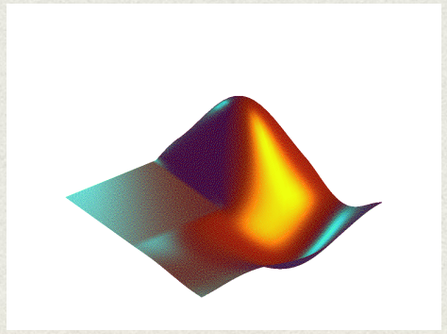
MATLAB程式語言在工作站計算環境平台上的使用



黃建智

國立台灣師範大學 數學系

2014.8.26~28 @ NCTU

課程內容介紹

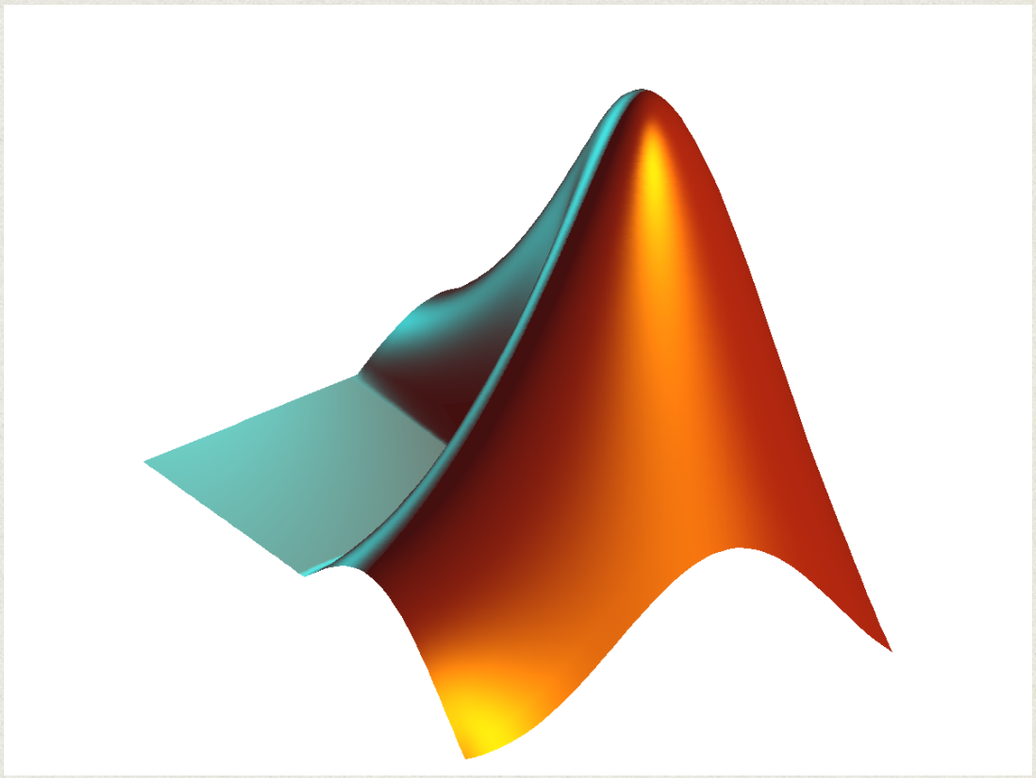
- 8/26：matlab基本功能介紹：

1.基本環境、功能簡介

2.在工作站如何使用matlab

3.基本繪圖指令介紹

4.實機演練與討論



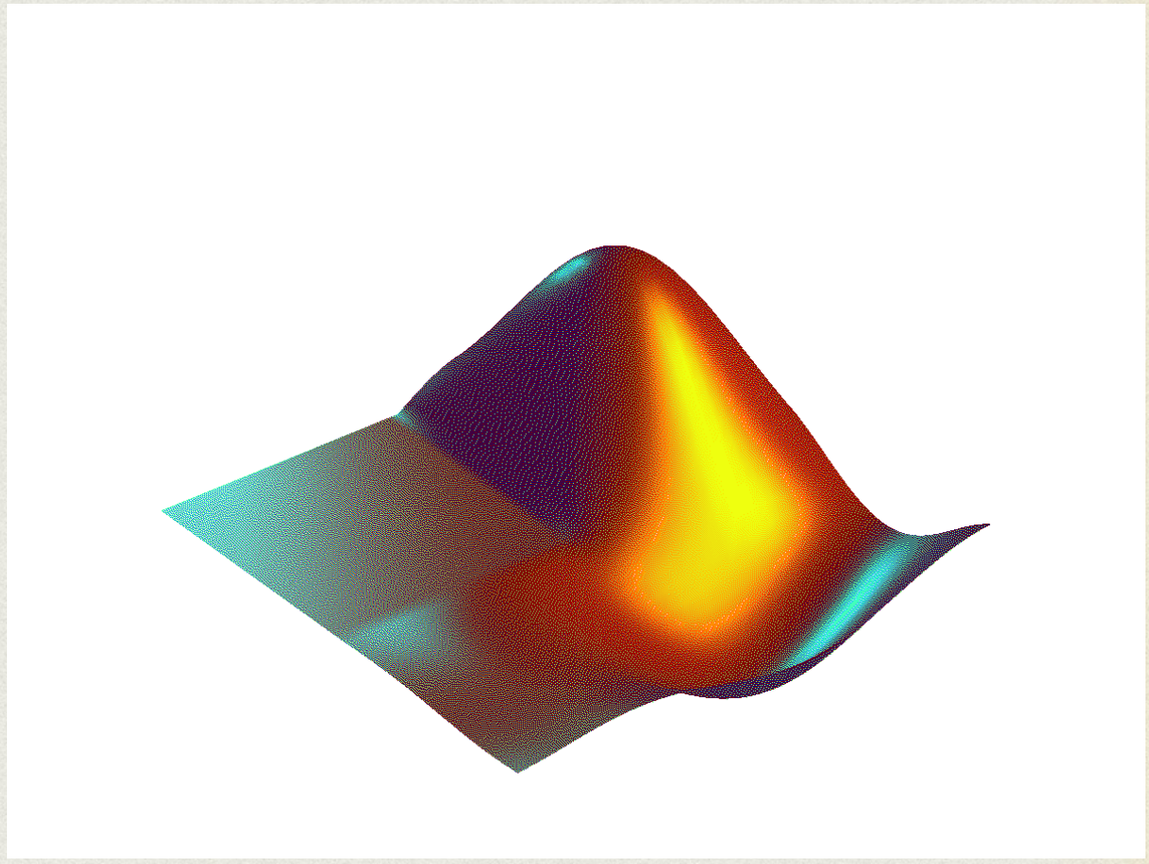
課程內容介紹

- 8/27：matlab進階功能介紹

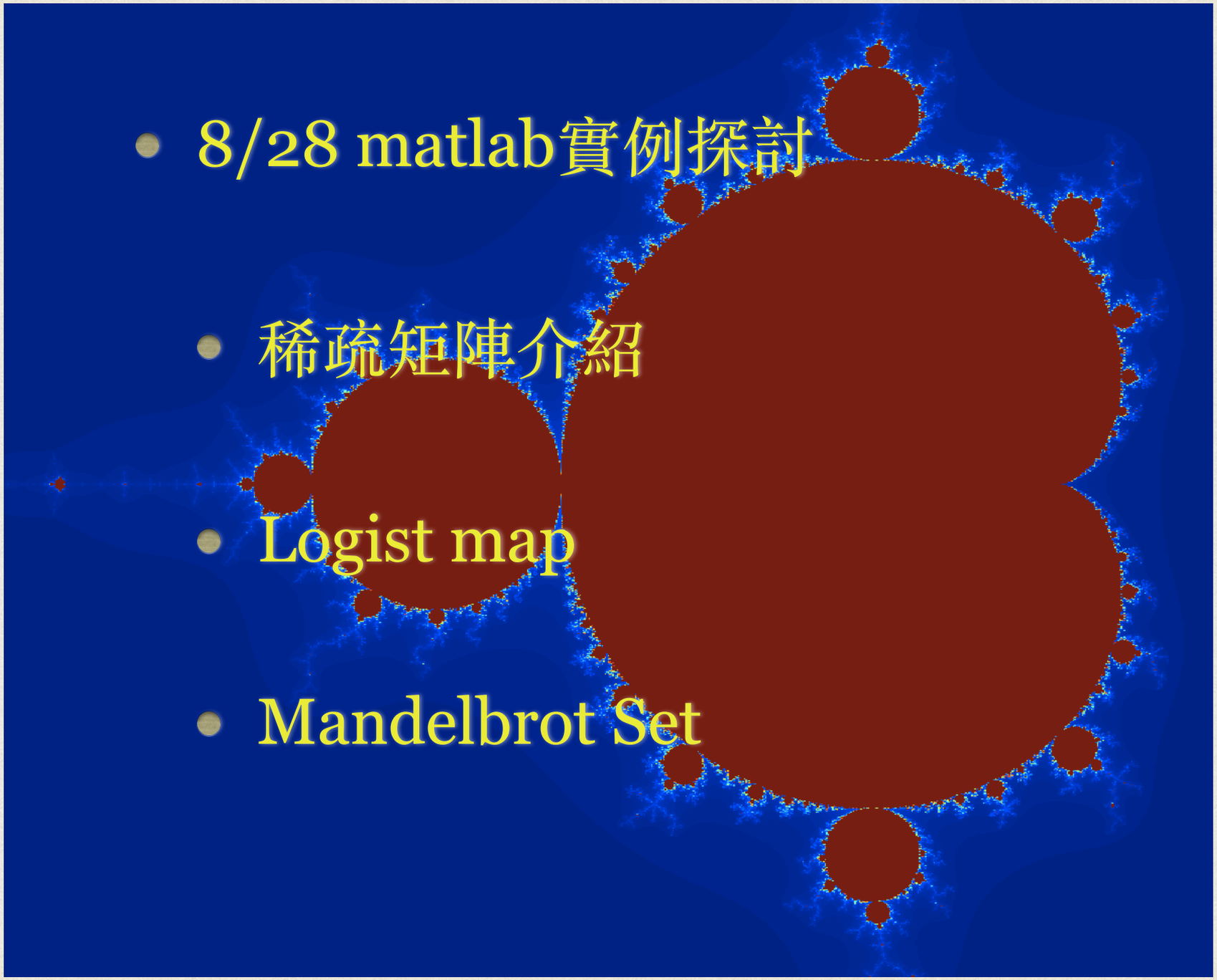
1.如何Debug與改進執行效能

2.動畫制作與GUI程式介紹

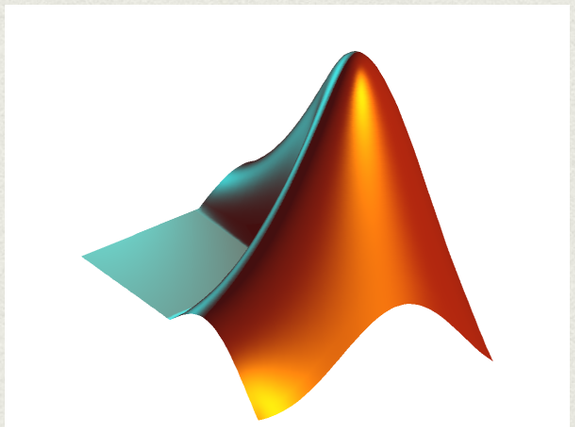
3.實機演練與討論



課程內容介紹



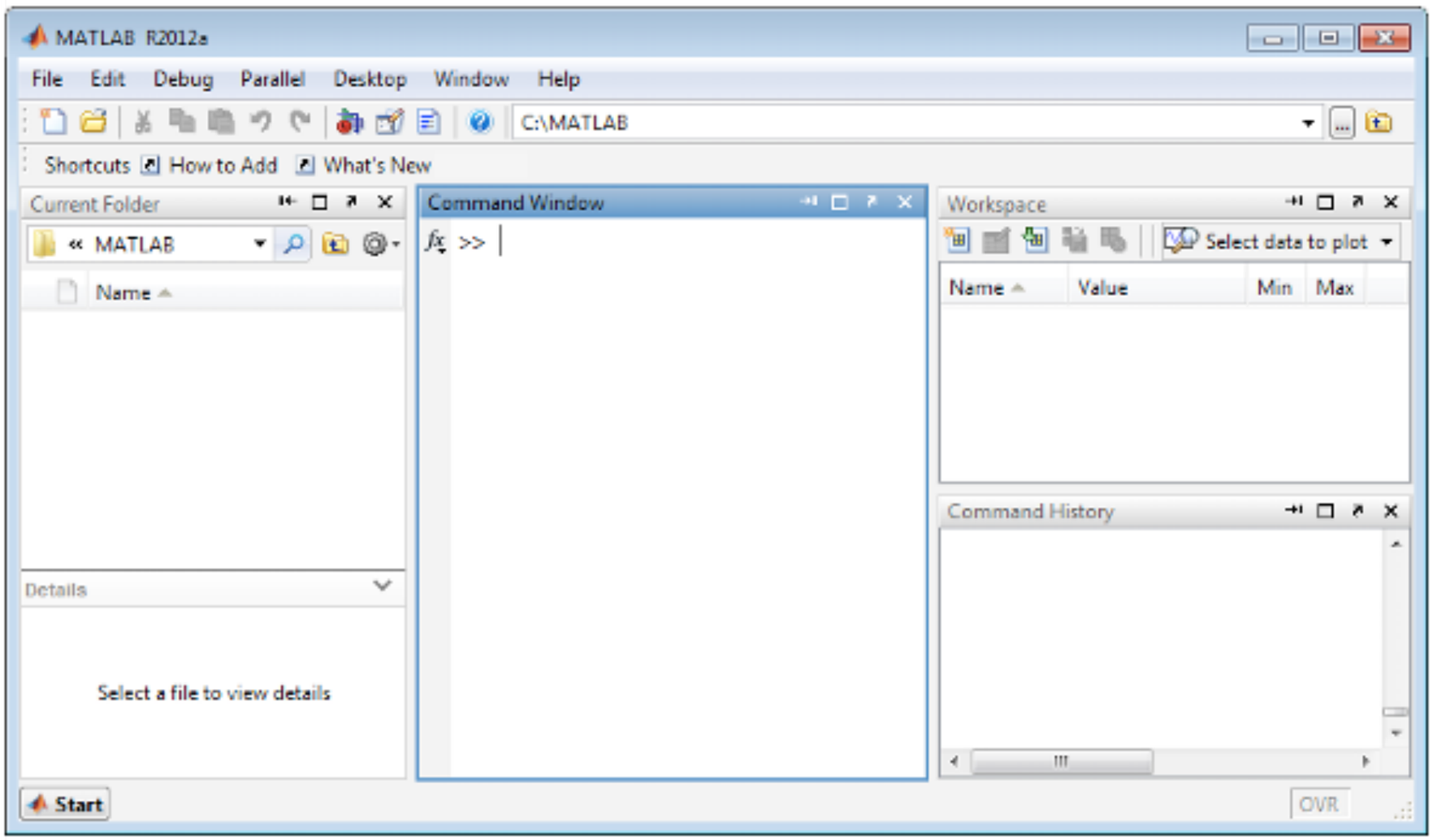
- 8/28 matlab實例探討
- 稀疏矩陣介紹
- Logist map
- Mandelbrot Set



Matlab基本功能介紹

基本環境、功能簡介

Matlab基本功能介紹 基本環境、功能簡介



The desktop includes these panels:

- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).
- **Workspace** — Explore data that you create or import from files.
- **Command History** — View or rerun commands that you entered at the command line.

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (,) or a space.

```
a = [1 2 3 4]
```

returns

```
a =

     1     2     3     4
```

This type of array is a *row vector*.

To create a matrix that has multiple rows, separate the rows with semicolons.

```
a = [1 2 3; 4 5 6; 7 8 10]

a =

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

Another way to create a matrix is to use a function, such as ones, zeros, or rand.
For example, create a 5-by-1 column vector of zeros

```
z = zeros(5,1)

z =

     0
     0
     0
     0
     0
```

Matrix and Array Operations

MATLAB allows you to process all of the values in a matrix using a single arithmetic operator or function.

```
a + 10

ans =

    11    12    13
    14    15    16
    17    18    20

sin(a)

ans =

    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
    0.6570    0.9894   -0.5440
```

Matrix Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Array Operators

+	Addition
-	Subtraction
.*	Element-by-element multiplication
./	Element-by-element division
.\	Element-by-element left division
.^	Element-by-element power
.'	Unconjugated array transpose

Array indexing

For example, consider the 4-by-4 magic square `A`:

```
A = magic(4)

A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
A(4,2)

ans =
    14
```

```
A(8)

ans =
    14
```

```
A(4,5) = 17

A =
    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

```
A(1:3,2)

ans =
     2
    11
     7
```

```
A(3,:)

ans =
     9     7     6    12     0
```

```
B = 0:10:100

B =

     0    10    20    30    40    50    60    70    80    90   100
```

Deleting Rows and Columns

You can delete rows and columns from a matrix using just a pair of square brackets. Start with

```
X = A;
```

Then, to delete the second column of `X`, use

```
X(:,2) = []
```

This changes `X` to

```
X =
     16      2     13
      5     11      8
      9      7     12
      4     14      1
```

If you delete a single element from a matrix, the result is not a matrix anymore. So, expressions like

```
X(1,2) = []
```

result in an error. However, using a single subscript deletes a single element, or sequence of elements, and reshapes the remaining elements into a row vector. So

```
X(2:2:10) = []
```

results in

```
X =
     16      9      2      7     13     12      1
```

Workspace

```
A = magic(4);
B = rand(3,5,2);
```

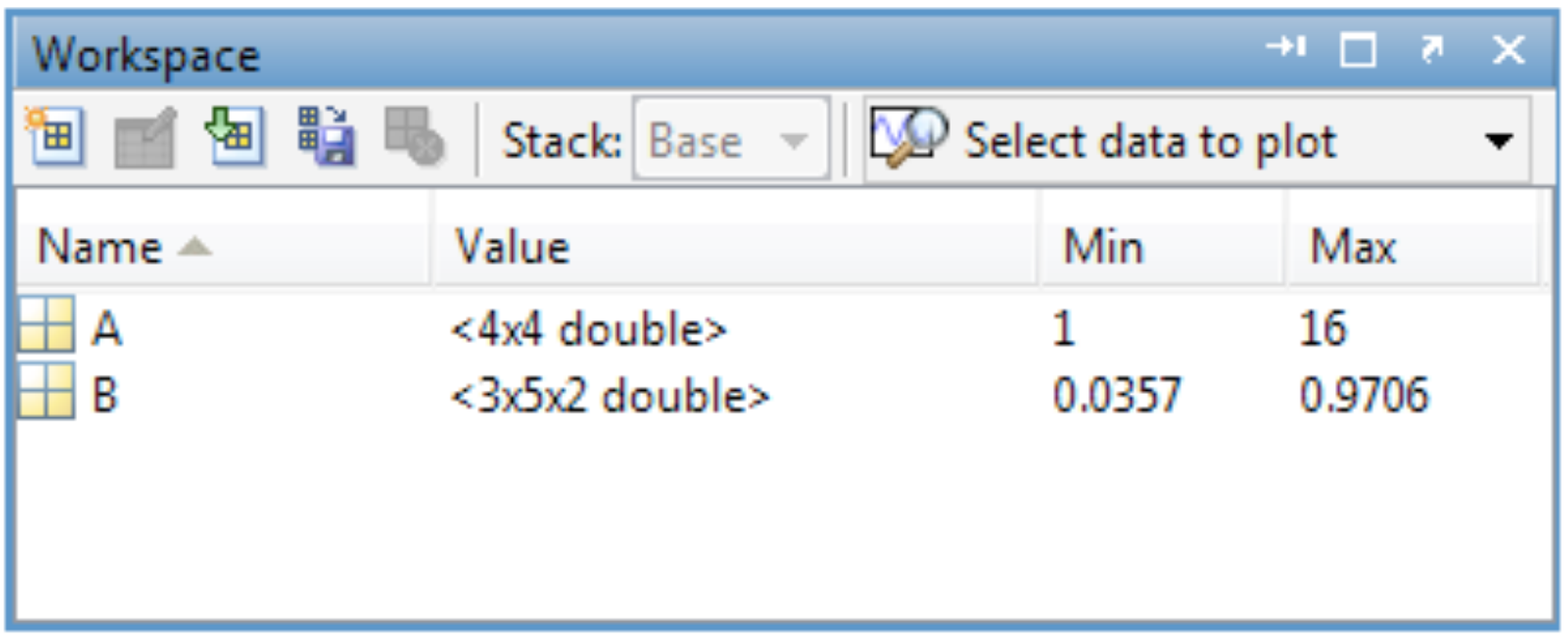
You can view the contents of the workspace using whos.

```
whos
```

```
  Name        Size                Bytes  Class     Attributes

  A           4x4                   128  double
  B           3x5x2                 192  double
```

The variables also appear in the Workspace pane on the desktop.



Character Strings

A *character string* is a sequence of any number of characters enclosed in single quotes. You can assign a string to a variable.

```
myText = 'Hello, world';
```

If the text includes a single quote, use two single quotes within the definition.

```
otherText = 'You''re right'

otherText =

You're right
```

`myText` and `otherText` are arrays, like all MATLAB variables. Their *class* or data type is `char`, which is short for *character*.

```
whos myText
  Name          Size                Bytes  Class     Attributes

  myText        1x12                   24  char
```

You can concatenate strings with square brackets, just as you concatenate numeric arrays.

```
longText = [myText,' - ',otherText]

longText =

Hello, world - You're right
```

To convert numeric values to strings, use functions, such as `num2str` or `int2str`.

```
f = 71;
c = (f-32)/1.8;
tempText = ['Temperature is ',num2str(c),'C']

tempText =

Temperature is 21.6667C
```

Functions

MATLAB provides a large number of functions that perform computational tasks. Functions are equivalent to *subroutines* or *methods* in other programming languages.

Suppose that your workspace includes variables `A` and `B`, such as

```
A = [1 3 5];
B = [10 6 4];
```

To call a function, enclose its input arguments in parentheses:

```
max(A);
```

If there are multiple input arguments, separate them with commas:

```
max(A,B);
```

Return output from a function by assigning it to a variable:

```
maxA = max(A);
```

When there are multiple output arguments, enclose them in square brackets:

```
[maxA,location] = max(A);
```

Enclose any character string inputs in single quotes:

```
disp('hello world');
```

To call a function that does not require any inputs and does not return any outputs, type only the function name:

```
clc
```

The `clc` function clears the Command Window.

PROGRAMMING

PROGRAMMING

Control Flow

- Conditional Control — if, else, switch
- Loop Control — for, while, continue, break
- Program Termination — return
- Vectorization
- Preallocation

Scripts and Functions

- Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

if

```
% Generate a random number
a = randi(100, 1);

% If it is even, divide by 2
if rem(a, 2) == 0
    disp('a is even')
    b = a/2;
end
```

else

```
a = randi(100, 1);

if a < 30
    disp('small')
elseif a < 80
    disp('medium')
else
    disp('large')
end
```

switch

```
[dayNum, dayString] = weekday(date, 'long', 'en_US');

switch dayString
   case 'Monday'
      disp('Start of the work week')
   case 'Tuesday'
      disp('Day 2')
   case 'Wednesday'
      disp('Day 3')
   case 'Thursday'
      disp('Day 4')
   case 'Friday'
      disp('Last day of the work week')
   otherwise
      disp('Weekend!')
end
```

Relational Operators

Relational Operators < Relational operations
> <= >= == ~=

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Logical Operators

See also Logical Operations for functions like `xor`, `all`, `any`, etc.

&&	Logical AND
||	Logical OR
&	Logical AND for arrays
|	Logical OR for arrays
~	Logical NOT

for

```
for n = 3:32
    r(n) = rank(magic(n));
end
r
```

```
for i = 1:m
    for j = 1:n
        H(i,j) = 1/(i+j);
    end
end
```

while

```
    a = 0; fa = -Inf;
    b = 3; fb = Inf;
    while b-a > eps*b
        x = (a+b)/2;
        fx = x^3-2*x-5;
        if sign(fx) == sign(fa)
            a = x; fa = fx;
        else
            b = x; fb = fx;
        end
    end
    x
```

The result is a root of the polynomial $x^3 - 2x - 5$, namely

```
    x =
        2.09455148154233
```

uses interval bisection to find a zero of a polynomial

continue

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) || strncmp(line,'%',1) || ~ischar(line)
        continue
    end
    count = count + 1;
end
fprintf('%d lines\n',count);
fclose(fid);
```

break

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if fx == 0
      break
   elseif sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
x
```

- Vectorization

```
x = .01;
for k = 1:1001
    y(k) = log10(x);
    x = x + .01;
end
```

A vectorized version of the same code is

```
x = .01:.01:10;
y = log10(x);
```

- Preallocation

```
r = zeros(32,1);
for n = 1:32
    r(n) = rank(magic(n));
end
```

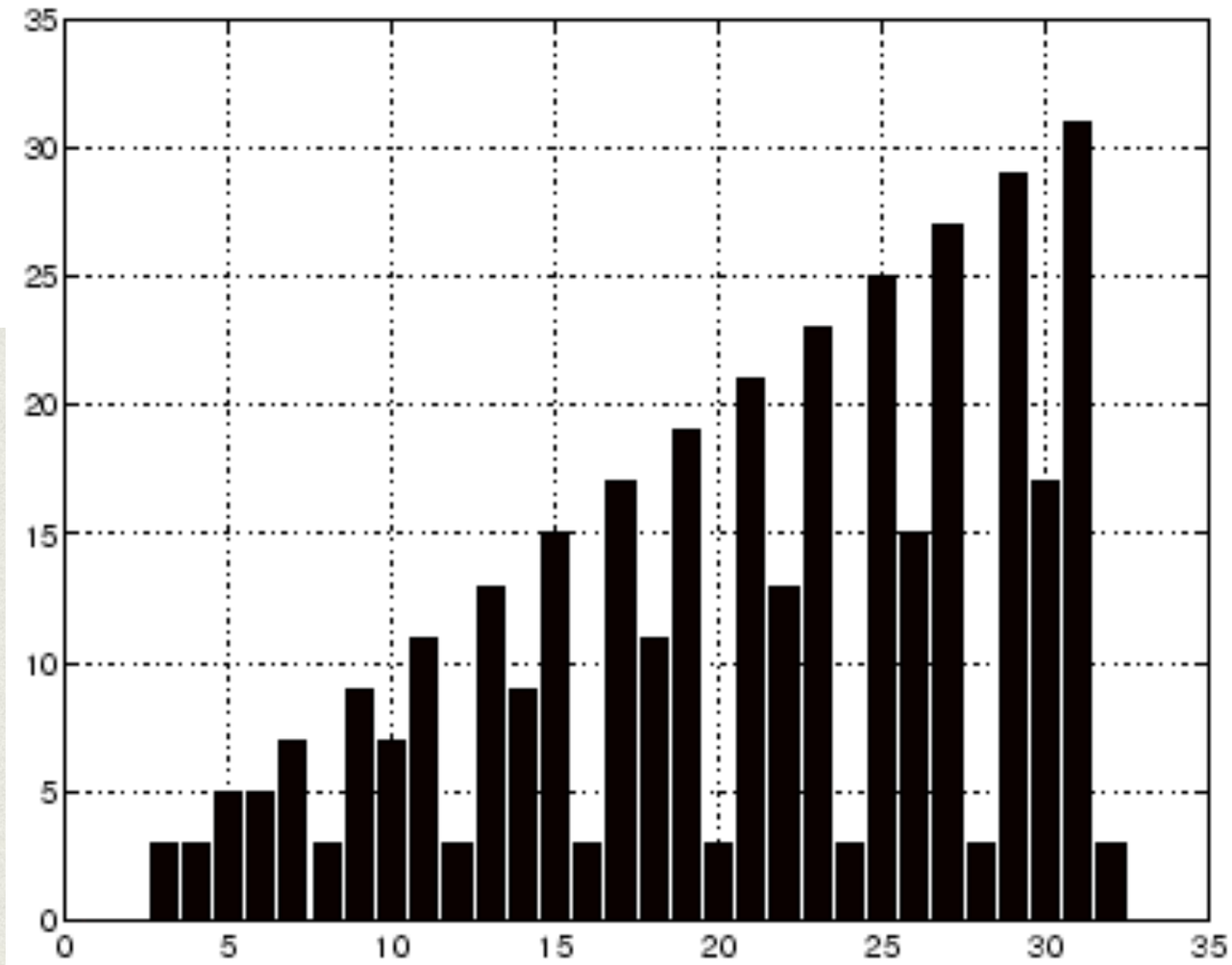
Scripts

For example, create a file called `magicrank.m` that contains these MATLAB commands:

```
% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
   r(n) = rank(magic(n));
end
r
bar(r)
```

Typing the statement

```
magicrank
```

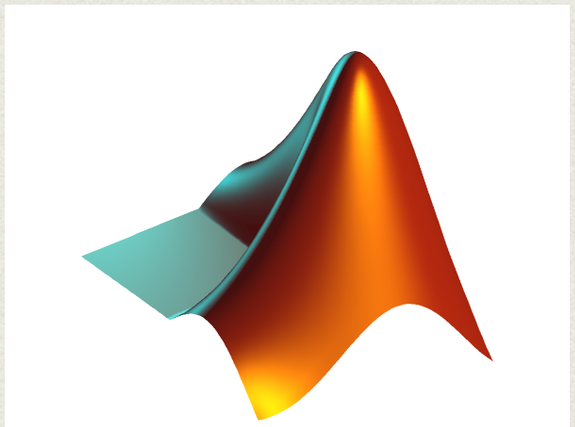


Functions

```
function r = rank(A,tol)
%   RANK Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
   tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

rank.m



Matlab基本功能介紹

在工作站如何使用matlab

SSH

PUTTY

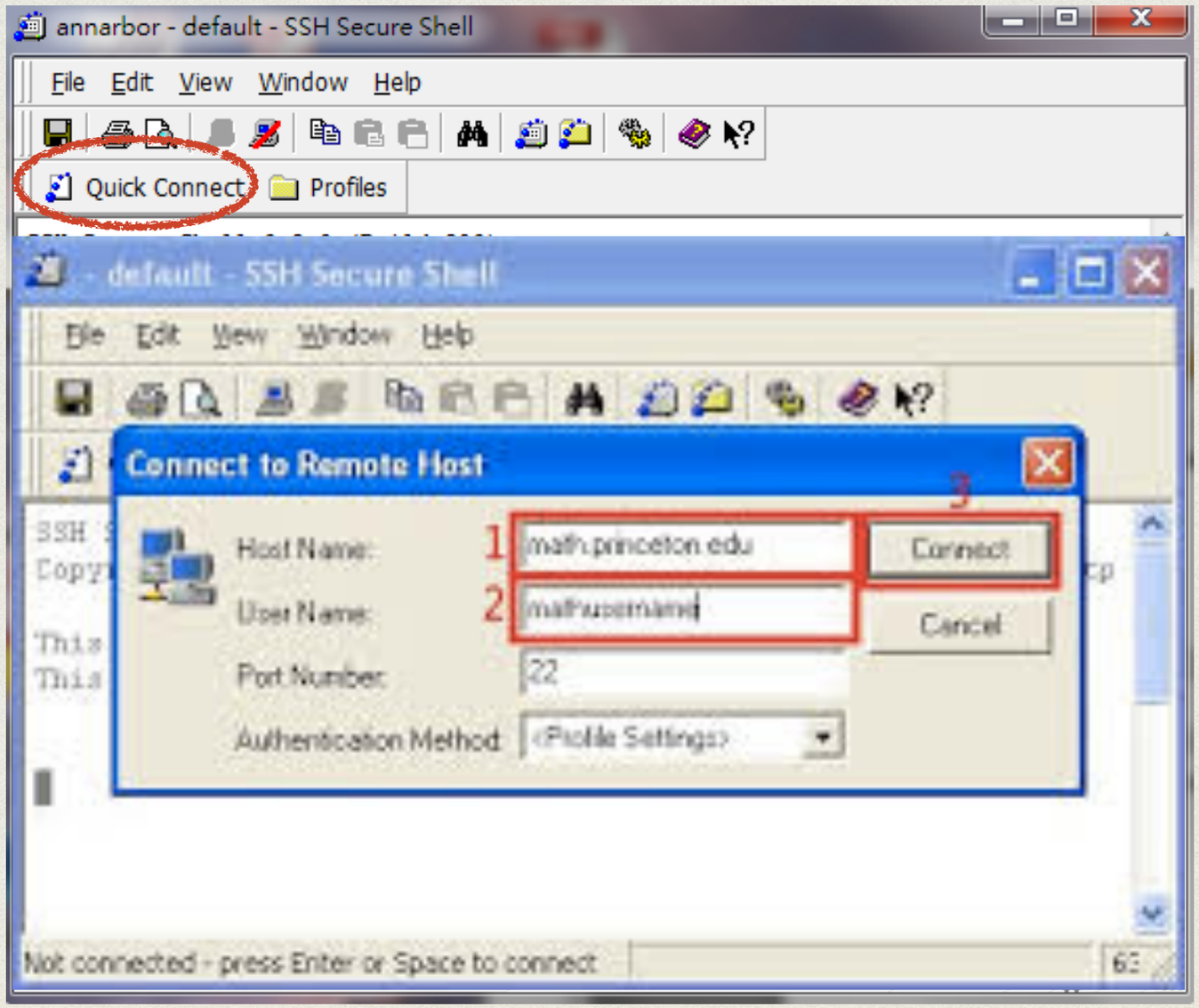
- Download:
http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
- Install “putty.exe”

- Open “putty”
  - Quick connect
  - Input
    - Host name:
    - User name:
    - Password:

SSH SECURE SHELL (ON WINDOWS)

- Download:
  - www.math.ntu.edu.tw/~wwang/cola_lab/knowledge/download/Bootcamp/SSHSecureShellClient.exe
- Install “SSH Secure Shell”

- Open “SSH Secure Shell”
  - Quick connect
  - Input
    - Host name:
    - User name:
    - Password:



X WINDOW

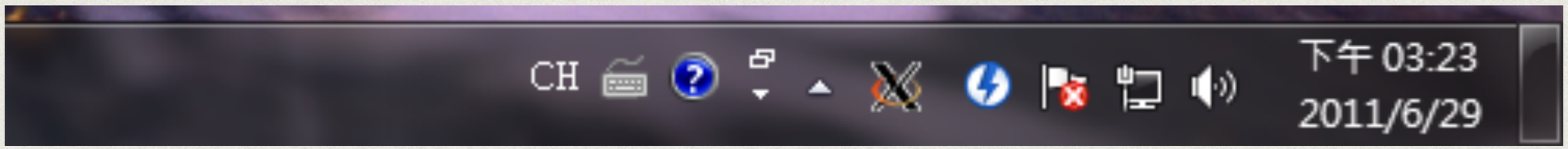
X WINDOW

- A basis for graphical user interfaces(GUI) for networked computer
- Unix-like:
  - Unix
  - Linux
  - Mac OS X
- **`$ssh username@host -X`**

Xming (X Window for Windows)

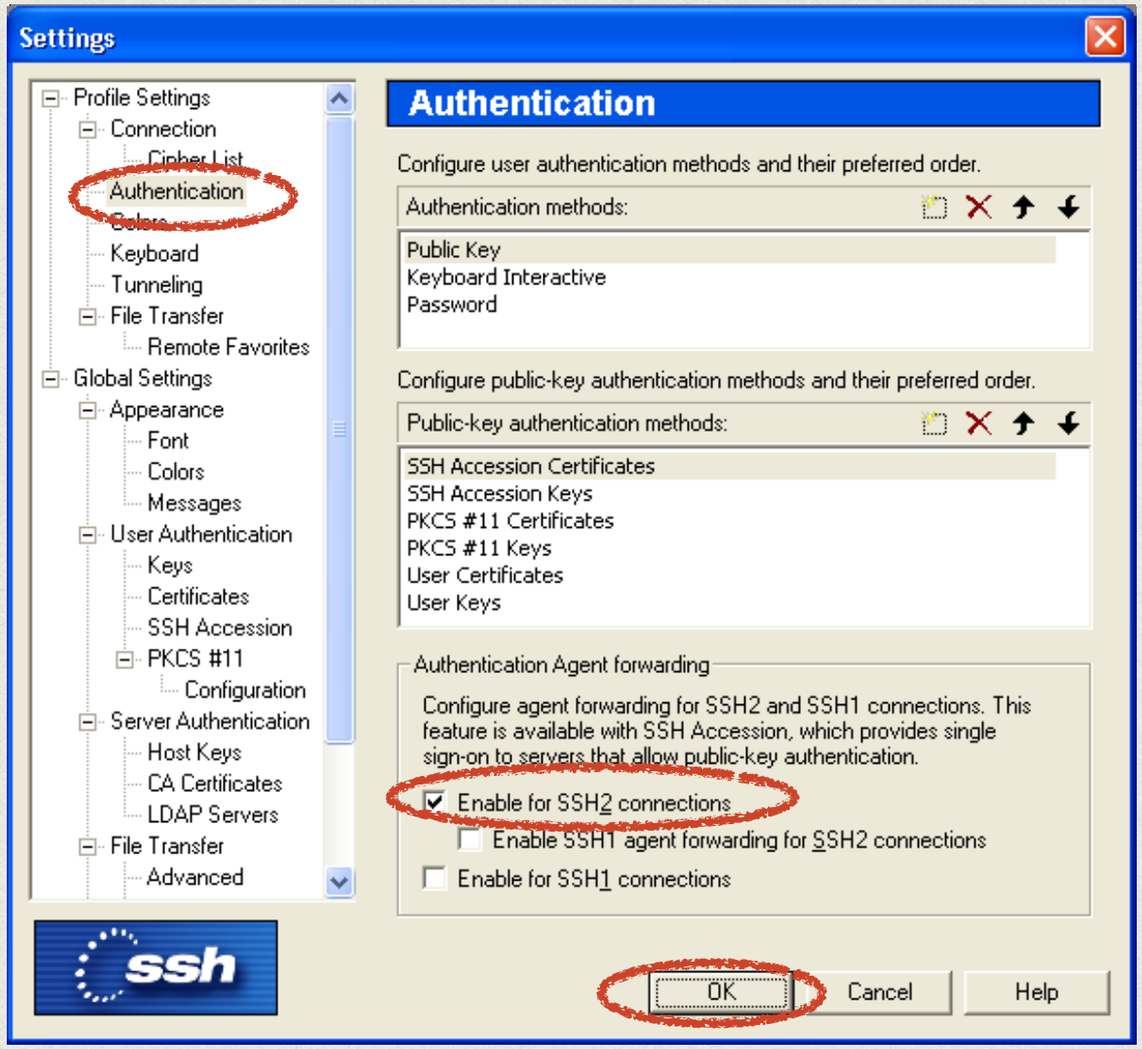
- Download
  - http://sourceforge.net/projects/xming/files/Xming/6.9.0.31/Xming-6-9-0-31-setup.exe/download
- Install “Xming-6-9-0-31-setup.exe”

- Open “Xming”



Configuring and Running Xming and SSH

- Start the Secure Shell Client by clicking on Start/Programs/SSH Secure Shell/Secure Shell
- In the Secure Shell Client click on Edit/Settings.
- In the Settings window on the left-hand side click on the + next to Profile Settings.
- Click on Tunneling.
- On the right-hand side of the window, make sure there is a check next to the words Tunnel X11 Connections
- On the left-hand side of the window, click on Authentication. Make sure there is a check next to the words "Enable for SSH2 Connections.“
- Click OK.
- Click File/Save Settings.

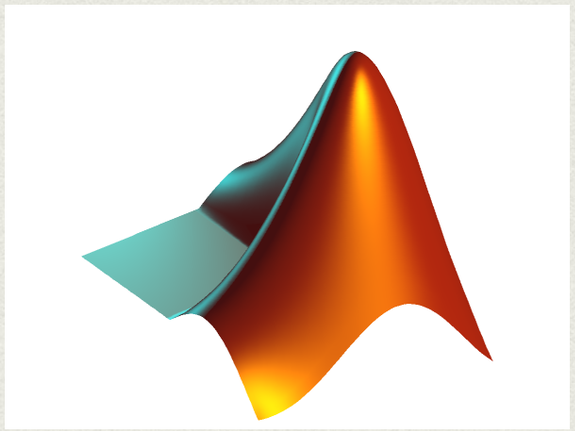


RUN A PROGRAM IN BACKGROUND

```
$ ./a.out >log_a &
$ more log_a

$ ./hello >log_hello &
$ more log_hello

$ nohup matlab -nodisplay <code.m> log_m &
$ more log_m
```

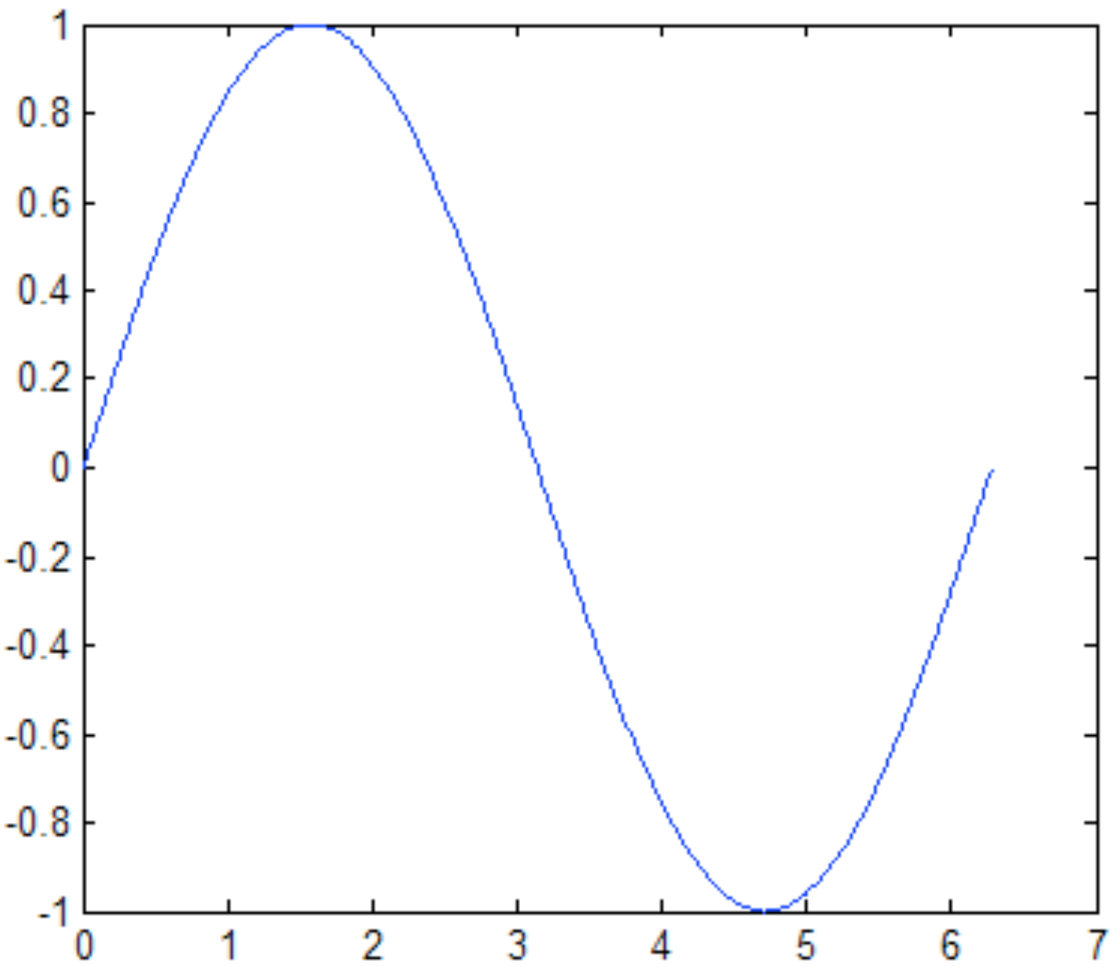


Matlab基本功能介紹

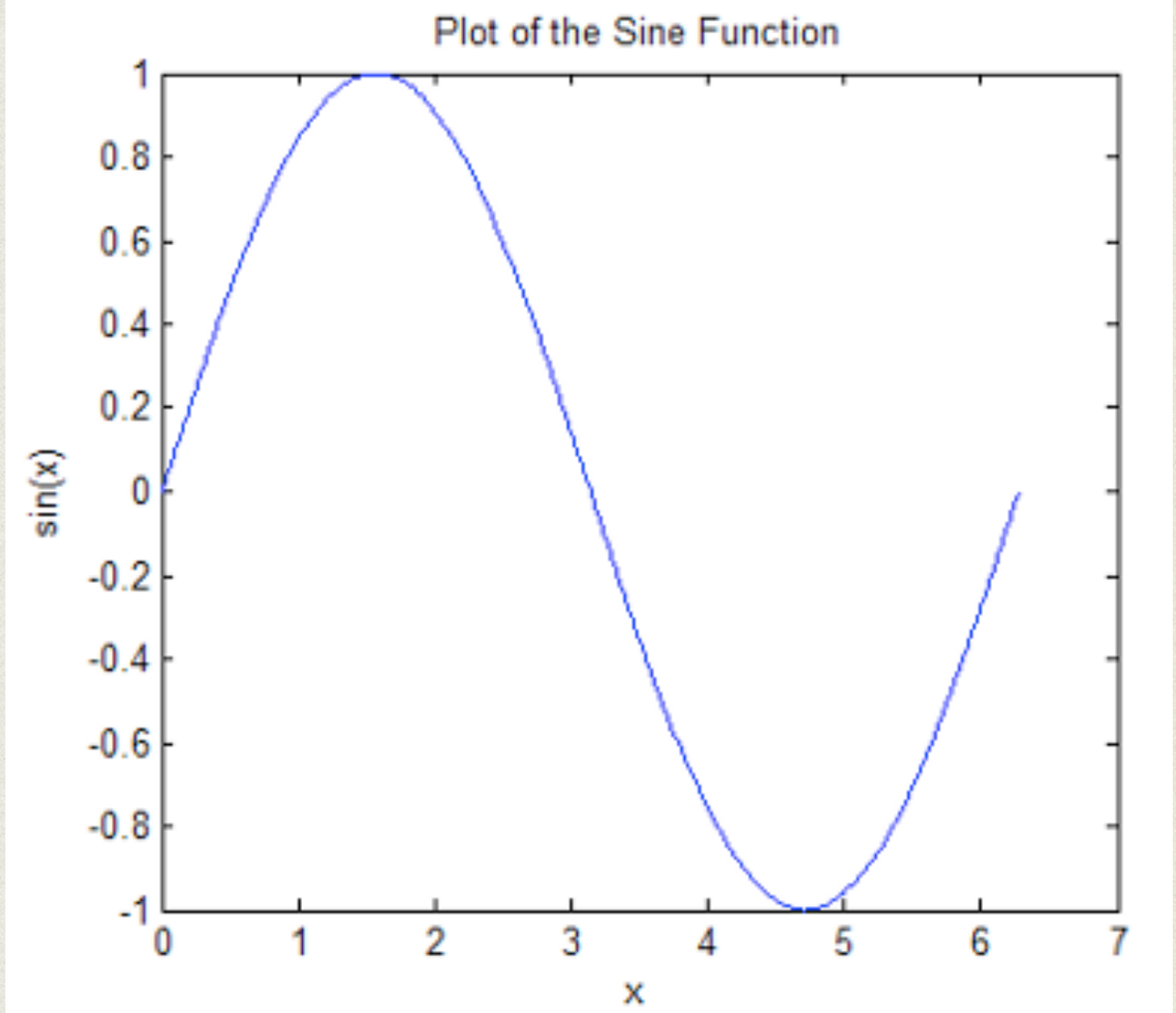
基本繪圖指令介紹

2D Plot:

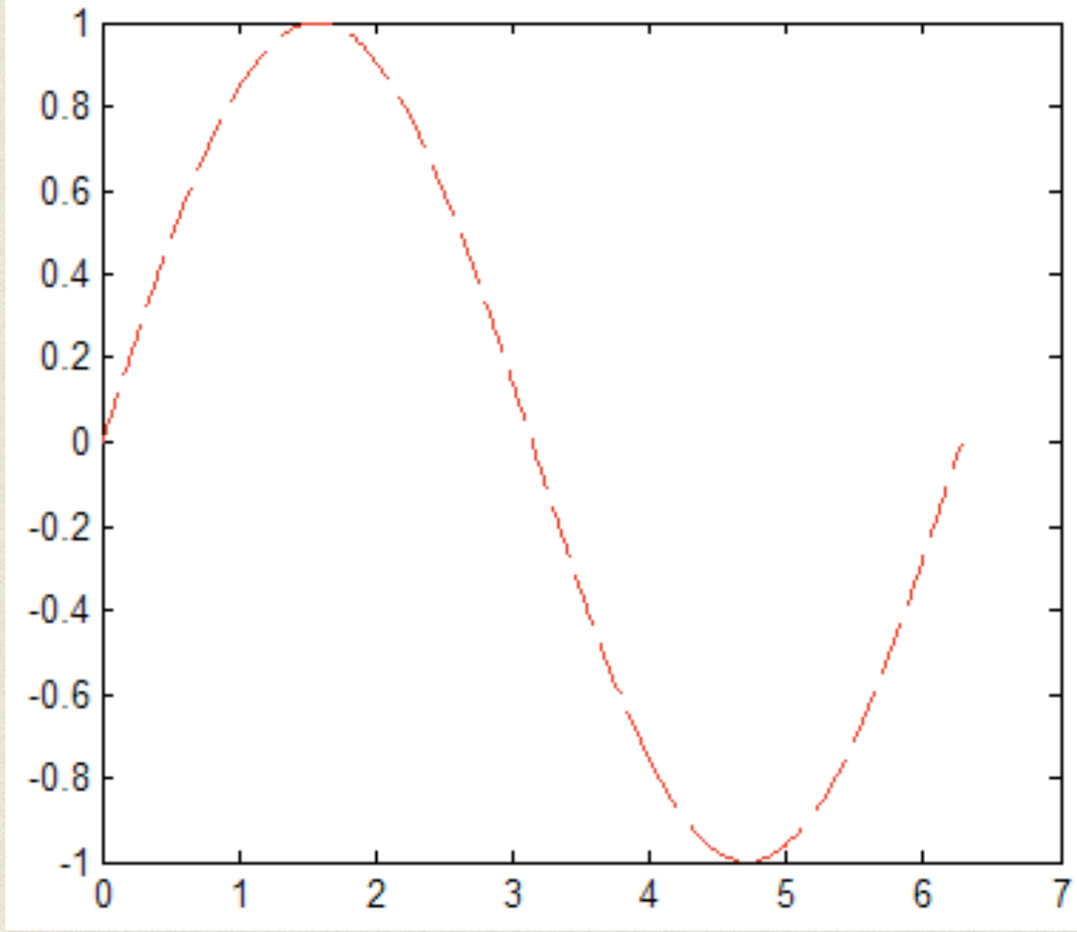
```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```




```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```



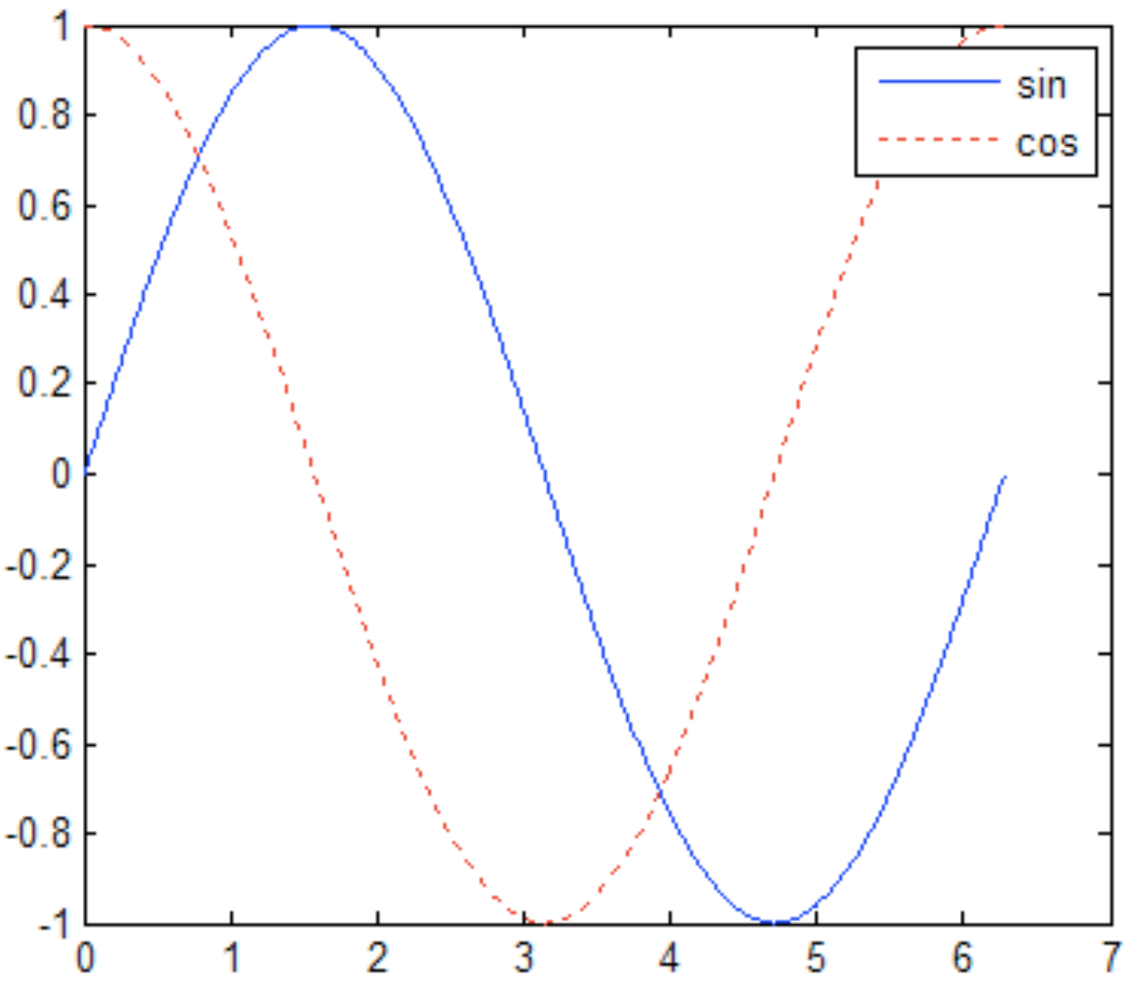
```
plot(x,y,'r--')
```




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

hold on

y2 = cos(x);
plot(x,y2,'r:')
legend('sin','cos')
```



Specifying Line Styles and Colors

```
plot(x,y,'r:+')
```

plots the data using a red-dotted line and places a + marker at each data point.

The strings are composed of combinations of the following elements.

Type	Values	Meanings
Color	'c' 'm' 'y' 'r' 'g' 'b' 'w' 'k'	cyan magenta yellow red green blue white black
Line style	'-' '--' ':' '-.' no character	solid dashed dotted dash-dot no line
Marker type	'+' 'o' '*' 'x' 's' 'd' '^' 'v' '>' '<' 'p' 'h' no character	plus mark unfilled circle asterisk letter x filled square filled diamond filled upward triangle filled downward triangle filled right-pointing triangle filled left-pointing triangle filled pentagram filled hexagram no marker

Setting Axis Limits

The `axis` command enables you to specify your own limits:

```
axis([xmin xmax ymin ymax])
```

or for three-dimensional graphs,

```
axis([xmin xmax ymin ymax zmin zmax])
```

Use the command

```
axis auto
```

to enable automatic limit selection again.

Setting the Axis Aspect Ratio

The `axis` command also enables you to specify a number of predefined modes. For example,

```
axis square
```

makes the *x*-axis and *y*-axis the same length.

```
axis equal
```

makes the individual tick mark increments on the *x*-axes and *y*-axes the same length. This means

```
plot(exp(i*[0:pi/10:2*pi]))
```

followed by either `axis square` or `axis equal` turns the oval into a proper circle:

```
axis auto normal
```

returns the axis scaling to its default automatic mode.

Setting Axis Visibility

You can use the `axis` command to make the axis visible or invisible.

```
axis on
```

makes the axes visible. This is the default.

```
axis off
```

makes the axes invisible.

Setting Grid Lines

The `grid` command toggles grid lines on and off. The statement

```
grid on
```

turns the grid lines on, and

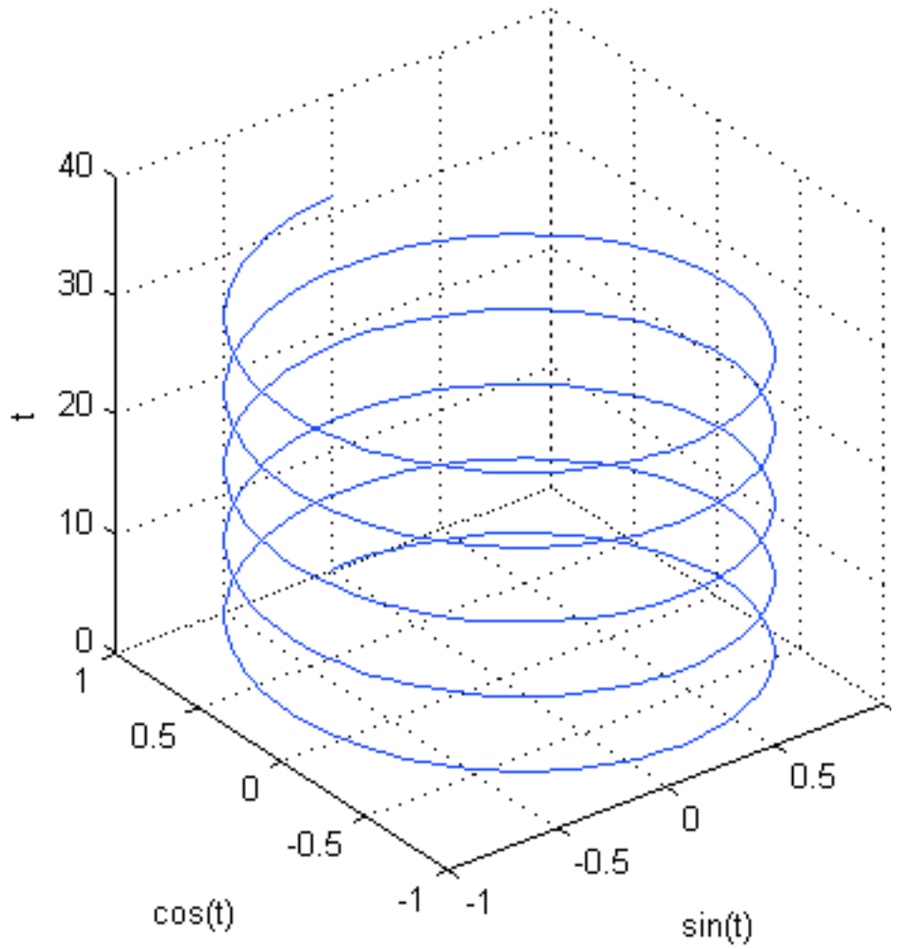
```
grid off
```

turns them back off again.

3D Plots

Plot a three-dimensional helix.

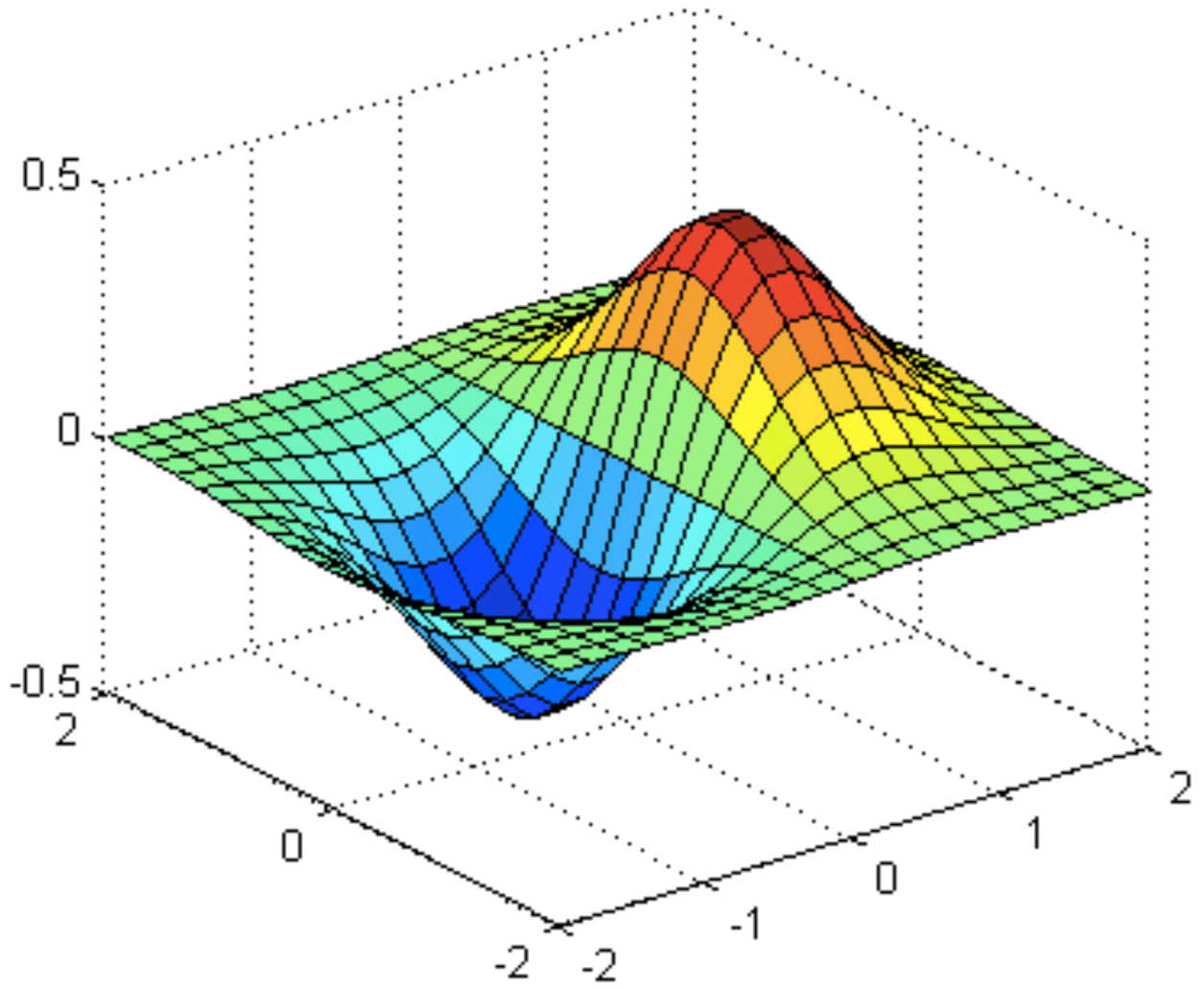
```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
xlabel('sin(t)')
ylabel('cos(t)')
zlabel('t')
grid on
axis square
```



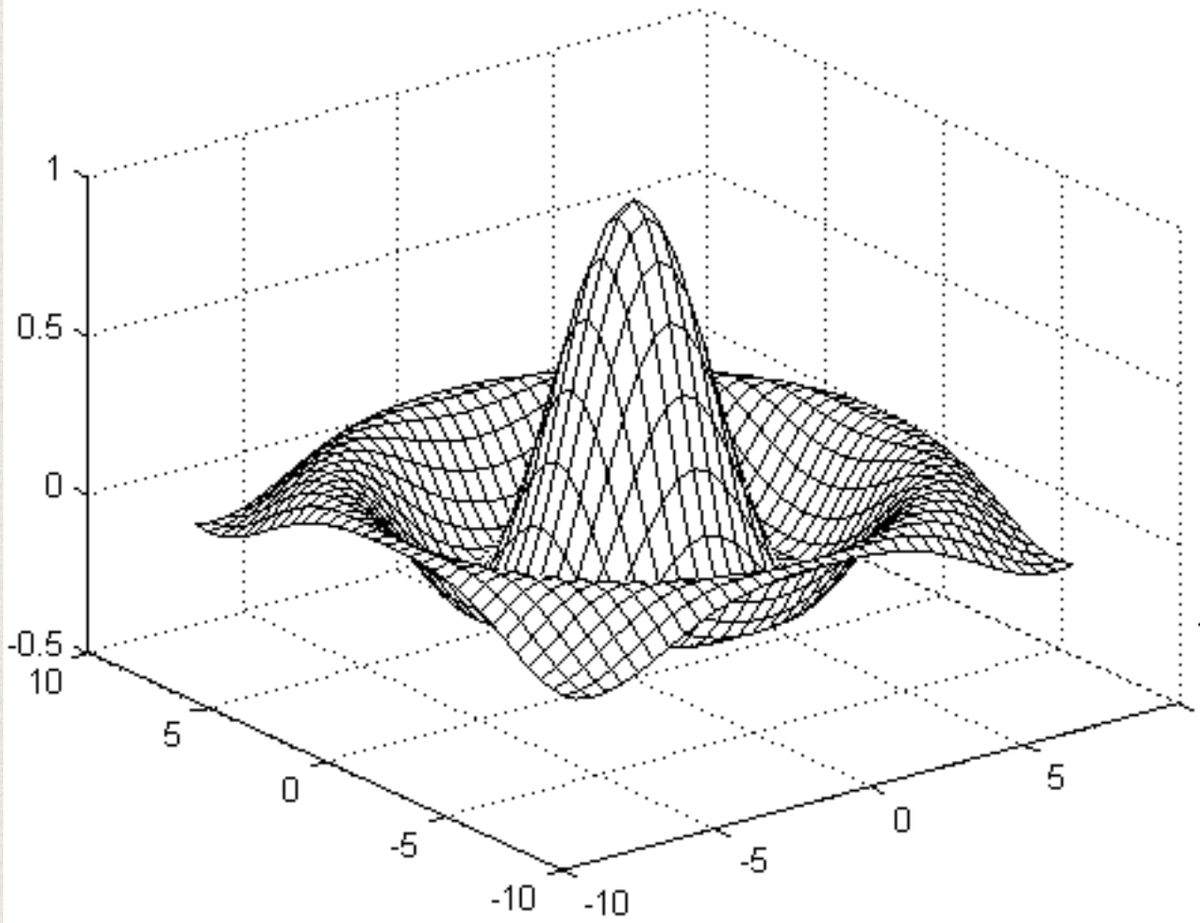

```
[X,Y] = meshgrid(-2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
```

Then, create a surface plot.

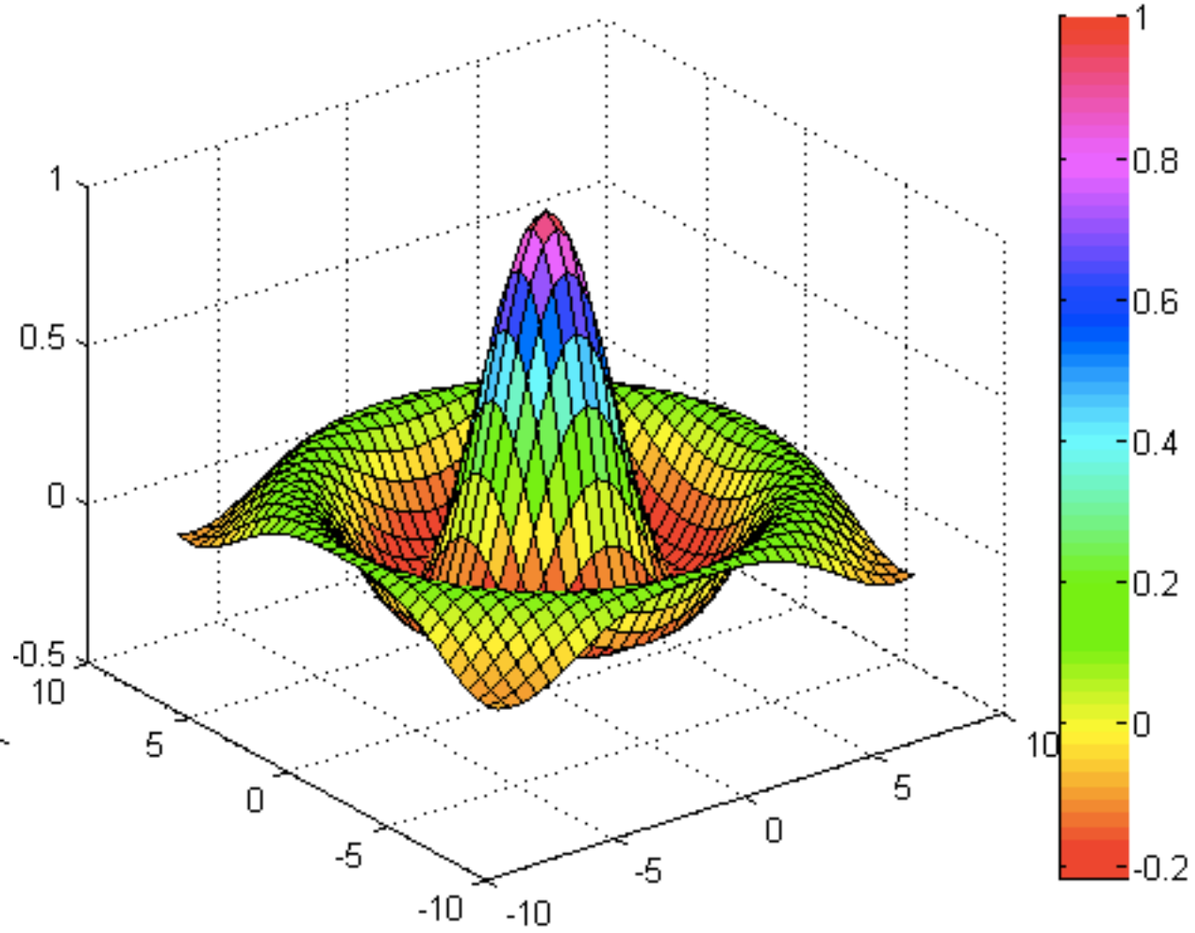
```
surf(X,Y,Z)
```



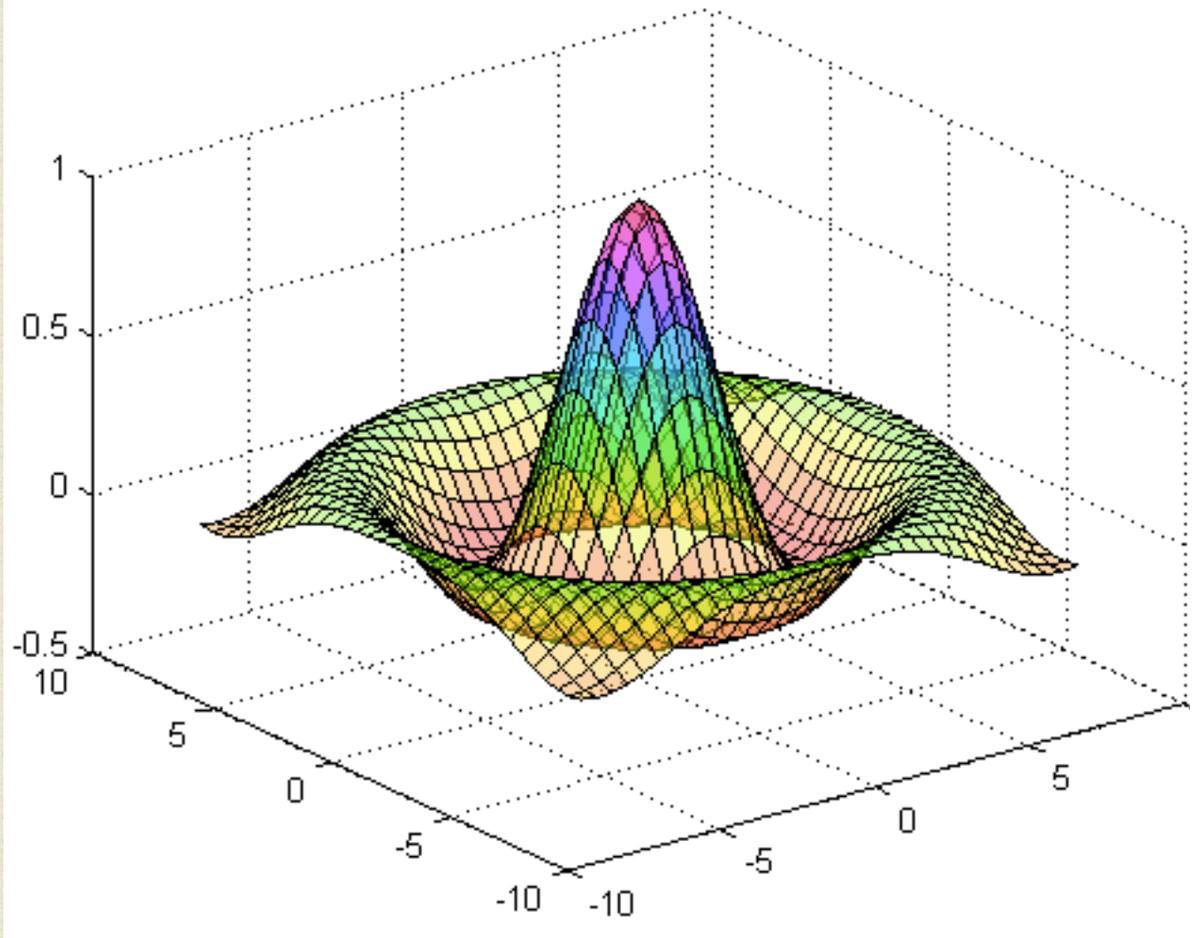
```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(X,Y,Z,'EdgeColor','black')
```



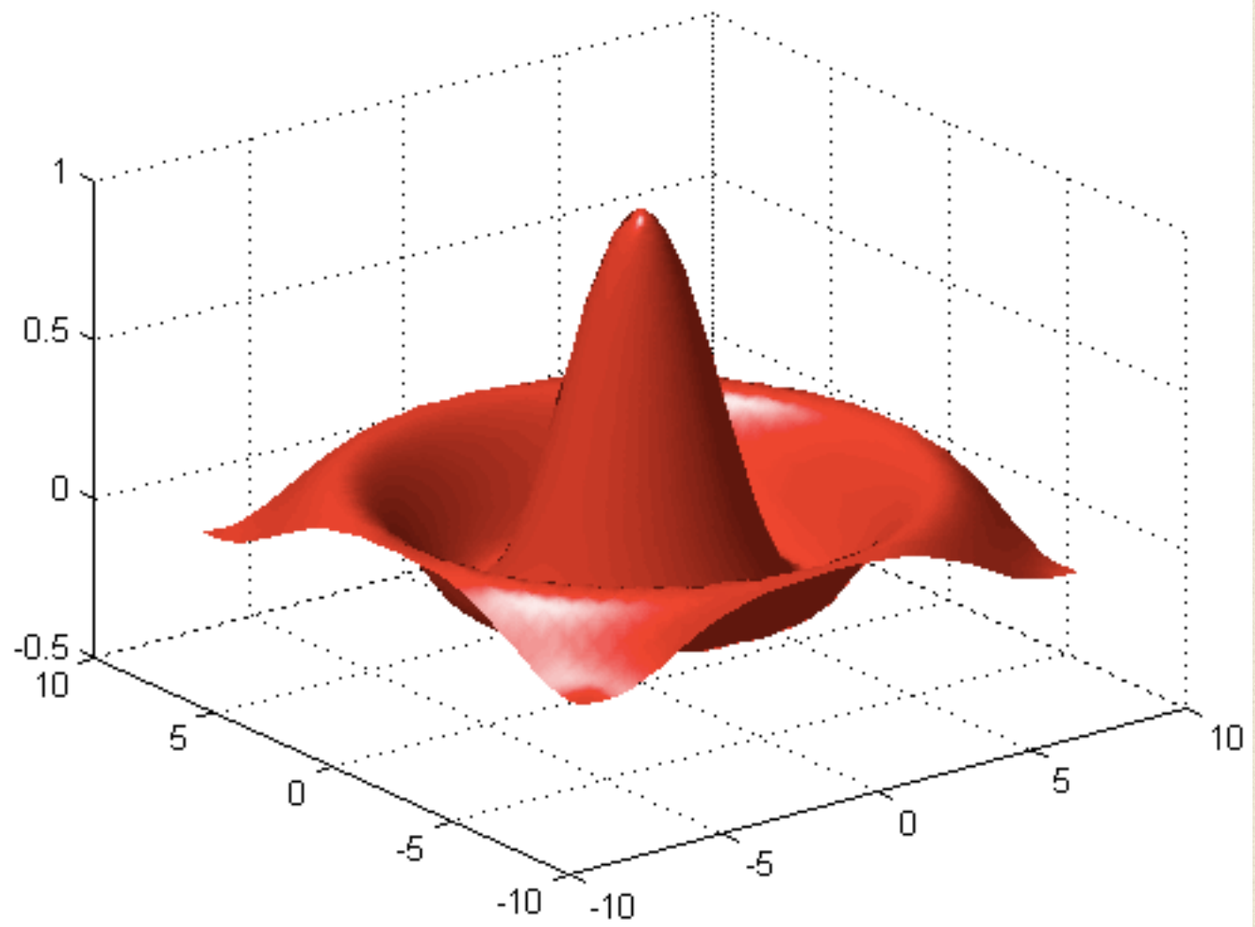

```
surf(X,Y,Z)
colormap hsv
colorbar
```



surf(X,Y,Z)
colormap hsv
alpha(.4)

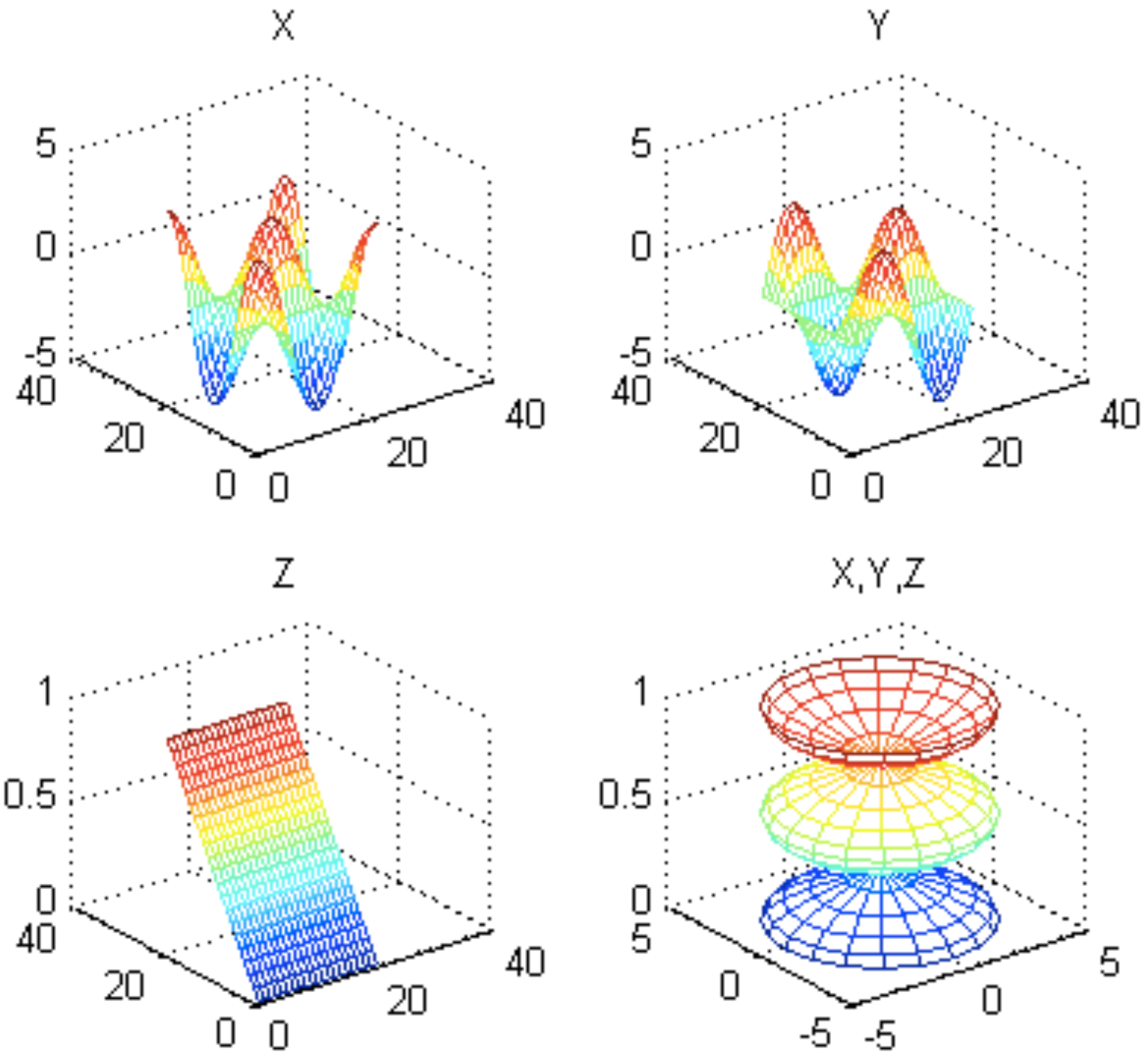



```
surf(X,Y,Z,'FaceColor','red','EdgeColor','none')
camlight left; lighting phong
```



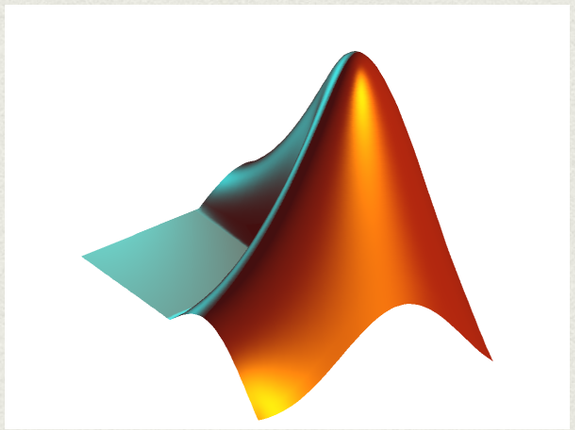
Subplots

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X); title('X');
subplot(2,2,2); mesh(Y); title('Y');
subplot(2,2,3); mesh(Z); title('Z');
subplot(2,2,4); mesh(X,Y,Z); title('X,Y,Z');
```



image

```
load mandrill
figure('color','k')
image(X)
colormap(map)
axis off            % Remove axis ticks and numbers
axis image          % Set aspect ratio to obtain square pixels
```



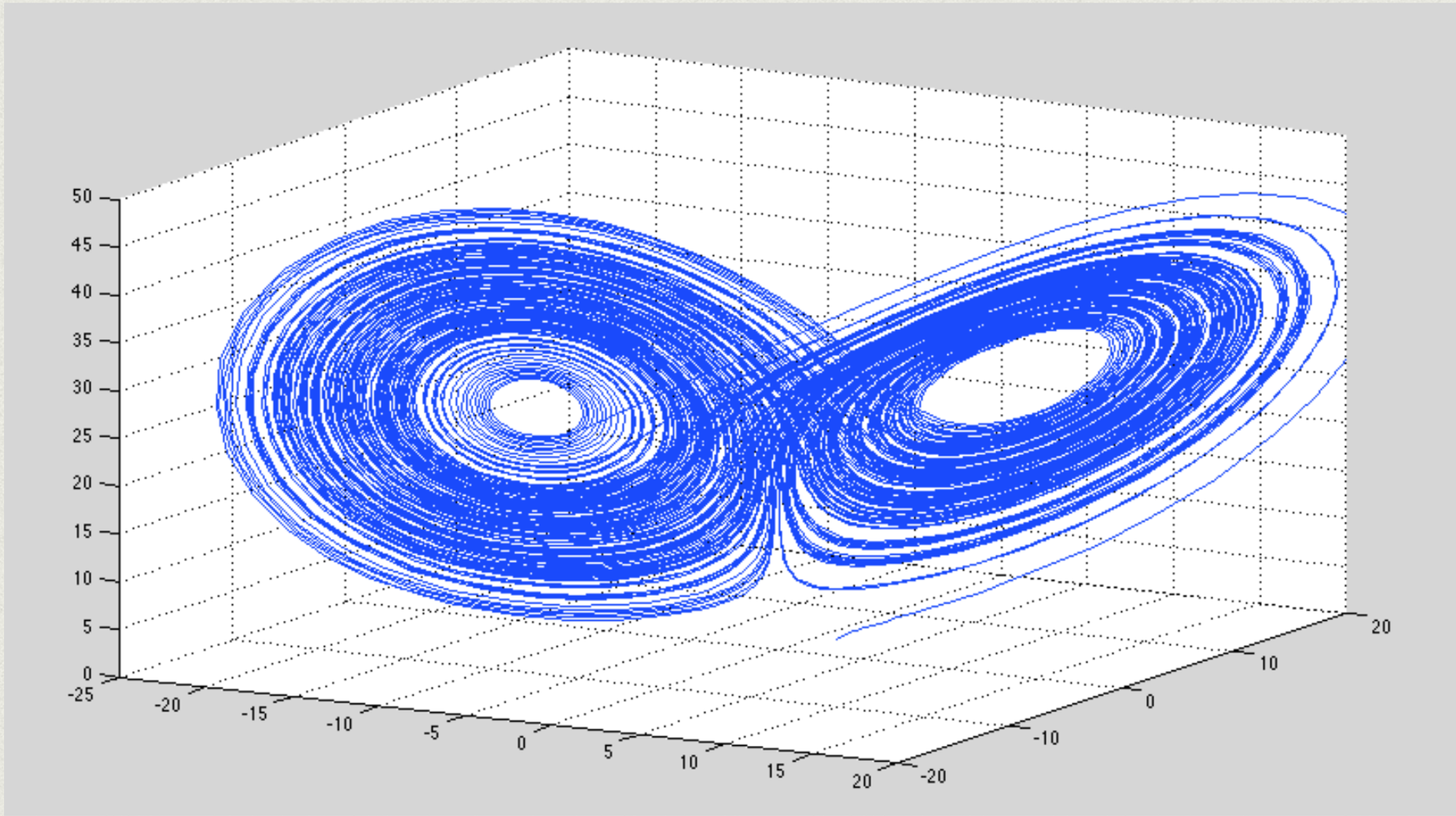
Matlab基本功能介紹

實機演練與討論

Lorenze equation

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

Lorenz used the values $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$.



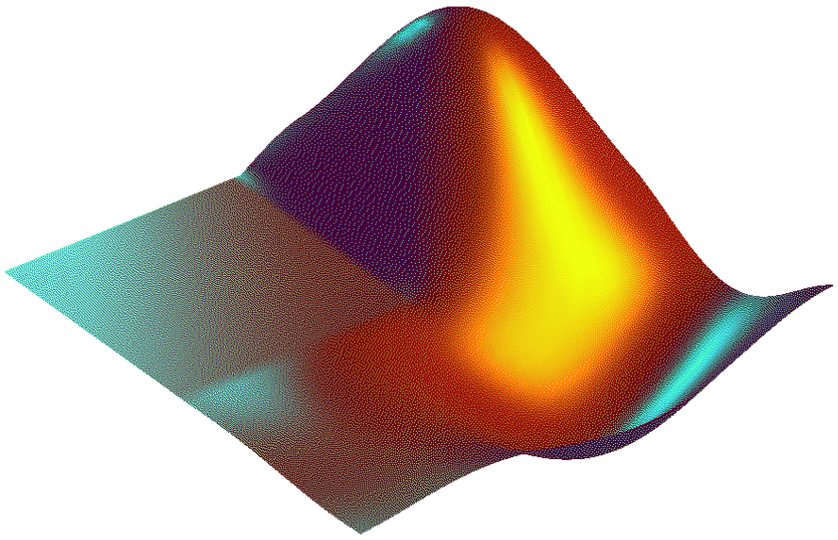
課程內容介紹

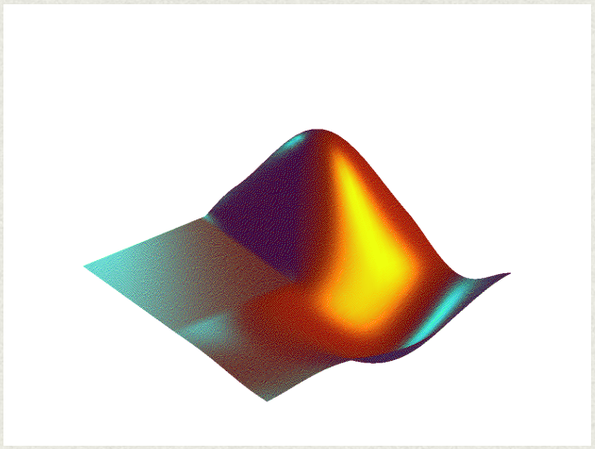
- 8/27：matlab進階功能介紹

1.如何Debug與改進執行效能

2.動畫制作與GUI程式介紹

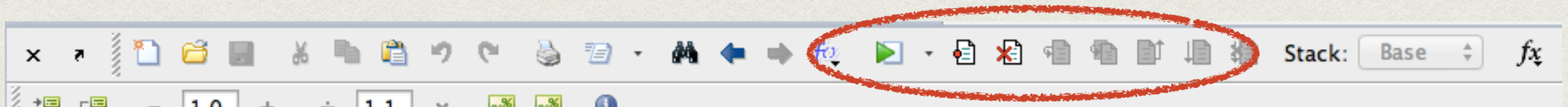
3.實機演練與討論





Matlab進階功能介紹

如何Debug與改進執行效能




```
372 -    if isVisible
373 -        set(logoFig,'color',[.8 .8 .8])
374 -    else
375 -        set(logoFig,'color',[.8 .8 .8], 'Visible', 'Off')
376 -    end
377 -    s = findobj(logoFig, 'type','surf', 'tag', 'TheMathWorksLogo');
378 -    set(s,'facelighting','gouraud')
379
380 -    tic
381 -    for j = 0:n
382 -        t = 0.5*(1-j/n);
383 -        L = cos(mu(1)*t)*L1 + sin(mu(2)*t)*L2 + sin(mu(3)*t)*L3;
384 -        set(s,'zdata',L)
385 -        drawnow
386 -    end
387 -    t = toc;
388 -    if isVisible
389 -        pause(1)
390 -    end
391 -    close(logoFig)
392 -    end
393
```

Toolbar Button	Debug Menu Item	Description	Function Alternative
	**Run** *file* or **Run Configuration for** *file*	Commence execution of file and run until completion or until a breakpoint is encountered. The **Run Configurations for** *file* menu option provides a submenu. The submenu enables you to select a particular run configuration or to edit the run configurations for the MATLAB file. If you choose **Run** *file*, MATLAB uses the default run configuration.	None
None	**Go Until Cursor**	Continue execution of file until the line where the cursor is positioned. Also available on the context menu.	None
	**Step**	Execute the current line of the file.	`dbstep`
	**Step In**	Execute the current line of the file and, if the line is a call to another function, step into that function.	`dbstep` in
	**Continue**	Resume execution of file until completion or until another breakpoint is encountered.	`dbcont`
	**Step Out**	After stepping in, run the rest of the called function or subfunction, leave the called function, and pause.	`dbstep` out
	**Exit Debug Mode**	Exit debug mode.	`dbquit` out

如何增進執行效能

- Preallocating Arrays
- Vectorization
- Changing a Variable's Data Type or Dimension
- Functions Are Generally Faster Than Scripts
- Load and Save Are Faster Than File I/O Functions
- Profile utility
- Stopwatch timer function : tic, toc

- # Preallocating Arrays

The following code creates a scalar variable `x`, and then gradually increases the size of `x` in a `for` loop instead of preallocating the required amount of memory at the start:

```
x = 0;
for k = 2:1000
   x(k) = x(k-1) + 5;
end
```

Change the first line to preallocate a 1-by-1000 block of memory for `x` initialized to zero. This time there is no need to repeatedly reallocate memory and move data as more values are assigned to `x` in the loop:

```
x = zeros(1, 1000);
for k = 2:1000
   x(k) = x(k-1) + 5;
end
```

Preallocating a Nondouble Matrix

When you preallocate a block of memory to hold a matrix of some type other than `double`, avoid using the method

```
A = int8(zeros(100));
```

This statement preallocates a 100-by-100 matrix of `int8` first by creating a full matrix of `doubles`, and then converting each element to `int8`. This costs time and uses memory unnecessarily.

The next statement shows how to do this more efficiently:

```
A = zeros(100, 'int8');
```

- Vectorization

Here is one way to compute the sine of 1001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

A vectorized version of the same code is

```
t = 0:.01:10;
y = sin(t);
```

The second example executes much faster than the first and is the way MATLAB is meant to be used. Test this on your system by creating scripts that contain the code shown, and then using the `tic` and `toc` functions to measure the performance.

Functions Used in Vectorizing

Some of the most commonly used functions for vectorizing are as follows

Function	Description
all	Test to determine if all elements are nonzero
any	Test for any nonzeros
cumsum	Find cumulative sum
diff	Find differences and approximate derivatives
find	Find indices and values of nonzero elements
ind2sub	Convert from linear index to subscripts
ipermute	Inverse permute dimensions of a multidimensional array
logical	Convert numeric values to logical
meshgrid	Generate X and Y arrays for 3-D plots
ndgrid	Generate arrays for multidimensional functions and interpolation
permute	Rearrange dimensions of a multidimensional array
prod	Find product of array elements
repmat	Replicate and tile an array
reshape	Change the shape of an array
shiftdim	Shift array dimensions
sort	Sort array elements in ascending or descending order
squeeze	Remove singleton dimensions from an array
sub2ind	Convert from subscripts to linear index
sum	Find the sum of array elements

The Profiler Utility

A good first step to speeding up your programs is to find out where the bottlenecks are. This is where you need to concentrate your attention to optimize your code.

The MATLAB software provides the MATLAB Profiler, a graphical user interface that shows you where your program is spending its time during execution. Use the Profiler to help you determine where you can modify your code to make performance improvements.

To start the Profiler, type `profile viewer` or select **Desktop > Profiler** in the MATLAB Command Window. See Profiling for Improving Performance in the MATLAB Desktop Tools and Development Environment documentation, and the `profile` function reference page.

Stopwatch Timer Functions

If you just need to get an idea of how long your program (or a portion of it) takes to run, or to compare the speed of different implementations of a program, you can use the stopwatch timer functions, `tic` and `toc`. Invoking `tic` starts the timer, and the first subsequent `toc` stops it and reports the time elapsed between the two.

Use `tic` and `toc` as shown here:

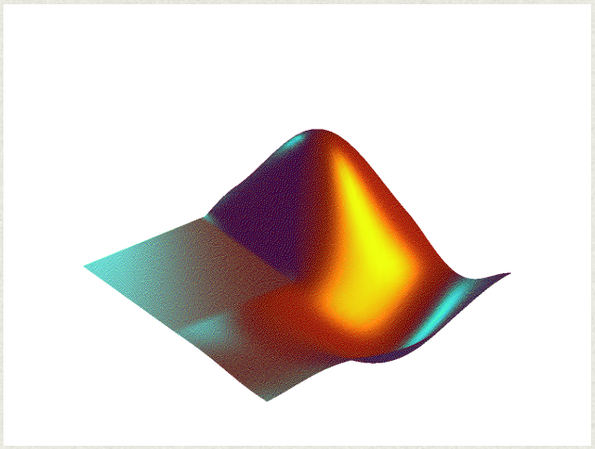
```
tic
   -- run the program section to be timed --
toc
```

Keep in mind that `tic` and `toc` measure overall elapsed time. Make sure that no other applications are running in the background on your system that could affect the timing of your MATLAB programs.

Measuring Smaller Programs

Shorter programs sometimes run too fast to get useful data from `tic` and `toc`. When this is the case, try measuring the program running repeatedly in a loop, and then average to find the time for a single run:

```
tic
   for k = 1:100
      -- run the program --
   end
toc
```



Matlab進階功能介紹

動畫制作與GUI程式介紹

動畫制作

MATLAB 產生動畫的方式有兩種：

電影方式：

以影像的方式預存多個畫面，再將這些畫面快速的呈現在螢幕上，就可以得到動畫的效果。此種方式類似於電影的原理，可以產生很繽紛亮麗的動畫，但是其缺點為每個畫面都必需事先備妥，無法進行及時成像（Real-time Rendering），而且每個畫面，以至於整套動畫，都必需佔用相當大的記憶體空間。

物件方式：

在 MATLAB 的「握把式圖形」（Handle Graphics）概念下，所有的曲線或曲面均可被視為一個物件，MATLAB 可以很快的抹去舊曲線，並產生相似但不同的新曲線，此時就可以看到曲線隨時間而變化的效果。使用物件方式（即握把式圖形）所產生的動畫，可以呈現即時的變化，也不需要太高的記憶體需求，但其缺點是較難產生太複雜的動畫。

Reference by 「MATLAB 程式設計入門篇 動畫製作 張智星」

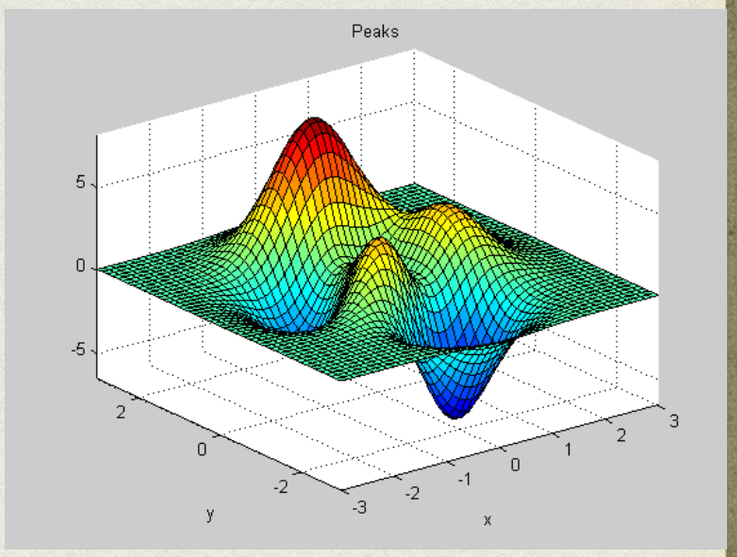
電影方式：

以電影方式來產生動畫，可由下列兩個步驟來達成：

使用 getframe 指令來抓取圖形做為電影的畫面，每個畫面都是以一個行向量的方式，置放於整個代表電影的矩陣。

使用 movie 指令來播放電影，並可指定播放的重複次數及每秒播放的畫面數目。

```
clear M                    % 清除電影資料矩陣 M
n = 50;                    % 抓取 50 個畫面
peaks;
fprintf('抓取畫面中...\n');
for i = 1:n
    view([-37.5+i*360/n, 30]);% 改變觀測角度
    M(i) = getframe;       % 抓取畫面，並存入電影資料矩陣 M
end
fprintf('播放電影中...\n');
movie(M, 3);               % 播放電影三次
```



物件方式

以電影方式產生動畫可以說是「暴力法」，因為此方法佔掉了許多記憶體空間。另一個技巧性較高的方法則是以物件方式產生動畫，此種方法不需要大量的記憶體，而且可以產生「即時」（Real-time）或「互動式」（Interactive）的動畫。

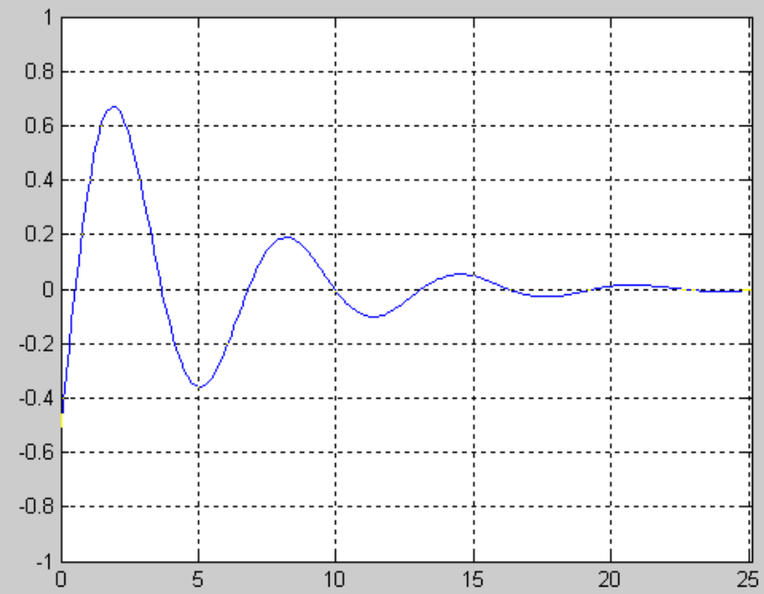
MATLAB 的所有圖形元件（曲線、曲面、圖軸等）都是物件，您可以控制這些物件的各種性質，此種特性稱為「握把式圖形」（Handle Graphics）。握把式圖形包含的層面很廣，但牽涉到動畫部份的基本概念並不複雜，以下我們以曲線的動畫來說明。

我們可以快速地改變圖形物件的性質（如顏色、座標等），就可以達到動畫的效果
每一條曲線都有下列三種性質：
xdata：此為一向量，代表曲線的 x 座標值
ydata：此為一向量，代表曲線的 y 座標值
EraseMode：此為一字串，代表曲線被抹除的方式，亦即當 xdata 或 ydata 被改變時，對於舊曲線的處理方式。

我們產生一條衰減的正弦曲線 $y = \sin(x-k)e^{-\frac{x}{5}}$

讓 k 隨時間而便大（即改變正弦波的相角），使整條曲線產生舞動的效果。

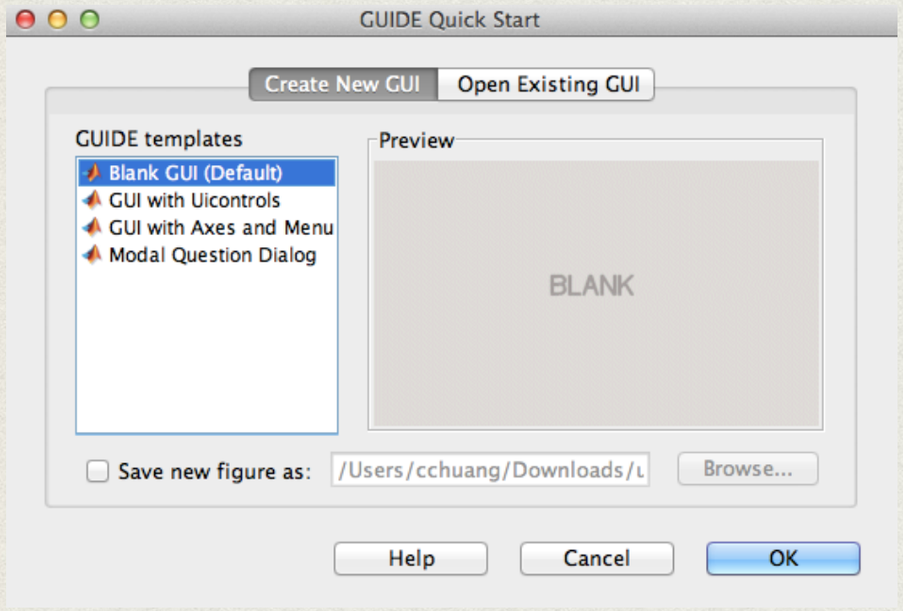
```
x = 0:0.1:8*pi;
h = plot(x, sin(x).*exp(-x/5), 'EraseMode', 'xor');
axis([-inf inf -1 1]);                    % 設定圖軸的範圍
grid on                                   % 畫出格線
for i = 1:5000
      y = sin(x-i/50).*exp(-x/5);
      set(h, 'ydata', y);                 % 設定新的 y 座標
      drawnow                             % 立即作圖
end
```

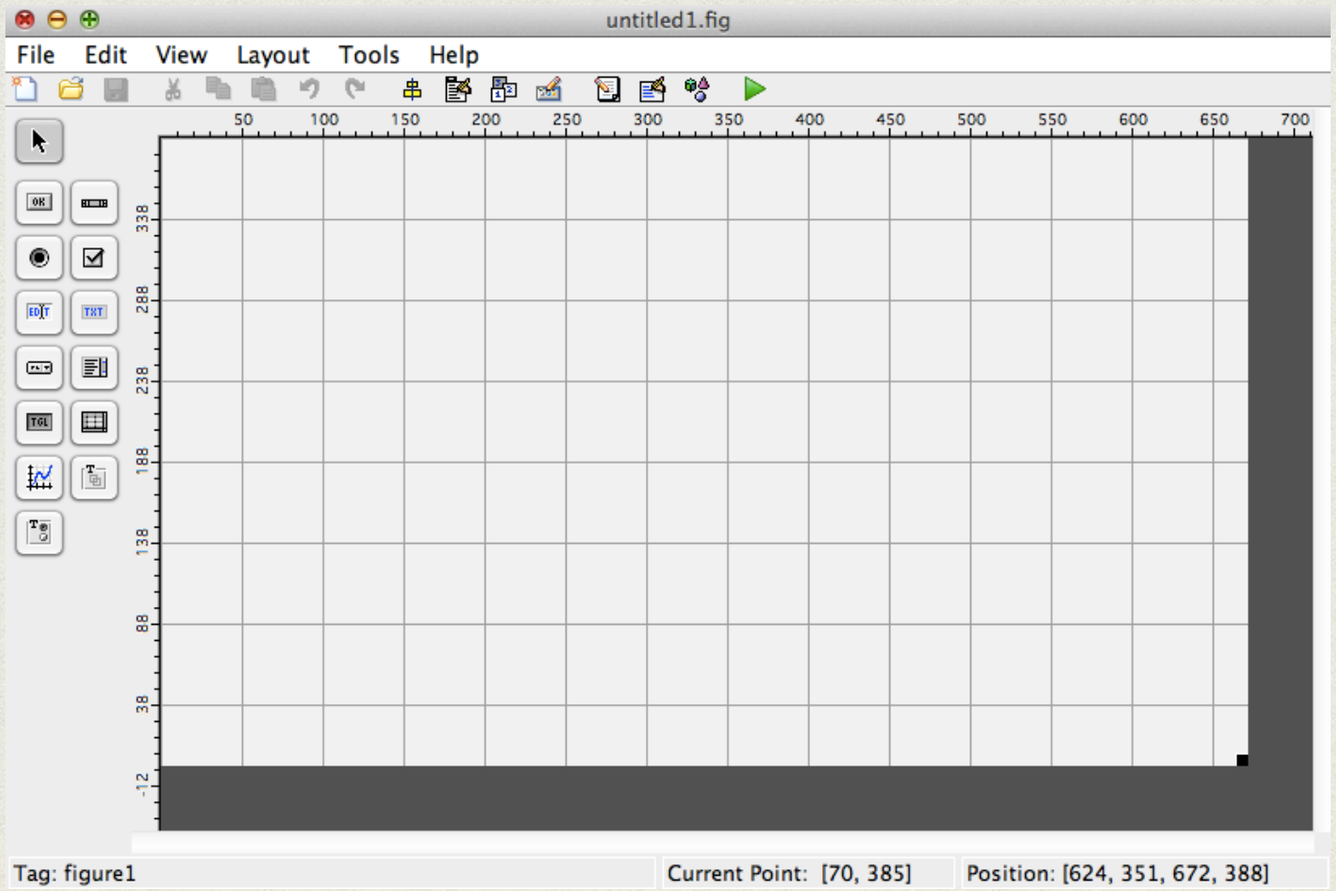


GUI程式簡介

You can build MATLAB GUIs in two ways:

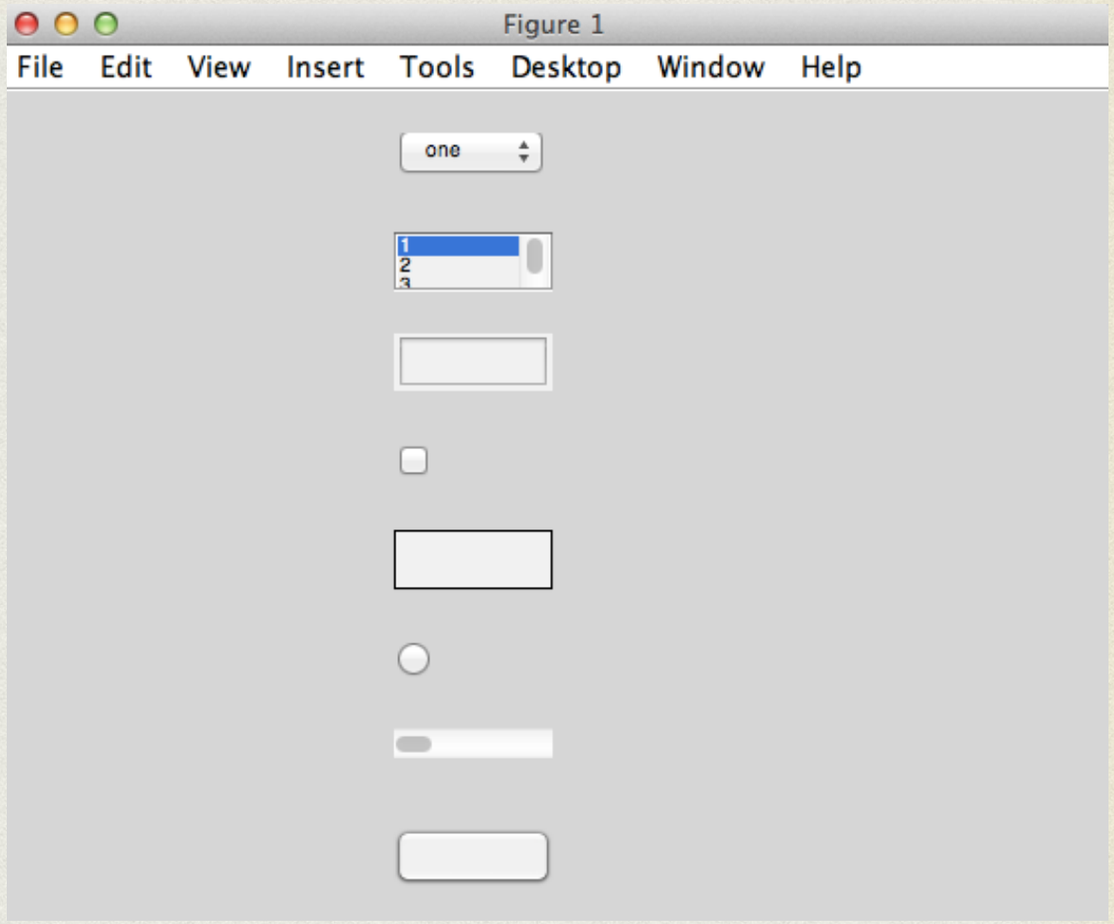
- Use GUIDE (GUI Development Environment), an interactive GUI construction kit.
- Create code files that generate GUIs as functions or scripts (programmatic GUI construction).





Uicontrol產生UI (User Interface) 控制物件

- 按鈕（Push Button）
- 滑動棒（Sliding Bar）
- 圓形按鈕（Radio Button）
- 框架（Frame）
- 核計方塊（Check Box）
- 文字欄位（Edit Box）
- 列表式選單（List Menu）
- 下拉式選單（Popup Menu）




```
uicontrol('style','push','position',[200  20 80 30]);
uicontrol('style','slide','position',[200 70 80 30]);
uicontrol('style','radio','position',[200 120 80 30]);
uicontrol('style','frame','position',[200 170 80 30]);
uicontrol('style','check','position',[200 220 80 30]);
uicontrol('style','edit','position',[200 270 80 30]);
uicontrol('style','list','position',[200 320 80 30],'string', '1|2|3|4');
uicontrol('style','popup','position',[200 370 80 30],'string','one|two|three');
```

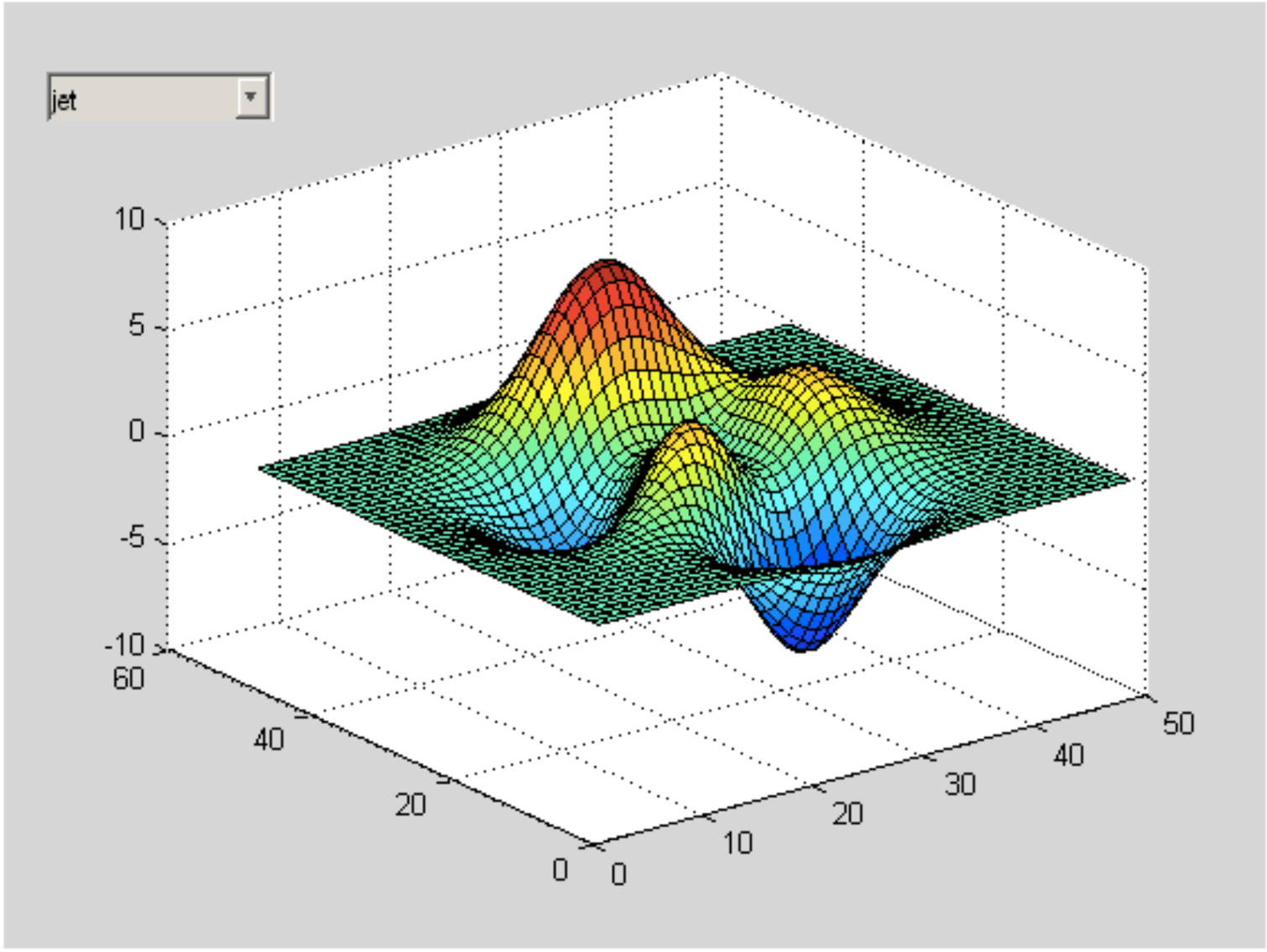
uicontrol

Create a figure and an axes to contain a 3-D surface plot.

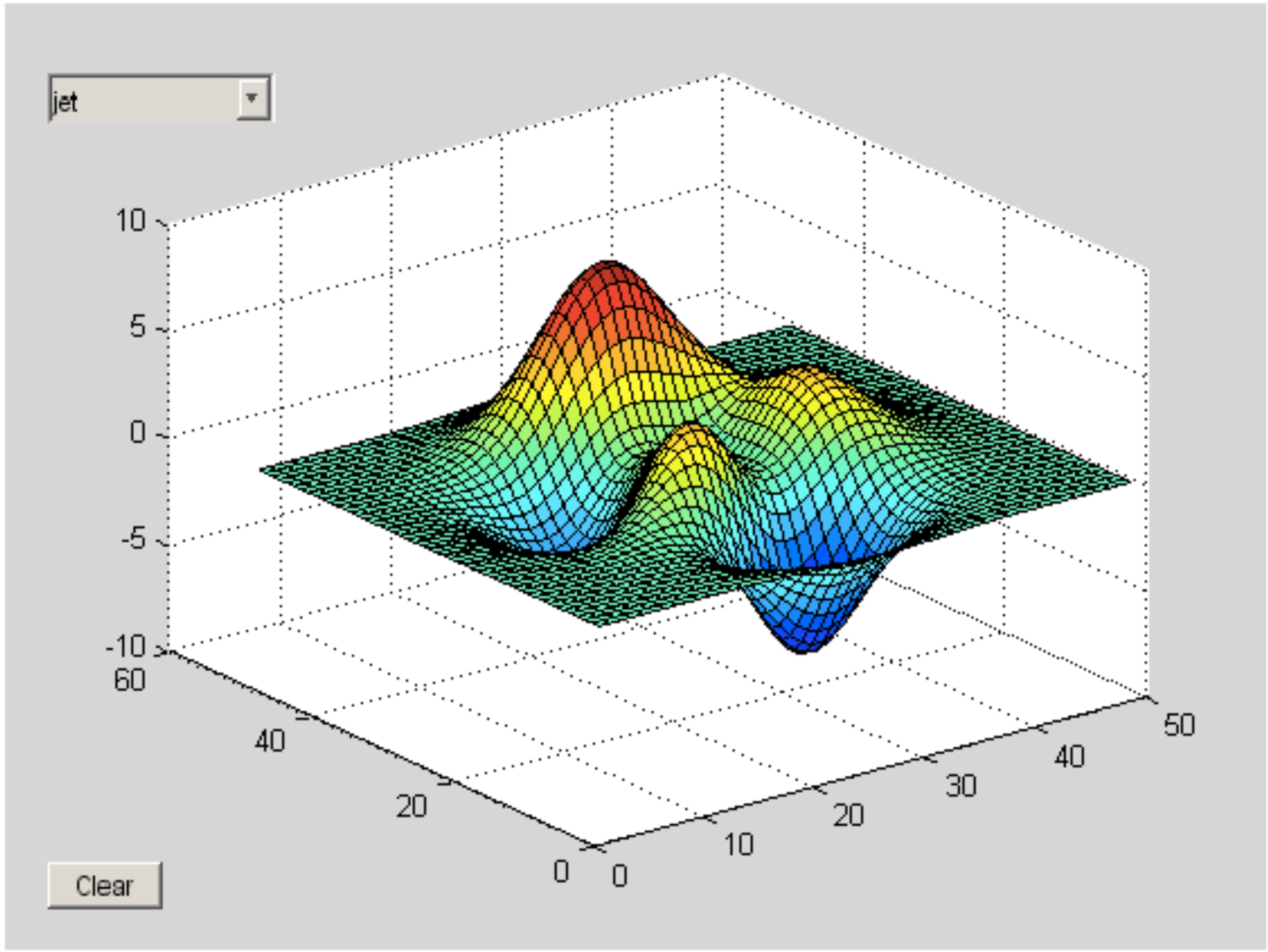
```
figure
hax = axes('Units','pixels');
surf(peaks)
```

Place a uicontrol object to let users change the colormap with a pop-up menu. Supply a function handle as the object's `Callback` property:

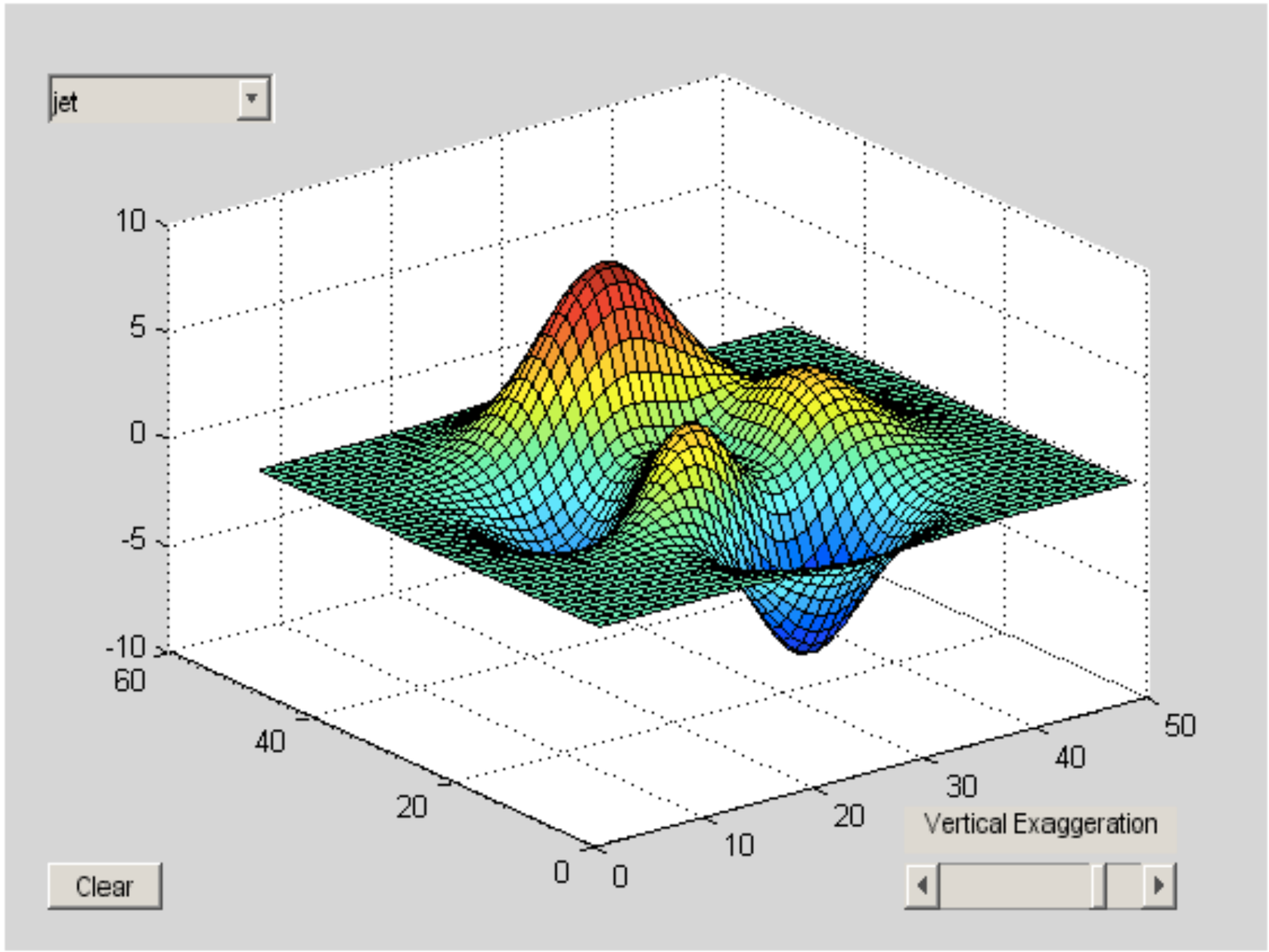
```
uicontrol('Style', 'popup',...
          'String', 'jet|hsv|hot|cool|gray',...
          'Position', [20 340 100 50],...
          'Callback', @setmap);       % Popup function handle callback
                                      % Implemented as a subfunction
```



```
uicontrol('Style', 'pushbutton', 'String', 'Clear',...
        'Position', [20 20 50 20],...
        'Callback', 'cla');          % Pushbutton string callback
                                     % that calls a MATLAB function
```



```
uicontrol('Style', 'slider',...
        'Min',1,'Max',50,'Value',41,...
        'Position', [400 20 120 20],...
        'Callback', {@surfzlim,hax});   % Uses cell array function handle callback
                                        % Implemented as a subfunction with an argument
uicontrol('Style','text',...
        'Position',[400 45 120 20],...
        'String','Vertical Exaggeration')
```



```
function setmap(hObj,event) %#ok<INUSD>
    % Called when user activates popup menu
    val = get(hObj,'Value');
    if val ==1
        colormap(jet)
    elseif val == 2
        colormap(hsv)
    elseif val == 3
        colormap(hot)
    elseif val == 4
        colormap(cool)
    elseif val == 5
        colormap(gray)
    end
end
```

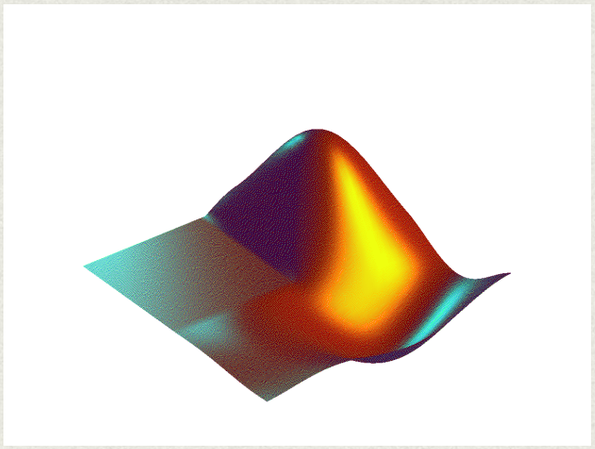
```
function surfzlim(hObj,event,ax) %#ok<INUSL>
    % Called to set zlim of surface in figure axes
    % when user moves the slider control
    val = 51 - get(hObj,'Value');
    zlim(ax,[-val val]);
end
```

Mouse Events

主要滑鼠事件

WindowButtonDownFcn：滑鼠按鈕按下時反應指令
WindowButtonMotionFcn：滑鼠移動時的反應指令
WindowButtonUpFcn：滑鼠按鈕釋放時的反應指令

```
set(gcf,’WindowButtonDownFcn','afun');
set(gcf,’WindowButtonMotionFcn’,'bfun');
set(gcf,’WindowButtonUpFcn’,'cfun');
```



Matlab進階功能介紹

實機演練與討論

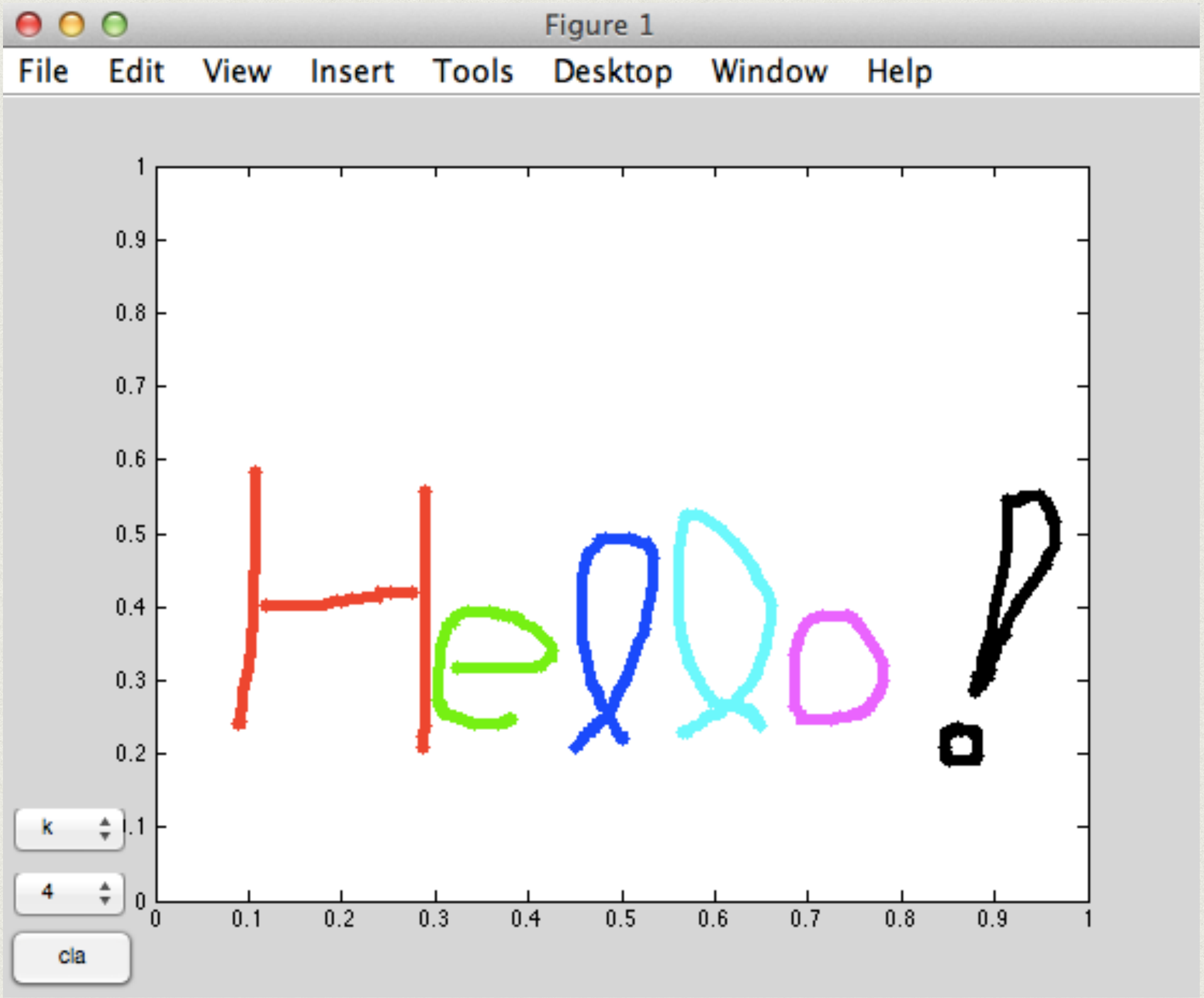
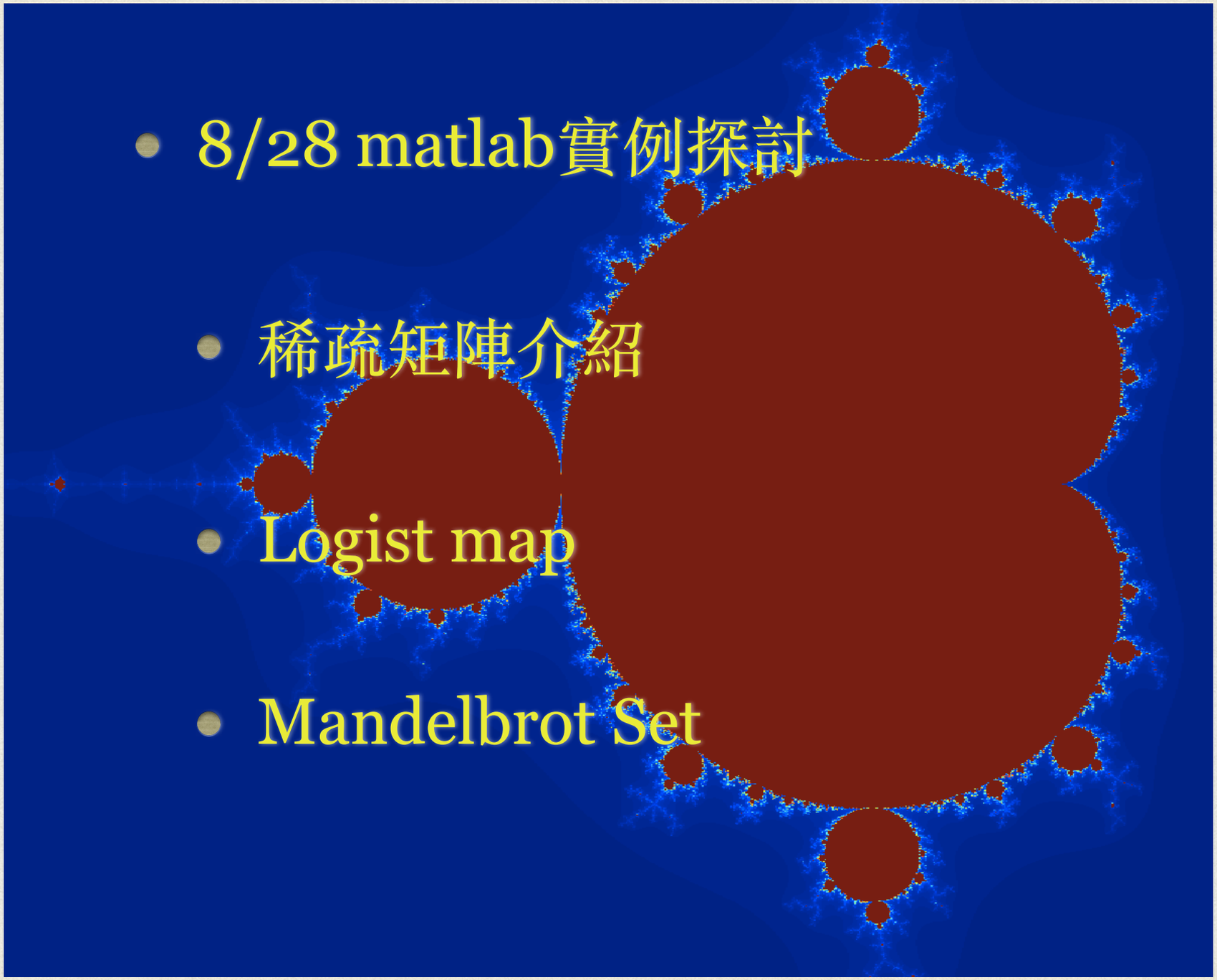


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
k
4
cla

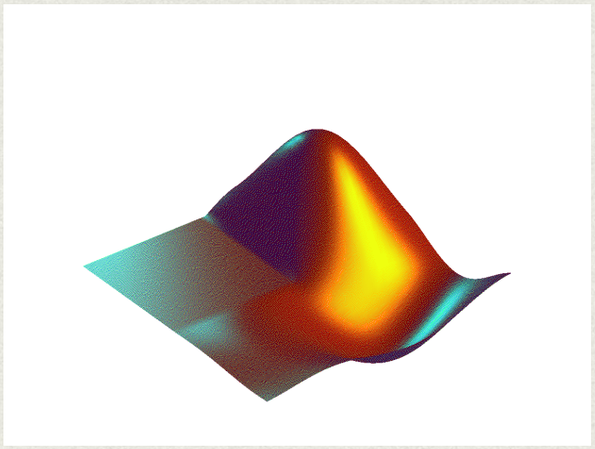
```
function paint(action)
persistent pt1;
persistent hpW;
persistent hpC;
if nargin == 0
   action = 'start';
end

switch(action)
   case 'start'
      figure(1);clf;
      axis([0 1 0 1])
      box on;
   hp=uicontrol('style','push',...
            'position',[5 5 60 30],...
            'string','cla',...
            'callback','cla');
   hpW=uicontrol('style','popup',...
            'position',[5 30 60 30],...
            'string', '1|2|3|4');
   hpC=uicontrol('style','popup',...
            'position',[5 60 60 30],...
            'string', 'r|g|b|c|m|k');
   set(gcf,'WindowButtonDownFcn','paint down');
case 'down'
   set(gcf,'WindowButtonMotionFcn','paint move');
   set(gcf,'WindowButtonUpFcn','paint up');
case 'move'
   paint('paint')
case 'up'
   paint('paint')
   set(gcf,'WindowButtonMotionFcn',[]);
   set(gcf,'WindowButtonUpFcn',[]);
   pt1=[];
case 'paint'
   cc='rgbcmk';
   pt=get(gca,'CurrentPoint');
   x=pt(1,1);y=pt(1,2);
   if isempty(pt1)
      plot(x,y,'linewidth',get(hpW,'value'),...
         'color',cc(get(hpC,'value')));hold on
   else
      plot([pt1(1) x],[pt1(2) y],'linewidth',get(hpW,'value'),...
         'color',cc(get(hpC,'value')));
   end
   pt1=[x,y];
   axis([0 1 0 1])
end
```

課程內容介紹



- 8/28 matlab實例探討
- 稀疏矩陣介紹
- Logist map
- Mandelbrot Set



Matlab進階功能介紹

稀疏矩陣介紹

根據元素的值，MATLAB 的矩陣可分為兩種：

- 完全矩陣（Full Matrix）
  每一個元素都存成 double 的資料型態，佔用 8 個 Byte
  一個 m×n 的完全矩陣，所佔用的記憶體空間是 8×m×n 個 Byte

- 稀疏矩陣（Sparse Matrix）
  大部分的元素都是 0
  只須儲存「非零元素的位置」及其「元素值」
  優點:
  節省記憶體儲存空間
  節省某一些不必要的運算

sparse 指令可將一個完全矩陣轉換成稀疏矩陣

範例5-1：sparse01.m

```
clear all;                              % 清除所有的變數
A = [2 0 0 0; 0 0 0 1; 0 4 0 0];        % 完全矩陣
S = sparse(A)                           % 將完全矩陣 A 轉換成稀疏矩陣 S
```

```
S =
   (1,1)        2
   (3,2)        4
   (2,4)        1
```

可看出，S 是一個稀疏矩陣，MATLAB 只儲存其各個非零元素的位置（即其二維下標 (1,1)、(3,2)、(2,4)）和元素值（2、4、1）

比較矩陣 A 和 S 所佔用的記憶體大小：

```
>> whos
 Name    Size            Bytes   Class      Attributes
 A       3x4                96      double
 S       3x4                88      double    sparse
```

看出稀疏矩陣 S 佔用記憶體的位元組數目（88 Bytes）比 矩陣A（佔用 96 Bytes）還要小。

使用 sparse 指令來直接產生稀疏矩陣

S = sparse(i, j, s, m, n);

- i 是列索引
- j 是行索引
- s 是非零元素所形成的向量
- m 是 s 的列維度
- n 是 s 的行維度
- i、j、s 都是長度相同的向量
- s(k) 的二維下標即是 i(k)及 j(k)

Elementary Sparse Matrices

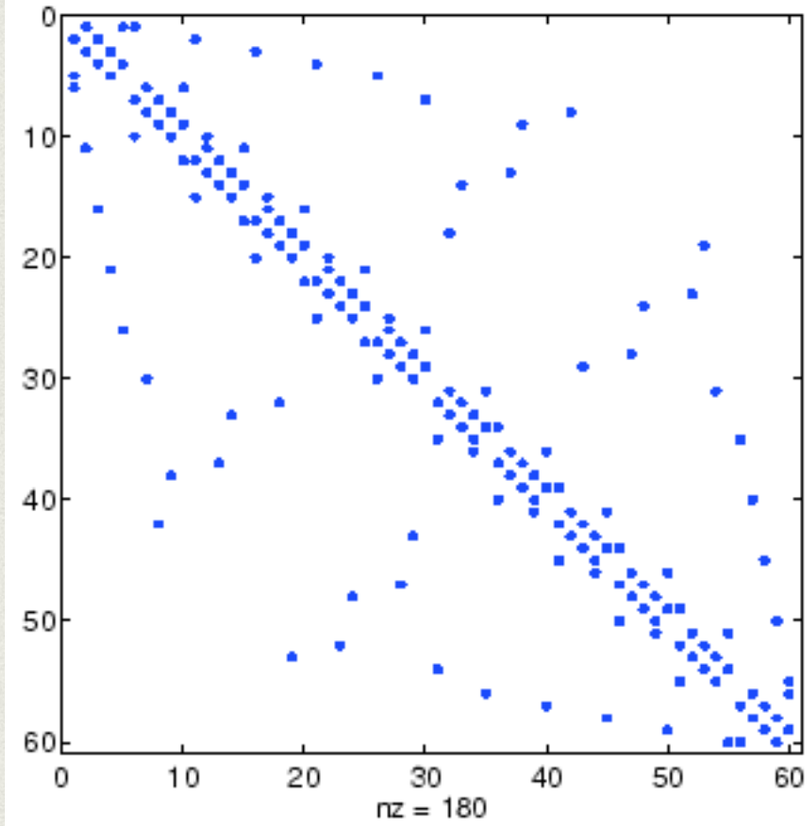
spdiags	Extract and create sparse band and diagonal matrices
speye	Sparse identity matrix
sprand	Sparse uniformly distributed random matrix
sprandn	Sparse normally distributed random matrix
sprandsym	Sparse symmetric random matrix

Full to Sparse Conversion

find	Find indices and values of nonzero elements
full	Convert sparse matrix to full matrix
sparse	Create sparse matrix
spconvert	Import matrix from sparse matrix external format

spy

```
B = bucky;
spy(B)
```



spconvert

Suppose the ASCII file `uphill.dat` contains

```
     1     1     1.000000000000000
     1     2     0.500000000000000
     2     2     0.333333333333333
     1     3     0.333333333333333
     2     3     0.250000000000000
     3     3     0.200000000000000
     1     4     0.250000000000000
     2     4     0.200000000000000
     3     4     0.166666666666667
     4     4     0.142857142857143
     4     4     0.000000000000000
```

Then the statements

```
load uphill.dat
H = spconvert(uphill)

H =
    (1,1)        1.0000
    (1,2)        0.5000
    (2,2)        0.3333
    (1,3)        0.3333
    (2,3)        0.2500
    (3,3)        0.2000
    (1,4)        0.2500
    (2,4)        0.2000
    (3,4)        0.1667
    (4,4)        0.1429
```

```
load airfoil
```

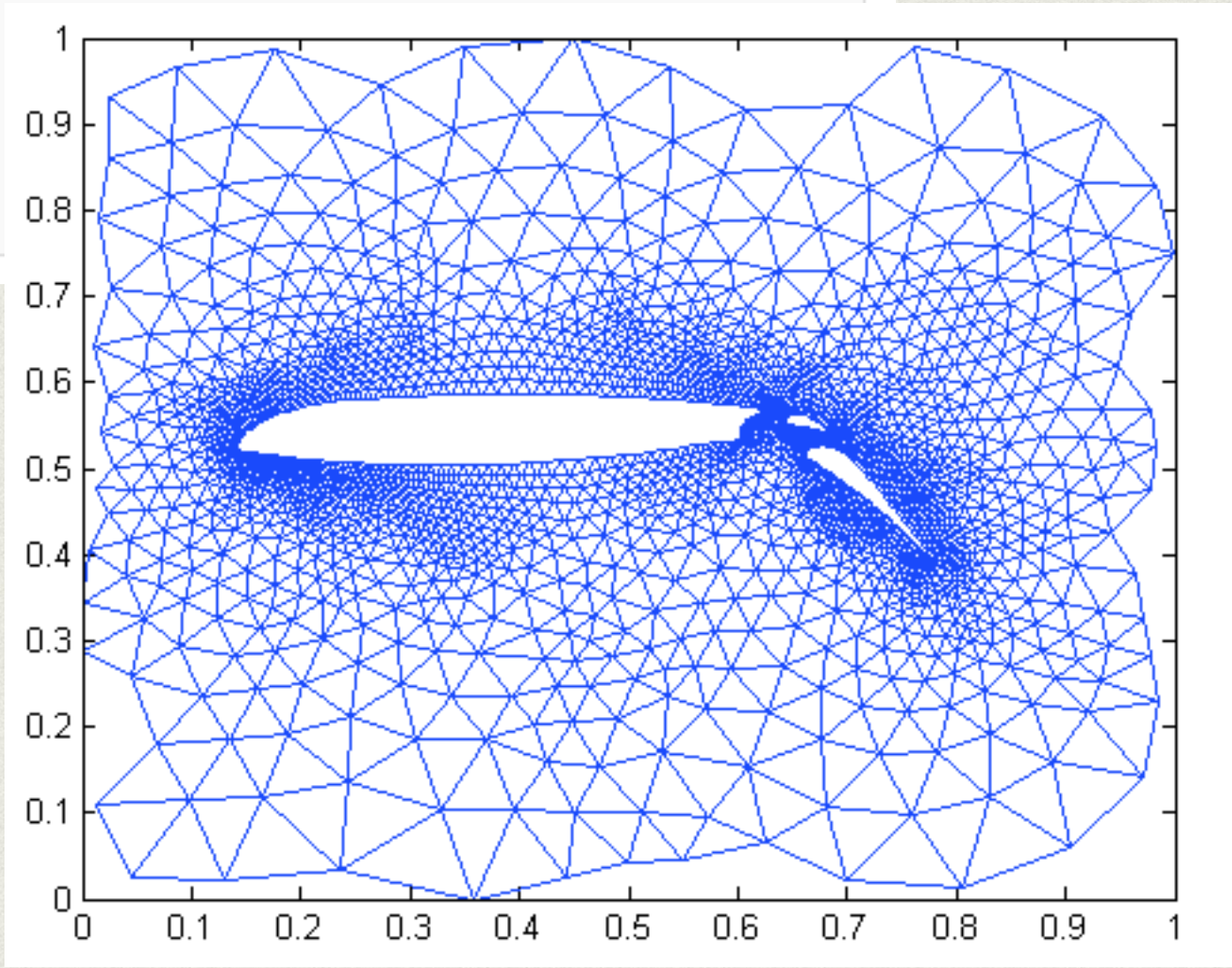
The Finite Element Mesh

First, scale x and y by 2^(-32) to bring them into the range [0,1]. Then form the sparse adjacency matrix and make it positive definite.

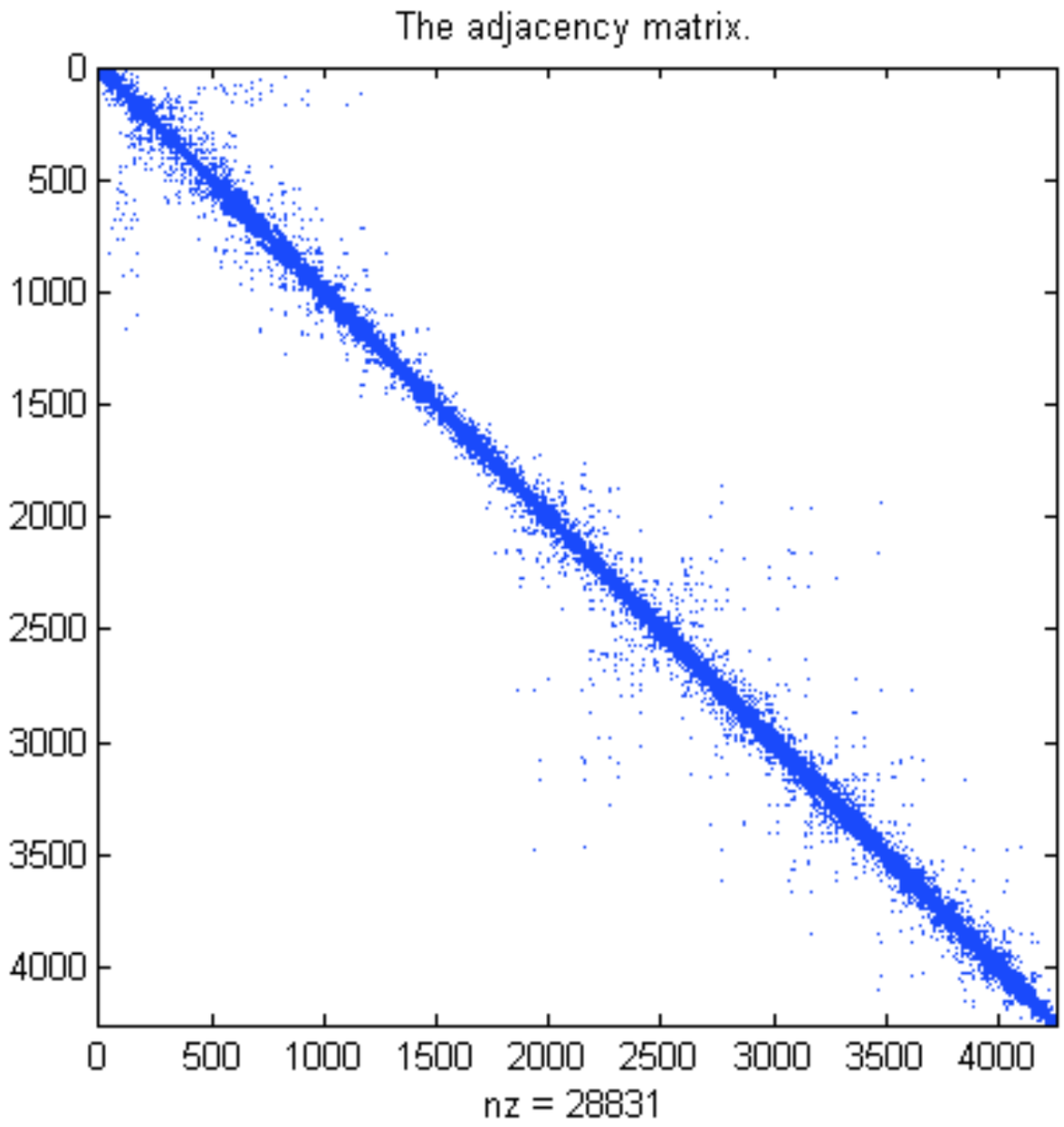
```
% Scaling x and y
x = pow2(x,-32);
y = pow2(y,-32);

% Forming the sparse adjacency matrix and making it positive definite
n = max(max(i),max(j));
A = sparse(i,j,-1,n,n);
A = A + A';
d = abs(sum(A)) + 1;
A = A + diag(sparse(d));

% Plotting the finite element mesh
gplot(A,[x y])
```



```
spy(A)
title('The adjacency matrix.')
```



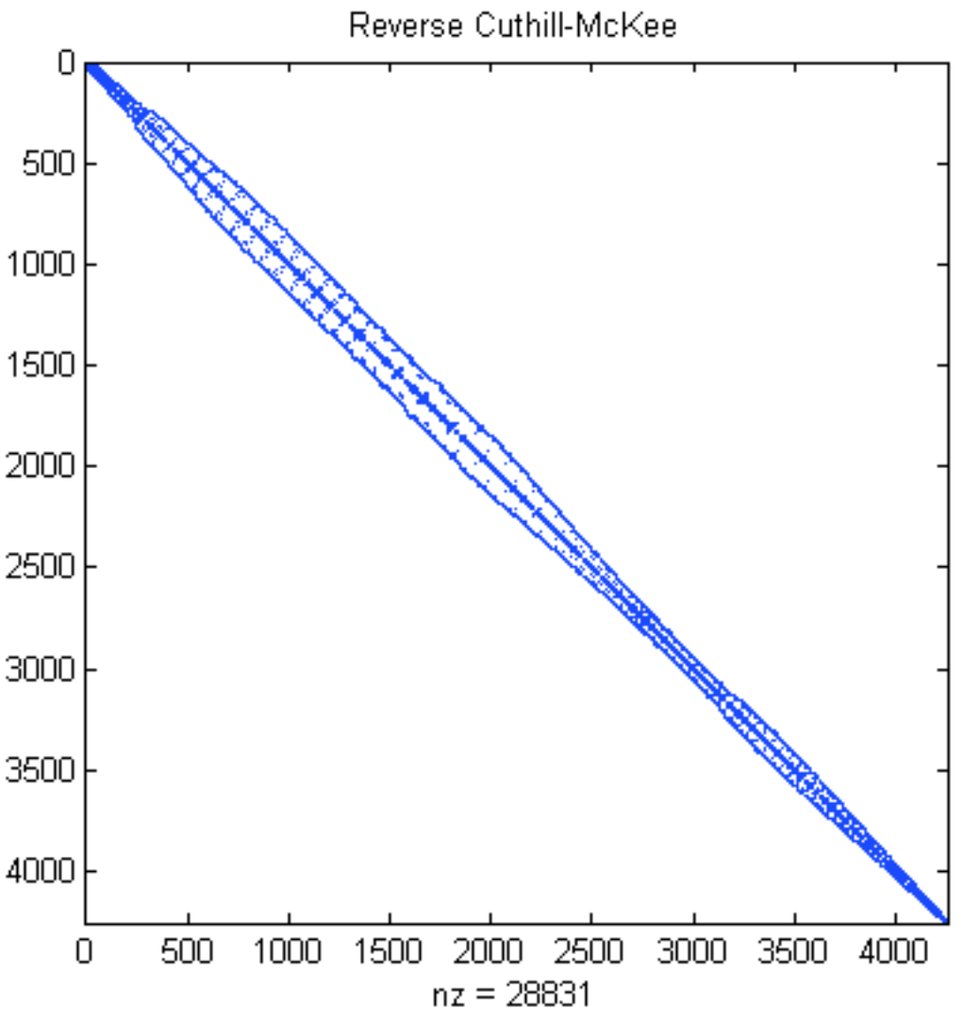
Reordering Algorithms

amd	Approximate minimum degree permutation
colamd	Column approximate minimum degree permutation
colperm	Sparse column permutation based on nonzero count
dmperm	Dulmage-Mendelsohn decomposition
ldl	Block LDL' factorization for Hermitian indefinite matrices
randperm	Random permutation
symamd	Symmetric approximate minimum degree permutation
symrcm	Sparse reverse Cuthill-McKee ordering

Symmetric Reordering - Reverse Cuthill-McKee

SYMRCM uses the Reverse Cuthill-McKee technique for reordering the adjacency matrix. r = SYMRCM(A) returns a permutation vector r such that A(r,r) tends to have its diagonal elements closer to the diagonal than A. This is a good preordering for LU or Cholesky factorization of matrices that come from "long, skinny" problems. It works for both symmetric and asymmetric A.

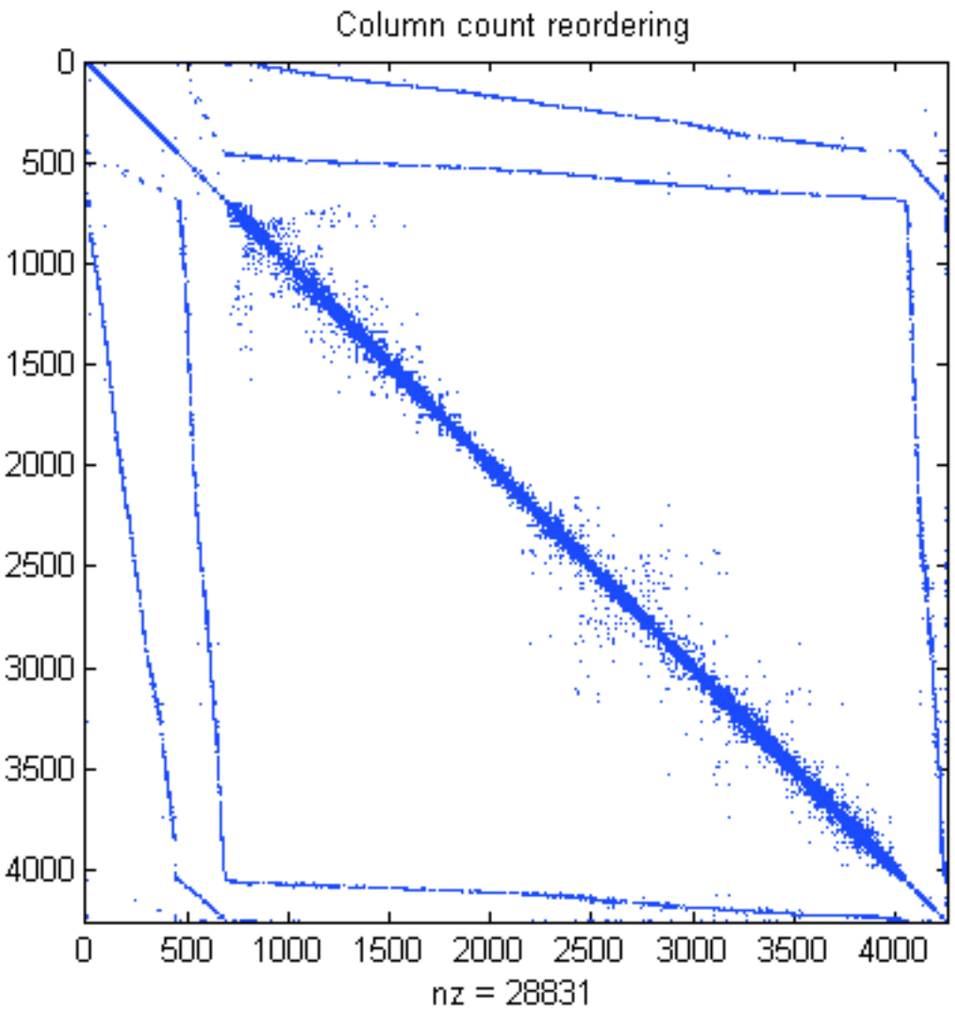
```
r = symrcm(A);
spy(A(r,r))
title('Reverse Cuthill-McKee')
```



Symmetric Reordering - COLPERM

Use j = COLPERM(A) to return a permutation vector that reorders the columns of the sparse matrix A in non-decreasing order of non-zero count. This is sometimes useful as a preordering for LU factorization: lu(A(:,j)).

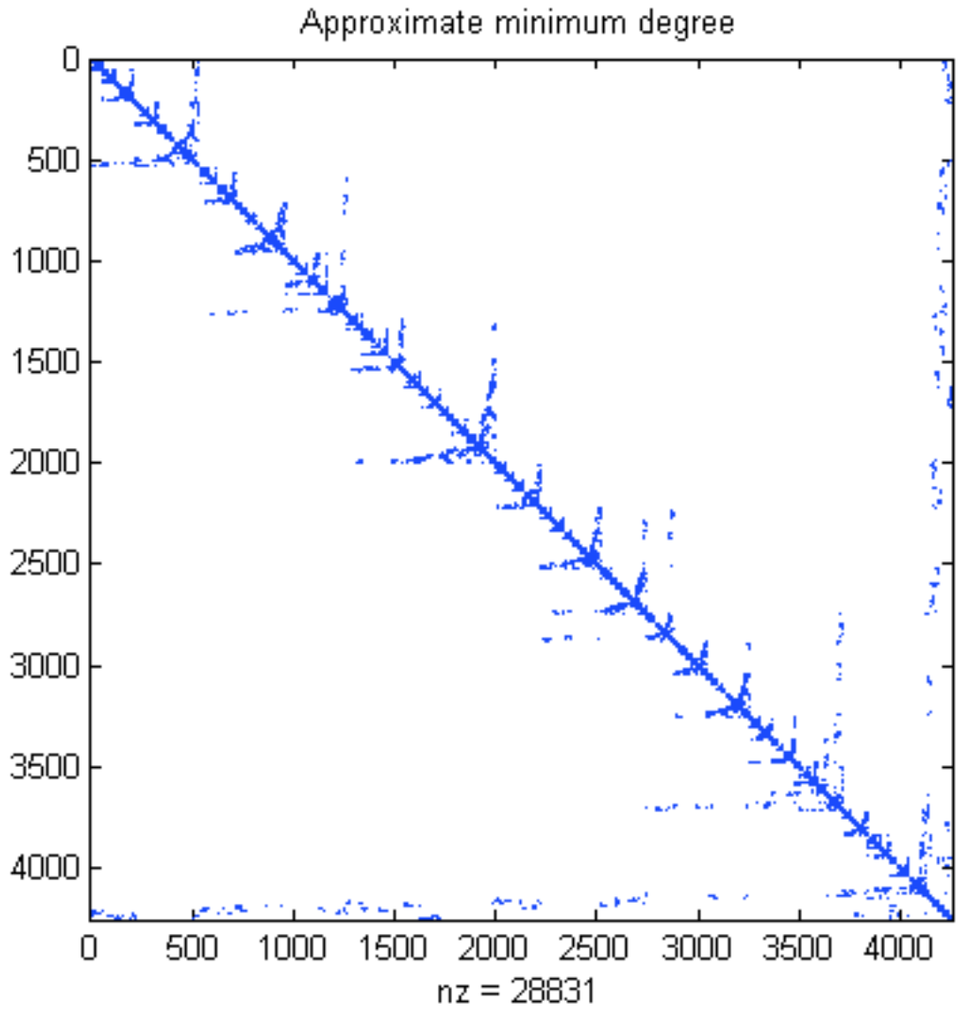
```
j = colperm(A);
spy(A(j,j))
title('Column count reordering')
```



Symmetric Reordering - SYMAMD

SYMAMD gives a symmetric approximate minimum degree permutation. p = SYMAMD(S), for a symmetric positive definite matrix A, returns the permutation vector p such that S(p,p) tends to have a sparser Cholesky factor than S. Sometimes SYMAMD works well for symmetric indefinite matrices too.

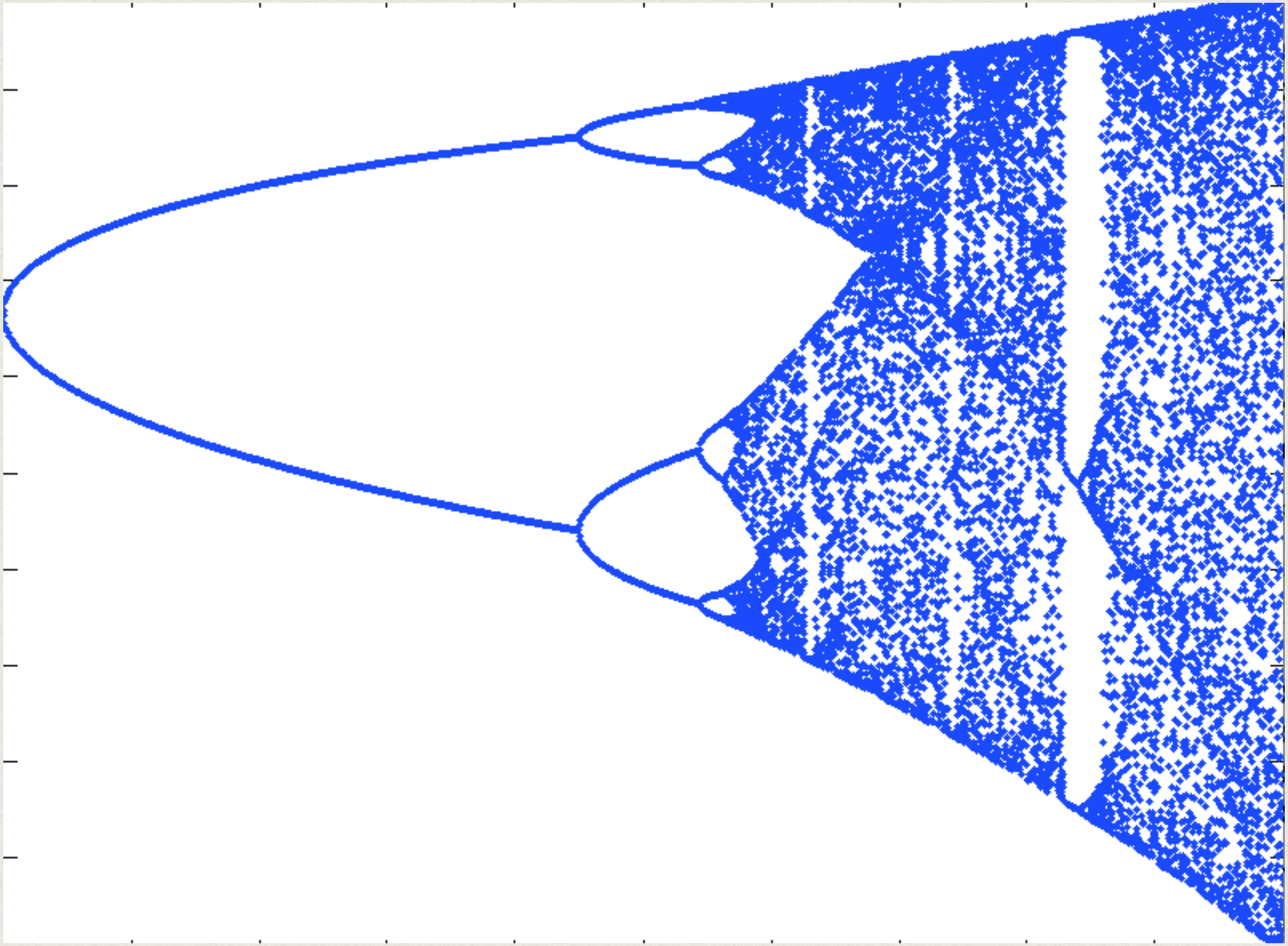
```
m = symamd(A);
spy(A(m,m))
title('Approximate minimum degree')
```



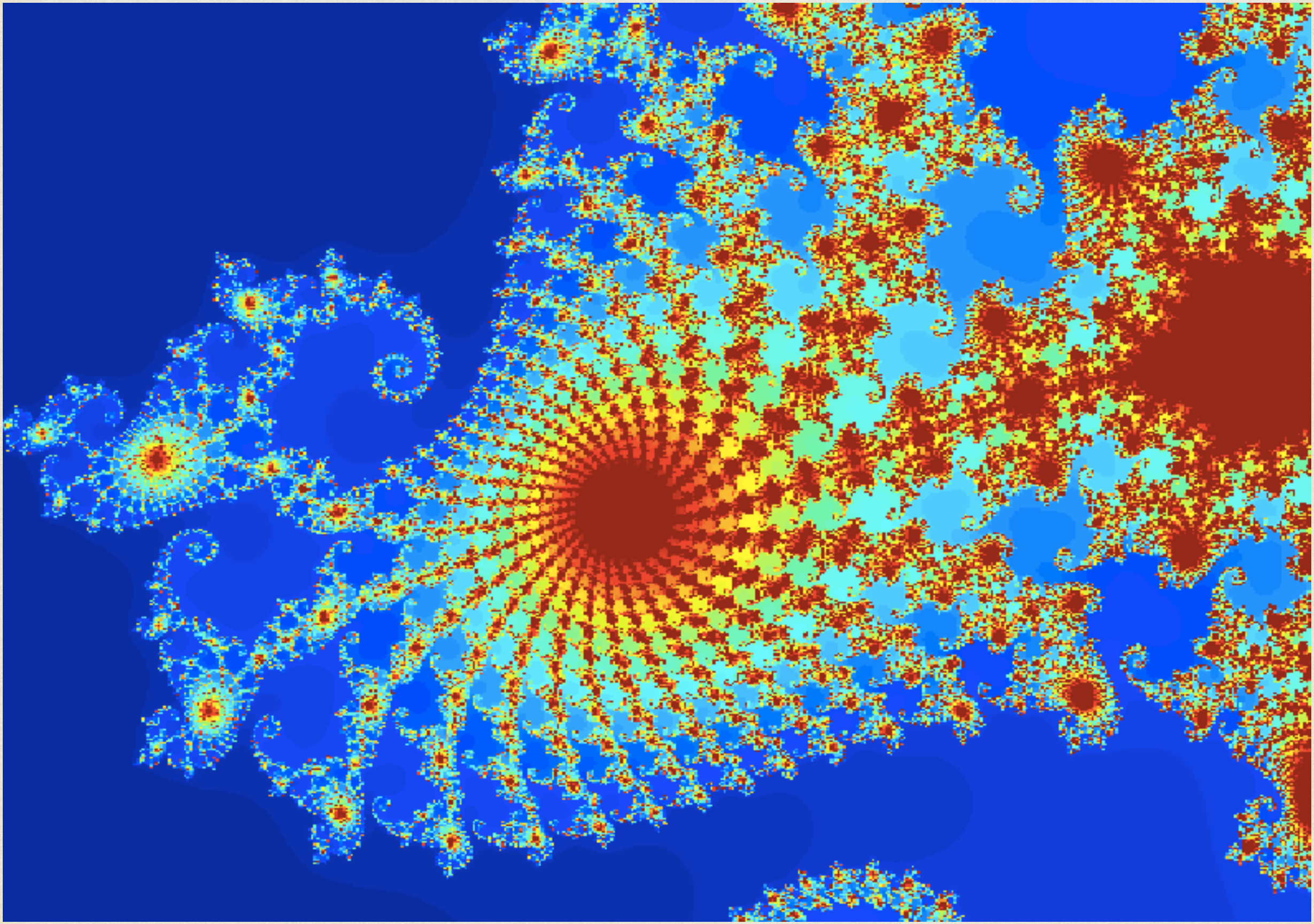
Linear Equations (Iterative Methods)

bicg	Biconjugate gradients method
bicgstab	Biconjugate gradients stabilized method
bicgstabl	Biconjugate gradients stabilized (l) method
cgs	Conjugate gradients squared method
gmres	Generalized minimum residual method (with restarts)
lsqr	LSQR method
minres	Minimum residual method
pcg	Preconditioned conjugate gradients method
qmr	Quasi-minimal residual method
symmlq	Symmetric LQ method
tfqmr	Transpose-free quasi-minimal residual method

LOGIST MAP



MANDELBROT SET



Formal definition [edit]

The Mandelbrot set M is defined by a family of complex quadratic polynomials

$$P_c : \mathbb{C} \to \mathbb{C}$$

given by

$$P_c : z \mapsto z^2 + c,$$

where c is a complex parameter. For each c, one considers the behavior of the sequence

$$(0, P_c(0), P_c(P_c(0)), P_c(P_c(P_c(0))), \ldots)$$

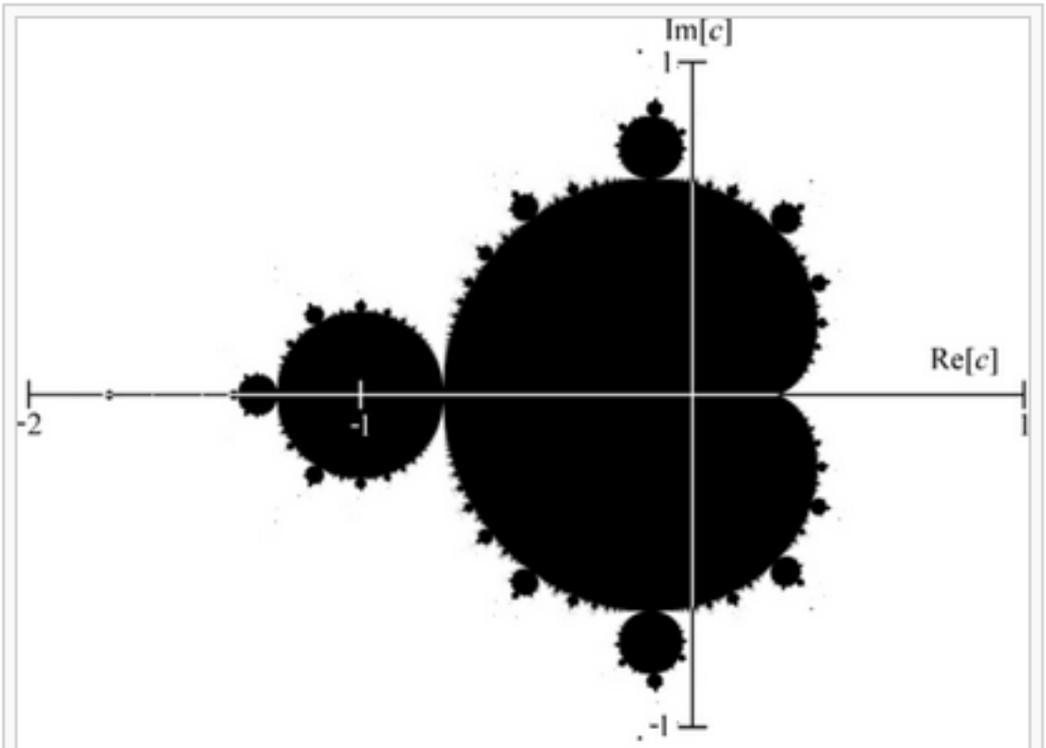
obtained by iterating $P_c(z)$ starting at critical point $z = 0$, which either escapes to infinity or stays within a disk of some finite radius. The Mandelbrot set is defined as the set of all points c such that the above sequence does *not* escape to infinity.

More formally, if $P_c^n(z)$ denotes the *n*th iterate of $P_c(z)$ (i.e. $P_c(z)$ composed with itself *n* times), the Mandelbrot set is the subset of the complex plane given by

$$M = \{c \in \mathbb{C} : \exists s \in \mathbb{R}, \forall n \in \mathbb{N}, |P_c^n(0)| \leq s\}\,.$$

As explained below, it is in fact possible to simplify this definition by taking $s = 2$.

Mathematically, the Mandelbrot set is just a set of complex numbers. A given complex number *c* either belongs to *M* or it does not. A picture of the Mandelbrot set can be made by colouring all the points c that belong to *M* black, and all other points white. The more colourful pictures usually seen are generated by colouring points not in the set according to which term in the sequence $|P_c^n(0)|$ is the first term with an absolute value greater than a certain cutoff value, usually 2. See the section on computer drawings below for more details.



A mathematician's depiction of the Mandelbrot set *M*. A point *c* is coloured black if it belongs to the set, and white if not. Re[c] and Im[c] denote the real and imaginary parts of *c*, respectively.

The Mandelbrot set can also be defined as the connectedness locus of the family of polynomials $P_c(z)$. That is, it is the subset of the complex plane consisting of those parameters c for which the Julia set of P_c is connected.

Thanks