

# Electrical Engineering Matlab Tutorial

## Part II: Programming and Visualization

**Michael D. DeVore**

mdd2@essrl.wustl.edu

**M. Ali Rauf**

ar2@ee.wustl.edu

Tutorial online at: <http://cis.wustl.edu/~mdd2/matlab/matlab.html>

# Outline

- Programming
  - Scripts
  - Functions
  - Debugging Environment
  - Conditional Execution
- Visualization
  - Two-Dimensional Graphing
  - Three-Dimensional Graphing
  - Saving Graphs
  - Presentation

# Scripts - New File

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

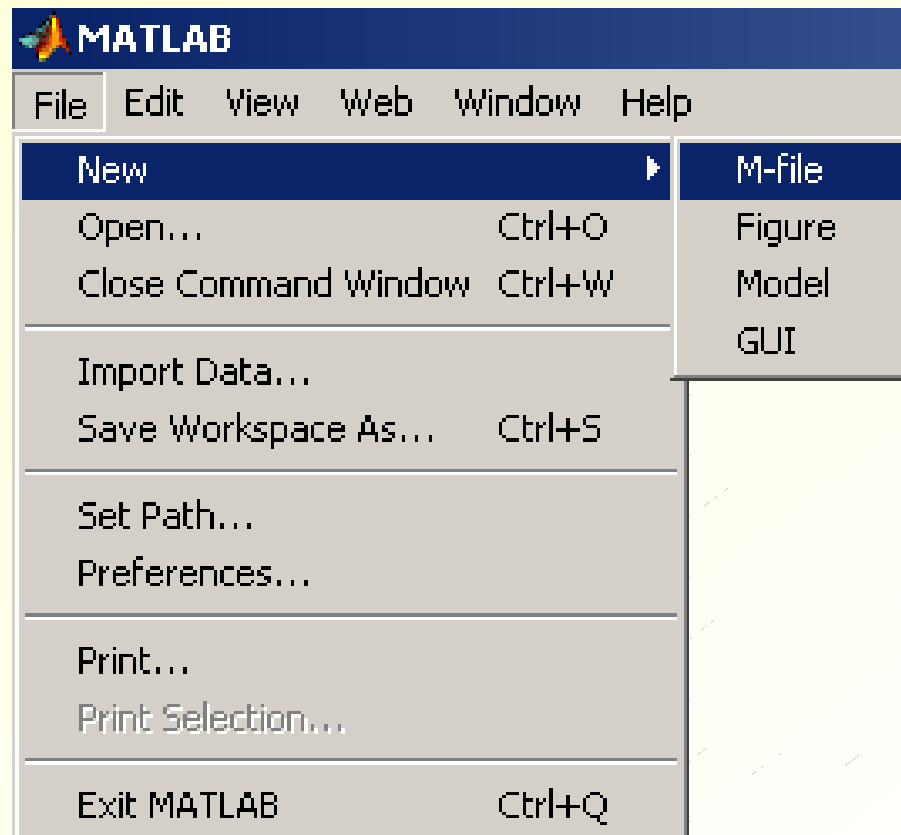
### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Matlab commands can be placed in a file for repeated execution
- Such files typically use the extension ‘.m’
- Files can be created in any text editor
- Create a new file in Matlab’s editor in the **File** menu



# Scripts - Editor/Debugger

Programming

**Scripts**

Functions

Debugging

Conditional  
Execution

Visualization

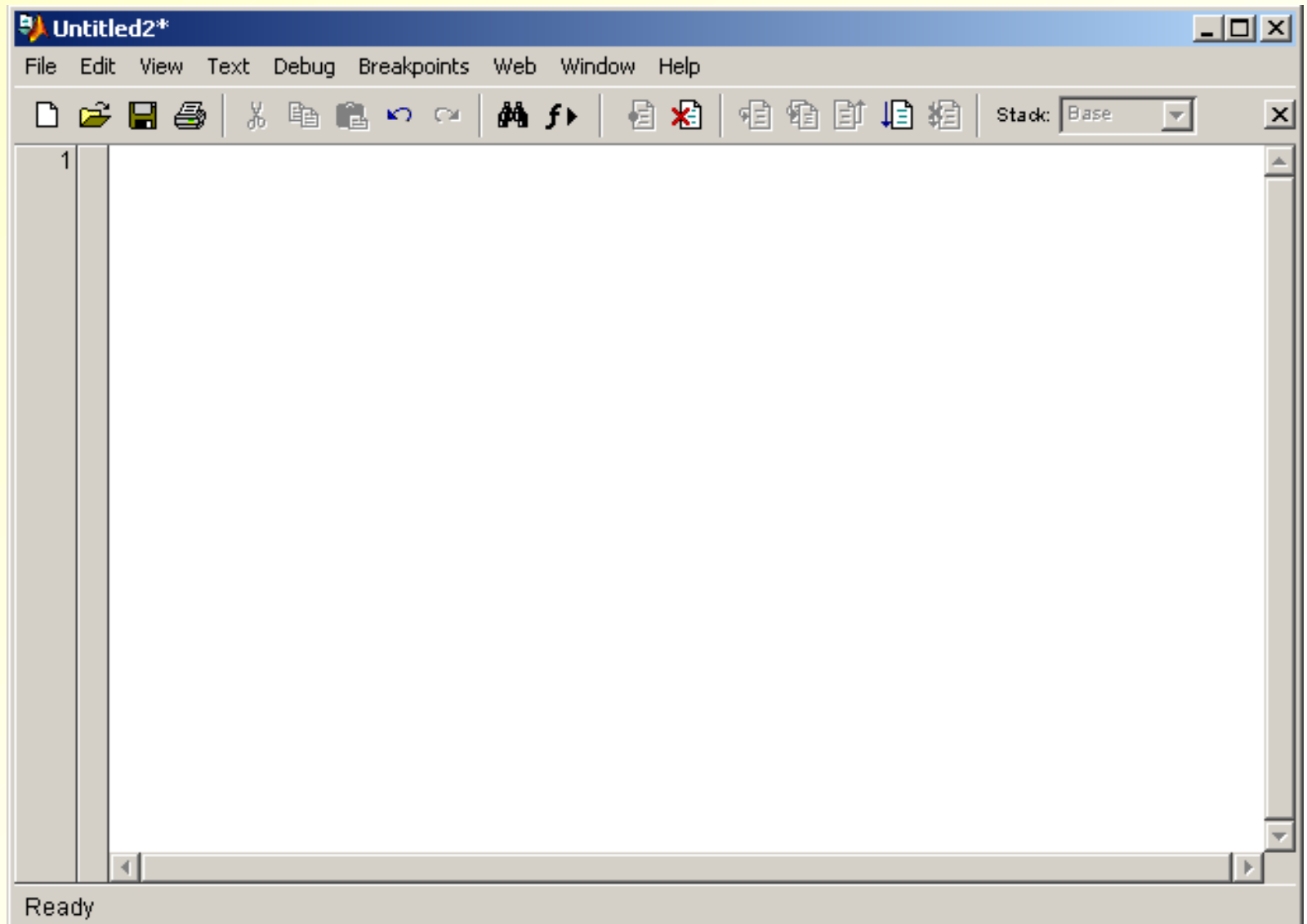
2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- The resulting window is the Matlab Editor/Debugger



# Scripts - Context

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- A script executes as if typed from the command line
- Same context
  - variables defined in the Command Window are visible in the script
  - variable creation/changes/clearing in the script will be visible in the Command Window
- No parameters can be passed to a script
- No return values can pass from a script

# Scripts - Demo

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- A demo script created in the Editor/Debugger
- Line numbers automatically included
- Comments begin with ‘%’
- Save As... to be able to execute the script

```
H:\Instruction\MatlabTutorial\DemoScript.m
File Edit View Text Debug Breakpoints Web Window Help
[New] [Open] [Save] [Print] | [Cut] [Copy] [Paste] | [Home] [End] | [Run] [Stop] [Close]
1 % Demo script file
2 % Assume variable A is already defined
3
4 myResult = A + 5
5
6 % Change the value of A
7
8 A = 1;
Ready
```

# Scripts - Execution

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Type the script name to execute it
- Script output appears in the Command Window
- Press **<Ctrl> - C** to stop execution of any command prior to termination

```
>> A=5;  
>> DemoScript  
  
myResult =  
  
    10  
  
>> A  
  
A =  
  
    1
```

# Scripts - Current Directory

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

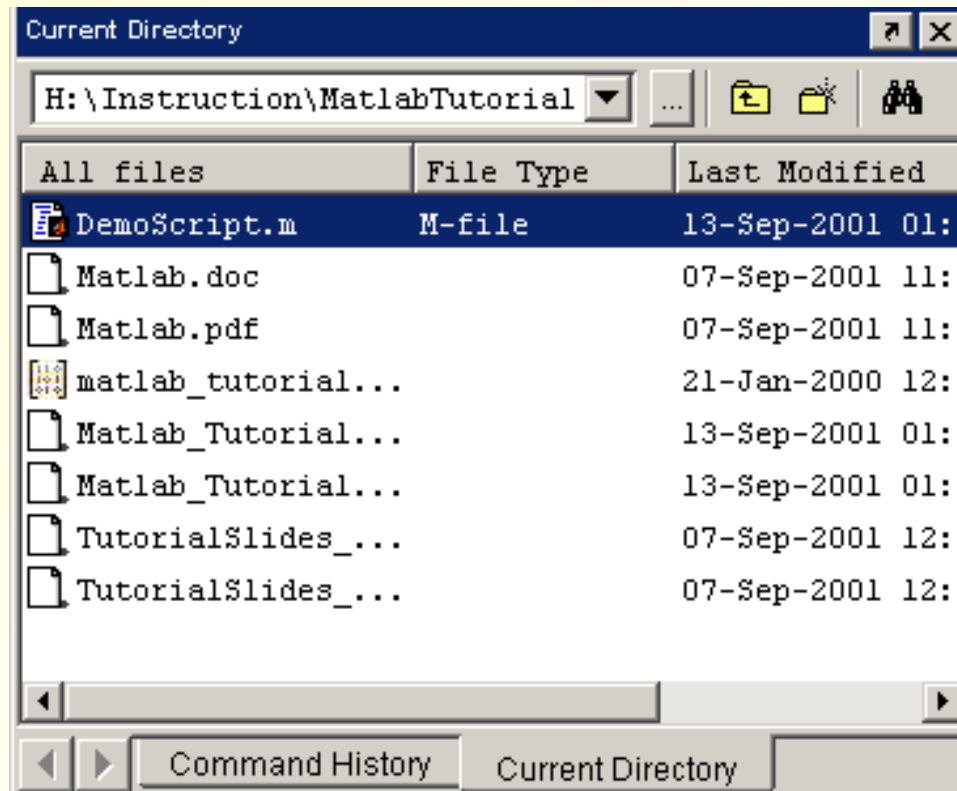
### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- To execute, the script file must be:
  - in the current directory, or
  - in a directory in the search path
- The Current Directory panel shows the contents of the current directory and lets you change the location
- Double-click an M-file to open it in the editor



# Scripts - Search Path

Programming

**Scripts**

Functions

Debugging

Conditional Execution

Visualization

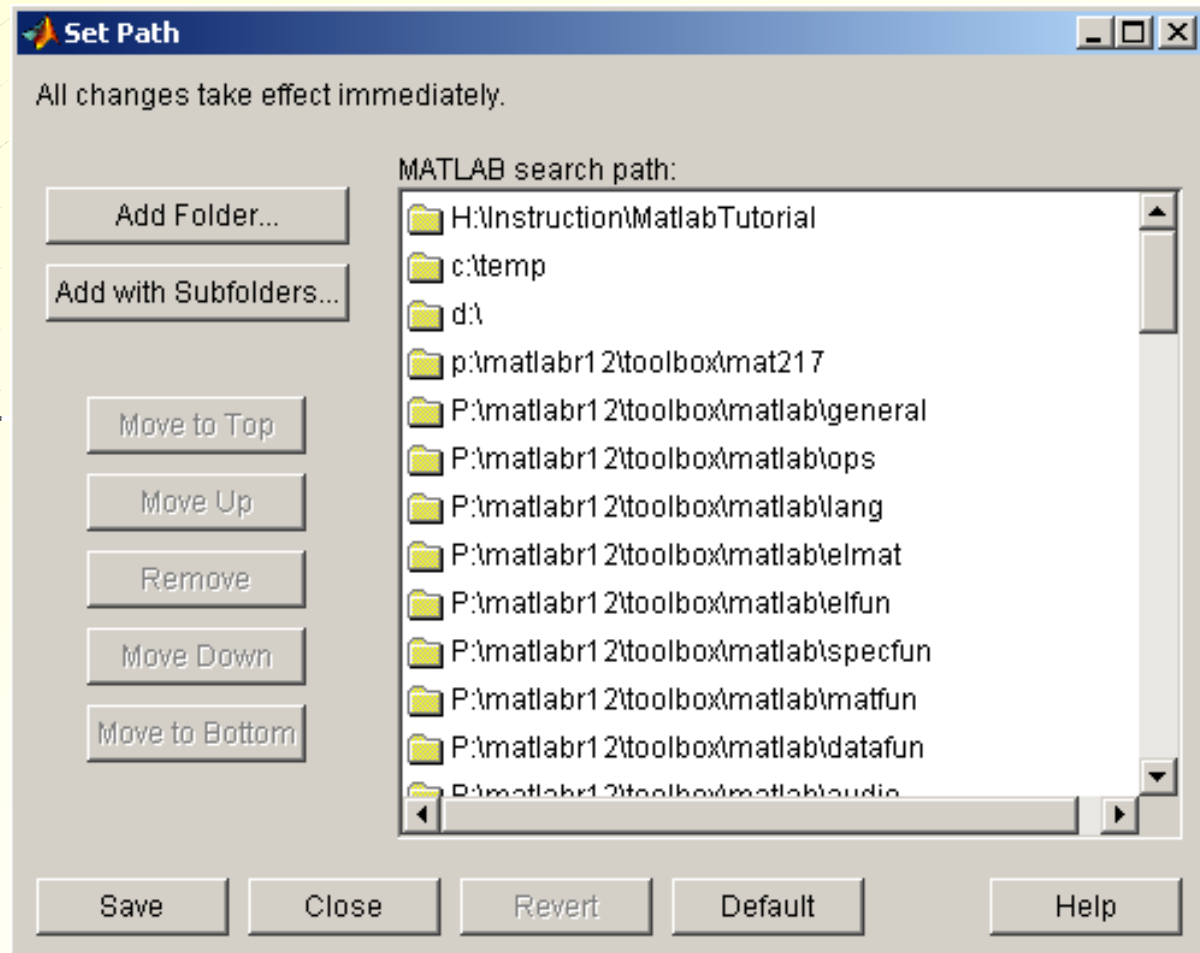
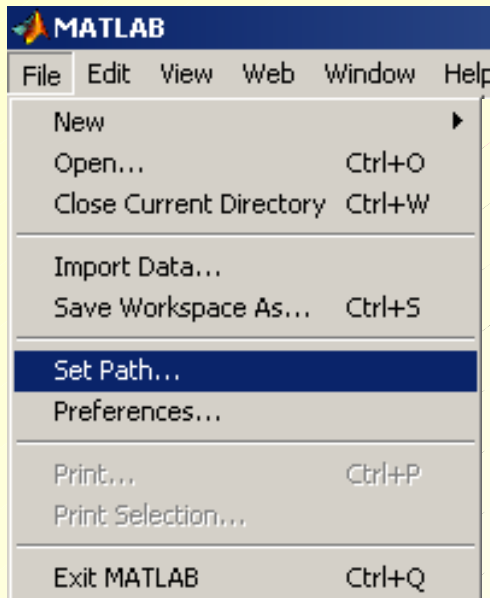
2-D Graphs

3-D Graphs

Saving Graphs

Presentation

- To examine the search path, open Set Path
- To update the path in a script, use **addpath**



# Functions - Context

Programming

Scripts

**Functions**

Debugging

Conditional  
Execution

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- A function is a special type of M-file
- Different context
  - variables defined in the Command Window are **not** visible in the function
  - variable creation/changes/clearing in the function will **not** be visible in the Command Window
- Parameters can be passed to a function
  - Parameters are passed by reference unless they are modified within the function
  - You cannot force pass by reference nor are data pointers supported
  - Matlab does support global variables (global) and simple context switching (evalin, assignin)
- Return values can pass from a function

# Functions - Demo

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

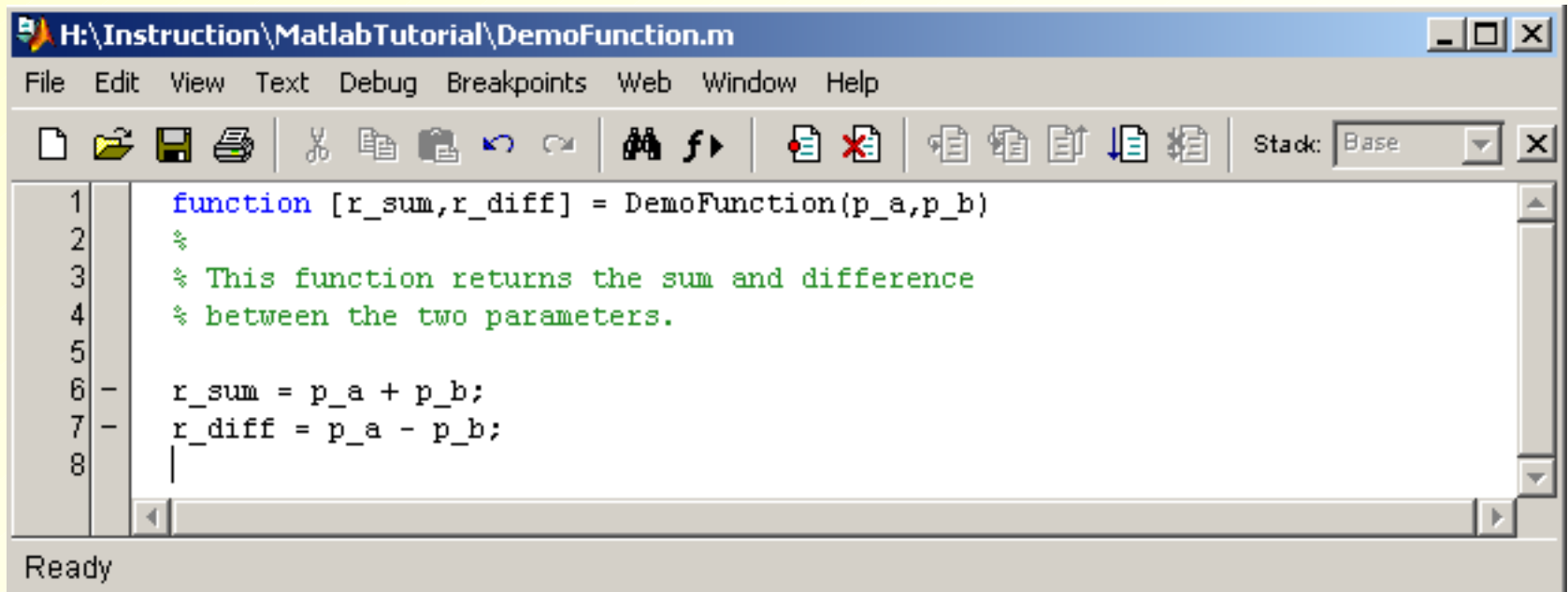
### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Function files begin with the syntax:  
`function [ret1, ret2, ...] = myfun(param1,param2,...)`
- Comments immediately after the declaration appear in the Command Window when you type 'help myfun'



The screenshot shows a MATLAB editor window titled "H:\Instruction\MatlabTutorial\DemoFunction.m". The window contains the following code:

```
1 function [r_sum,r_diff] = DemoFunction(p_a,p_b)
2 %
3 % This function returns the sum and difference
4 % between the two parameters.
5
6 - r_sum = p_a + p_b;
7 - r_diff = p_a - p_b;
8 |
```

The status bar at the bottom of the window displays "Ready".

# Functions - Invoking

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Invoke the function like any other Matlab function
- To accept multiple return values, place them in square brackets [ ]

```
>> help DemoFunction
```

```
This function returns the sum and difference  
between the two parameters.
```

```
>> [theSum,theDiff] = DemoFunction([1 2 3],1)
```

```
theSum =
```

```
     2     3     4
```

```
theDiff =
```

```
     0     1     2
```

# Debugging - Set Breakpoint

Programming

Scripts

Functions

**Debugging**

Conditional  
Execution

Visualization

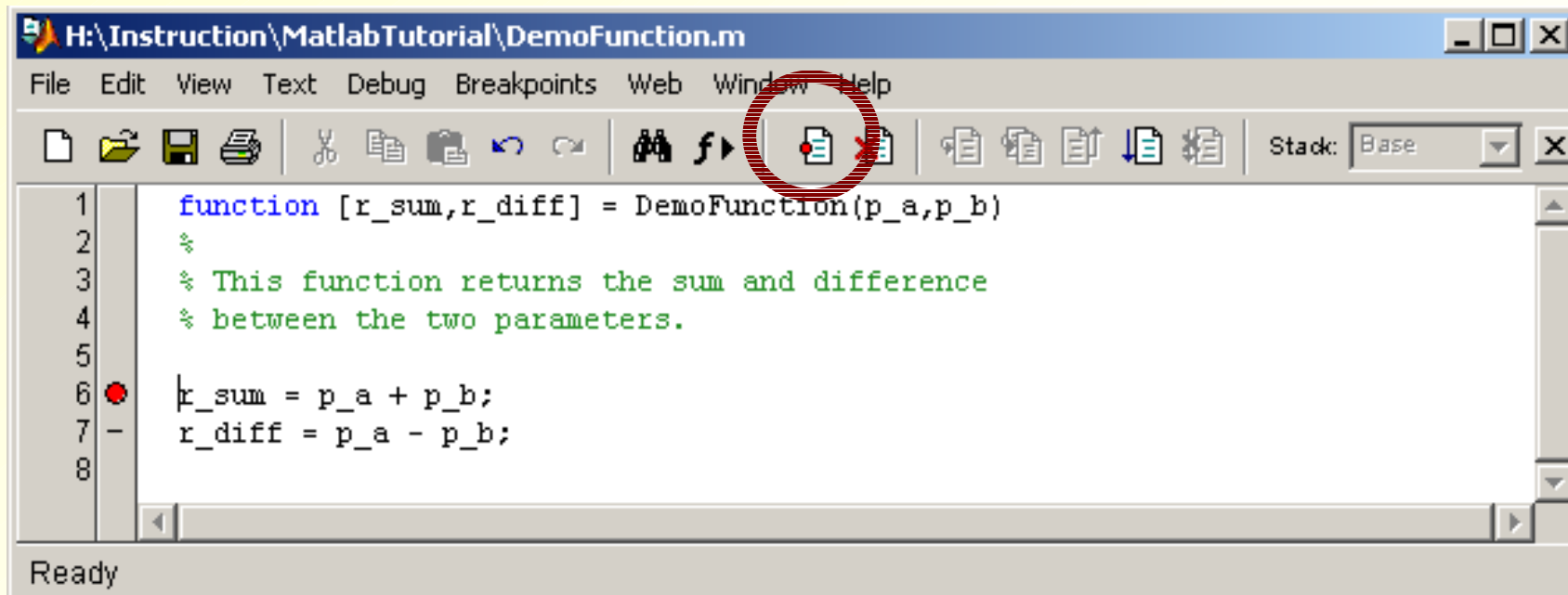
2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- The Editor/Debugger also allows interactive debugging of functions and scripts
- Move the cursor to the line debugging should start and click the Set/clear breakpoint icon



# Debugging - Control

Programming

Scripts

Functions

**Debugging**

Conditional Execution

Visualization

2-D Graphs

3-D Graphs

Saving Graphs

Presentation

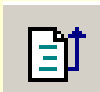
- The debugger provides control over execution



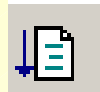
– executes the current instruction



– jumps into a subroutine



– executes through the end of a subroutine



– continue execution



– stop debugging

The screenshot shows the MATLAB debugger window for a file named 'DemoFunction.m'. The window title is 'H:\Instruction\MatlabTutorial\DemoFunction.m'. The menu bar includes 'File', 'Edit', 'View', 'Text', 'Debug', 'Breakpoints', 'Web', 'Window', and 'Help'. The toolbar contains various icons for file operations, editing, and debugging. The main area displays the following code:

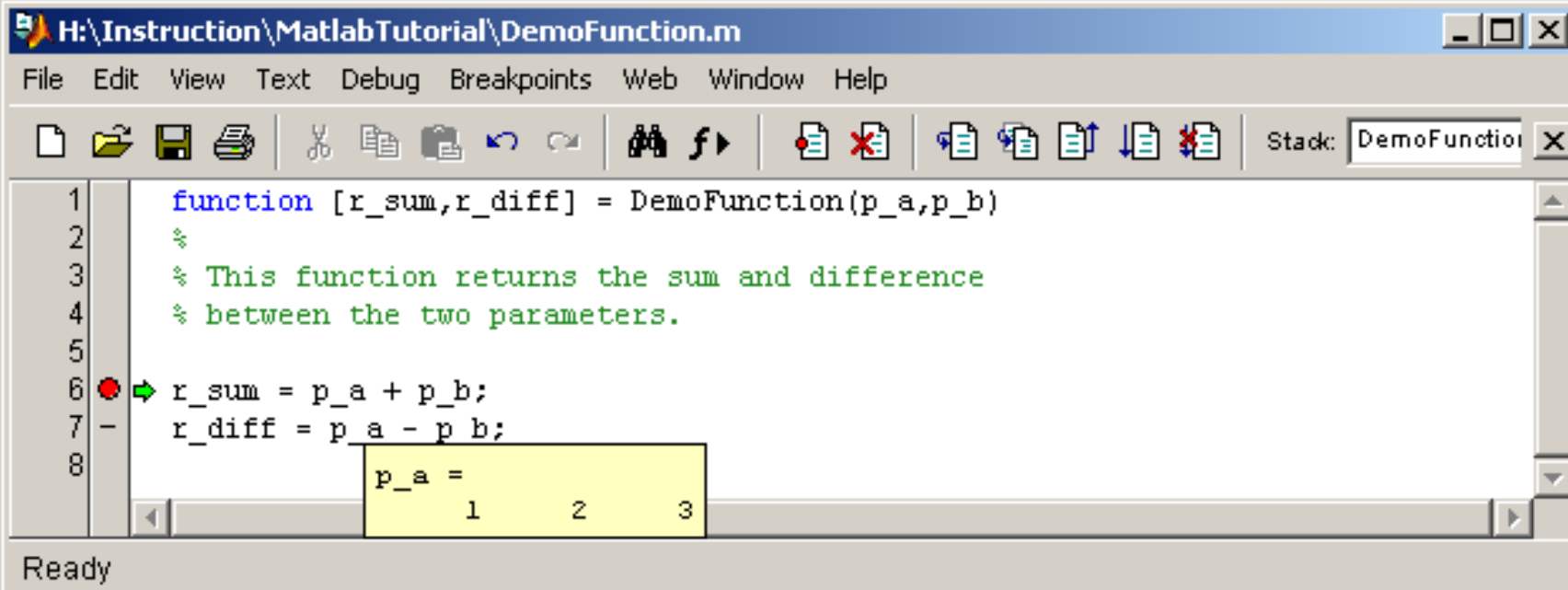
```
1 function [r_sum,r_diff] = DemoFunction(p_a,p_b)
2 %
3 % This function returns the sum and difference
4 % between the two parameters.
5
6 ● → r_sum = p_a + p_b;
7 -   r_diff = p_a - p_b;
8
```

The status bar at the bottom indicates 'Ready'. The stack on the right shows 'DemoFunction'.

# Debugging - Variables and Errors

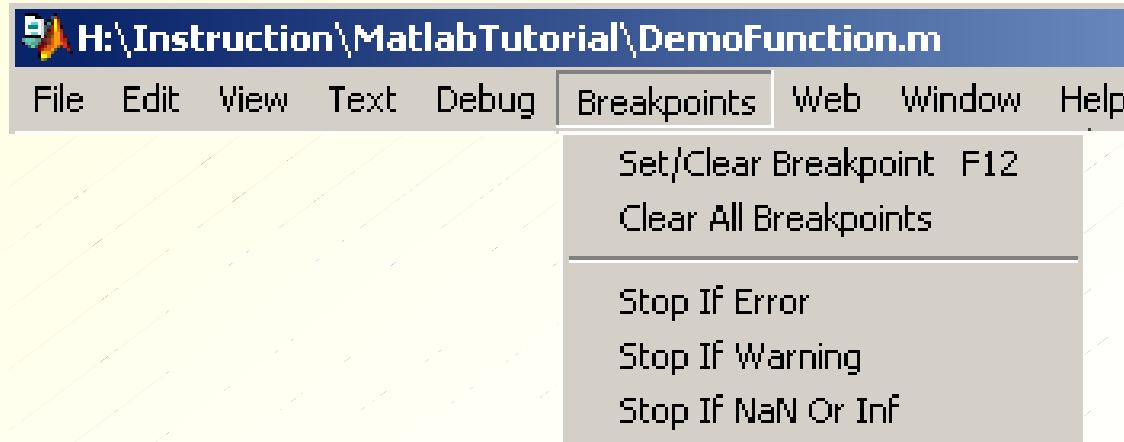
- Programming
- Scripts
- Functions
- Debugging**
- Conditional Execution
- Visualization
  - 2-D Graphs
  - 3-D Graphs
- Saving Graphs
- Presentation

- Pausing the pointer over a variable shows its value



```
H:\Instruction\MatlabTutorial\DemoFunction.m
File Edit View Text Debug Breakpoints Web Window Help
[Icons] Stack: DemoFunction
1 function [r_sum,r_diff] = DemoFunction(p_a,p_b)
2 %
3 % This function returns the sum and difference
4 % between the two parameters.
5
6 ● → r_sum = p_a + p_b;
7 -   r_diff = p_a - p_b;
8
p_a =
     1     2     3
Ready
```

- Breakpoints can also be set for errors, etc.



# Debugging - Command Window

Programming

Scripts

Functions

**Debugging**

Conditional  
Execution

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- Command Window prompt changes when debugging
- You can evaluate expressions in the function context
- You can change the value of parameters and local variables

```
K>> p_a
p_a =
     1     2     3

K>> sin(p_b)
ans =
     0.8415

K>> p_b = 10;
```

# Conditional Execution - Logical Arrays

Programming

Scripts

Functions

Debugging

**Conditional  
Execution**

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- Some operators return logical arrays
- Logical arrays can be operated upon like floating-point
- Bitwise operators are:

& - AND

| - OR

~ - NOT

```
>> D = [1 2; 3 4];
```

```
>> D > 2
```

```
ans =
```

```
    0    0  
    1    1
```

```
>> (D>2) & (D<4)
```

```
ans =
```

```
    0    0  
    1    0
```

# Conditional Execution - if Construct

Programming

Scripts

Functions

Debugging

**Conditional  
Execution**

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- Construct

```
if expr
    statement1
else
    statement2
end
```

executes statement1 if all elements of expr are non-zero, otherwise statement2 is executed

```
>> if (D>3)
    disp('All elements are greater than three');
else
    disp('Some elements are not greater than three');
end
Some elements are not greater than three
```

# Conditional Execution - for Construct

Programming

Scripts

Functions

Debugging

**Conditional  
Execution**

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- Construct

```
for var = expr  
    statements  
end
```

assigns columns of expr to var in turn

```
>> for v=1:5  
    disp(v);  
end  
1  
2  
3  
4  
5
```

# Conditional Execution - while Construct

Programming

Scripts

Functions

Debugging

**Conditional  
Execution**

Visualization

2-D Graphs

3-D Graphs

Saving  
Graphs

Presentation

- Construct  
    while expr  
        statements  
    end  
  
iterates until all elements of expr are zero

```
>> A=0;  
>> while A ~= 5  
        disp(A);  
        A = A + 1;  
end  
0  
  
1  
  
2  
  
3  
  
4
```

# 2-D Graphs - Figure Windows

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Create a figure window with 'figure'
- The figure command returns a handle for later use
- Also use figure(H) to make H the active figure
- Use close(H) to close figure H

```
>> myFig = figure
myFig =
     1

>> myFig2 = figure;
>>
>> % This will bring myFig forward and make it active
>> figure(myFig)
>>
>> % This will close myFig2
>> close(myFig2)
```

# 2-D Graphs - Plotting

## Programming

Scripts

Functions

Debugging

Conditional Execution

## Visualization

**2-D Graphs**

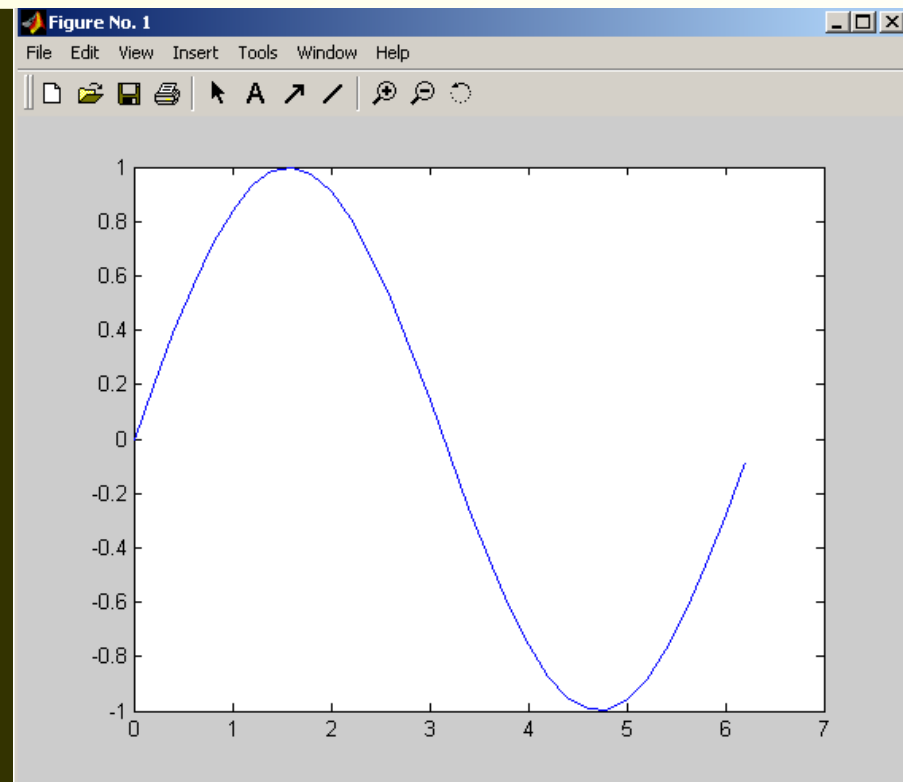
3-D Graphs

Saving Graphs

Presentation

- Make a plot in the current figure window with 'plot'
  - First parameter is an array of X values
  - Second parameter is an array of Y values
  - Third parameter can specify a color and line style

```
>> theta=0:0.2:2*pi;  
>> plot(theta,sin(theta))
```



# 2-D Graphs - Overlaying Plots

## Programming

Scripts

Functions

Debugging

Conditional  
Execution

## Visualization

**2-D Graphs**

3-D Graphs

Saving  
Graphs

Presentation

- Simply plotting again will replace the old plot
- To overlay a plot, use 'hold on'. To replace again, use 'hold off'...
- `>> hold on`
- `>> plot(theta,sin(theta),'r')`

```
>> plot(theta,cos(theta),'g')  
>> hold on  
>> plot(theta,sin(theta),'r')
```

# 2-D Graphs - Overlaying Plots

Programming

Scripts

Functions

Debugging

Conditional Execution

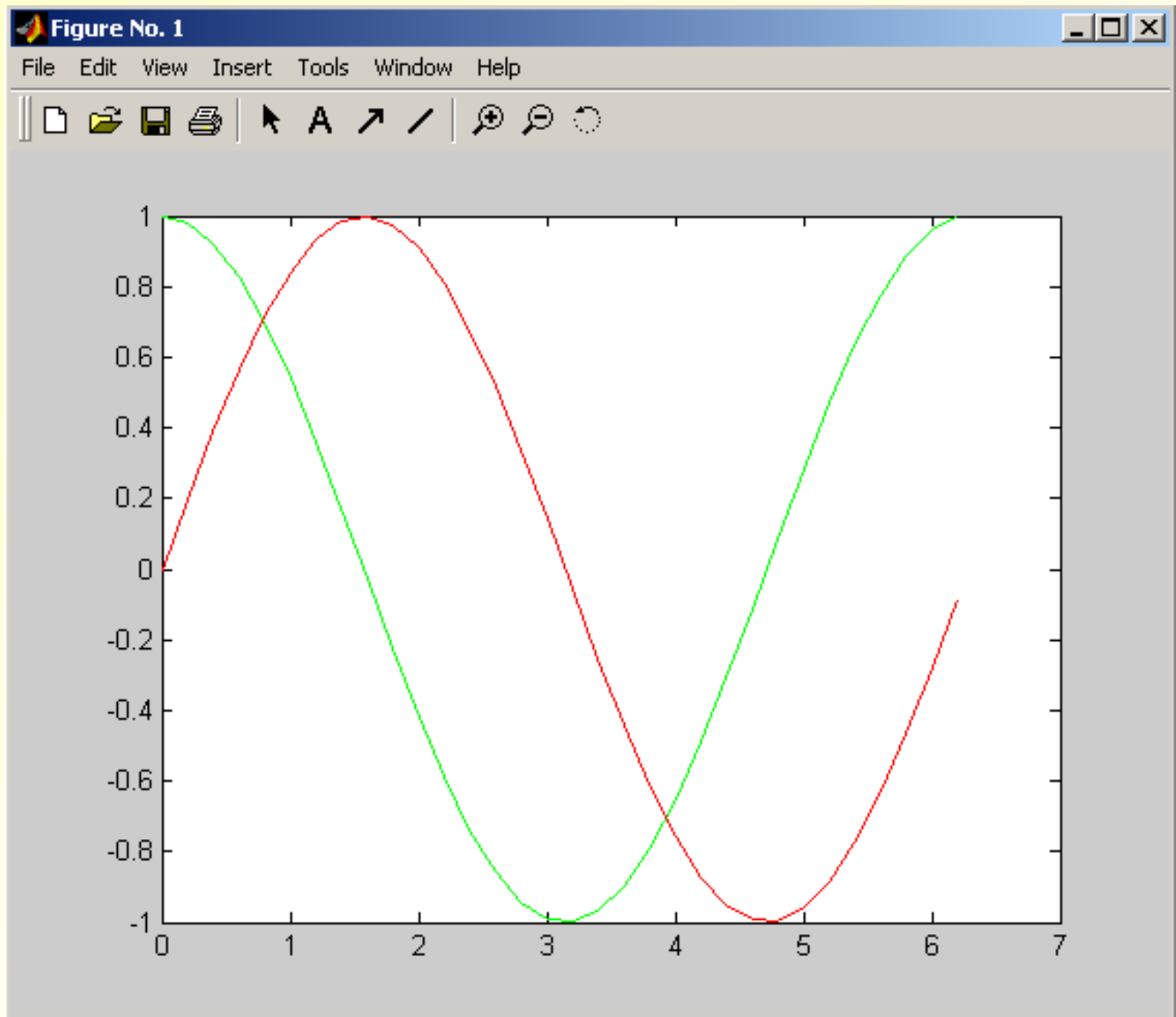
Visualization

**2-D Graphs**

3-D Graphs

Saving Graphs

Presentation



# 2-D Graphs - Labeling Plots

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Change axis ranges with `axis([Xmin, Xmax, Ymin, Ymax])`
- Always label a plot with legible information
  - Title text
  - Legend if more than one plot
  - Labels for X and Y axes

```
>> title('Sine and Cosine values','FontSize',14)
>> L = legend('Cosine','Sine');
>> xlabel('Angle in Radians','FontSize',12)
>> ylabel('Function Value','FontSize',12)
```

# 2-D Graphs - Labeling Plots

Programming

Scripts

Functions

Debugging

Conditional Execution

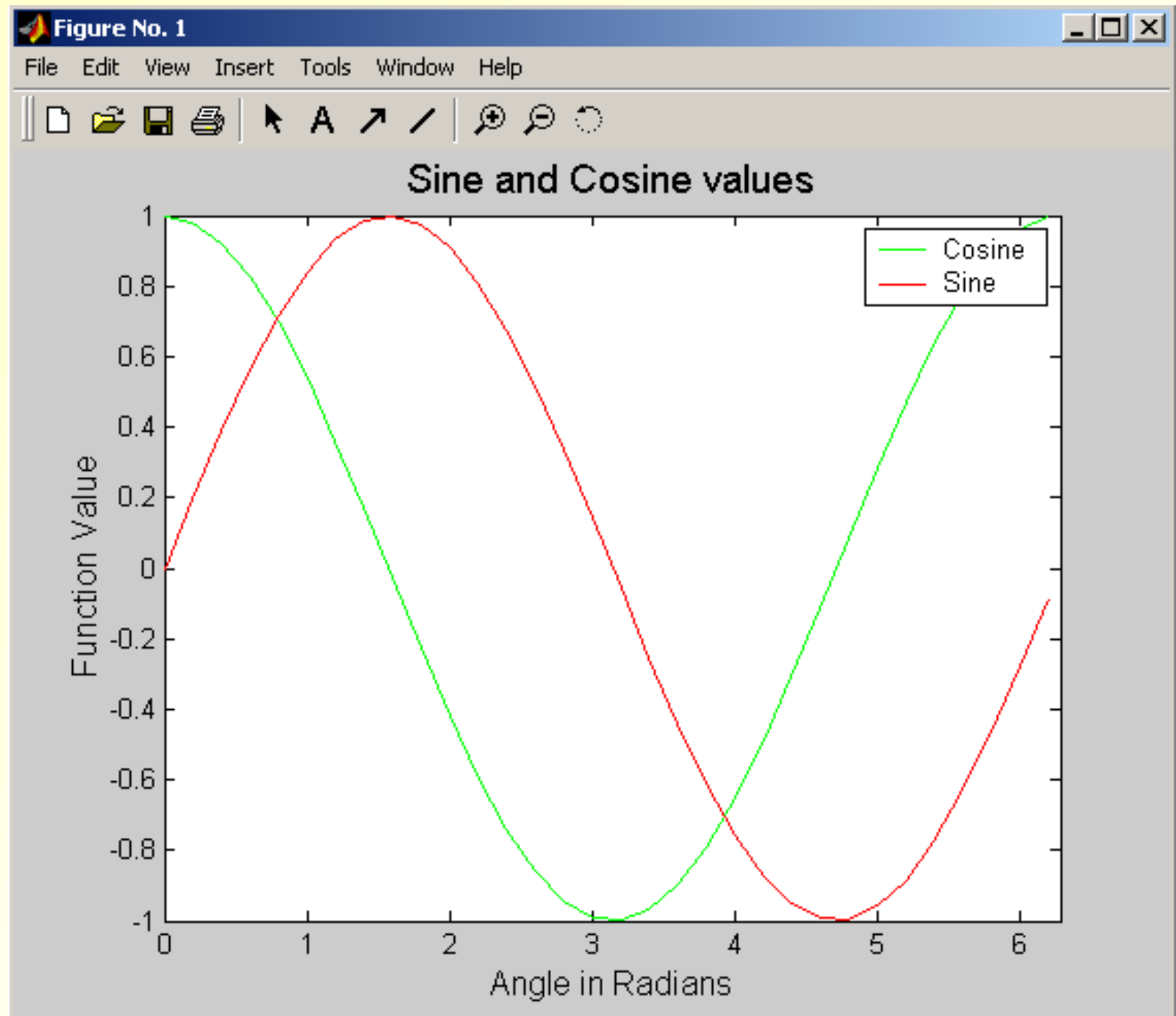
Visualization

**2-D Graphs**

3-D Graphs

Saving Graphs

Presentation



# 2-D Graphs - Subplots

Programming

Scripts

Functions

Debugging

Conditional  
Execution

Visualization

**2-D Graphs**

3-D Graphs

Saving  
Graphs

Presentation

- Tile more than one plot in a figure with `subplot(rows, columns, current)`
- Make polar plots with `polar(theta, rho)`
- Plot discrete sequence data with `stem(X, Y)`
- Make piecewise constant plots with

```
>> myFig2 = figure;  
>>  
>> % First we make a polar plot...  
>> subplot(2,1,1)  
>> title('Advanced Plotting','FontSize',14)  
>> polar(theta,1-sin(theta));  
>> xlabel('Polar Plot Example','FontSize',12)  
>>  
>> % ...then we add a stem plot...  
>> subplot(2,1,2)  
>> stem(theta,cos(theta))  
>> xlabel('Sample Location','FontSize',12)  
>> ylabel('F[n]','FontSize',12)
```

# 2-D Graphs - Subplots

Programming

Scripts

Functions

Debugging

Conditional Execution

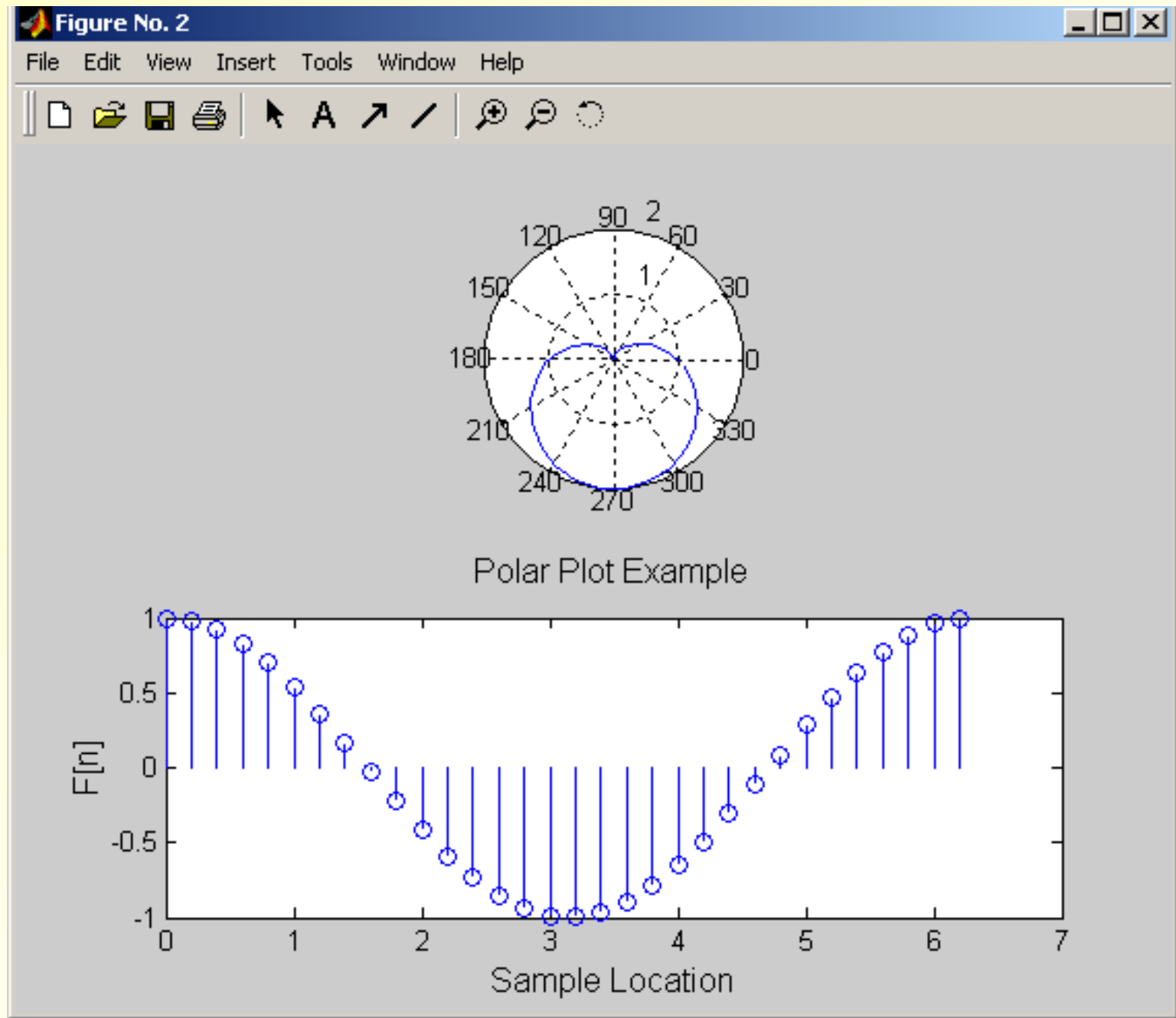
Visualization

**2-D Graphs**

3-D Graphs

Saving Graphs

Presentation



# 2-D Graphs - Other Plots and Editing

Programming

Scripts

Functions

Debugging

Conditional  
Execution

Visualization

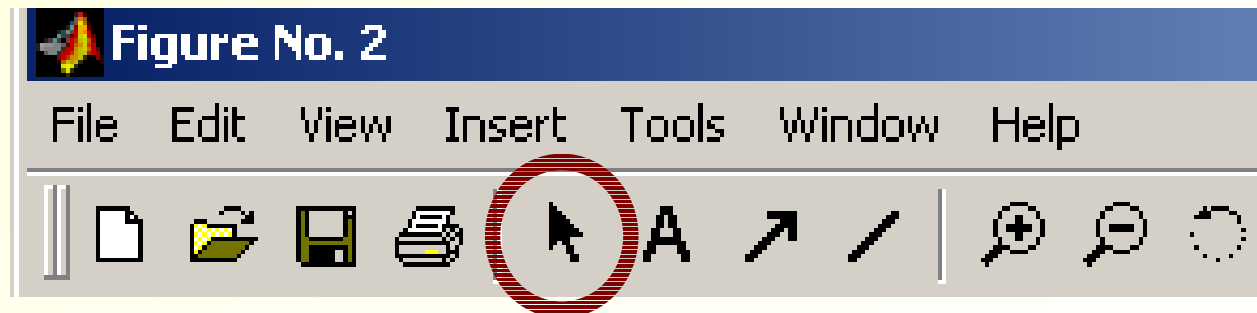
**2-D Graphs**

3-D Graphs

Saving  
Graphs

Presentation

- Matlab provides many other two-dimensional graphs
  - `bar( )` creates bar charts
  - `pie( )` creates pie charts
  - `loglog( )` creates plots with logarithmically space axes
  - `semilogx( )` and `semilogy( )` create plots with one axis logarithmically spaced
- Use the Edit Plot tool to select items in a plot
  - double-click axes, plots, etc. to change their properties



# 3-D Graphs - Plotting

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### **3-D Graphs**

### Saving Graphs

### Presentation

- Use 'plot3' to plot three-dimensional curves
  - First parameter is an array of X values
  - Second parameter is an array of Y values
  - Third parameter is an array of Z values

```
>> figure
>> theta2 = 0:0.2:(20*pi);
>> plot3(theta2,cos(theta2),sin(theta2))
```

# 3-D Graphs - Plotting

## Programming

Scripts

Functions

Debugging

Conditional Execution

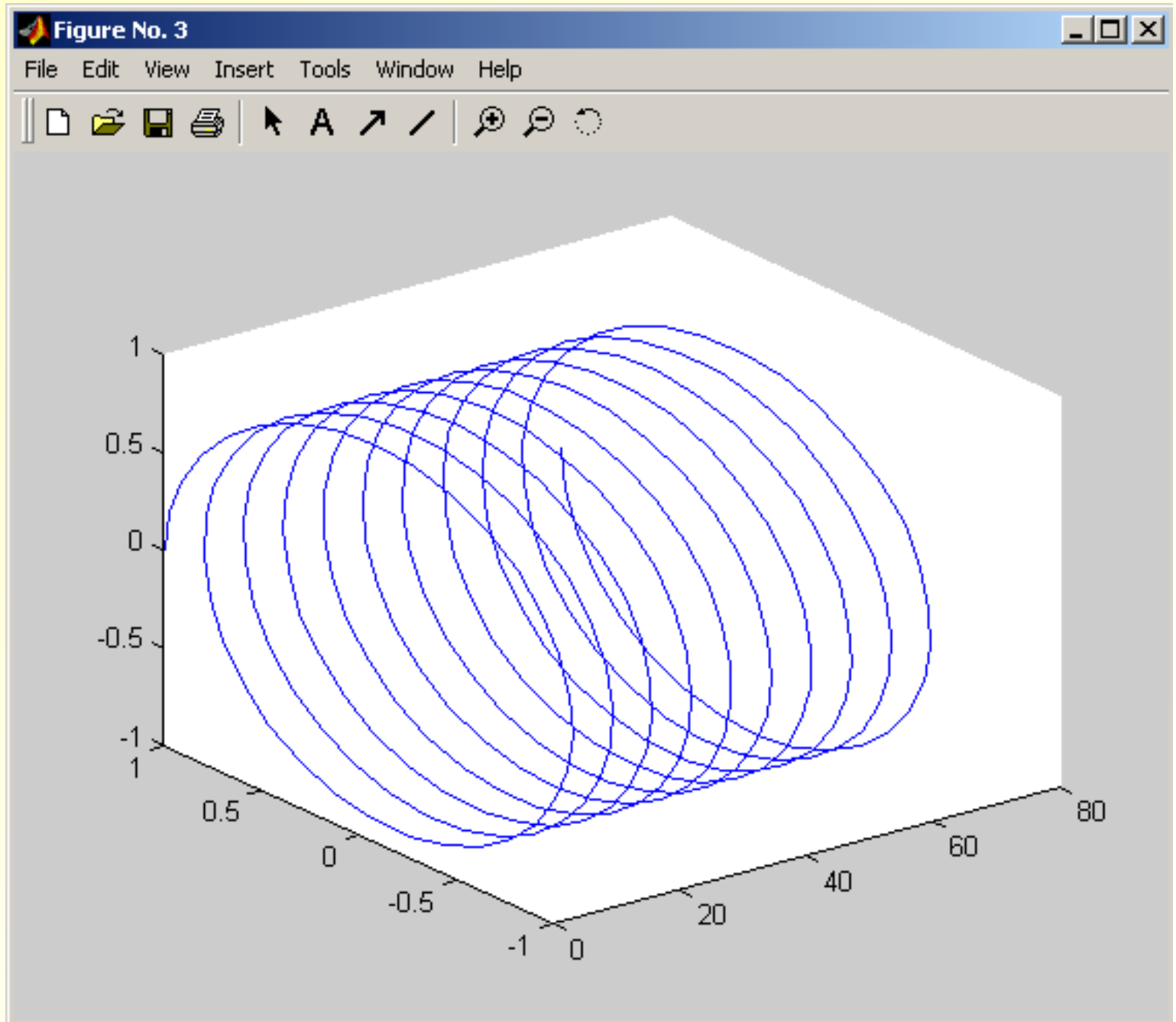
## Visualization

2-D Graphs

**3-D Graphs**

Saving Graphs

Presentation



# 3-D Graphs - View Direction

Programming

Scripts

Functions

Debugging

Conditional Execution

Visualization

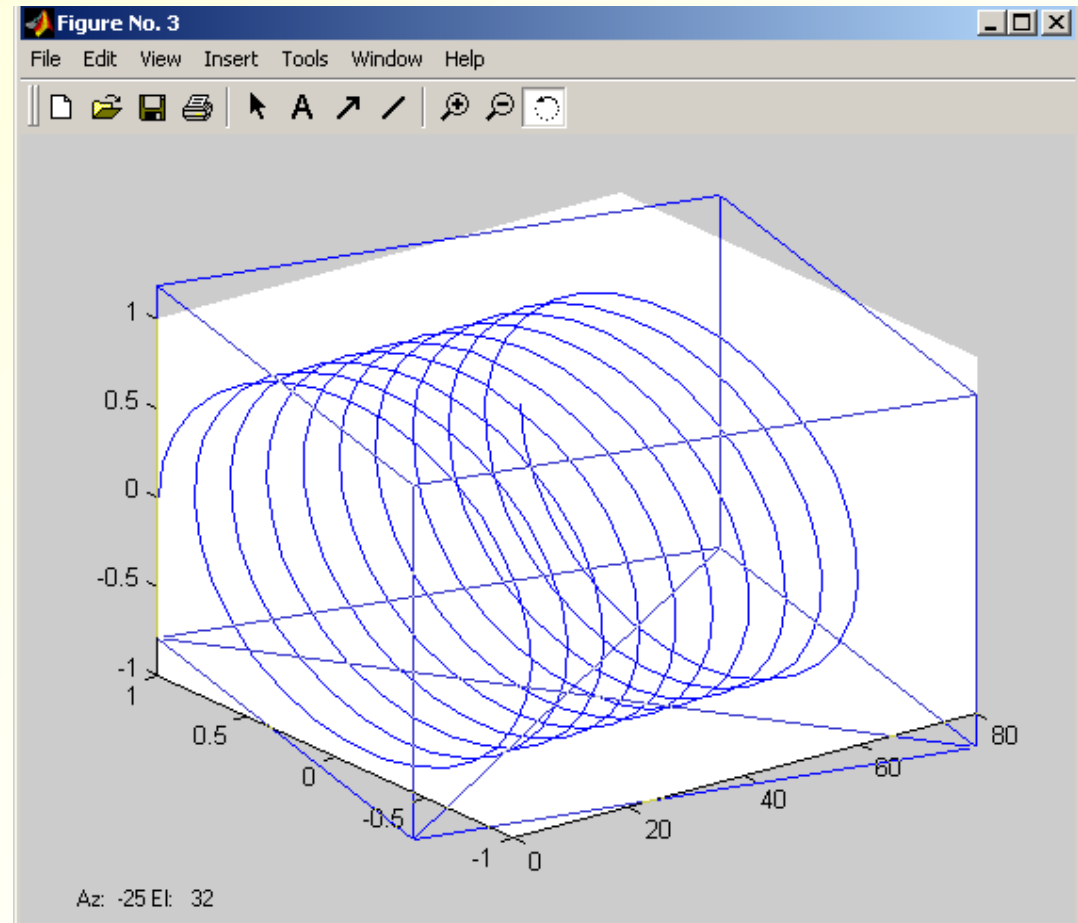
2-D Graphs

**3-D Graphs**

Saving Graphs

Presentation

- Use the rotate tool to change the viewing direction - click and drag in the plot



# 3-D Graphs - Mesh Plots

## Programming

### Scripts

### Functions

### Debugging

### Conditional Execution

## Visualization

### 2-D Graphs

### 3-D Graphs

### Saving Graphs

### Presentation

- Use 'mesh' to plot three-dimensional wire-frame surfaces
  - First parameter is a matrix of X values
  - Second parameter is a matrix of Y values
  - Third parameter is a matrix of Z(x,y) values
- Create evenly spaced coordinates with 'meshgrid'

```
>> figure
>> [X,Y] = meshgrid(0:0.1:2*pi);
>> mesh(X,Y,cos(X).*cos(Y))
>>
>> xlabel('\theta_1','FontSize',12)
>> ylabel('\phi_2','FontSize',12)
>> zlabel('F(\theta_1,\phi_2)','FontSize',12)
>> title('3-D Mesh Plot','FontSize',14)
```

# 3-D Graphs - Mesh Plots

Programming

Scripts

Functions

Debugging

Conditional Execution

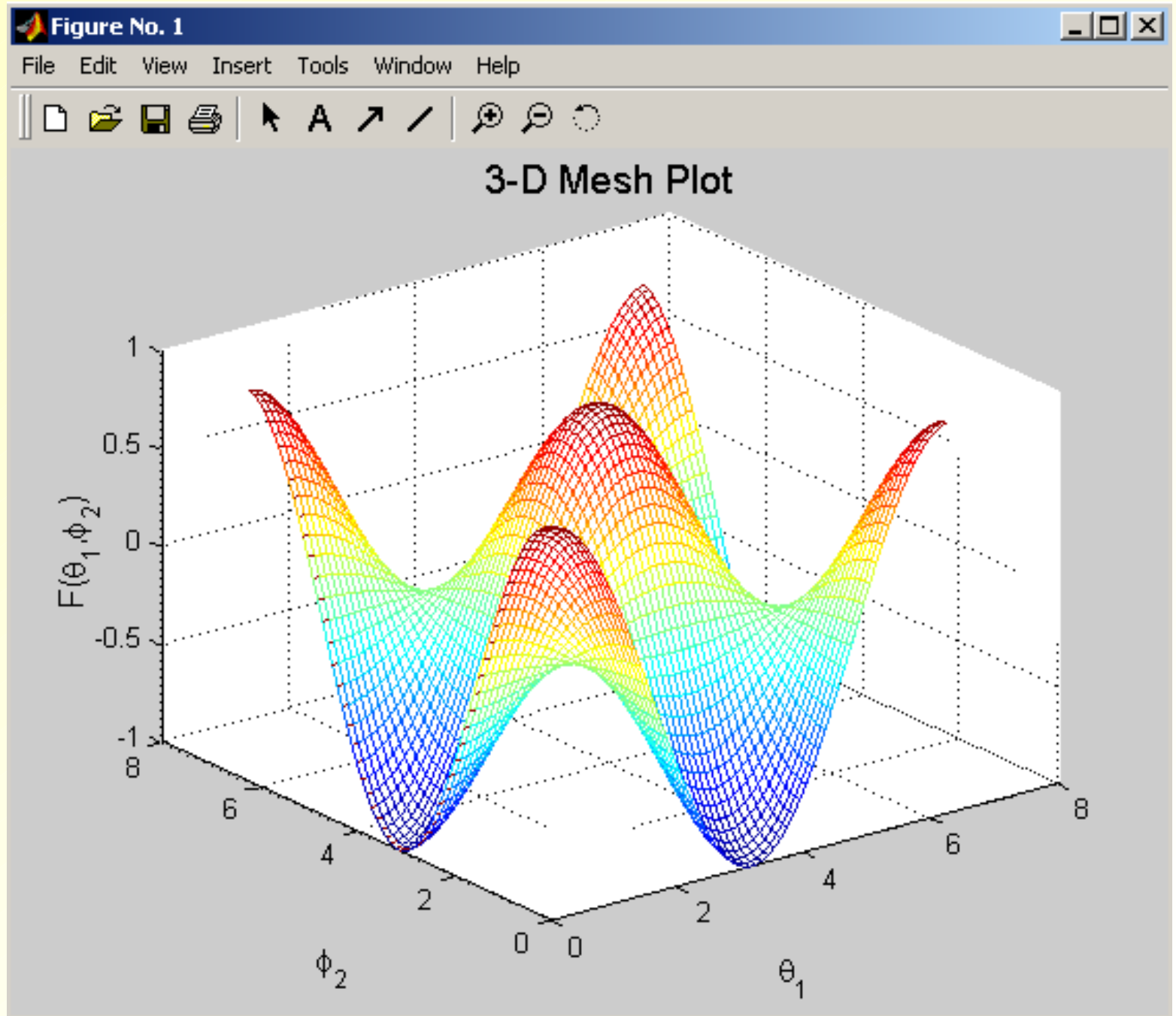
Visualization

2-D Graphs

**3-D Graphs**

Saving Graphs

Presentation



# 3-D Graphs - Surface Plots

Programming

Scripts

Functions

Debugging

Conditional Execution

Visualization

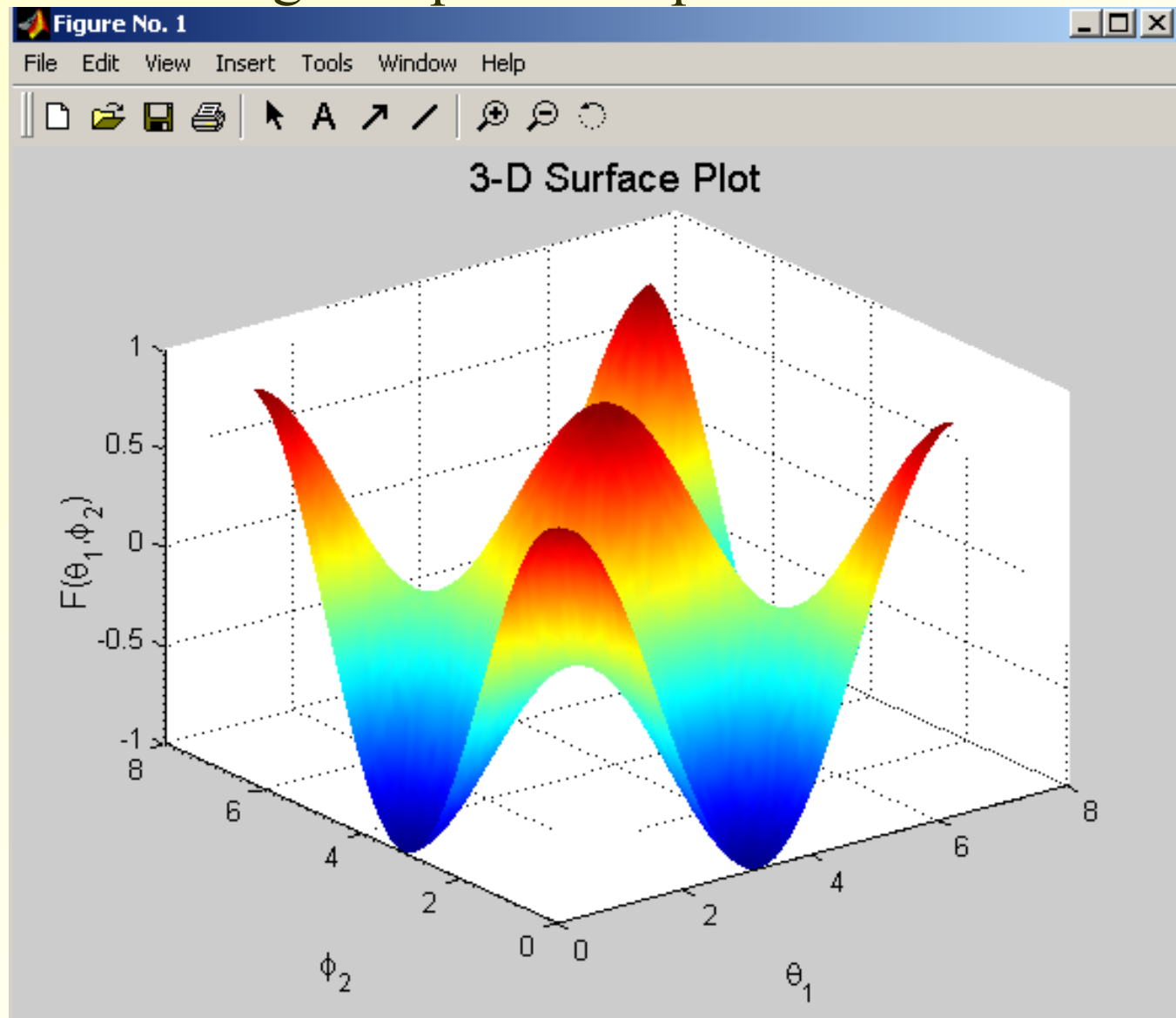
2-D Graphs

**3-D Graphs**

Saving Graphs

Presentation

- Use 'surf' instead of mesh to fill the plot with color
- Use 'shading interp' to interpolate the color smoothly



# Saving Graphs - .fig Files

Programming

Scripts

Functions

Debugging

Conditional  
Execution

Visualization

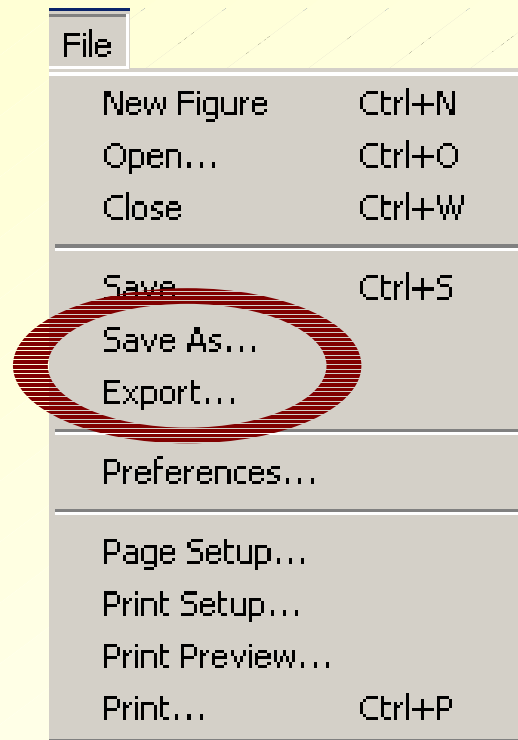
2-D Graphs

3-D Graphs

**Saving  
Graphs**

Presentation

- Figure saving and exporting are available through the figure's file menu



- ‘Save As...’ enables you to recreate the figure later
  - be sure to specify a file extension ‘.fig’
- Recreate the figure with the ‘Open...’ menu command

# Saving Graphs - Exporting

Programming

Scripts

Functions

Debugging

Conditional  
Execution

Visualization

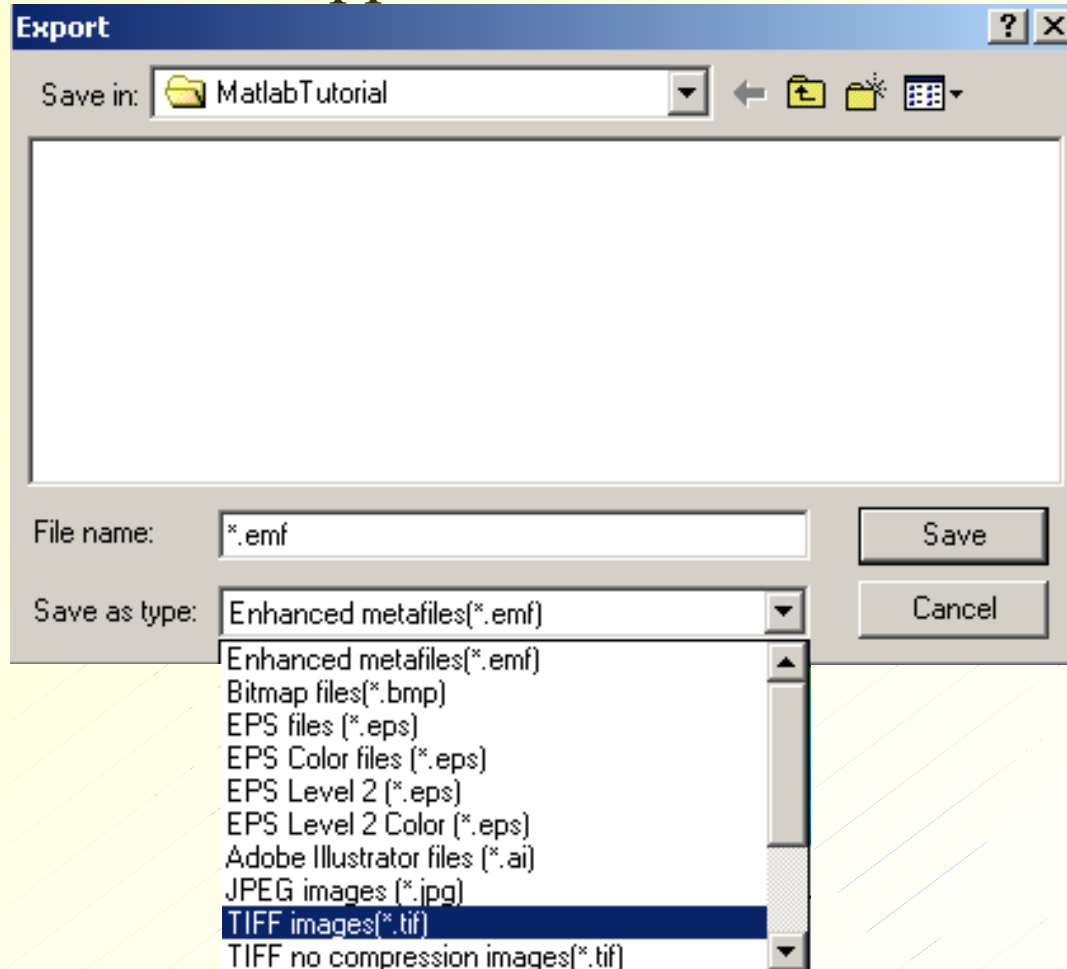
2-D Graphs

3-D Graphs

**Saving  
Graphs**

Presentation

- Export saves a graphic file in one of many formats for use in other applications



- TIFF and JPEG are useful for including in Word, PowerPoint, and WWW documents

# Presentation - Microsoft Office

Programming

Scripts

Functions

Debugging

Conditional  
Execution

Visualization

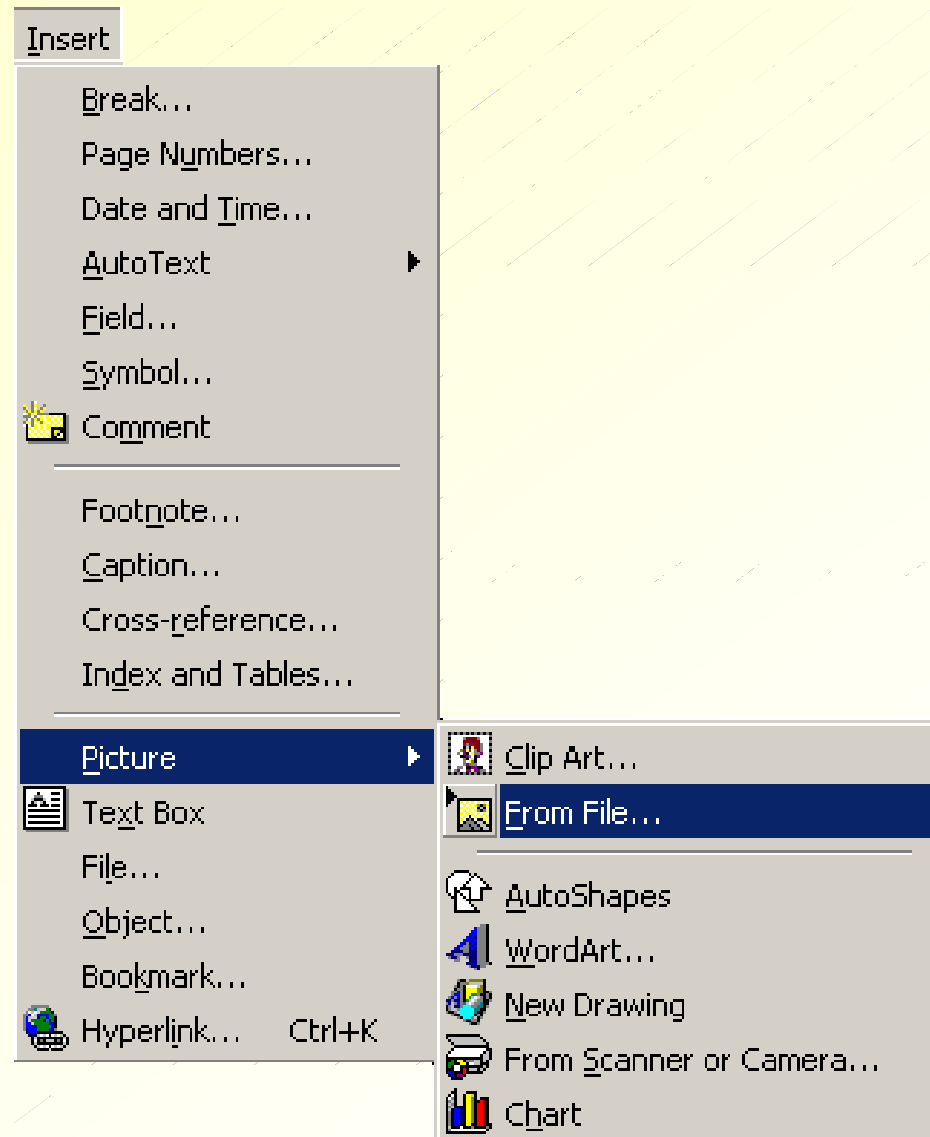
2-D Graphs

3-D Graphs

Saving  
Graphs

**Presentation**

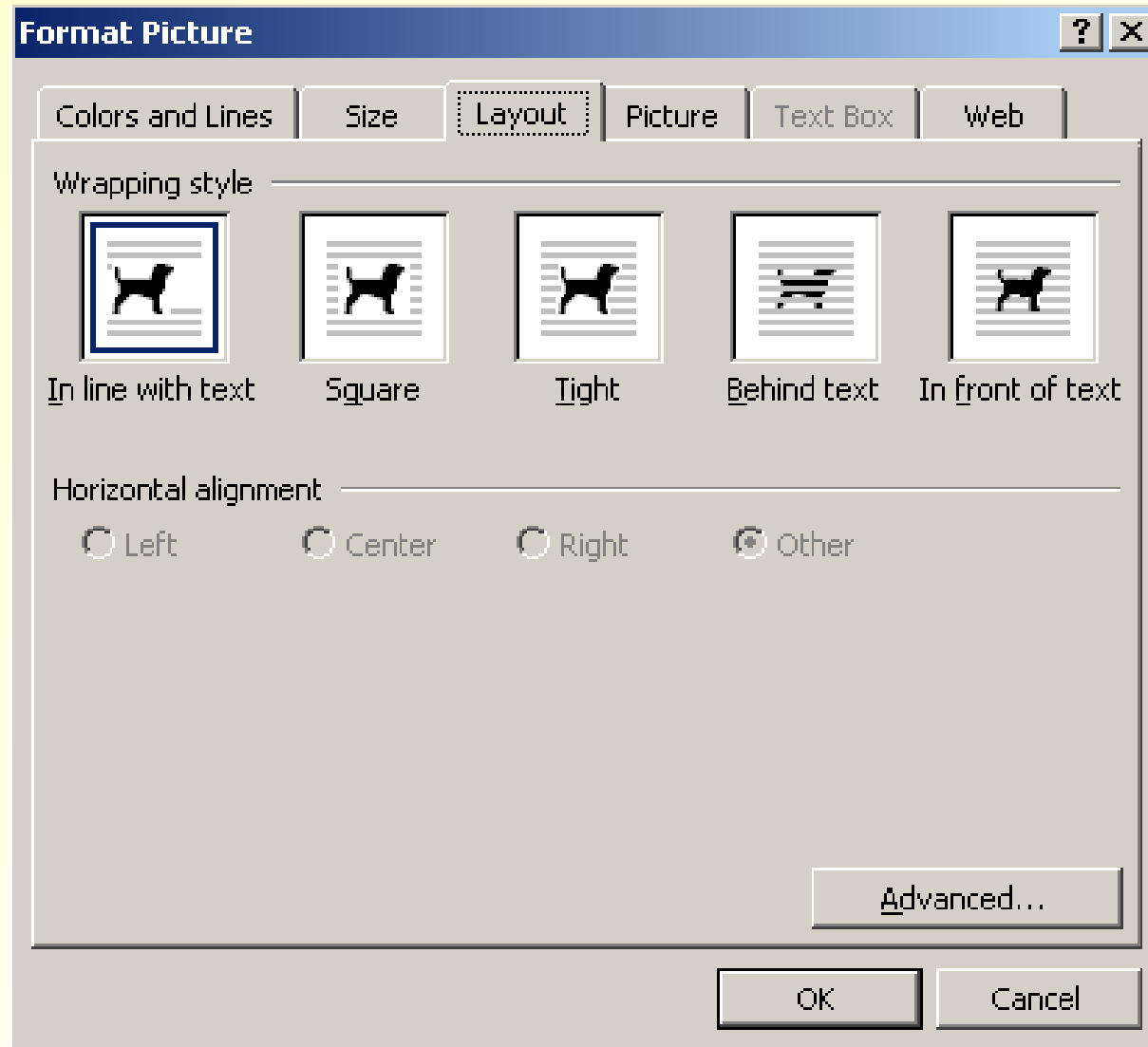
- In Microsoft Word or PowerPoint, insert a graphic into your document from the 'Insert' menu



# Presentation - Microsoft Word Layout

Programming  
Scripts  
Functions  
Debugging  
Conditional Execution  
Visualization  
2-D Graphs  
3-D Graphs  
Saving Graphs  
**Presentation**

- Double-click the graphic and choose the 'Layout' tab to select text wrap options



# Presentation - LaTeX

Programming  
Scripts  
Functions  
Debugging  
Conditional  
Execution

Visualization  
2-D Graphs  
3-D Graphs  
Saving  
Graphs  
**Presentation**

- Exporting a graphic as encapsulated PostScript (EPS) is very useful for preparing typeset documents as with LaTeX
- In the preamble of your LaTeX document, type `\usepackage{epsfig}`
- To create a floating figure within your document:

```
\begin{figure}
\begin{center}
\begin{tabular}{c}
\psfig{figure=SurfaceColor.eps,width=5in}
\end{tabular}
\end{center}
\caption{Surface plot of my function.}
\end{figure}
```