 4
```

```
A =
```

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

Aggregate: Means and Standard Deviations

Standard deviation by **second** index (s.d. of rows)

```
std_A_row=std(A,[],2)
```

↑
[] MUST be here

Command Window

```
>> std_A_row=std(A,[],2)
```

```
std_A_row =
```

```
1.2910
```

```
1.2910
```

```
1.2910
```

```
A =
```

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

Special matrices

Diagonal matrix

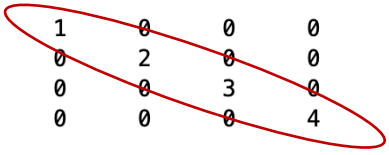
```
diagonal_matrix=diag([1 2 3 4])
```

Command Window

```
>> diagonal_matrix=diag([1 2 3 4])
```

```
diagonal_matrix =
```

```
1 0 0 0  
0 2 0 0  
0 0 3 0  
0 0 0 4
```



Special matrices

Zero matrix

```
all_zeros=zeros(3,4)
```

Command Window

```
>> all_zeros=zeros(3,4)
```

```
all_zeros =
```

```
0 0 0 0  
0 0 0 0  
0 0 0 0
```

Special matrices

One matrix

```
all_ones=ones(4,2)
```

Command Window

```
>> all_ones=ones(4,2)
```

```
all_ones =
```

```
    1    1  
    1    1  
    1    1  
    1    1
```

Special matrices

Unit matrix

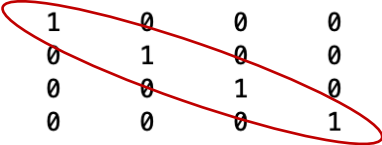
```
unit_matrix=eye(4)
```

Command Window

```
>> unit_matrix=eye(4)
```

```
unit_matrix =
```

```
    1    0    0    0  
    0    1    0    0  
    0    0    1    0  
    0    0    0    1
```



Transform of matrix

Matrix (array) repetition

```
v=[1 2 3];
```

```
V1= repmat(v,3,1)
```

Number of col repetition

Number of row repetition

Command Window

```
>> v=[1 2 3];
>> V1=repmat(v,3,1)
```

```
V1 =
```

```
1     2     3
1     2     3
1     2     3
```

Transform of matrix

Matrix (array) repetition

```
v=[1 2 3];
```

```
V2= repmat(v,1,3)
```

Number of col repetition

Number of row repetition

Command Window

```
>> V2=repmat(v,1,3)
```

```
V2 =
```

```
Columns 1 through 7
```

```
1     2     3     1     2     3     1
```

```
Columns 8 through 9
```

```
2     3
```

Transform of matrix

Matrix (array) repetition

```
v_tran=v';
vv_tran= repmat(v_tran,3,1)
```

Command Window

```
>> v_tran=v';
vv_tran=repmat(v_tran,3,1)

vv_tran =

     1
     2
     3
     1
     2
     3
     1
     2
     3
```

Transform of matrix

Matrix (array) reshape

```
A=[1 2 3 4 5 6 7 8 9 10 11 12];

A_matrix=reshape(A,3,4)
```

Command Window

```
>> A=[1 2 3 4 5 6 7 8 9 10 11 12];
A_matrix=reshape(A,3,4)

A_matrix =

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

$3*4=length(A)$, i.e. total number of elements must not change

Transform of matrix

Multidimensional arrays

```
MX=zeros(3,3,3)
```

Command Window

```
>> MX=zeros(3,3,3)
```

```
MX(:,:,1) =
```

```
    0    0    0
    0    0    0
    0    0    0
```

```
MX(:,:,2) =
```

```
    0    0    0
    0    0    0
    0    0    0
```

```
MX(:,:,3) =
```

```
    0    0    0
    0    0    0
    0    0    0
```

Transform of matrix

Multidimensional arrays

```
A=[1 2 3 4 5 6 7 8 9 10 11 12];
```

```
MX=reshape(A,2,3,2)
```

Command Window

```
>> A=[1 2 3 4 5 6 7 8 9 10 11 12];
```

```
>> MX=reshape(A,2,3,2)
```

```
MX(:,:,1) =
```

```
    1    3    5
    2    4    6
```

```
MX(:,:,2) =
```

```
    7    9   11
    8   10   12
```

$2*3*2=\text{length}(A)$, i.e. total number of elements must not change

Structures

Structure is an object with accessible attributes by its name

```
weather=struct('temp',72,'rainfall',0.0)
```

```
weather.temp
```

Command Window

```
>> weather=struct('temp',72,'rainfall',0.0)

weather =

  struct with fields:

    temp: 72
    rainfall: 0

>> weather.temp

ans =

    72
```

Structures

Initialize 3rd element of weather array

```
weather(3)=struct('temp',72,'rainfall',0.0)
```

Command Window

```
>> weather(3)=struct('temp',72,'rainfall',0.0);
>> weather(1)

ans =

  struct with fields:

    temp: []
    rainfall: []

>> weather(2)

ans =

  struct with fields:

    temp: []
    rainfall: []

>> weather(3)

ans =

  struct with fields:

    temp: 72
    rainfall: 0
```

Structures

creates weather matrix with the same initial values

```
weather= repmat(struct('temp', 72, 'rainfall', 0.0), 1, 3)
```

```
Command Window
>> weather = repmat(struct('temp',72,'rainfall',0.0),1,3);
>> weather(1)

ans =

    struct with fields:

        temp: 72
        rainfall: 0

>> weather(2)

ans =

    struct with fields:

        temp: 72
        rainfall: 0

>> weather(3)

ans =

    struct with fields:

        temp: 72
        rainfall: 0
```

Cell Structures

Cell can contain multiple elements with different types

```
A = {[1 4 3; 0 5 8; 7 2 9], 'Anne Smith'; 3+7i, -pi:pi/4:pi}
```

```
Command Window
>> A = {[1 4 3; 0 5 8; 7 2 9], 'Anne Smith'; 3+7i, -pi:pi/4:pi};
>> A

A =

    2x2 cell array

    {3x3 double      }    {'Anne Smith'}
    {[3.0000 + 7.0000i]}    {1x9 double  }
```

Cell Structures

`A(1)`: access cell element

`A{1}`: returns the **content** of a cell element

```
Command Window
>> A(1)
ans =
    1x1 cell array
    {3x3 double}
>> A{1}
ans =
     1     4     3
     0     5     8
     7     2     9
>> A{1}(1,1)
ans =
     1
```

Part 5. Special topics

- How to plot results
- Find function
- Sort function
- Random number generators and histograms
- Random matrices

Plot Results

Prepare data

```
A=[1 2 3 4 5 7 9];  
B=[12 13 14 14 11 10 8];
```

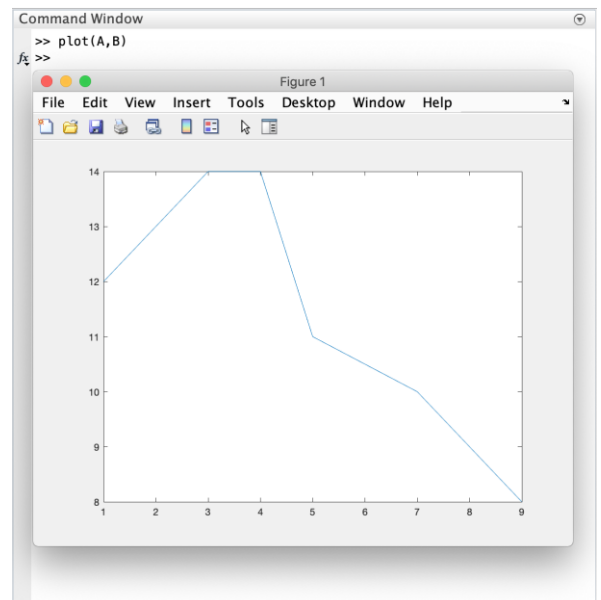
Command Window

```
>> A=[1 2 3 4 5 7 9];  
B=[12 13 14 14 11 10 8];
```

Plot Results

Plot two vectors in same length

```
plot(A,B);
```

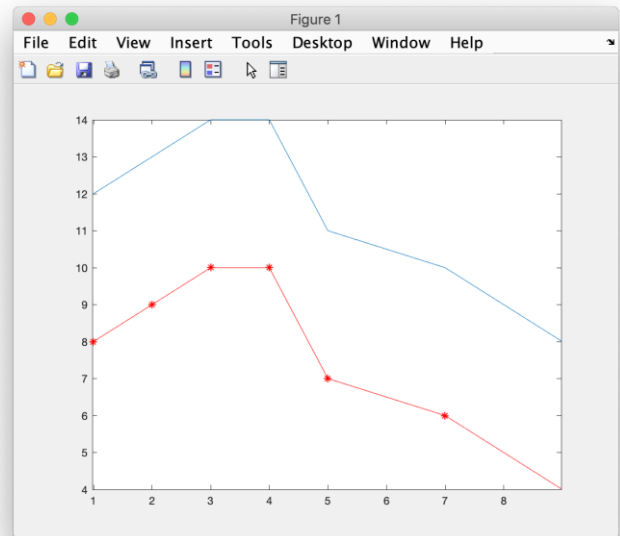


Plot Results

Retain previous figure and
add new graph on it

```
C=B-4;  
Hold on; ← Retain old graph  
plot(A,C, '*-r')
```

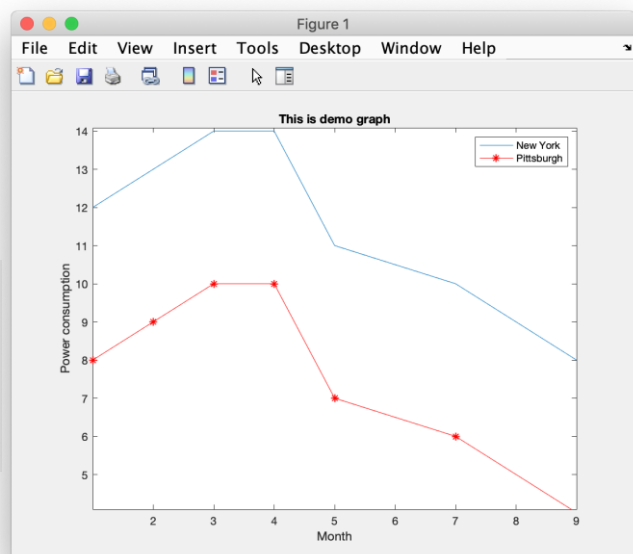
New line with read color and has stars and lines



Plot Results

Add label, title, and legend

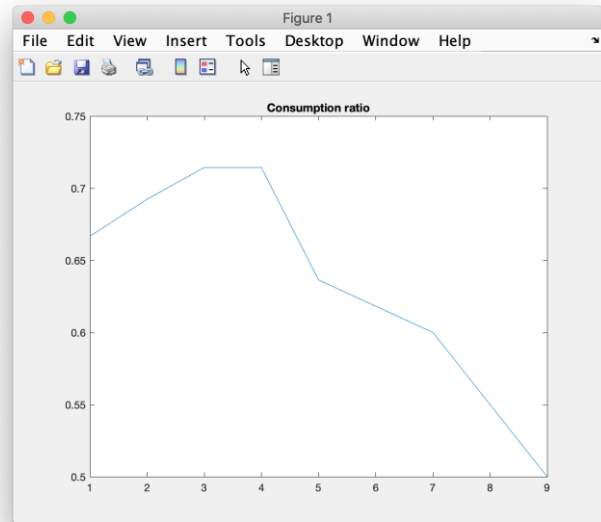
```
xlabel('Month')  
ylabel('Power consumption')  
title('This is demo graph')  
legend('New York','Pittsburgh')
```



Plot Results

Create new plot (hold off)

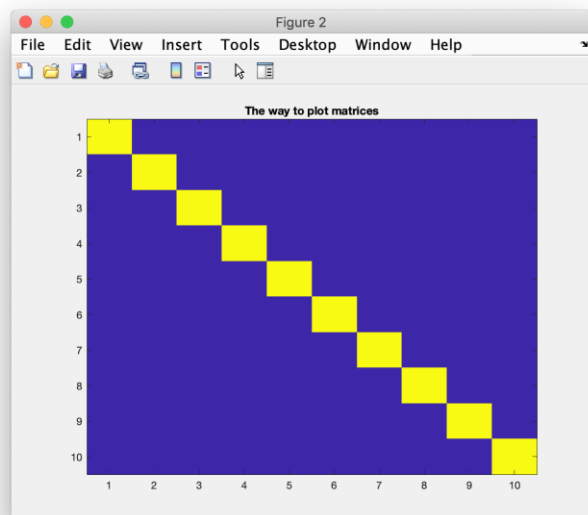
```
hold off;  
plot(A,C./B);  
title('Consumption ratio')
```



Plot Results

Open new plot (figure;)

```
figure;  
C=eye(10,10);  
imagesc(C);  
title('The way to plot matrices')
```

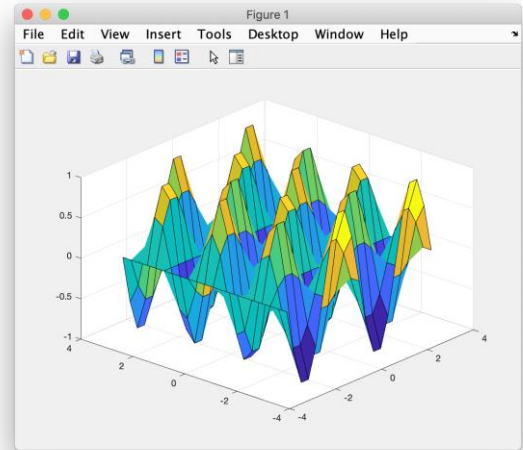


imagesc: display image (matrix) with scaled colors

Plot Results

Let's plot a 3D plot

```
[X Y]=meshgrid(-pi:pi/10:pi,-pi:pi/10:pi);
Z=sin(2*X).*cos(3*Y);
surf(X,Y,Z);
```

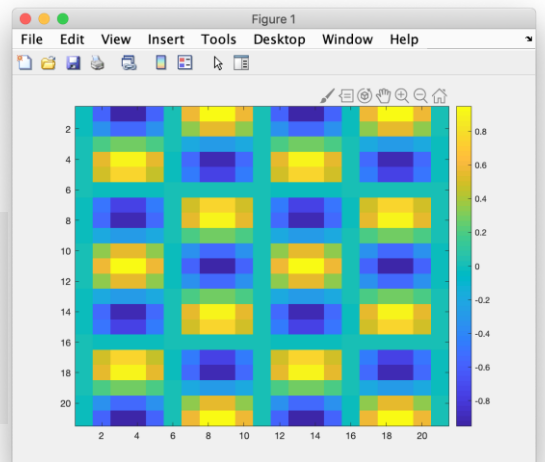


meshgrid: grid to compute function in even intervals
surf: plots 3d function

Plot Results

2D plot of the same function

```
imagesc(Z);colorbar
```



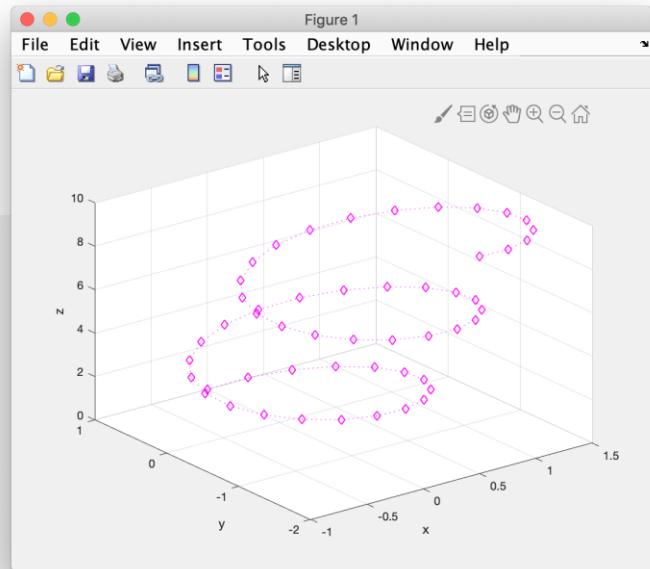
Plot Results

Plot 3D trajectory

```
t=0:0.1:5;
x=sin(pi*t)+0.1*t;
y=cos(pi*t)-0.2*t;
z=2*t;
```

```
plot3(x,y,z,'d:m');
grid on
xlabel('x')
ylabel('y')
zlabel('z')
```

shows magenta diamonds in addition to dotted line



Find

Picks components satisfying a condition

```
a=[ 1 2 3 4 5];
find(a>2)
```

```
Command Window
>> a=[ 1 2 3 4 5];
find(a>2)
ans =
     3     4     5
```

Outputs are **indices** that correspond to values of vector that satisfy the predicate ($a>2$)

Find and Replace

e.g., Replaces all values larger than 0.6 with 1

```
some_probability_vector=[0.2 0.7 0.4 0.6 0.12 0.44 0.72];
```

```
some_probability_vector(find(some_probability_vector>0.6))=1
```

Command Window

```
>> some_probability_vector=[0.2 0.7 0.4 0.6 0.12 0.44 0.72];
```

```
some_probability_vector(find(some_probability_vector>0.6))=1
```

```
some_probability_vector =
```

```
    0.2000    1.0000    0.4000    0.6000    0.1200    0.4400    1.0000
```

Sort function

Let's say we have students ids and corresponding scores

```
scores=[94 55 23 12 10];
```

```
students=[10223 234324 234345 1223 232];
```

Sort function

Student points sorted according to the results on test

```
[scores, index]=sort(scores);
```

Command Window

```
>> scores
scores =
    94    55    23    12    10
>> [scores, index]=sort(scores);
>> scores
scores =
    10    12    23    55    94
>> index
index =
     5     4     3     2     1
```

Sort function

Reorder student ids by the index from sort function

```
B=B(index)
```

Command Window

```
>> B=B(index)
B =
    12    13    14    14    11
```

Sort function

Let's sort rows or columns of matrices

```
A=[ 1 2 3; 10 9 8; 5 6 7]
```

Command Window

```
>> A=[ 1 2 3; 10 9 8; 5 6 7]
```

```
A =
```

1	2	3
10	9	8
5	6	7

Sort function

Each column is independently sorted

A =

1	2	3
10	9	8
5	6	7

```
sort(A,1)
```

Command Window

```
>> sort(A,1)
```

```
ans =
```

1	2	3
5	6	7
10	9	8

Sort function

Each row is independently sorted

A =

1	2	3
10	9	8
5	6	7

```
sort(A,2)
```

Command Window

```
>> sort(A,2)
```

```
ans =
```

1	2	3
8	9	10
5	6	7

Random number generator

Random vectors from Normal
(Gaussian) distribution

```
normal_vector=randn(1000,1);
```

↑ ↑
Number of col
Number of row

Command Window

```
>> normal_vector=randn(1000,1);  
>> normal_vector
```

```
normal_vector =
```

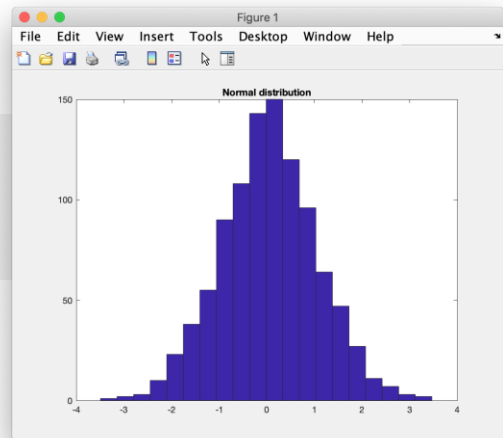
```
-0.6086  
-0.7371  
-1.7499  
0.9105  
0.8671  
-0.0799  
0.8985  
0.1837  
0.2908  
0.1129  
0.4400  
0.1017  
2.7873  
-1.1667
```

Random number generator

Create histogram of random numbers

```
normal_vector=randn(1000,1);
hist(normal_vector, 20)
title('Normal distribution')
```

20: number of bins in histogram



Random number generator

Random vectors from Uniform distribution

```
uniform_vector=rand(1000,1);
```

Number of row
Number of col

Command Window

```
>> uniform_vector=rand (1000,1);
>>
>> uniform_vector

uniform_vector =

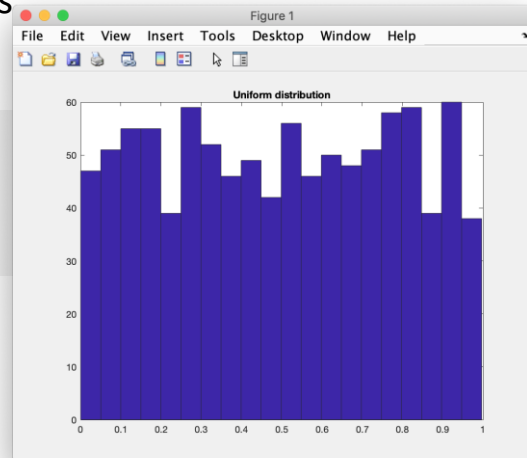
    0.0015
    0.8838
    0.4044
    0.3012
    0.9506
    0.4606
    0.2876
    0.0846
    0.5822
    0.1531
    0.0731
    0.5806
    0.2870
    0.3619
    0.7248
    0.8583
    0.3479
    0.9617
    0.9536
```

Random number generator

Create histogram of random numbers

```
uniform_vector=rand(1000,1);  
hist(uniform_vector,20)  
title('Uniform distribution')
```

20: number of bins in histogram

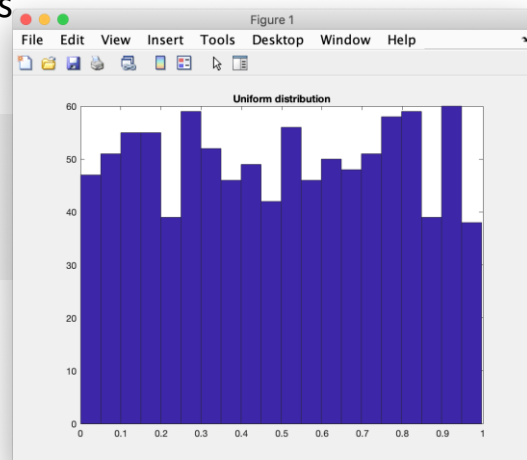


Random number generator

Create histogram of random numbers

```
uniform_vector=rand(1000,1);  
hist(uniform_vector,20)  
title('Uniform distribution')
```

20: number of bins in histogram

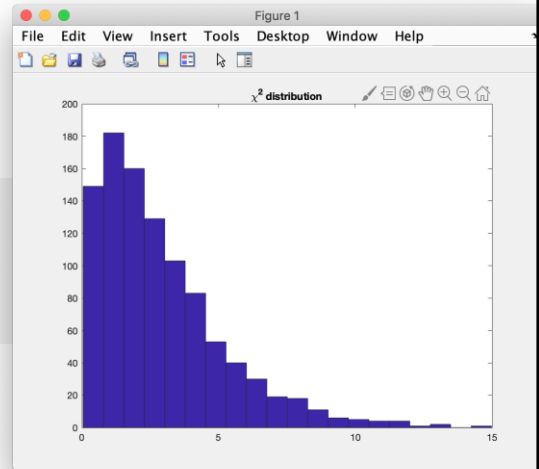


Random number generator

Histogram of random vector from
chi-square distribution

```
chi_square_vector_df3= chi2rnd(3,1000,1)
hist(chi_square_vector_df3, 20)
title('\chi^2 distribution')
```

20: number of bins in histogram



Generate random permutation

```
for i=1:3
    A=randperm(10)
end
```

Command Window

```
>> for i=1:3
    A=randperm(10)
end
```

```
A =
|
| 4 1 3 9 6 2 5 10 7 8
```

```
A =
|
| 3 9 4 10 6 7 1 8 5 2
```

```
A =
|
| 1 4 9 8 5 6 10 3 7 2
```

Generate random matrices

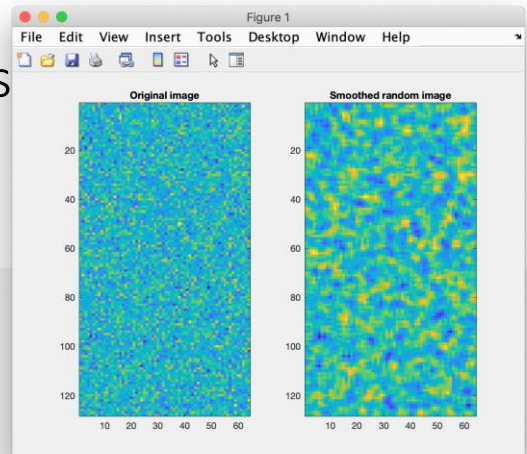
```
A=randn(128,64);
```

```
subplot(1,2,1),imagesc(A),title('Original image')
```

```
B=[1 1 1;1 1 1;1 1 1]/9;
```

```
A=filter2(B,A);
```

```
subplot(1,2,2),imagesc(A),title('Smoothed random image')
```



filter2: function for 2-dimensional filtering (just for illustration, here smooths the image)

Thanks!
-
Questions?