

Beyond the Mouse – A Short Course on Programming

5. Matlab IO: Getting data in and out of Matlab

Ronni Grapenthin and Glenn
Thompson

Geophysical Institute, University of Alaska
Fairbanks

October 11, 2010

YOU'LL NEVER FIND A
PROGRAMMING LANGUAGE
THAT FREES YOU FROM
THE BURDEN OF
CLARIFYING
YOUR IDEAS.



"The Uncomfortable Truths Well",
<http://xkcd.com/568> (April 13, 2009)

Outline

- 1 File access
- 2 Plotting Data
- 3 Annotating Plots
- 4 Many Data - one Figure
- 5 Saving your Figure
- 6 Misc
- 7 Examples

Outline

- 1 File access
- 2 Plotting Data
- 3 Annotating Plots
- 4 Many Data - one Figure
- 5 Saving your Figure
- 6 Misc
- 7 Examples

File access 1: Excel-data

`xlsread`

- `[num, txt, raw] = xlsread('myfile.xls', 'sheet23');`
attempts to read sheet 23 (first sheet if parameter omitted)
- `num` – a matrix that contains all numeric data
- `txt` – a cell array that contains all text data
- `raw` – cell array with columns `xlsread` could not interpret

File access 1: Excel-data

`xlsread`

- `[num, txt, raw] = xlsread('myfile.xls', 'sheet23');`
attempts to read sheet 23 (first sheet if parameter omitted)
- `num` – a matrix that contains all numeric data
- `txt` – a cell array that contains all text data
- `raw` – cell array with columns `xlsread` could not interpret

`xlswrite`

- `[status, msg] = xlswrite('myfile.xls', M, 'sheet42');`
attempts to write matrix `M` to sheet 42 of `myfile.xls`
- `status` – 1 on success, 0 on error
- `msg` – error message object with fields `message` and `identifier`

File access 1: Excel-data

`xlsread`

- `[num, txt, raw] = xlsread('myfile.xls', 'sheet23');`
attempts to read sheet 23 (first sheet if parameter omitted)
- `num` – a matrix that contains all numeric data
- `txt` – a cell array that contains all text data
- `raw` – cell array with columns `xlsread` could not interpret

`xlswrite`

- `[status, msg] = xlswrite('myfile.xls', M, 'sheet42');`
attempts to write matrix `M` to sheet 42 of `myfile.xls`
- `status` – 1 on success, 0 on error
- `msg` – error message object with fields `message` and `identifier`

See Also:

`dlmread`, `dlmwrite`, `csvread`, `csvwrite`

File access 1.5: Opening and closing files

`fopen`

- `fid = fopen('filename', mode);`
Open a file, **do not discard fid!**
- mode is:
 - 'r' – read (default)
 - 'w' – write (overwrite if file exists)
 - 'a' – append (append if file exists)

Wherever **fid** is used in the following, these functions must be used!

File access 1.5: Opening and closing files

`fopen`

- `fid = fopen('filename', mode);`
Open a file, **do not discard fid!**
- mode is:
 - 'r' – read (default)
 - 'w' – write (overwrite if file exists)
 - 'a' – append (append if file exists)

`fclose`

- `fid = fclose(fid);`
close file with identifier `fid`

Wherever **fid** is used in the following, these functions must be used!

File access 1.5: Opening and closing files

`fopen`

- `fid = fopen('filename', mode);`
Open a file, **do not discard fid!**
- mode is:
 - 'r' – read (default)
 - 'w' – write (overwrite if file exists)
 - 'a' – append (append if file exists)

`fclose`

- `fid = fclose(fid);`
close file with identifier `fid`

Wherever **fid** is used in the following, these functions must be used!

File access 2: Text Files 1/3

textread

- `[A, B, C, ...] = textread('filename', 'format', N);`
reads data from file 'filename' to **multiple outputs** A,B,C,... using specified `format` until **entire** file is read, or `N` times.

File access 2: Text Files 1/3

textread

- `[A, B, C, ...] = textread('filename', 'format', N);`
reads data from file 'filename' to **multiple outputs** A,B,C,... using specified `format` until **entire** file is read, or `N` times.

textscan

- `C = textscan(fid, 'format', N);`
reads data from file `fid` **OR** a string to cell array `C` using specified `format` until **entire** file is read, or `N` times (resume from where left by calling `textscan` again later).

File access 2: Text Files 1/3

`textread`

- `[A, B, C, ...] = textread('filename', 'format', N);`
reads data from file 'filename' to **multiple outputs** A,B,C,... using specified `format` until **entire** file is read, or `N` times.

`textscan`

- `C = textscan(fid, 'format', N);`
reads data from file `fid` **OR** a string to cell array `C` using specified `format` until **entire** file is read, or `N` times (resume from where left by calling `textscan` again later).

Use `textscan` if you want ...

- to read large files (better performance than `textread`)
- **one** cell array as opposed to many outputs
- read from any point in the file (use `fseek` on `fid` first)
- more options and choices in data conversion (see doc)

`fprintf`

- `count = fprintf(fid, 'format', A, ...);`
formats data in matrix `A` (and additional arguments) according to format string and writes to the file associated with `fid`
- `count` – number of bytes written

`fprintf`

- `count = fprintf(fid, 'format', A, ...);`
formats data in matrix `A` (and additional arguments) according to format string and writes to the file associated with `fid`
- `count` – number of bytes written

`fprintf`

- `count = fprintf(fid, 'format', A, ...);`
formats data in matrix `A` (and additional arguments) according to format string and writes to the file associated with `fid`
- `count` – number of bytes written

See also

- `dlmwrite`: Write matrix to ASCII delimited file
- `csvwrite`: Write matrix to comma-separated value file

fprintf example

```
clear all, clc, close all;  
  
% create data here  
x = 1:10  
y = rand(1,10)  
  
% open a file in write mode  
fout = fopen('random_numbers.txt', 'w');  
  
% write our data:  
% x is first column,  
% y is second column  
fprintf(fout, '%d\t%f\n', [x; y])  
  
% don't forget to close the file!  
fclose(fout)
```


fprintf example

```
clear all, clc, close all;  
  
% create data here  
x = 1:10  
y = rand(1,10)  
  
% open a file in write mode  
fout = fopen('random_numbers.txt', 'w');  
  
% write our data:  
% x is first column,  
% y is second column  
fprintf(fout, '%d\t%f\n', [x; y])  
  
% don't forget to close the file!  
fclose(fout)
```

output

```
1 0.706046  
2 0.031833  
3 0.276923  
4 0.046171  
5 0.097132  
6 0.823458  
7 0.694829  
8 0.317099  
9 0.950222  
10 0.034446
```

Outline

- 1 File access
- 2 Plotting Data**
- 3 Annotating Plots
- 4 Many Data - one Figure
- 5 Saving your Figure
- 6 Misc
- 7 Examples

plot

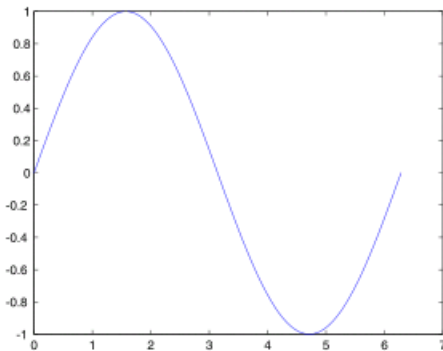
Graphics

[edit]

Function `plot` can be used to produce a graph from two vectors `x` and `y`. The code:

```
x = 0:pi/100:2*pi;  
y = sin(x);  
plot(x,y)
```

produces the following figure of the `sine` function:



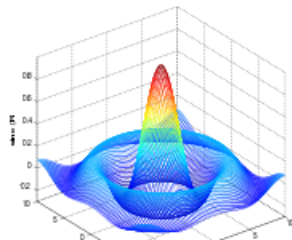
TEXT TOO SMALL!
LOOK AT YOUR SCREEN!

Three-dimensional graphics can be produced using the functions `surf`, `plot3` or `mesh`.

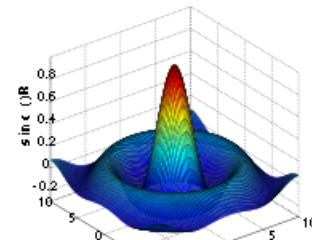
```
[X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);  
f = sinc(sqrt((X/pi).^2+(Y/pi).^2));  
mesh(X,Y,f);  
axis([-10 10 -10 10 -0.3 1])  
xlabel('\bfx')  
ylabel('\bfy')  
zlabel('\bfsinc' '(\bfR)')  
hidden off
```

```
[X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);  
f = sinc(sqrt((X/pi).^2+(Y/pi).^2));  
surf(X,Y,f);  
axis([-10 10 -10 10 -0.3 1])  
xlabel('\bfx')  
ylabel('\bfy')  
zlabel('\bfsinc' '(\bfR)')
```

This code produces a **wireframe** 3D plot of the two-dimensional unnormalized `sinc` function:

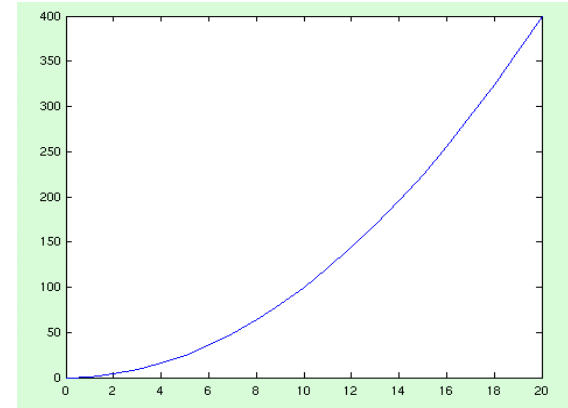


This code produces a **surface** 3D plot of the two-dimensional unnormalized `sinc` function:



2D plotting

1. Define x-vector `>> x = 1:20;`
2. Define y-vector `>> y = x^2;`
3. `plot(x,y)` `>> plot(x, y)`



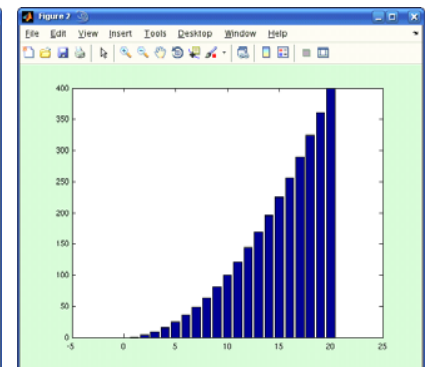
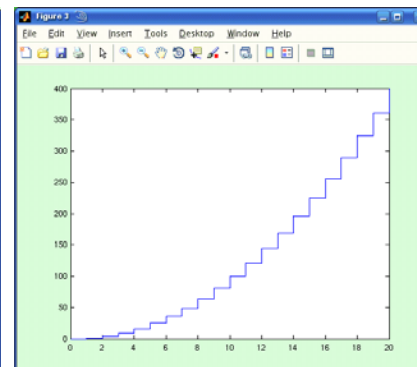
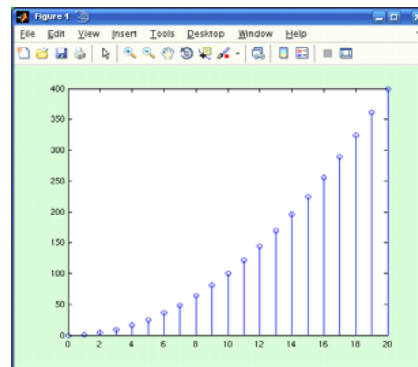
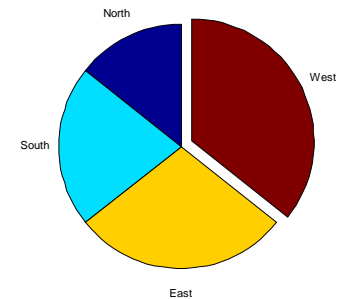
plot just gives a normal x-y graph with linear axes.

There are other 2D plotting commands, e.g:

semilogy, semilogx, loglog

stem, stairs, bar

pie, hist



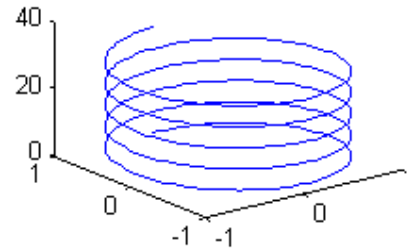
3D plotting

1. Define x-vector
2. Define y-vector
3. Define z-vector
4. `plot3(x,y,z)`

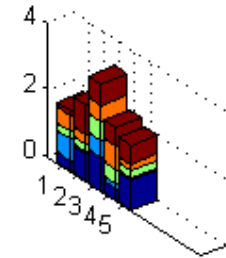
There are other 3D plotting commands, e.g:
`surf`, `mesh`, `contour`
`pie3`, `bar3`, `hist3`

Examples of simple 3D plots

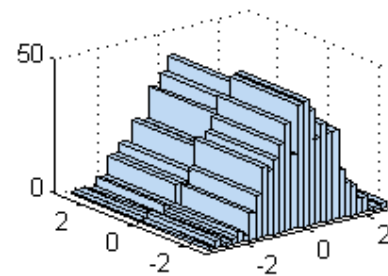
plot3



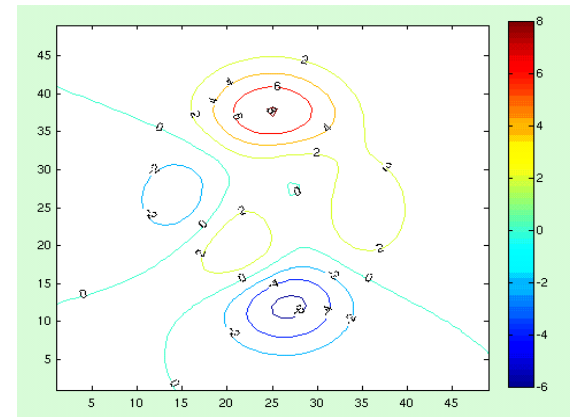
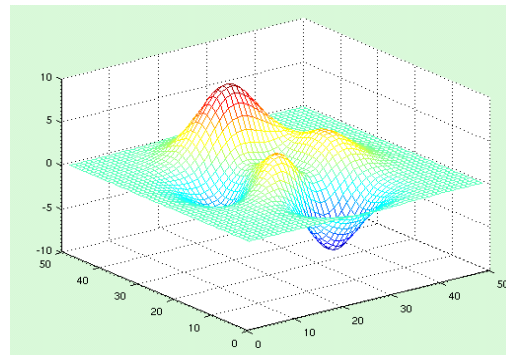
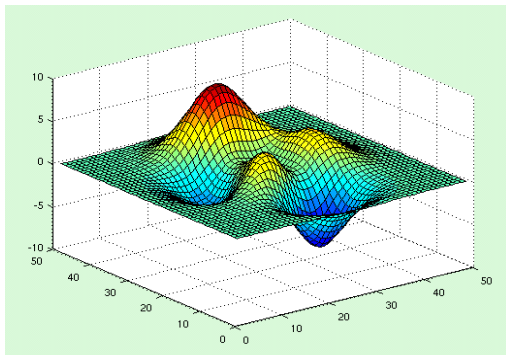
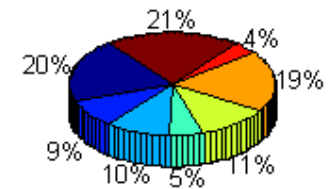
bar3



hist3



pie3

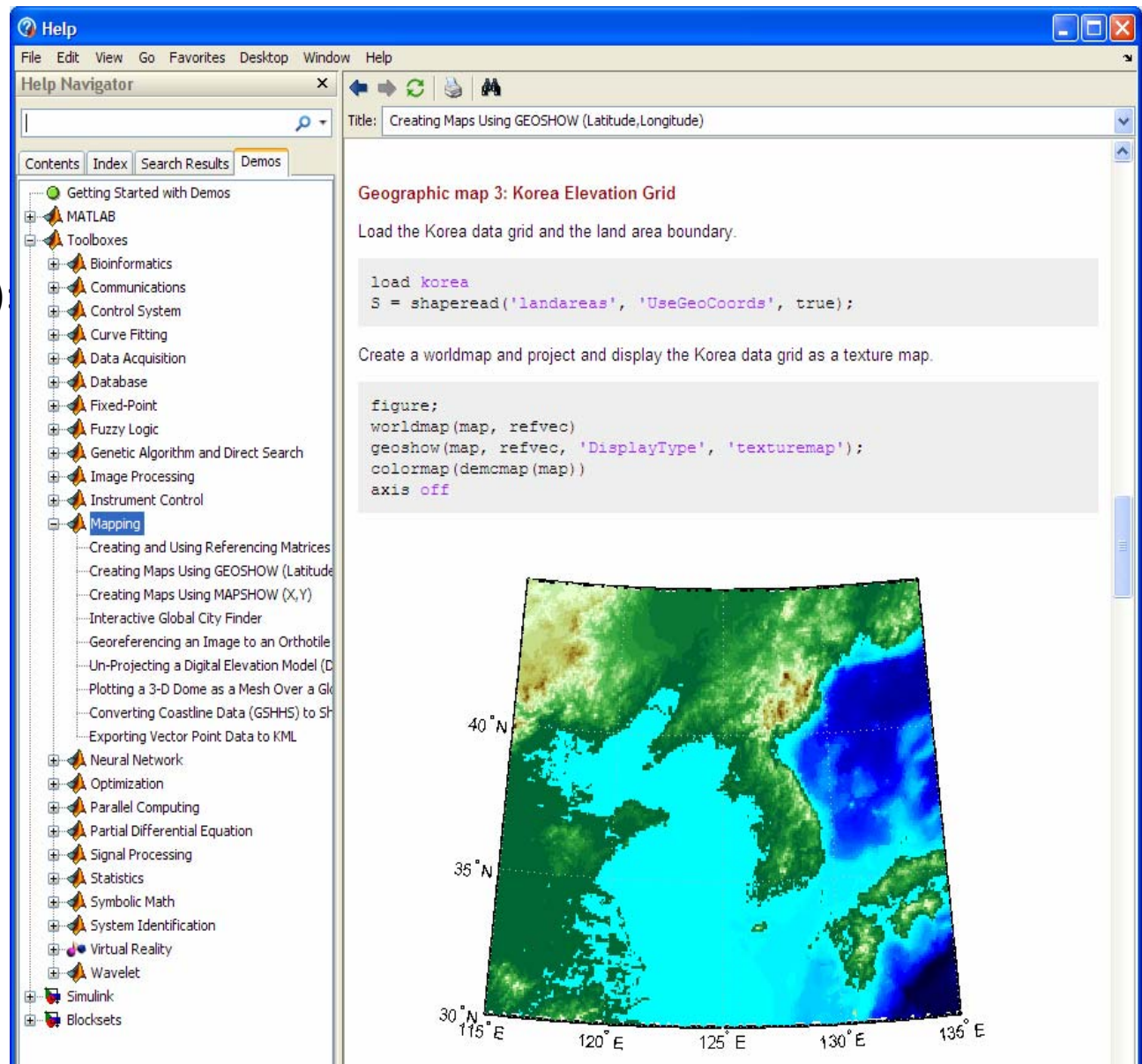


Plotting maps: the Mapping Toolbox

>> help map
>> mapdemos

Can write KML (GoogleEarth)
>> help kmlwrite

Alternative to GMT



The screenshot shows the MATLAB Help Navigator window. The left pane displays a tree view of toolboxes, with the 'Mapping' toolbox selected. The right pane shows the help page for 'Creating Maps Using GEOSHOW (Latitude,Longitude)'. The page title is 'Geographic map 3: Korea Elevation Grid'. The text describes loading the Korea data grid and the land area boundary, and creating a worldmap and projecting and displaying the Korea data grid as a texture map. The code block shows the following MATLAB commands:

```
load korea
S = shaperead('landareas', 'UseGeoCoords', true);

figure;
worldmap(map, refvec)
geoshow(map, refvec, 'DisplayType', 'texturemap');
colormap(demcmap(map))
axis off
```

Below the code block is a 3D topographic map of Korea, showing elevation data. The map is displayed in a perspective view, with latitude and longitude axes. The latitude axis ranges from 30°N to 40°N, and the longitude axis ranges from 115°E to 136°E. The map shows the Korean peninsula and surrounding regions, with elevation data represented by a color scale from blue (low elevation) to red (high elevation).

Outline

- 1 File access
- 2 Plotting Data
- 3 Annotating Plots**
- 4 Many Data - one Figure
- 5 Saving your Figure
- 6 Misc
- 7 Examples

Changing the line style: plot(x,y,s)

By default, plot(x,y) uses a blue line to connect data points

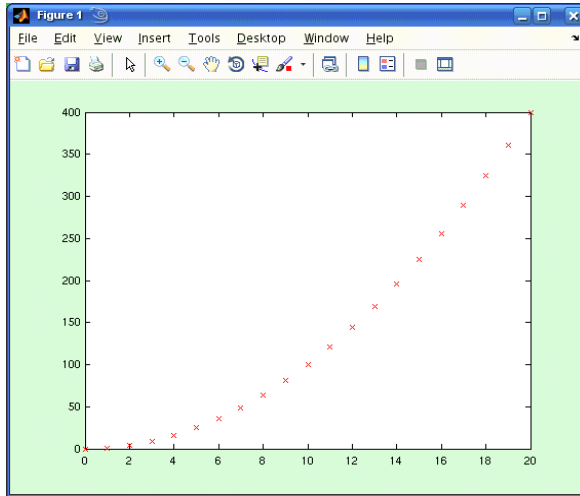
>> help plot

Various line types, plot symbols and colors may be obtained with PLOT(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

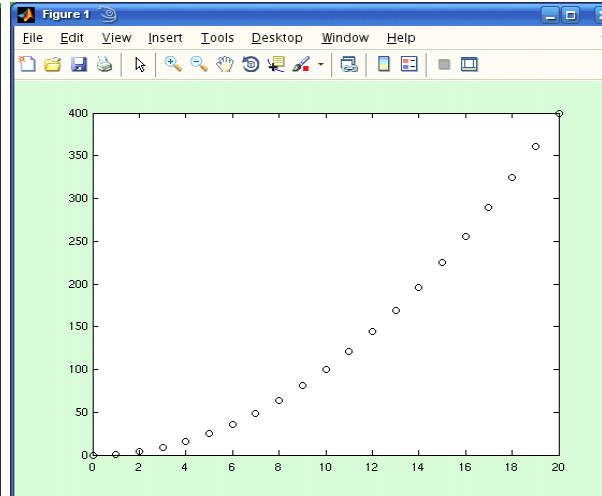
plot(x,y,s)

plot(x,y,'rx')



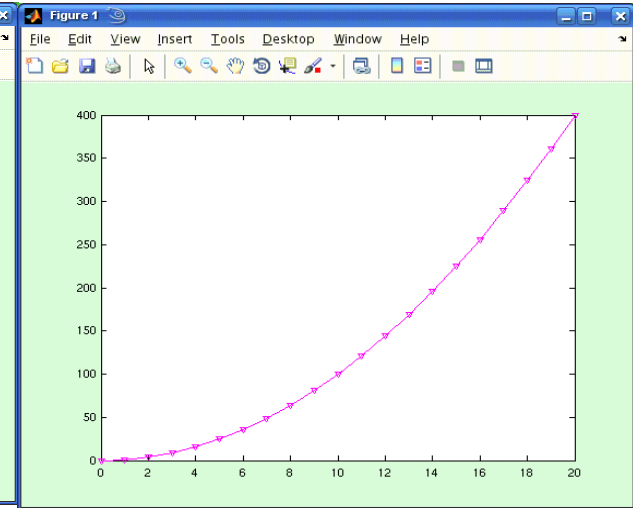
red crosses

plot(x,y,'^kbo')



black circles

plot(x, y, 'mv-')



magenta triangles + line

Labelling axes

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>> xlabel('time elapsed (s)');
>> ylabel('distance fallen (m)');
>> title('Plot showing that distance \alpha time^2');
>> grid on
fx >>
```

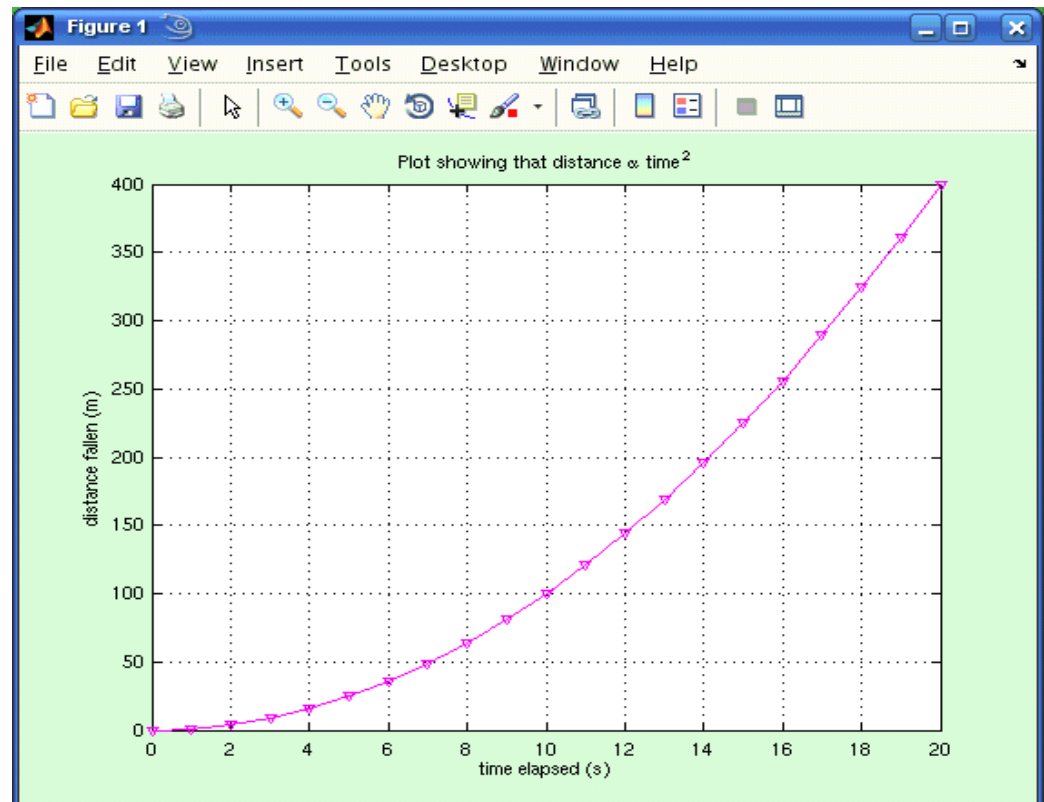
xlabel
ylabel
title
grid on

Superscripts: 'time^2' => time²

Subscripts: 'SO_2' => SO₂

Greek characters: \alpha => α

LaTeX Interpreter, doesn't
work for '...TickLabel'



Adding text

To add text at the position `xpos`, `ypos` to the current axes use:

```
>> text(xpos, ypos, 'some_string');
```

Remember you can use `sprintf`.

```
>> text(2.3, 5.1, sprintf('station %s',station{stationNum}) );
```

Changing the data range shown

Default: show all the data.

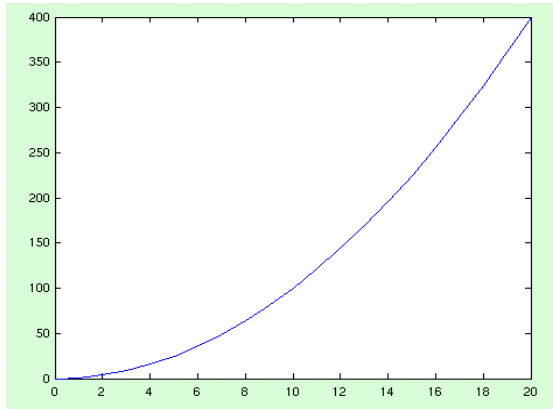
To override use:

```
>> set(gca, 'XLim', [xmin xmax]); % x-axis only
```

```
>> set(gca, 'YLim', [ymin ymax]); % y-axis only
```

```
>> set(gca, 'XLim', [xmin xmax], 'YLim', [ymin ymax]); % both axes
```

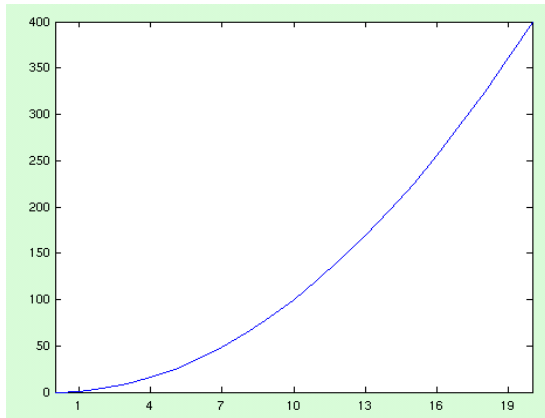
Changing the tick positions/labels



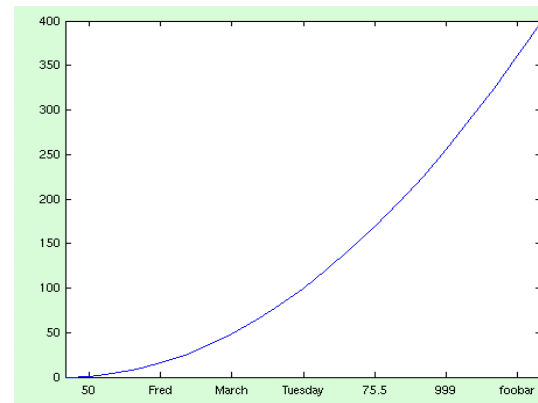
```
>> get(gca, 'XTick')  
ans =  
    0     2     4     6     8    10    12    14    16    18    20
```

TOO SMALL! Look at your screen!

```
set(gca, 'XTick', 1:3:22)
```



```
set(gca, 'XTickLabel', {50, 'Fred', 'March', 'Tuesday', 75.5, 999, 'foobar'})
```



Plotting against date/time: datenum & datetick

datenum() returns the day number (and fractional day number) in the calendar starting 1st January in the year 0 AD.

Excel dates and times are similar except Excel uses the origin 1st January 1900. But you normally ask Excel to format those cells with a particular date/time format, so you don't see the raw numbers. In MATLAB, datenum gives those raw numbers.

To convert from Excel day-numbers to MATLAB datenum format:

```
mtime = etime + datenum(1900, 1, 1);
```

Call it like:

```
datenum(YYYY, MM, DD)
```

```
datenum(YYYY, MM, DD, hh, mi, ss)
```

```
datenum('2009/04/29 18:27:00')
```

Remember to use vectorisation:

```
redoubtEventTimes = {'2009/03/22 22:38'; '2009/03/23 04:11'; '2009/03/23 06:23'}
```

```
dnum = datenum(redoubtEventTimes); % result is a 3 x 1 vector of datenums.
```

```
datetick('x'); % can give unexpected results, ask for help.
```

datestr

I often use dates in plot labels, or in file paths/names.

datestr(array, dateform) is used to generate a human-readable string from an array of dates/times in datenum format.

```
>> lectureTime = datenum(2009, 4, 29, 12, 30, 0)
733890.5208
>> datestr(lectureTime, 30)
20090427T123000
>> datestr(lectureTime, 31)
2009-04-29 12:30:00
>> datestr(lectureTime, 'mm/dd/yyyy')
04/29/2009
>> xlabel( sprintf('This plot was generated at %s', datestr(now, 31) ) );
```

An aside – making dates work for you:

YYYYMMDD, not MMY YDD (U.S.) or DDMMYY (Europe).

Outline

- 1 File access
- 2 Plotting Data
- 3 Annotating Plots
- 4 Many Data - one Figure**
- 5 Saving your Figure
- 6 Misc
- 7 Examples

MATLAB Graphics Object Hierarchy

Screen

Figure1

Axis1 (xlabel, ylabel, title, tick marks, tick labels)

Graph1 (linestyle, legendlabel)

Graph2

...

Axis2

Graph1

...

Figure2

Axis1

Graph1

Graph2

Axis2

Graph1

...

figure

axes

plot

figure

To create a new figure with no axes:

```
>> figure;
```

To highlight a figure that is already displayed (if it doesn't already exist, it will be created):

```
>> figure(2)
```

To get all the properties associated with a figure:

```
>> get(figure(2))
```

To get a particular property associated with a figure:

```
>> get(figure(1), 'Position')
```

```
[420 528 560 420]
```

To modify a particular property associated with a figure:

```
>> set(figure(1), 'Position', [100 100 560 420])
```

This particular example will just move where figure(1) is plotted on the screen.

To get a 'handle' for the current active figure window use **gcf**.

```
>> get(gcf, 'Position')
```

Will return the screen position of the current active figure window.

axes

New figures are created without a set of axes.

To get a 'handle' for the current active set of axes use **gca** (get current axes).

Example: get a list of all properties associated with current axes

```
>> get(gca)
```

```
>> get(gca, 'position')
```

This will return the screen position of the current active figure window, which by default is:

```
[0.13 0.11 0.775 0.815]
```

Format here is [xorigin yorigin xwidth yheight] in fractions of the figure window width.

To modify the position of the current axes within a figure:

```
>> set(gca, 'position', [0.2 0.3 0.6 0.4])
```

The axes would start 20% of the way across the screen, 30% of the way up, and be 60% the screen width, and 40% the screen height.

An alternative syntax is just to call the axes command:

```
>> axes('position', [0.2 0.3 0.6 0.4]);
```

Either will create a figure if none already exists. Or modify the current set of axes on the current figure.

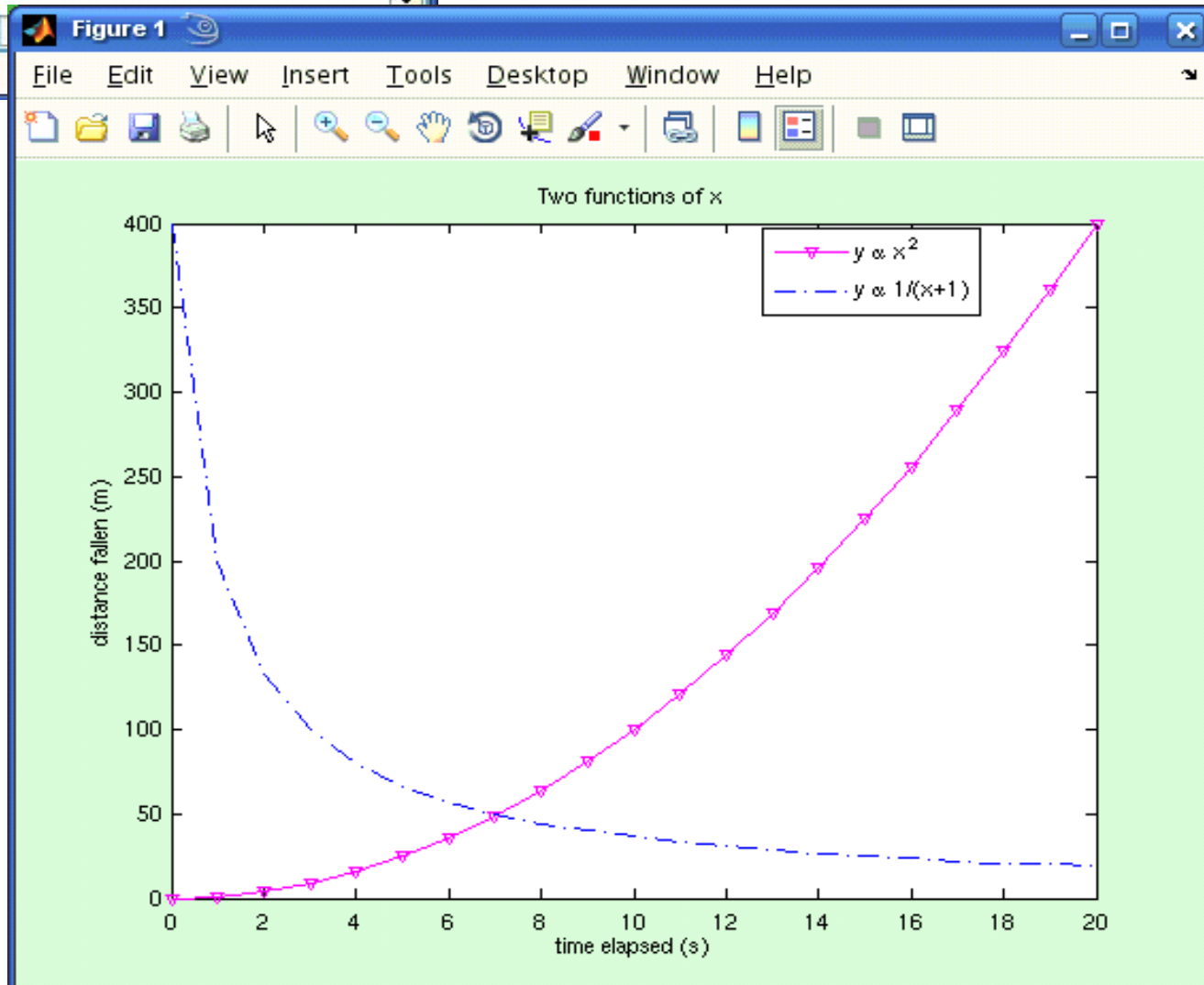
Multiple plots on a figure 1: hold on

```
>> grid off
>> y2 = 400./(x+1);
>> hold on
>> plot(x,y2,'b-.')
>> title('Two functions of x')
>> legend('y \alpha x^2', 'y \alpha 1/(x+1)')
fx >>
```

hold on “holds on” to graphs already in the current axes.
Normally they would be erased

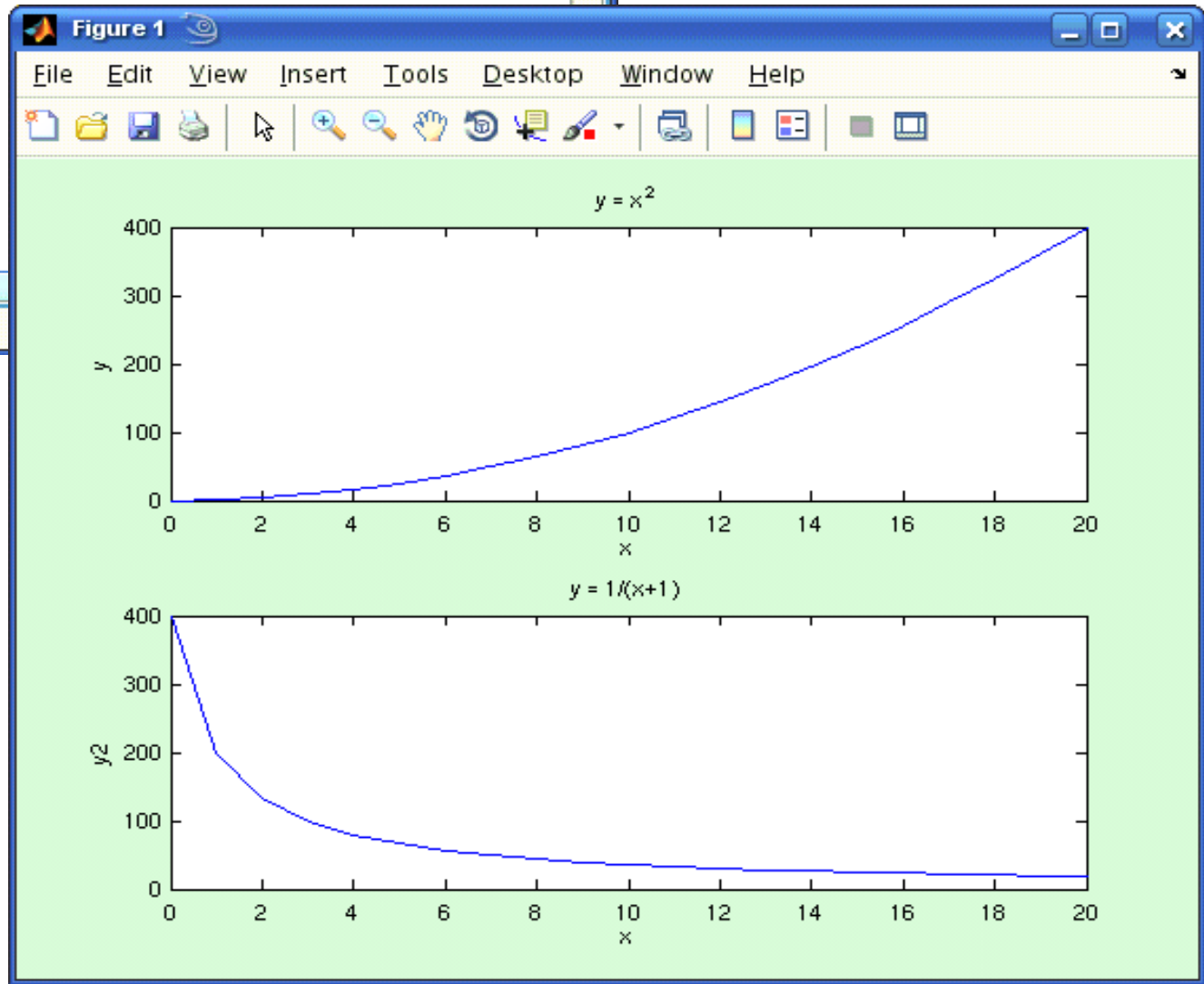
hold on
plot(x,y,'-.')
title
legend
hold off

If your graphs have very different scales, and you have just two, try **plotyy**



Multiple plots on a figure 2: subplot

```
New to MATLAB? Watch this Video, see Demos, or read Getting Started.  
  
>> close all  
>> figure  
>> subplot(2,1,1), plot(x,y)  
>> title('y = x^2')  
>> xlabel('x')  
>> ylabel('y')  
>> subplot(2,1,2), plot(x,y2)  
>> title('y = 1/(x+1)')  
>> xlabel('x')  
>> ylabel('y2')  
>>  
fx >> |  
Start
```



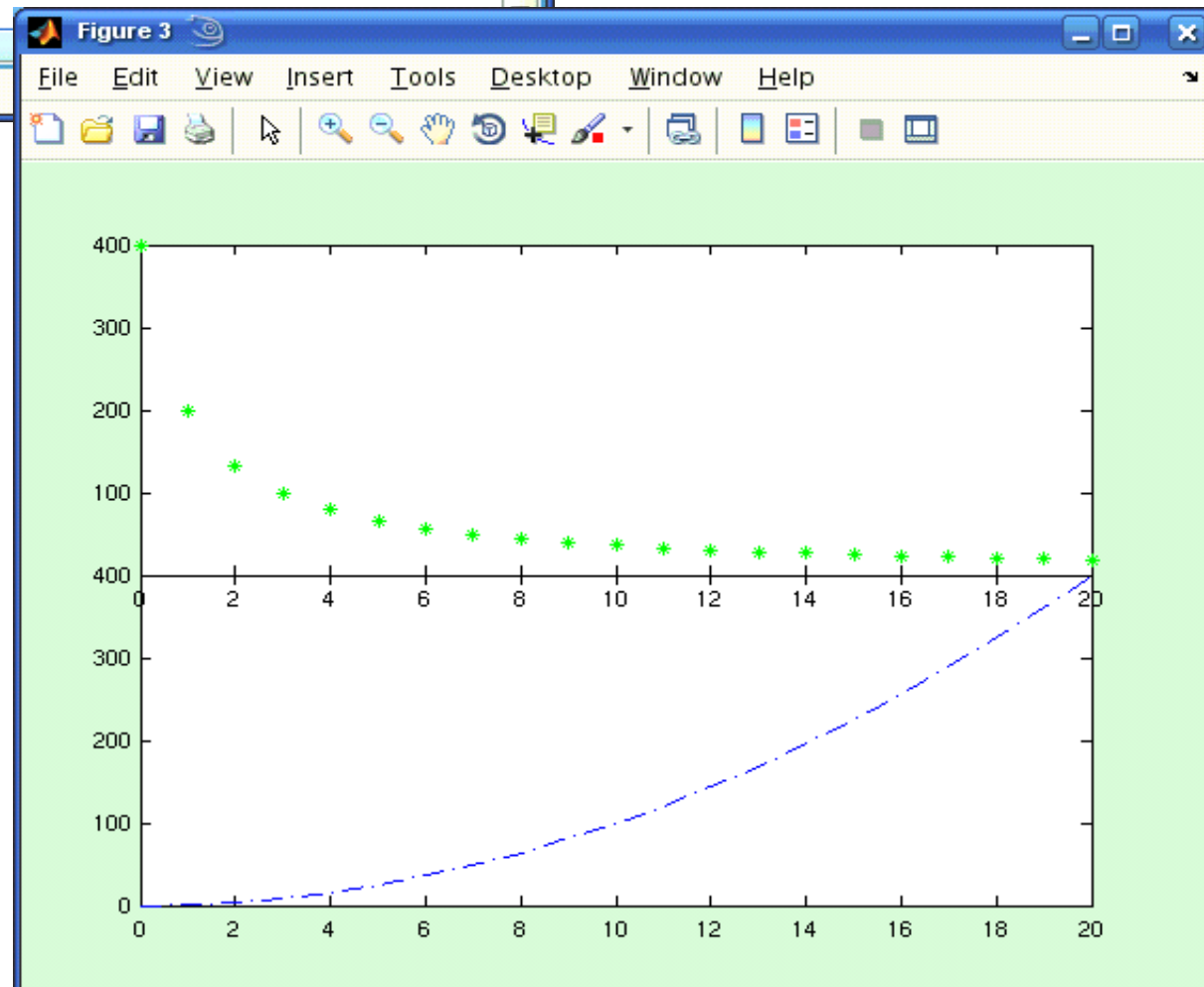
close all
figure

subplot(M, N, plotnum)

- an M x N array of plot axes

Multiple plots on a figure 3: axes('position', [...])

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>> figure
>> axes('position', [0.1 0.1 0.8 0.4])
>> plot(x,y,'b-.')
>> axes('position', [0.1 0.5 0.8 0.4])
>> plot(x,y2,'g*')
fx
>>
```



**axes('position', [xorigin
yorigin xwidth yheight]);**
– for finer control than
subplot

set(gca, 'XTickLabel', {})
- remove x tick labels

Multiple plots on a figure 4: long form of plot command

plot(x1, y1, x2, y2, ..., xn, yn)

% a way of plotting multiple graphs
without using **hold on**

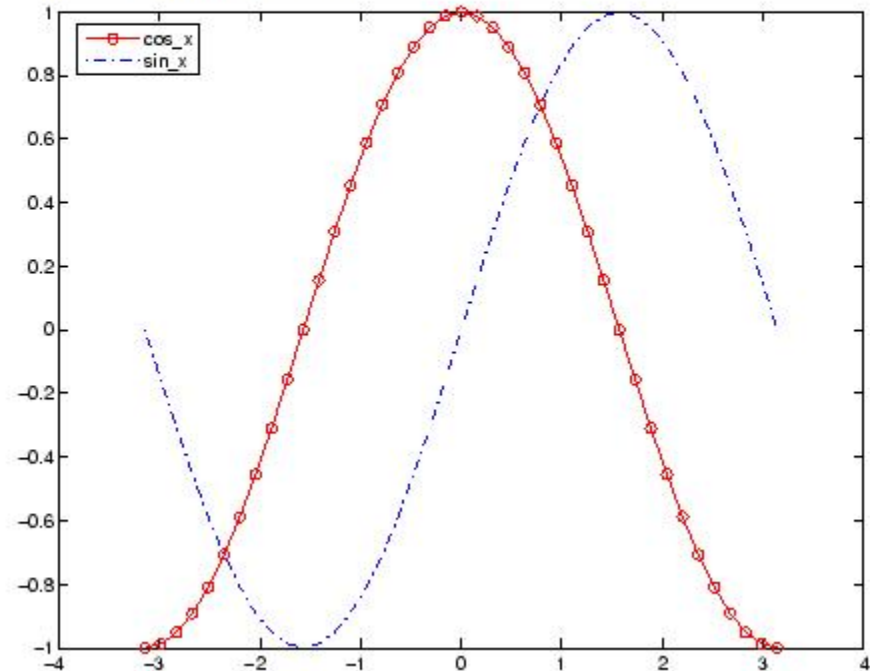
plot(x1, y1, s1, x2, y2, s2, ..., xn, yn, sn)

% as above, but override the default lin
styles.

You can then use **legend** to create a key
for the different graphs in your figure.

Add a legend to a graph showing a sine and cosine function:

```
x = -pi:pi/20:pi;  
plot(x, cos(x), '-ro', x, sin(x), '-.b')  
h = legend('cos_x', 'sin_x', 2);  
set(h, 'Interpreter', 'none')
```



Outline

- 1 File access
- 2 Plotting Data
- 3 Annotating Plots
- 4 Many Data - one Figure
- 5 Saving your Figure**
- 6 Misc
- 7 Examples

Writing an image file - print

`print -f1 -dpng myplotfilename.png`

- script form

`print('-f1', '-dpng', '-r200', 'myplotfilename.png')`

- functional form

-r200 means print with resolution 200 dots per inch (use lower number for small plot)

-f2 means print figure 2

Devices include:

ps, psc, ps2, psc2

- Postscript (c = colour, 2 = level 2)

eps, epsc, eps2, eps2

- Encapsulated Postscript (c = colour, 2 =

level 2)

ill

- Adobe Illustrator format

jpeg90

- JPEG with quality 90 (can be 01 to 99)

tiff

- TIFF

png

- PNG

Can also capture a figure window with:

`>> print -dmeta`

on a Windows system, and paste it into your document. It does the same thing as ALT-PRT SC.

Writing an image file - print

Example:

You have (numberOfPlots) figures and you want to save all of them as level-2 color encapsulated postscript files with names like myplot1.eps, myplot2.eps:

```
for plotNum = 1 : numberOfPlots  
    print('-depsc2', sprintf('-f%d',plotNum), '-r70',  
sprintf('myplot%d.eps',plotNum) );  
end
```

For plotNum = 2, the print line would evaluate to:

```
print('-depsc2', '-f2', '-r70', 'myplot2.eps')
```

Saving your Figure (2): `saveas`

`saveas`

- `saveas(h, 'filename.ext');`
`saveas(h, 'filename', 'format');`
- saves figure with the handle `h` to file `filename`.
- file format is either handled by the extension `ext` or the specified `format`:

<code>ai</code>	Adobe Illustrator	<code>bmp</code>	Windows bitmap
<code>emf</code>	Enhanced metafile	<code>eps</code>	EPS Level 1
<code>fig</code>	Matlab figure	<code>jpg</code>	JPEG
<code>m</code>	Matlab M-file	<code>pbm</code>	Portable bitmap
<code>pcx</code>	Paintbrush 24-bit	<code>pdf</code>	Portable Document Format
<code>pgm</code>	Portable Graymap	<code>png</code>	Portable Network Graphics
<code>ppm</code>	Portable Pixmap	<code>tif</code>	TIFF image, compressed

Saving your Figure: Font Sizes and Line Weights

There is a difference between screen mode and presentation mode:

screen mode vs. presentation mode:

- Matlab defaults work great on a screen, **too small** for presentations!
- don't be the guy that says "Well sorry, you can't read this ..." there is a cure!
- two functions help: `get`, `set`
- 'property', 'value' pairs help in respective functions (`title`, `(x,y,z)label`,...)

List of Figure/Axis Properties:

try: `plot(0:10,0:10)`, `get(gcf)`, `get(gca)`

check:

http://www.mathworks.com/help/techdoc/ref/axes_props.html

Saving your Figure: Font Sizes and Line Weights

Change Figure/Axis Properties:

```
clear all, clc, close all;  
  
%plot some data  
plot(0:10:100, 0:10, 'rx-');  
  
%change axis defaults  
set(gca, 'FontSize', 12, 'FontWeight', 'bold', 'FontAngle', 'oblique', ...  
      'XAxisLocation', 'top')  
  
%now for the labeling  
title('Depth plot', 'FontSize', 20, 'FontAngle', 'normal');  
xlabel('Distance (km)');  
ylabel('Depth (km)');  
  
%now for non-default ticks  
set(gca, 'XTick', 0:10:100, ...  
      'YMinorTick', 'on', ...  
      'YDir', 'reverse')           %reverse y-direction to allow for  
                                   %positive depth values
```