MATLAB Functions and Graphics

We continue our brief overview of MATLAB by looking at some other areas:

- MATLAB Functions: built-in and user defined
- Using MATLAB M-files to store and execute MATLAB statements and functions
- A brief overview of MATLAB Graphics

MATLAB functions

MATLAB makes extensive use of functions
(We have seen many in action already)

- MATLAB provide an extensive set of functions for almost every kind of task.
- Extensible through toolboxes — essentially a collection of functions.
- Functions can operate on Scalars, Matrices, Structures, sometime in subtly different ways.
- Soon we will learn how to create our own functions.

◀◀
▶▶
◀
▶
Back
Close

MATLAB Scalar Functions

Certain MATLAB functions operate essentially on scalars:

- These will **operate element-by-element** when applied to a **matrix**.

Some common scalar functions are:

`sin`	`asin`	`exp`		`abs`	`round`
`cos`	`acos`	`log`	(natural log)	`sqrt`	`floor`
`tan`	`atan`	`rem`	(remainder)	`sign`	`ceil`

◀◀
▶▶
◀
▶
Back
Close

MATLAB Vector functions

Some MATLAB functions operate essentially on a **vector** (row or column):

- These will act on an m-by-n matrix ($m \geq 2$) in a **column-by-column** fashion to produce a **row vector** containing the results of their application to each column.
- **Row-by-row** operation can be obtained by using the transpose, `’`; for example, `mean(A’)’`.
- You may also specify the dimension of operation using the second optional parameter. So `mean(A, 1)` works column-by-column and `mean(A, 2)` works row-by row.

Some common vector functions are

```
max   sum   median  any
min   prod  mean    all
sort  std
```

MATLAB Vector Function Examples

```
>> A = rand(4,4)
A =
   0.8600   0.8998   0.6602   0.5341
   0.8537   0.8216   0.3420   0.7271
   0.5936   0.6449   0.2897   0.3093
   0.4966   0.8180   0.3412   0.8385

>> max(A)
ans =
   0.8600   0.8998   0.6602   0.8385

>> max(max(A))
ans =
   0.8998
```

```
>> mean(A)
ans =
   0.7009 0.7961 0.4083 0.6022

>> mean(A')
ans =
   0.7385 0.6861 0.4594 0.6236

>> mean(A')'
ans =
   0.7385
   0.6861
   0.4594
   0.6236

>> mean(A, 2)
ans =
   0.7385
   0.6861
   0.4594
   0.6236
```

Back
Close

Matrix functions

Much of MATLAB's power comes from its matrix functions, many are concerned with specific aspects of linear algebra and the like (which does not really concern us in this module)

Some common ones include:

`inv`	inverse	`det`	determinant
`size`	size	`rank`	rank

MATLAB functions may have single or multiple output arguments.

For example, `rank()` always returns a scalar:

```
>> A
A =
    0.8600    0.8998    0.6602    0.5341
    0.8537    0.8216    0.3420    0.7271
    0.5936    0.6449    0.2897    0.3093
    0.4966    0.8180    0.3412    0.8385
>> rank(A)
ans =
     4
```

Back
Close

Return Multiple Output Arguments

`size`, for example, **always** returns 2 values even for a vector:

```
>> X = [1 2 3 4]; size(X)
ans =
     1     4
```

Therefore it's best to usually return multiple arguments to scalar variables:

```
>> [n] = size(A)
n =
     5     5
>> [n m] = size(A)
n =
     5
m =
     5
```

Note: In the first call above `n` is returned as vector.

Back
Close

Writing Your Own Functions

The basic format for declaring a function in MATLAB is:

```
function a = myfunction(arg1, arg2, ....)
% Function comments used by MATLAB help
%
matlab statements;
a = return value;
```

The function **should** be stored as `myfunction.m` somewhere in your MATLAB file space.

- This is a common use of a MATLAB, **M-file**.
- To call this function simply do something like:

  ```
  b = myfunction(c, d);
  ```

- May need to set MATLAB search path to locate the file (**More soon**).

Back
Close

MATLAB Function Format Explained

```
function a = myfunction(arg1, arg2, ....)
% Function comments used by MATLAB help
%
matlab statements;
a = return value;
```

- A function may have 1 or more input argument. Arguments maybe matrices or structures.
- The return value, in this case, `a`, may return multiple output (see example soon)
- Contiguous **comments** immediately after (**no blank line**) are used by MATLAB to output help when you type: `help myfunction` — **This is very neat and elegant**

Simple MATLAB Example: Single Output Argument

mymean.m (**Note**: A better built-in `mean()` function exists):

```
function  mean = mymean(x)
% MyMean  Mean
% For a vector x, mymean(x) returns the mean of x;
%
% For a matrix x, mymean(x) acts columnwise.
 [m  n] = size(x);
 if m == 1
   m = n;    % handle case of a row vector
 end
 mean = sum(x)/m;

>> help mymean
MyMean  Mean
For a vector x, mymean(x) returns the mean of x;

For a matrix x, mymean(x) acts columnwise.

>> A = [1 2 3;4 5 6; 7 8 9]; mymean(A)
ans =  4  5  6
```

Simple MATLAB Example: Multiple Output Argument

```
function  [mean, stdev] = stat(x)
% STAT  Mean and standard deviation
% For a vector x, stat(x) returns the mean of x;
% [mean, stdev] = stat(x) both the mean and standard deviation.
% For a matrix x, stat(x) acts columnwise.
 [m  n] = size(x);
 if m == 1
   m = n;     % handle case of a row vector
 end
 mean = sum(x)/m;
 stdev = sqrt(sum(x.^ 2)/m - mean.^ 2);
```

```
 >> help stat
  STAT  Mean and standard deviation
  For a vector x, stat(x) returns the mean of x;
  [mean, stdev] = stat(x) both the mean and standard deviation.
  For a matrix x, stat(x) acts columnwise.
>> [mean sdev] = stat(A)
mean =
     4     5     6
sdev =
    2.4495    2.4495    2.4495
```

Back
Close

Useful Function MATLAB commands

`type` : list the function from the command line. *E.g.* `type stat`

`edit` : edit or create M-file. *E.g.* `edit stat`

`nargin` : The special variable that holds the current number of function input arguments — useful for checking a function has been called correctly. (`nargout` similar)

`disp` : Text strings can be displayed by the `disp()` function: *E.g.* `disp('Warning: You cant type')`.

`error` : More useful in functions is the `error()` function which displays text and also **aborts** the M-file. *E.g.* `error('Error: You''re an idiot')`

`global` : Variables are **local** to functions. Arguments may be passed in but also `global` variables may be used — See `help global` for more details.

Debugging : Many debugging tools are available — See `help dbtype`

MATLAB Script M-files

Not all MATLAB M-files need to contain function declarations:

- A **script** file simply consists of a sequence of normal MATLAB statements.
- It is called in much the same way: If the file has the filename, say, `myscript.m`, then the MATLAB command:

  ```
  >> myscript
  ```

  will cause the statements in the file to be executed.
- Variables in a script file are global and will change the value of variables of the same name in the environment of the current MATLAB session.
- An M-file can reference other M-files, including referencing itself recursively.
- Useful to create batch processing type files, inputting data, or just saving last session. Put scripts that run every time MATLAB starts in `startup.m`.

Creating function and script M-files

There are a few ways you can create an M-file in MATLAB:

- Use MATLAB's or you computer system's text editor:
  - Type commands directly into text editor.
  - Copy and Paste commands you have entered from MATLAB's **Command** or **History** windows to your text editor.
- Use `diary` command:

  `diary FILENAME` causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file 'diary' is used.

  `diary off` : suspends it.

  `diary on` : turns it back on.

  `diary` : initially creates a file 'diary', afterwards toggles diary state.

Back
Close

MATLAB Paths

- M-files **must** be in a directory accessible to MATLAB.
- M-files in the **present** in current working directory are **always** accessible.
- The current list of directories in MATLAB's search path is obtained by the command `path`.
- This command can also be used to add or delete directories from the search path — See `help path`.
- If you use all lot of libraries all the time then the `startup.m` in your MATLAB top level directory (`/Users/username/matlab`) can be edited to set such paths.
- You can use the Main Menu: File⟶Set Path to set paths also.

◀◀
▶▶
◀
▶
Back
Close

Matlab Graphics

We have already seen some simple examples of how we do simple plots of audio and images.

Lets formalise things and dig a little deeper. MATLAB can produce:

- 2D plots — `plot`
- 3D plots — `plot3`
- 3D mesh surface plots — `mesh`
- 3D faceted surface plots — `surf`.

We are not so concerned with **3D Plots** in this course so we wont deal with these topics here except for one simple example.

See MATLAB `help graphics` and plenty of MATLAB demos (type `demo` or using IDE) for further information.

2D Plots

The main function we use here is the `plot` command:

- `plot` creates linear x-y plots;
- If x and y are vectors of the same length, the command:

  ```
  plot(x,y)
  ```

  - opens a MATLAB figure (graphics window)
  - draws an x-y plot of the elements of x versus the elements of y.
- Example: To draw the graph of the sin function over the interval 0 to 8 with the following commands:

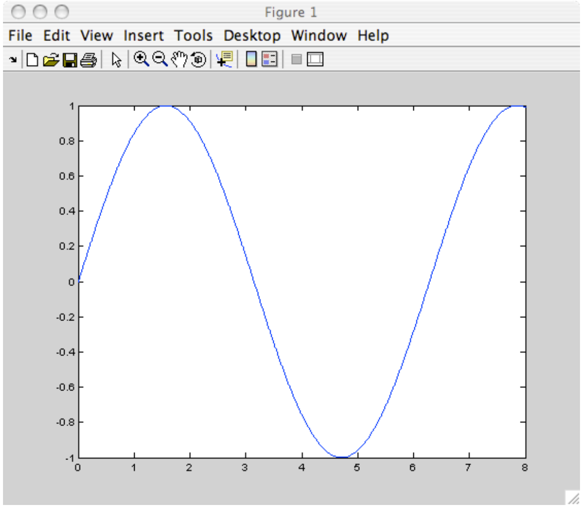
  ```
  x = 0:.01:8; y = sin(x); plot(x,y)
  ```

  - The vector x is a partition of the domain with step size 0.01 while y is a vector giving the values of sine at the nodes of this partition

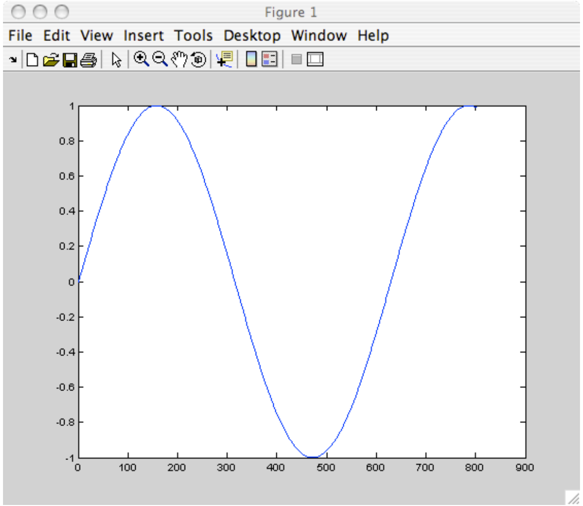
2D Plots (cont.)

- It is generally more useful to plot elements of x versus the elements of y using `plot(x,y)`
- `plot(y)` plots the columns of Y versus their index.

  Note the difference in the x axes in the two figures below:



Result of `plot(x,y)`



Result of `plot(y)`

Back
Close

Controlling MATLAB Figures

So far we have let `plot` (or `imshow`) automatically create a MATLAB figure.

- In fact it will only create a figure if one **does not exist**.
- If a figure **exists** it will draw to the current figure
  - Possible overwriting currently displayed data
  - This may not be ideal?

MATLAB affords **much greater control** over figures.

The MATLAB `figure` command

To create a new figure in MATLAB simply enter the MATLAB command:

`figure` or `figure(n)`

where `n` is an index to the figure, we can use later.

- If you just enter `figure` then figure indices are numbered consecutively automatically by MATLAB
- Example:
  - If figure 1 is the current figure, then the command `figure(2)` (or simply `figure`) will open a second figure (if it does not exist) and make it the *current figure*.
  - The command `figure(1)` will then expose figure 1 and make it once more the *current figure*.

The MATLAB `figure` command (cont.)

- Several figures can exist, **only one** of which will at any time be the designated *current figure* where graphs from subsequent plotting commands will be placed.
- The command `gcf` will return the number of the current figure.

MATLAB figure control

The figures/graphs can be given titles, axes labeled, and text placed within the graph with the following commands which take a string as an argument.

`title`	graph title
`xlabel`	x-axis label
`ylabel`	y-axis label
`gtext`	place text on the graph using the mouse
`text`	position text at specified coordinates

For example, the command:

```
title(’Plot of Sin 0-8’)
```

gives a graph a title.

Figure Axis Scaling

- By default, the axes are **auto-scaled**.
- This can be **overridden** by the command `axis`.
- Some features of `axis` are:

axis([x_{min},x_{max},y_{min},y_{max}])	set axis scaling to given limits
`axis manual`	freezes scaling for subsequent graphs
`axis auto`	returns to auto-scaling
`v = axis`	returns vector v showing current scaling
`axis square`	same scale on both axes
`axis equal`	same scale and tic marks on both axes
`axis image`	same as `axis equal` but tight bounded
`axis off`	turns off axis scaling and tic marks
`axis on`	turns on axis scaling and tic marks

Note: The `axis` command should be type *after* the `plot` command.

◀◀
▶▶
◀
▶
Back
Close

Multiple Plots in the Same Figure

There are a few ways to make multiple plots on a single graph:

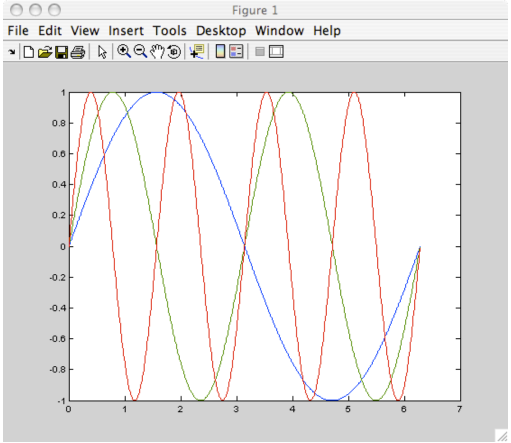
- Multiple plot arguments within the `plot` command:

```
x=0:.01:2*pi;
y1=sin(x);y2=sin(2*x); y3=sin(4*x);
plot(x,y1,x,y2,x,y3)}
```

- By forming a matrix `Y` containing the functional values as columns and calling the `plot` command:

```
x=0:.01:2*pi;
Y=[sin(x)', sin(2*x)',
sin(4*x)'];
plot(x,Y)
```

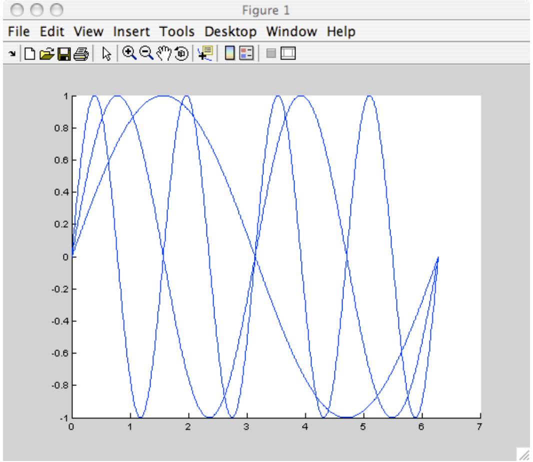
- Using the `hold` command (**next slide**)



The hold Command

- Use the hold command:
  - The command hold on freezes the current figure
  - Subsequent plots are superimposed on it. **Note**:The axes may become rescaled.
  - Entering hold off releases the hold.
  - Example:

```
figure(1);
hold on;
x=0:.01:2*pi;
y1=sin(x);
plot(x,y1);
y2=sin(2*x);
plot(x,y2);
y3=sin(4*x);
plot(x,y3);
hold off;
```

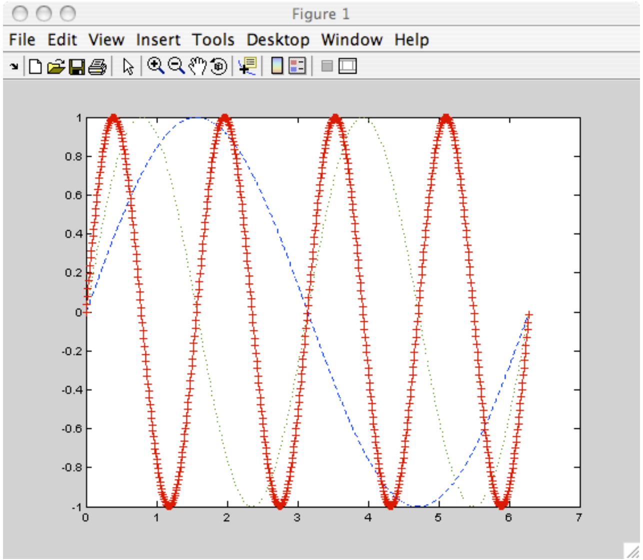


Line, Point and Colour Plot Styles

- One can override the default linetypes, pointtypes and colors.
- The `plot` function has additional arguments:

Example:

```
x=0:.01:2*pi;
y1=sin(x); y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,’--’,x,y2,’:’,x,...
y3,’+’)
```



- renders a dashed line and dotted line for the first two graphs
- the third the symbol + is placed at each node.

Back
Close

Line and Mark Types

- The line and mark types are

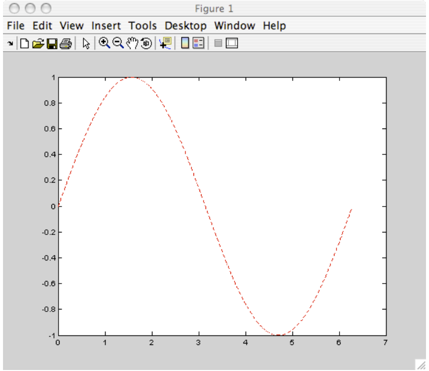
Linetypes	solid (−), dashed (−−). dotted (:), dashdot (− .)
Marktypes	point (.), plus (+), star (⋆), circle (o), x-mark (x)
Colors	yellow (y), magenta (m), cyan (c), red (r) green (g), blue (b), white (w), black (k)

- Colors can be specified for the line and mark types.

- Example:

```
plot(x,y1,’r−−’)
```

  plots a red dashed line:

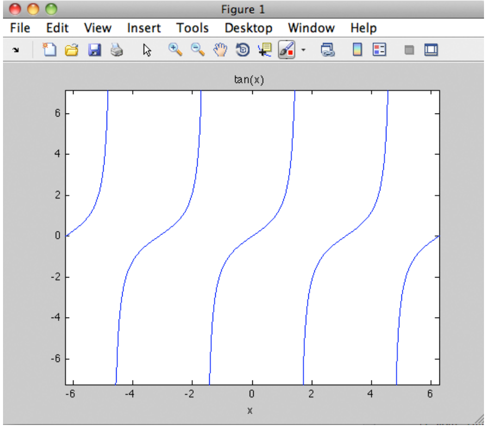


Easy Plot Commands

- Sometimes it may be difficult / time-consuming to set up the discrete sampling to `plot` a function. MATLAB provides an `ezplot` command for "easy plotting".
- To plot a function over the default -2π to 2π domain, simply use `ezplot` `'function'`. Example:

```
ezplot('tan(x)')
```

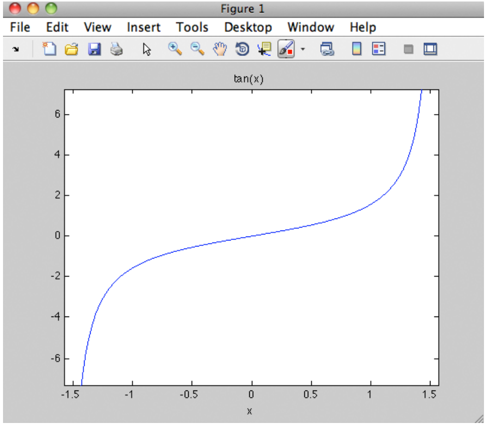
renders the tangent function. Sampling rate, axis, and labels will be appropriately produced.



Easy Plot Commands (cont.)

- To plot a function over a specific domain, use `ezplot ‘function’ [x_min x_max]`.
  Example:

```
ezplot('tan(x)', [-pi/2, pi/2])
```



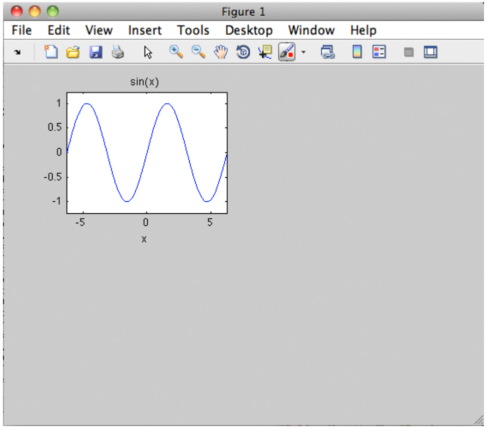
- More easy-to-use plotting commands, see help for details: `ezcontour`, `ezpolar` etc.

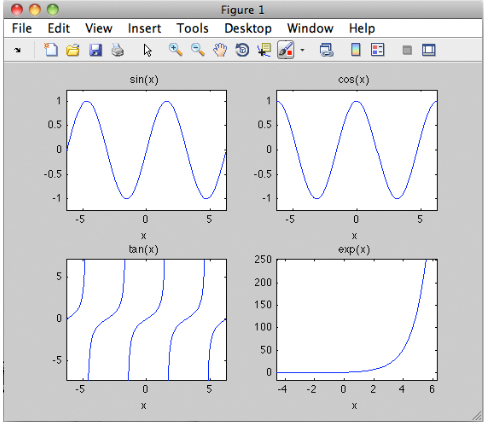
Multiple Plots in a Figure: `subplot`

- To put multiple subplots in a single figure, first use `subplot (m, n, p)` to break the figure window into a m-by-n matrix of subspace, and switch the current plotting target to the p^{th} subfigure (counted from top-left corner and in a row priority order).
- Use a few `subplot` with the same `m` and `n`, and different `p` to target different subspace, followed by actual plotting commands.
- Example:

```
>> subplot(2, 2, 1); ezplot('sin(x)');
>> subplot(2, 2, 2); ezplot('cos(x)');
>> subplot(2, 2, 3); ezplot('tan(x)');
>> subplot(2, 2, 4); ezplot('exp(x)');
```

Multiple Plots in a Figure: `subplot` (cont.)





- To return to the default whole figure configuration, use `subplot(1, 1, 1)`

Other Related Plot Commands

- Other specialized 2-D plotting functions you may wish to explore via `help` are:

  `polar, bar, hist, quiver, compass, feather, rose, stairs, fill`

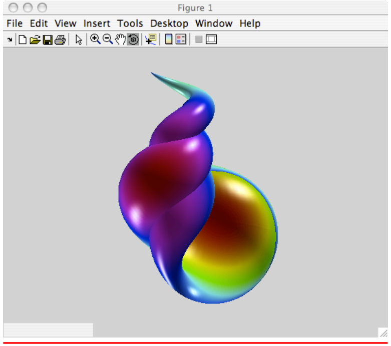
Saving/Exporting Graphics

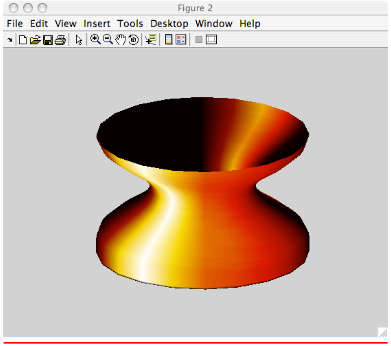
- Use File→Save As.. menu bar option from any figure window you wish to save
- From the MATLAB command line use the command `print`.
  - Entered by itself, it will send a high-resolution copy of the current graphics figure to the default printer.
  - The command `print` *filename* saves the current graphics figure to the designated filename in the default file format.

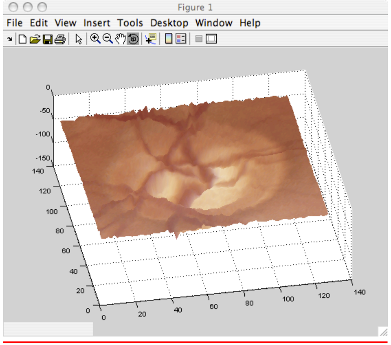
    If *filename* has no extension, then an appropriate extension such as `.ps`, `.eps`, or `.jet` is appended.
- See `help print` for more details

3D Plot Example

We have seen some 3D examples in earlier examples and the code is available for study:

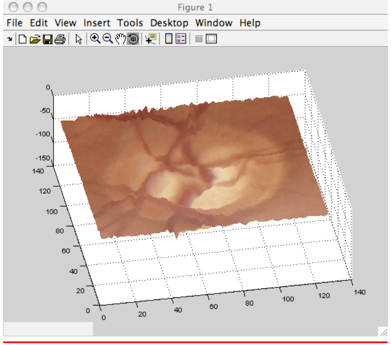






Back
Close

3D Plot Example Explained



```
% Read Heightmap
d = imread('ddd.gif');
% Read Image
[r,map] = imread('rrr.gif');
% Set Colourmap
colormap(map);
r = double(r);
d = double(d);
d = -d;
% Set figure and draw surface
figure(1)
surf(d,r)
shading interp;
```

- Two Images store 3D Information: Height map `ddd.gif`, Image: `rrr.gif`
- **Note:** use `imread` to extract image values and colour map
- Set `colormap` for display
- Use `surf(d,r)` to plot a 3D surface, `d`, with colour values, `r` — see `help surf`
- Set `shading` style.
- `mesh` similar — see `help mesh`
- `plot3(x,y,z)` similar to `plot(x,y)` — see `help plot3`