What is Matlab ?

- MATLAB is very popular in the academic sector
- It provides built-in functions for manipulating vectors and matrices.
- Other emerging tool (Phyton that is open-source)
- There are lot of online resources for quick tutorials or references of built-in functions :
  - http://www.mathworks.com/help/matlab/functionlist.html
- A Matlab script (e.g. Myscript.m) is a text file le containing lines of commands
- You run the script by typing “Myscript” at the MATLAB command prompt, MATLAB then executes each line one at a time exactly as written into the command line. Lines beginning with % are treated by MATLAB as comments and are ignored.
- A script can be written with any text editor, but MATLAB has a built-in m-file editor.

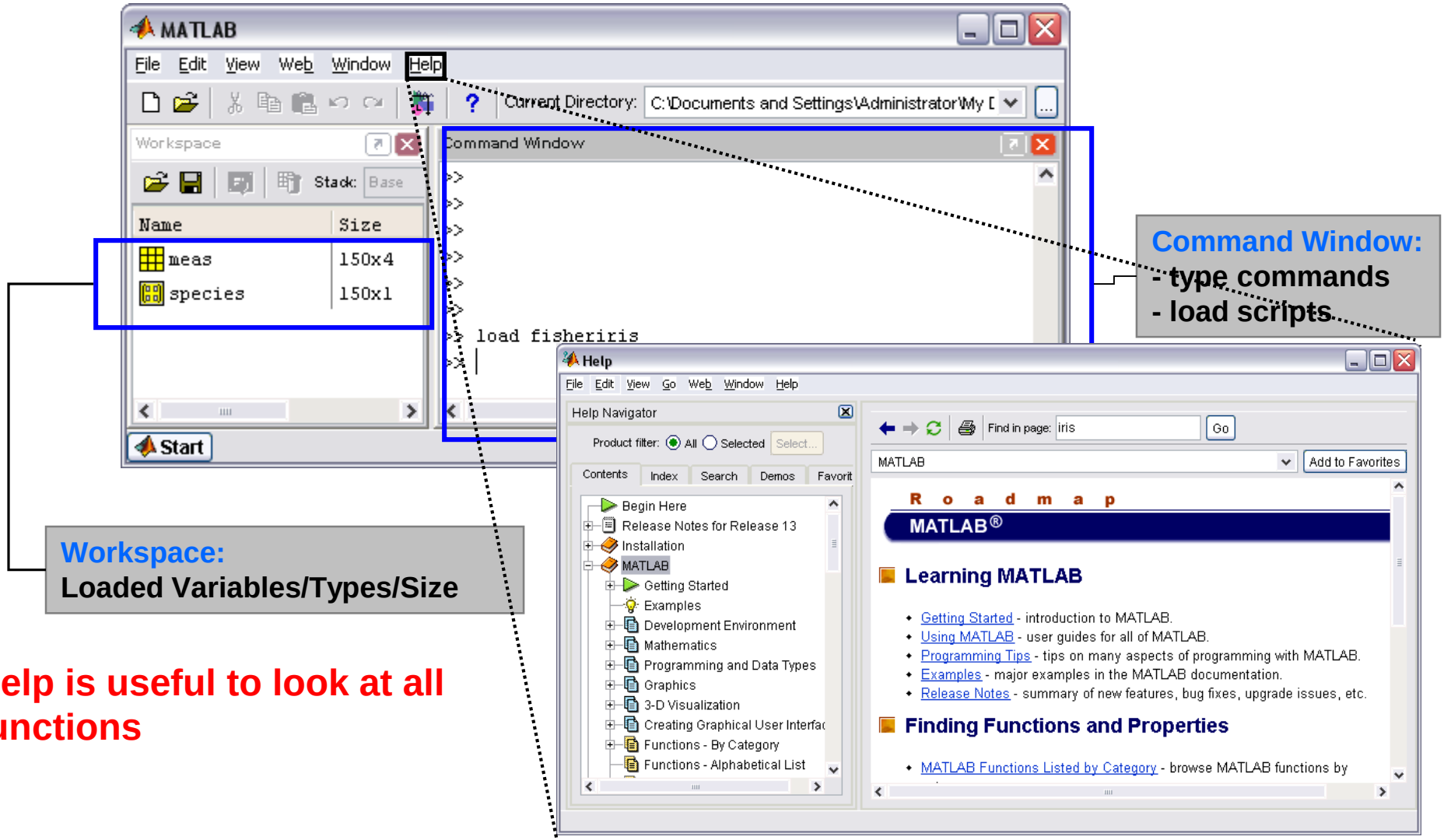
T_0

Starting

- **Dos/Unix like directory navigation**
- **Commands like:**
  - cd
  - pwd
  - mkdir
- **For navigation it is easier to just copy/paste the path from explorer**
  E.g.:
  cd ‘c:\myscripts\’

T_0

Matlab GUI



Command Window:
- type commands
- load scripts

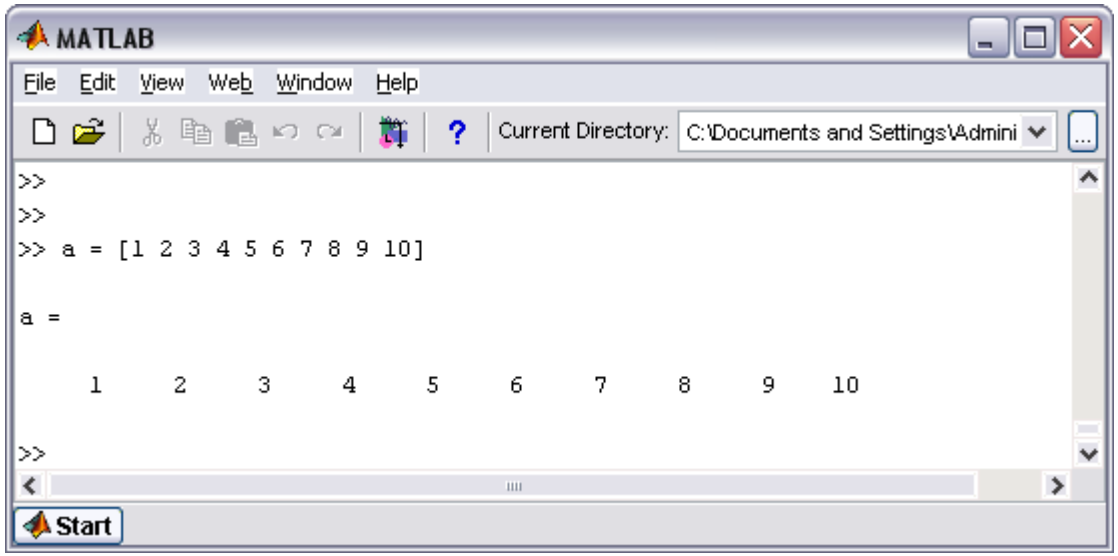
Workspace:
Loaded Variables/Types/Size

Help is useful to look at all functions

Matlab is arrays

- **Manipulation of arrays is faster than regular manipulation with for-loops**

```
a = [1 2 3 4 5 6 7 9 10] % define an array
```



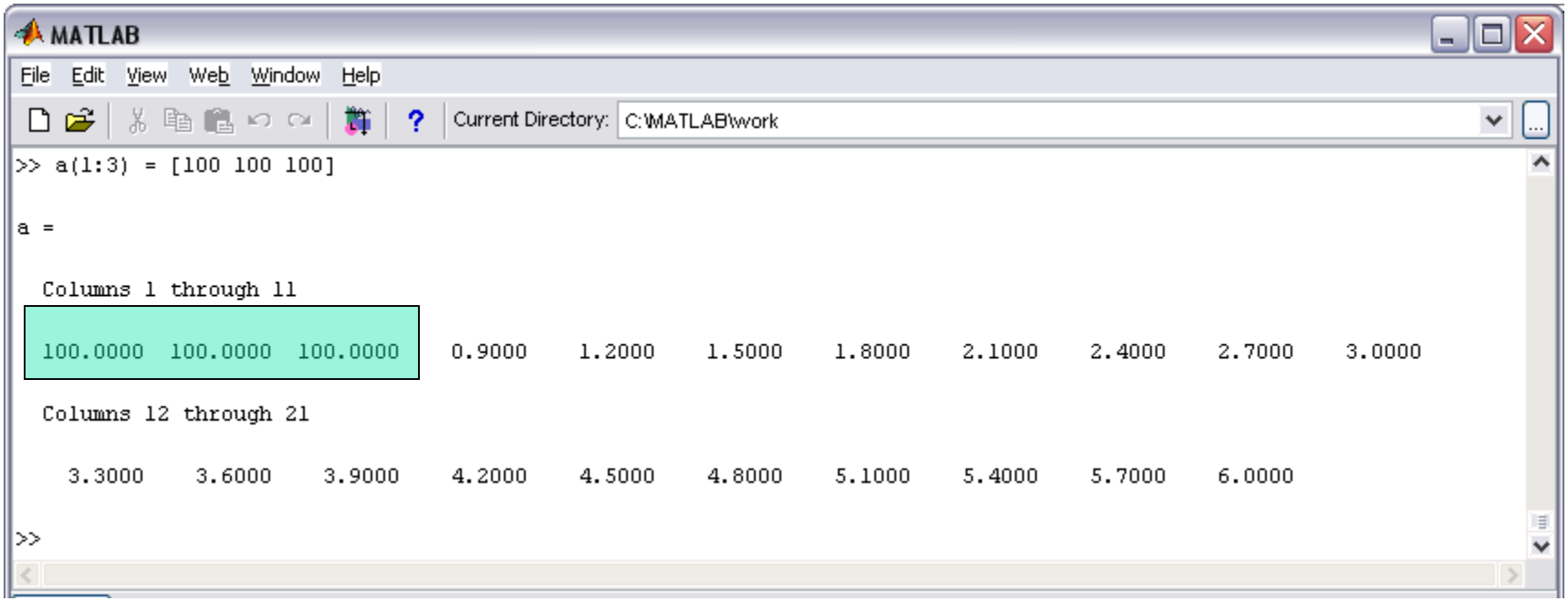
Accessing 1D arrays

```
>> a(1)

ans =

           0
```

```
>> a(1:3)
ans =
           0     0.3000     0.6000
```

```
>> a(1) = 100
```

```
>> a(1:3) = [100 100 100]
```



Accessing 2D arrays

- 2D arrays can be accessed by columns or rows

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

     1     2     3
     4     5     6

>> a(2,2)

ans =

     5
```

```
>> a(1,:)

ans =

     1     2     3

>> a(:,1)

ans =

     1
     4
```

Row-wise access

Column-wise access

Computation is by column

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

     1     2     3
     3     2     1

>> mean(a)

ans =

    2.0000    2.0000    2.0000
```

```
>> max(a)

ans =

     3     2     3

>> sort(a)

ans =

     1     2     1
     3     2     3
```

Concatenation is by column or row

Row next to row

```
>> a = [1 2 3];
>> b = [4 5 6];
>> c = [a b]

c =

     1     2     3     4     5     6
```

Column next to column

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

     1     3
     2     4
```

Row below row

```
>> a = [1 2 3];
>> b = [4 5 6];
>> c = [a; b]

c =

     1     2     3
     4     5     6
```

Column below column

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

c =

     1
     2
     3
     4
```

Initializing arrays

- Create array of ones [`ones`]

```
>> a = ones(1,3)
a =

     1     1     1

>> a = ones(1,3)*inf
a =
     Inf   Inf   Inf
```

```
>> a = ones(2,2)*5;
a =

     5     5
     5     5
```

- **Create array of zeroes [`zeros`]**
  - Good for initializing arrays

```
>> a = zeros(1,4)
a =

     0     0     0     0
```

```
>> a = zeros(3,1) + [1 2 3]'
a =
     1
     2
     3
```

Reshaping and resizing arrays

- Changing the array shape [reshape]
  - (eg, for easier column-wise computation)

```
>> a = [1 2 3 4 5 6]'; % make it into a column
>> reshape(a,2,3)

ans =

     1     3     5
     2     4     6
```

reshape(X,[M,N]):
[M,N] matrix of
columnwise version
of X

- **Replicating an array [repmat]**

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

ans =    1     2     3     1     2     3

>> repmat(a,2,1)
ans =
         1     2     3
         1     2     3
```

repmat(X,[M,N]):
make [M,N] tiles of X

Useful functions to work with arrays (I)

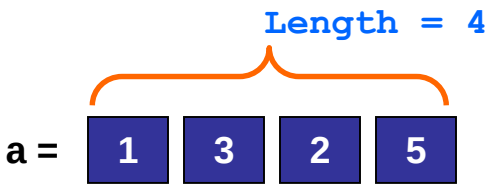
- Last element of array [`end`]

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

ans =

     5
```

```
>> a = [1 3 2 5];
>> a(end-1)

ans =

     2
```

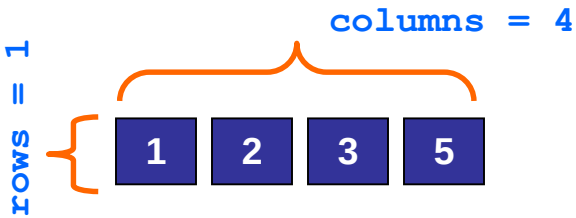
- **Length of array [`length`]**

```
>> length(a)

ans =

     4
```



- **Dimensions of array [`size`]**

```
>> [rows, columns] = size(a)
rows = 1

columns = 4
```



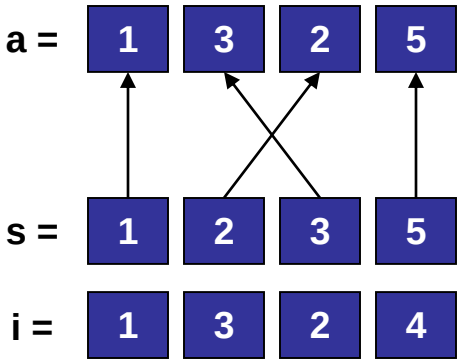
Useful functions to work with arrays (II)

- Find a specific element [`find`] **

```
>> a = [1 3 2 5 10 5 2 3];
>> b = find(a==2)

b =

     3     7
```

- **Sorting [`sort`] *****

```
>> a = [1 3 2 5];
>> [s,i]=sort(a)

s =
     1     2     3     5

i =
     1     3     2     4
```



Indicates the index where the element came from

Some operators must be handled with care:

Some operators must be handled with care:

A = [1 2 ; 4 5]

B = A * A prints

```
 9   12
24   33
```

B = A .* A prints

```
 1    4
16   25
```

Element by element multiplication

Sub-matrixes

A matrix can be indexed using another matrix, to produce a subset of its elements:

a = [100 200 300 400 500 600 700] b = [3 5 6]

c = a(b):

To get a subsection of a matrix, we can produce the index matrix with the colon operator:

a(2:5)

prints

ans = 200 300 400 500

- This works in 2-D as well, e.g. c(2:3, 1:2) produces a 2 x 2 submatrix.

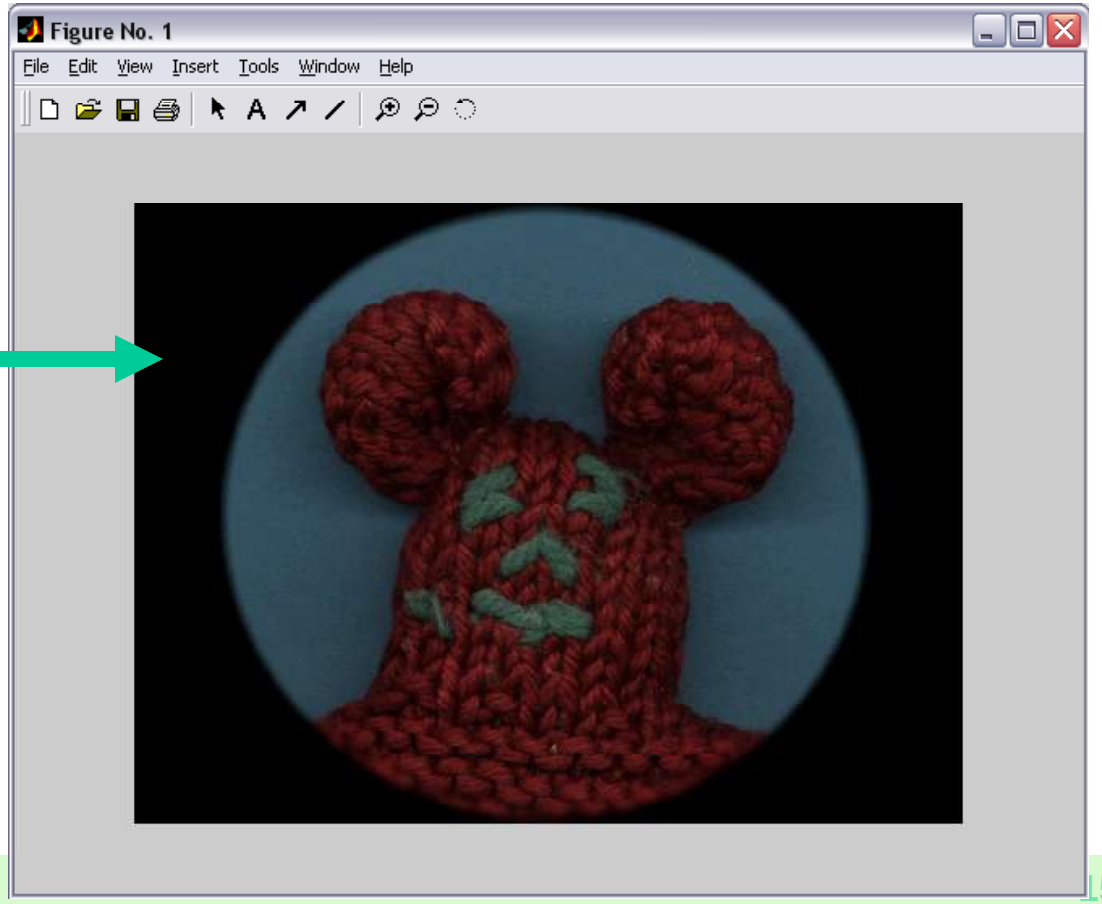
- The rows and columns of the submatrix are renumbered.

Images can be treated as matrix

Loading an image:

a = imread(‘picture.jpg’);

imshow(a);



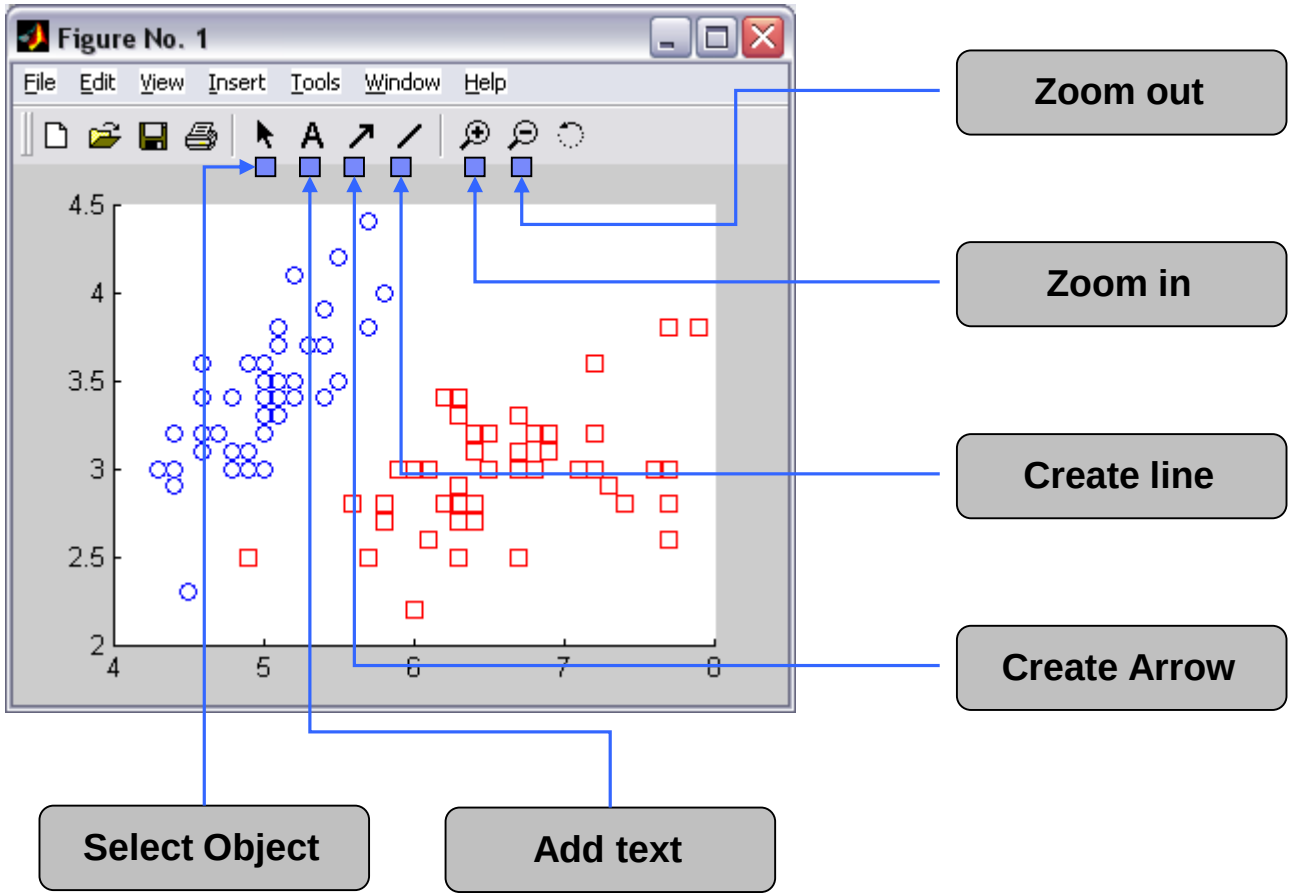
Plotting

- Commands covered: plot, xlabel, ylabel, title grid, axis, stem, subplot
- *xlabel('time (sec)'); ylabel('step response'); title('My Plot');*

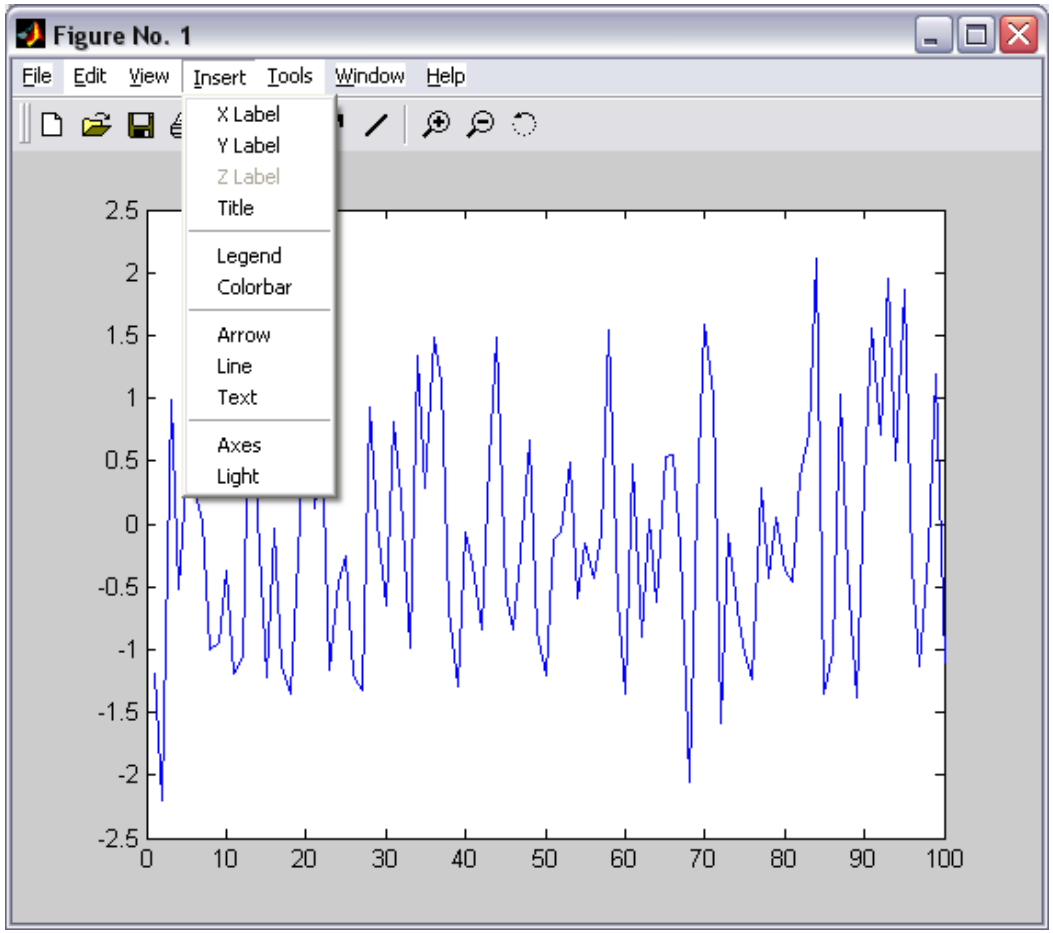
Eg:To plot more than one graph on the screen, use the command subplot(mnp) which partitions the screen into an mxn grid where p determines the position of the particular graph counting the upper left corner as p=1. For example,

- subplot(211),semilogx(w,magdb);
- subplot(212),semilogx(w,phase);

Modifying plots



Changing plots as visualization



- ***Add titles***
- ***Add labels on axis***
- ***Change tick labels***
- ***Add grids to axis***
- ***Change color of line***
- ***Change thickness/ Linestyle***
- ***etc***

Example (I)

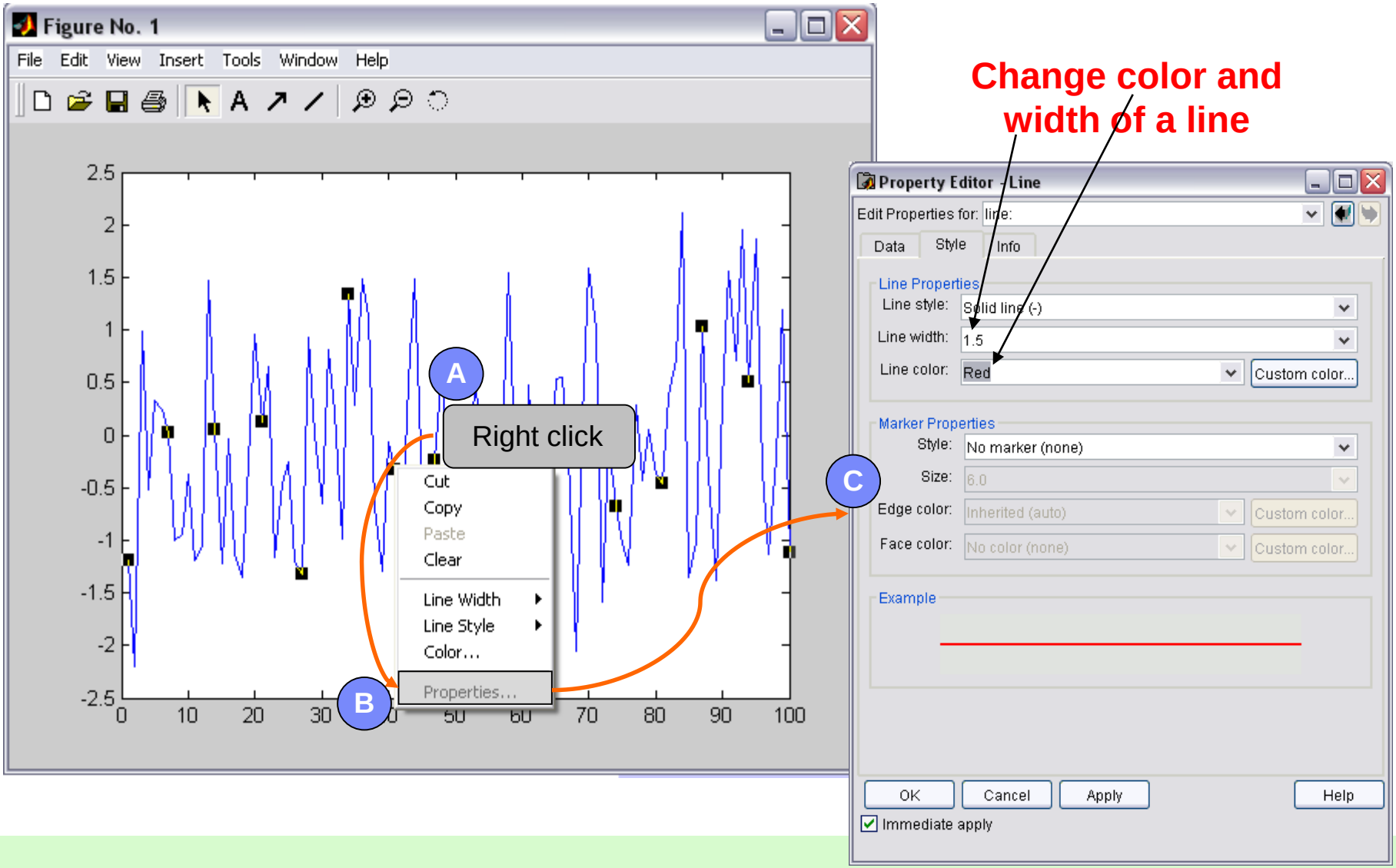
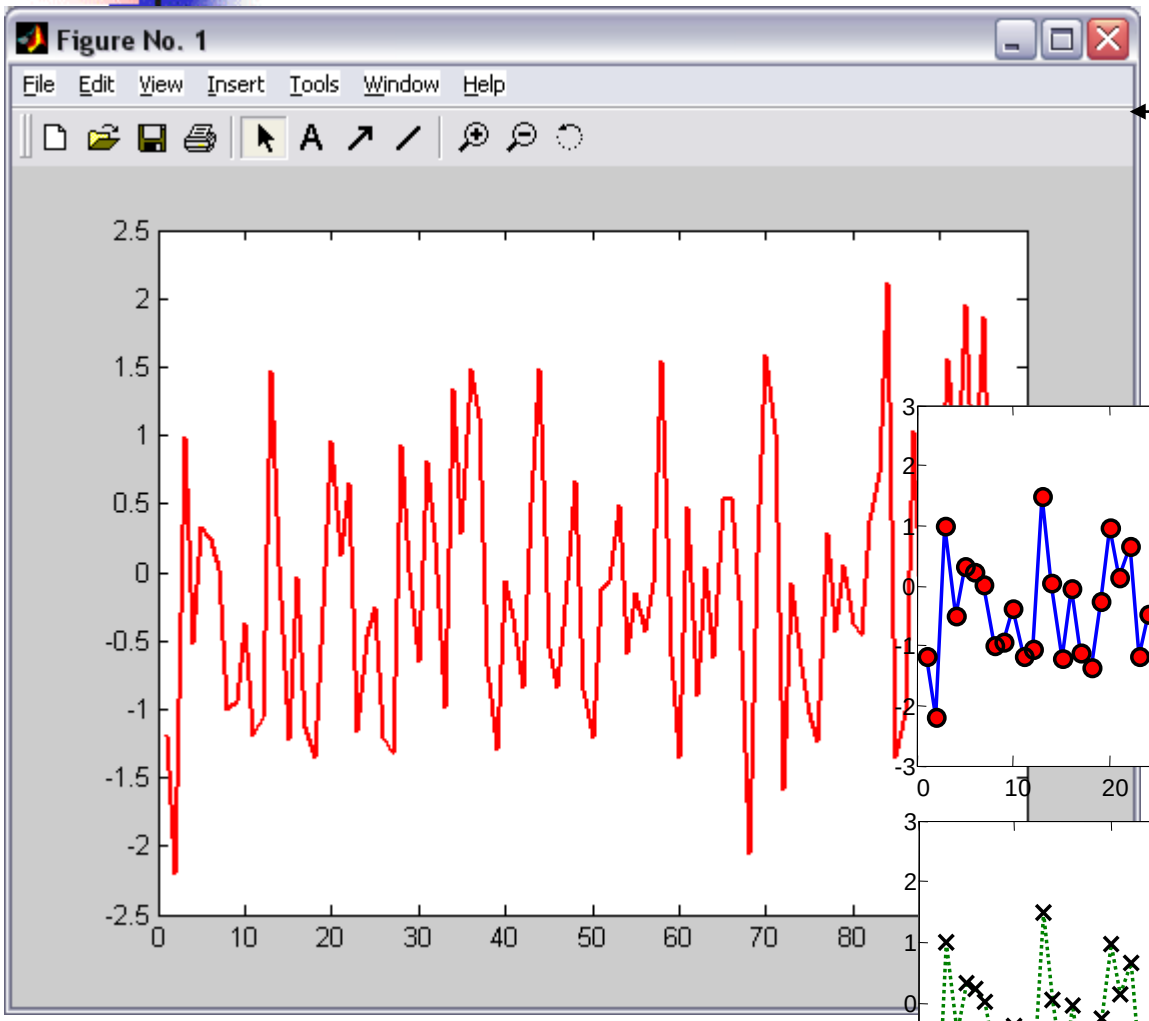


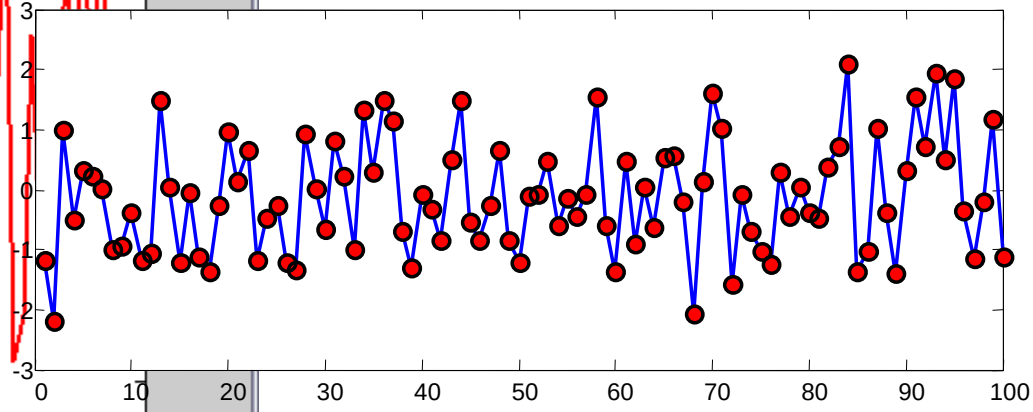
Figure No. 1
File Edit View Insert Tools Window Help
Change color and width of a line
Right click
A
B
C
Cut
Copy
Paste
Clear
Line Width
Line Style
Color...
Properties...
Property Editor - Line
Edit Properties for: line:
Data Style Info
Line Properties
Line style: Solid line (-)
Line width: 1.5
Line color: Red
Custom color...
Marker Properties
Style: No marker (none)
Size: 6.0
Edge color: Inherited (auto)
Face color: No color (none)
Example
OK Cancel Apply Help
Immediate apply

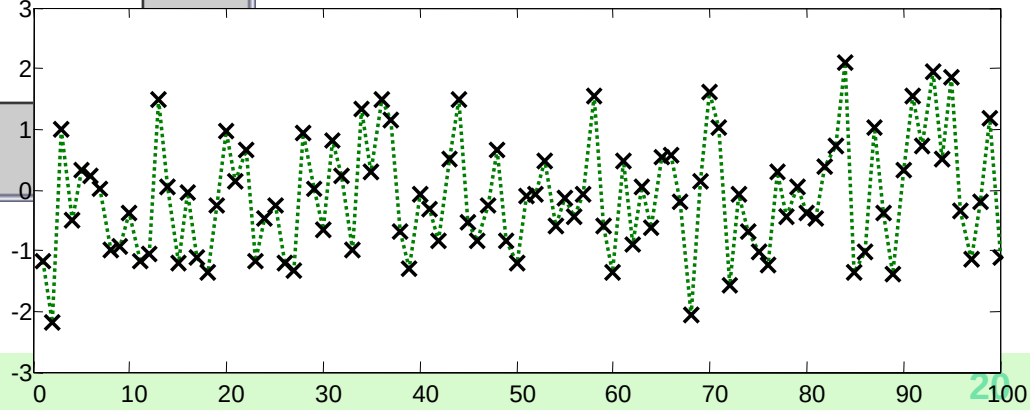
Example (II)



The result …

Other Styles:

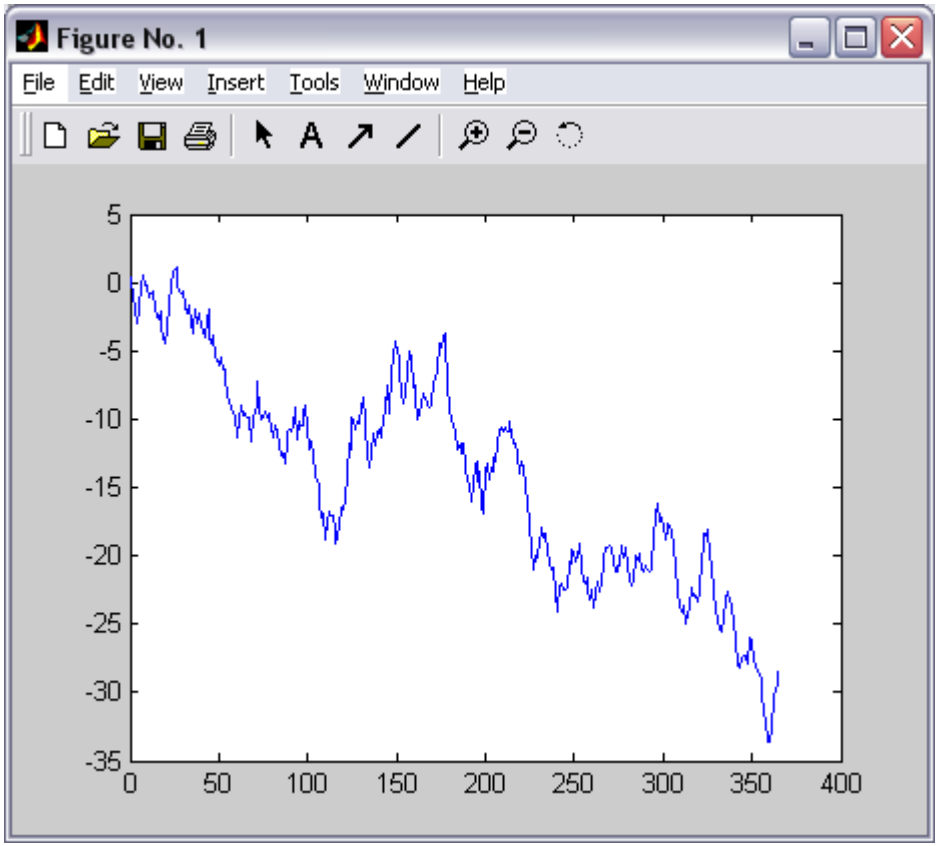




Changing figure properties with code (I)

- GUI's are easy, but sooner or later we realize that coding is faster

```
>> a = cumsum(randn(365,1)); % random walk of 365 values
```



If this represents a year's worth of measurements of an imaginary quantity, we will change:

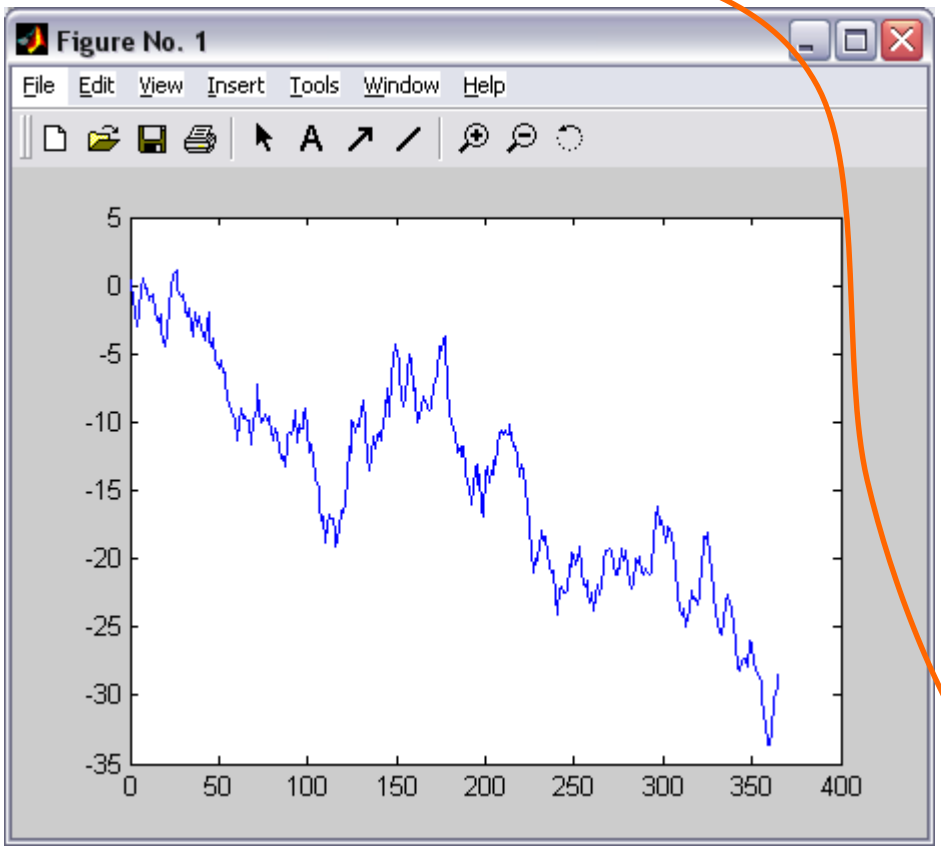
- x-axis annotation to months
- Axis labels
- Put title in the figure
- Include some greek letters in the title *just for fun*

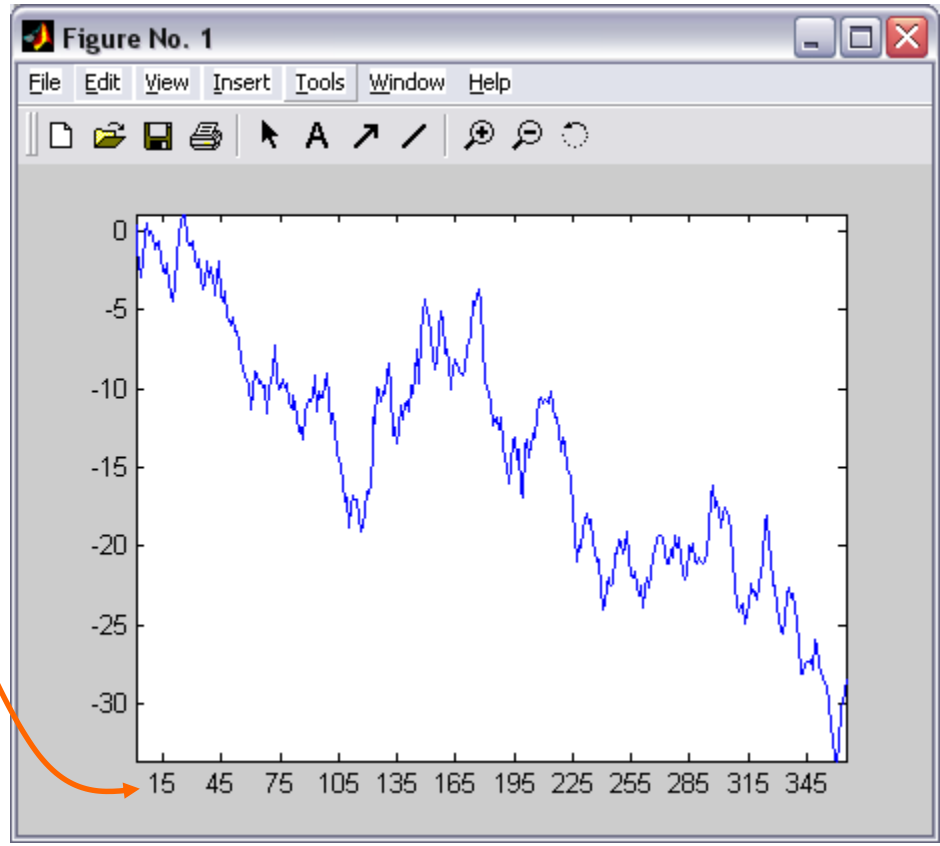
Changing figure properties with code (II)

- Axis annotation to months

```
>> axis tight; % irrelevant but useful...
>> xx = [15:30:365];
>> set(gca, 'xtick',xx)
```

The result …



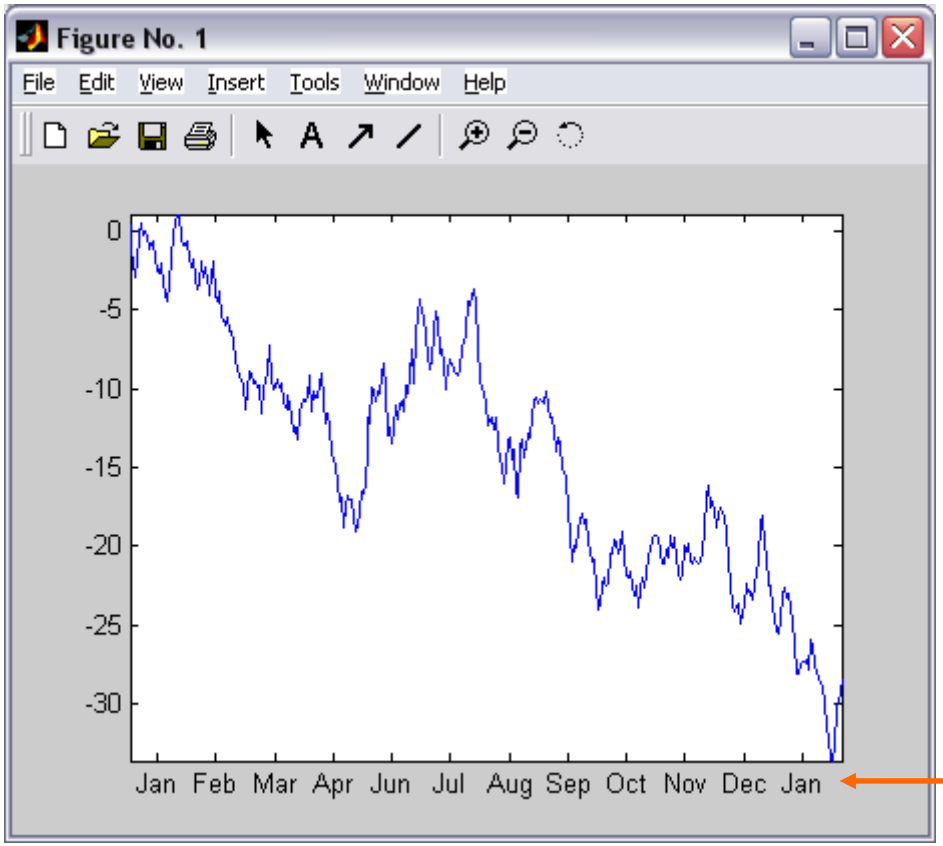


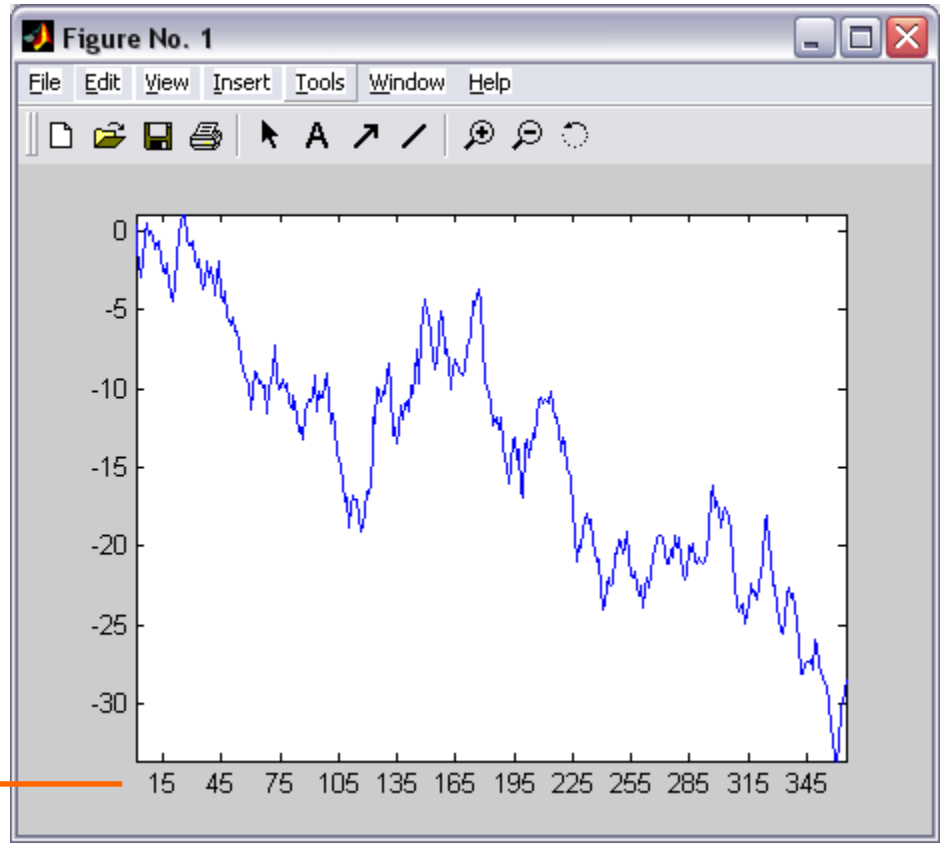
Changing properties of figures with code (III)

- **Axis annotation to months**

```
>> set(gca,'xticklabel',['Jan'; ...
                         'Feb';'Mar'])
```

The result …





Changing properties of figures with code (IV)

- **Axis labels and title**

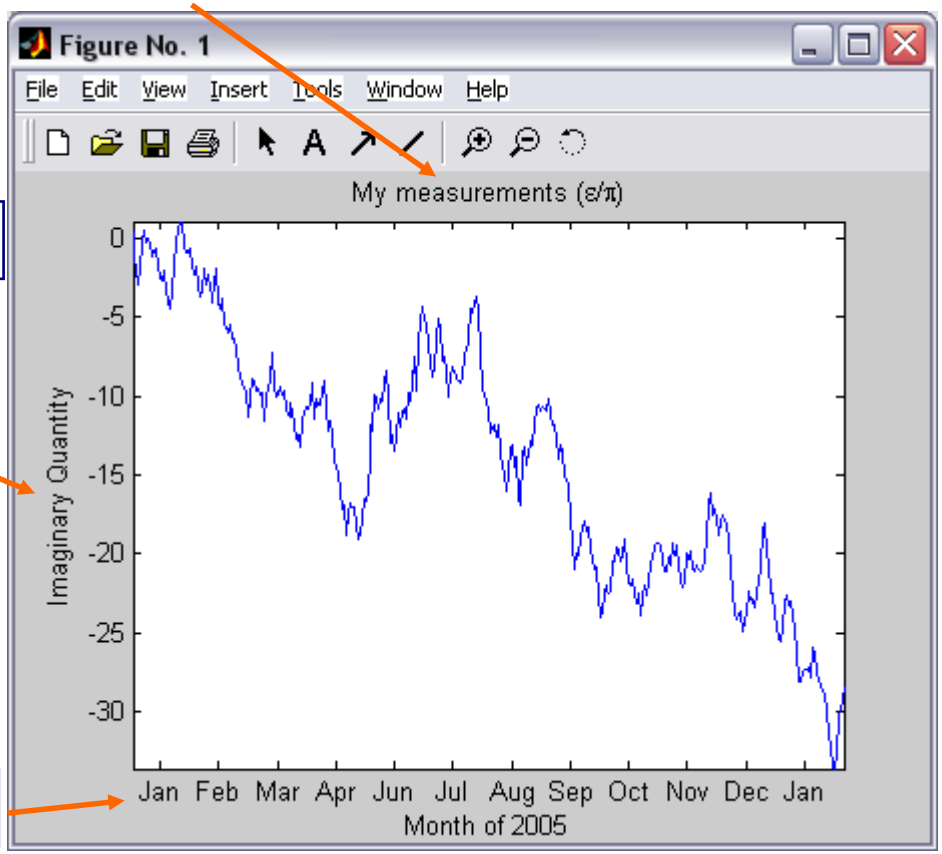
Other latex examples:

\alpha, \beta, e^{-\alpha} etc

```
>> title('My measurements (\epsilon/\pi)')
```

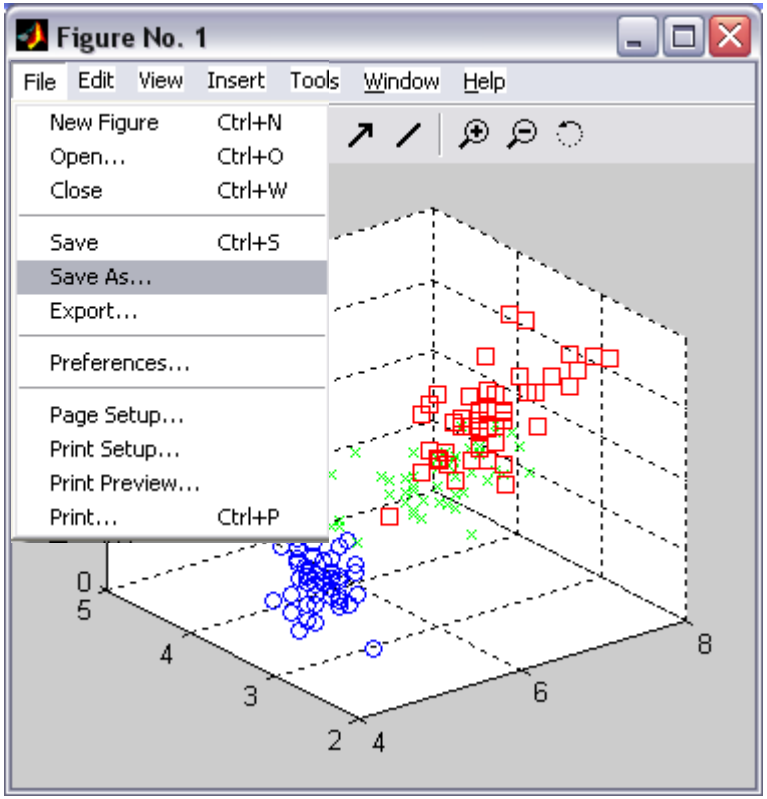
```
>> ylabel('Imaginary Quantity')
```

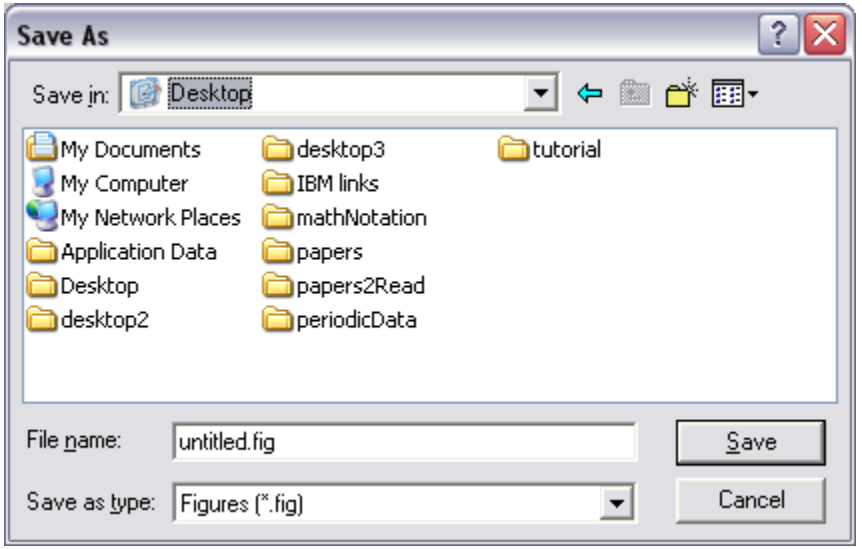
```
>> xlabel('Month of 2005')
```



How to save figures

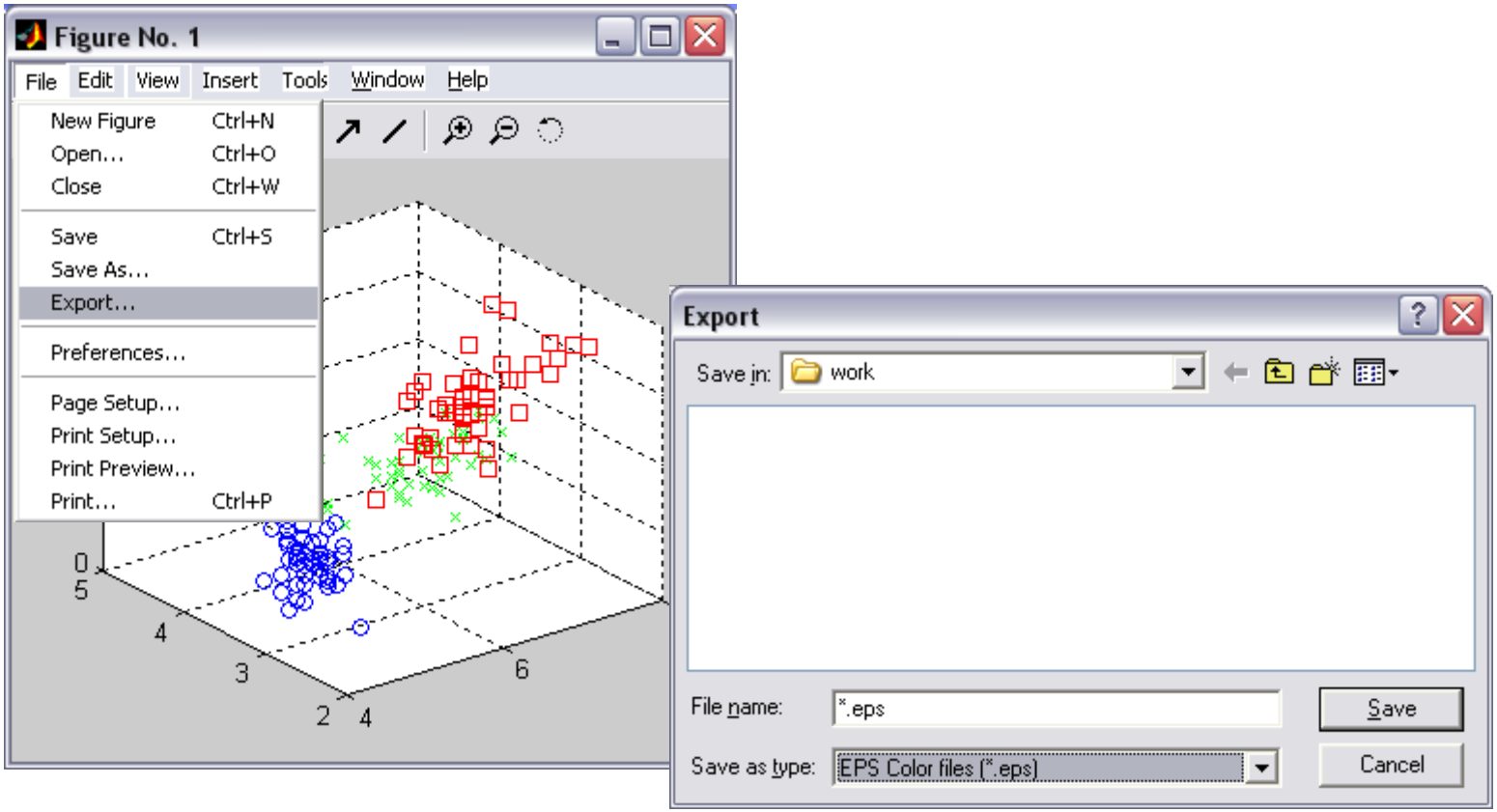
- **Matlab allows to save the figures (.fig) for later processing**





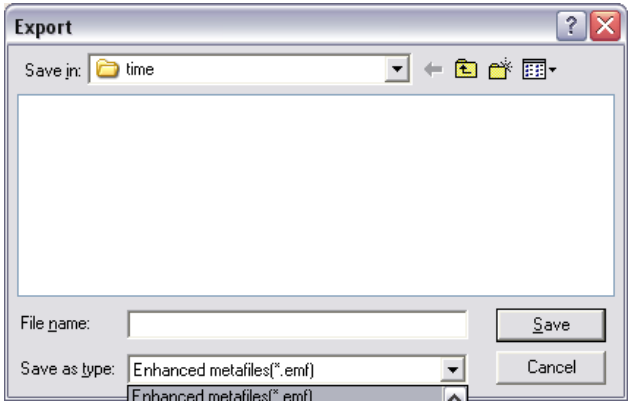
.fig can be later opened through Matlab

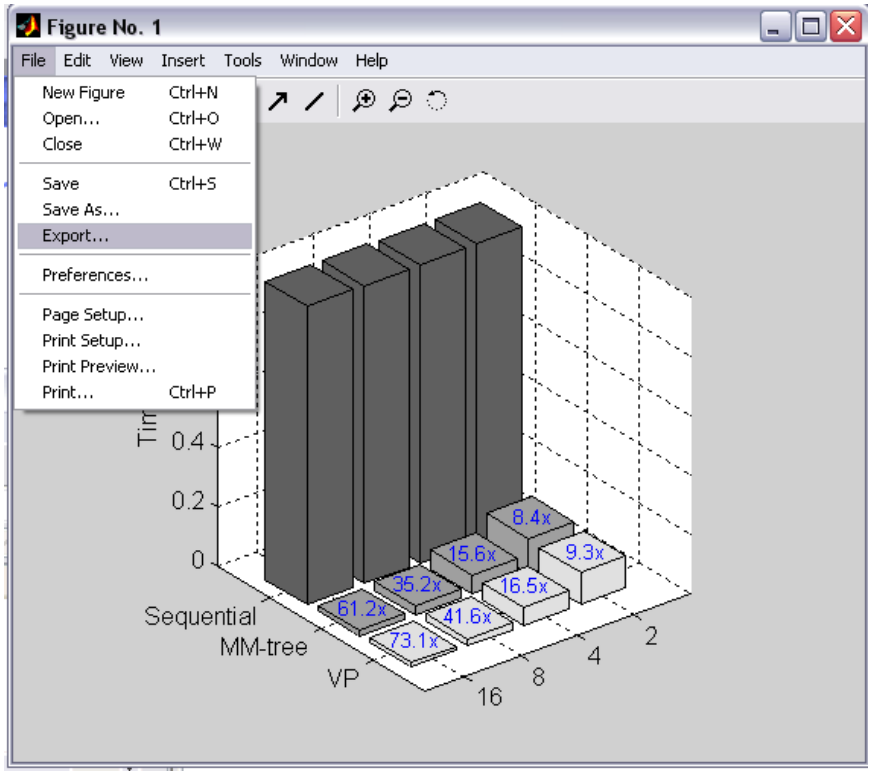
How to export figures



Export to:
emf, eps, jpg, etc

You can also achieve the same result with Matlab code

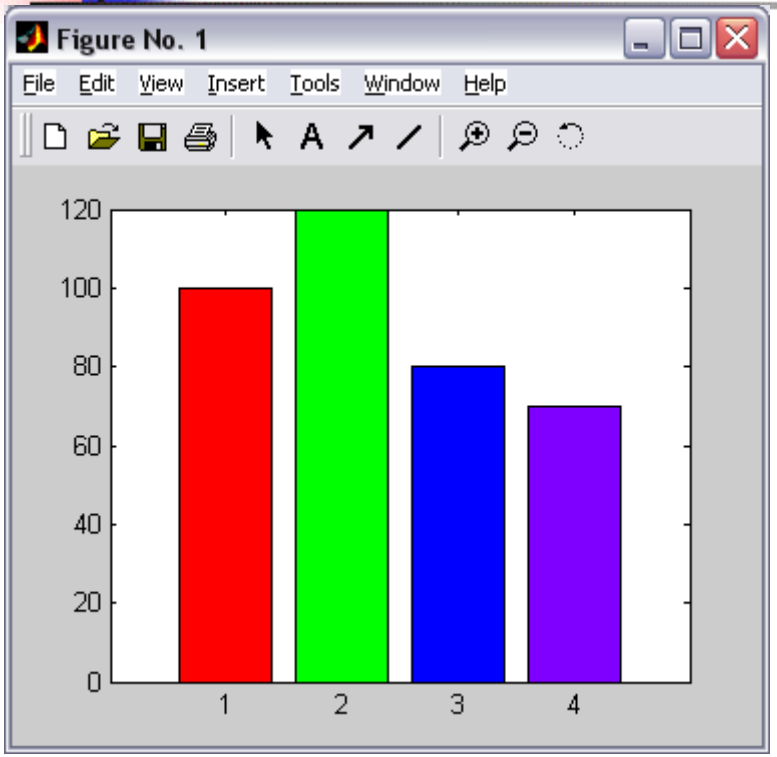


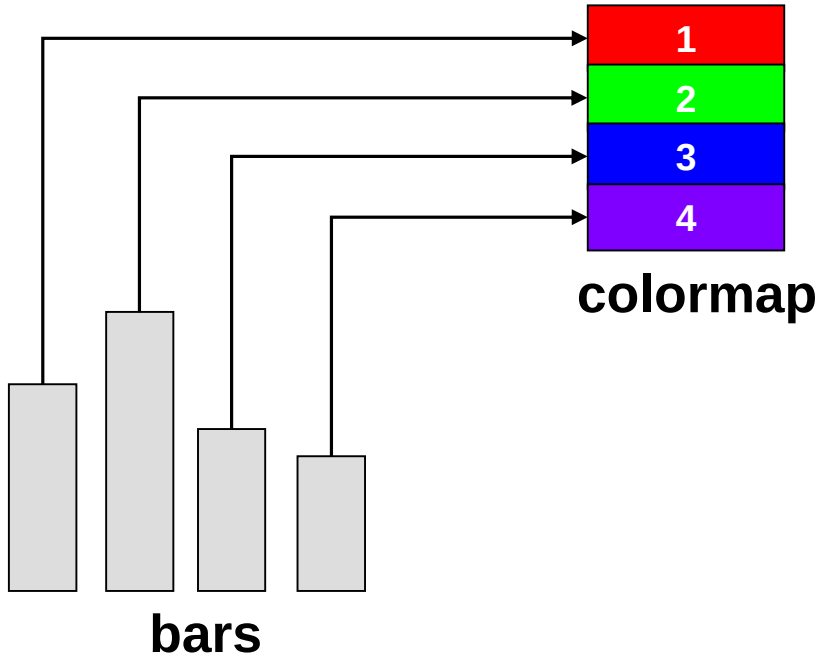


- **Matlab code:**

```
% extract to color eps
print -depsc myImage.eps; % from command-line
print(gcf,'-depsc','myImage') % using variable as name
```

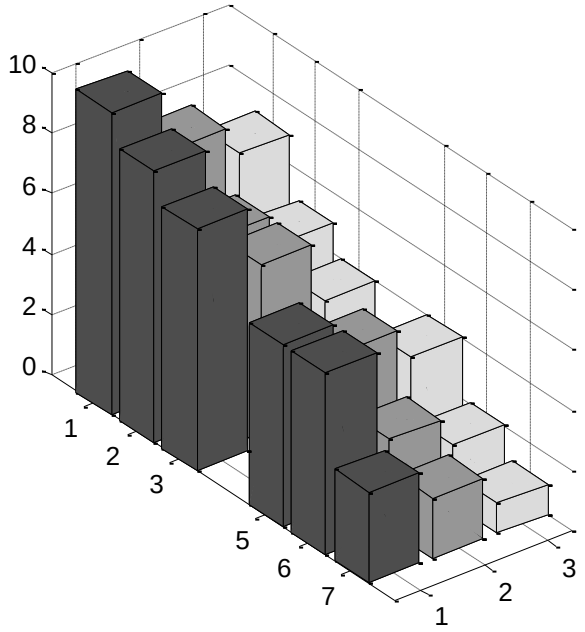
Visualizing data – 2D bars






```
time = [100 120 80 70]; % our data
h = bar(time); % get handle
cmap = [1 0 0; 0 1 0; 0 0 1; .5 0 1]; % colors
colormap(cmap); % create colormap

cdata = [1 2 3 4]; % assign colors
set(h,'CDataMapping','direct','CData',cdata);
```

Visualizing data – 3D bars



data

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

colormap

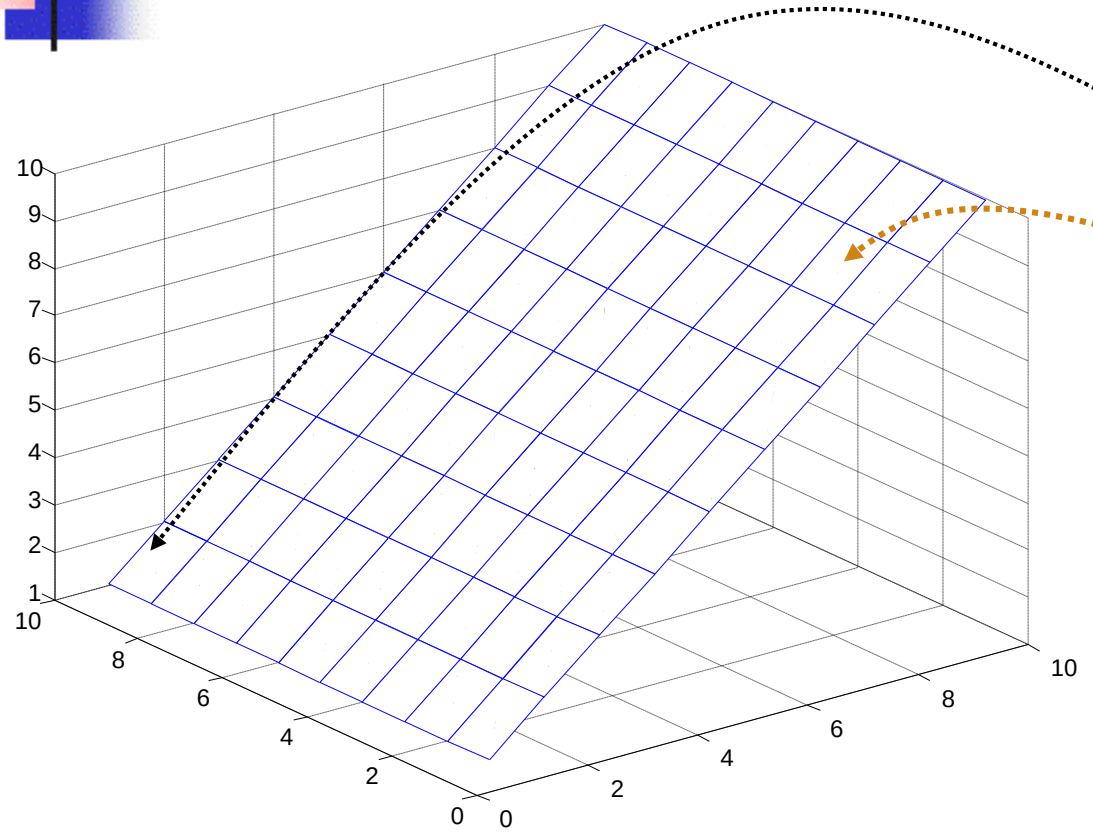
64 ×

0	0	0
0.0198	0.0124	0.0079
0.0397	0.0248	0.0158
0.0595	0.0372	0.0237
0.0794	0.0496	0.0316
0.0992	0.0620	0.0395
	. . .	
1.0000	0.7440	0.4738
1.0000	0.7564	0.4817
1.0000	0.7688	0.4896
1.0000	0.7812	0.4975

3

```
data = [ 10 8 7; 9 6 5; 8 6 4; 6 5 4; 6 3 2; 3 2 1];
bar3([1 2 3  5 6 7], data);

c = colormap(gray); % get colors of colormap
c = c(20:55,:); % get some colors
colormap(c); % new colormap
```

Visualizing data – Surface



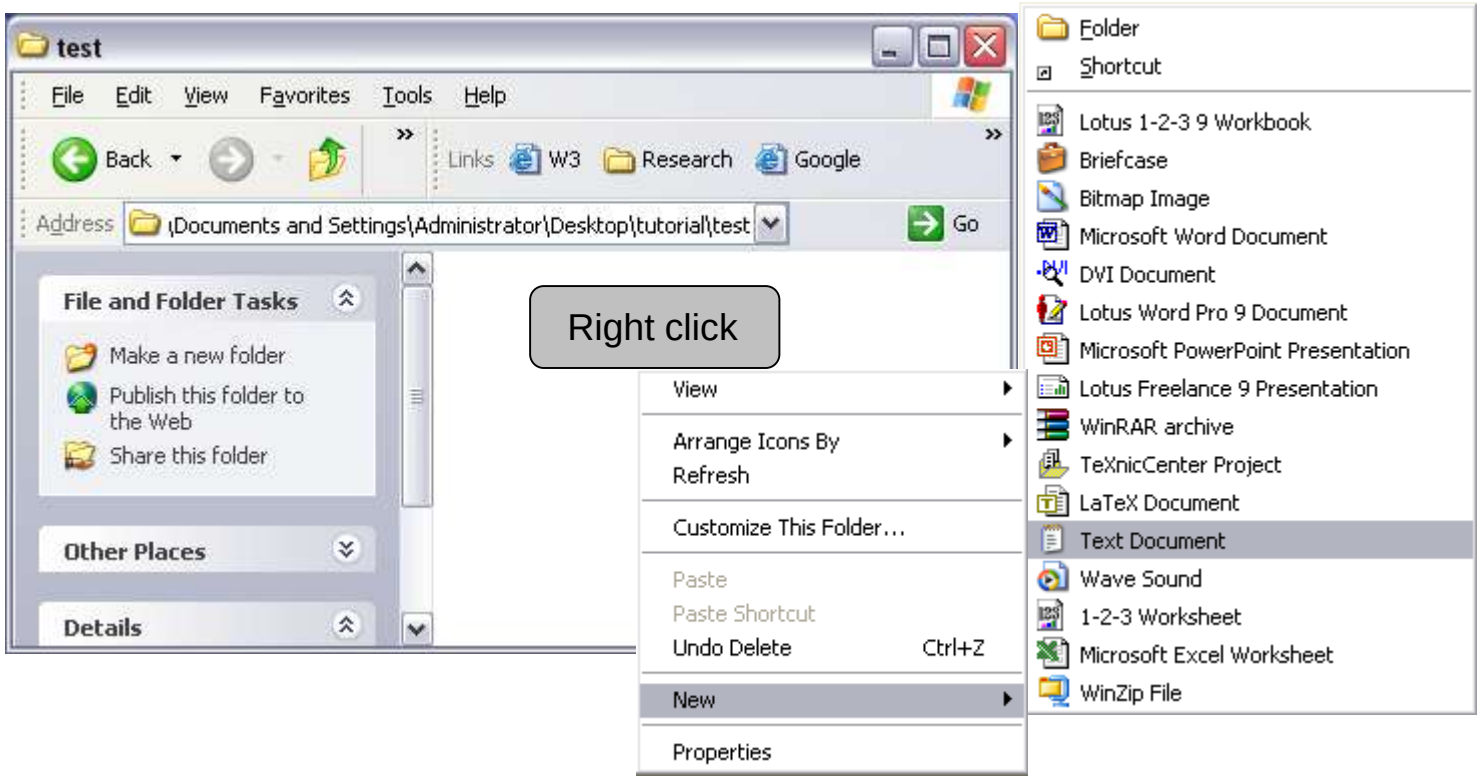
data

1	2	3	…		10
1					
				9	10
1					10

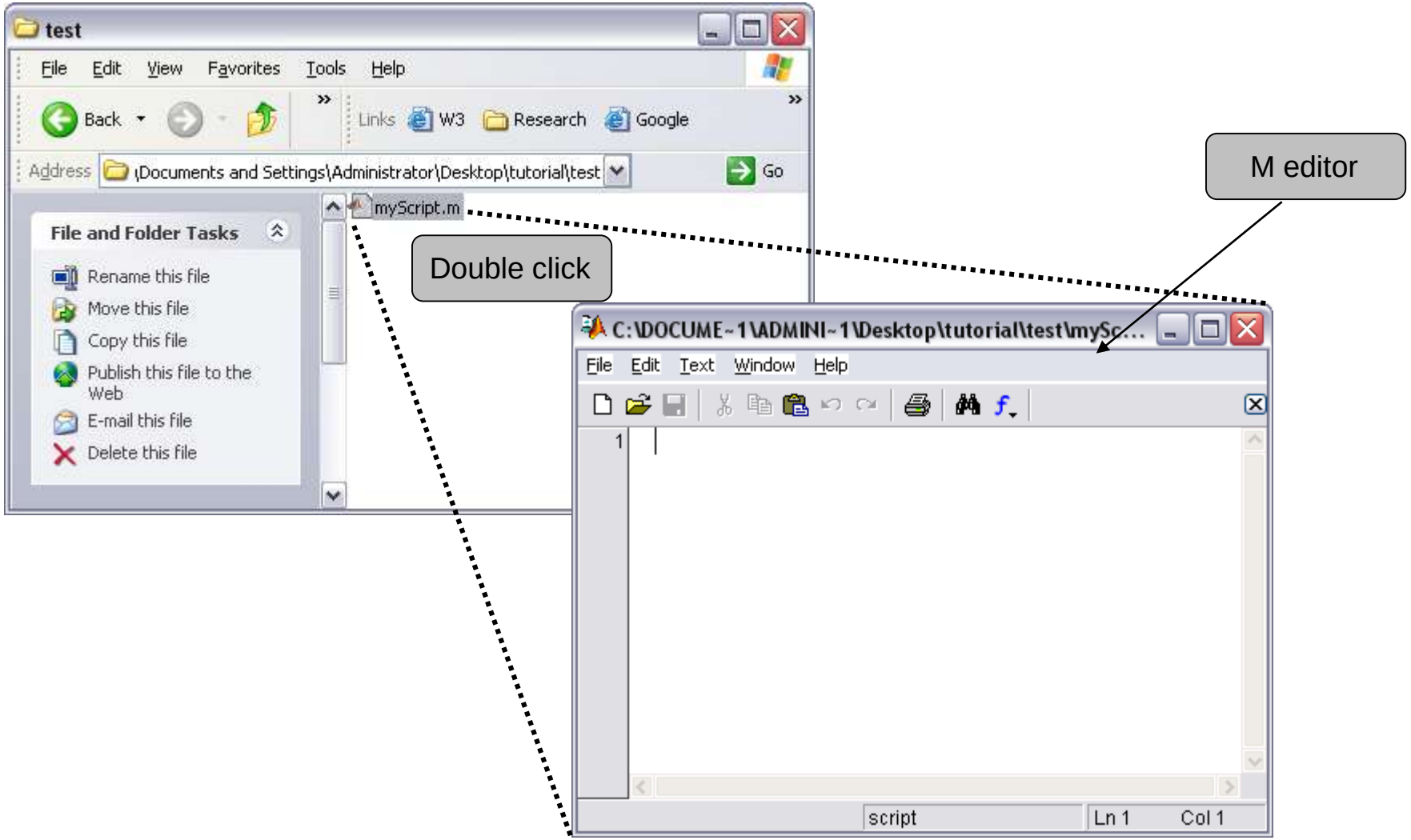
The value at position x-y of the array indicates the height of the surface

```
data = [1:10];
data = repmat(data,10,1); % create data
surface(data,'FaceColor',[1 1 1], 'Edgecolor', [0 0 1]); % plot data
view(3); grid on; % change viewpoint and put axis lines
```

How to create M files (I)



How to create M files (II)



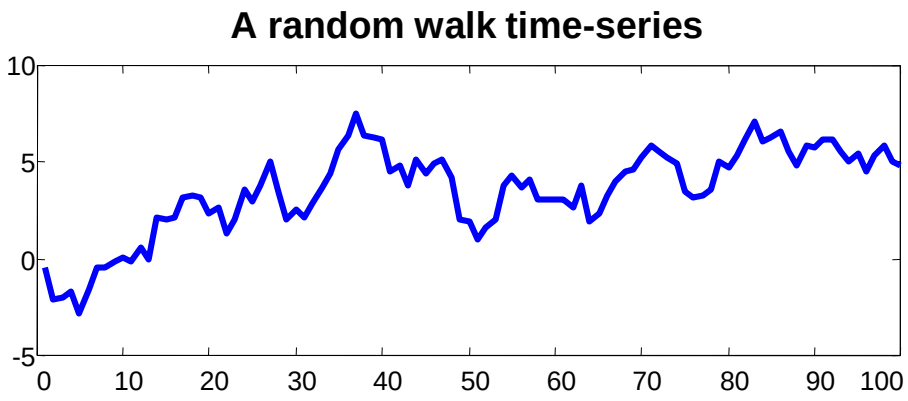
How to create M files (III)

- **The following script will create:**
  - An array with 10 random walk vectors
  - Will save them under text files: 1.dat, …, 10.dat

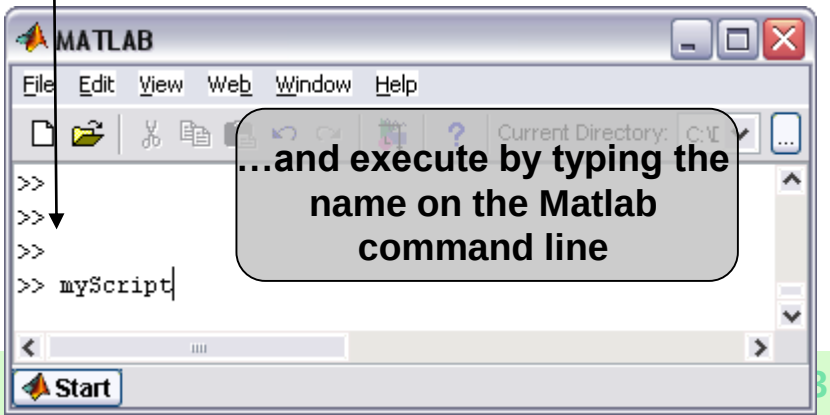
myScript.m

Sample Script

```
a = cumsum(randn(100,10)); % 10 random walk data of length 100
for i=1:size(a,2),         % number of columns
    data = a(:,i);
    fname = [num2str(i) '.dat']; % a string is a vector of characters!
    save(fname, 'data','-ASCII'); % save each column in a text file
end
```

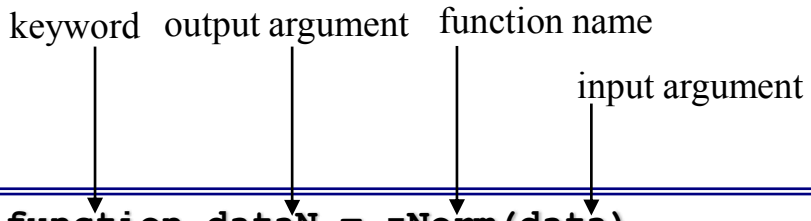


Write this in the M editor…



…and execute by typing the name on the Matlab command line

Functions in M scripts




```
function dataN = zNorm(data)
% ZNORM zNormalization of vector
% subtract mean and divide by std

if (nargin<1), % check parameters
    error('Not enough arguments');
end
data = data - mean(data); % subtract mean
data = data/std(data); % divide by std
dataN = data;
```

Help Text
(help *function_name*)

Function Body

```
function [a,b] = myFunc(data, x, y) % pass & return more arguments
```

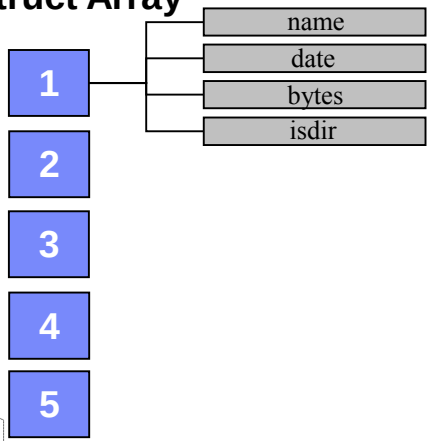
Cell arrays in matlab

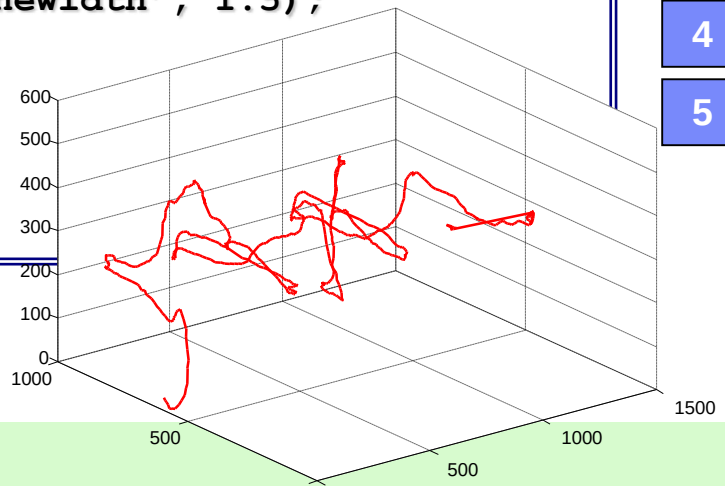
- **Cells that hold other Matlab arrays**
  - Let's read the files of a directory

```
>> f = dir('*.dat') % read file contents
f =
15x1 struct array with fields:
    name
    date
    bytes
    isdir
for i=1:length(f),
    a{i} = load(f(i).name);
    N = length(a{i});
    plot3([1:N], a{i}(:,1), a{i}(:,2), ...
          'r-', 'Linewidth', 1.5);
    grid on;
    pause;
    cla;
end
```

f(1).name

Struct Array





Reading and writing files

- **Load/Save are faster than C style I/O operations**
  - But fscanf, fprintf can be useful for file formatting or reading non-Matlab files

```
fid = fopen('fischer.txt', 'wt');

for i=1:length(species),
    fprintf(fid, '%6.4f %6.4f %6.4f %6.4f %s\n', meas(i,:), species{i});
end
fclose(fid);
```

Output file:

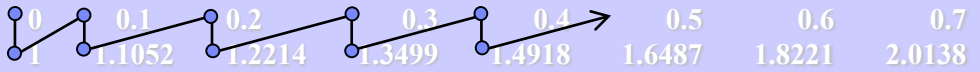
fischer.txt - Notepad

File Edit Format View Help

```
5.1000 3.5000 1.4000 0.2000 setosa
4.9000 3.0000 1.4000 0.2000 setosa
4.7000 3.2000 1.3000 0.2000 setosa
4.6000 3.1000 1.5000 0.2000 setosa
5.0000 3.6000 1.4000 0.2000 setosa
5.4000 3.9000 1.7000 0.4000 setosa
4.6000 3.4000 1.4000 0.3000 setosa
5.0000 3.4000 1.5000 0.2000 setosa
4.4000 2.9000 1.4000 0.2000 setosa
4.9000 3.1000 1.5000 0.1000 setosa
5.4000 3.7000 1.5000 0.2000 setosa
4.8000 3.4000 1.6000 0.2000 setosa
```

- **Elements are accessed column-wise (again…)**

```
x = 0:.1:1; y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);
```



Flow, control, loops

- if (else/elseif) , switch
  - Check logical conditions
- while
  - Execute statements infinite number of times
- for
  - Execute statements a fixed number of times
- break, continue
- return
  - Return execution to the invoking function

Advantages of vectorialization

```
clear all;
tic;
for i=1:50000
    a(i) = sin(i);
end
toc
```

elapsed_time =

5.0070

```
clear all;
a = zeros(1,50000);
tic;
for i=1:50000
    a(i) = sin(i);
end
toc
```

elapsed_time =

0.1400

```
clear all;
tic;
i = [1:50000];
a = sin(i);
toc;
```

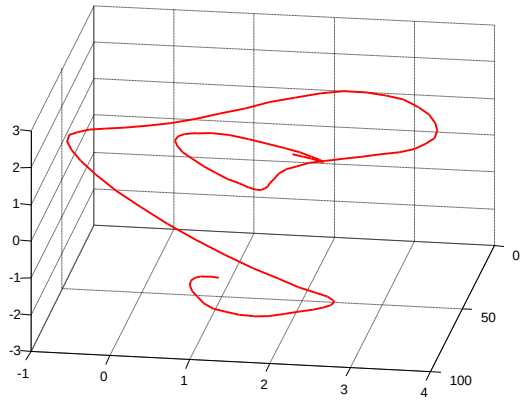
elapsed_time =

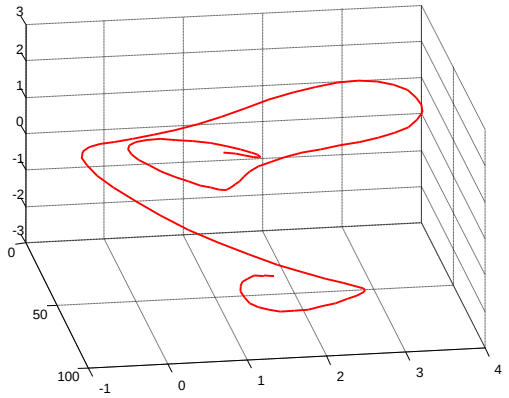
0.0200

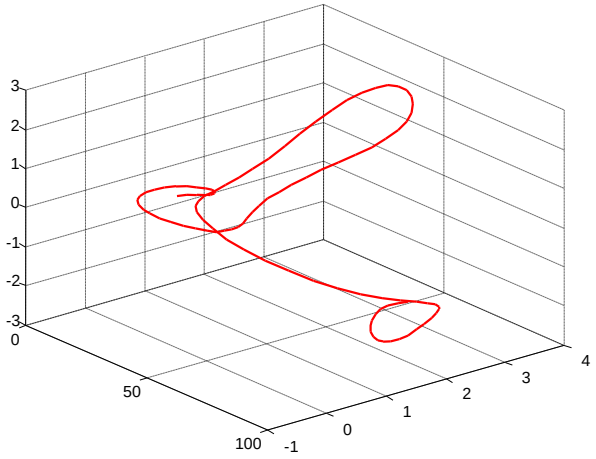
- **Pre-allocate arrays that store output results**
  - No need for Matlab to resize everytime
- **Functions are faster than scripts**
  - Compiled into pseudo-code
- **Load/Save faster than Matlab I/O functions**
- **After v. 6.5 of Matlab there is for-loop vectorization (interpreter)**
- **Vectorizations help, but not so obvious how to achieve many times**

Advanced Features – Making Animations

- Create animation by changing the camera viewpoint








```
azimuth = [50:100 99:-1:50]; % azimuth range of values
for k = 1:length(azimuth),
    plot3(1:length(a), a(:,1), a(:,2), 'r', 'Linewidth',2);
    grid on;
    view(azimuth(k),30); % change new
    M(k) = getframe; % save the frame
end

movie(M,20); % play movie 20 times
```

Matlab Toolboxes

There are many specialized toolboxes from Mathworks

- Image Processing, Statistics, Bio-Informatics, etc

There are many equivalent *free* toolboxes too:

- SVM toolbox
  - **http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox/**
- Wavelets
  - **http://www.math.rutgers.edu/~ojanen/wavekit/**
- Speech Processing
  - **http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html**
- Bayesian Networks
  - **http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html**

Helpers

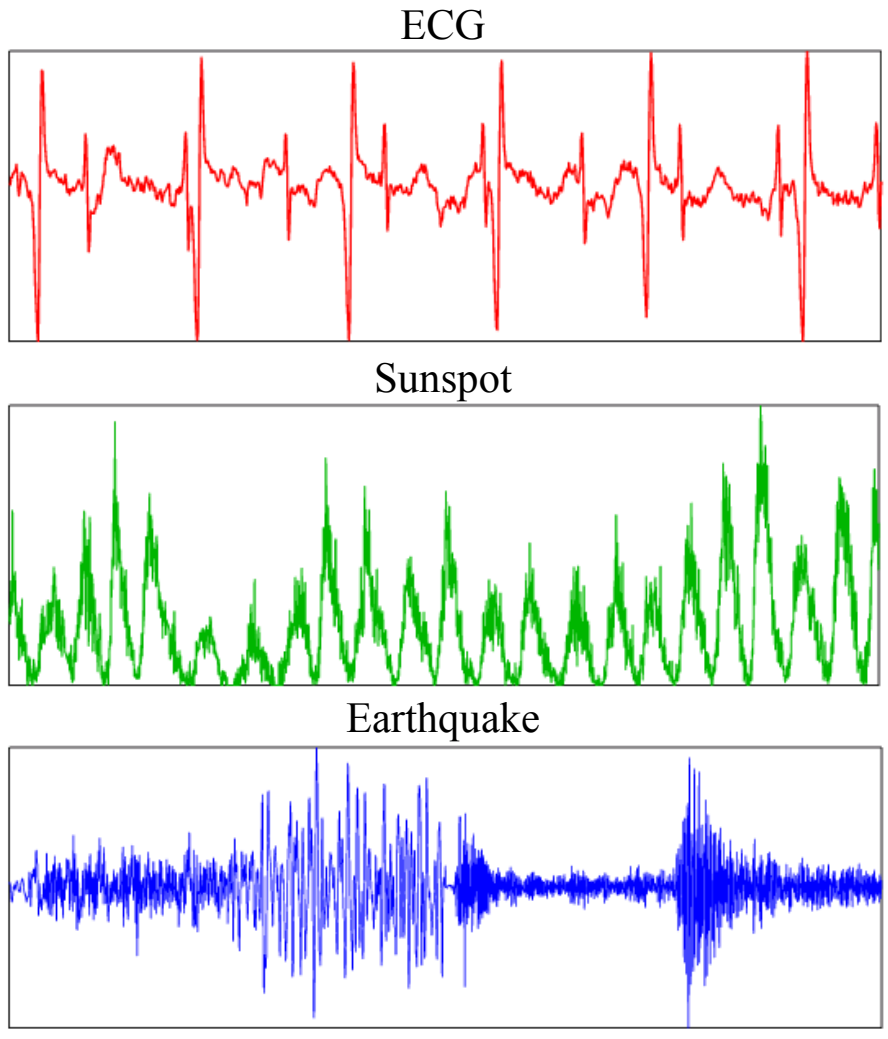
- **help [command] (on the command line)**
  eg. `help fft`
- **Menu: help -> matlab help**
  - Excellent introduction on various topics
- **Matlab webinars**
  - http://www.mathworks.com/company/events/archived_webinars.html?fp
- **Google groups**
  - **comp.soft-sys.matlab**
  - **You can find *anything* here**
  - **Someone else had the same problem before you!**

Time series (I)

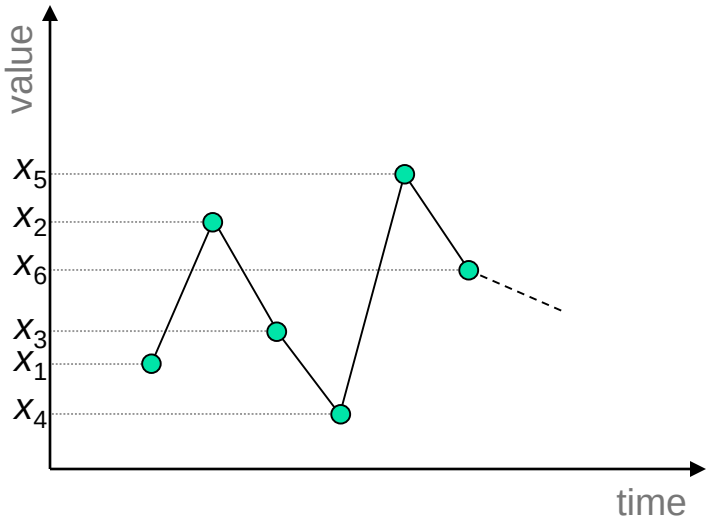
Definition: *A sequence of measurements over time*

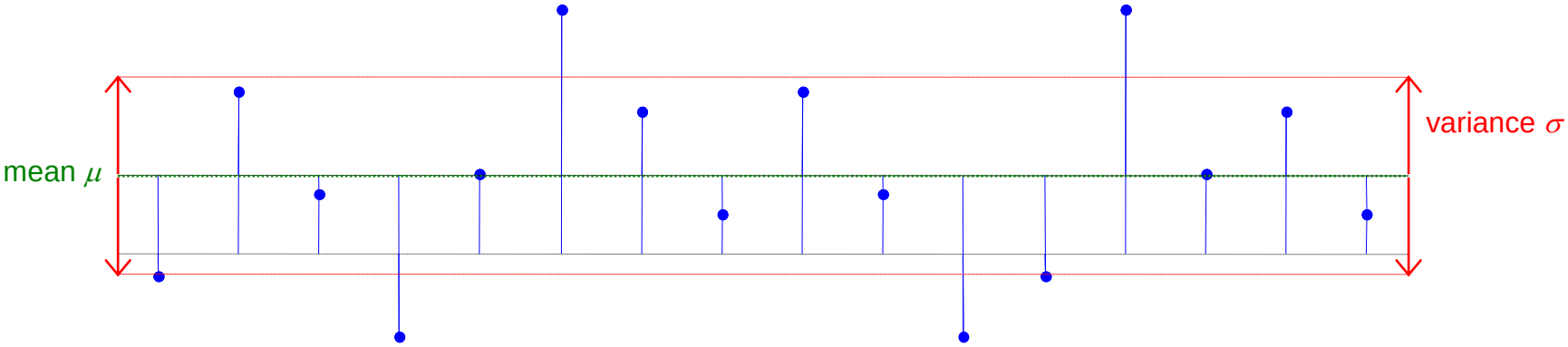
- ***Medicine***
- ***Stock Market***
- ***Meteorology***
- ***Geology***
- ***Astronomy***
- ***Chemistry***
- ***Biometrics***
- ***Robotics***
- ***Altimetry (SSH, SLA)***

64.0
62.8
62.0
66.0
62.0
32.0
86.4
. . .
21.6
45.2
43.2
53.0
43.2
42.8
43.2
36.4
16.9
10.0
…



Time series (II)





Normalization

- Intuitively, the notion of "shape" is *generally* independent of
  - Average level (mean)
  - Magnitude (variance)
- Unless otherwise specified, we normalize to zero mean and unit variance

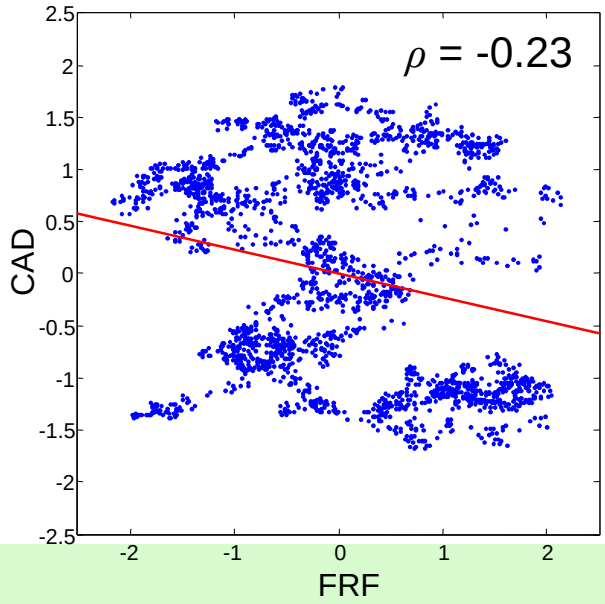
Correlation and similarity

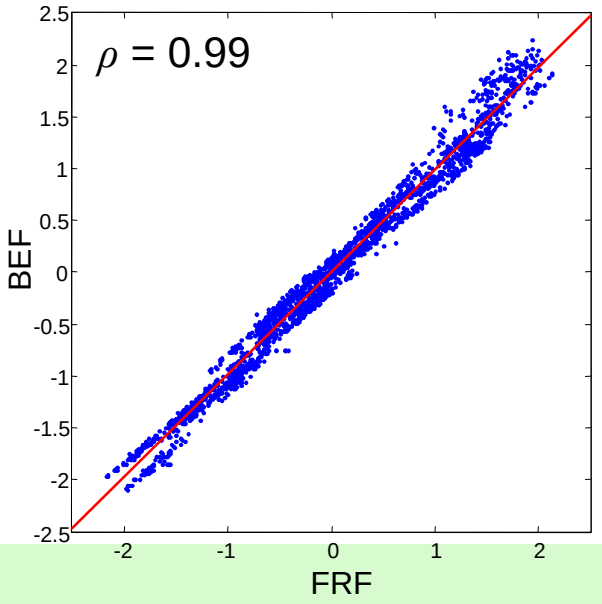
$$y_t = \alpha x_t + \epsilon_t, \quad \text{for } 1 \le t \le N$$

residual

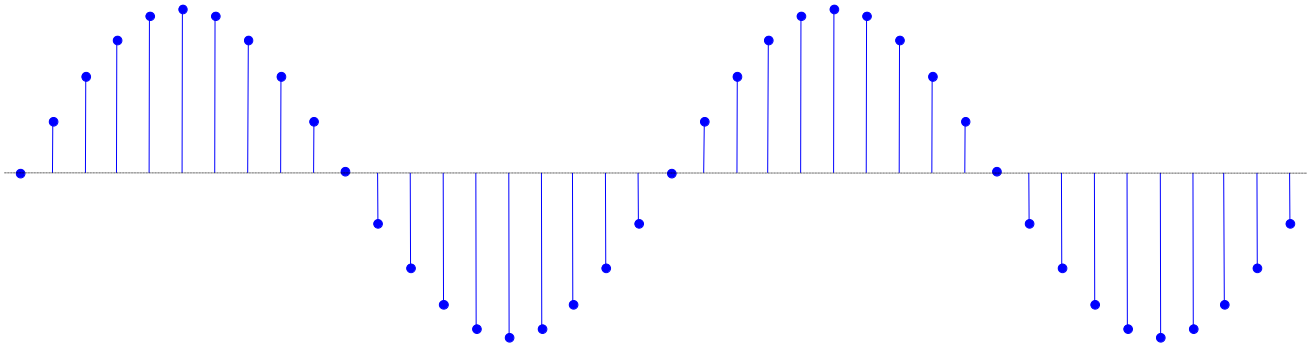
$$\alpha = \rho_{x,y} \quad \text{and} \quad \sum_t \epsilon_t^2 / N = 1 - \rho_{x,y}^2$$

slope





Fourier transform



- One cycle every 20 time units (period)

Example (I)

- Let's write a script to generate a sine function 2048 points long.
- Generate exactly 64 full cycles over this interval.
- Plot the results and add labels to the axes.

T_0

Result

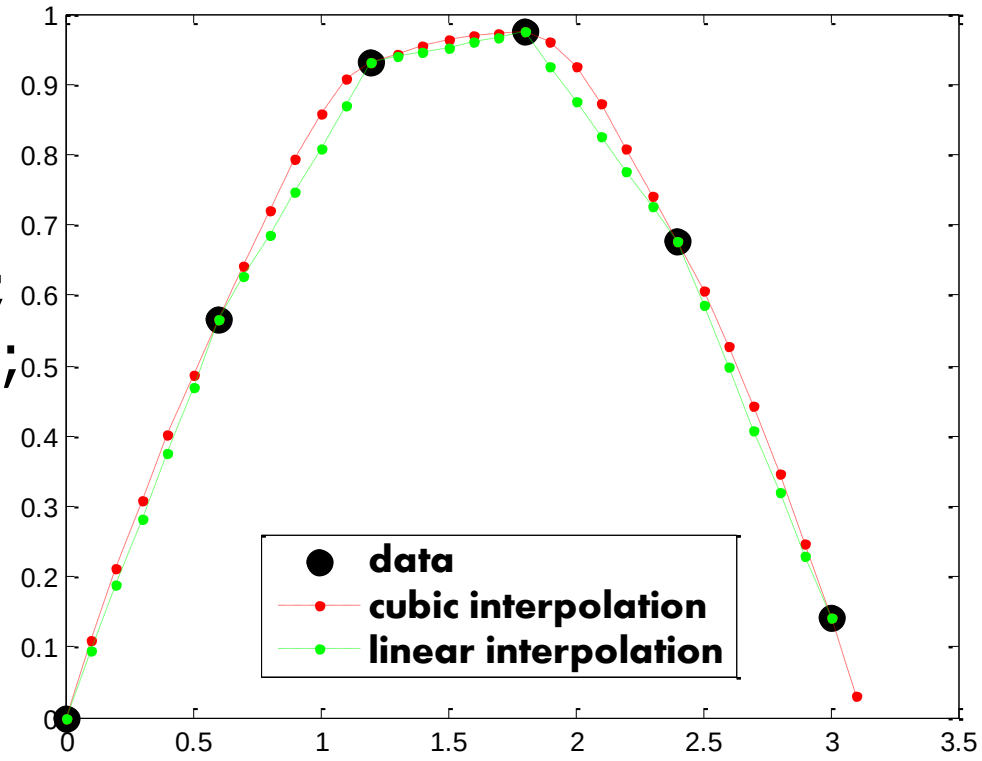
```
% Create a sine function 2048 points long...
%
figure(1)
nx=2048;

% 64 Full cycles
nc=2/2048;
%nc=64/2048;
x=0:nx-1;
%
%   now make the function and plot it.
%

y=sin(x*pi*nc);
plot(x,y)
```

- xlabel('x')
- ylabel('y')

T_0

Example interpolation

```
x = 0:.6:pi;
y = sin(x);
xi = 0:.1:pi;

figure
yi = interp1(x,y,xi,'cubic');
yj = interp1(x,y,xi,'linear');
plot(x,y,'ko')
hold on
plot(xi,yi,'r:')
plot(xi,yj,'g.:')
```



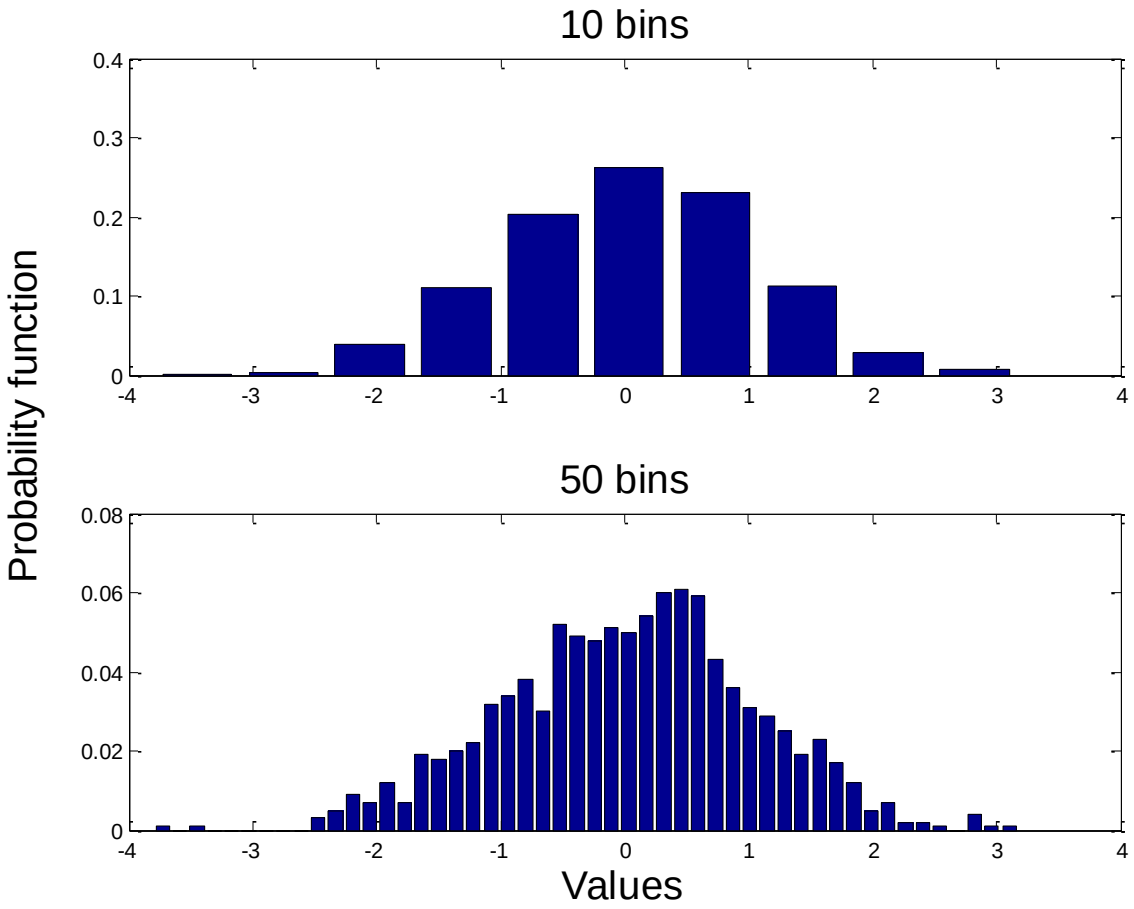
Histograms

```
X = randn(1,1000);
[C, N] = hist(X, 50);
bar(N,C/sum(C))
```

(N = location of bins,

C = counts in each location)

```
[C, N] = hist(X, 10);
bar(N,C/sum(C))
```



Vectorizing code

You might from time to time be tempted to create a matrix by defining each element one-by-one in a for loop or something like that. That will work but using

Consider two ways to create a 10,000 x 1 vector
[1, 4, 9, 16,…,10000^2]

tic
a=zeros(1,10000);
for i=1:10000
  a(i)=i^2;
end
toc
	tic
	a=[1:10000].^2;
	toc

Elapsed time is 0.000282 seconds.	Elapsed time is 0.000063 seconds.

The code on the right is said to be *vectorized*[To]. It's usually a good idea to try to vectorize your code. Just don't go crazy with it.

Geo-referencing in matlab

```
% Extract and display a subset of full resolution data for the state of
% Massachusetts.
% Read the stateline polygon boundary and calculate boundary limits.
   Massachusetts = shaperead('usastatehi', 'UseGeoCoords', true, ...
   'Selector',{@(name) strcmpi(name,'Massachusetts'), 'Name'});
   latlim = [min(Massachusetts.Lat(:)) max(Massachusetts.Lat(:))];
   lonlim = [min(Massachusetts.Lon(:)) max(Massachusetts.Lon(:))];

% Read the gtopo30 data at full resolution.
   [Z,refvec] = gtopo30('W100N90',1,latlim,lonlim);
% Display the data grid and
% overlay the stateline boundary.
   figure
   usamap('Massachusetts');
   geoshow(Z, refvec, 'DisplayType', 'surface')
   geoshow([Massachusetts.Lat], ...
                  [Massachusetts.Lon],'Color','m')
   demcmap(Z)
```