COMS W3101-2

Programming Languages: MATLAB

Lecture 2

Spring 2010

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw3101-2.html

Quick Review of Lecture 1

- MATLAB does not use explicit type initialization like other languages

- Just assign some value to a variable name, and MATLAB will automagically understand its type
  - ~~int~~ x
  - x = 3 — double
  - x = 'hello' — char

  } Most common types

- We can assign mathematical expressions to directly create variable
  - x = (3 + 4)/2

Quick Review of Lecture 1

- Row vectors
  - `r = [2 3 5 7];`
  - `r = [2, 3, 5, 7];`

[1x4]

2	3	5	7

- Column vectors
  - `c = [2; 3; 5; 7];`
  - `c = [2 3 5 7]';`

Transpose operator

[4x1]

2
3
5
7

Quick Review of Lecture 1

- Special Vectors Constructors
  - `:` operator
    - `x = 1:3:13;`

      Spacing, default = 1

[1x5]

1	4	7	10	13

  - `linspace()`
    - `x = linspace(0,10,100);`

      Creates a vector of 100 elements with values equally spaced between 0 and 10 (included)

    - Equivalent notation with `:` operator?

Quick Review of Lecture 1

- Explicit Definition
  - `M = [2 4; 3 6; 8 12];`

[3x2]

2	4
3	6
8	12

- Concatenation of vectors
  - `r1 = [2 4];`
  - `r2 = [3 6];`
  - `r3 = [8 12];`
  - `M = [r1; r2; r3];`

- Concatenation of vectors and matrices
  - `r1 = [2 4];`
  - `m1 = [3 6; 8 12];`
  - `M = [r1; m1];`

Dimensions and Type must coincide!

Quick Review of Lecture 1

- Some Predefined Matrix Creation Functions

double

- `M = zeros(2,3);` [3x2] matrix of zeros

  rows columns

0	0	0
0	0	0

- `M = ones(2,3);` [3x2] matrix of ones

1	1	1
1	1	1

- `M = eye(2);` [2x2] identity matrix

1	0
0	1

- `M = rand(2,3);` [2x3] matrix of uniformly distributed random numbers in range [0,1]

0.2	0.86	0.1
1	0	0.33

- `M = randn(2,3)` [2x3] matrix of normally distributed random numbers (mean 0, std dev. 1)

-1.2	-0.86	0.1
1.256	0.435	-1.33

Quick Review of Lecture 1

- Replicating and concatenating matrices
  - repmat
    - `X = [1 2 3; 4 5 6];`
    - `Y = repmat(X,2,4);`

X

1	2	3
4	5	6

Y

1	2	3	1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6	4	5	6
1	2	3	1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6	4	5	6

  - vertcat
    - `x1 = [2 3 4];`
    - `x2 = [1 2 3];`
    - `X = vertcat(x1,x2);`

x1

2	3	4

x2

1	2	3

X

2	3	4
1	2	3

  - horzcat
    - `x1 = [2; 3; 4];`
    - `x2 = [1; 2; 3];`
    - `X = horzcat(x1,x2);`

x1

2
3
4

x2

1
2
3

X

2	1
3	2
4	3

Quick Review of Lecture 1

- Basic Mathematical Operators
  - `+ - * / \ ^`
- Some more complex mathematical functions
  - `sqrt()`
  - `log(), exp()`
  - `sin(), cos(), tan(), atan()`
  - `abs(), angle()`
  - `round(), floor(), ceil()`
  - `conj(), imag(), real()`
  - `sign()`
- Logical Operators
  - `& | ~`
- Relational Operators
  - `> < >= <= == ~=`

Quick Review of Lecture 1

- Operators on matrices
  - `X = [2  3 4; 5 4 6];`
  - `Y = [1 2 3; 3 3 3];`
  - `Rplus = X + Y;`
  - `Rminus = X - Y;`
  - `Rmult = X * Y;`
  - `X2 = X';`
  - `Rmult = X2 * Y;`
  - `Rpoint_mult = X .* Y;`

??? Error using ==> mtimes
Inner matrix dimensions must agree.

X

2	3	4
5	4	6

Y

1	2	3
3	3	3

Rplus

3	5	7
8	7	9

Rminus

1	1	1
2	1	3

Rmult

4	9	16
25	16	36

Rpoint_mult

2	6	12
15	12	18

Some operators, like + and −, are always element wise !
Other operators, like * and /, must be disambiguated with . !

Quick Review of Lecture 1

- Accessing Elements of Matrix M
  - Matrix indexing starts with 1 !
  - Explicit access
    - `element = M(2,3);`
    - `element = M(5);`
  - `:` operator
    - `element = M(1,1:2);`
    - `element = M(:,1);`
  - `end` operator
    - `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33

Quick Review of Lecture 1

- Accessing Elements of Matrix M
  - Matrix indexing starts with 1 !
  - Explicit access
    - `element = M(2,3);`
    - `element = M(5);`
  - `:` operator
    - `element = M(1,1:2);`
    - `element = M(:,1);`
  - `end` operator
    - `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33

Quick Review of Lecture 1

- Accessing Elements of Matrix M
  - Matrix indexing starts with 1 !
  - Explicit access
    - `element = M(2,3);`
    - `element = M(5);`
  - `:` operator
    - `element = M(1,1:2);`
    - `element = M(:,1);`
  - `end` operator
    - `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33

Quick Review of Lecture 1

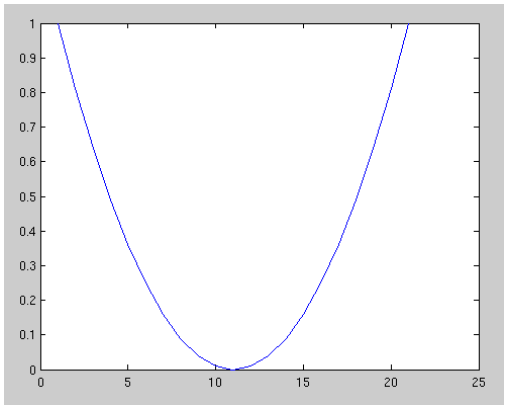
- Accessing Elements of Matrix M
  - Matrix indexing starts with 1 !
  - Explicit access
    - `element = M(2,3);`
    - `element = M(5);`
  - `:` operator
    - `element = M(1,1:2);`
    - `element = M(:,1);`
  - `end` operator
    - `element = M(1,2:end);`

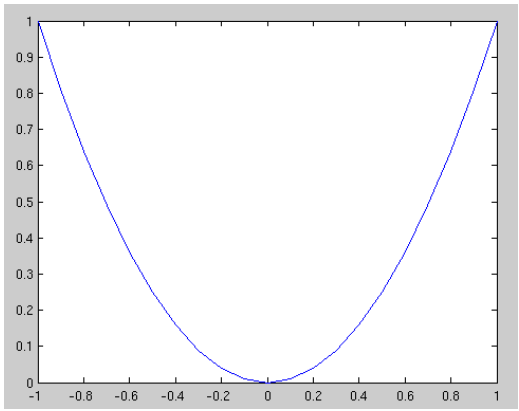
M

-1.2	-0.86	0.1
1.256	0.435	-1.33

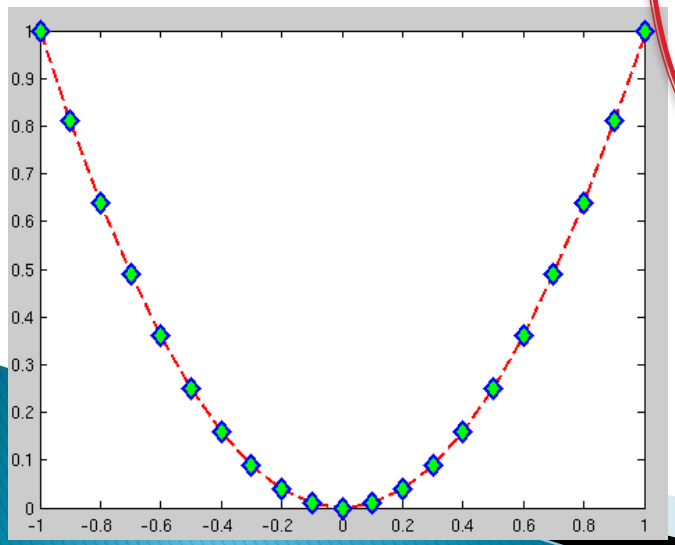
Quick Review of Lecture 1

- plot()
  - `x = [-1:0.1:1];`
  - `y = x.^2;`
  - `plot(y);`
  - `plot(x,y);`






```
plot(x,y,'--rd','LineWidth',2,...
         'MarkerEdgeColor','b',...
         'MarkerFaceColor','g',...
         'MarkerSize',10);
```



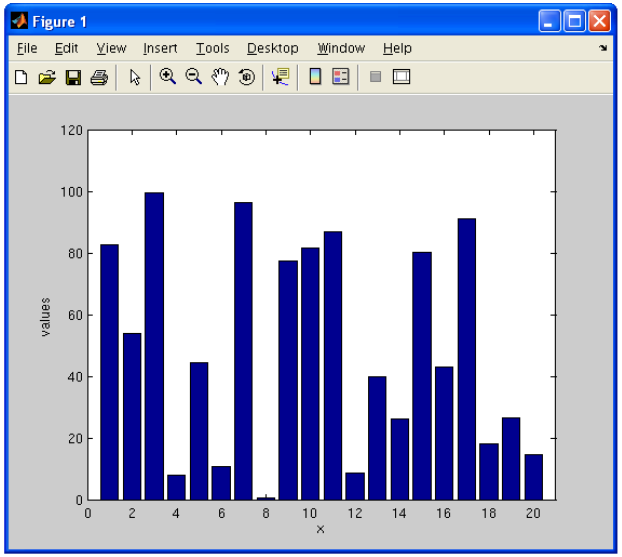
- Line style – –
- Line color ‘red’
- Marker Type ‘diamond’

Quick Review of Lecture 1

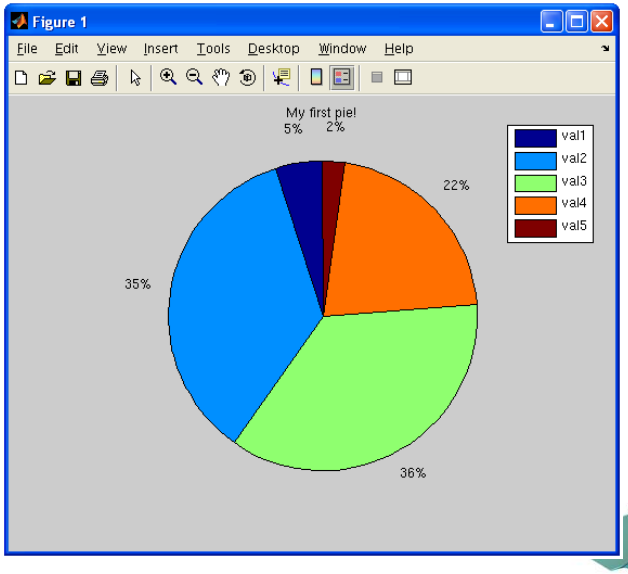
- bar()
  - x = 100*rand(1,20);
  - bar(x);
  - xlabel('x');
  - ylabel('values');
  - axis([0 21 0 120]);

    x range y range

  xlim([0 21]); ylim([0 120]);



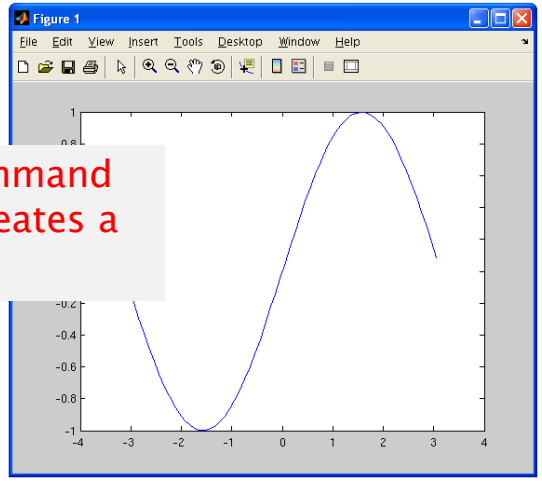
- pie()
  - x = 100*rand(1,5);
  - pie(x);
  - title('My first pie!');
  - legend('val1','val2',...
    'val3','val4','val5');

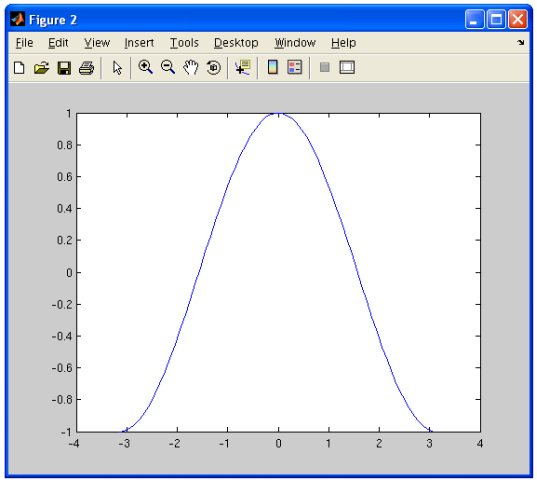


Quick Review of Lecture 1

- figure
  - To open a new Figure and avoid overwriting plots
  - `x = [-pi:0.1:pi];`
  - `y = sin(x);`
  - `z = cos(x);`
  - `plot(x,y);`
  - `figure`
  - `plot(x,z);`

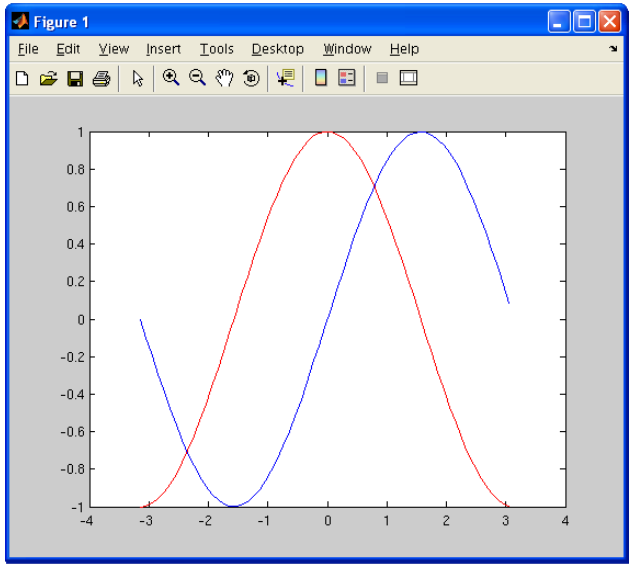
The fist plot command automatically creates a new Figure!





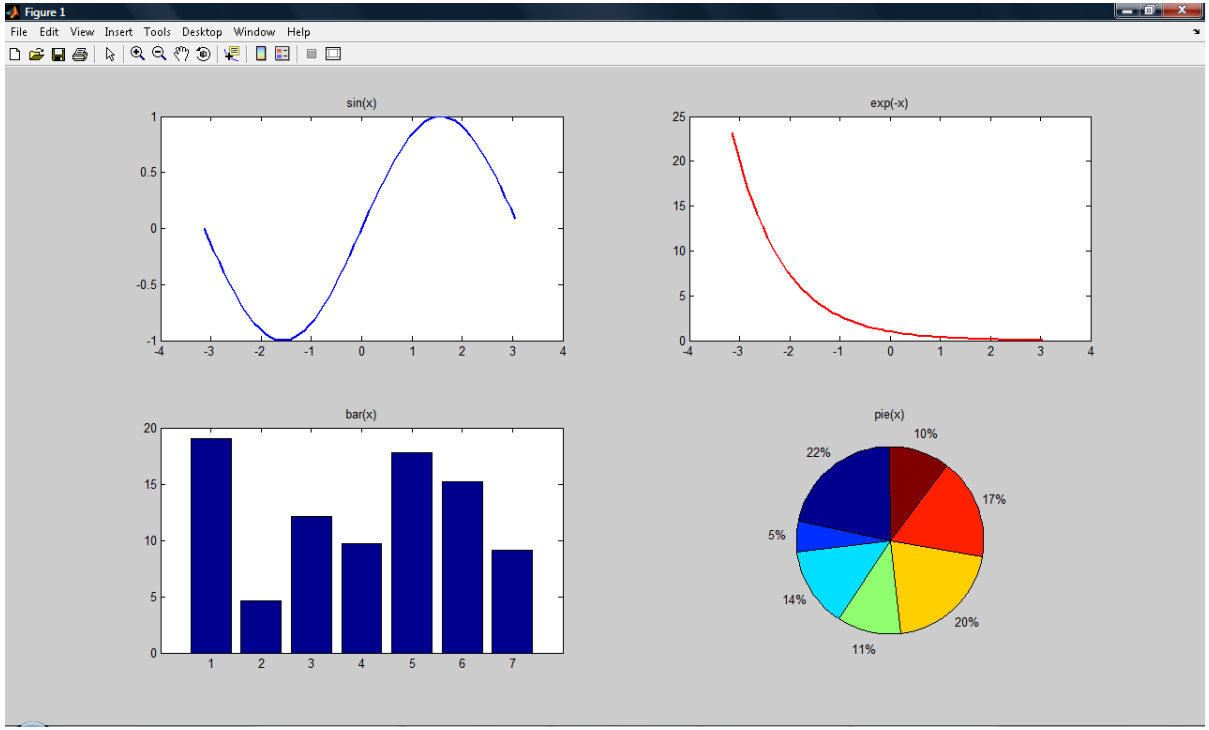
- Close figures
  - `close 1`
  - `close all`

- Multiple plots in same Graph
  - `plot(x,y);`
  - `hold on`
  - `plot(x,z,'r');`
  - `hold off`



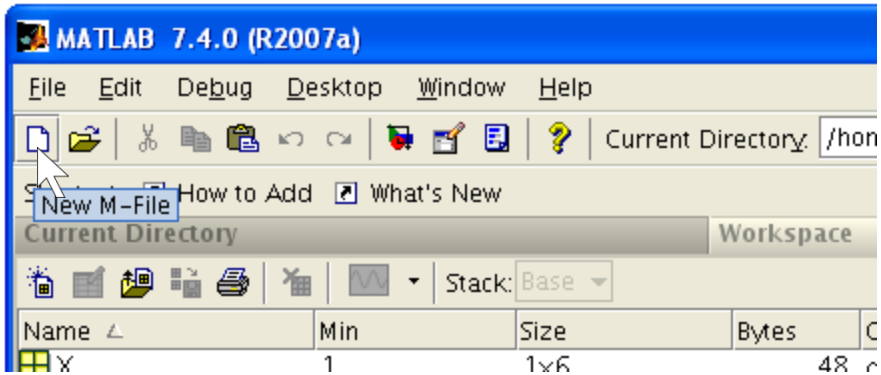
Quick Review of Lecture 1

- Multiple plots in same Figure
  - `figure(1)`
  - `subplot(2,2,1)`
  - `plot(x,y);`
  - `title('sin(x)');`
  - `subplot(2,2,2)`
  - `plot(x,z,'r');`
  - `title('exp(-x)');`
  - `subplot(2,2,3)`
  - `bar(x);`
  - `title('bar(x)');`
  - `subplot(2,2,4)`
  - `pie(x);`
  - `title('pie(x)');`

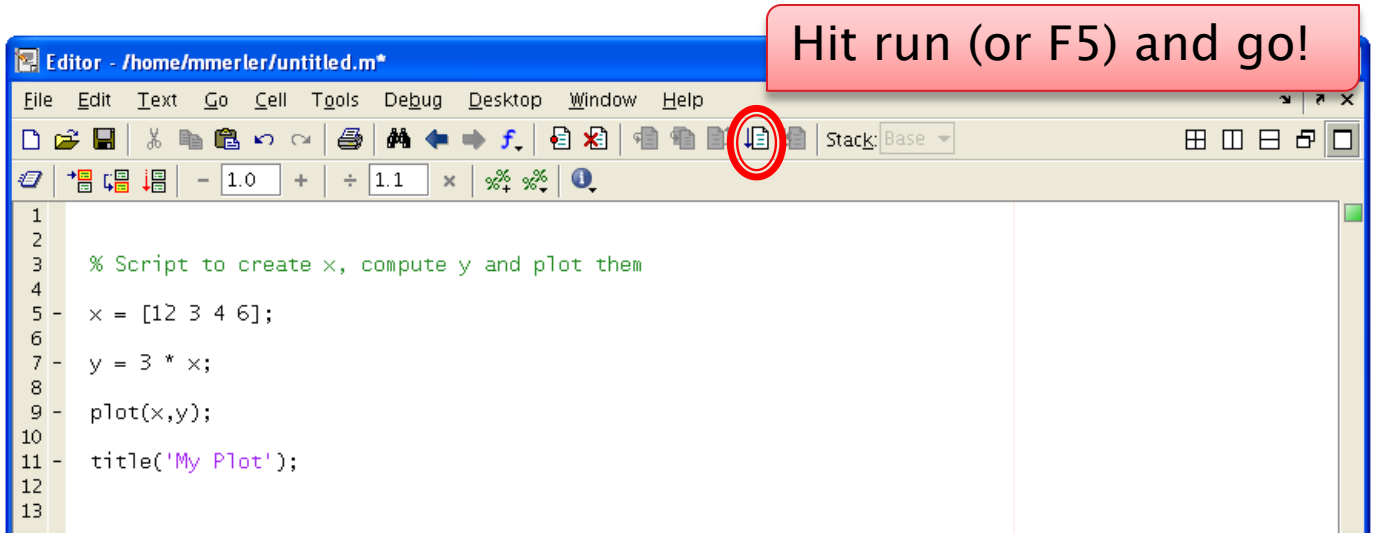


Quick Review of Lecture 1

- Like a notebook, but for code!



- M-files are MATLAB specific script files, they are called *namefile*.m

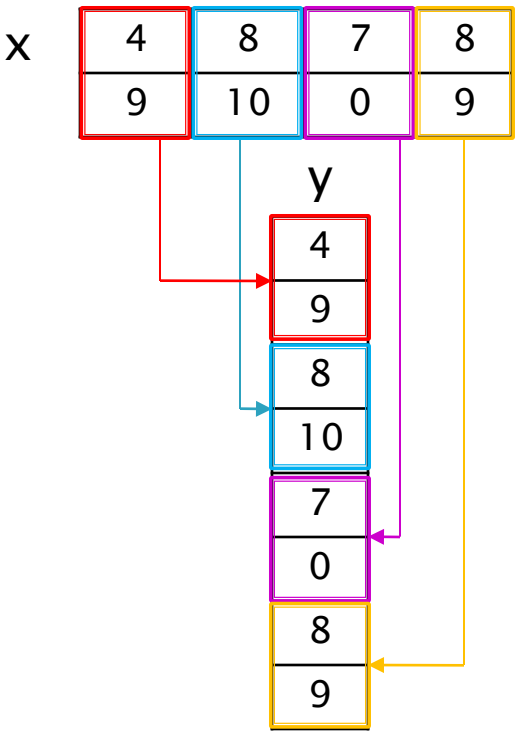


End of review

Reshaping Matrices

- Using the `:` operator
  - `x = round(10*rand(2,4));`
  - `y = x(:);`

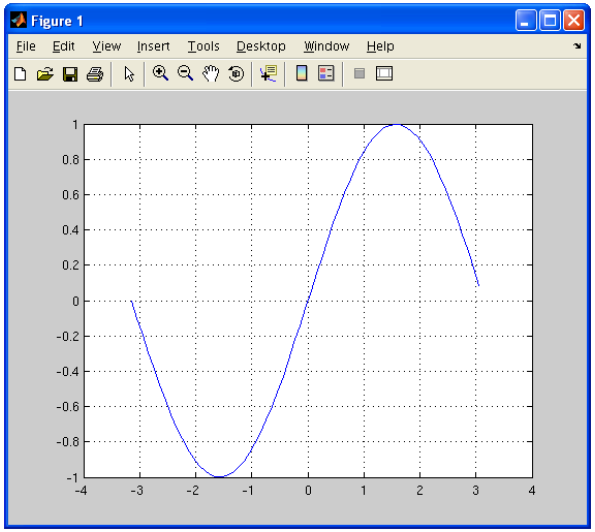
The elements of x are stacked in a column vector, column after column



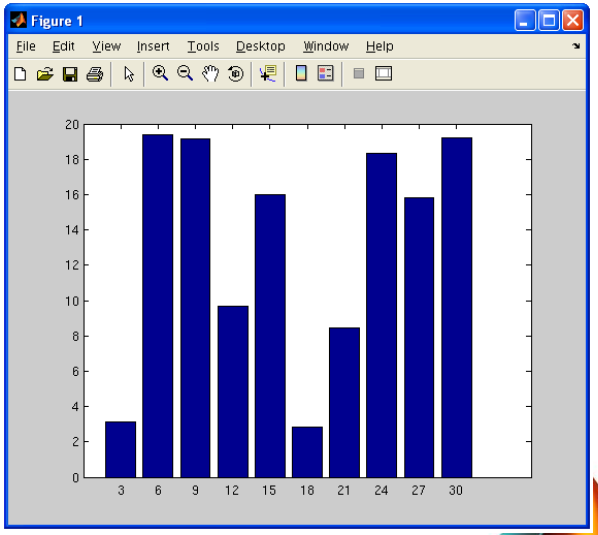
- Using the `reshape()` function
  - `x2 = reshape(y,2,4);`

More Plotting

- grid
  - `x = [-pi:0.1:pi];`
  - `y = sin(x);`
  - `plot(x,y);`
  - `grid on`

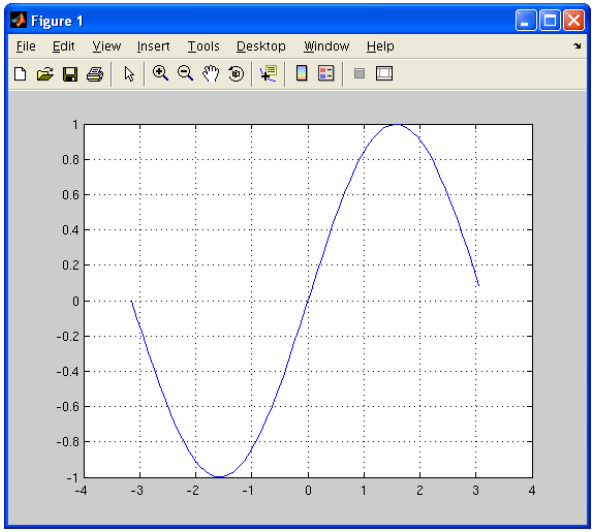


- Specify Tickmarks
  - `x = 20 * rand(1,10);`
  - `bar(x);`
  - `set(gca,'XTickLabel', [3:3:30]);`

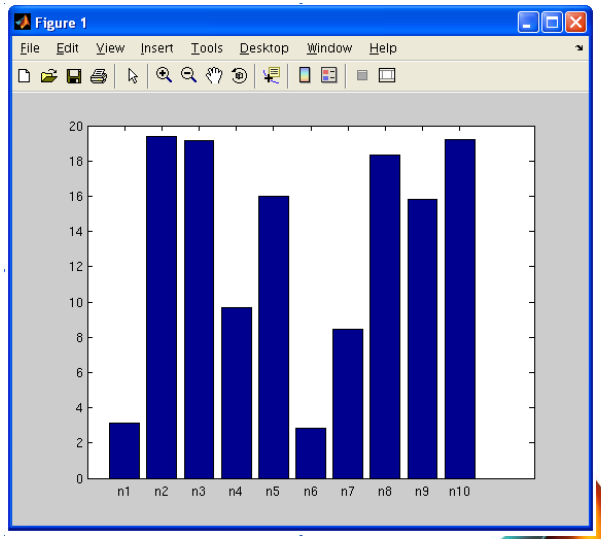


More Plotting

- grid
  - `x = [-pi:0.1:pi];`
  - `y = sin(x);`
  - `plot(x,y);`
  - `grid on`



- Specify Tickmarks
  - `x = 20 * rand(1,10);`
  - `bar(x);`
  - `set(gca,'XTickLabel', [3:3:30]);`
  - `names = {'n1', 'n2', 'n3' ,'n4',...`
    `'n7', 'n8' ,'n9', 'n10'};`
  - `set(gca,'XTickLabel', names);`



Plotting Areas

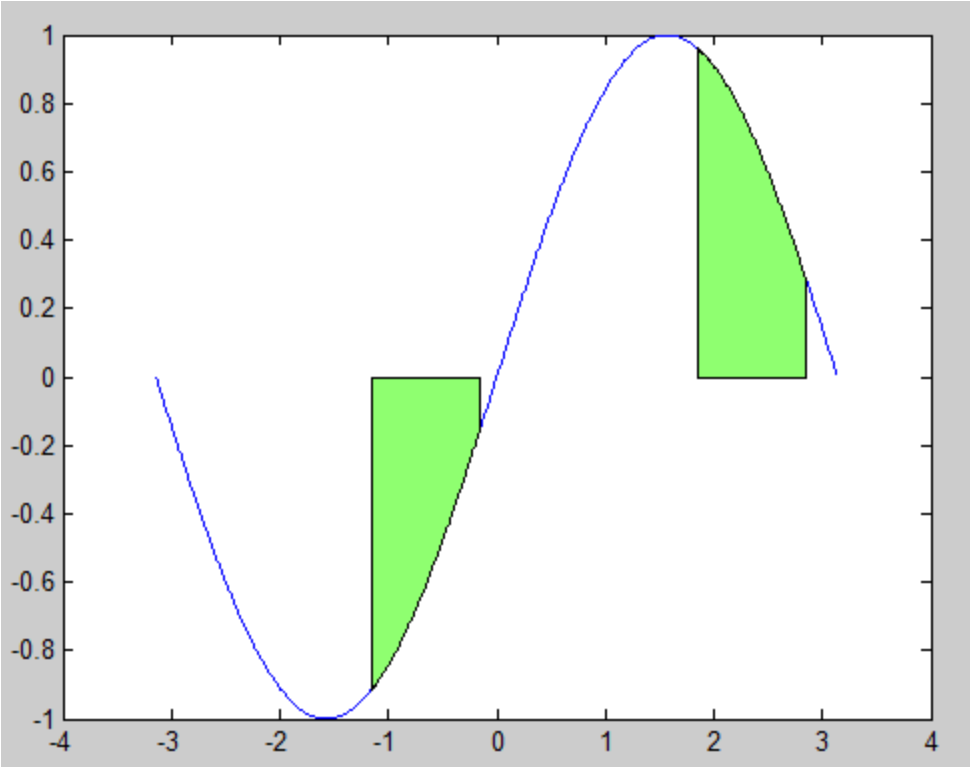
- area()

```
x = [-pi:0.01:pi];
y = sin(x);
plot(x,y);
hold on;
area(x(200:300),y(200:300));
area(x(500:600),y(500:600));
hold off
```

Plotting Areas

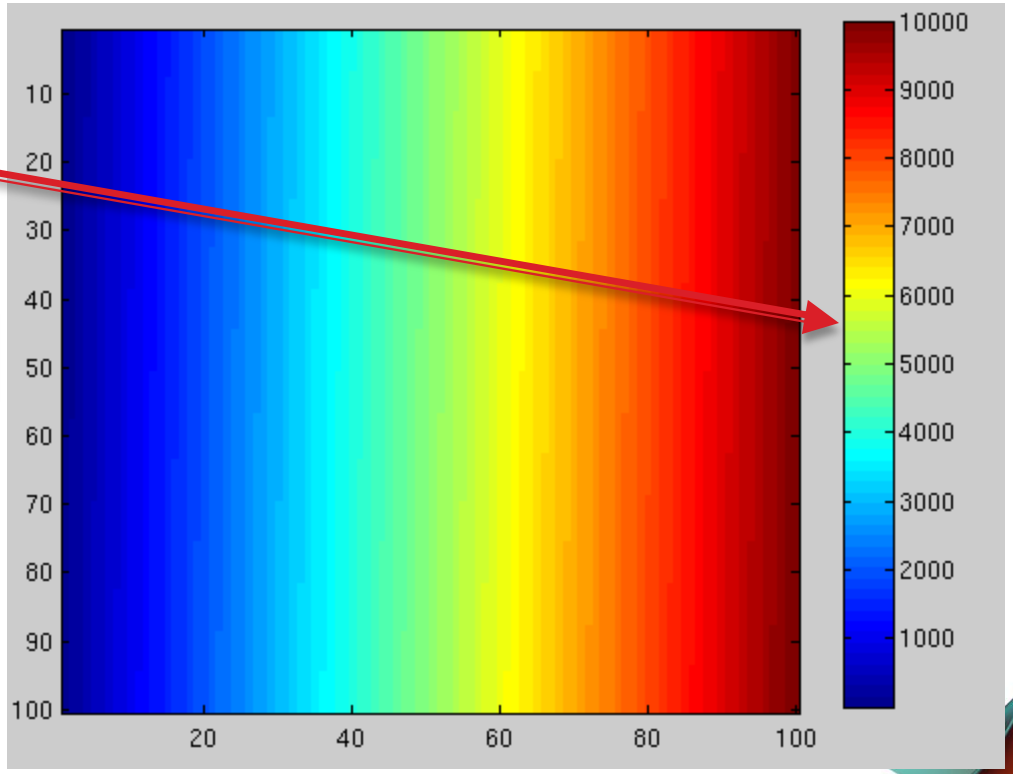
- area()

```
x = [-pi:0.01:pi]
y = sin(x);
plot(x,y);
hold on;
area(x(200:300),y(200:300));
area(x(500:600),y(500:600));
hold off
```



More Plotting – Colormaps

- colormap
  - `x = reshape(1:10000,100,100);`
  - `imagesc(x);`
  - `colorbar`



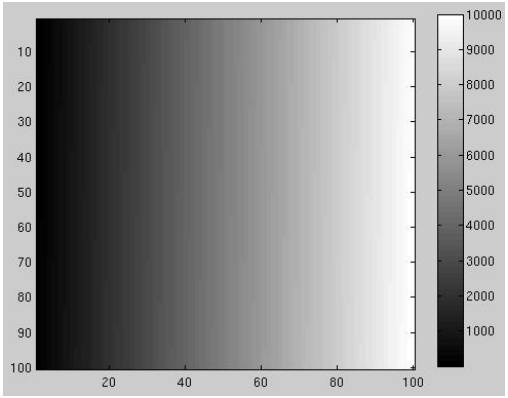
Default colormap is `'jet'`

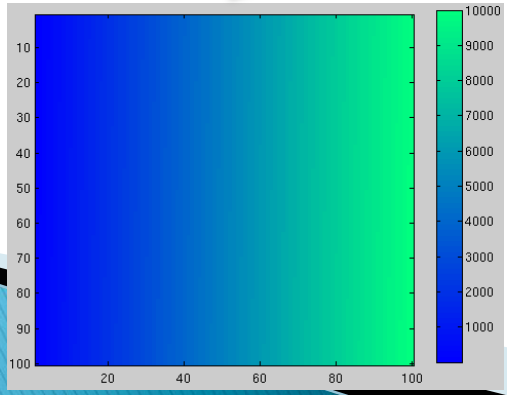
More Plotting – Colormaps

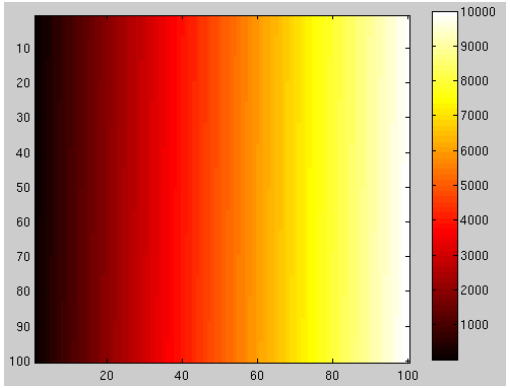
- `colormap`
  - `x = reshape(1:10000,100,100);`
  - `imagesc(x);`
  - `colorbar`

Built-in colormaps

- `colormap(gray)`
- `colormap(hot)`
- `colormap(winter)`



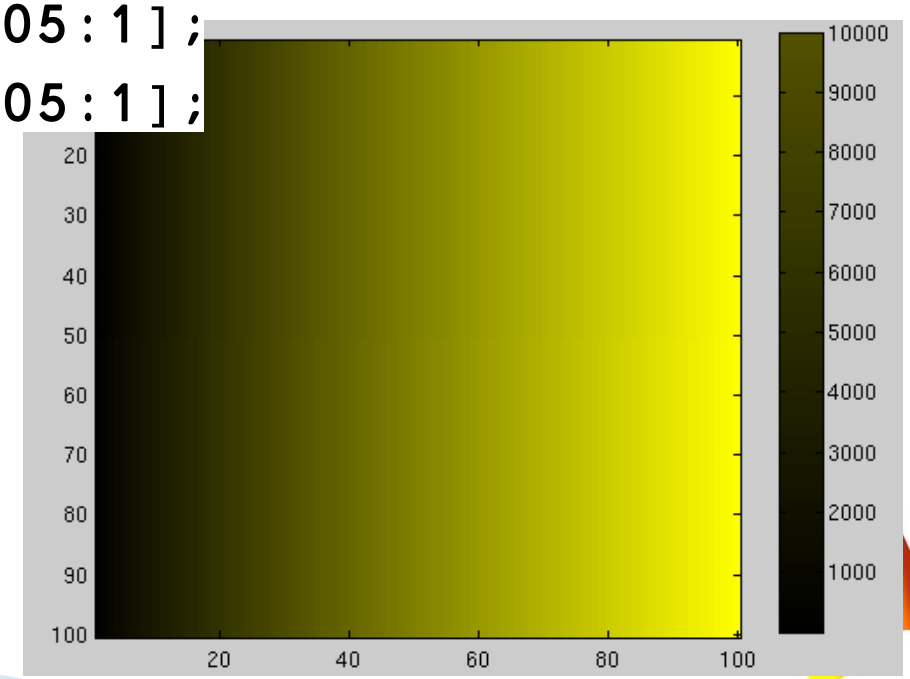




More plotting – Colormaps

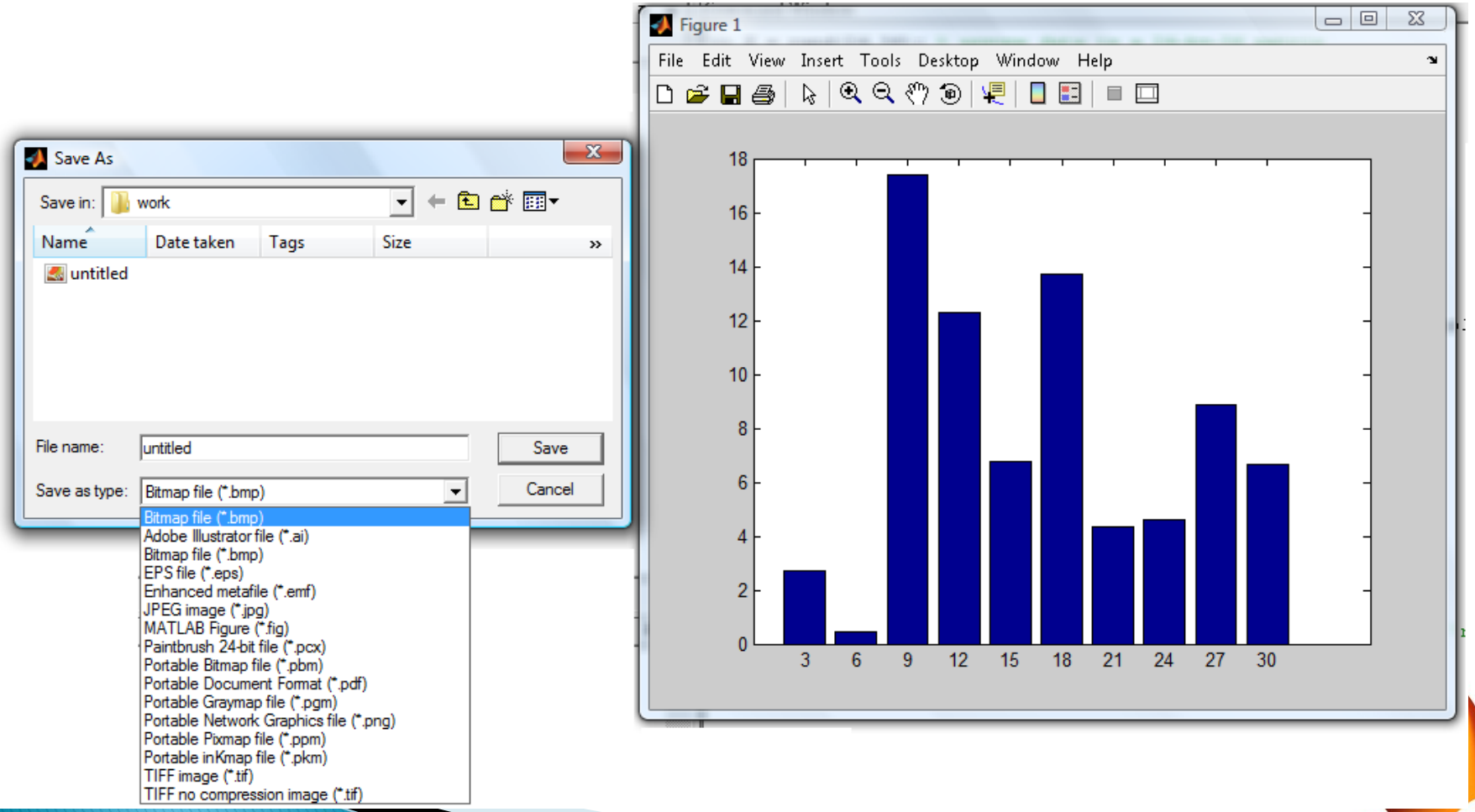
- Create your own colormap
  - A colormap is a m-by-3 matrix of real numbers between 0.0 and 1.0. Each row defines one color

```
map=zeros(200,3);
map(:,1)= [0.005:0.005:1];
map(:,2)= [0.005:0.005:1];
colormap(map);
```



Saving Figures

Traditional Way



Saving and Loading Figures

- `.fig` format (MATLAB format for figures)

- `openfig`
  - this function is used to load previously saved MATLAB figures

  - `openfig('figFileName')`

Saving Figures

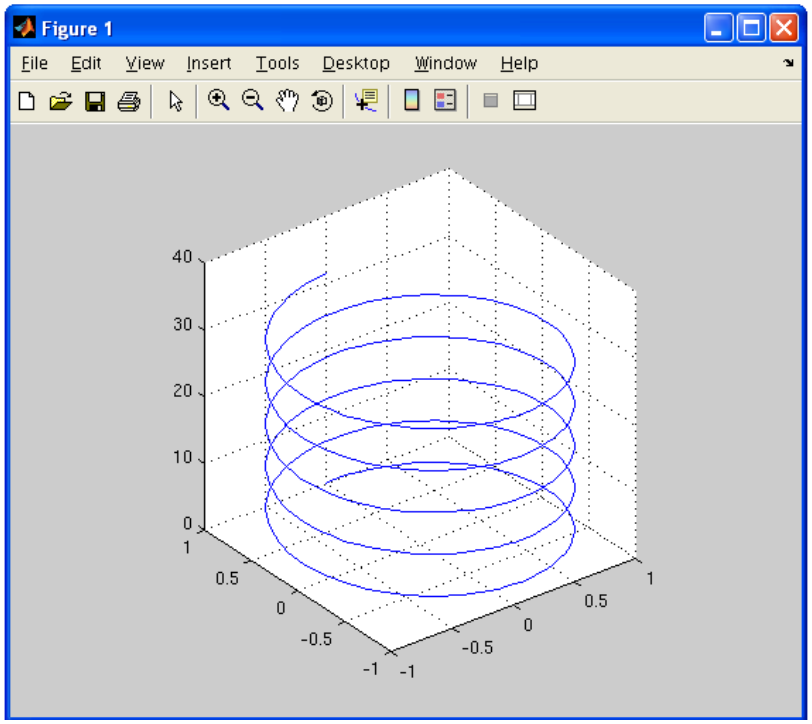
Smart way

- `print`
  - **General Form**
  - `print` -d*format* filename
  - **Example**
  - `print -depsc 'figure.eps'`

eps is a cool format that stores your image in a vectorized way, which avoids quality loss after rescaling. It's particularly useful when used within Latex

3D Plotting

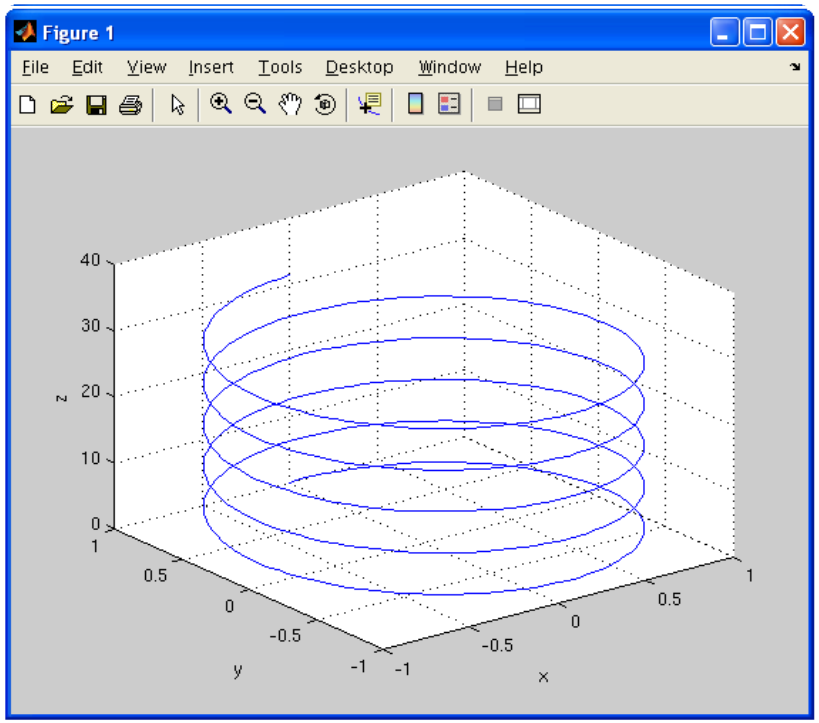
- **Line Plot same as 2D, just add a 3 suffix!**
  - `t = 0:pi/50:10*pi;`
  - `plot3(sin(t),cos(t),t);`
  - `grid on`
  - `axis square`



3D Plotting

- **Line Plot same as 2D, just add a 3 suffix!**
  - `t = 0:pi/50:10*pi;`
  - `plot3(sin(t),cos(t),t);`
  - `grid on`
  - `axis square`

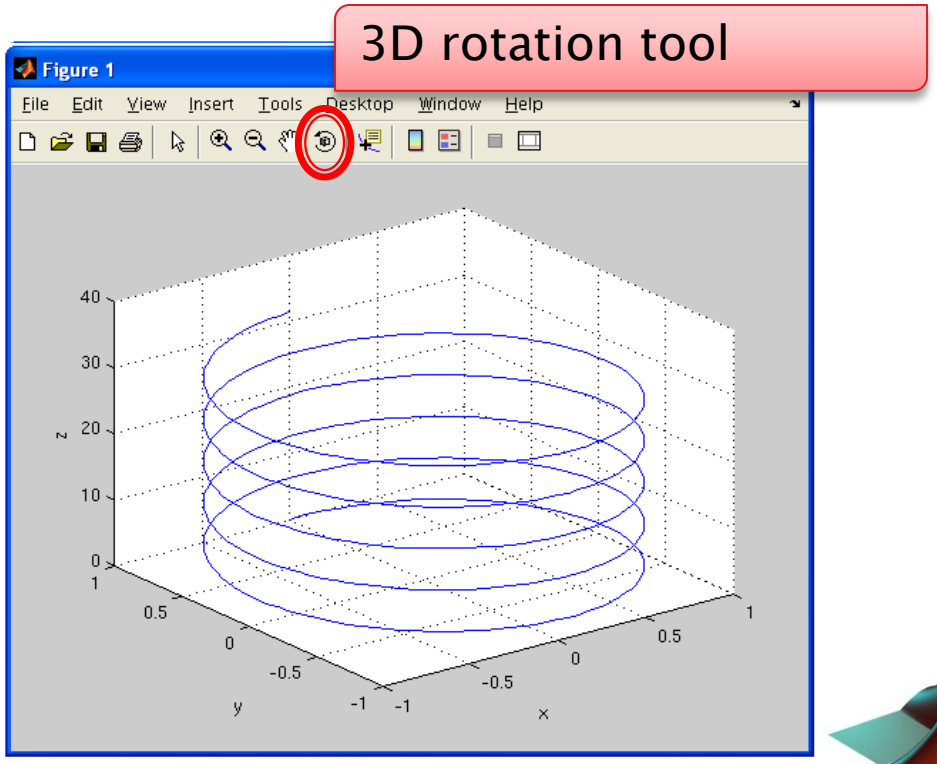
  - `xlim([-1 1]);`
  - `ylim([-1 1]);`
  - `zlim([0 40]);`
  - `xlabel('x');`
  - `ylabel('y');`
  - `zlabel('z');`



3D Plotting – Functions

- **Line Plot same as 2D, just add a 3 suffix!**
  - `t = 0:pi/50:10*pi;`
  - `plot3(sin(t),cos(t),t);`
  - `grid on`
  - `axis square`

  - `xlim([-1 1]);`
  - `ylim([-1 1]);`
  - `zlim([0 40]);`
  - `xlabel('x');`
  - `ylabel('y');`
  - `zlabel('z');`



3D Plotting – Surfaces

- meshgrid
  - `xvec = [1:10];`
  - `yvec = [-2:2];`
  - `[X Y] = meshgrid(xvec,yvec);`

X = [1x10]

1	2	3	4	5	6	7	8	9	10

−2	−1	0	1	2

y = [1x5]

`meshgrid(xvec,yvec)` produces grids containing all combinations of *xvec* and *yvec* elements, in order to create the domain for a 3D plot of a function z = f(xvec,yvec)

X, Y= [5x10]

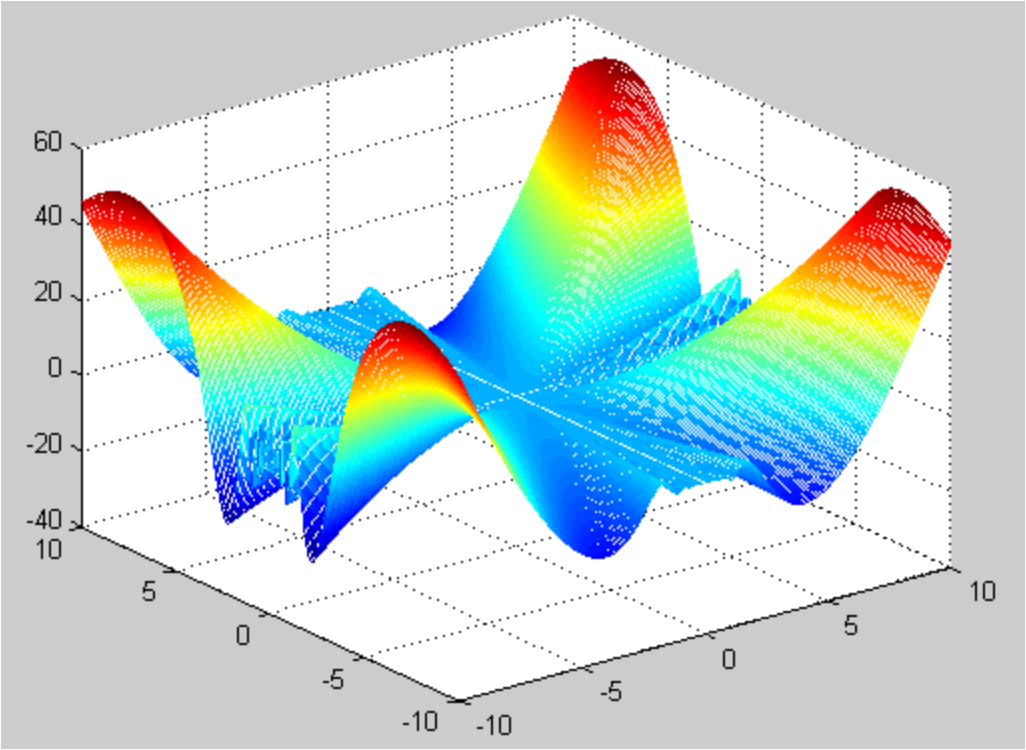
X

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

Y

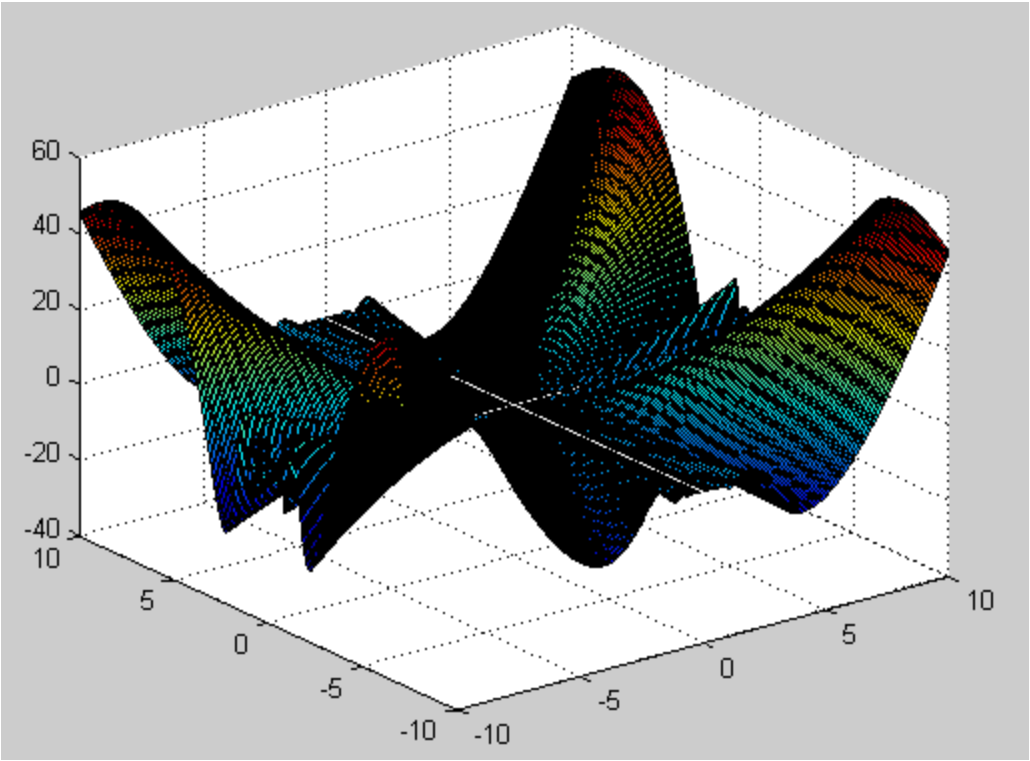
−2	−2	−2	−2	−2	−2	−2	−2	−2	−2
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2

3D Plotting Surfaces



- mesh
  - `xvec = [-10:0.1:10];`
  - `yvec = xvec;`
  - `[X Y] = meshgrid(xvec,yvec);`
  - `Z = X.*Y.*sin(X./Y).* cos(Y./X);`
  - `mesh(X,Y,Z);`

3D Plotting Surfaces



- surf
  - `xvec = [-10:0.1:10];`
  - `yvec = xvec;`
  - `[X Y] = meshgrid(xvec,yvec);`
  - `Z = X.*Y.*sin(X./Y).* cos(Y./X);`
  - `surf(X,Y,Z);`

3D Plotting – Surfaces

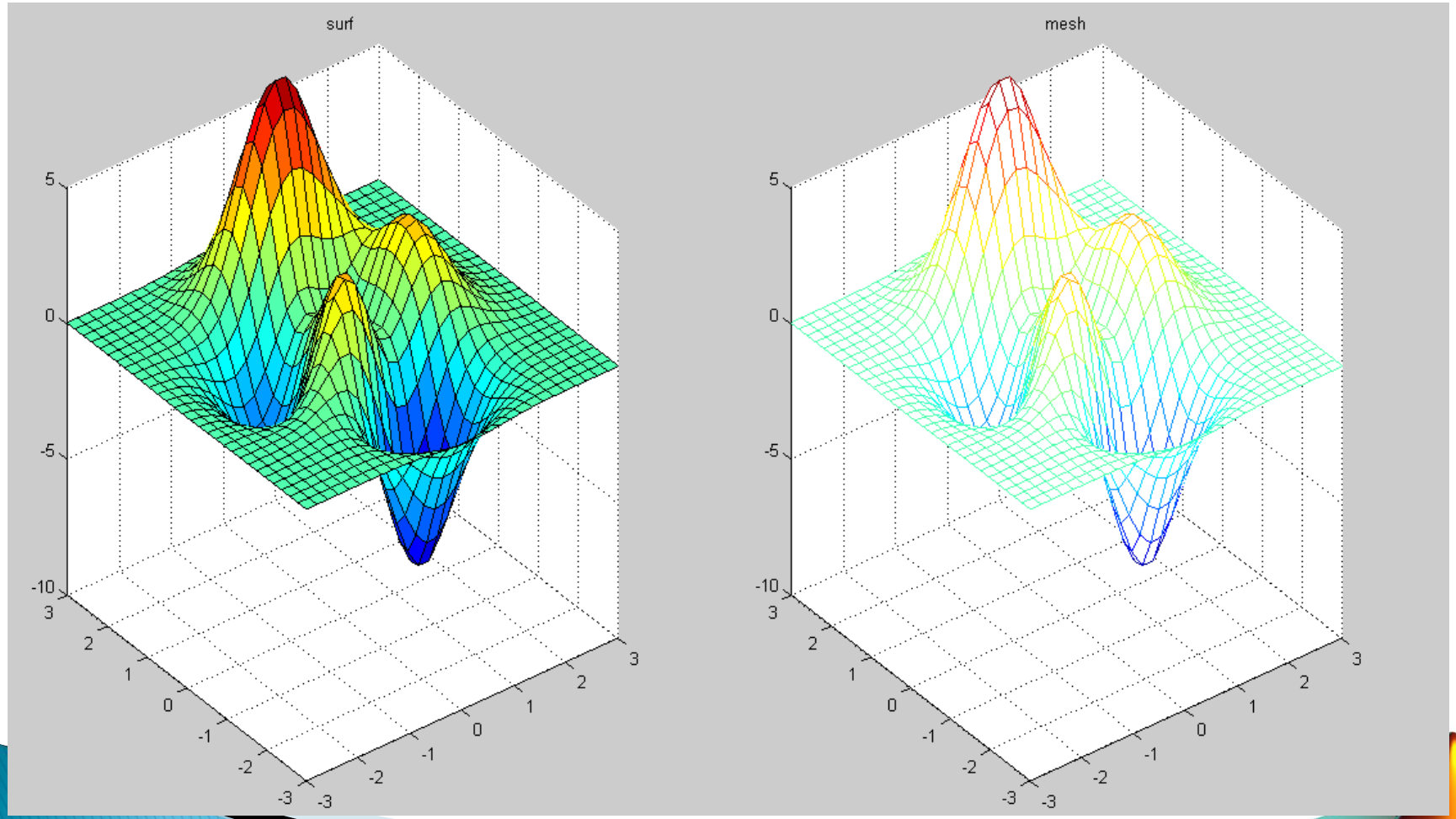
- surf vs. mesh

```
[X,Y,Z] = peaks(30);
subplot(1,2,1)
surf(X,Y,Z)
title('surf')
axis([-3 3 -3 3 -10 5]);

subplot(1,2,2)
mesh(X,Y,Z)
title('mesh')
axis([-3 3 -3 3 -10 5]);
```

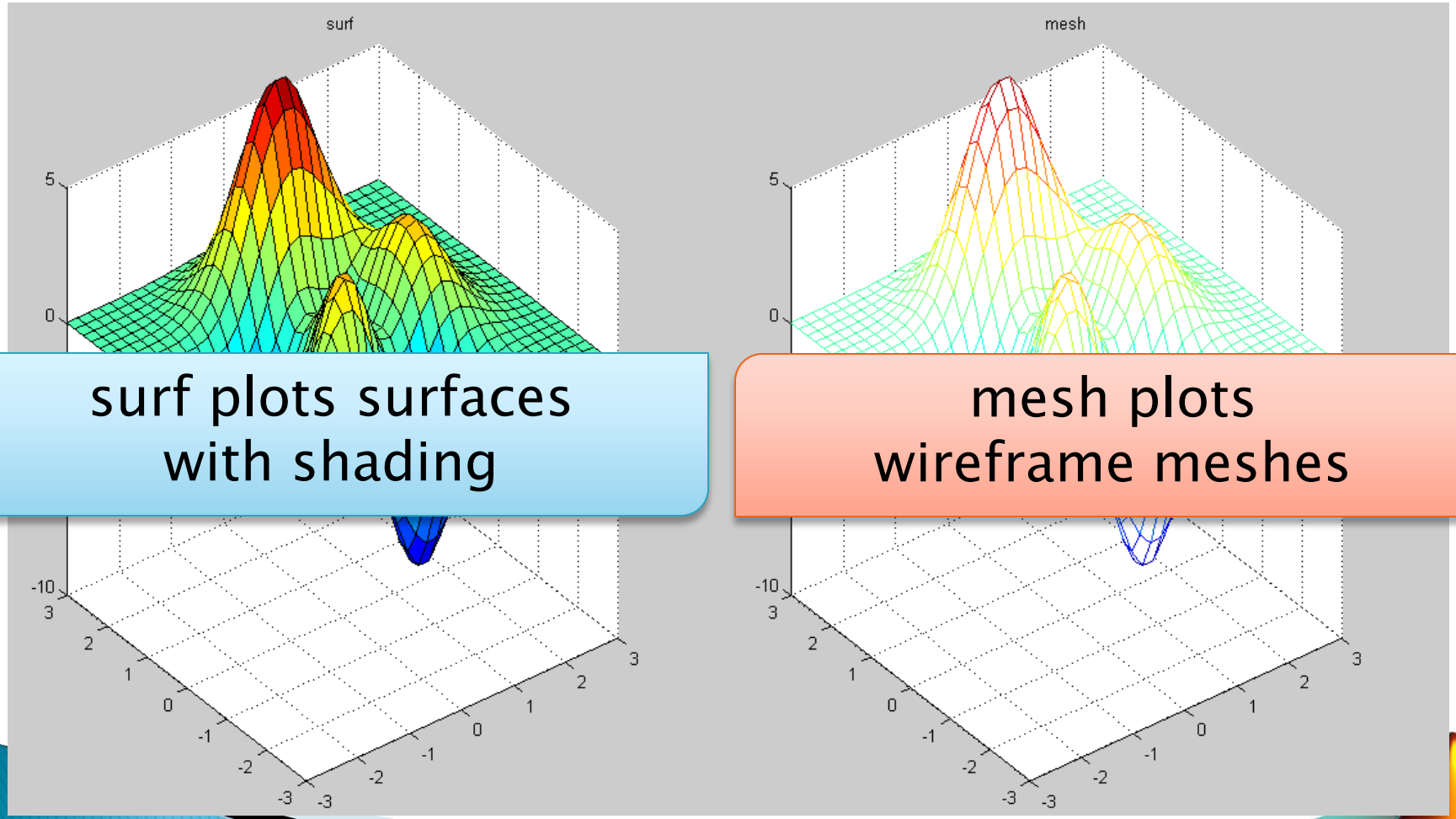
3D Plotting – Surfaces

- `surf` vs. `mesh`



3D Plotting – Surfaces

- `surf` vs. `mesh`

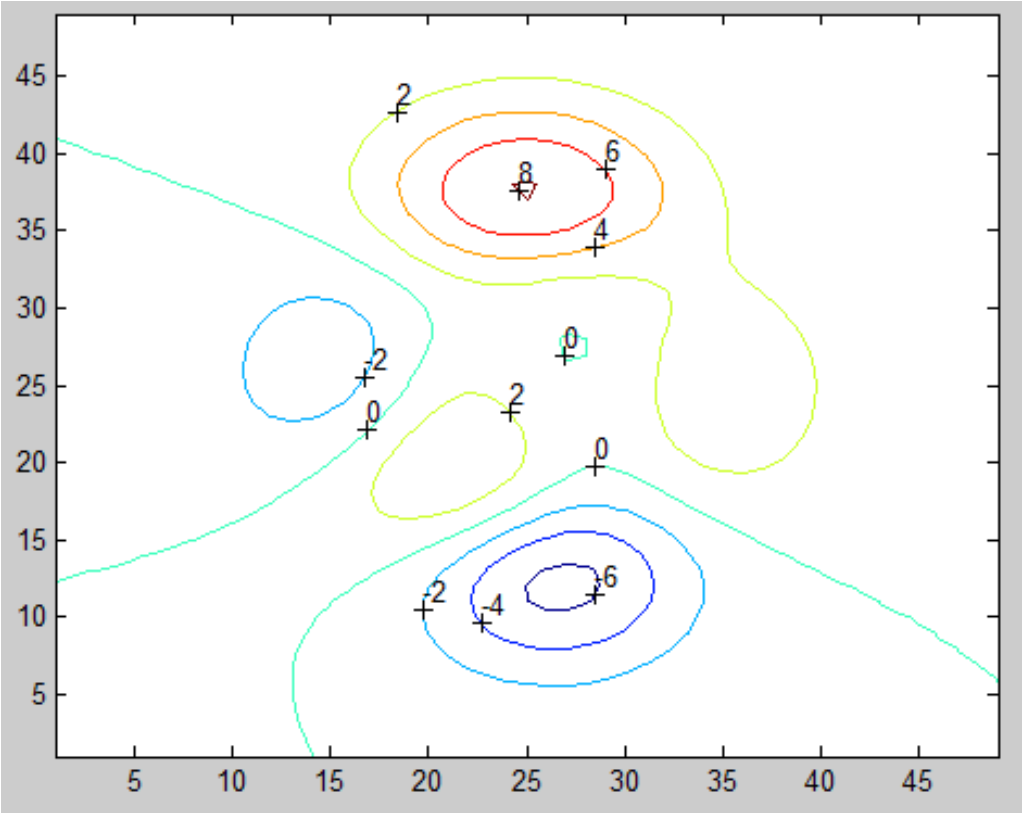


surf plots surfaces with shading

mesh plots wireframe meshes

3D Plotting

- `contour`
  - `Z = peaks;`
  - `c = contour(Z);`
  - `clabel(c)`

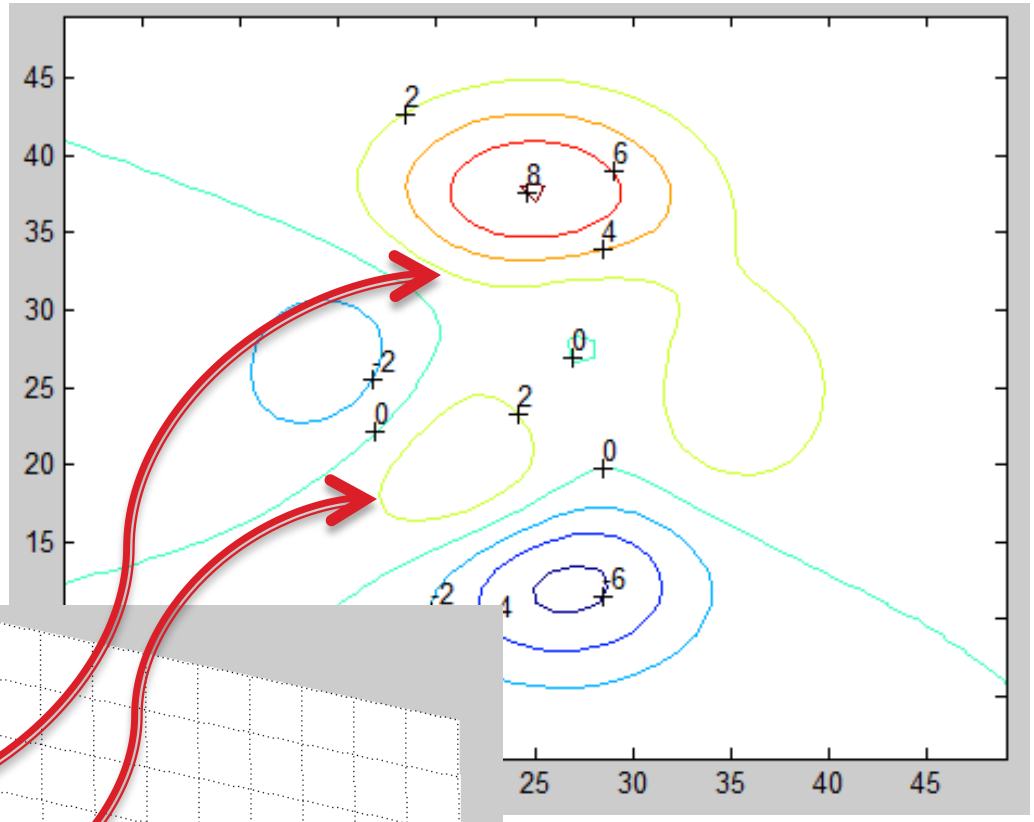


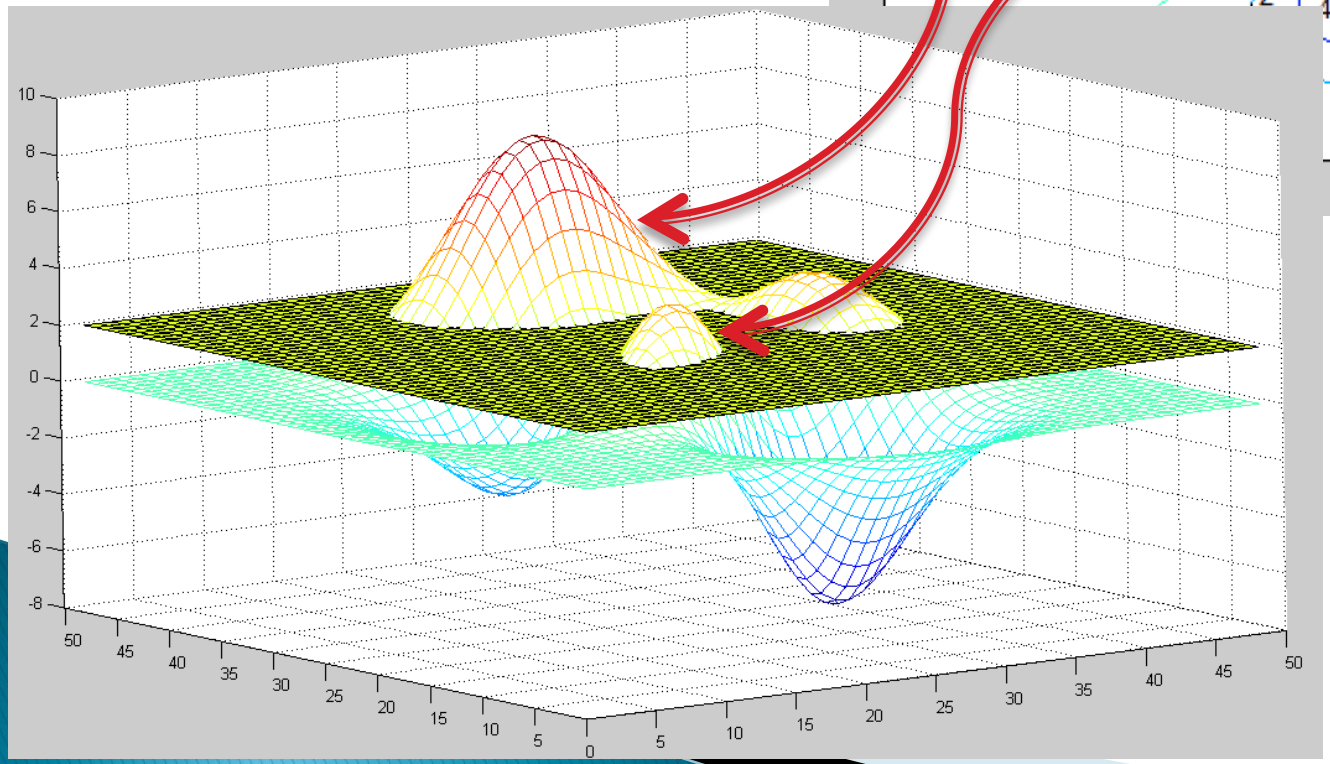
Contour plots the intersections between a surface and replicas of its domain plane shifted by constant values (in the example, shifts = [−6 −4 −2 0 2 4 6 8])

3D Plotting

- `contour`

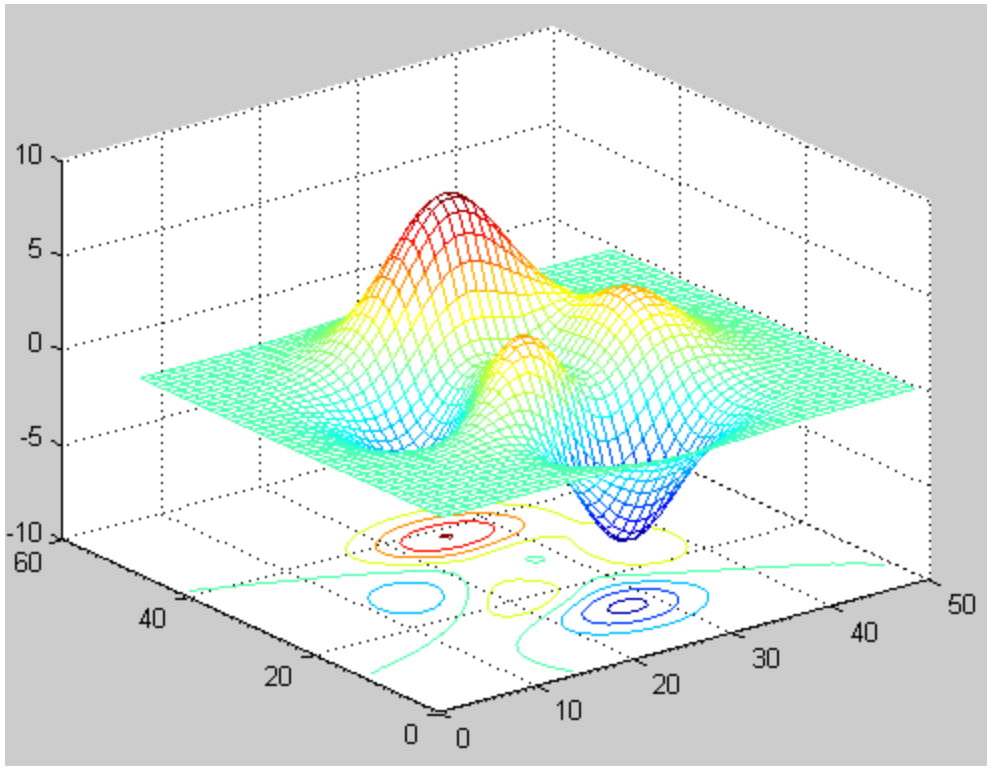
shift = 2





3D Plotting

- surf(mesh) + contour = surfc(meshc)
  - Z = peaks;
  - meshc(Z);



Control Flow

- All flow control statements begin with a *keyword* and finish with the word `end`

- `if`

Example	General Form
`x = 2`	
`if x==2`	`if (`*conditional statement*`)`
`  y = x`	`  body`
`end`	`end`

Control Flow

- `if`

Example 2

```
x = 2
if x==2
   y = x
else
   y = 3
end
```

General Form 2

```
if (conditional statement)
    body 1
else
    body 2
end
```

Control Flow

- `if`
  - **Example 3**
  - `x = 1`
  - `if x==2`
  - `  y = x`
  - `elseif x==3`
  - `  y = 3`
  - `else`
  - `  y = 4`
  - `end`

General Form 3

```
if (conditional statement 1)
   body 1
elseif (conditional statement 2)
   body 2
else
   body 3
end
```

Control Flow

- switch

Example

```
x = 1
switch x
 case 0
    y = 1
 case 1
    y = 1
 case 2
    y = 2
 otherwise
    y = 3
end
```

General Form

```
switch(variable)
   case variable value1
        body 1
   case variable value2
        body 2
   case variable value2
        body 3
   otherwise (for all other variable vaues)
        body 4
end
```

Control Flow

- `try, catch`
  - **Example**
  - `a = rand(3,1);`
  - `try`
  - `    x = a(10);`
  - `catch`
  - `    disp('error')`
  - `end`

General Form

```
try
    statement 1
catch
    statement 2
end
```

Loops

- Loops are used to repeat the same operations multiple times

- `for`
  - **Example**
  - `for x=1:2:9`
  - `  disp(x)`
  - `end`

General Form

```
for (iteration variable)
    body
end
```

Loops

- while
  - **Example**
  - x=1;
  - while (x < 10)
  - x = x+1;
  - end

General Form

```
while (conditional statement)
    body
end
```

Loops

- continue
  - **Example**
  - `y = 0;`
  - `for x=1:10`
  - `  if(x == 3)`
  - `    continue`
  - `  end`
  - `  y = y + 1;`
  - `end`

- break
  - **Example**
  - `y = 0;`
  - `for x=1:10`
  - `  if(x == 3)`
  - `    break`
  - `  end`
  - `  y = y + 1;`
  - `end`

$y = ?$

Loops

- continue
  - **Example**

```
y = 0;
for x=1:10
  if(x == 3)
    continue
  end
  y = y + 1;
end
```

  - y = 9

- break
  - **Example**

```
y = 0;
for x=1:10
  if(x == 3)
    break
  end
  y = y + 1;
end
```

  - y = 2

Computing Time

- tic – toc
  - tic
  - y = 0;
  - for x=1:2e5
  - y = y+1;
  - end
  - toc

  2e5 = 200000

  Elapsed time is 0.094 seconds.

  - endtime = toc;

- cputime
  - t1 = cputime;
  - y = 0;
  - for x=1:2e5
  - y = y+1;
  - end
  - t2 = cputime-t1

  t2 = 0.0938 (seconds)

Loops – Efficiency

- Loops are very inefficient in MATLAB
- Only one thing to do: AVOID THEM !!!
- Try using built-in functions instead

```
y = rand(2e6,1);
c = 0;
tic
for x=1:length(y)
  if(y(x)>0.5)
     c = c + 1;
  end
end
toc
```

Elapsed time is 0.2374 seconds.

```
y = rand(2e6,1);
tic
c = length(find(y>0.5));
toc
```

Elapsed time is 0.1872 seconds.

Loops – Efficiency

- Only one thing to do: AVOID THEM !!!

```
mat = rand(2e4,3e2);
s = 0;
tic
for y=1:size(mat,1)
  for x=1:size(mat,2)
     s = s + mat(y,x);
  end
end
toc
```

Elapsed time is 0.7915 seconds.

```
mat = rand(2e4,3e2);
tic
s = sum(sum(mat));
toc
```

Elapsed time is 0.5409 seconds.

Loops – Efficiency

- Allocating memory before loops greatly speeds up computation time !!!

```
x = rand(1,1e4);
y = zeros(1,1e4);
for n=1:1e4
  if n==1
    y(n) = x(n);
  else
    y(n) = x(n-1) + x(n);
  end
end
```

Elapsed time is 2.846e-04 seconds.

```
x = rand(1,1e4);

for n=1:1e4
  if n==1
    y(n) = x(n);
  else
    y(n) = x(n-1) + x(n);
  end
end
```

Elapsed time is 0.0309 seconds.

Loops – Efficiency

- Still, usually MATLAB offers more efficient solutions

```
x = rand(1,1e4);
y = zeros(1,1e4);
for n=1:1e4
  if n==1
    y(n) = x(n);
  else
    y(n) = x(n-1) + x(n);
  end
end
```

Elapsed time is 2.846e-04 seconds.

```
x = rand(1,1e4);
y = [0 x(1:end-1)] + x;
```

Elapsed time is 1.885e-4 seconds.

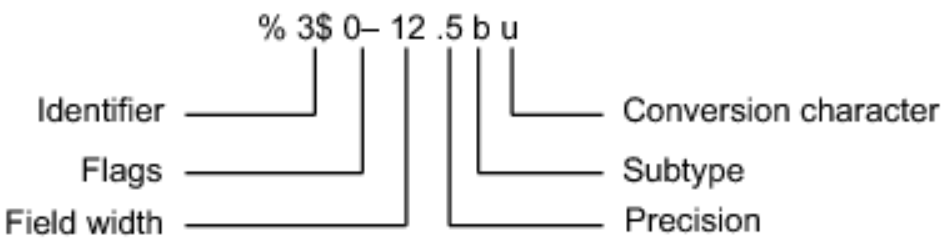
Printing functions

- `fprintf, sprintf`
  - `printf(file,'format1 format2…',var1, var2, …)`
  - `s = sprintf('format1 format2…',var1, var2, …)`
- `sprintf` writes to a string
- `fprintf` prints to command window if no file is specified (more on this in next Lecture)

Printing functions

- `fprintf, sprintf`
  - `printf(file,'format1 format2…',var1, var2, …)`
  - `s = sprintf('format1 format2…',var1, var2, …)`

Format

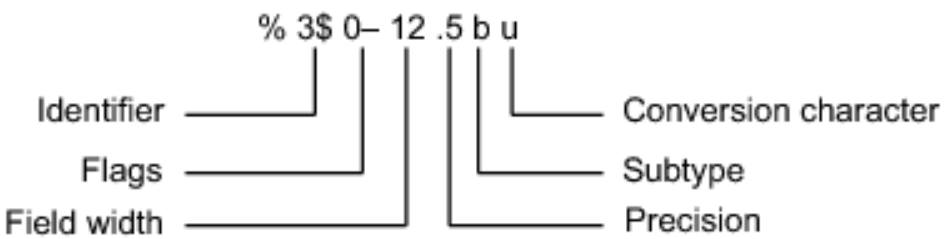


	Escape Characters
"	Single quotation mark
%%	Percent character
\\	Backslash
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\xN	Hexadecimal number, N
\N	Octal number, N

Printing functions

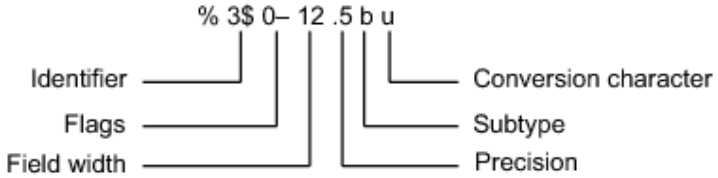
- `fprintf, sprintf`
  - `printf(file,'format1 format2…',var1, var2, …)`
  - `s = sprintf('format1 format2…',var1, var2, …)`

Format



Action	Flag	Example
Left-justify.	'–'	%-5.2f
Print sign character (+ or –).	'+'	%+5.2f
Insert a space before the value.	' '	% 5.2f
Pad with zeros.	'0'	%05.2f

Printing functions



Value Type	Conversion Character & Subtype	Details
Integer, signed	%d or %i	Base 10 values
	%ld or %li	64-bit base 10 values
Integer, unsigned	%u	Base 10
	%o	Base 8 (octal)
	%x	Base 16 (hexadecimal), lowercase letters a-f
	%X	Same as %x, uppercase letters A-F
	%lu %lo %lx or %lX	64-bit values, base 10, 8, or 16
Floating-point number	%f	Fixed-point notation
	%e	Exponential notation, such as 3.141593e+00
	%E	Same as %e, but uppercase, such as 3.141593E+00
	%g	The more compact of %e or %f, with no trailing zeros
	%G	The more compact of %E or %f, with no trailing zeros
	%bx or %bX %bo %bu	Double-precision hexadecimal, octal, or decimal value Example: %bx prints pi as 400921fb54442d18
	%tx or %tX %to %tu	Single-precision hexadecimal, octal, or decimal value Example: %tx prints pi as 40490fdb
Characters	%c	Single character
	%s	String of characters

Printing functions

- fprintf, sprintf

Example 1

```
prefix = 'C:/matlab';
num = 5;
filename = sprintf('%s/%d.txt',prefix,num)
```

filename = C:/matlab/5.txt

Example 2

```
B = [8.8  7.7; 8800  7700];
fprintf('X is %4.2f meters or %8.3f mm\n', 9.9, 9900, B)
```

X is 9.90 meters or 9900.000 mm
X is 8.80 meters or 8800.000 mm
X is 7.70 meters or 7700.000 mm

Putting it all together

```
for count=1:10
  A = rand(7, 3);
  B = rand(2, ceil(3*rand()));
  try
    res = A * B';
  catch
    disp('Loop caused an error');
    fprintf('Second dimension of B is %d\n' ,size(B, 2));
  end
end
res
```

Saving/Loading Data

MATLAB stores data in specific files, with extension *.mat*

- `save`
  - **Example**
  - `x = rand(7, 3);`
  - `y = 'cool';`
  - `save('myfile','x');`
  - `save('myfile2','x','y');`

General Form

```
save('namefile(.mat)','variable');
save('namefile(.mat)','var1','var2',…);
```

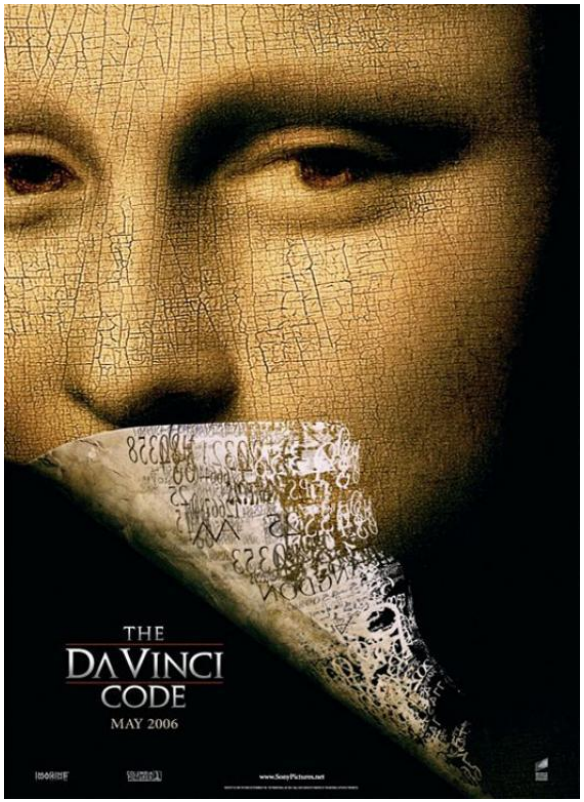
- `load`
  - **Example**
  - `load myfile2;`
  - `newX = load('myfile2','x');`

General Form

```
load 'namefile(.mat)' ;
var = load('namefile(.mat)');
```

Homeworks policy

- Due at beginning of class, no exceptions
- Put your code (.m files) and additional files in a single folder, name it *youruni*_hw_*X* and zip it
- Upload the zipped folder to CourseWorks
- Bring a printout of your code to class
- Good Luck and have fun !!!



NO CLASS NEXT WEEK, HAVE A NICE BREAK!!!