# Matlab Workshop MFE 2006
## Lecture 1

Stefano Corradin

Peng Liu

http://faculty.haas.berkeley.edu/peliu/computing

# Introduction:

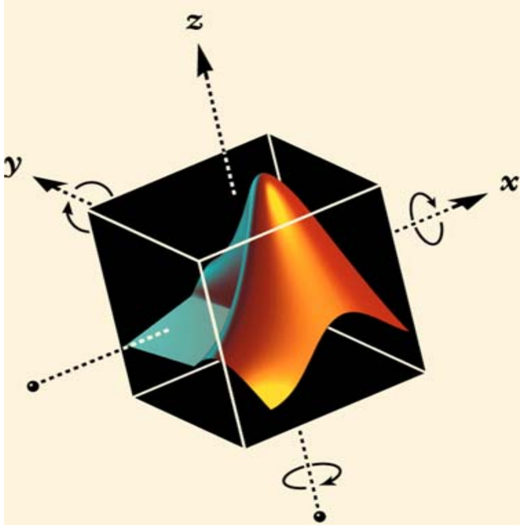
- **Peng Liu:** peliu@haas.berkeley.edu (1)
- **Stefano Corradin:** corradin@haas.berkeley.edu (2-4)

- **The MathWorks documentation page**

  **http://www.mathworks.com/access/helpdesk/help/helpdesk.html**

Download Materials:

http://faculty.haas.berkeley.edu/peliu/computing



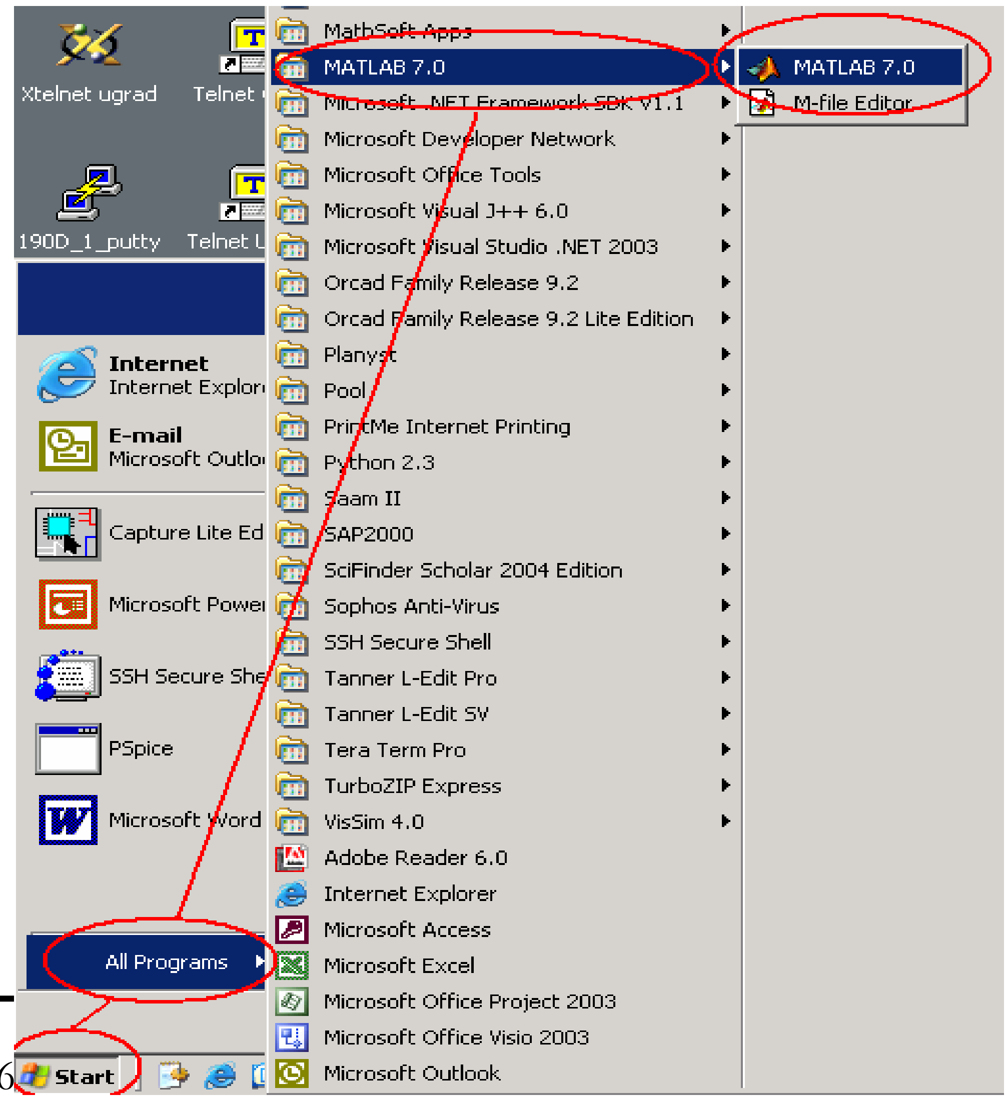
# What is MatLab?

- What is MATLAB ?
  - MATLAB is a computer program that combines computation and visualization power that makes it particularly useful for engineers.
  - MATLAB is an executive program, and a script can be made with a list of MATLAB commands like other programming language.
- MATLAB Stands for MATrix LABoratory.
  - The system was designed to make matrix computation particularly easy.
- The MATLAB environment allows the user to:
  - manage variables
  - import and export data
  - perform calculations
  - generate plots
  - develop and manage files for use with MATLAB.

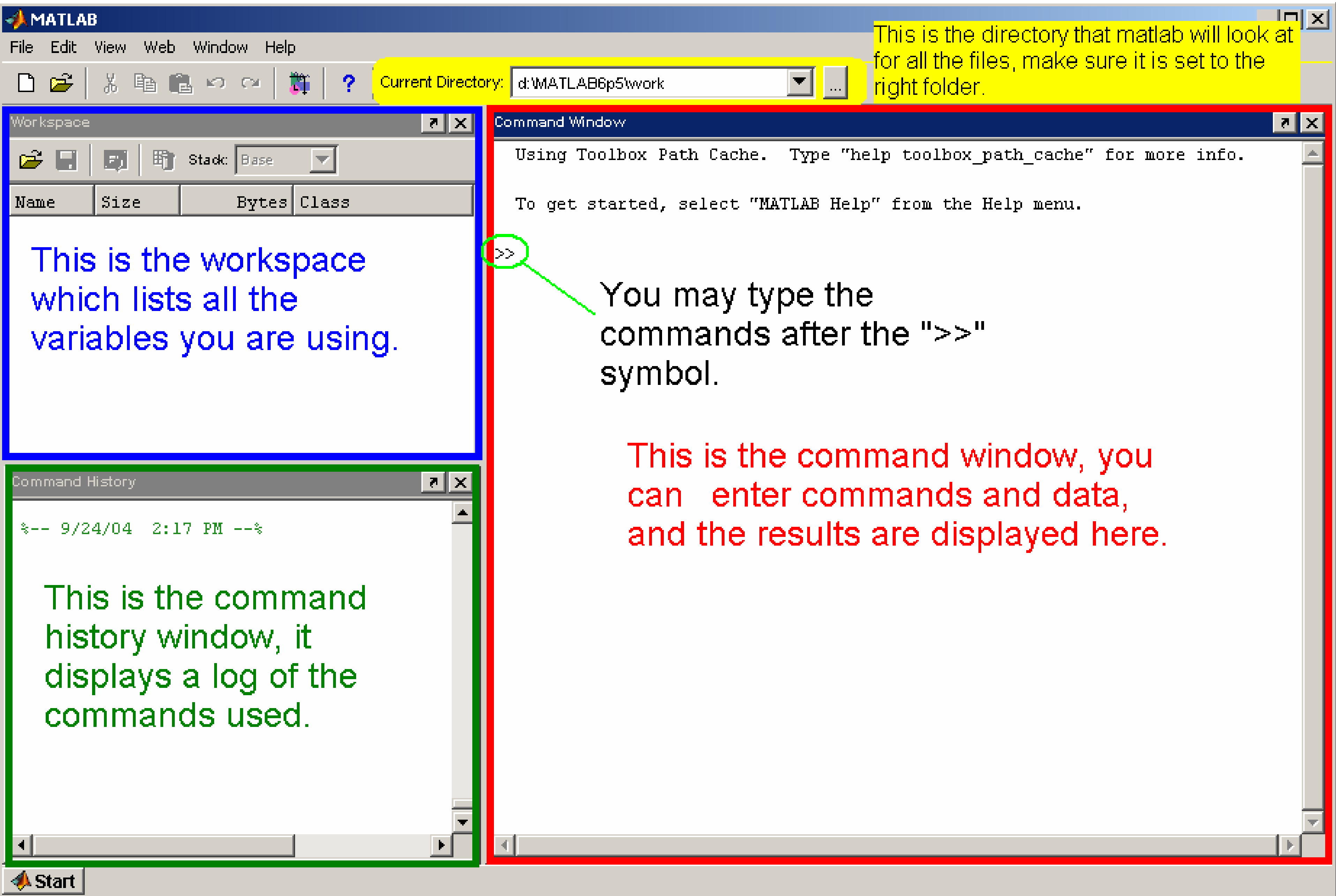
# MATLAB Environment

To start MATLAB:

START ➔ PROGRAMS
➔ PhD & MFE
Applications ➔
MATLAB 7.1



# Display Windows



# Display Windows (con't…)

- Graphic (Figure) Window
  - Displays plots and graphs
  - Created in response to graphics commands.
- M-file editor/debugger window
  - Create and edit scripts of commands called M-files.

# Getting Help

- type one of following commands in the command window:
  - **help** – lists all the help topic
  - **help** *topic* – provides help for the specified topic
  - **help** *command* – provides help for the specified command
    - **help help** – provides information on use of the help command
  - **helpwin** – opens a separate help window for navigation
  - **lookfor** *keyword* – Search all M-files for *keyword*
- Google “MATLAB helpdesk”
- Go to the online HelpDesk provided by www.mathworks.com

# Basic Syntax

- Variables
- Vectors
- Array Operations
- Matrices
- Solutions to Systems of Linear Equations.

# Variables

- Variable names:
  - Must start with a letter
  - May contain only letters, digits, and the underscore "_"
  - Matlab is case sensitive, i.e. one & OnE are different variables.
  - Matlab only recognizes the first 31 characters in a variable name.
- Assignment statement:
  - *Variable = number;*
  - *Variable = expression;*
- Example:

```
>> A = 1234;
>> a = 1234
a =
     1234
```

NOTE: when a semi-colon ";" is placed at the end of each command, the result is not displayed.

# Variables (con't…)

- Special variables:
  - **ans** : default variable name for the result
  - **pi**: $\pi$ = 3.1415926…………
  - **eps**: $\in$ = 2.2204e-016, smallest amount by which 2 numbers can differ.
  - **Inf** or **inf** : $\infty$, infinity
  - **NaN** or **nan**: not-a-number
- Commands involving variables:
  - **who**: lists the names of defined variables
  - **whos**: lists the names and sizes of defined variables
  - **clear**: clears all varialbes, reset the default values of special variables.
  - **clear** *name*: clears the variable *name*
  - **clc**: clears the command window
  - **clf**: clears the current figure and the graph window.

# Vectors

- A row vector in MATLAB can be created by an explicit list, starting with a left bracket, entering the values separated by spaces (or commas) and closing the vector with a right bracket.
- A column vector can be created the same way, and the rows are separated by semicolons.
- To input a matrix, you basically define a variable. For a matrix the form is:

  variable name = [#, #, #; #, #, #; #, #, #;…..]

- Example: 1st row 2nd row 3rd row

```
>> x = [ 0   0.25*pi   0.5*pi   0.75*pi   pi ]
x =
       0    0.7854    1.5708    2.3562    3.1416
>> y = [ 0; 0.25*pi; 0.5*pi; 0.75*pi; pi ]
y =
       0
  0.7854
  1.5708
  2.3562
  3.1416
```

x is a row vector.

y is a column vector.

# Vectors (con't…)

- Vector Addressing – A vector element is addressed in MATLAB with an integer index enclosed in parentheses.
- Example:

```
>> x(3)
ans =
   1.5708
```

← 3rd element of vector x

- The colon notation may be used to address a block of elements.

(start : increment : end)

start is the starting index, increment is the amount to add to each successive index, and end is the ending index. A shortened format (start : end) may be used if increment is 1.

- Example:

```
>> x(1:3)
ans =
        0    0.7854    1.5708
```

← 1st to 3rd elements of vector x

NOTE: MATLAB index starts at 1.

# Vectors (con't…)

## Some useful commands:

| | |
|---|---|
| x = start:end | create row vector x starting with start, counting by one, ending at end |
| x = start:increment:end | create row vector x starting with start, counting by increment, ending at or before end |
| linspace(start,end,number) | create row vector x starting with start, ending at end, having number elements |
| length(x) | returns the length of vector x |
| y = x' | transpose of vector x |
| dot (x, y) | returns the scalar dot product of the vector x and y. |

# Array Operations

- **Scalar-Array Mathematics**

For addition, subtraction, multiplication, and division of an array by a scalar simply apply the operations to all elements of the array.

- Example:

```
>> f = [ 1 2; 3 4]
f =
      1     2
      3     4
>> g = 2*f – 1
g =
      1     3
      5     7
```

Each element in the array f is multiplied by 2, then subtracted by 1.

# Array Operations (con't…)

- **Element-by-Element Array-Array Mathematics.**

| ***Operation*** | ***Algebraic Form*** | ***MATLAB*** |
|---|---|---|
| Addition | a + b | a + b |
| Subtraction | a – b | a – b |
| Multiplication | a x b | a .* b |
| Division | a ÷ b | a ./ b |
| Exponentiation | $a^b$ | a .^ b |

- **Example:**

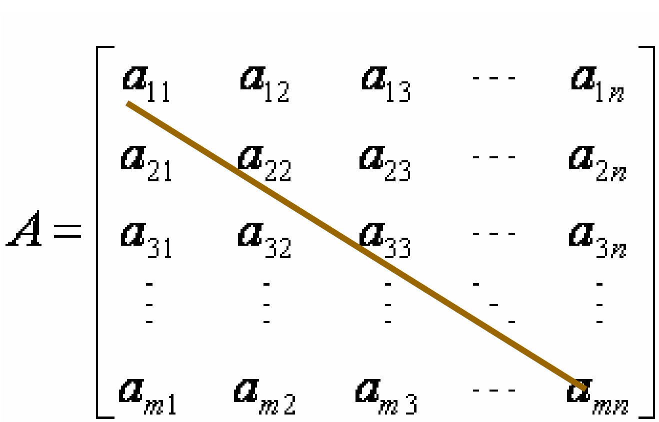
```
>> x = [ 1 2 3 ];
>> y = [ 4 5 6 ];
>> z = x .* y
z =
      4     10     18
```

Each element in x is multiplied by the corresponding element in y.

# Matrices

- A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:
  - it begins with [, and end with ]
  - spaces or commas are used to separate elements in a row
  - semicolon or enter is used to separate rows.

A is an m x n matrix.



the main diagonal

•Example:

```
>> f = [ 1 2 3; 4 5 6]
f =
     1     2     3
     4     5     6
>> h = [ 2 4 6
1 3 5]
h =
     2     4     6
     1     3     5
```

# Matrices (con't…)

- Matrix Addressing:
  -- *matrixname(row, column)*
  -- **colon** may be used in place of a row or column reference to select the entire row or column.

- Example:

```
>> f(2,3)

ans =

     6

>> h(:,1)

ans =

     2
     1
```

recall:

```
f =
    1    2    3
    4    5    6
h =
    2    4    6
    1    3    5
```

# Matrices (con't…)

## Some useful commands:

| | |
|---|---|
| zeros(n) | returns a n x n matrix of zeros |
| zeros(m,n) | returns a m x n matrix of zeros |
| ones(n) | returns a n x n matrix of ones |
| ones(m,n) | returns a m x n matrix of ones |
| size (A) | for a m x n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix. |
| length(A) | returns the larger of the number of rows or columns in A. |

# Matrices (con't…)

## more commands

| | |
|---|---|
| Transpose | B = A' |
| Identity Matrix | eye(n) ➔ returns an n x n identity matrix<br>eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere. |
| Addition and subtraction | C = A + B<br>C = A – B |
| Scalar Multiplication | B = $\alpha$A, where $\alpha$ is a scalar. |
| Matrix Multiplication | C = A*B |
| Matrix Inverse | B = inv(A), A must be a square matrix in this case.<br>rank (A) ➔ returns the rank of the matrix A. |
| Matrix Powers | B = A.^2 ➔ squares each element in the matrix<br>C = A * A ➔ computes A*A, and A must be a square matrix. |
| Determinant | det (A), and A must be a square matrix. |

A, B, C are matrices, and m, n, $\alpha$ are scalars.

# Solutions to Systems of Linear Equations

- Example: a system of 3 linear equations with 3 unknowns ($x_1$, $x_2$, $x_3$):

$$
\begin{aligned}
3x_1 + 2x_2 - x_3 &= 10 \\
-x_1 + 3x_2 + 2x_3 &= 5 \\
x_1 - x_2 - x_3 &= -1
\end{aligned}
$$

Let :

$$A = \begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad b = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Then, the system can be described as:

$$Ax = b$$

# Solutions to Systems of Linear Equations (con't…)

- Solution by Matrix Inverse:
  $Ax = b$
  $A^{-1}Ax = A^{-1}b$
  $x = A^{-1}b$
- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = inv(A)*b
x =
  -2.0000
   5.0000
  -6.0000
```

Answer:
$x_1 = -2, x_2 = 5, x_3 = -6$

- Solution by Matrix Division:
  The solution to the equation
  $$Ax = b$$
  can be computed using left division.
- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = A\b
x =
  -2.0000
   5.0000
  -6.0000
```

Answer:
$x_1 = -2, x_2 = 5, x_3 = -6$

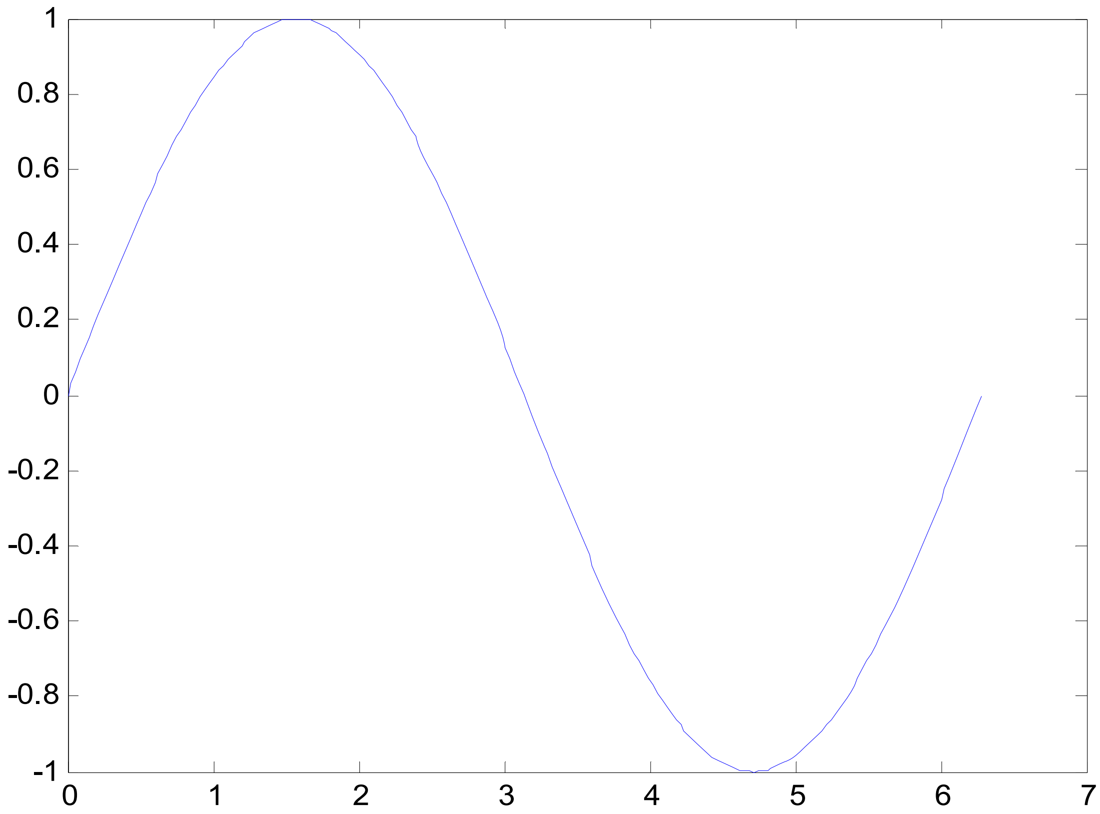
NOTE:
left division: A\b ➔ b ÷ A
right division: x/y ➔ x ÷ y

# Plotting in Matlab

- Goal: plot $y = \sin(x)$
- Matlab code

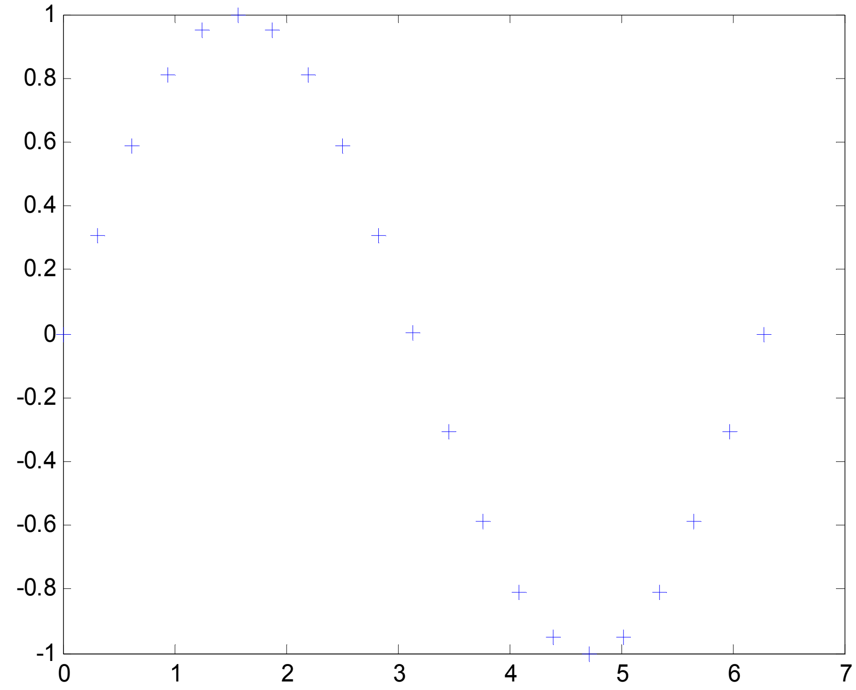
```
xplot = (0 : 0.01 : 2)*pi;

yplot = sin(xplot);
plot(xplot, yplot)
```

# Plotting in Matlab (cont.)



# Plotting points

```
xpts = (0 : 0.1 : 2)*pi;  % 21 evenly spaced points
ypts = sin(xpts);
plot(xpts, ypts, '+')
```

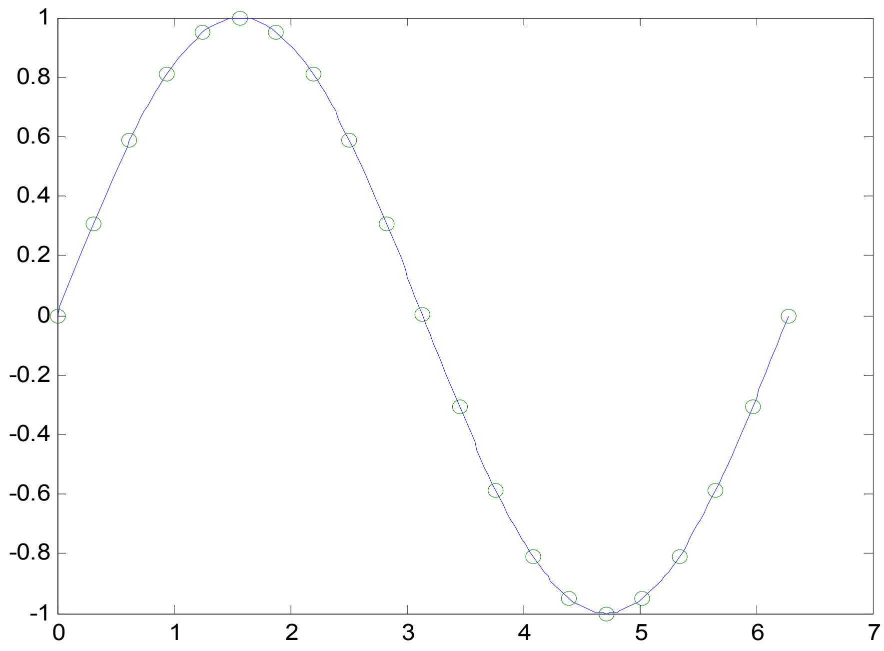



Type `help plot` to see point specification options in addition to '+'

# Plotting more than one thing

- Option 1: inside one `plot` command

```
plot(xplot, yplot, xpts, ypts, 'o')
```

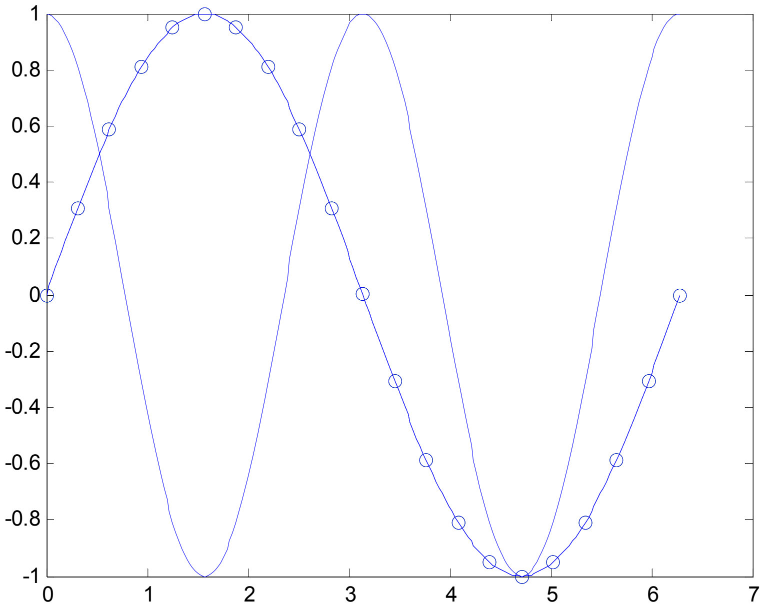


# Plotting more than one thing

- Option 2: using `hold on`, `hold off`

Add plot of $y = \cos(2x)$

```
yplot2 = cos(2*xplot);
hold on
plot(xplot, yplot)
plot(xpts, ypts, 'o')
plot(xplot, yplot2)
hold off
```



# Adding color to plots

```
clf
xplot = (0 : 0.01 : 2)*pi;
yplot = sin(xplot);

xpts = (0 : 0.1 : 2)*pi;
ypts = sin(xpts);

yplot2 = cos(2 * xplot);

hold on
plot(xplot, yplot, 'r')  % y = sin(x), red line
plot(xpts, ypts, 'ko')   % y = sin(x), black circles
plot(xplot, yplot2, 'g') % y = cos(2x), green line
hold off
```

Type `help plot` to see color options

# Plotting (con't…)

- Plotting Curves:
  - **plot (x,y)** – generates a linear plot of the values of x (horizontal axis) and y (vertical axis).
  - **semilogx (x,y)** – generate a plot of the values of x and y using a logarithmic scale for x and a linear scale for y
  - **semilogy (x,y)** – generate a plot of the values of x and y using a linear scale for x and a logarithmic scale for y.
  - **loglog(x,y)** – generate a plot of the values of x and y using logarithmic scales for both x and y
- Multiple Curves:
  - **plot (x, y, w, z) –** multiple curves can be plotted on the same graph by using multiple arguments in a plot command. The variables x, y, w, and z are vectors. Two curves will be plotted: y vs. x, and z vs. w.
  - **legend ('string1', 'string2',…)** – used to distinguish between plots on the same graph
    - exercise: type **help legend** to learn more on this command.
- Multiple Figures:
  - **figure (n)** – used in creation of multiple plot windows. place this command before the plot() command, and the corresponding figure will be labeled as "Figure n"
  - **close** – closes the figure n window.
  - **close all** – closes all the figure windows.
- Subplots:
  - **subplot (m, n, p)** – m by n grid of windows, with p specifying the current plot as the $p^{th}$ window

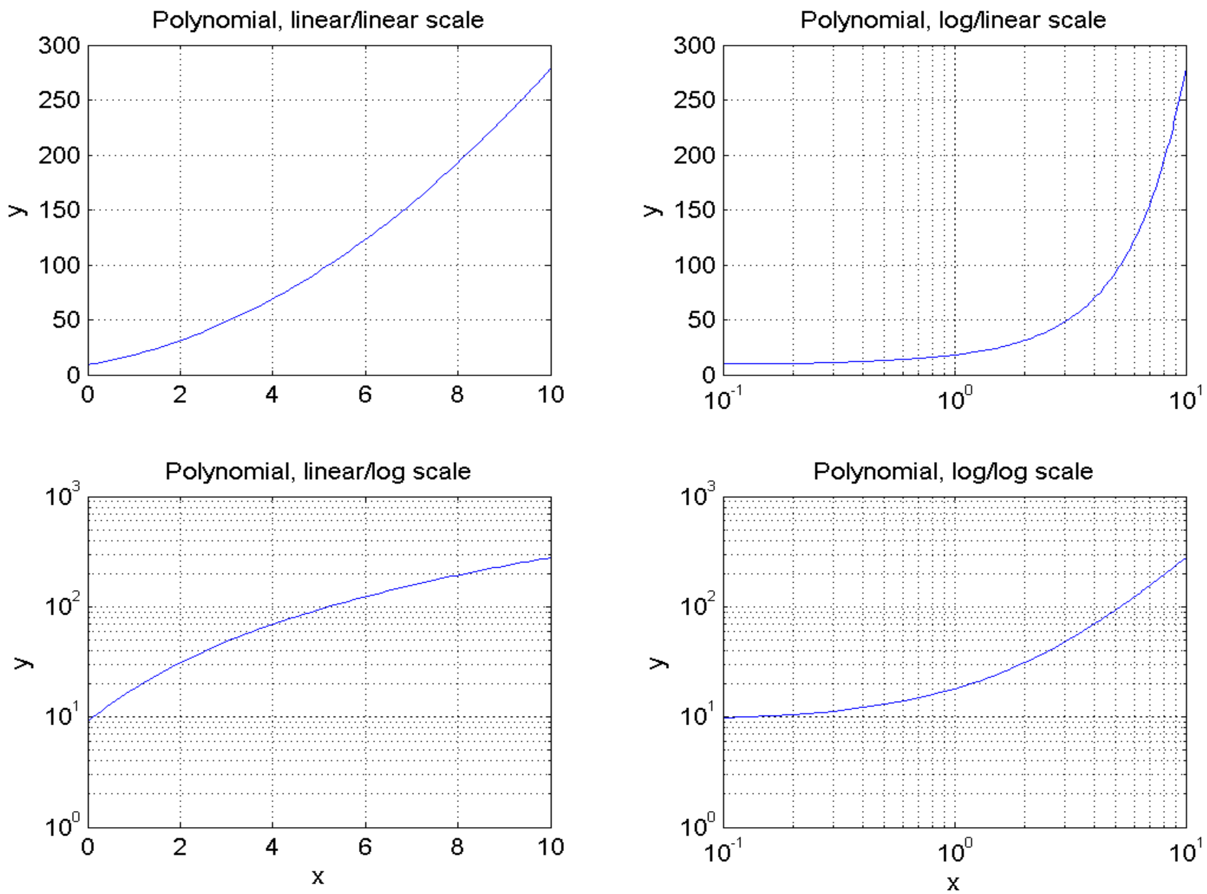
# Plotting (con't…)

- **<u>Example: (polynomial function</u>)**
  plot the polynomial using linear/linear scale, log/linear scale, linear/log scale, & log/log scale:

$$y = 2x^2 + 7x + 9$$

```
% Generate the polynomial:
x = linspace (0, 10, 100);
y = 2*x.^2 + 7*x + 9;

% plotting the polynomial:
figure (1);
subplot (2,2,1), plot (x,y);
title ('Polynomial, linear/linear scale');
ylabel ('y'), grid;
subplot (2,2,2), semilogx (x,y);
title ('Polynomial, log/linear scale');
ylabel ('y'), grid;
subplot (2,2,3), semilogy (x,y);
title ('Polynomial, linear/log scale');
xlabel('x'), ylabel ('y'), grid;
subplot (2,2,4), loglog (x,y);
title ('Polynomial, log/log scale');
xlabel('x'), ylabel ('y'), grid;
```

# Plotting (con't…)



# Plotting (con't…)

- Adding new curves to the existing graph:
- Use the **hold** command to add lines/points to an existing plot.
  - hold on – retain existing axes, add new curves to current axes. Axes are rescaled when necessary.
  - hold off – release the current figure window for new plots
- Grids and Labels:

| Command | Description |
|---|---|
| grid on | Adds dashed grids lines at the tick marks |
| grid off | removes grid lines (default) |
| grid | toggles grid status (off to on, or on to off) |
| title ('text') | labels top of plot with text in quotes |
| xlabel ('text') | labels horizontal (x) axis with text is quotes |
| ylabel ('text') | labels vertical (y) axis with text is quotes |
| text (x,y,'text') | Adds text in quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot. |

# Additional commands for plotting

color of the point or curve

| Symbol | Color |
|---|---|
| y | yellow |
| m | magenta |
| c | cyan |
| r | red |
| g | green |
| b | blue |
| w | white |
| k | black |

Marker of the data points

| Symbol | Marker |
|---|---|
| . | ● |
| o | ○ |
| x | × |
| + | + |
| * | ∗ |
| s | □ |
| d | ◊ |
| v | ∇ |
| ^ | Δ |
| h | hexagram |

Plot line styles

| Symbol | Line Style |
|---|---|
| – | solid line |
| : | dotted line |
| –. | dash-dot line |
| – – | dashed line |

# Flow control - selection

- The if-elseif-else construction

  if <logical expression>

      <commands>

  elseif <logical expression>

      <commands>

  else

      <commands>

  end

# Logical expressions (try help)

- Relational operators (compare arrays of same sizes)
- == (equal to)  ~= (not equal)
  < (less than)  <= (less than or equal to)
  > (greater than)  >= (greater than or equal to)

- Logical operators (combinations of relational operators)
- & (and)
  | (or)
  ~ (not)

- Logical functions
  xor
  isempty
  any
  all

# M-Files

*So far, we have executed the commands in the command window. But a more practical way is to create a M-file.*

- The M-file is a text file that consists a group of MATLAB commands.
- MATLAB can open and execute the commands exactly as if they were entered at the MATLAB command window.
- To run the M-files, just type the file name in the command window. (make sure the current working directory is set correctly)

# Scripts or function: when use what?

- Functions
  - Take inputs, generate outputs, have internal variables
  - Solve general problem for arbitrary parameters
- Scripts
  - Operate on global workspace
  - Document work, design experiment or test
  - Solve a very specific problem once

# User-Defined Function

- Add the following command in the beginning of your m-file:

**function** [output variables] = **function_name** (input variables);

NOTE: the function_name should be the same as your file name to avoid confusion.

- calling your function:
  -- a user-defined function is called by the name of the m-file, ***<u>not</u>*** the name given in the function definition.
  -- type in the m-file name like other pre-defined commands.

- Comments:
  -- The first few lines should be comments, as they will be displayed if help is requested for the function name.  the first comment line is reference by the lookfor command.

# Branching-IF ELSEIF (example)

- Type ***a=2, if a>1,b=1,else b=0,end***
- Or make a m-file (script) named aa.m

```
a=11
if a>10
    b=2
elseif a>1
    b=1
else b=0
end
```

type.m

```
% example of branching for type of options
K=105
if S==K
    disp('At the Money Option')
elseif S > K
    disp('In the Money Option')
else
    disp('Out the Money Option')
end
```

- Give a stock price ***S=125***; enter ***type*** in command window

# Flow control - repetition

- Repeats a code segment a fixed number of times

```
for index=<vector>
   <statements>
end
```

- The <statements> are executed repeatedly.
  At each iteration, the variable index is assigned a new value from <vector>.
- Example: CRR Binomial Model

# Flow control – conditional repetition

- while-loops

```
while <logical expression>
  <statements>
End
```

- <statements> are executed repeatedly as long as the <logical expression> evaluates to true

# Flow control – conditional repetition

- Solutions to nonlinear equations

$$f(x) = 0$$

- can be found using Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- **Task**: write a function that finds a solution to

$$f(x) = e^{-x} - \sin(x)$$

- Given $x_0$, iterate `maxit` times or until $|x_n - x_{n-1}| \leq \text{tol}$

# Flow control – conditional repetition

newton.m

```
function [x,n] = newton(x0,tol,maxit)
% NEWTON - Newton's method for solving equations
% [x,n] = NEWTON(x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
  n = n + 1;
  x_new = x - (exp(-x)-sin(x))/(-exp(-x)-cos(x));
  done=(n>=maxit) | ( abs(x_new-x)<tol );
  x=x_new;
end
```

- `>> [x,n]=newton(0,eps,10)`

# Black Vol using Newton Method

- Result:

```
x =
    0.5885
n =
     6
```

- **Question**: code a function that produce Black-Scholes Volatility from Option prices!!

# Function functions

- Do we need to re-write `newton.m` for every new function?
- No! General purpose functions take other m-files as input.

```
>> help feval
>> [f,f_prime]=feval('myfun',0);
```

**myfun.m**

```
function [f,f_prime] = myfun(x)
% MYFUN- Evaluate f(x) = exp(x)-sin(x)
% and its first derivative
% [f,f_prime] = myfun(x)

f=exp(-x)-sin(x);
f_prime=-exp(-x)-cos(x);
```

# Function functions

newtonf.m

- Can update newton.m

```
function [x,n] = newtonf(fname,x0,tol,maxit)
% NEWTON – Newton’s method for solving equations
% [x,n] = NEWTON(fname,x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
  n = n + 1;
  [f,f_prime]=feval(fname,x);
  x_new = x – f/f_prime;
  done=(n>maxit) | ( abs(x_new-x)<tol );
  x=x_new;
end
```

  - `>> [x,n]=newtonf(’myfun’,0,1e-3,10)`

# Example: Pricing options in CRR Binomial

- Open P:\PodiumPC\2006MFE
- Double Click CRR.m
- It will open an editor window beginning with

```
function [] = CRR(CallPut, AssetP, Strike, RiskFree, Div, Time, Vol, nSteps)
% Computes the Cox, Ross & Rubinstein (1979) Binomial Tree for European
%Call/Put Option Values based on the following inputs:
% CallPut     =    Call = 1, Put = 0
% AssetP      =    Underlying Asset Price
% Strike      =    Strike Price of Option
% RiskFree    =    Risk Free rate of interest annualized eg. 0.05
% Div         =    Dividend Yield of Underlying
% Time        =    Time to Maturity in years
% Vol         =    Volatility of the Underlying
% nSteps      =    Number of Time Steps for Binomial Tree to take
```

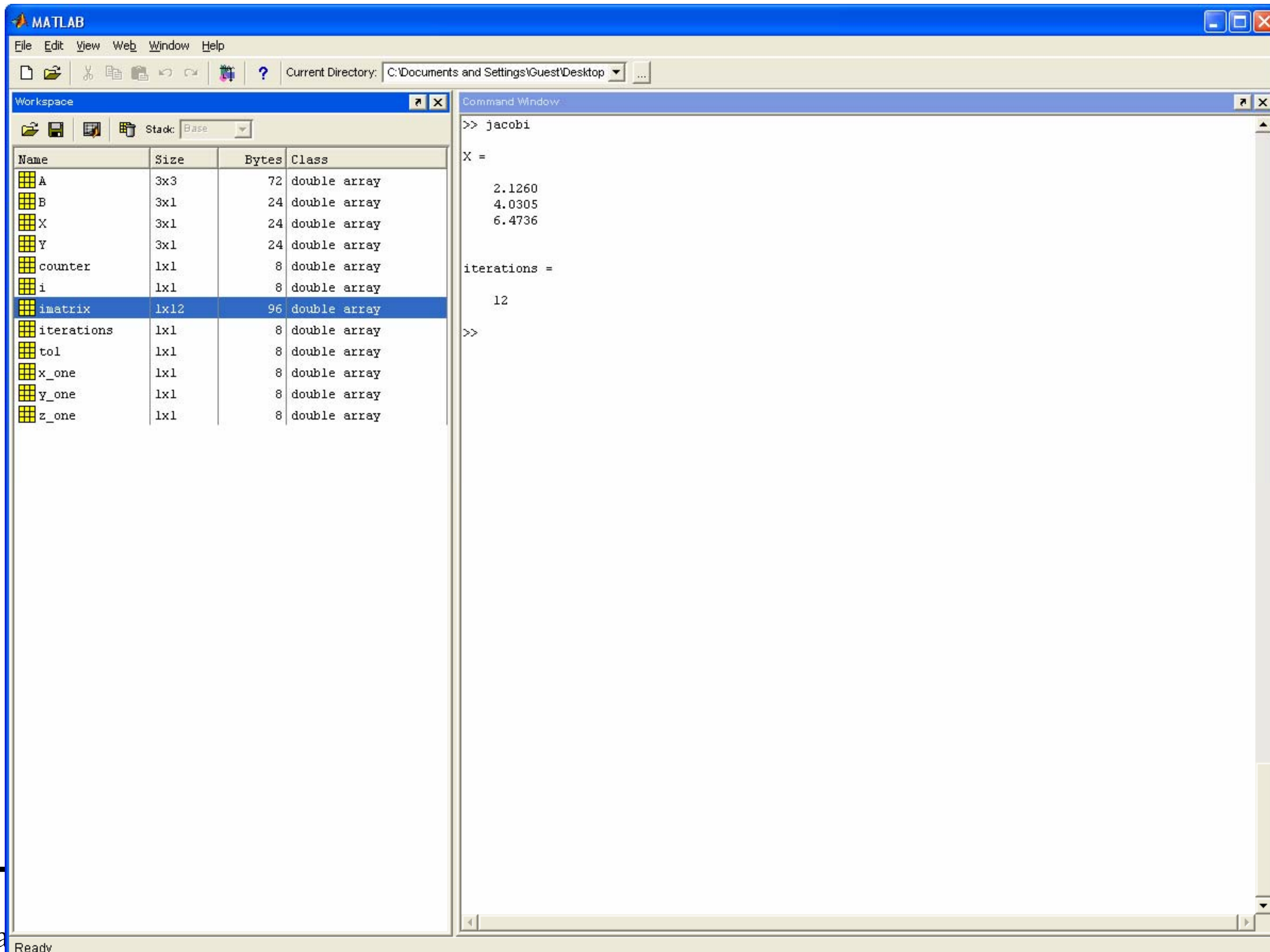
# Pricing options in CRR Binomial (cont.)

```
dt = Time / nSteps;

if CallPut
    b = 1;
end
if ~CallPut
    b = -1;
end

RR = exp(RiskFree * dt);
Up = exp(Vol * sqrt(dt));
Down = 1 / Up;
Q_up = (exp((RiskFree - Div) * dt) - Down) / (Up - Down);
Q_down = 1 - Q_up;
Df = exp(-RiskFree * dt); %Df: Discount Factor
```

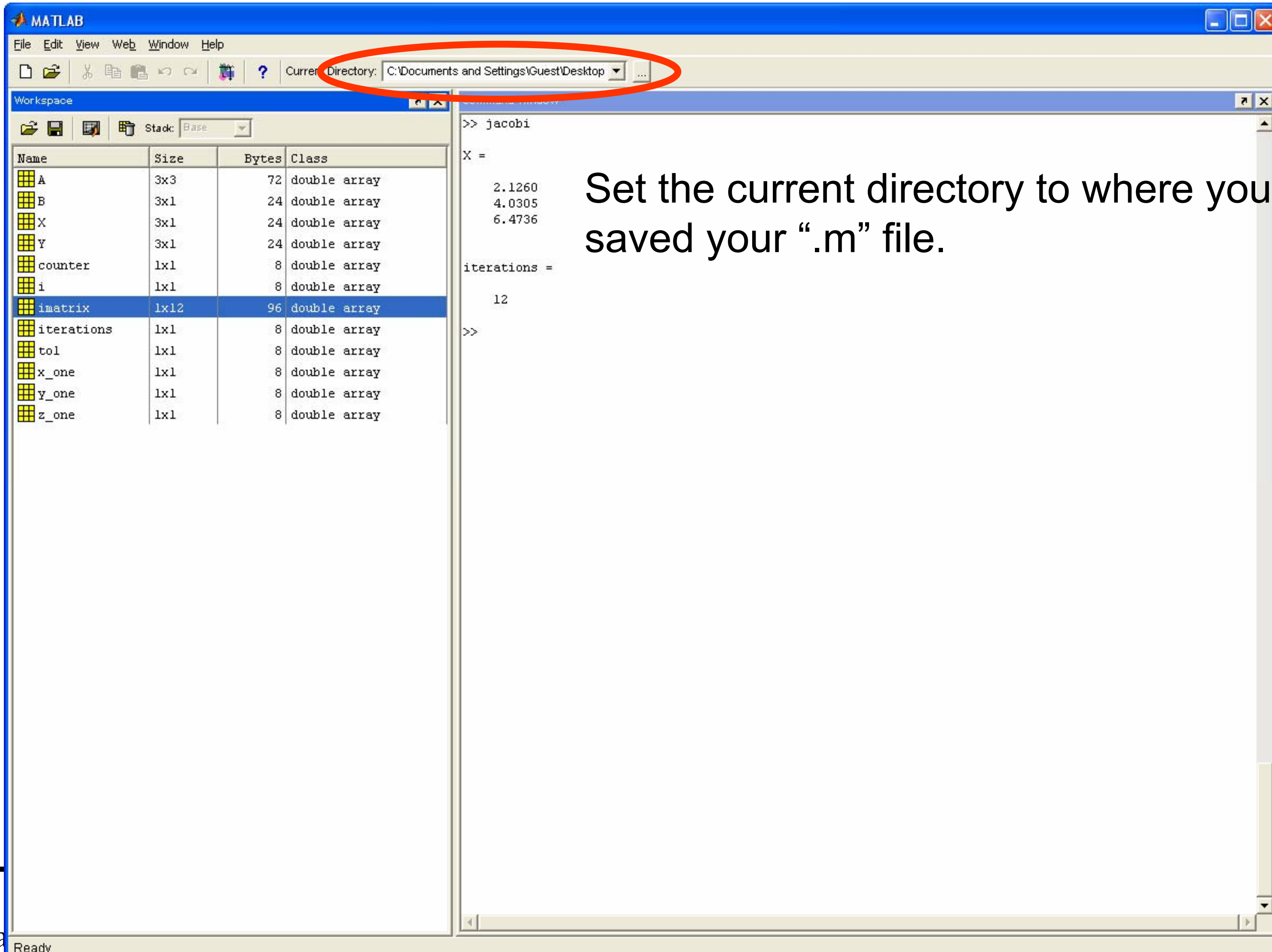
# Example: Pricing options in CRR Binomial

```
%Populate all possible stock prices and option values on the end notes of the tree

for i = 0:nSteps
    state = i + 1;
    St = AssetP * Up ^ i * Down ^ (nSteps - i);
    Value(state) = max(0, b * (St - Strike));
End

%Since value on the end nodes are known by above,
% we start from nSteps-1 working backwards
% double for loop: outter-every steps; innter-every nodes on each step

for k = nSteps - 1 : -1 : 0
    for i = 0:k
        state = i + 1;
        Value(state) = (Q_up * Value(state + 1) + Q_down * Value(state)) * Df;
    end
end

Binomial = Value(1)
```

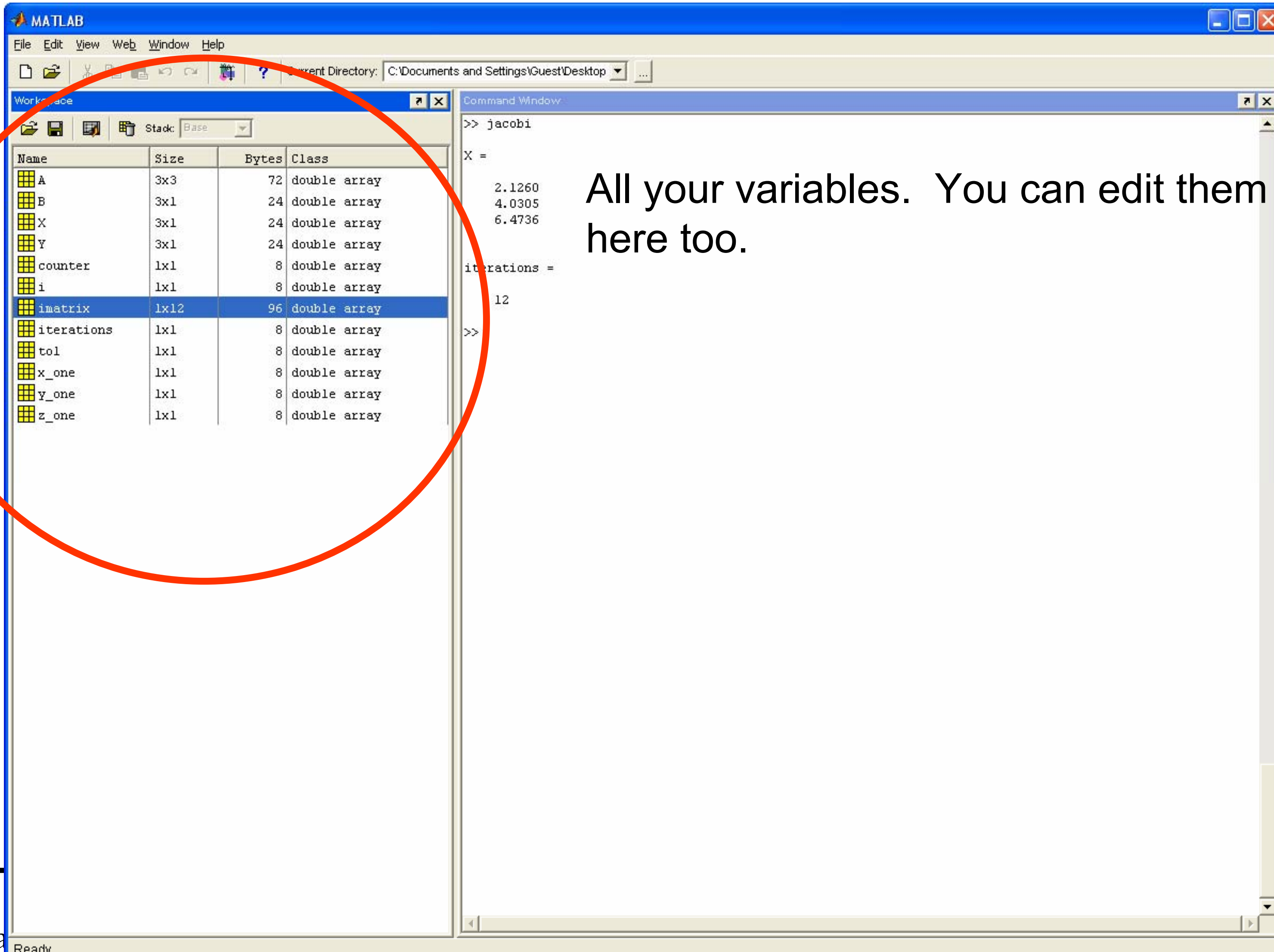
# Results


MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\Guest\Desktop
Workspace
Stack: Base
Name Size Bytes Class
A 3x3 72 double array
B 3x1 24 double array
X 3x1 24 double array
Y 3x1 24 double array
counter 1x1 8 double array
i 1x1 8 double array
imatrix 1x12 96 double array
iterations 1x1 8 double array
tol 1x1 8 double array
x_one 1x1 8 double array
y_one 1x1 8 double array
z_one 1x1 8 double array
Command Window
>> jacobi
X =
2.1260
4.0305
6.4736
iterations =
12
>>
Ready

# Results

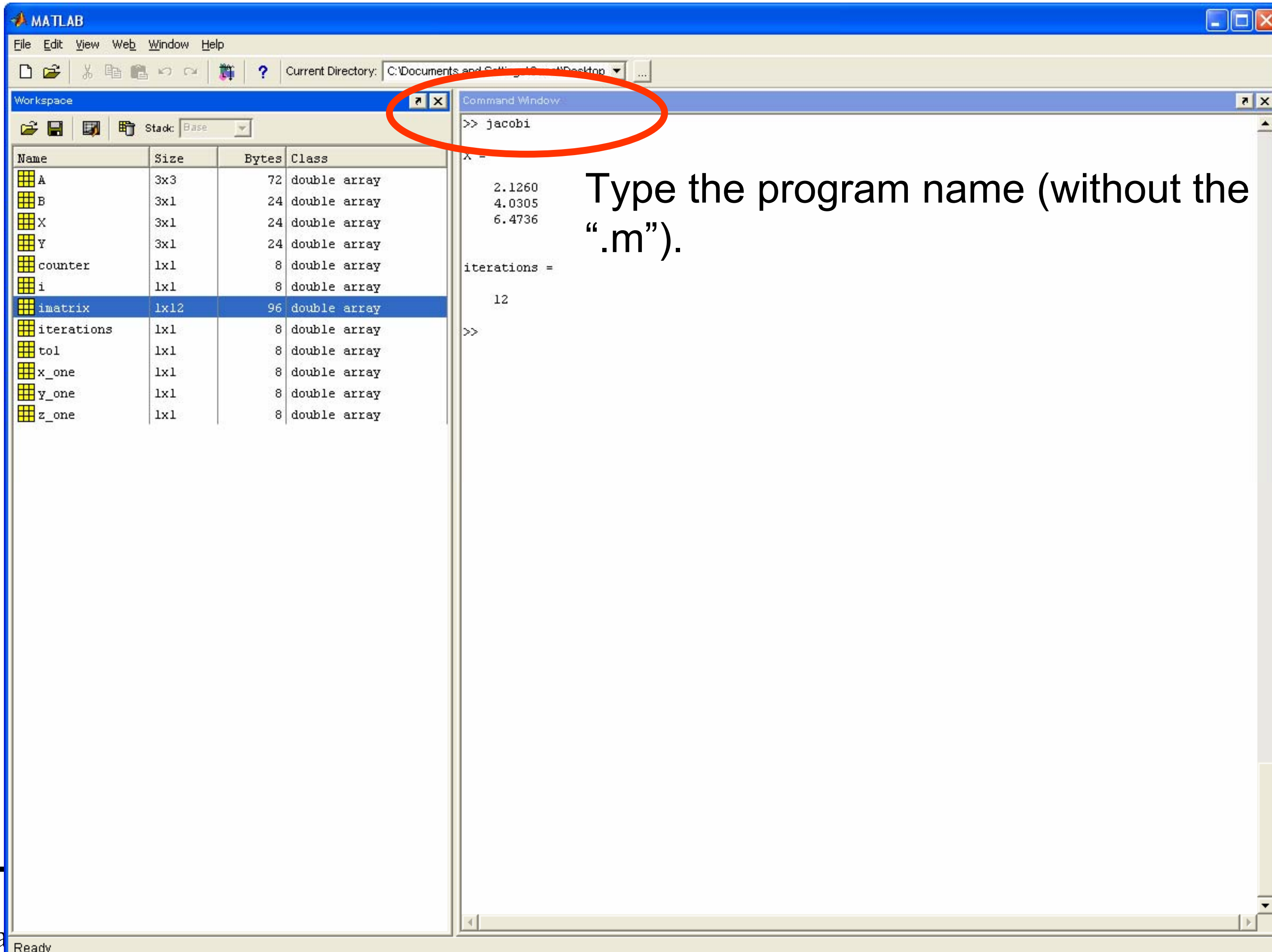

MATLAB
File Edit View Web Window Help
Current Directory: C:\Documents and Settings\Guest\Desktop
Workspace
Stack: Base
Name Size Bytes Class
A 3x3 72 double array
B 3x1 24 double array
X 3x1 24 double array
Y 3x1 24 double array
counter 1x1 8 double array
i 1x1 8 double array
imatrix 1x12 96 double array
iterations 1x1 8 double array
tol 1x1 8 double array
x_one 1x1 8 double array
y_one 1x1 8 double array
z_one 1x1 8 double array
>> jacobi
X =
2.1260
4.0305
6.4736
iterations =
12
>>
Set the current directory to where you saved your ".m" file.
Ready

# Results




All your variables. You can edit them here too.

# Results




Type the program name (without the ".m").

# Example: Pricing options in CRR Binomial

- >> CRR(1,100,105,0.05,0,2,0.4,100)

  Binomial =

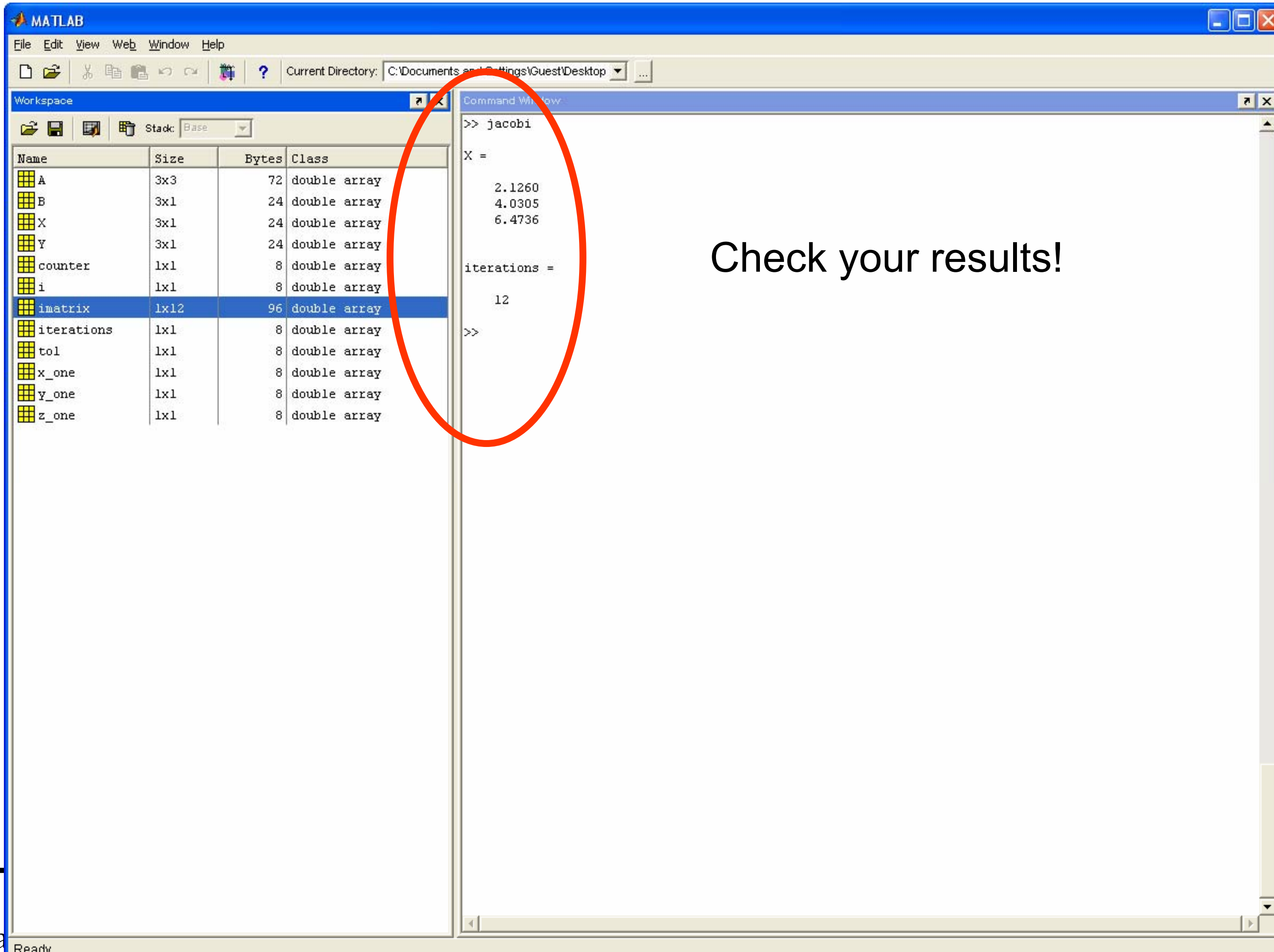
  24.3440

- >> CRR(0,100,105,0.05,0,2,0.4,100)

  Binomial =

  19.3520

# Results


MATLAB
File Edit View Web Window Help
Current Directory:
Workspace
Stack: Base
Name Size Bytes Class
A 3x3 72 double array
B 3x1 24 double array
X 3x1 24 double array
Y 3x1 24 double array
counter 1x1 8 double array
i 1x1 8 double array
imatrix 1x12 96 double array
iterations 1x1 8 double array
tol 1x1 8 double array
x_one 1x1 8 double array
y_one 1x1 8 double array
z_one 1x1 8 double array
>> jacobi
X =
2.1260
4.0305
6.4736
iterations =
12
>>
Check your results!
Ready

# How to Leave Matlab?

- The answer to the most popular question concerning any program is this: leave a Matlab session
- Leave Matlab by typing

  *quit*
- or by typing

  *exit*
- To the Matlab prompt.