CS395T: Introduction to Scientific and Technical Computing

Post-processing with MATLAB

Instructors

Dr. Victor Eijkhout, Research Scientist, TACC
Dr. Karl W. Schulz, Research Associate, TACC

Scientific Simulation

- Running simulation codes is only part of the battle
- Pre- and post-processing may account for a large fraction of the total time to solution
  - model creation and mesh generation
  - data integration
  - visualization of results
  - derived quantities

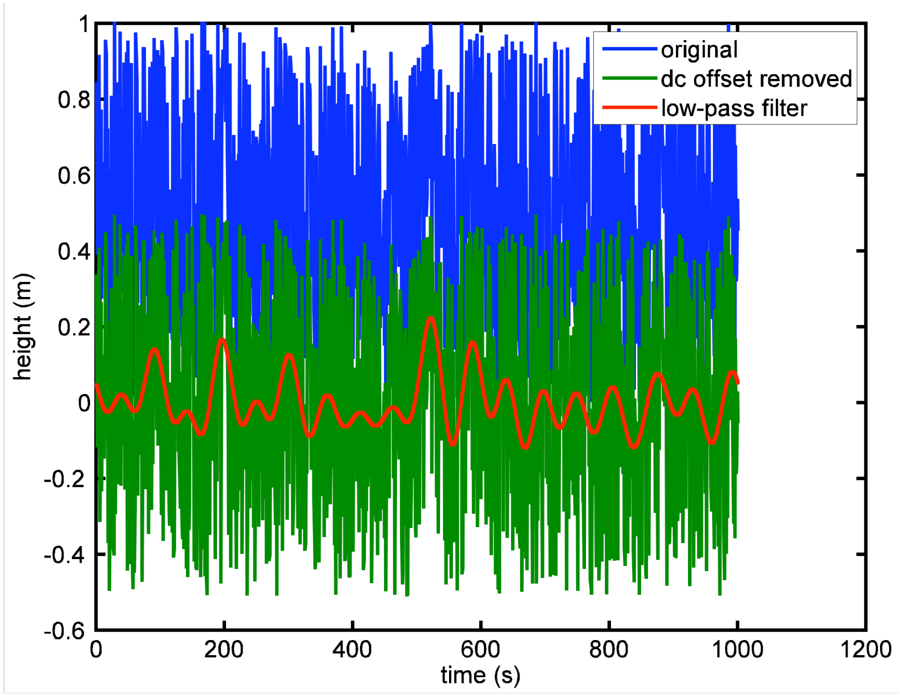
Post-processing

- Simulation results have to be interpreted
  - are they “right”?
    - are the errors acceptable?
    - does the model match the physics?
  - meaning needs to be extracted
- Plot pictures of the results
- Derive quantities of interest
  - average temperature
  - maximum temperature

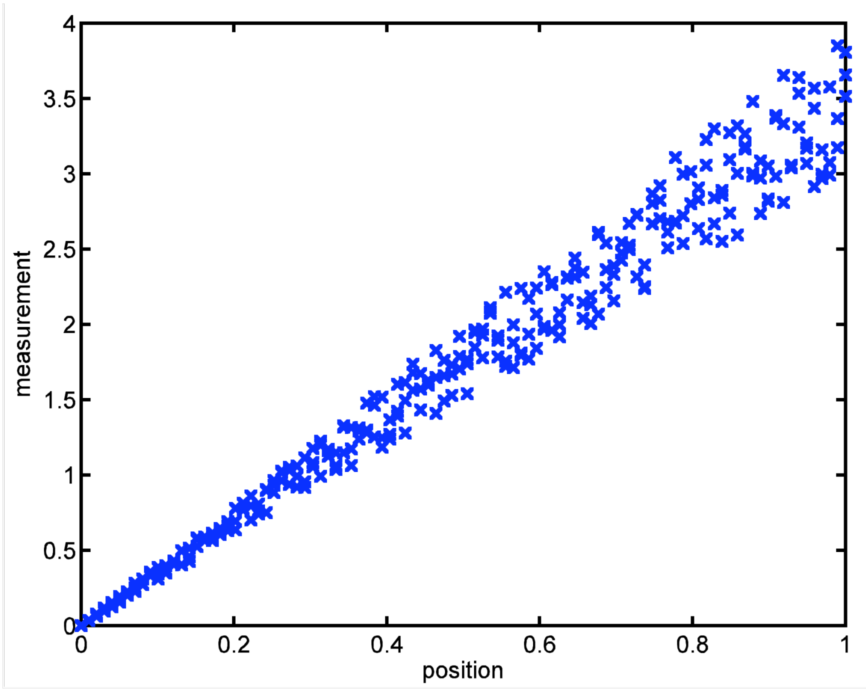
Plotting Results

- 1-D
  - function graphing
  - scatter plots
- 2-D
  - contour/isoline plots
  - surface plots
  - pseudo color plots
  - vector arrows
- 3-D
  - isosurface plots
  - slices with 2-D plots
  - volume rendering

1-D

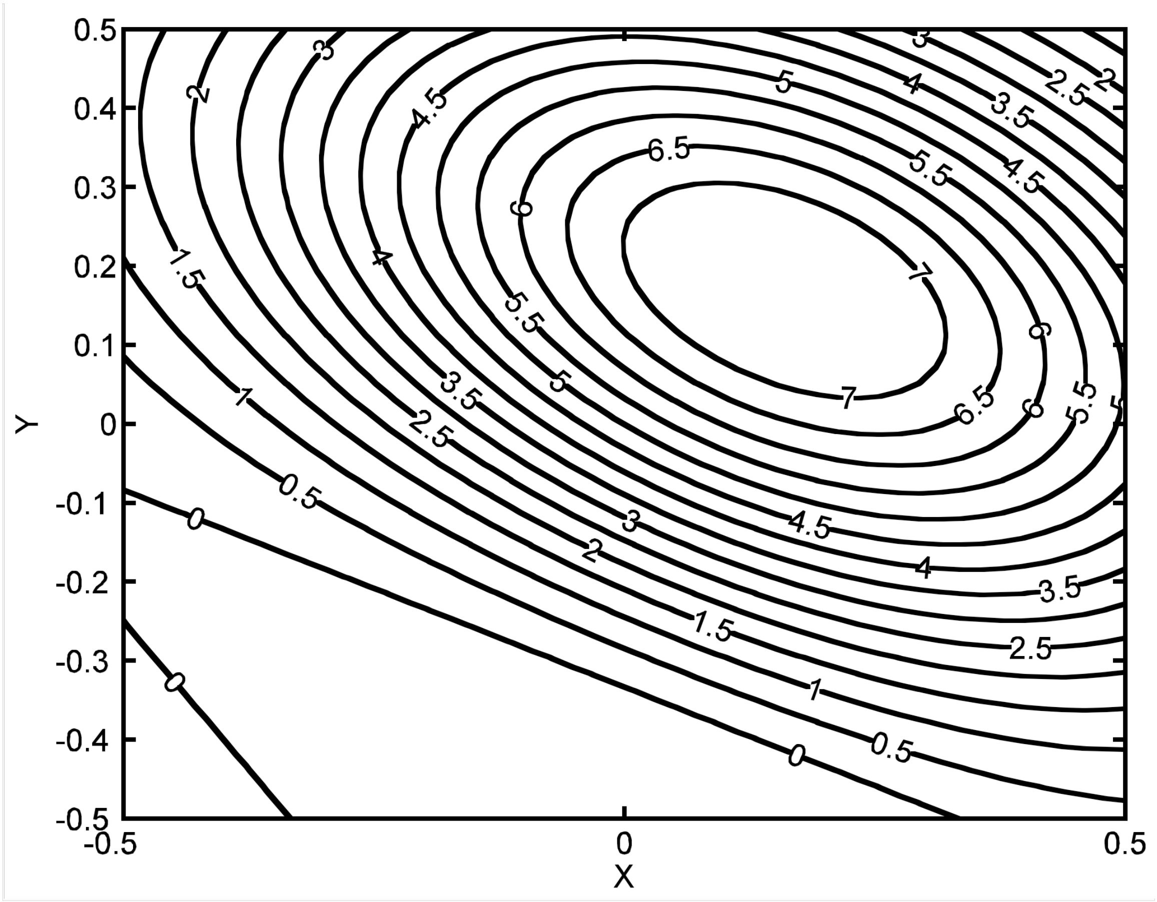


Line Plot



Scatter Plot

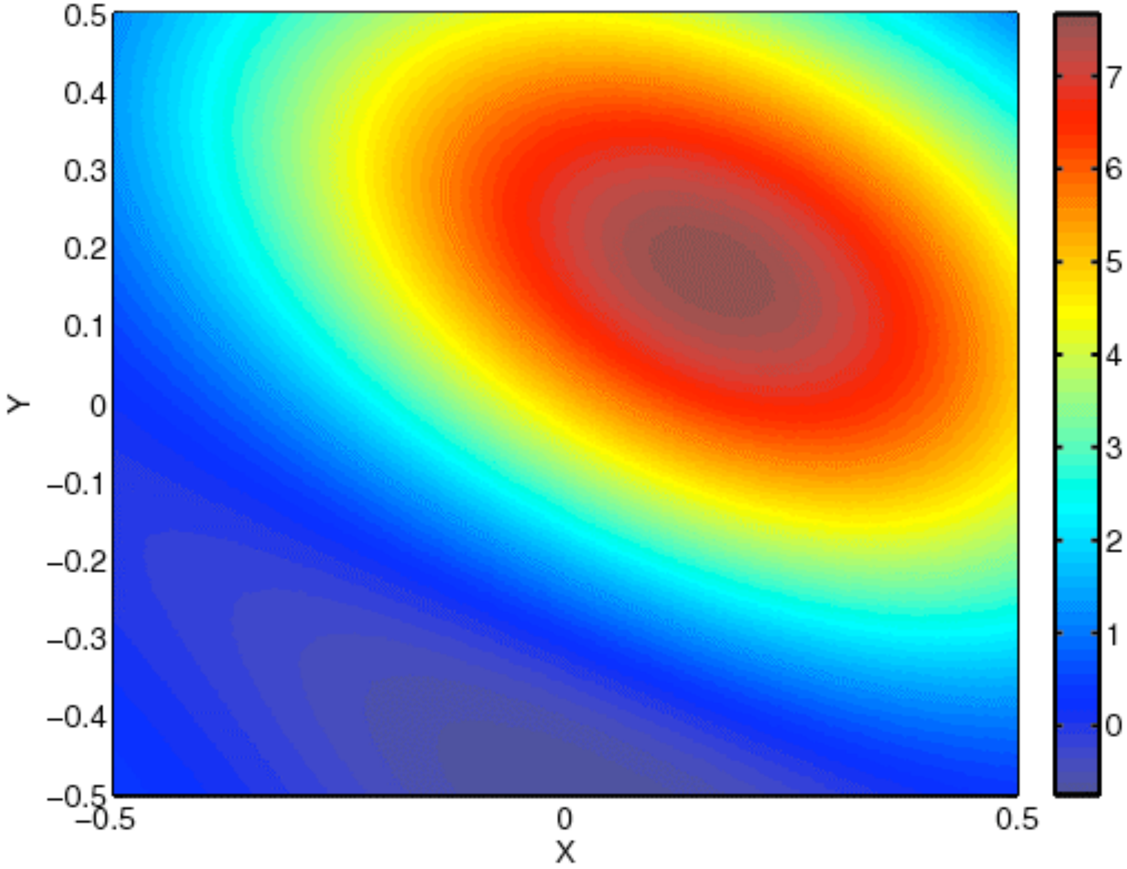
2-D—Contour Plots



$$c = f(x, y)$$

- Draw a curve for each contour level

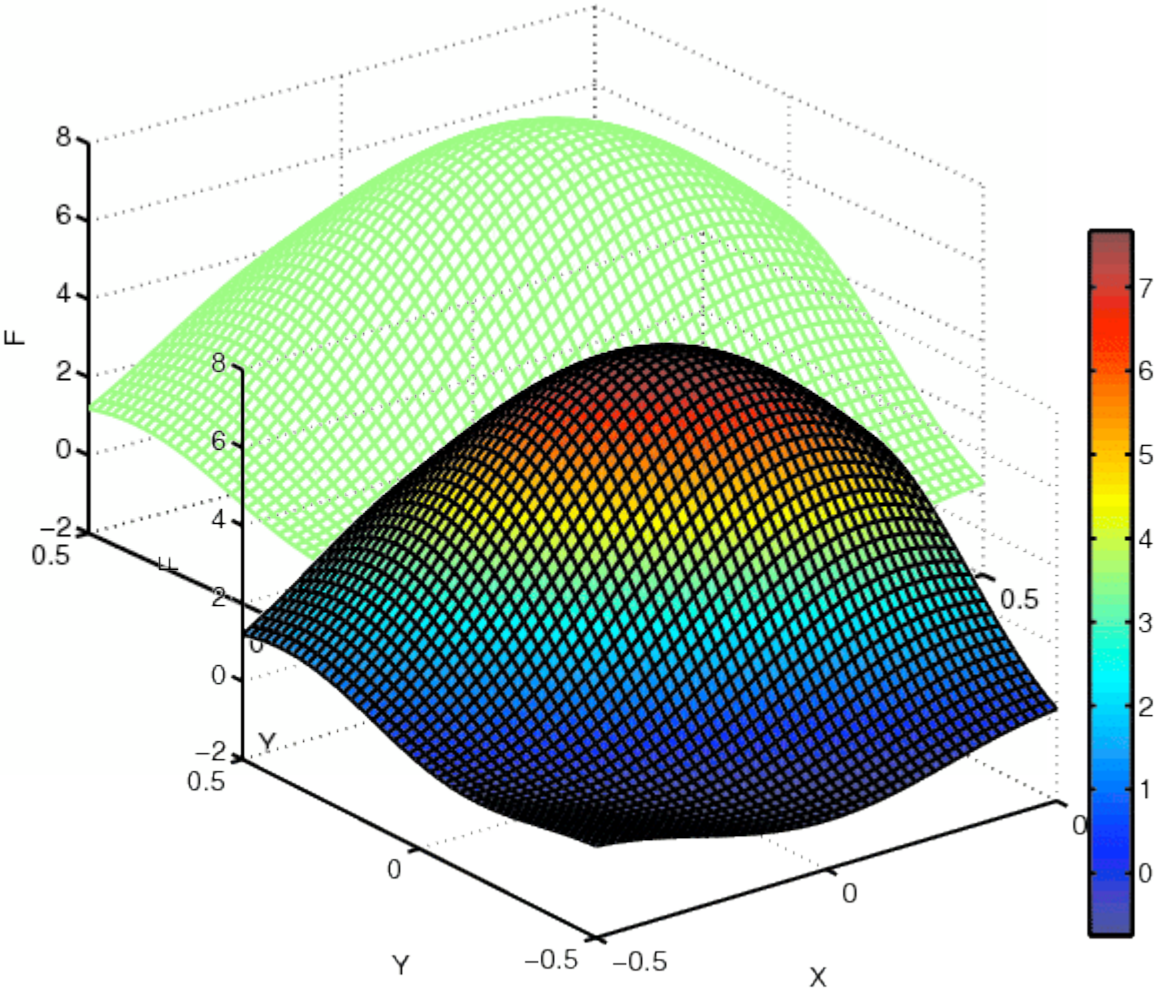
2-D—Pseudocolor Plots



$$c = f(x, y)$$

- Map function values to colors
- Plot a blob of color at each data point

2-D—Surface/Function Plot



$$z = f(x, y)$$

- Function value is height
- May use pseudocolor on the plotted surface
- Needs 3-D projection

MATLAB

- MATrix LABoratory http://www.mathworks.com/
- Good for rapid prototyping of scientific problems
- Can be useful for post-processing data
- Interactive
- Built-in plotting functions
- Free & open source: Octave http://www.octave.org/

Language Structures

- Everything is a matrix/array (basically) of two possible types
  - doubles (real and complex)
  - characters
- Indices start at 1
  - colon notation for sub-indexing
  - similar to the Fortran90 notation with some extensions
- No need to declare variables
  - just assign to them to bring them into existence
  - names are case-sensitive

Starting Up

```
localhost$ matlab -nodesktop

                                 < M A T L A B >
                    Copyright 1984-2006 The MathWorks, Inc.
                           Version 7.2.0.294 (R2006a)
                                January 27, 2006


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
```

- MATLAB has a GUI environment, too, but I don't like it
  - somewhat heavyweight
  - has many nice features (debugging, etc.)

Defining Variables

- Results of commands are printed back to you by default
  - use a semicolon at the end of the line to suppress the printing
- `who [variables]`
  - describes what variables are defined in the workspace
  - optionally add a space-separated list of variables to restrict the output

```
>> a=1

a =
     1

>> a=1;
>> a

a =
     1

>> who a
  Name      Size                        Bytes  Class

  a         1x1                             8  double array

Grand total is 1 element using 8 bytes
```

Arrays

```
>> a=[1 2 3; 4 5 6; 7 8 9]
a =
     1     2     3
     4     5     6
     7     8     9
>> size(a)
ans =
     3     3
>> a(2,3)
ans =
     6
>> a(2)
ans =
     4
>> a'
ans =
     1     4     7
     2     5     8
     3     6     9
```

- Literal arrays are built up using square brackets, spaces, and semicolons
  - new lines may be used to take the place of semicolons
  - commas may be used instead of spaces
- Array indexing mimics FORTRAN indexing
  - array ordering is column-major like FORTRAN, too
- Apostrophe after a variable gives its transpose
  - gives the conjugate transpose for complex matrices

Accessing Sub-arrays

```
>> a(:,2)
ans =

     2
     5
     8

>> a(2:end,1:end-1)
ans =

     4     5
     7     8
```

- Colon notation
  - Similar to Fortran90
- A bare colon means everything in that array dimension
- `x:y` means `x, x+1, x+2, ..., y`
- `x:s:y` means `x, x+s, x+2s, ..., y`
- `end` means the highest index for that array dimension
- indices may be expressions

Character Strings

```
>> a='asdfasdf'

a =

asdfasdf

>> b=[a 'foo']

b =

asdfasdffoo
```

- Character strings are just row-vectors of individual characters
- Literals given between single quotes
  - be careful with transpose/quote confusion
- Concatenation done with array notation
- `num2str`, `sprintf`, and `sscanf` can be used to convert between numbers and strings

Operators

- MATLAB operators are matrix operators
  - array dimensions must correct
- `+` adds arrays of the same dimensions
- `a*b` multiplies using matrix multiplication rules
  - requires a be m-by-n and `b` be `n*k`
  - unless `m=n=1` or `n=k=1` which gives matrix multiplication by `a` scalar

Operators

- Division is special
- a\b is inv(a)*b; a/b is a*inv(b).
- a/b and b\a are the same as (1/b)*a if b is a scalar
- If b is a vector, and the dimensions are right, A\b computes the solution to $\mathbf{Ax} = \mathbf{b}$ with Gaussian elimination or Least-squares if a isn't square

Element-wise Operators

- `a.*b`, `a./b`, `a.^b` compute element-wise multiplication, division, and exponentiation for a and b that are the same size

Useful Commands

- clear
- whos
- help
- lookfor
- size
- length
- cd
- ls
- unix

M-files

- MATLAB commands written into text files with the extention `.m` can be run as scripts
- If `foo.m` is in the current directory, it can be run by

```
>> foo
```

- MATLAB has a search path for M-files
  - check out the `path` command if you want to change/add to it
- `%` is the comment character

Control Structures

- For Loops

```
for idx=1:n
  do_something(idx)
end
```

- for iterates over the columns of the RHS expression
  - `for col=A`
    `A` a 2-D array assigns the columns of `A` to `col`
  - 1:n is a row-vector

Control Structures

- Conditionals

```
if expr1
  statements
elseif expr2
  statements
else
  statements
end
```

- Statements are executed when the all the real parts of the elements of the expression are non-zero
- Logical operators:

  `==, <, >, <=, >=, ~=`

Control Structures

- While Loops

```
while expr
   statements
end
```

- Same expression rules as `if`
- `break` can be used to exit early

Common Plotting Functions

- `plot`—line plots
- `contour`—contour plots
- `pcolor`—pseudocolor plots
- `mesh`—wireframe function plots
- `surf`—pseudocolor function plots
- `figure`—opens/selects a new figure window
- `axis`—controls the range of each axis

`plot`

`plot(x,y)`

- Makes a line plot `y` vs. `x`

`plot(x,y,`*`linespec`*`)`

- Makes a plot of `y` vs. `x` with the line style described by *`linespec`*

`plot(x1,y1,ls1,x2,y2,ls2,...)`

- Plots multiple curves with different line styles on the same graph

Line Styles

- Line styles are (up to) 4 character strings with

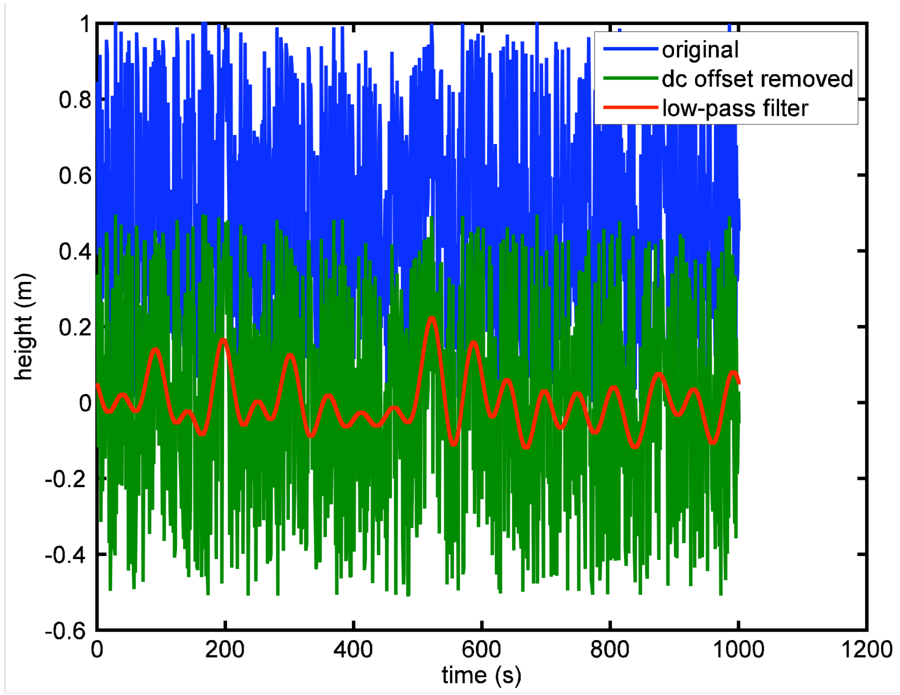
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

```
>> plot(x,y,’bx-.’)
```

Labeling

- `xlabel(xstring)` and `ylabel(ystring)` set the x- and y-axis labels
- `title(tstring)` adds a title
- `legend(string1,string2,...)` creates a legend/key with each string corresponding to a curve plotted with plot

plot Example



```
plot(t,x,t,z,t,q);
xlabel('time (s)');
ylabel('height (m)');
legend('original',...
   'dc offset removed',...
   'low-pass filter');
```

contour

`contour(x,y,f)`

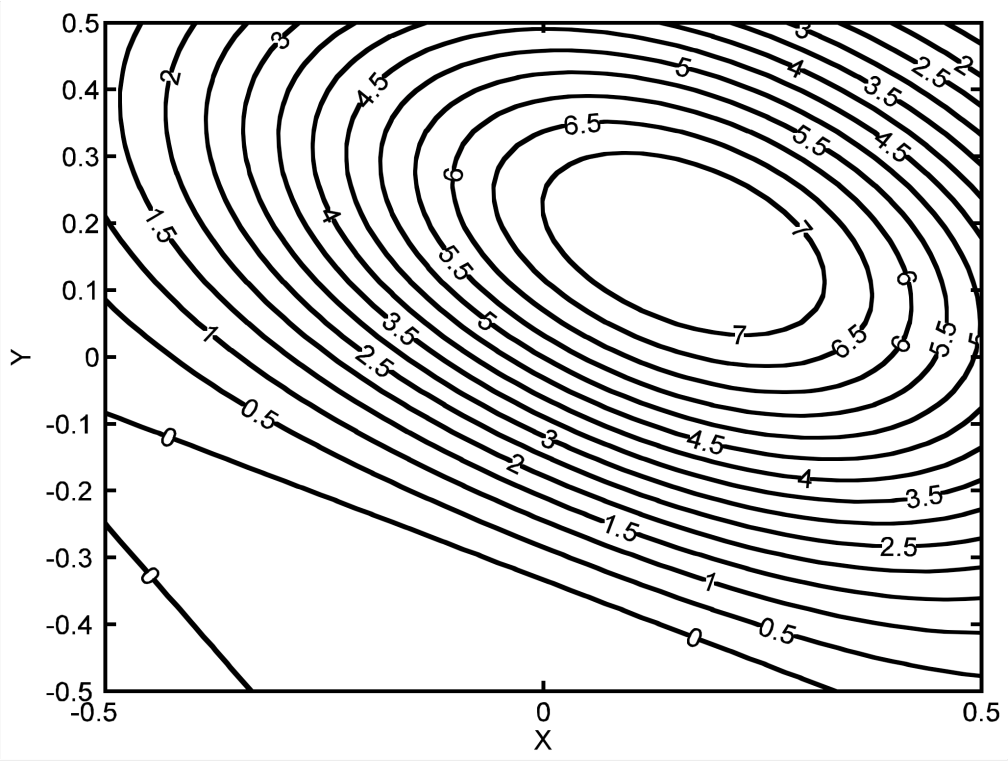
- Plots contours of the 2-d array `f` at the points given in the 2-d arrays `x` and `y`
  - `x` and `y` can be generated from 1-d arrays using

    `[x,y]=meshgrid(x1d,y1d);`

`contour(x,y,f,levels)`

- Plots contours of `f` at levels described by `levels`
  - `levels` a scalar, plots that many evenly-spaced contours
  - `levels` a vector, plots a contour at the level given by each element of `levels`

`contour` Example



```
[c,h]=contour(x,y,f,...
linspace(0,7,15),'k');
clabel(c,h,...
'labelspacing',288);
```

pcolor

`pcolor(x,y,f)`

- Makes a psuedo color plot of `f` over `x` and `y`
- Same argument requirements as `contour`

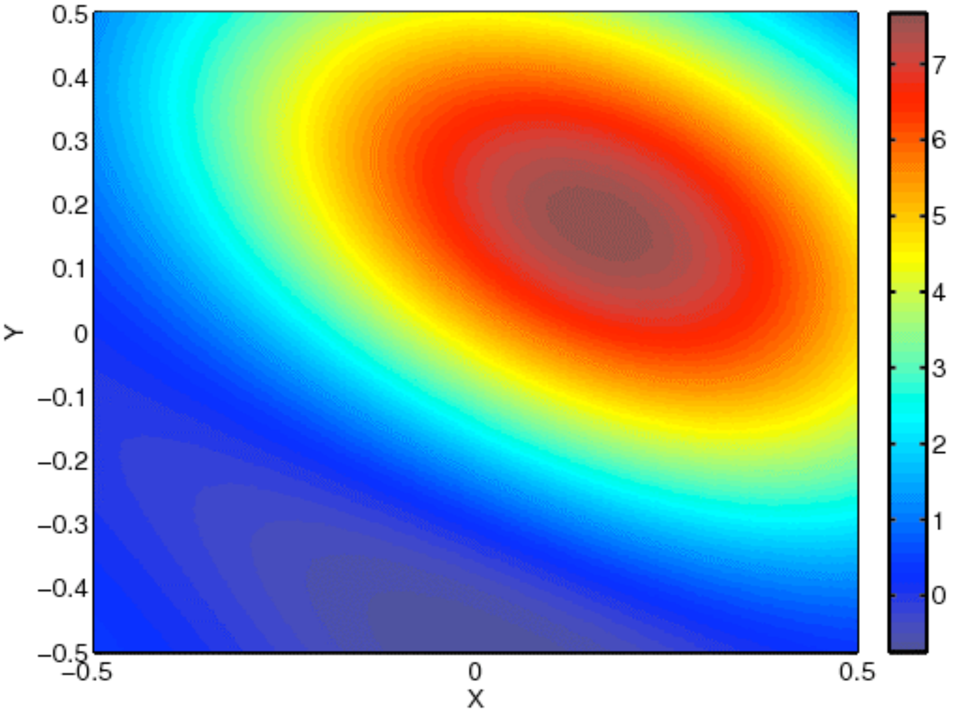
`colorbar`

- Adds a colorbar to the plot showing the mapping between colors and function values

`shading interp`

- Changes the shading mode from `faceted` to interpoloated (makes prettier graphs)

`pcolor` Example



```
pcolor(x,y,f);
shading interp;
colorbar;
```

`mesh` and `surf`

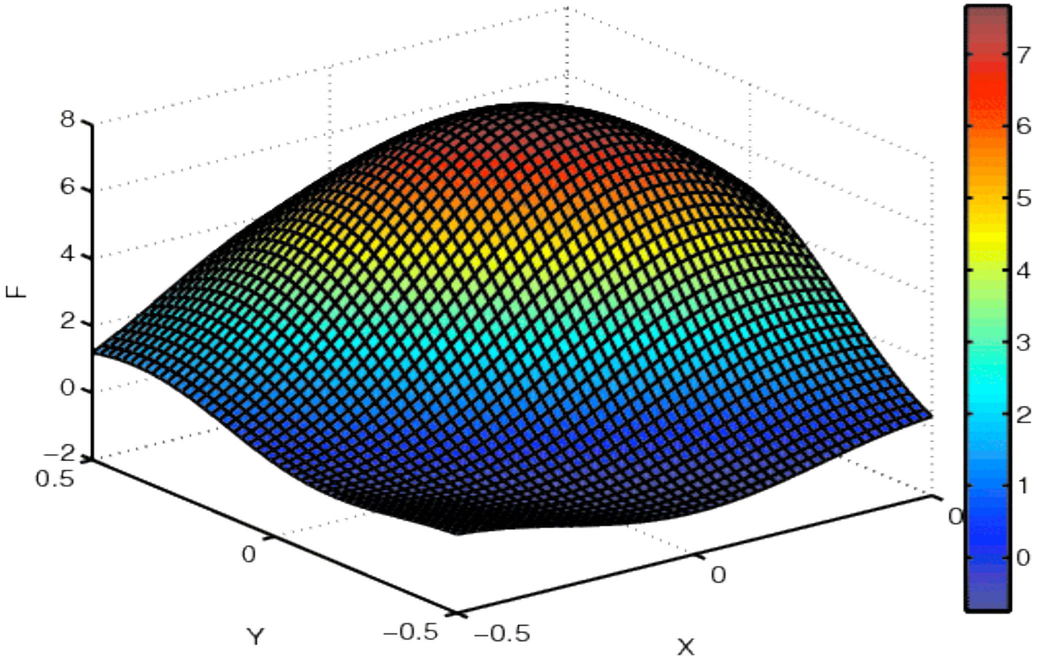
```
mesh(x,y,f)
surf(x,y,f)
```

- Makes a 3-D height plot of f above the x-y plane (wireframe and filled-in respectively)
- Same argument requirements as `contour`
- Uses the color map from pcolor

```
mesh(x,y,f,c)
surf(x,y,f,c)
```

- Uses color given in `c` as the color map

`surf` Example



```
surf(x,y,f);
colorbar;
```

Getting data in & out of Matlab

- Save the contents of your workspace:

  ```
  save
  save filename
  save filename A b x result
  ```

- Loading goes the other way:

  ```
  load
  load filename
  load filename A x
  ```

Loading from other programs

- `load -ascii filename`
  loads numbers into a single array
- Load with C-like syntax:

```
[fid,msg] = fopen(mfile,'r');
if fid<0, fprintf(msg); return; end;
elements = …
    fscanf(fid,'%d %d %e',[3,inf]);
```

- Note: last statement loads arbitrary amounts of data

Output

print -f*fignum* -d*driver* file

- Writes figure *fignum* in the *driver* format to file.ext in the current directory
- Drivers
  - ps, psc, ps2, ps2c—PostScript (ext=ps)
  - eps, epsc, eps2, eps2c
    Encapsulated PostScript (ext=eps)
  - jpeg—JPEG (ext=jpg)
  - tiff—TIFF (ext=tif)
  - png—PNG (ext=png)