MATLAB 科学计算语言与应用

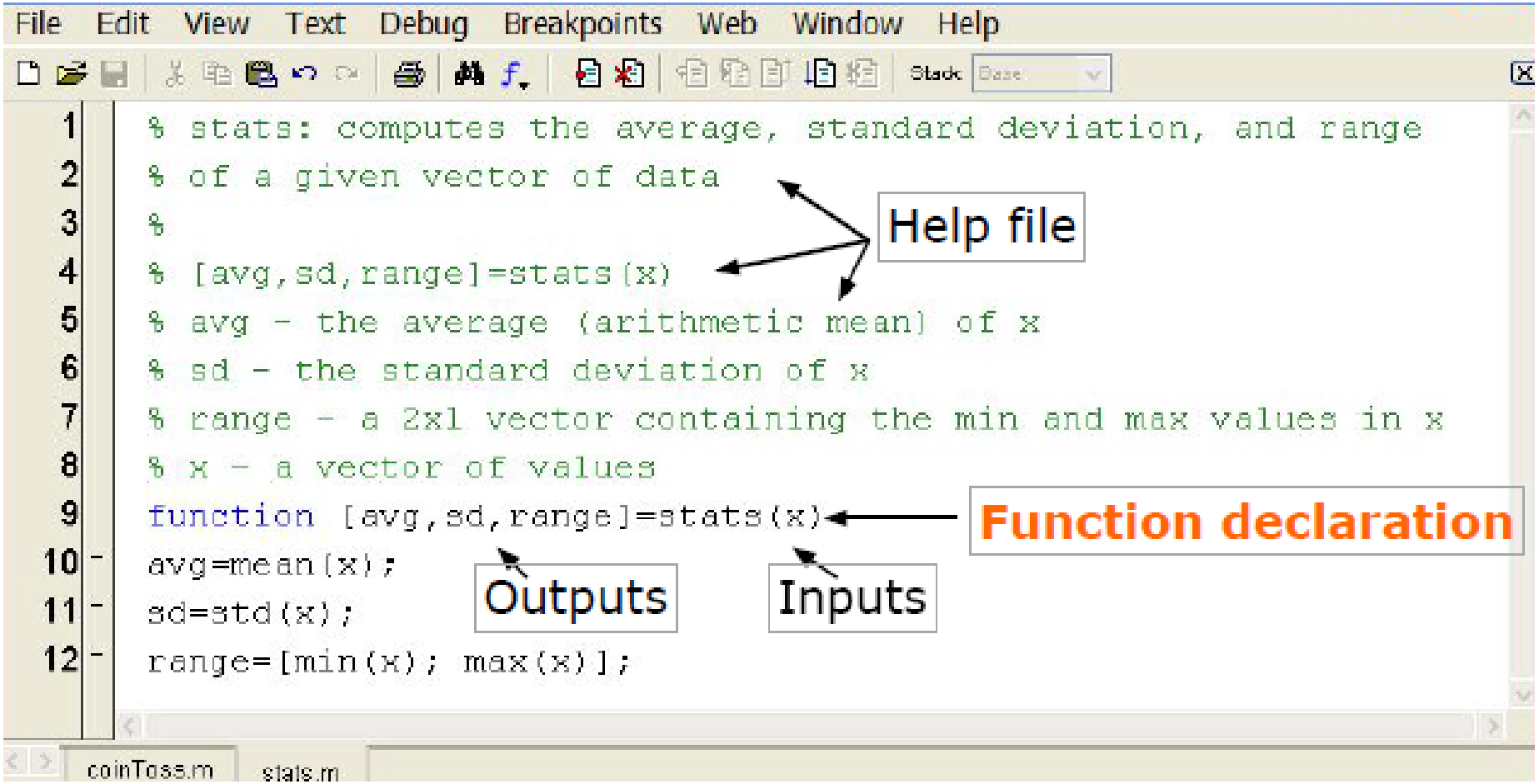
Lecture 2: Visualization and Programming

Outline

- **(1) Functions**
- (2) Flow Control
- (3) Line Plots
- (4) Image/Surface Plots
- (5) Efficient Codes
- (6) Debugging

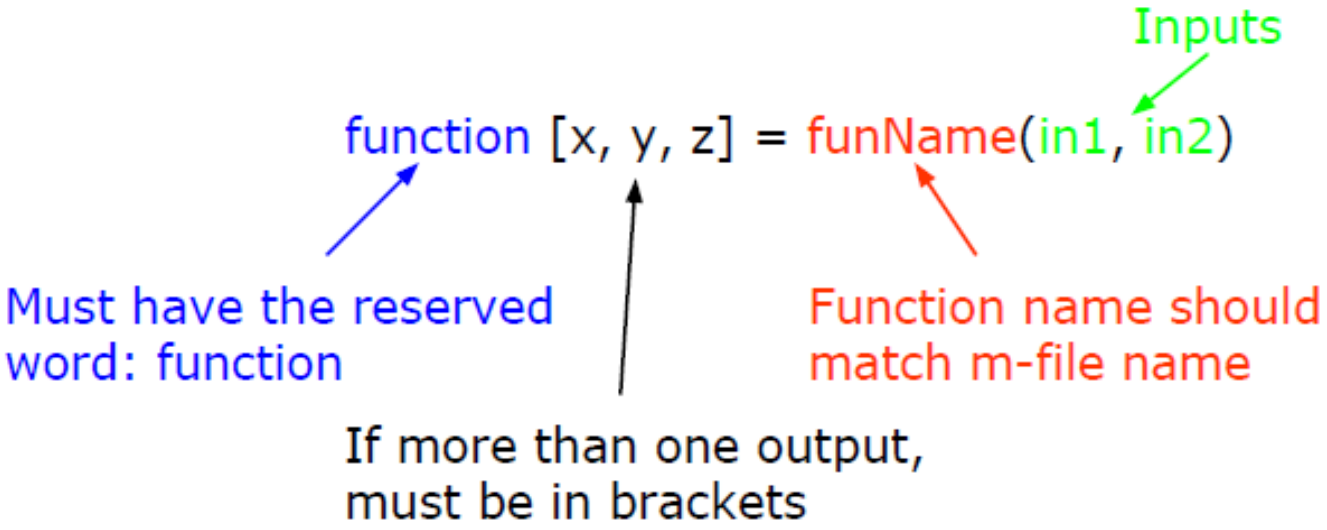
User-defined Functions

- Functions look exactly like scripts, but for **ONE difference**
  - ➢ **Functions must have a function declaration**



User-defined Functions

- Some comments about the function declaration



- No need for return: MATLAB 'returns' the variables whose names match those in the function declaration (though, you can use return to break and go back to invoking function)
- Variable scope: Any variable created within the function but not returned disappears after the function stops running(They’ re called “local variables”)

Functions: overloading

- We're familiar with
  - »zeros
  - »size
  - »length
  - »sum
- Look at the help file for size by typing
  - »help size
- Help file describes several ways to invoke the function
  - ➢ D = size(X)
  - ➢ [M,N] = size(X)
  - ➢ [M1,M2,M3,...,MN] = size(X)
  - ➢ M = size(X,DIM)

Functions: overloading

- MATLAB functions are generally overloaded
  - ➢ Can take a variable number of inputs
  - ➢ Can return a variable number of outputs
- What would the following commands return:

```
»a = zeros(2,4,8);
»D = size(a);
»[m, n] = size(a);
»[x, y, z] = size(a);
»m2 = size(a, 2);
```

- You can overload your own functions by having variable number of input and output arguments (see varargin, nargin, varargout, nargout)

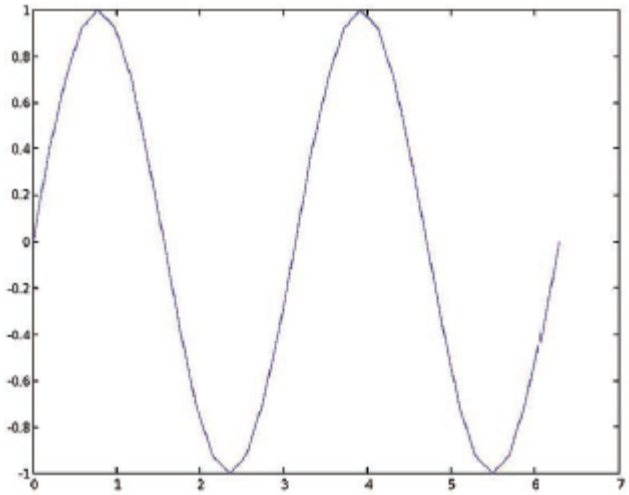
Functions: Exercise

- Write a function with the following declaration:

  function plotSin(f1)

- In the function, plot a sine wave with frequency f1, on the interval [0, 2π]: $\sin(f_1 x)$

- To get good sampling, use 16 points per period.



Note about functions in files

- Whenever possible, write your functions in their own files
  - e.g. myfun should be in a file by itself, and the file should be called myfun.m*
  - If you include more than one function per file, only the first function is accessible in other scripts
  - More info here: https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

* If filename and function name differs, MATLAB recognizes your function by its filename**, not the function name

Outline

- (1) Functions
- **(2) Flow Control**
- (3) Line Plots
- (4) Image/Surface Plots
- (5) Efficient Codes
- (6) Debugging

Relational Operators

- MATLAB uses mostly standard relational operators
  - ➢ equal **==**
  - ➢ not equal **~=**
  - ➢ greater than **>**
  - ➢ less than **<**
  - ➢ greater or equal **>=**
  - ➢ less or equal **<=**
- Logical operators element wise
  - ➢ And **&** **&&**
  - ➢ Or **|** **||**
  - ➢ Not **~**
  - ➢ Xor **xor**
  - ➢ All true **all**
  - ➢ Any true **any**
- Boolean values: zero is false, nonzero is true
- See **help** . for a detailed list of operators

if/else/elseif

- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique

IF

```
if cond
    commands
end
```

Conditional statement: evaluates to true or false

ELSE

```
if cond
    commands1
else
    commands2
end
```

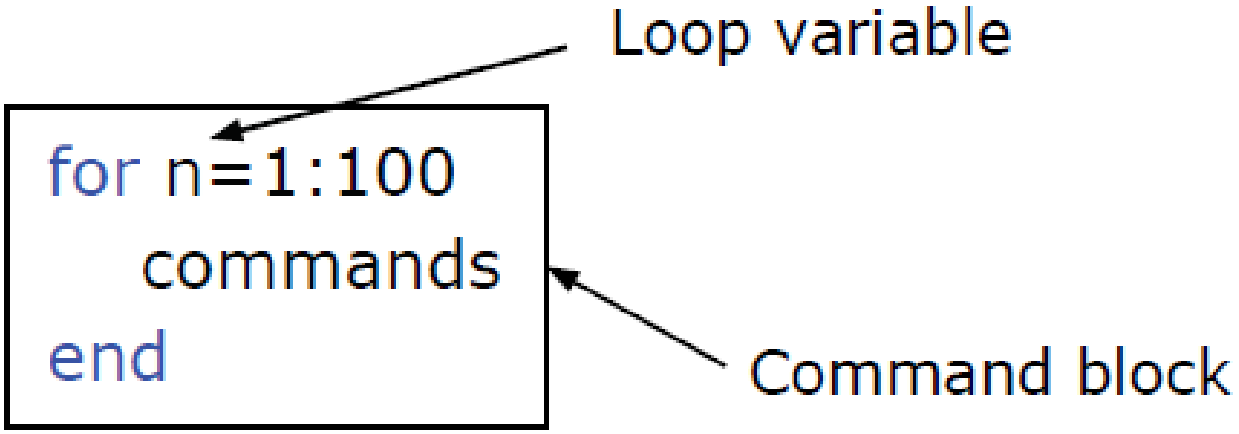
ELSEIF

```
if cond1
    commands1
elseif cond2
    commands2
else
    commands3
end
```

- No need for parentheses: command blocks are between reserved words
- Lots of **elseif**? consider using **switch-case-otherwise-end**

for

- **for** loops: use for a known number of iterations
- MATLAB syntax:



- The loop variable
  - ➢ Is defined as a vector
  - ➢ Is a scalar（标量） within the command block
  - ➢ Does not have to have consecutive values (but it's usually cleaner if they're consecutive)
- The command block
  - ➢ Anything between the **for** line and the **end**

while

The **while** is like a more general for loop:

➢ No need to know number of iterations

```
while cond
    commands
end
```

- The command block will execute while the conditional expression is true
- Beware of infinite loops! CTRL+C?!
- You can use **break** to exit a loop

Exercise: Conditionals

- Modify your **plotSin(f1)** function to take two inputs:

plotSin(f1, f2)

- If the number of input arguments is 1, execute the plot command you wrote before. Otherwise, display the line

Two inputs are given.

- Hint: the number of input arguments is stored in the built-in variable **nargin**.

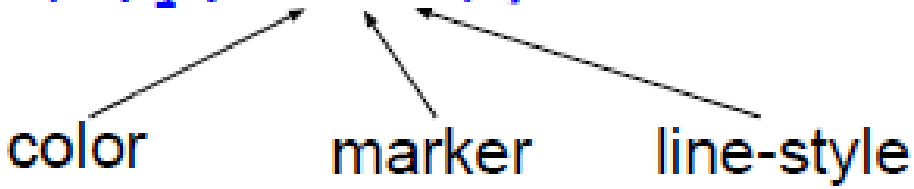
Outline

- (1) Functions
- (2) Flow Control
- **(3) Line Plots**
- (4) Image/Surface Plots
- (5) Efficient Codes
- (6) Debugging

Plot Options

- Can change the line color, marker style, and line style by adding a string argument

```
» plot(x,y,'k.-');
```

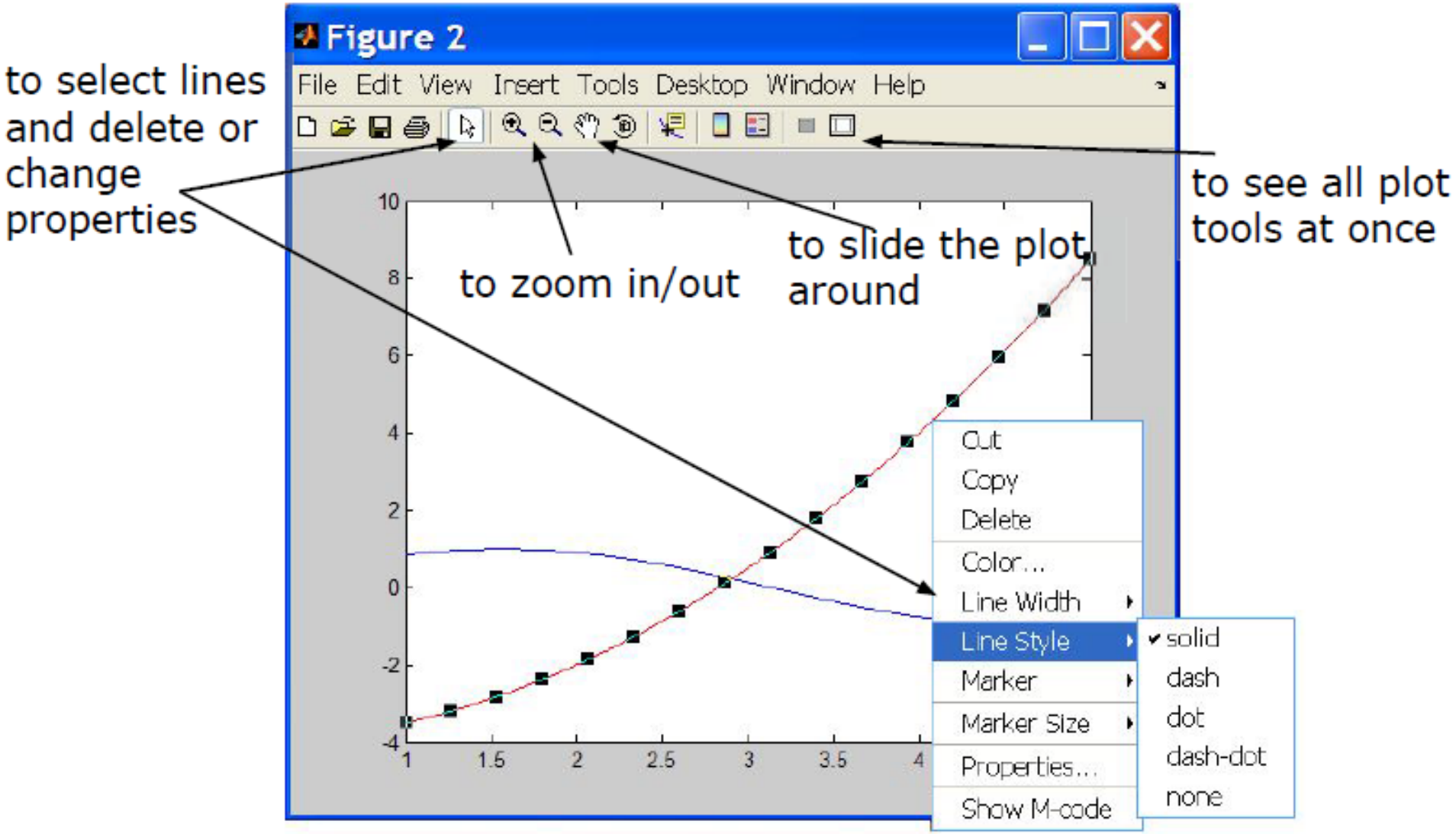


- Can plot without connecting the dots by omitting line style argument

```
» plot(x,y,'.')
```

- Look at help plot for a full list of colors, markers, and line styles

Playing with the Plot



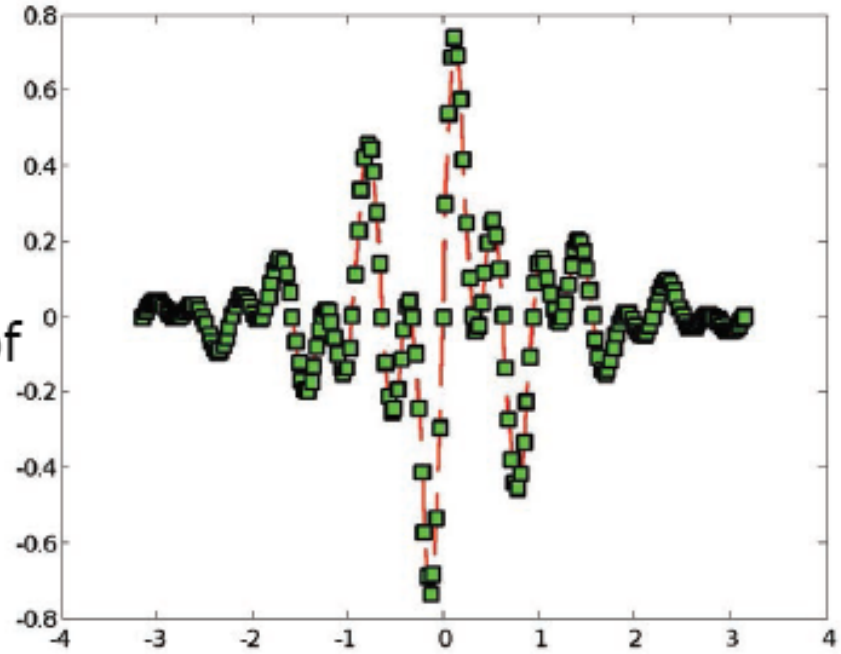
Line and Marker Options

- Everything on a line can be customized

```
» plot(x,y,'s--','LineWidth',2,...
       'Color', [1 0 0], ...
       'MarkerEdgeColor','k',...
       'MarkerFaceColor','g',...
       'MarkerSize',10)
```

You can set colors by using a vector of [R G B] values or a predefined color character like 'g', 'k', etc.

- See **doc line_props** for a full list of properties that can be specified



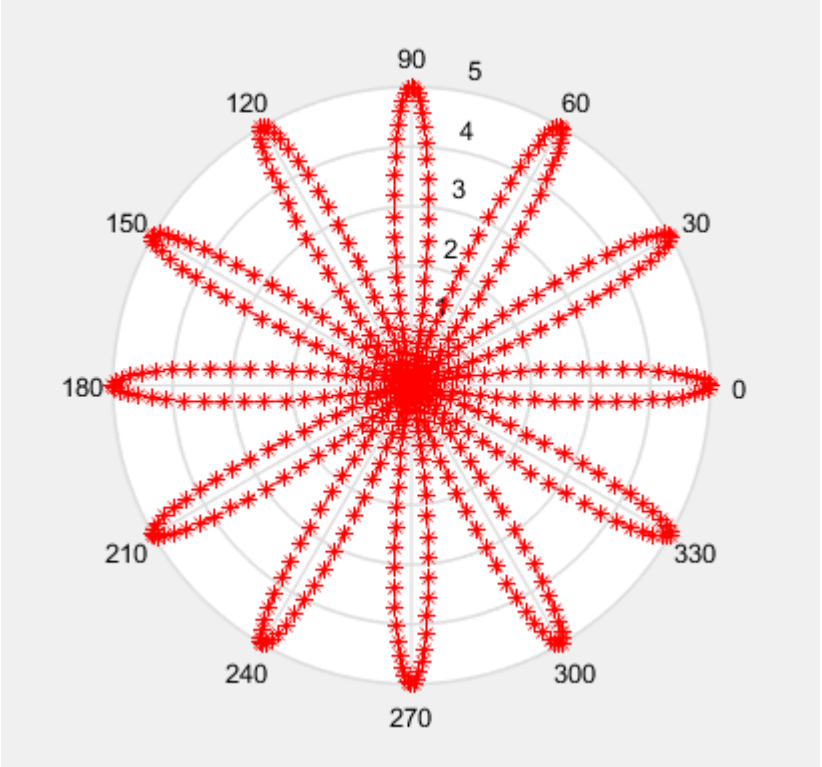
polar：极坐标绘图

polar(theta, rho, ‘参数’)

% 按相角theta（弧度）和半径rho绘制极坐标图，’参数’定义同plot

例：绘制极坐标曲线$\rho = 5cos^3(6\theta), -\pi \le \theta \le \pi$

```
>> theta=-pi:.01:pi;
>> rho = 5*cos(6*theta).^3;
>> polar(theta,rho,'--r*');
```



semilog（单轴对数）
loglog（双轴对数）

- We have already seen the plot function

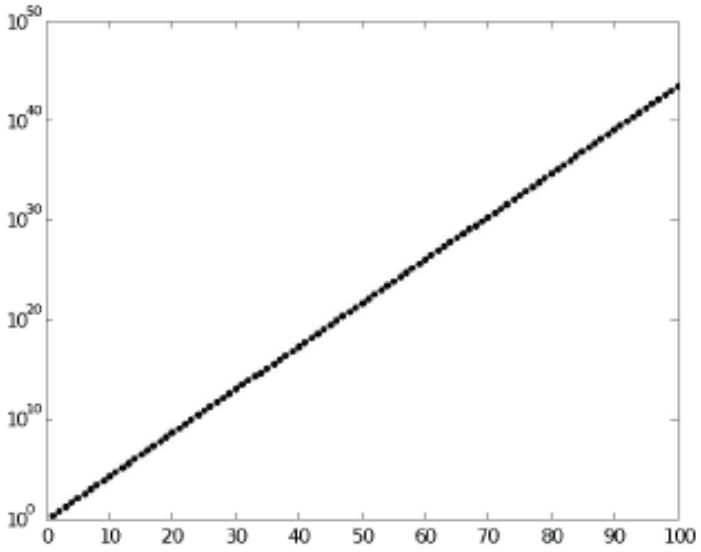
```
» x=-pi:pi/100:pi;
» y=cos(4*x).*sin(10*x).*exp(-abs(x));
» plot(x,y,'k-');
```

- The same syntax applies for semilog and loglog plots

```
» semilogx(x,y,'k');
» semilogy(y,'r.-');
» loglog(x,y);
```

- For example:

```
» x=0:100;
» semilogy(x,exp(x),'k.-');
```

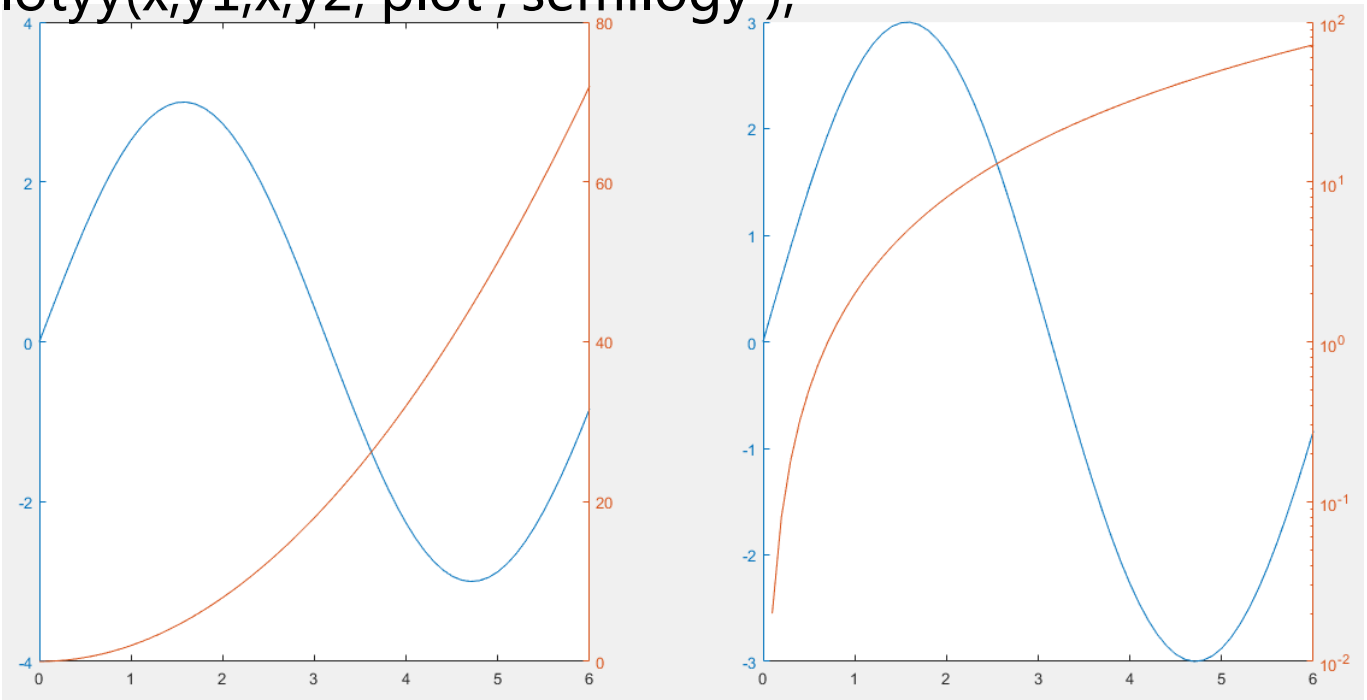


plotyy：双纵坐标绘图

Plotyy(x1, y1, x2, y2, ‘绘图方式1’, ‘绘图方式2’)

例：在同一图形中，实现两条曲线$y1 = 3\sin(x), y2 = 2x^2$的双纵坐标绘制，$0 \leq x \leq 6$

```
>> x=0:0.1:6;
>> y1=3*sin(x);
>> y2=2*x.^2;
>> subplot(1,2,1);plotyy(x,y1,x,y2);
>>subplot(1,2,2);plotyy(x,y1,x,y2,'plot','semilogy');
```

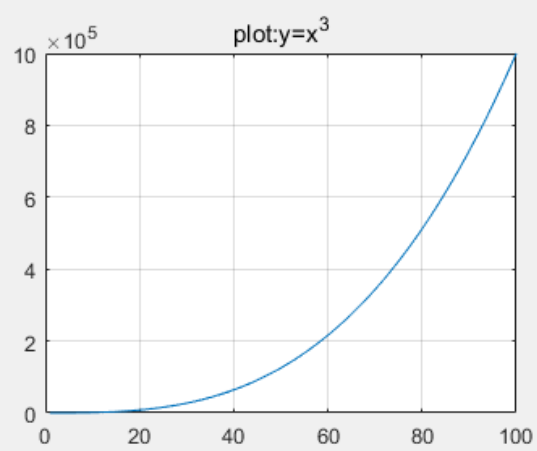


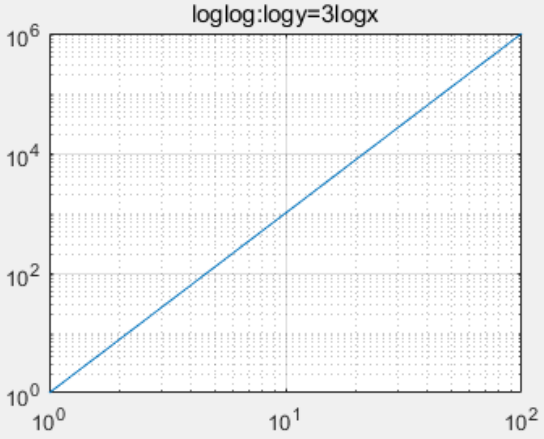
Multiple Plots in one Figure

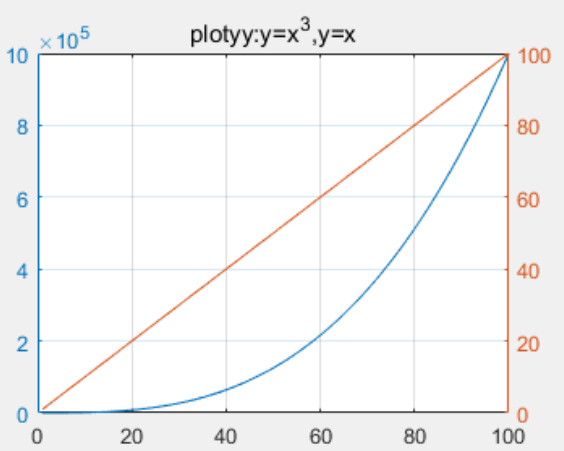
- To have multiple axes in one figure
  - » `subplot(2,3,1)`
    - ➢ makes a figure with 2 rows and 3 columns of axes, and activates the first axis for plotting
    - ➢ each axis can have labels, a legend, and a title
  - » `subplot(2,3,4:6)`
    - ➢ activates a range of axes and fuses them into one
- To close existing figures
  - » `close([1 3])`
    - ➢ closes figures 1 and 3
  - » `close all`
    - ➢ closes all figures (useful in scripts)

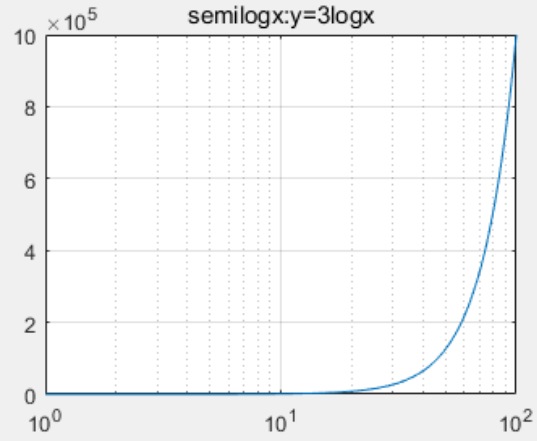
Exercise:
plot, plotyy, semilog, loglog

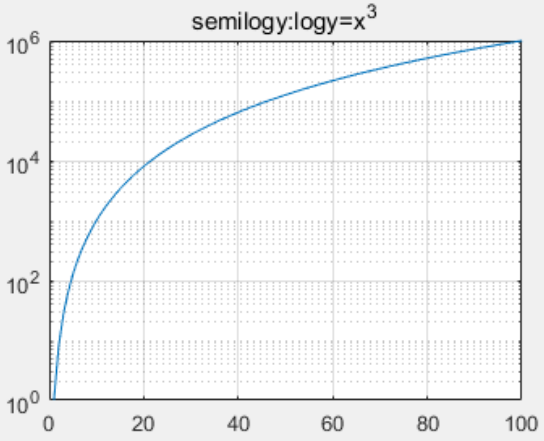
- 绘制y=x^3的函数图、对数坐标图、半对数坐标图.







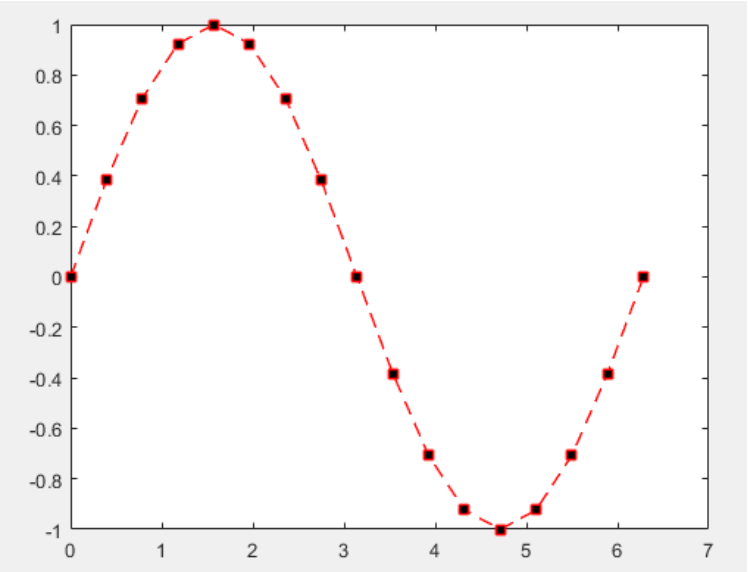


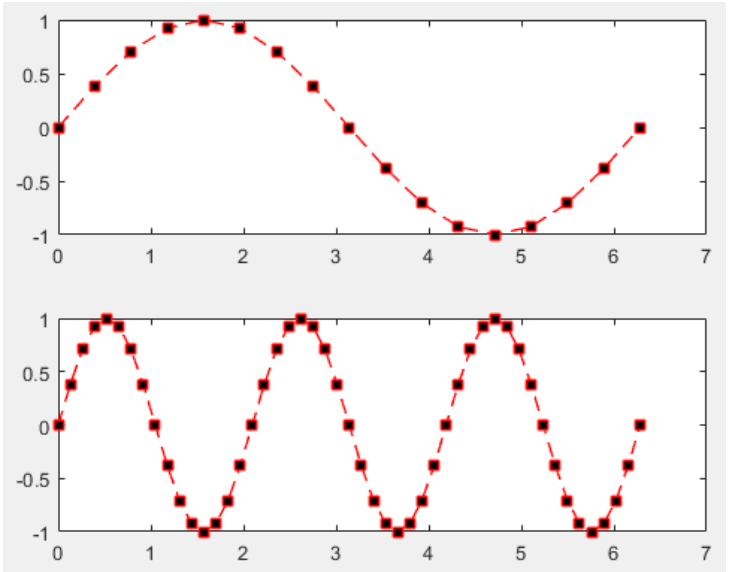


Advanced Plotting: Exercise

• Modify the plot command in your **plotSin** function to use **squares** as markers and a **dashed red line** of **thickness 2** as the line. Set the marker face color to be **black** (properties are **LineWidth, MarkerFaceColor**)

• If there are 2 inputs, open a new figure with 2 axes, one on top of the other (not side by side), and plot both frequencies (**subplot**)





Outline

- (1) Functions
- (2) Flow Control
- (3) Line Plots
- **(4) Image/Surface Plots**
- (5) Efficient Codes
- (6) Debugging

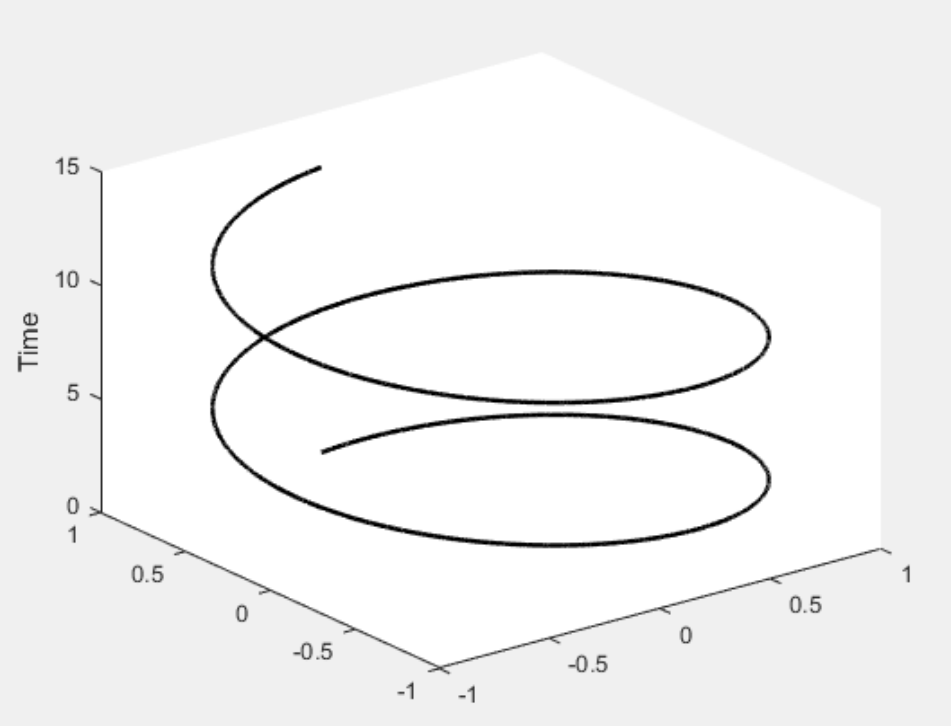
3D Line Plots

- We can plot in 3 dimensions just as easily as in 2D

```
» time=0:0.001:4*pi;
» x=sin(time);
» y=cos(time);
» z=time;
» plot3(x,y,z,'k','LineWidth',2);
» zlabel('Time');
```

- Use tools on figure to rotate it
- Can set limits on all 3 axes

```
» xlim, ylim, zlim
```



3D Surface Plots

1. 3D grid（三维网格图）

mesh(X, Y, Z, C)

X,Y为网格化的坐标矩阵，C为指定的colormap（一般略）

meshc(X, Y, Z, C)：带等高线

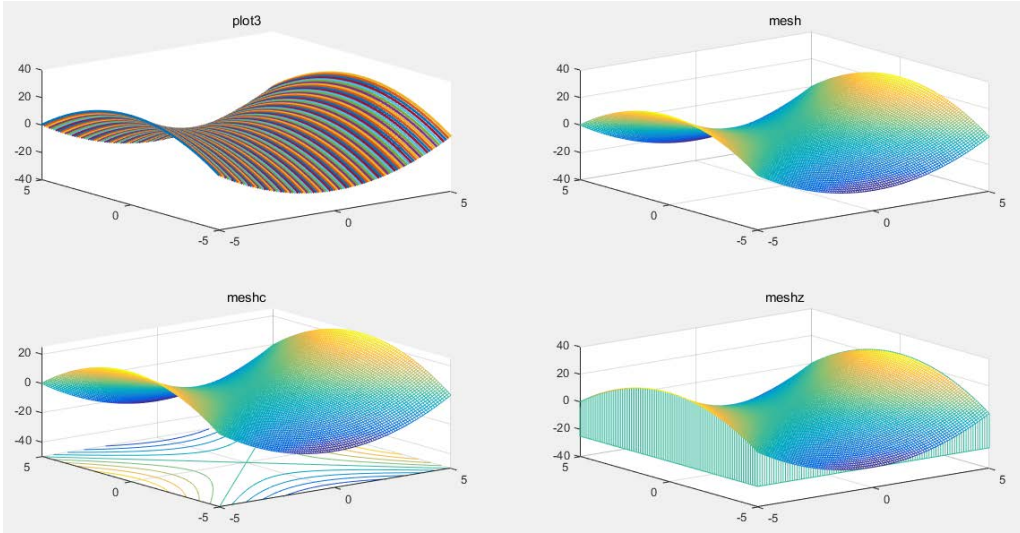
meshz(X, Y, Z, C)：带基准平面

生成网格化坐标矩阵：

[X, Y]=meshgrid(x, y);

例：已知$z = x^2 - y^2, x, y \in [-5, 5]$. 分别用 mesh, meshc meshz绘制三维曲面

```
>> x=-5:0.1:5;
>> y=-5:0.1:5;
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2-Y.^2;
```



3D Surface Plots

2. 三维表面图

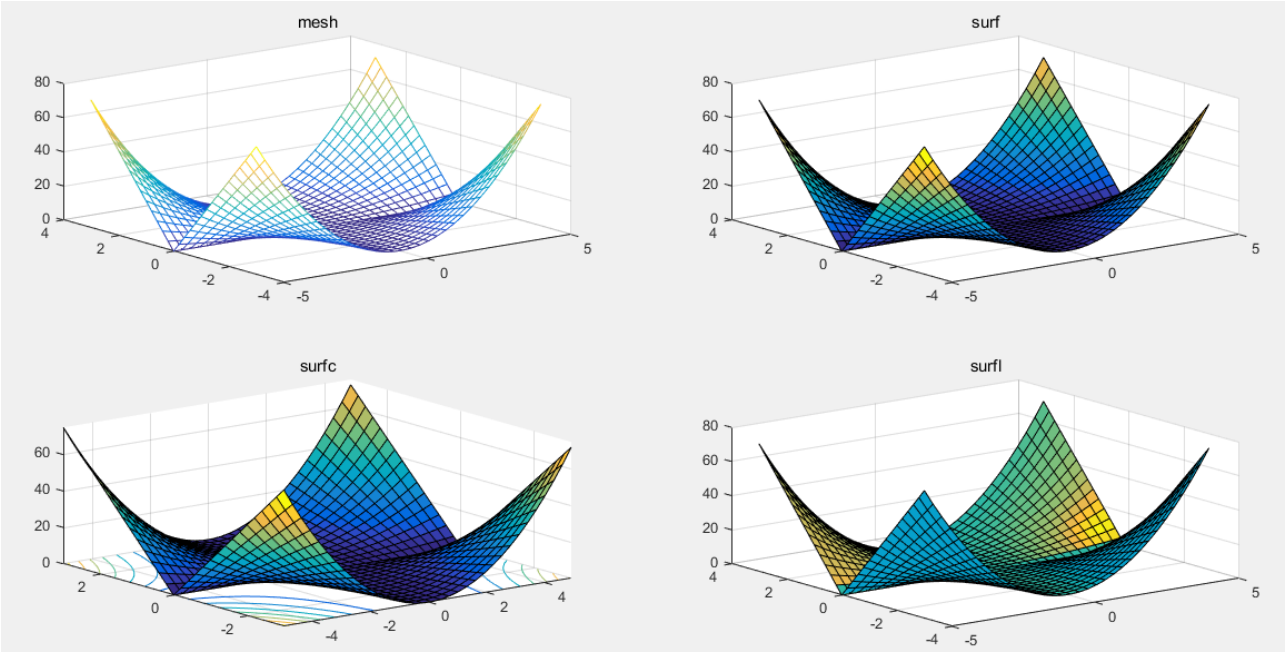
surf(X, Y, Z, C)

surfc(X, Y, Z, C) % 带等高线

surfl(X, Y, Z, C) % 带打光效果

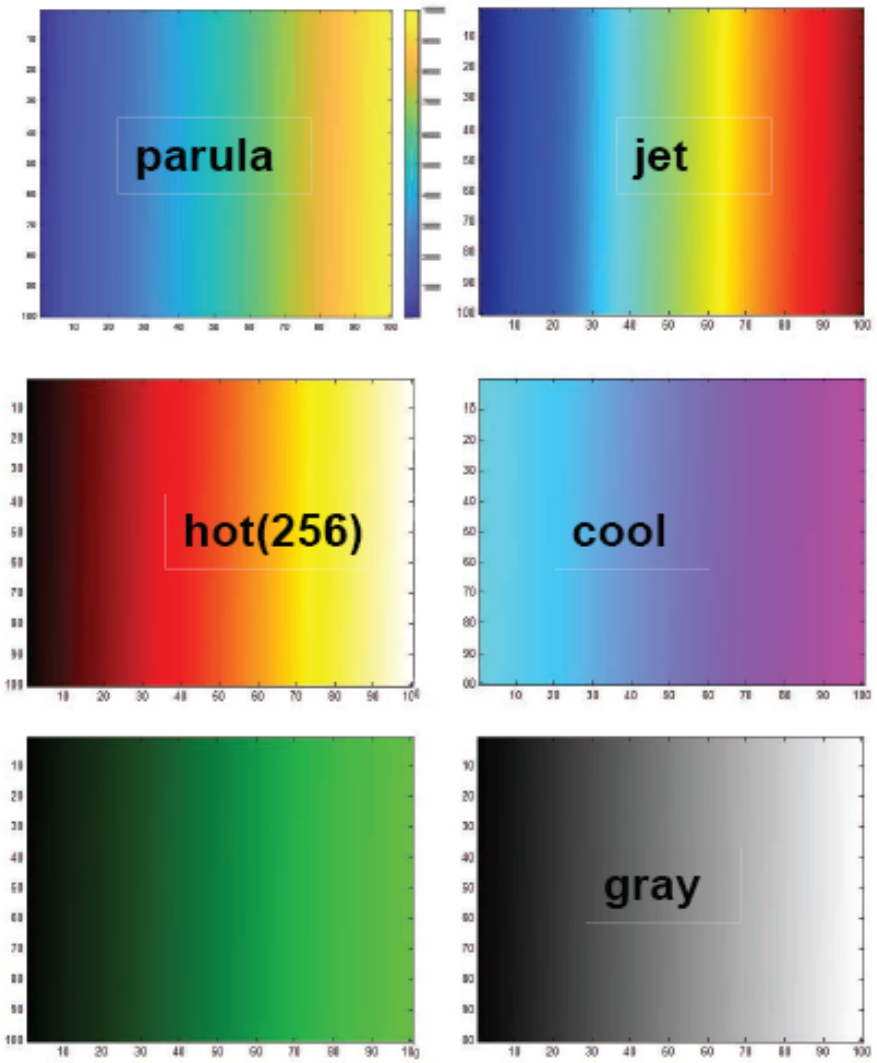
例：在$x \in [-5,5], y \in [-3,3]$上做出$z^2 = x^4 y^2$所对应的三维表面图

```
>> x=-5:0.3:5;
>> y=-3:0.2:3;
```



Colormaps

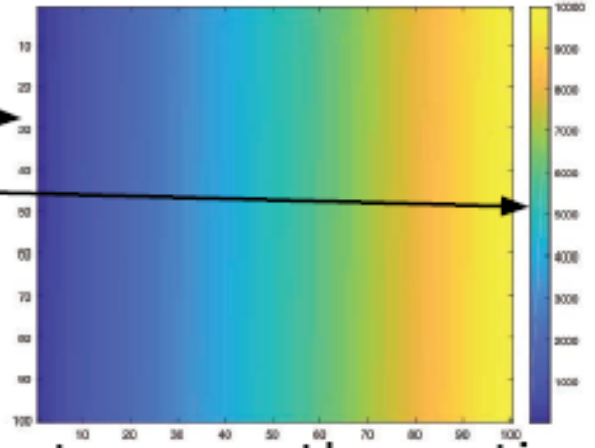
- You can change the colormap:
  - » `imagesc(mat)`
    - ➢ default map is parula
  - » `colormap(gray)`
  - » `colormap(cool)`
  - » `colormap(hot(256))`
- See `help hot` for a list
- Can define custom color-map
  - » `map=zeros(256,3);`
  - » `map(:,2)=(0:255)/255;`
  - » `colormap(map);`



Visualizing matrices

- Any matrix can be visualized as an image

```
» mat=reshape(1:10000,100,100);
» imagesc(mat);
» colorbar
```



- **imagesc** automatically scales the values to span the entire colormap

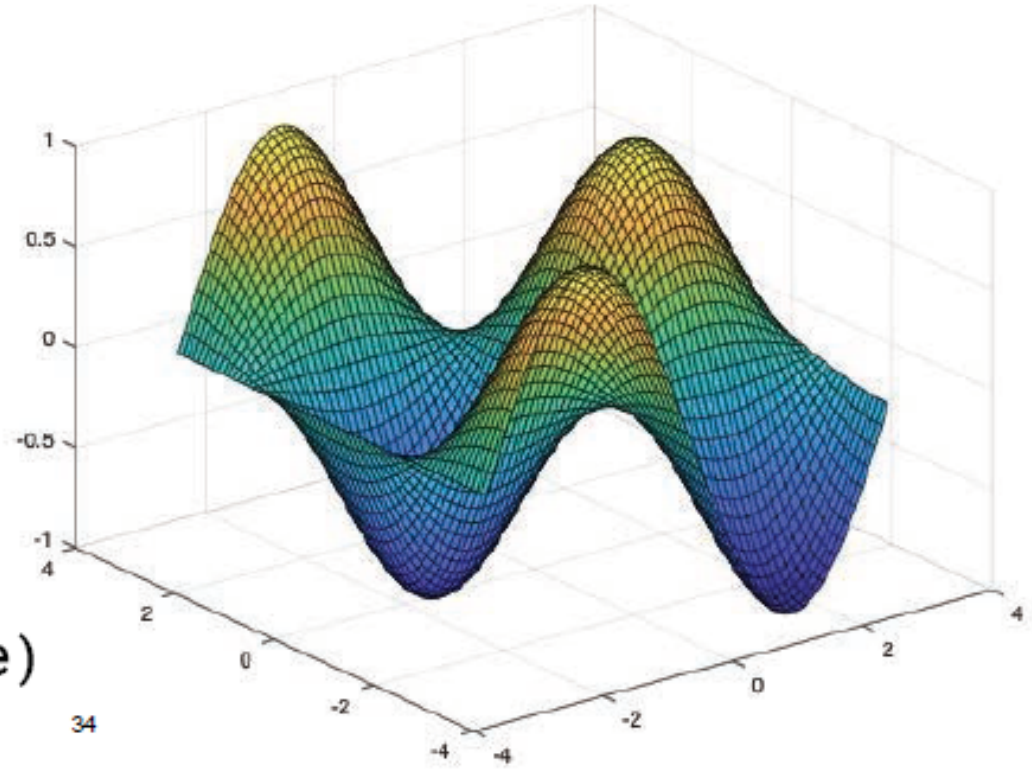
- Can set limits for the color axis (analogous to `xlim`, `ylim`)

```
» caxis([3000 7000])
```

surf

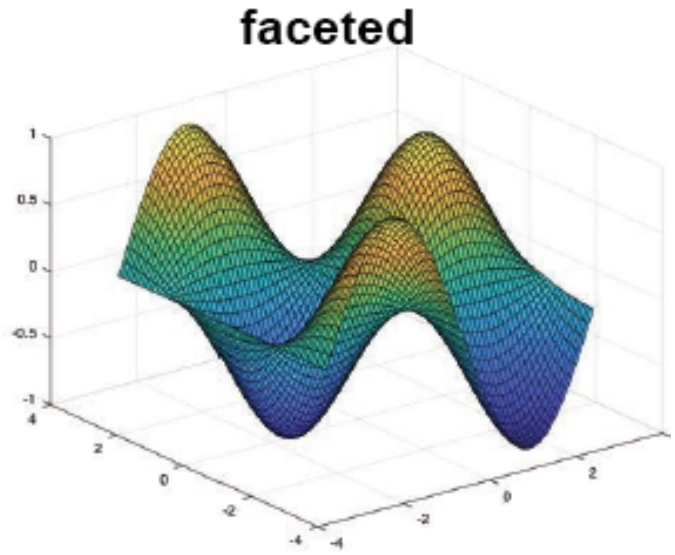
- Make the x and y vectors
  - » `x=-pi:0.1:pi;`
  - » `y=-pi:0.1:pi;`
- Use meshgrid to make matrices
  - » `[X,Y]=meshgrid(x,y);`
- To get function values, evaluate the matrices
  - » `Z =sin(X).*cos(Y);`
- Plot the surface
  - » `surf(X,Y,Z)`
  - » `surf(x,y,Z);`

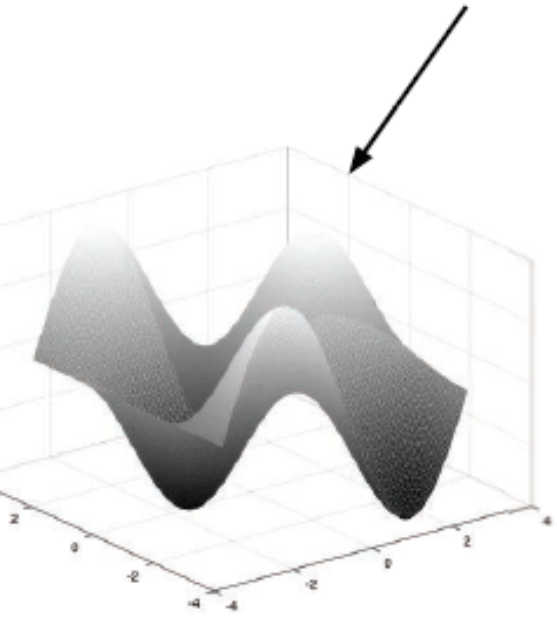
*Try typing `surf(membrane)`

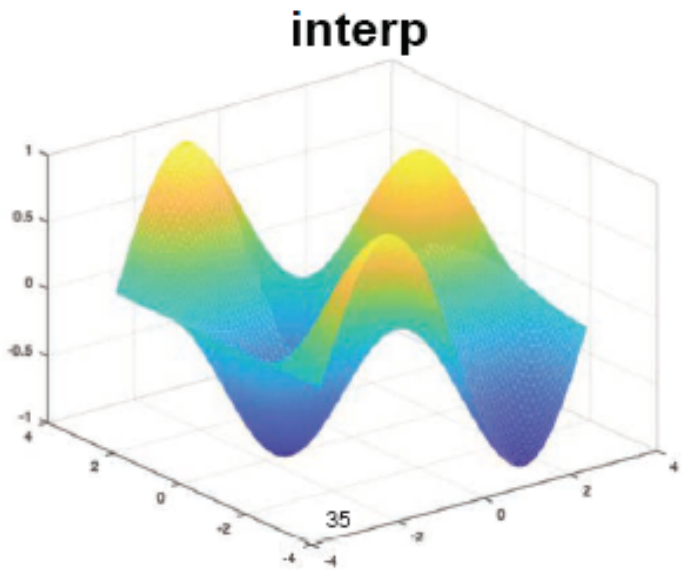


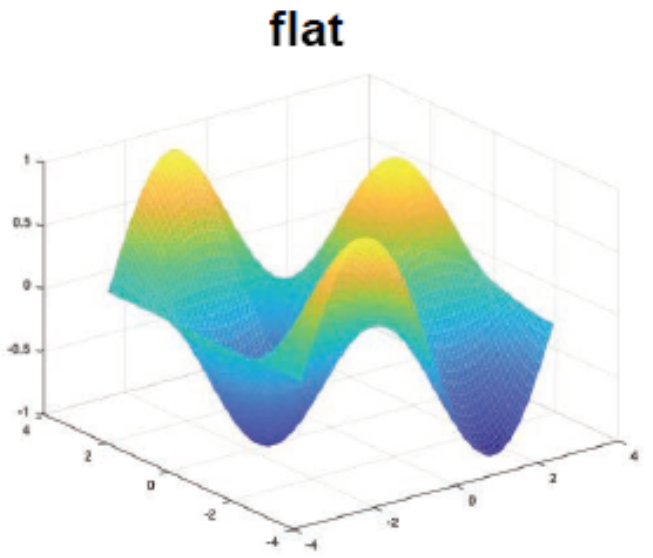
surf Options

- See **help surf** for more options
- There are three types of surface shading
  - » `shading faceted`
  - » `shading flat`
  - » `shading interp`
- You can also change the colormap
  - » `colormap(gray)`



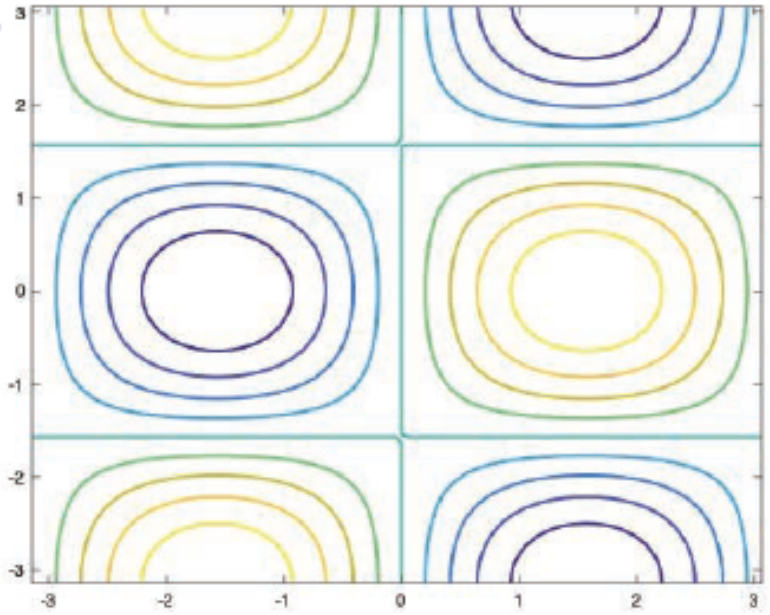


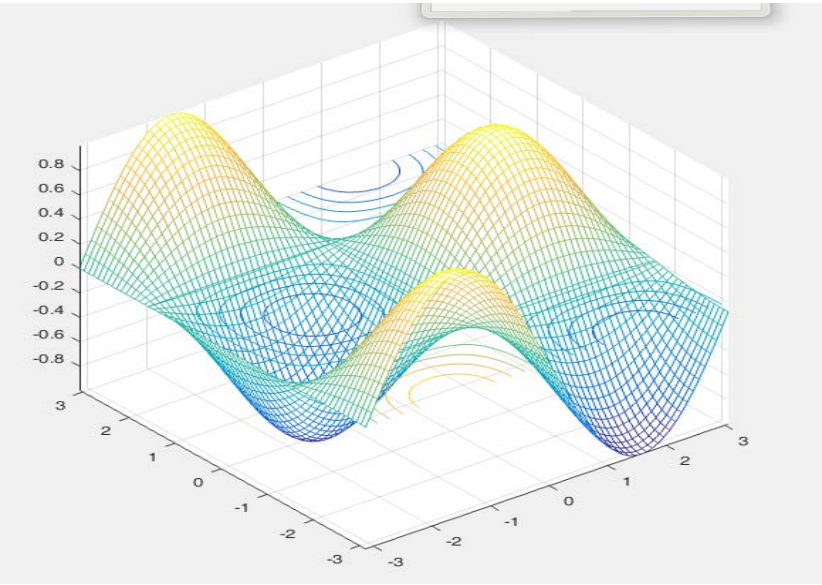




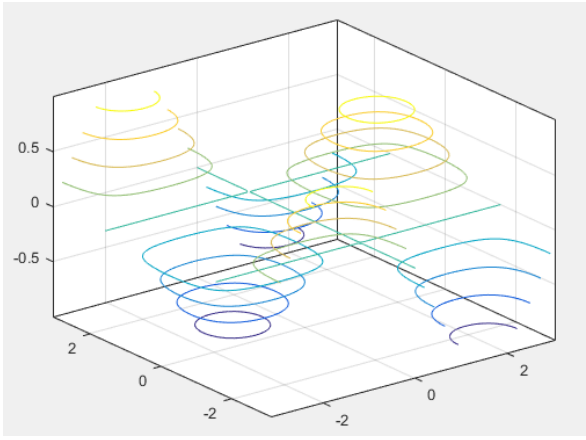
Contour（等高线）

- You can make surfaces two-dimensional by using contour
  - » `contour(X,Y,Z,'LineWidth',2)`
    - ➢ takes same arguments as surf
    - ➢ color indicates height
    - ➢ can modify linestyle properties
    - ➢ can set colormap
  - » `hold on`
  - » `mesh(X,Y,Z)`





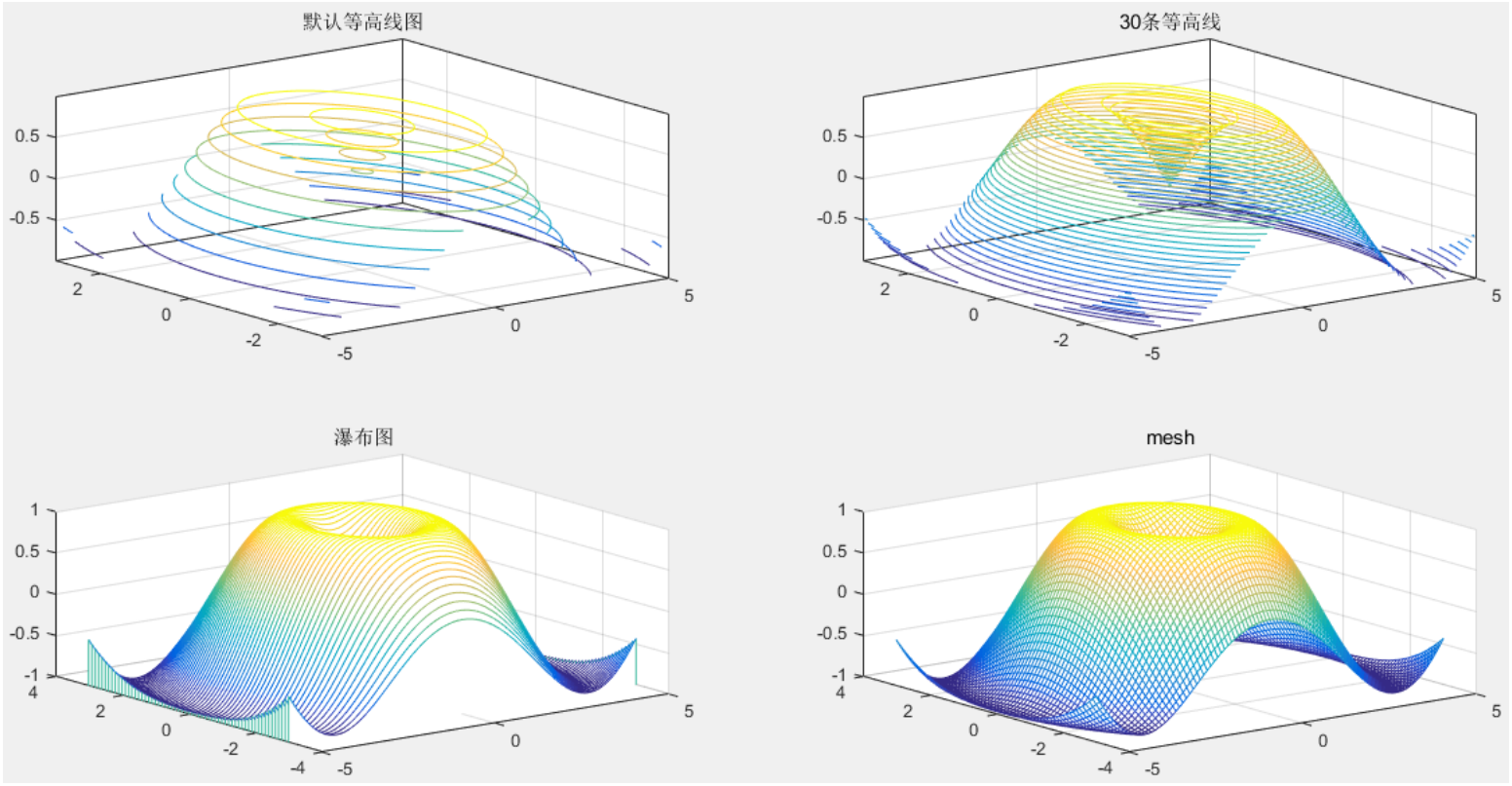
3D contour: **contour3(X, Y, Z, n)**



瀑布图

waterfall(X, Y, Z)

例：在$x \in [-5,5], y \in [-3,3]$上做出$z = sin\sqrt{x^2 + y^2}$所对应的等高线图、瀑布图和三维网格图



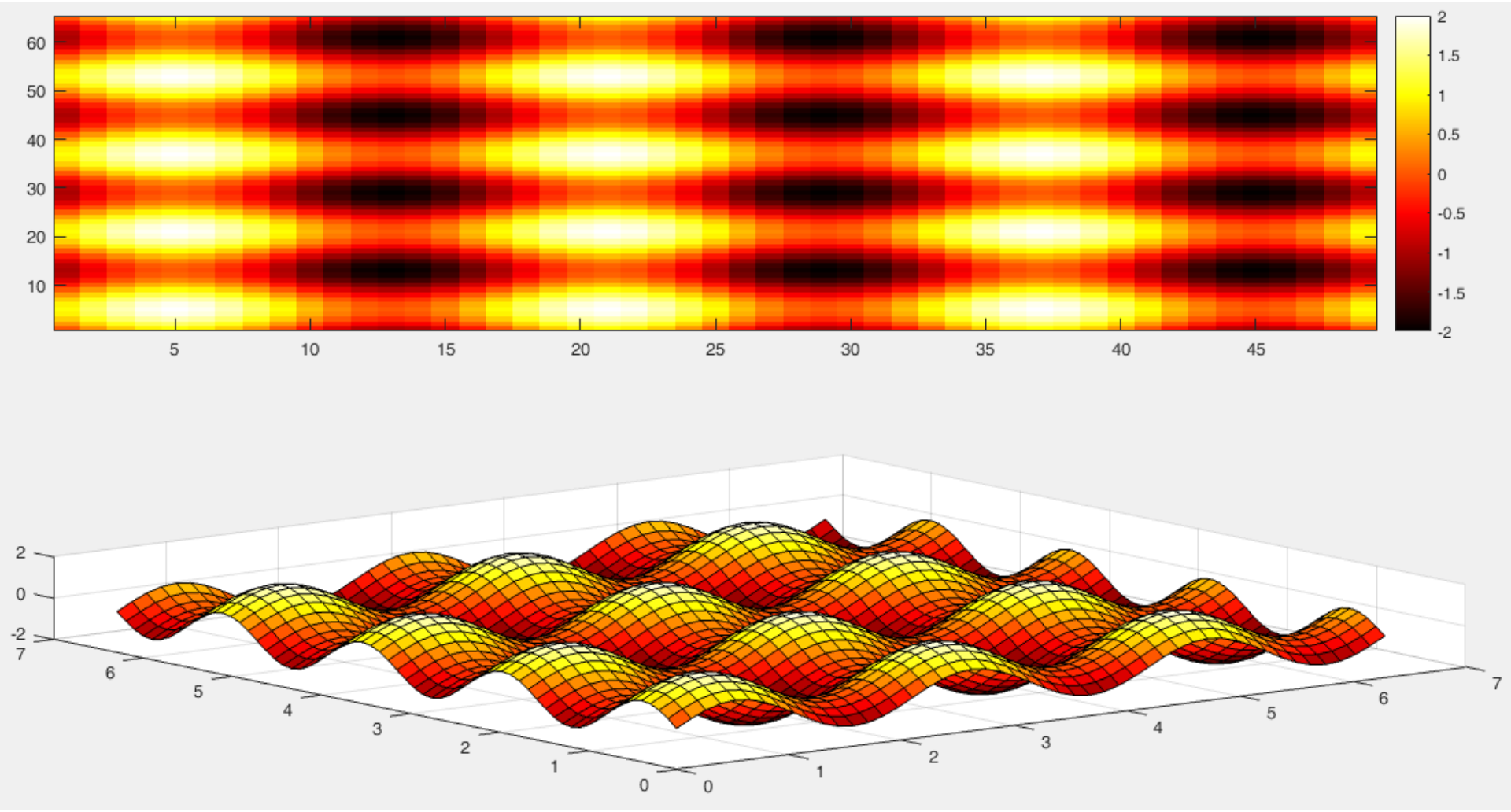
Exercise: 3-D Plots

- Modify `plotSin` to do the following:
- If two inputs are given, evaluate the following function:

$$Z = \sin(f_1 x) + \sin(f_2 y)$$

- y should be just like x, but using f2. (use `meshgrid` to get the X and Y matrices)
- In the top axis of your subplot, display an image of the Z matrix. Display the colorbar and use a `hot` colormap. Set the axis to xy (`imagesc`, `colormap`, `colorbar`, `axis`)
- In the bottom axis of the subplot, plot the 3-D surface of Z (`surf`)

plotSin2_3D(3,4)



绘制动画

M = movie(n) %预先分配一个能存储n帧的矩阵

M(i) = getframe %录制动画的每一帧

movie(M, k) %播放动画

例：矩形函数的傅里叶变换是辛格函数sinc, sinc(r)=sin(r)/r，r为X-Y平面的向径。制作sinc函数的立体图并播放动画效果。

```
x=-9:0.1:9;
y=x;
[X,Y]=meshgrid(x,y);
R=sqrt(X.^2+Y.^2);
Z=sin(R)./R;
h=surfc(X,Y,Z);
M=moviein(20);
for i=1:20
    rotate(h,[0 0 1],15);
    M(i)=getframe;
end
movie(M,10,6);
```

Outline

- (1) Functions
- (2) Flow Control
- (3) Line Plots
- (4) Image/Surface Plots
- **(5) Efficient Codes**
- (6) Debugging

find

- **find** is a very important function
  - ➢ Returns indices of nonzero values
  - ➢ Can simplify code and help avoid loops

- Basic syntax: index=find(cond)

```
» x=rand(1,100);
» inds = find(x>0.4 & x<0.6);
```

`inds` contains the indices at which x has values between 0.4 and 0.6. This is what happens:

x>0.4 returns a vector with 1 where true and 0 where false

x<0.6 returns a similar vector

& combines the two vectors using logical **and** operator

find returns the indices of the 1's

Example: Avoiding Loops

- Given x= sin(linspace(0,10*pi,100)), how many of the entries are positive?

Using a loop and if/else

```
count=0;
for n=1:length(x)
    if x(n)>0
        count=count+1;
    end
end
```

Being more clever

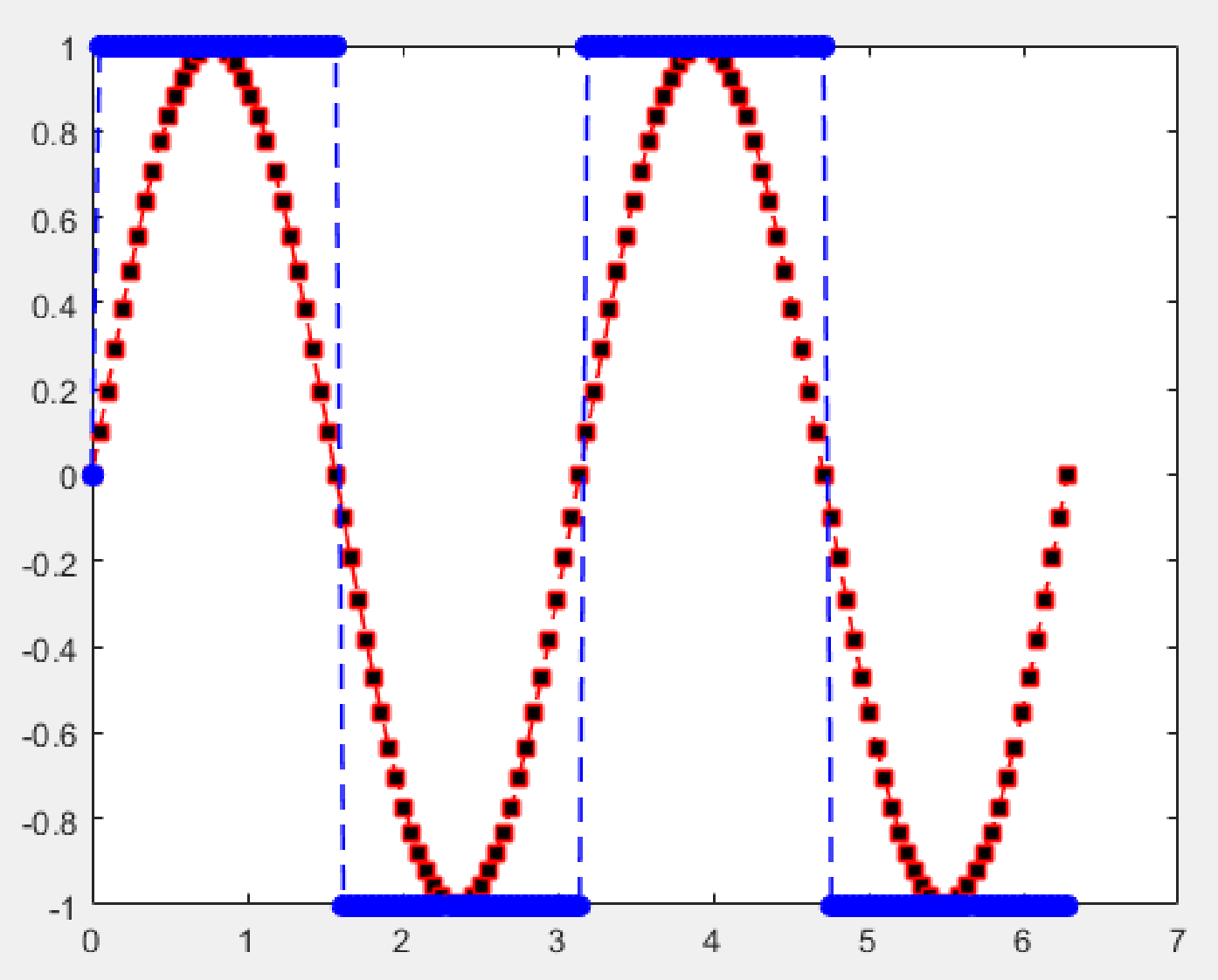
```
count=length(find(x>0));
```

Is there a better way?!

length(x)	Loop time	Find time
100	0.01	0
10,000	0.1	0
100,000	0.22	0
1,000,000	1.5	0.04

- Avoid loops!
- Built-in functions will make it faster to write and execute

画图练习：在正弦波上叠加一个方波



Vectorization

- Avoid loops
  - ➢ This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For instance, to add every two consecutive terms:

```
» a=rand(1,100);
» b=zeros(1,100);
» for n=1:100
»     if n==1
»         b(n)=a(n);
»     else
»         b(n)=a(n-1)+a(n);
»     end
» end
```

➢ Slow and complicated

```
» a=rand(1,100);
» b=[0 a(1:end-1)]+a;
```

➢ Efficient and clean.

Preallocation

- Avoid variables growing inside a loop
- Re-allocation of memory is time consuming
- Preallocate the required memory by initializing the array to a default value
- For example:

```
» for n=1:100
»     res = % Very complex calculation %
»     a(n) = res;
» end
```

  - ➢ Variable `a` needs to be resized at every loop iteration

Preallocation

- Avoid variables growing inside a loop
- Re-allocation of memory is time consuming
- Preallocate the required memory by initializing the array to a default value
- For example:

```
» a = zeros(1, 100);
» for n=1:100
»     res = % Very complex calculation %
»     a(n) = res;
» end
```

  - ➢ Variable `a` is only assigned new values. No new memory is allocated

Outline

- (1) Functions
- (2) Flow Control
- (3) Line Plots
- (4) Image/Surface Plots
- (5) Efficient Codes
- **(6) Debugging**

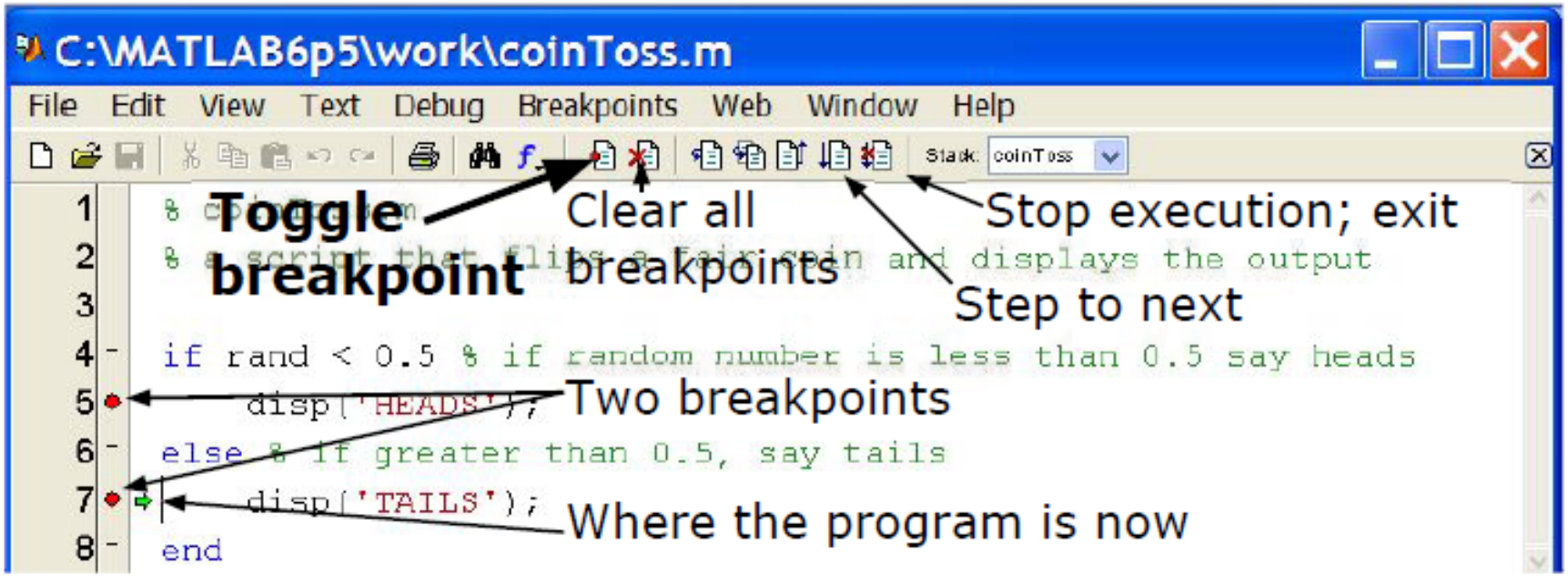
Display

- When debugging functions, use **disp** to print messages
  - `» disp('starting loop')`
  - `» disp('loop is over')`
    - ➢ `disp` prints the given string to the command window

- It's also helpful to show variable values
  - `» disp(['loop iteration ' num2str(n)]);`

Debugging

- To use the debugger, set breakpoints
  - Click on – next to line numbers in m-files
  - Each red dot that appears is a breakpoint
  - Run the program
  - The program pauses when it reaches a breakpoint
  - Use the command window to probe variables
  - Use the debugging buttons to control debugger



Performance Measures

- It can be useful to know how long your code takes to run
  - ➢ To predict how long a loop will take
  - ➢ To pinpoint inefficient code

- You can time operations using **tic**/**toc**:

```
» tic
» Mystery1;
» a=toc;
» Mystery2;
» b=toc;
```

  - ➢ tic resets the timer
  - ➢ Each toc returns the current value in seconds
  - ➢ Can have multiple tocs per tic

Performance Measures

- Example: Sparse matrices

```
» A=zeros(10000); A(1,3)=10; A(21,5)=pi;
» B=sparse(A);
» inv(A);   % what happens?
» inv(B);   % what about now?
```

- If system is sparse, can lead to large memory/time savings

```
» A=zeros(1000); A(1,3)=10; A(21,5)=pi;
» B=sparse(A);
» C=rand(1000,1);
» tic; A\C; toc;   % slow!
» tic; B\C; toc;   % much faster!
```

Performance Measures

- For more complicated programs, use the profiler
  - » `profile on`
    - ➢ Turns on the profiler. Follow this with function calls
  - » `profile viewer`
    - ➢ Displays gui with stats on how long each subfunction took

函数名称	调用	总时间	自用时间*	总时间图 (深色条带 = 自用时间)
plotSin2_3D	1	1.583 s	0.007 s	
subplot	2	0.869 s	0.718 s	
colorbar	1	0.523 s	0.009 s	

函数名；
函数被调用次数；
函数运行总时间（包含子函数）；
函数自身运行时间（不包含子函数）；
运行总时间图（函数运行总时间相对于整个时间）。

Profile 作用:

- 找出没有实际运行的代码；
- 找出耗时较长的代码