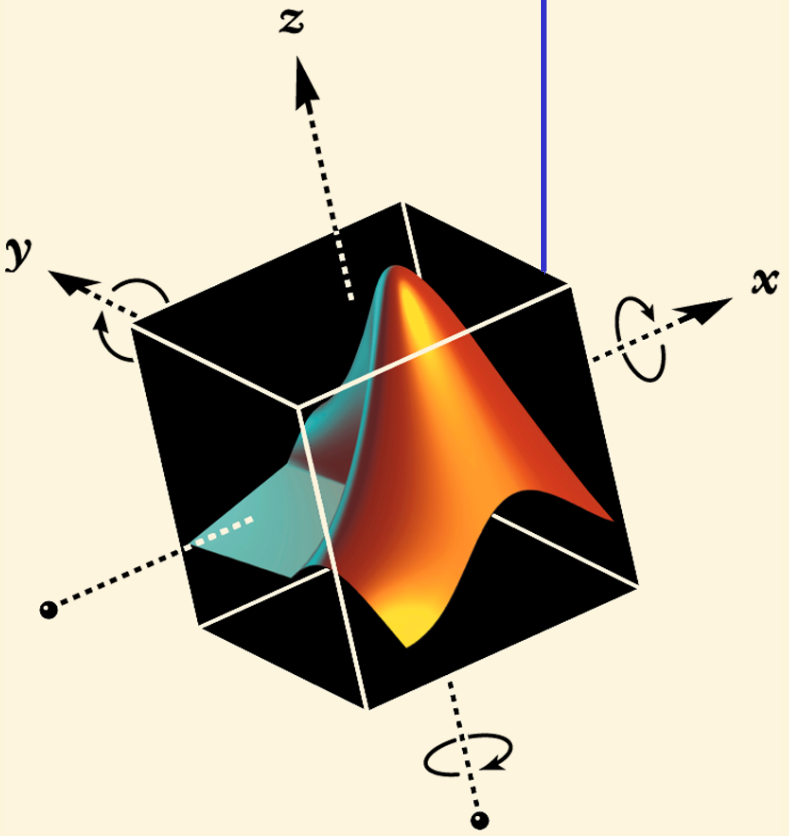
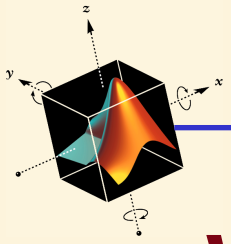
Matlab Basics



Ondrej Lexa

lexa@natur.cuni.cz

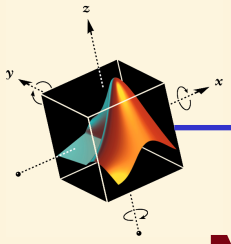


What is Matlab?

A software environment for interactive numerical computations

Examples:

- Matrix computations and linear algebra
- Solving nonlinear equations
- Numerical solution of differential equations
- Mathematical optimization
- Statistics and data analysis
- Signal processing
- Modelling of dynamical systems
- Solving partial differential equations
- And much more ...



Matlab Background

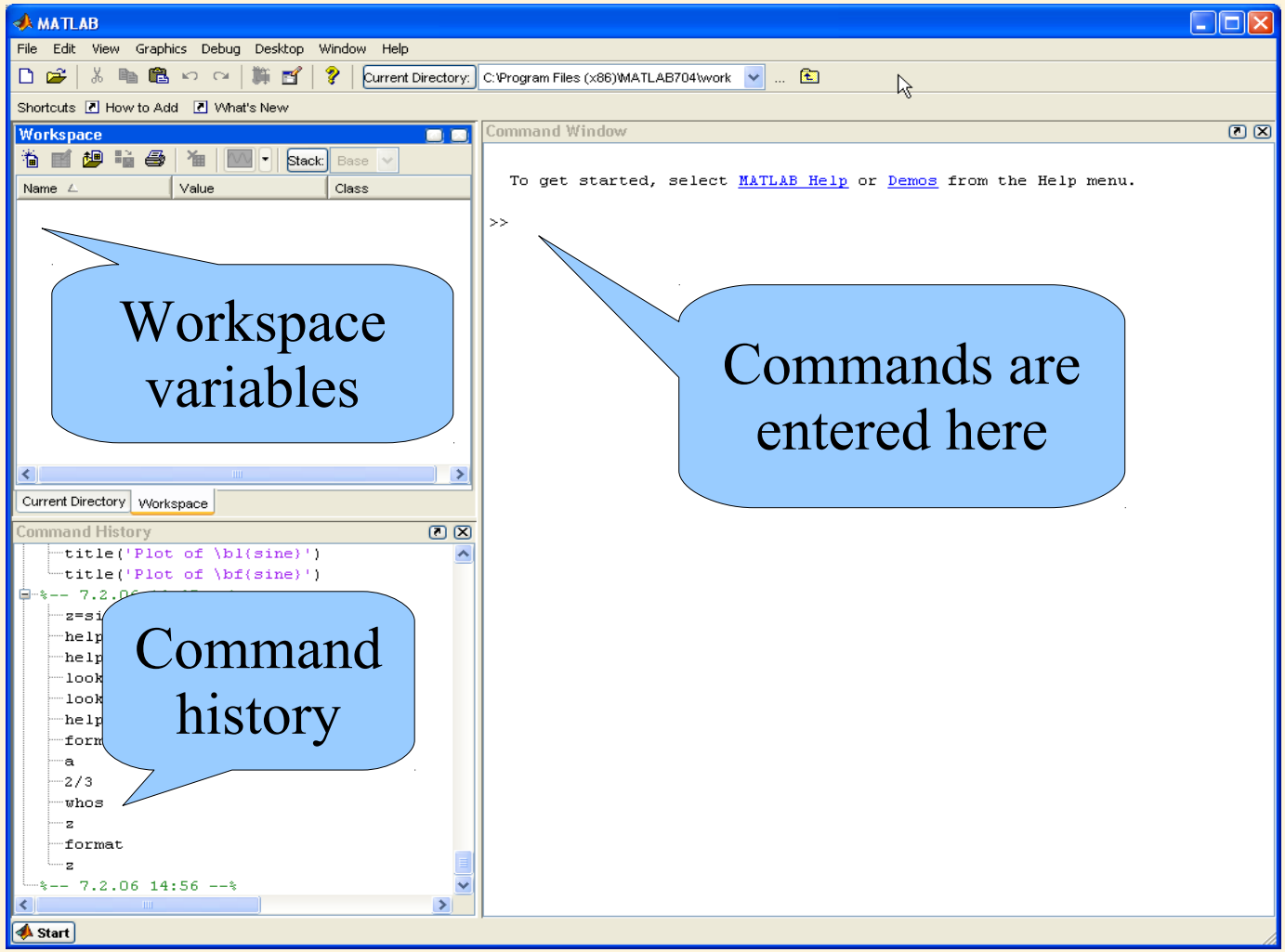
Matlab = **Mat**rix **Lab**oratory

Originally a user interface for numerical linear algebra routines (Lapak/Linpak)

Commercialized 1984 by The Mathworks

Since then heavily extended (defacto-standard)

Matlab environment



Calculations at the Command Line

MATLAB as a calculator

```
» -5/(4.8+5.32)^2
ans =
   -0.0488
» (3+4i)*(3-4i)
ans =
    25
» cos(pi/2)
ans =
  6.1230e-017
» exp(acos(0.3))
ans =
    3.5470
```

Assigning Variables

```
» a = 2;
» b = 5;
» a^b
ans =
    32
» x = 5/2*pi;
» y = sin(x)
y =
     1
» z = sin(pi)
z =
  1.2246e-016
```

Semicolon suppresses screen output

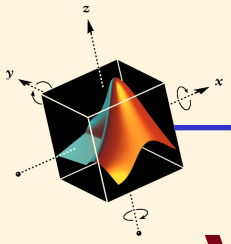
Results assigned to "ans" if name not specified

() parentheses for function inputs

1.2246e-016 ???

Numbers stored in double-precision floating point format

»cmd_line



Variable and Memory Management

Matlab uses double precision (approx. 16 significant digits)

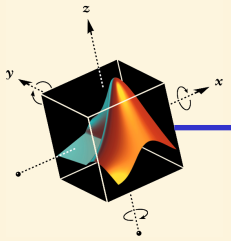
```
>> format long
>> format compact
```

All variables are shown with

```
>> who
>> whos
```

Variables can be stored on file

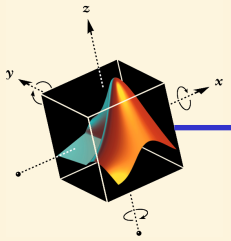
```
>> save filename
>> clear
>> load filename
```



Working with Files & Variables

- CD / PWD, LS / DIR - navigating directories
- WHAT - displays the files within a directory (grouped by type)
- ! - invoke operating system
- WHICH - identifies the object referenced by given name (function / variable)
- CLEAR - remove function / variable from memory
- WHOS - lists workspace variables and details (size, memory usage, data type)
- SIZE - returns the size of matrix

Ref: Utility Commands



The Help System

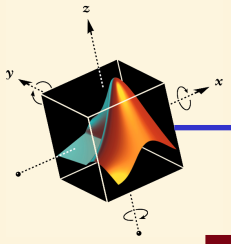
- The help command `>> help`
- The help window `>> helpwin`
- The lookfor command `>> lookfor`

```
» help cd
CD     Change current working directory.
  CD directory-spec sets the current directory to the one specified.
  CD .. moves to the directory above the current one.
  CD, by itself, prints out the current directory.

  WD = CD returns the current directory as a string.

  Use the functional form of CD, such as CD('directory-spec'),
  when the directory specification is stored in a string.

  See also PWD.
```



The Help System

Search for appropriate function

```
>> lookfor keyword
```

Rapid help with syntax and function definition

```
>> help function
```

An advanced hyperlinked help system is launched by

```
>> helpdesk
```

```
Complete manuals as PDF files
```

Vectors and Matrices

Vectors (arrays) are defined as

```
>> v = [1, 2, 4, 5]
>> w = [1; 2; 4; 5]
```

$$v = \begin{bmatrix} 1 & 2 & 4 & 5 \end{bmatrix}$$

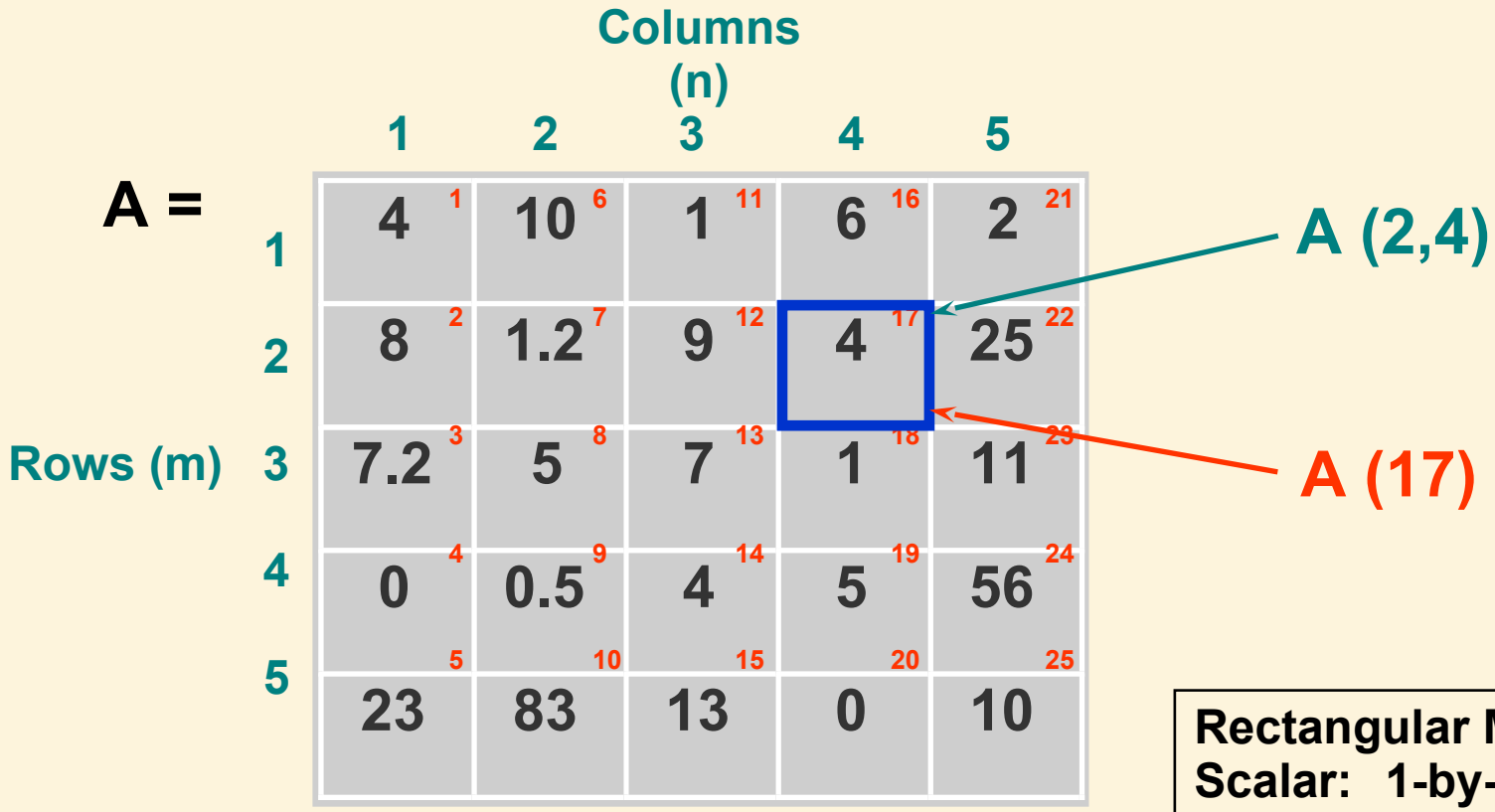
$$w = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}$$

Matrices (2D arrays) defined similarly

```
>> A = [1,2,3;4,-5,6;5,-6,7]
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

The Matrix in MATLAB



Matrix elements can be EITHER numbers OR characters

Rectangular Matrix:
Scalar: 1-by-1 array
Vector: m-by-1 array
1-by-n array
Matrix: m-by-n array

Entering Numeric Arrays

Row separator:
semicolon (;)

Column separator:
space / comma (,)

```
» a=[1 2;3 4]
a =
     1     2
     3     4
» b=[-2.8, sqrt(-7), (3+5+6)*3/4]
b =
   -2.8000    0 + 2.6458i   10.5000
» b(2,5) = 23
b =
   -2.8000   0 + 2.6458i   10.5000    0         0
         0             0         0    0   23.0000
```

Use square brackets [ ]

Matrices must be rectangular.
(Set undefined elements to zero)

Any MATLAB expression can be entered as a matrix element

»num_array1

Entering Numeric Arrays - cont.

Scalar expansion

Creating sequences:
colon operator (:)

Utility functions for creating matrices.
(Ref: Utility Commands)

```
» w=[1 2;3 4] + 5
w =
     6     7
     8     9
» x = 1:5
x =
     1     2     3     4     5
» y = 2:-0.5:0
y =
   2.0000    1.5000    1.0000    0.5000         0
» z = rand(2,4)
z =
   0.9501    0.6068    0.8913    0.4565
   0.2311    0.4860    0.7621    0.0185
```

»num_array2

Numerical Array Concatenation - [ ]

Use [ ] to combine existing arrays as matrix "elements"

Row separator: semicolon (;)

Column separator: space / comma (,)

```
» a=[1 2;3 4]
a =
     1     2
     3     4
» cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
     1     2     2     4
     3     4     6     8
     3     6     4     8
     9    12    12    16
     5    10     6    12
    15    20    18    24
```

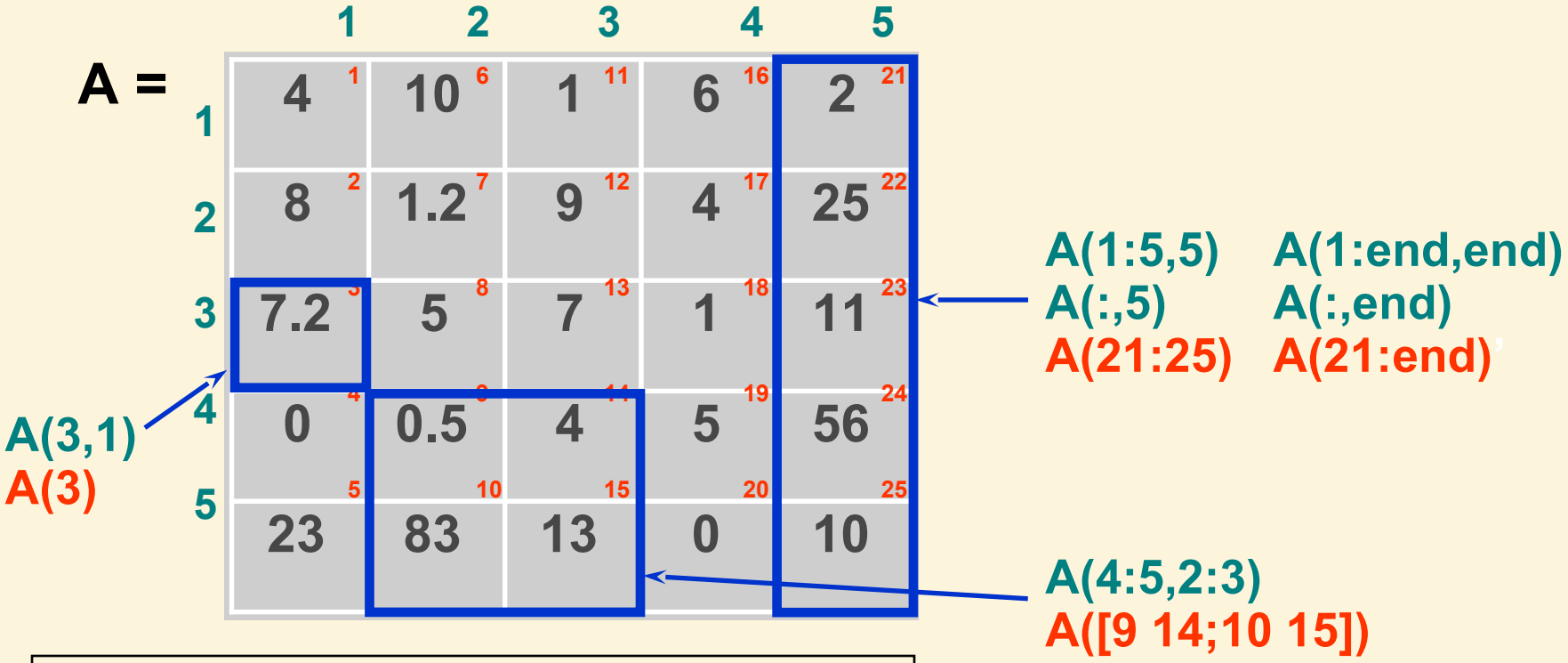
Use square brackets [ ]

4*a

The resulting matrix must be rectangular.

»num_cat

Array Subscripting / Indexing



- Use () parentheses to specify index
- colon operator (:) specifies range / ALL
- [ ] to create matrix of index subscripts
- ‘end’ specifies maximum index value

Generating Vectors from functions

- zeros(M,N) MxN matrix of zeros

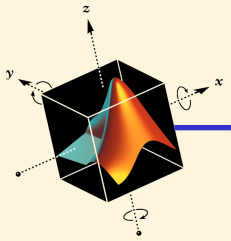
```
x = zeros(1,3)
x =
   0     0     0
```

- ones(M,N) MxN matrix of ones

```
x = ones(1,3)
x =
   1     1     1
```

- rand(M,N) MxN matrix of uniformly distributed random numbers on (0,1)

```
x = rand(1,3)
x =
    0.9501   0.2311   0.6068
```



Operators

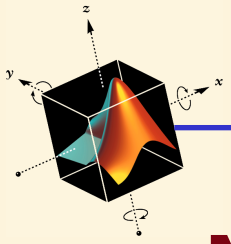
[ ] concatenation

```
x = [ zeros(1,3) ones(1,2) ]
x =
     0   0   0   1   1

x = [ 1  3  5  7  9]
x =
     1   3   5   7   9
```

( ) subscription

```
y = x(2)
y =
     3
y = x(2:4)
y =
     3   5   7
```



Matrix Operators

All common operators are overloaded

```
>> v + 2
```

Common operators are available

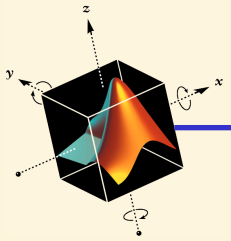
```
>> B = A'
>> A*B
>> A+B
```

Note:

- Matlab is case-sensitive

  `A` and `a` are two different variables
- Transponate conjugates complex entries; avoided by

  ```
  >> B=A.'
  ```



Operators (arithmetic)

+	addition
-	subtraction
*	multiplication
/	division
^	power
'	complex conjugate transpose
.*	element-by-element mult
./	element-by-element div
.^	element-by-element power
.'	transpose

Operators (relational, logical)

==	equal
~=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
&	AND
|	OR
~	NOT

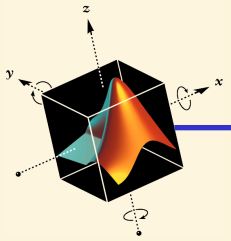
pi 3.14159265…

j imaginary unit

i same as j

```
» Mass = [-2 10 NaN 30 -11 Inf 31];
» all_pos = all(Mass>=0)
all_pos =
     0
» each_pos = Mass>=0
each_pos =
     0     1     0     1     0     1     1
» pos_fin = (Mass>=0)&(isfinite(Mass))
pos_fin =
     0     1     0     1     0     0     1
```

1 = TRUE
0 = FALSE



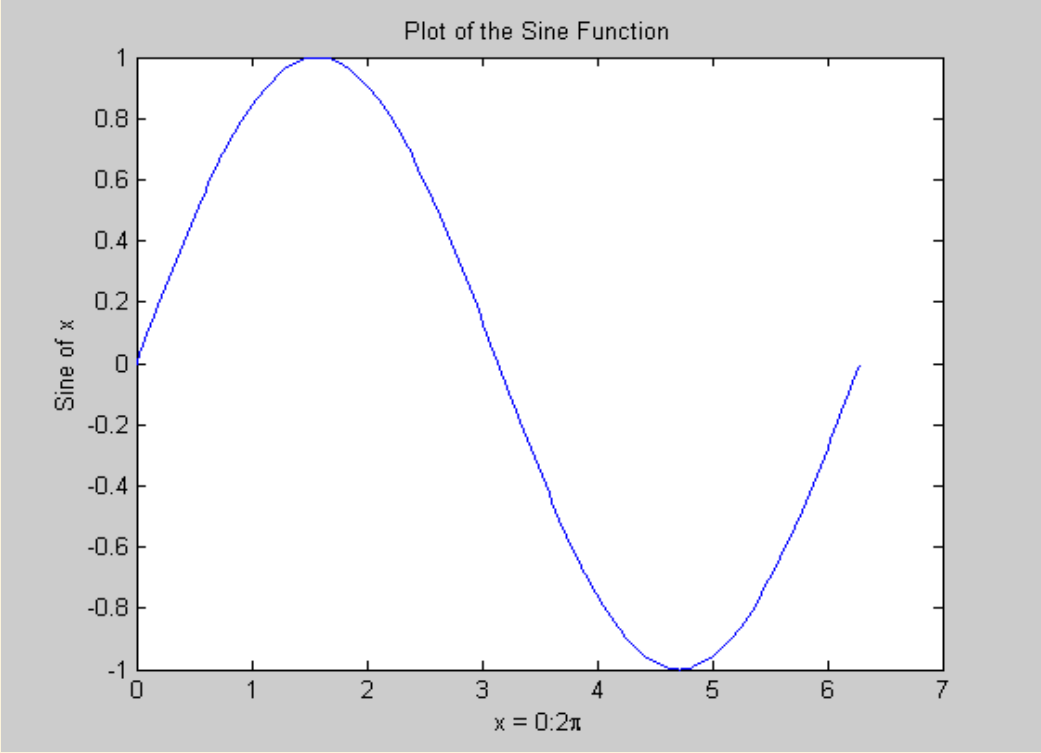
Math Functions

- Elementary functions (sin, cos, sqrt, abs, exp, log10, round)
  - type `help elfun`

- Advanced functions (bessel, beta, gamma, erf)
  - type `help specfun`
  - type `help elmat`

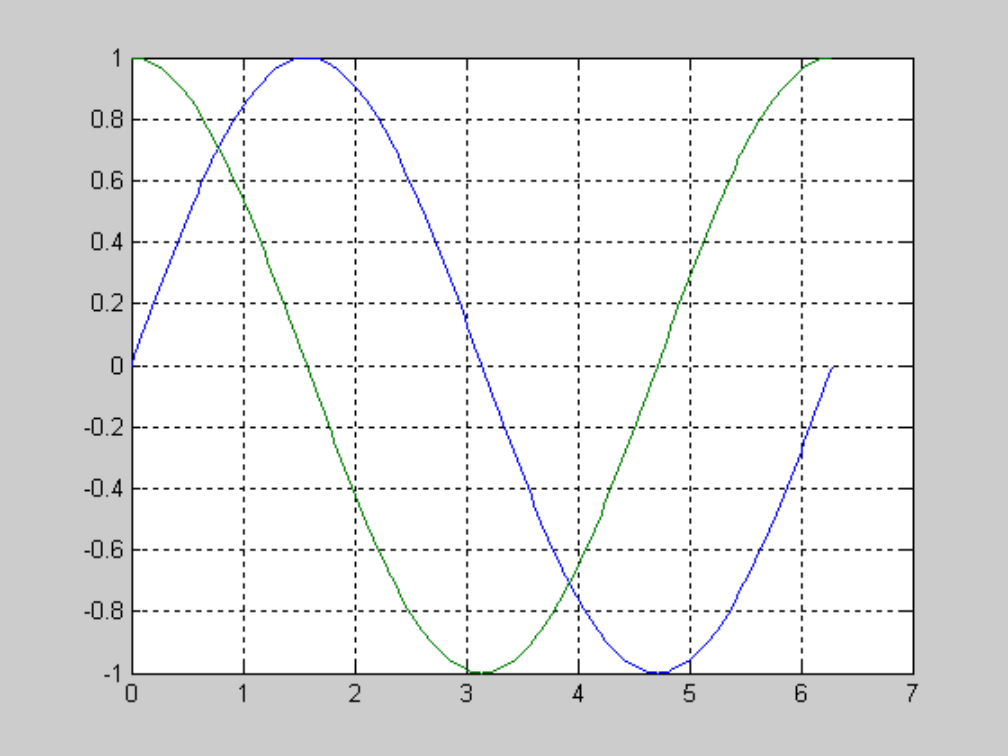
Matlab Graphics

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
xlabel('x = 0:2\pi')
ylabel('Sine of x')
title('Plot of the
   Sine Function')
```



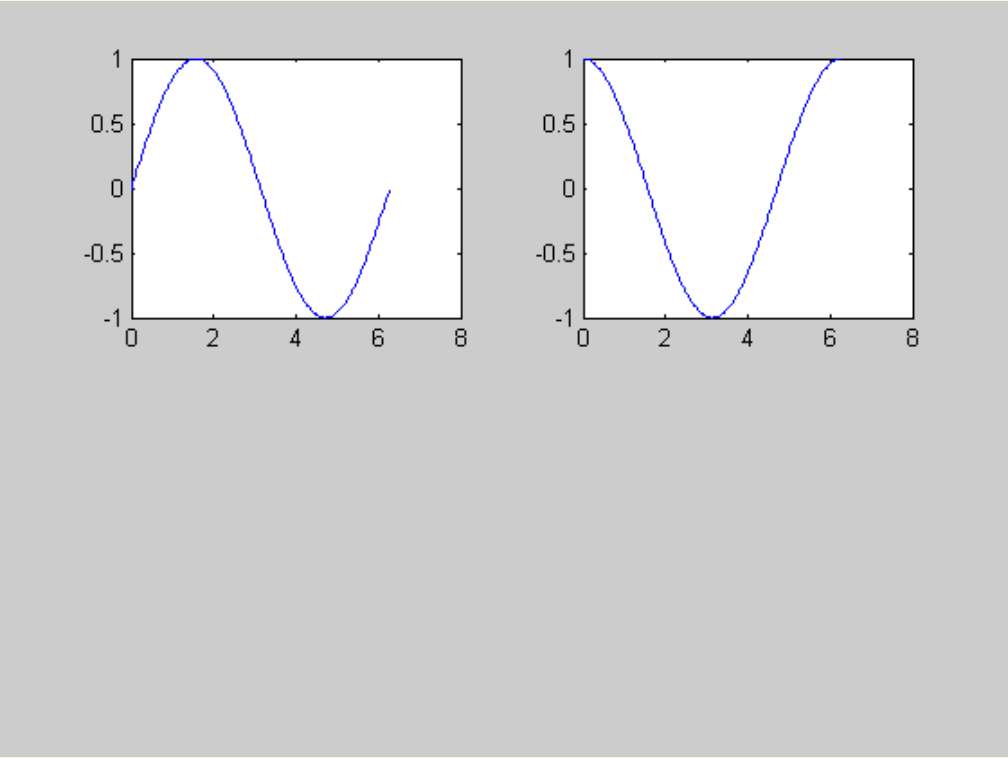
Multiple Graphs

```
t = 0:pi/100:2*pi;
y1=sin(t);
y2=sin(t+pi/2);
plot(t,y1,t,y2)
grid on
```



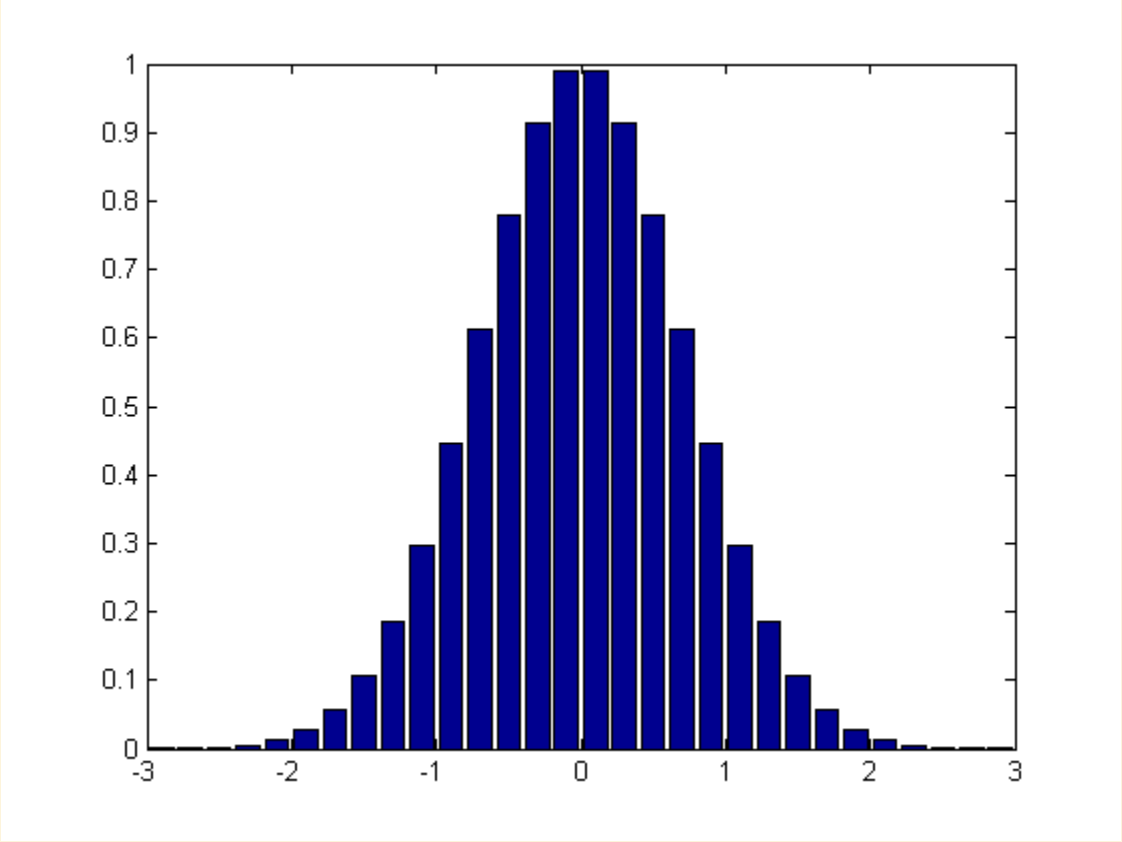
Multiple Plots

```
t = 0:pi/100:2*pi;
y1=sin(t);
y2=sin(t+pi/2);
subplot(2,2,1)
plot(t,y1)
subplot(2,2,2)
plot(t,y2)
```



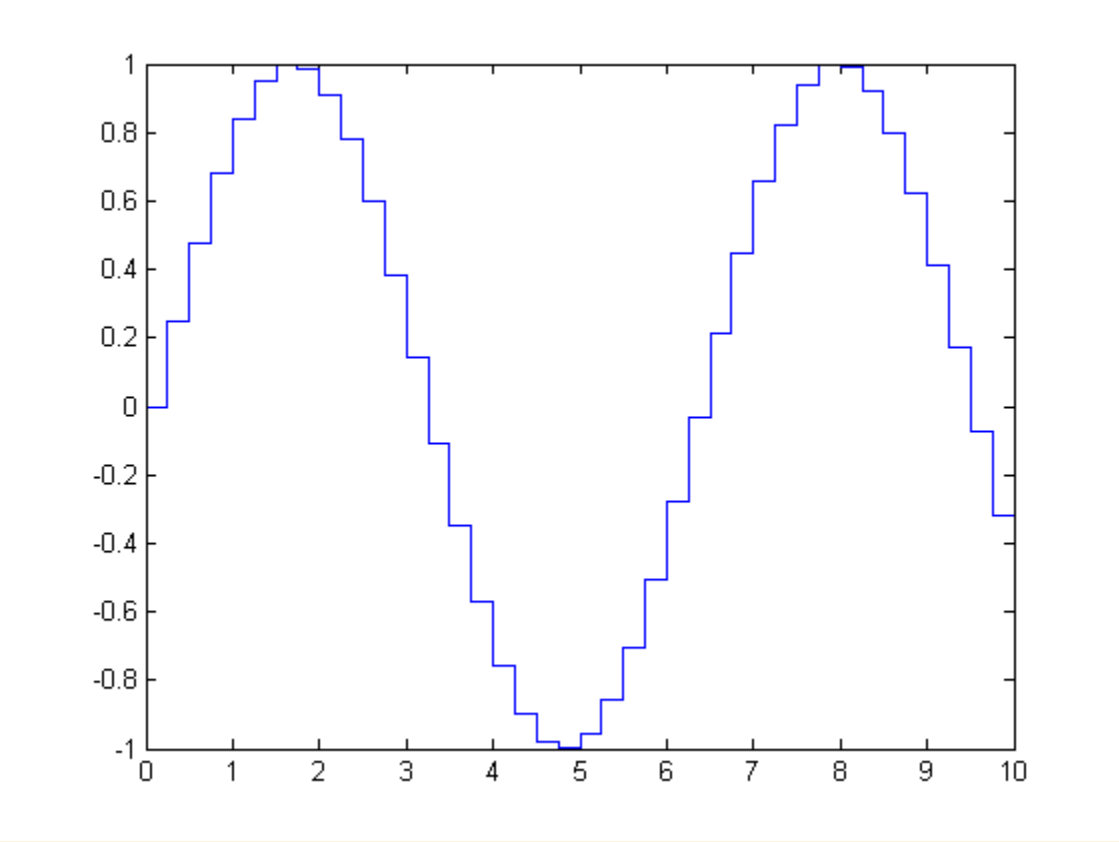
Bar plot of a bell shaped curve

```
x = -2.9:0.2:2.9;
bar(x,exp(-x.*x));
```



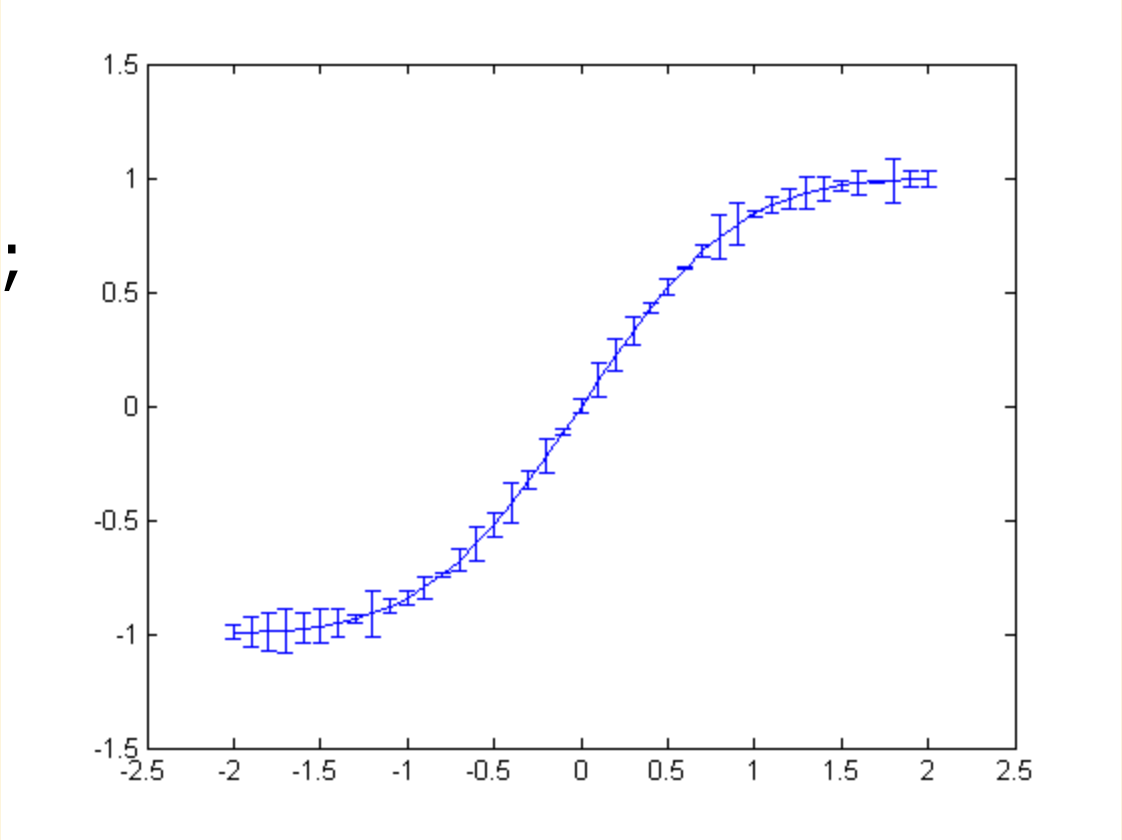
Stairstep plot of a sine wave

```
x=0:0.25:10;
stairs(x,sin(x));
```



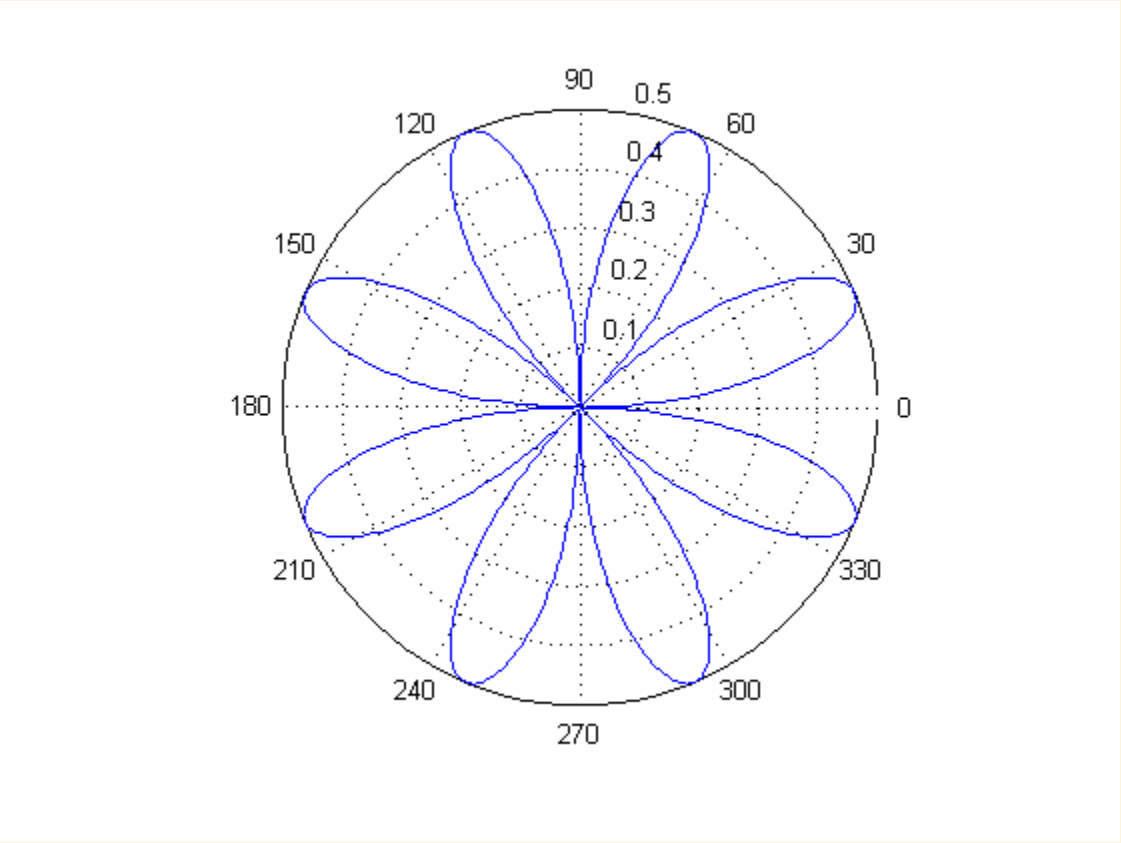
Errorbar plot

```
x=-2:0.1:2;
y=erf(x);
e = rand(size(x))/10;
errorbar(x,y,e);
```



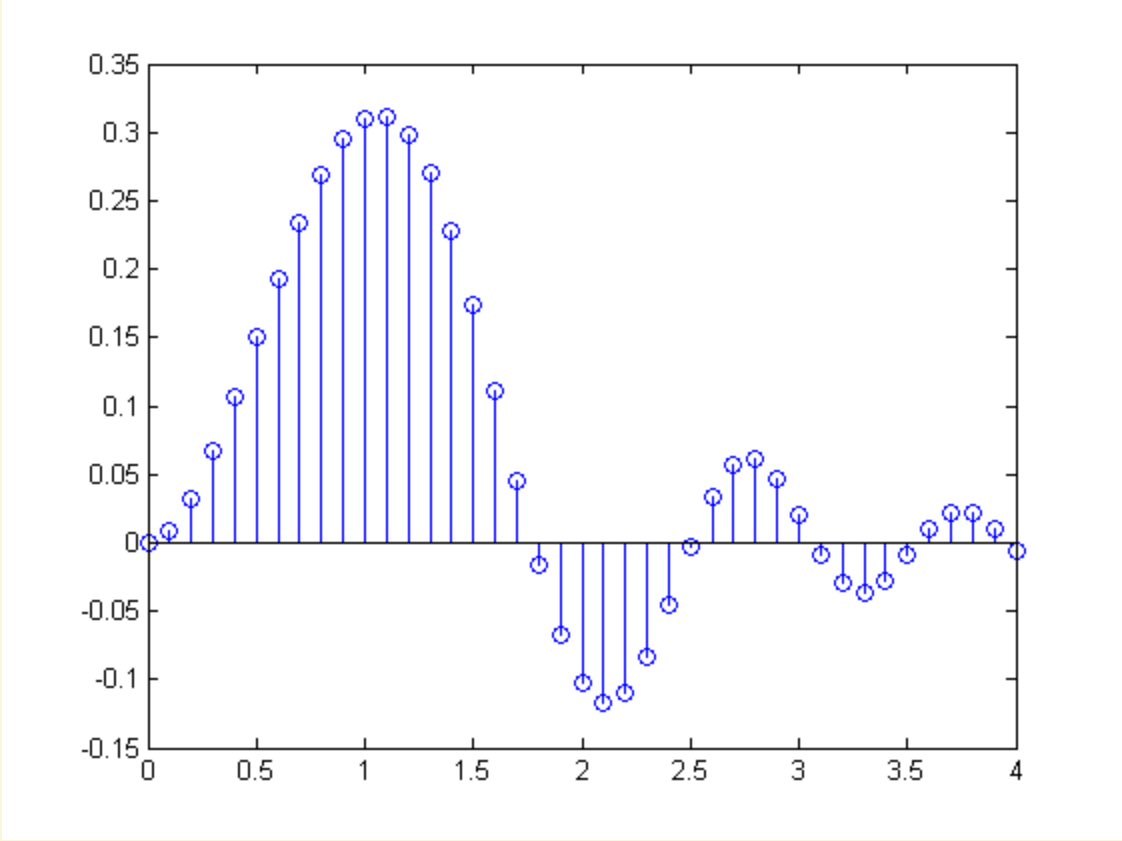
Polar plot

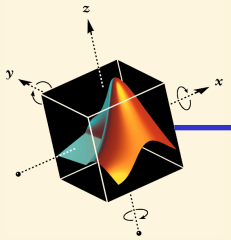
```
t=0:.01:2*pi;
polar(t,abs(sin(2*t).*cos(2*t)));
```



Stem plot

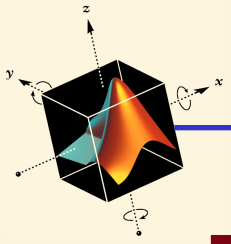
```
x = 0:0.1:4;
y = sin(x.^2).*exp(-x);
stem(x,y)
```





Graph Functions (summary)

• plot	linear plot
• stem	discrete plot
• grid	add grid lines
• xlabel	add X-axis label
• ylabel	add Y-axis label
• title	add graph title
• subplot	divide figure window
• figure	create new figure window
• pause	wait for user response



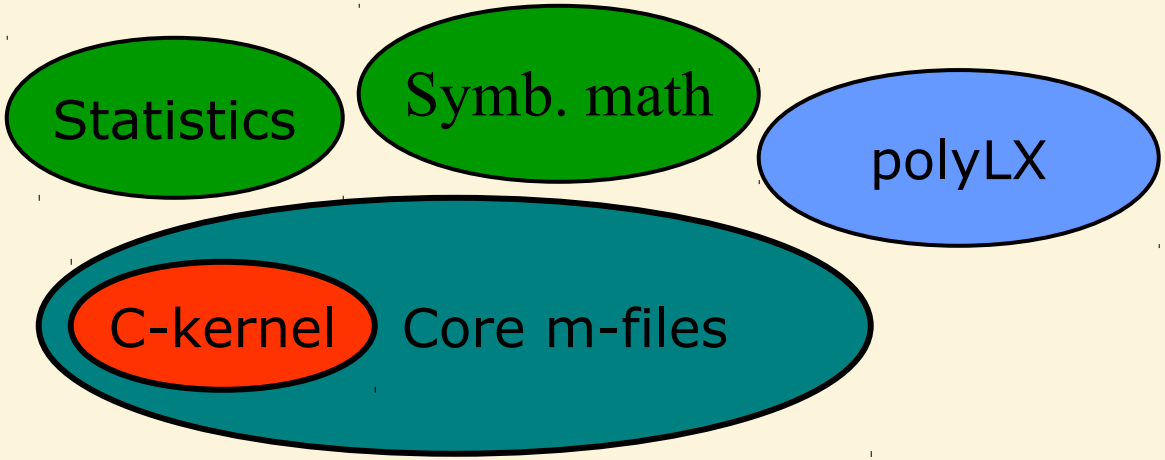
Programming in MATLAB

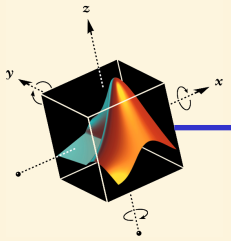
Matlab environment

Matlab construction

- Core functionality as compiled C-code, m-files
- Additional functionality in toolboxes (m-files)

Matlab programming (construct own m-files)





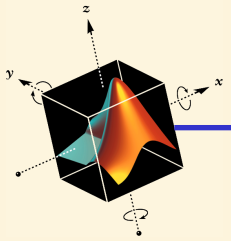
The programming environment

The working directory is controlled by

```
>> dir
>> cd catalogue
>> pwd
```

The path variable defines where matlab searches for m-files

```
>> path
>> addpath
>> pathtool
>> which function
```



The programming environment

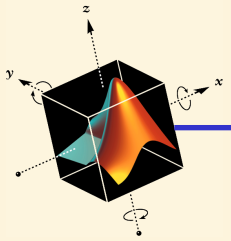
Matlab can't tell if identifier is variable or function

```
>> z=theta;
```

Matlab searches for identifier in the following order

1. variable in current workspace
2. built-in variable
3. built-in m-file
4. m-file in current directory
5. m-file on search path

Note: m-files can be located in current directory, or in path



Script files

Script-files contain a sequence of Matlab commands

factscript.m

```
%FACTSCRIPT – Compute n-factorial, n!=1*2*...*n
y = prod(1:n);
```

- Executed by typing its name

  ```
  >> factscript
  ```

- Operates on variables in global workspace
  - Variable `n` must exist in workspace
  - Variable `y` is created (or over-written)
- Use comment lines (starting with %) to document file!

Script M-files

- Standard ASCII text files
- Contain a series of MATLAB expressions
  (Typed as you would at the command line)
- Commands parsed & executed in order

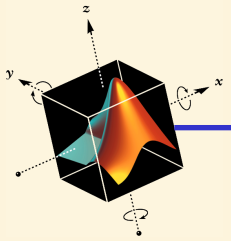
```
% Comments start with "%" character

pause     % Suspend execution - hit any key to continue.

keyboard  % Pause & return control to command line.
          % Type "return" to continue.

break     % Terminate execution of current loop/file.

return    % Exit current function
          % Return to invoking  function/command line.
```



Displaying code and getting help

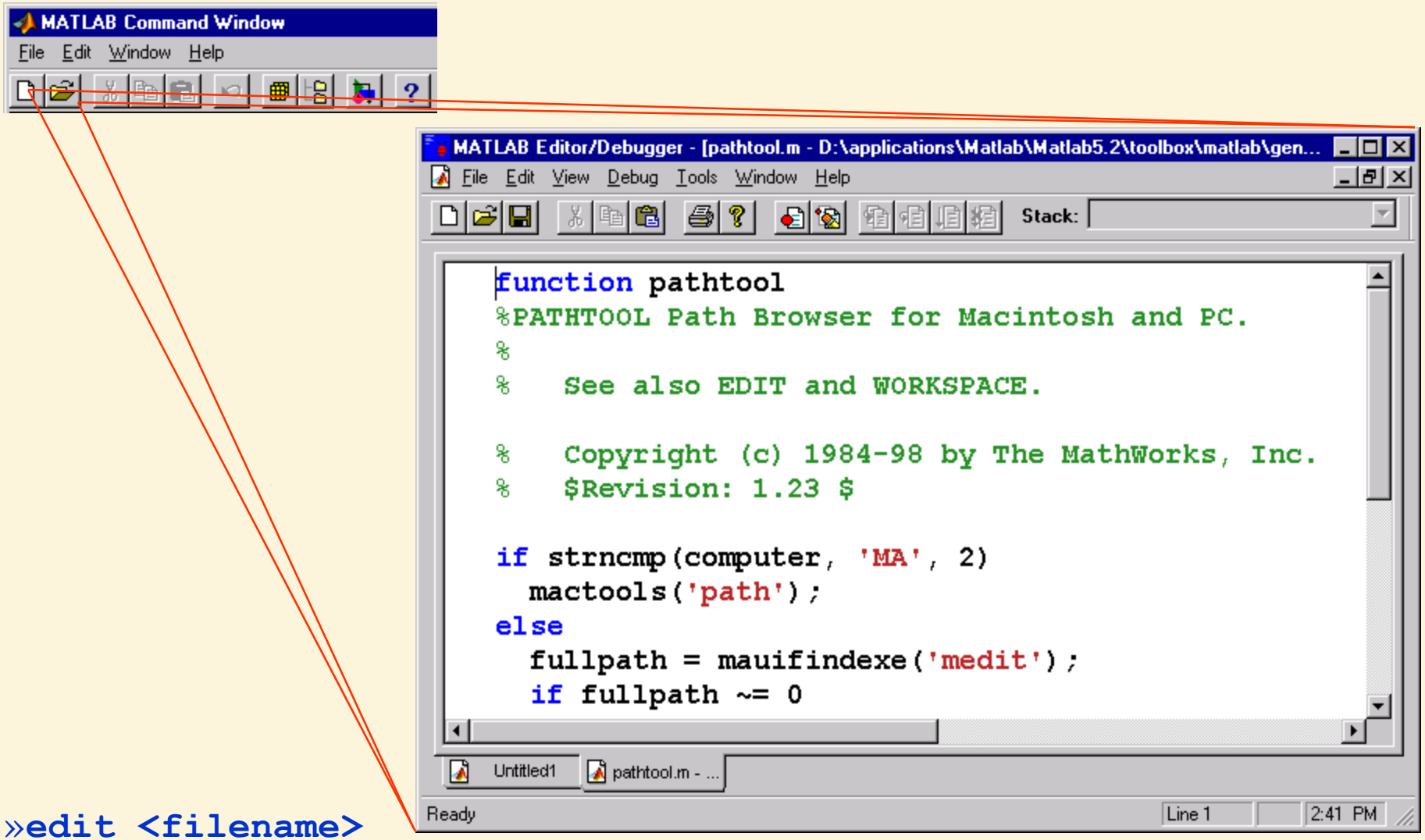
To list code, use `type` command

```
>> type factscript
```

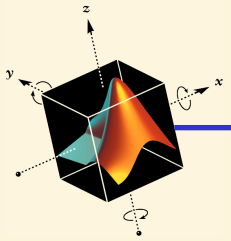
The `help` command displays first consecutive comment lines

```
>> help factscript
```

MATLAB Editor/Debugger



»edit <filename>



Functions

Functions describe subprograms

- Take inputs, generate outputs
- Have local variables (invisible in global workspace)

```
function [output_args]=
function_name(input_args)
% Comment lines
<function bod
```

factfun.m

```
function [z]=factfun(n)
% FACTFUN – Compute factorial
% Z=FACTFUN(N)

z = prod(1:n);
```

```
>> y=factfun(10);
```

Structure of a Function M-file

Keyword: function

Function Name (same as file name .m)

Output Argument(s)

Input Argument(s)

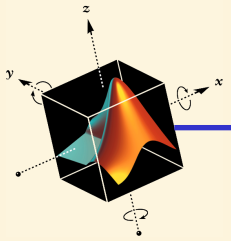
Online Help

MATLAB Code

```
function y = mean(x)
% MEAN Average or mean value.
% For vectors, MEAN(x) returns the mean value.
% For matrices, MEAN(x) is a row vector
% containing the mean value of each column.
[m,n] = size(x);
if m == 1
  m = n;
end
y = sum(x)/m;
```

```
»output_value = mean(input_value)
```

Command Line Syntax



Subfunctions

- Allows more than one function to be within the same M-file (modularize code)
- M-file must have the name of the first (*primary*) function
- Subfunctions can only be called from within the same M-file
- Each subfunction has its own workspace

Example: Subfunctions

Primary Function

Sub-Function

```
function [totalsum,average] = subfunc (input_vector)
% SUBFUNC Calculates cumulative total & average
totalsum = sum(input_vector);
average = ourmean(input_vector);  %Call to subfunction

  function y = ourmean(x)
  % (OURMEAN) Calculates average
  [m,n] = size(x);
  if m == 1
    m = n;
  end
  y = sum(x)/m;
```

```
»[SUM, MEAN] = subfunc(rand(1,50))
```

Multiple Input & Output Arguments

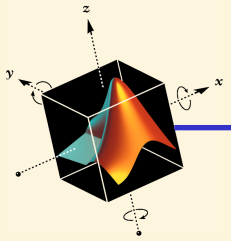
```
function r = ourrank(X,tol)
% OURRANK Rank of a matrix
s = svd(X);
if (nargin == 1)
   tol = max(size(X))*s(1)*eps;
end
r = sum(s > tol);
```

Multiple Input Arguments (,)

Multiple Output Arguments [,]

```
function [mean,stdev] = ourstat(x)
% OURSTAT Mean & std. deviation
[m,n] = size(x);
if m == 1
   m = n;
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m - mean.^2);
```

```
»RANK = ourrank(rand(5),0.1);
»[MEAN,STDEV] = ourstat(1:99);
```



Scripts or function: when use what?

Functions

- Take inputs, generate outputs, have internal variables
- Solve general problem for arbitrary parameters

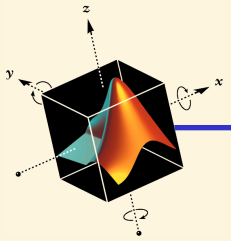
Scripts

- Operate on global workspace
- Document work, design experiment or test
- Solve a very specific problem once

facttest.m

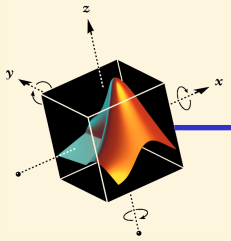
```
% FACTTEST – Test factfun

N=50;
y=factfun(N);
```



Flow Control Constructs

- Logic Control:
  - IF / ELSEIF / ELSE
  - SWITCH / CASE / OTHERWISE
- Iterative Loops:
  - FOR
  - WHILE



Logical expressions

Relational operators (compare arrays of same sizes)

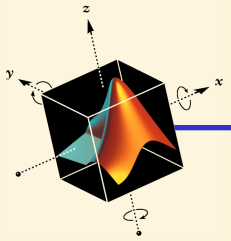
== (equal to) ~= (not equal)
< (less than) <= (less than or equal to)
> (greater than) >= (greater than or equal to)

Logical operators (combinations of relational operators)

& (and)
| (or)
~ (not)

Logical functions
xor
isempty

```
if (x>=0) & (x<=10)
  disp('x is in range [0,10]')
else
  disp('x is out of range')
end
```



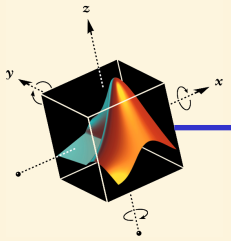
Flow control - repetition

Repeats a code segment a fixed number of times

```
for index=<vector>
   <statements>
end
```

The <statements> are executed repeatedly.
At each iteration, the variable `index` is assigned a new value from `<vector>`.

```
for k=1:12
   kfac=prod(1:k);
   disp([num2str(k),' ',num2str(kfac)])
end
```

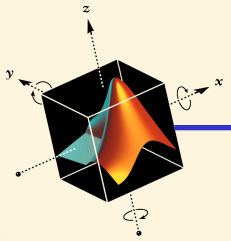


Flow control - selection

The if-elseif-else construction

```
if <logical expression>
   <commands>
elseif <logical expression>
   <commands>
else
   <commands>
end
```

```
if height>170
   disp('tall')
elseif height<150
   disp('small')
else
   disp('average')
end
```

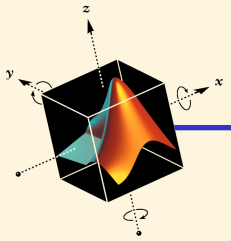


Example – selection and repetition

fact.m

```
function y=fact(n)
% FACT – Display factorials of integers 1..n
if nargin < 1
   error('No input argument assigned')
elseif n < 0
   error('Input must be non-negative')
elseif abs(n-round(n)) > eps
   error('Input must be an integer')
end

for k=1:n
   kfac=prod(1:k);
   disp([num2str(k),' ',num2str(kfac)])
   y(k)=kfac;
end;
```



Switch, Case, and Otherwise

- More efficient than elseif statements
- Only the first matching case is executed

```
switch input_num
case -1
  input_str = 'minus one';
case 0
  input_str = 'zero';
case 1
  input_str = 'plus one';
case {-10,10}
  input_str = '+/- ten';
otherwise
  input_str = 'other value';
end
```

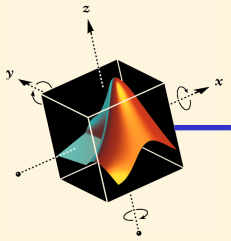
»switch_examp

The while loop

- Similar to other programming languages
- Repeats loop until logical condition returns FALSE.
- Can be nested.

```
I=1; N=10;
while I<=N
   J=1;
   while J<=N
      A(I,J)=1/(I+J-1);
      J=J+1;
   end
   I=I+1;
end
```

»while_examp



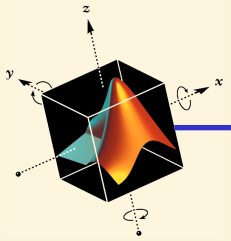
Flow control – conditional repetition

while-loops

```
while <logical expression>
   <statements>
end
```

<statements> are executed repeatedly as long as the <logical expression> evaluates to true

```
k=1;
while prod(1:k)~=Inf,
    k=k+1;
end
disp(['Largest factorial in Matlab:',num2str(k-1)]);
```



Flow control – conditional repetition

Solutions to nonlinear equations

$$f(x) = 0$$

can be found using Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Task: write a function that finds a solution to

$$f(x) = e^{-x} - \sin(x)$$

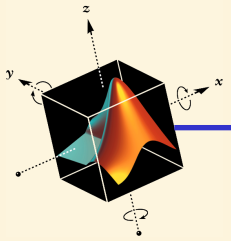
Given x_0, iterate until $|x_n - x_{n-1}| \leq \text{tol}$

Flow control – conditional repetition

newton.m

```
function [x,n] = newton(x0,tol,maxit)
% NEWTON – Newton's method for solving equations
% [x,n] = NEWTON(x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
  n = n + 1;
  x_new = x - (exp(-x)-sin(x))/(-exp(-x)-cos(x));
  done=(n>=maxit) | ( abs(x_new-x)<tol );
  x=x_new;
end
```

```
>> [x,n]=newton(0,1e-3,10)
```



Function functions

Do we need to re-write `newton.m` for every new function?

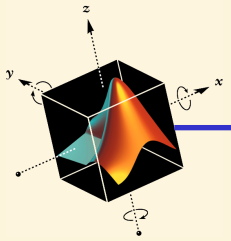
No! General purpose functions take other m-files as input.

```
>> help feval
```

myfun.m

```
function [f,f_prime] = myfun(x)
% MYFUN— Evaluate f(x) = exp(x)-sin(x)
% and its first derivative
% [f,f_prime] = myfun(x)

f=exp(-x)-sin(x);
f_prime=-exp(-x)-cos(x);
```



Function functions

Can update newton.m

newtonf.m

```
function [x,n] = newtonf(fname,x0,tol,maxit)
% NEWTON – Newton's method for solving
equations
% [x,n] = NEWTON(fname,x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
  n = n + 1;
  [f,f_prime]=feval(fname,x);
  x_new = x - f/f_prime;
  done=(n>maxit) | ( abs(x_new-x)<tol );
  x=x_new;
end
```

```
>> [x,n]=newtonf('myfun',0,1e-3,10)
```

Programming tips and tricks

Programming style has huge influence on program speed!

slow.m

```
tic;
x=-2500:0.1:2500;
for ii=1:length(x)
  if x(ii)>=0,
    s(ii)=sqrt(x(ii));
  else
    s(ii)=0;
  end;
end;
toc
```

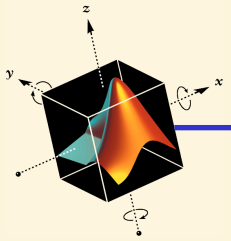
fast.m

```
tic
x=-2500:0.1:2500;
s=sqrt(x);
s(x<0)=0;
toc;
```

Loops are slow: Replace loops by vector operations!

Memory allocation takes a lot of time: Pre-allocate memory!

Use profile to find code bottlenecks!



Recall: Array Operations

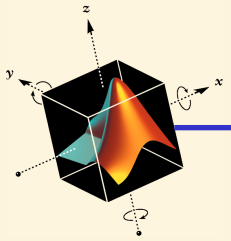
- Using Array Operations:

```
Density = Mass(I,J)/(Length.*Width.*Height);
```

- Using Loops:

```
[rows, cols] = size(M);
for I = 1:rows
  for J = 1:cols
    Density(I,J) = M(I,J)/(L(I,J)*W(I,J)*H(I,J));
  end
end
```

»array_vs_loops



Summary

User-defined functionality in m-files

- Stored in current directory, or on search path

Script-files vs. functions

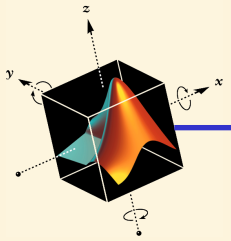
- Functions have local variables,
- Scripts operate on global workspace

Writing m-files

- Header (function definition), comments, program body
- Have inputs, generate outputs, use internal variables
- Flow control: ”if...elseif...if”, ”for”, ”while”
- General-purpose functions: use functions as inputs

Programming style and speed

- Vectorization, memory allocation, profiler



Advanced Matlab Programming

Functions

- Can have variable number of inputs and outputs (see: nargin, nargout, varargin, varargout)
- Can have internal functions

Data types: more than just arrays and strings:

- Structures
- Cell arrays

File handling

- Supports most C-commands for file I/O (fprintf,…)