MathViews™ for Windows

A MATLAB® Compatible

Interactive Program with Built-in Graphics

for

Digital Signal Processing

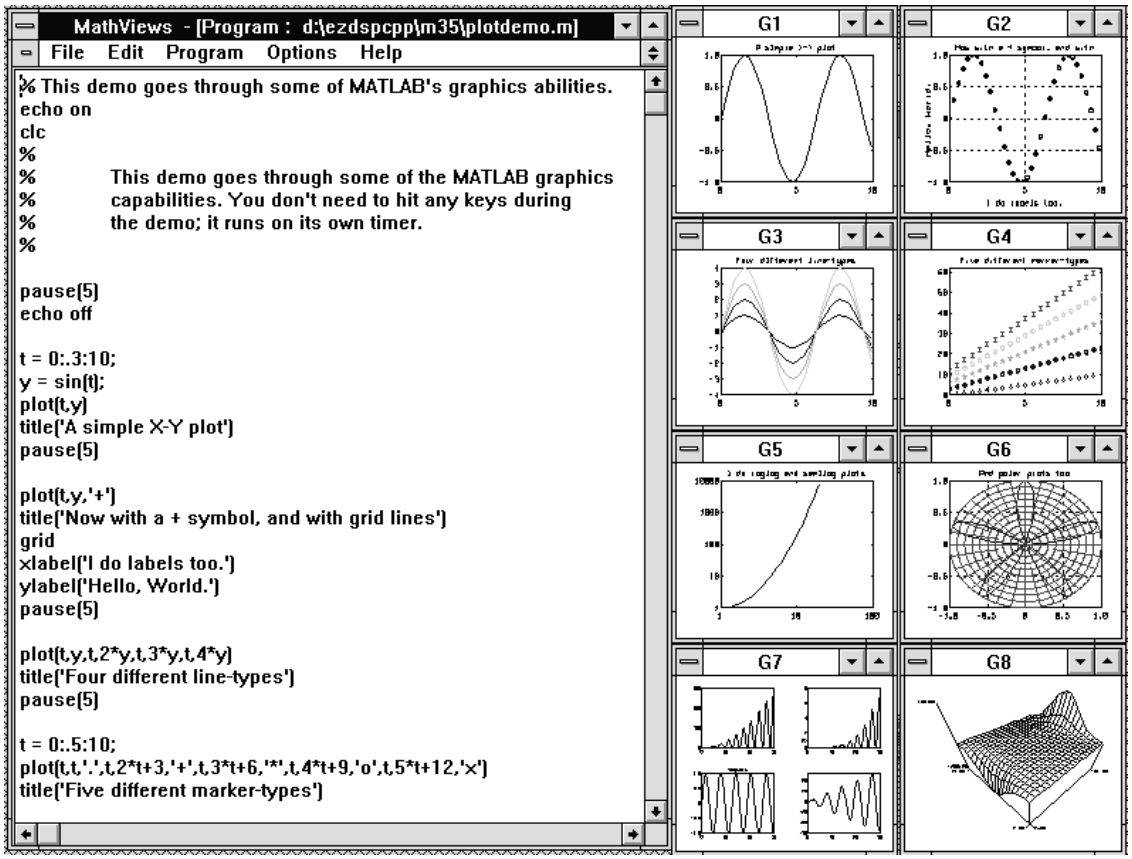
and

Linear Algebra

by

The MathWizards™

MathViews™ Demonstration Disk



Copyright © 1992-1994, The MathWizards™ all rights reserved.

All Rights Reserved

The MathWizards™
PO Box 22564
San Diego, CA 92192
Tel: (619) 552-9031
Fax: (619) 552-9031
CompuServe: 70274,2564
INTERNET: 70274.2564@compuserve.com
Printed, Developed, and Assembled in the United States

Warning

MathViews software and the manual are protected by United States Copyright Law and are licensed for use on one computer per copy only.

List of Trademarks

MATLAB is a registered trademark of The MathWorks, Inc.
MS-DOS is a trademark of Microsoft Corporation.
IBM and IBM PC are trademarks of International Business Machines Corporation.
Windows is a trademark of Microsoft Corporation.
MathViews is a trademark of The MathWizards.
MathPad is a trademark of The MathWizards.
AutoAssign is a trademark of The MathWizards.

MathViews™ the First MATLAB® Compatible Interpreter under Windows

TABLE of CONTENTS

Introduction 1
What Is MathViews 1
MATLAB Compatibility 2
System Requirements 3
What's in this manual 4
Typefaces used in this manual 4
Limitations of this Demonstration Disk Application 4

1

Getting Started 5
Installation 5
Installing MathDemo 5
Starting MathDemo 6

2

Environment 7
User Interface Windows 7
Program Window 7
Interactive Window 7
Output Window 8
Graphics Window 8
Menus 9
File Menu 9
Edit Menu 10
Program Menu 11
Options Menu 12
Help Menu 13
Control Menu (–) 13

3

Tutorial 15
Lets Try Some Examples 15
Running an Example M-file 17
AutoAssign Demonstration 18

Fundamental Concepts....18
Variables....19
Complex Variables....19
Vectors and Matrices....20
Matrix and Array Operations....22
Operations....22
Matrix Manipulations....25
Piping....33
Dependence Assignments....34
Graphic Visualization....36
Creating X-Y Plots....37
Creating Polar Plots....39
Creating Contour Plots....40
Creating Three-Dimensional Plots....41
Relational And Logical Operations....42
Relational Operators....42
Logical Operators....43
Loops And Conditionals....44
Conditional Statements....44
Iteration Statements....46
Break Statement....48
Creating Programs and Functions....49
Scripts....50
Functions....50
Modules....51

4

Function Listing....53

Introduction

What Is MathViews

MathViews is a powerful, high-level, interactive, SCIENTIFIC COMPUTING ENVIRONMENT. It is designed for engineers and scientists who need a powerful visual computational and analytical tool to assist them in solving their mathematical problems. MathViews provides simple access to some of the most difficult-to-program yet most frequently used algorithms in engineering and scientific computing. Algorithms encompass such topics as linear algebra and Digital Signal Processing (DSP). MathViews also has extensive graphical visualization capabilities. In addition, MathViews is compatible with existing MATLAB® M-files.

MathViews serves as a calculator, analysis tool, modeling tool, graphics package, and applications development system combined into one powerful application. The package provides an extensive set of arithmetic functions customized and optimized for technical users.

The most unique and time-saving aspect of MathViews is its ability to process arrays of numbers. It provides a comprehensive set of operators and functions for performing real and complex vector and matrix operations, 1- and 2-dimensional FFT, convolutions and correlation, and data manipulation. MathViews also has complex graphics such as polar, linear, 3-D hidden line, and contour plots.

MathViews is a command driven program. This means that each function of the program is initiated by typing a command at the keyboard. MathViews is more than just a simple interpreter. Each command is actually compiled as the command is entered rather than interpreted at run time. This feature allows for instant syntax checking of the input to provide immediate feedback. The incremental compilation enables the final interpretation to occur much faster than with any conventional interpreter, resulting in a much faster program development and execution cycle.

Most tasks in MathViews are solved by using built-in functions and commands. MathViews can be easily extended by creating new commands using the MathViews language. Programming under MathViews is extremely easy because of its natural language syntax and versatile subroutine structure.

MathViews has powerful debugging tools. You can select and execute a specified section of code. You can also single-step your program and selectively trace into or around any M-file functions called by the program. An animate mode lets you view each line as MathViews steps through your program.

MathViews library management freature makes building large projects easy. Simply put all the functions of a specific project into a MathViews module, compile the module, and instantly, all the functions in the module are available for use. By contrast, in Matlab™ each module can only have one function defined.

MathViews also fully supports the Windows Dynamic Data Exchange (DDE) standard for communicating with other Windows applications. MathViews can operate as either a DDE server or client. All of the powerful features of MathViews are now available to your other applications. You could, for example, execute a M-file from an Excel spreadsheet and pass the results back to Excel. MathViews supports DDE cold, warm, and hot links.

You can also extend MathViews by writing your own Windows Dynamic Link Library (DLL) modules. DLLs provide a common means of adding custom code to applications under Windows. The MathViews DDE layer and Graphics layer are examples of DLLs which can be added to MathViews. You can develop DLLs in any language, i.e. Pascal, C or C++, which supports Windows DLLs.

MathViews was crafted in C++ for easy portability to other computing environments. Accuracy is assured by performing all calculations using double precision numbers.

MATLAB® Compatibility

MathViews is compatible with MATLAB syntax and will execute MATLAB M-files. However MathViews will execute MATLAB M-files only when all the interdependent M-files are present. The highest level of compatibility is provided when using M-files which utilize only built-in functions of MathViews.

The MathViews integrated environment is much more versatile than the simple environment of the DOS version of Matlab™. Features of MathViews include:

- Different windows for editing, text output, graphics output, and interactive input.
- Cut/Copy/Paste of commands and script files; Copy/Print for plots.
- Back scrolling of text for examining past results and statements in all MathViews windows.
- MathPad™ utility for editing multiple M-files under Windows.
- Building and compiling of MathViews libraries. Multiple M-files can be combined into a single module and compiled into a MathViews library. The library can be brought into MathViews with a single command.
- Precompilation of M-files and libraries.
- Execution of a selected section of consecutive statements in an M-file.
- A mechanism for stepping through M-file statements or selected number of statements.
- AutoAssign™ for automatic variable updates; for example, y := abs(x), when x is changed, y is automatically updated.
- C syntax for simple and compound assignment(+=,*=,...); postfix operators(++,--).
- Multiple graphics windows, with quick zoom utility.
- Subplot utility supports up to up to 81 subplots, in a 9x9 pattern.
- Easily extensible through Windows Dynamic Link Library (DLL) interface.

- Fully supports Windows Dynamic Data Exchange (DDE) for interfacing to other Windows applications.

System Requirements

MathViews runs on the IBM PC family of computers, including the AT, 386, 486, along with all true IBM compatibiles. MathViews is designed to operate under the Windows 3.0 or 3.1 environment.

Minimum system:

IBM PC/AT, 386, 486, or compatible

DOS 3.3 or higher

Windows 3.x

Microsoft compatible mouse

20 MB hard drive minimum

2 MB of RAM

EGA display or higher resolution monitors

Suggested system:

IBM 386 + coprocessor or 486

DOS 5.0

Windows 3.x

Microsoft compatible mouse

40 MB hard drive

4 MB of RAM

VGA display or higher resolution

What's in this manual

This manual explains how to use the features of MathViews to quickly solve complex analytical problems. MathViews is used throughout this manual to refer to the complete MathViews package. MathDemo is used to refer to items specific to the Demonstration version of MathViews. The manul is divided into four main sections:

Getting Started

This section describes how to install and start MathDemo. It includes detailed installation instructions and how to invoke the program from the Windows environment.

Environment

The Environment section describes the MathViews user-interface. It includes a description of the primary user-interface windows and the menu structure.

Tutorial

This section provides an introduction to the features of MathViews. It covers features such as matrix and array manipulations, language control structures, and graphics.

Function List

This section contains a list of all of the functions available in the complete MathViews package.

Typefaces used in this manual

This manual uses the special fonts to represent things like menu commands, commands, and functions:

`Monospace type`	This typeface represents text as it appears on the various text based window in MathViews. It is also used to identify MathViews commands and functions.
Boldface	This typeface is used for MathViews reserved key words and for items which can be selected from a menu.

Limitations of this Demonstration Disk Application

There are a few minor limitations of MathDemo that you should be aware of:

- ~~The maximum number of elements for a vector or a matrix is 32 elements.~~
- Each session of MathDemo is time limited to approximately 10 minutes.
- Only a subset of the complete MathViews functions are provided.

1

Getting Started

Installation

MathDemo and all of its related files must be installed before it can be properly used. An installation program is provided with MathDemo to automatically do the complete installation.

Installing MathDemo

There are several things that are important to note before installing MathDemo:

- MathDemo requires that a math coprocessor has been installed. If one is not found, the installation program will alert you, and MathDemo WILL NOT BE INSTALLED.
- The MathDemo installation program does not modify your WIN.INI file in any way.
- MathDemo requires 4 MB of hard disk space. Please ensure that enough room is available on your hard disk BEFORE beginning the installation.
- You should be familiar with Windows before using MathDemo.

Procedure

1. Turn your system on if it is not already started.
2. Insert the MathDemo Installation Disk into the floppy Disk Drive. This can be A: or B:, as appropriate.
3. Make sure your system is in Windows. If in DOS, type `win` and press the **[Enter]** key..
4. Select the Run option from the Windows Program Manager File menu..
5. In the Run dialog box, type the appropriate floppy drive letter (A: or B:), **setup** and then then **[Enter]** key.
6. The MathDemo installation program will prompt you for all required information.

   The installation begins, and the message bar displays a continuous indication of the percent completion of the installation. Note that the program creates a new applications group, named MathDemo, containing MathDemo.

Starting MathDemo

If your computer started at the DOS prompt, you can enter the Windows environment by typing **win**, and pressing the Enter key.

MathViews is started by either double clicking on the MathViews icon or by selecting the run option in the Windows Program Manager file menu option and then typing **mathdemo**. After a moment, you will see the MathViews Copyrights window with an OK button. This window will stay on the screen for approximately two seconds before continuing with the initialization of the MathViews environment. Upon starting, MathViews executes the 'startup.m' M-file. You can select the ok button to immediately close the Copyrights window.

2

Environment

MathViews is developed to work in the Windows environment. This allows you to take full advantage of other excellent Windows utilities, like the Clipboard, PaintBrush, Word, and other third party applications. To take full advantage of MathViews, you should familiarize yourself with Windows.

MathViews is a computing environment specifically designed for matrix computations. It provides many unique and convenient features that make it particularly well suited to scientists and engineers doing demanding circuit designs, filter designs, signal processing, numerical analysis, and many other fields. MathViews is easy to learn and master.

User Interface Windows

You interact with MathViews through four windows: Program. Interactive, Output, and Graphics. These windows allow you to enter commands and programs, and to view text and graphical output of you data.

Program Window

The **Program** window is where you load, edit, program, and run MathViews script files. Script files in MathViews are called M-files. The Program window is a simple editor that is easily permits you to update changes in an M-file and to immediately execute the new statements. It is not necessary to close the file and execute the M-file command afterwards, as in the Matlab™ language. Text can be cut and pasted into the Program window.

Interactive Window

The **Interactive** window allows you to immediately enter and execute commands. The interactive window is you interface to the MathViews interpreter. The Interactive window can be used as a testing ground for statements to be inserted in the M-file contained in the Program window. This versatility permits you to test statements as you compose application-specific M-files. Text can be cut and pasted into the Interactive window.

Output Window

The **Output** window is where all the text output goes. The Output window keeps a 64 line buffer. You can scroll the output window to review previous results.

Graphics Window

The **Graphics** window is independent of the other MathViews windows, and can therefore reside anywhere on the screen. Graphics windows are used to display graphical outputs or your data using MathViews' extensive set of functions for generating x-y plots, contour, and three-dimensional plots.

MathViews allows you to generate multiple graphics windows, one for each piece of data you wish to plot. MathViews also has commands to let you overlay multiple data sets on the same graphics window. This allows you to easily examine you data.

Any area of a Graphics window image can be zoomed in and out by simply selecting the area with the mouse. To select an area, press on the left mouse button and hold it down while you highlight the desired area. To return to the original scale of the image just click on the right mouse button.

Each Graphics-window image can be scrolled both vertically and horizontally. The windows can be scrolled in either original-scale or zoomed mode.

Graphics images in a Graphics window can be printed by choosing the print option in the Control menu of the Graphics window.

Note:

In each of the windows described above, when the window is in focus, i.e. when the caption bar is highlighted, text or graphics can be copied to or pasted from the Windows clipboard. However, in a Graphics window, the user is only permitted to copy the current graphics image to the clipboard or print the image onto a plotter or graphics printer (in the current version).

Menus

The main MathViews window has a top-level menu bar that provides a list of choices for which actions to take. The top level menu options are: File, Edit, Program, Options, and Help. In addition, MathViews supports the Windows standard Control menu. The control menu is accessed by clicking on the button in the upper left corner of the MathViews window.

File Menu

The file menu contains the following menu items: New, Open, Save, Save as…, Open Library, Save Library as…, and Exit.

New

Creates a new M-file editing session in the Program window. Issuing the new command will close the current file, before beginning a new editing session.

Open

Opens an ASCII text M-file for editing.

A dialog box will prompt you to insert the filename to be opened. Selecting a file can be accomplished in one of two ways. The first way is to simply type in the file name. The second way is to click on its name in the list of files provided in the dialog box. You can also open other directories for search by double-clicking on the drive letter or the path specifier in the dialog box. When the dialog box comes up, only files with the '.m' extension are displayed. To see other files, type in the appropriate extension.

Save

Saves the current editing session in the Program window onto disk.

MathViews saves the data into the file if the filename to be saved to already exists. If the file has not previously been saved, a dialog box will be displayed to prompt you for a filename.

Save As ...

Saves the current editing session in the Program window onto disk.

A dialog box will prompt you for the filename to be saved to. You can specify any new filename or an entirely new path. The '.m' extension must be explicitly entered.

Open Library

Opens a Library file.

A dialog box will prompt you to insert the library filename to be opened. Library files are files with the '.l' extension. Selecting a file can be accomplished by either typing in the filename or clicking on its name in the list of files provided in the dialog box. You can also open other directories for search. When the dialog box comes up, only files with the '.l' extension are displayed. To see other files, type in the appropriate extension.

Save Library as ...

Save the current editing session in the Program window onto disk as a library file.

A dialog box will prompt the user for the filename to be saved to. The user can specify any new filename or an entirely new path. The '.l' extension must be explicitly entered.

Exit

Exits the current session of MathViews, and returns to Windows.

If the current editing session Program window has not been saved, a dialog box will ask you to either save changes and exit (YES option) or abandon changes and exit (NO option).

Edit Menu

The Edit menu contains the following menu items: Clear, Select All, Cut, Copy, Paste, Delete, and Goto Line…

Clear

Clears the currently selected (highlighted) area.

Select All

Selects all the text in the currently active MathViews window.

Cut (Shift+Del)

Cuts the currently selected (highlighted) area onto the Windows clipboard, removing the selected area in the currently active window.

You can cut a selected area from either the program window or the interactive window. Those are the only two windows from which items can be cut.

To select an area, place the cursor to the desired beginning location, click the left mouse button, drag to the desired end location, and release the mouse button. To unselect text, click on the mouse button again.

Copy (Ctrl+Ins)

Copies the currently selected area into the Windows clipboard, without removing the selected area in the currently active window.

The copy option can be used in all MathViews text-based windows: Program, Interactive, Output. In the Graphics windows, the copy command is issued through the control menu on the Graphics window.

Paste (Shift+Ins)

Pastes the content of Windows clipboard onto the currently active window.

MathViews does not allow pasting into either the Output window or the Graphics windows. Pasting ASCII text is permitted in either the Program window or the Interactive window.

Delete (Del)

Deletes the character to the right of the current cursor position or deletes the selected area.

Deletions can only be made from the Program window or the Interactive window. To delete the character to the left of the cursor, use the backspace key.

Goto line…

Lets you immediately go to a specified line in the currently active window.

Program Menu

The Program menu contains the following menu items: Run, Step, Continue, Reset, Stop, and Compile. These items only affect the Program window.

Run (Ctrl+r)

Runs the current M-file in the Program window. If a highlighted area exists, choosing the run option will execute the statements in the highlighted area instead.

Animate (Ctrl+a)

Automatically single step through the current program or highlighted portion thereof.

The sequence will animate, pausing between execution of statements. The animation sequence operates as if you were stepping through the program.

Step (Ctrl+s, F8)

Executes one statement.

By continuously typing in 'Ctrl+s', you can step through the M-file in the Program window. If a selected area exists, then only the statements in the selected area will be executed.

Trace into (Ctrl+r,F7)

Single-step into the specified function. Note, you can only trace into functions which are defined in M-files.

MathViews will automatically display the code from the M-file which describes the function. You can single step through the M-file by entering 'Ctrl+s'.

Continue (Ctrl+n,F9)

Resumes execution of a stopped run session.

The continue option lets you resume execution of the M-file, or the selected part of it, after a stop command was issued or an error occurred.

Reset (Ctrl+x)

Resets the starting position for execution to the beginning of the M-file in the Program window.

Stop (Ctrl+y)

Stops the session currently running in the Program window. A continue or run command restarts the session.

Compile

Compiles the current editing session into MathViews memory.

The current editing session in the Program window must be a file with the '.M' extension. Otherwise, Mathviews will not compile the file. After the compilation, the old M-file function in memory will be replaced by the most recently compiled M-file.

Options Menu

The Options menu contains the following items: Cascade, Tile, Arrange, Path…, and Format…

Cascade

Cascades the three text windows in MathViews: Program, Interactive, Output.

Tile

Tiles the three text windows in MathViews: Program, Interactive, Output.

Arrange

Manually arranges the location of windows.

Path ...

Changes the search path of MathViews. The search path is used in MathViews to search for M-files, libraries, ′.mat′ and ′.dry files′.

A dialog box will show the current search path, and prompts you to enter a new search path. You can either choose CANCEL to exit without making any changes to the current search path, or you can type in a new search path and click OK.

Format ...

Selects floating-point output format.

Instead of specifying the floating-point numbers output format through the format command in the interactive window, this option permits you to interactively select an output format using the mouse.

Help Menu

The Help menu contains the following menu items: Help, About ..., and Memory.

Help

Display a help window. The help window lets you view help information contained in M-files.

About ...

This selection displays version and copyright information for MathViews.

Memory

Displays the current memory usage of MathViews.

Control Menu (–)

The control menu is a standard Windows feature. It contains the following menu items: Restore, Move…, Size, Minimize, Maximize, Close, Switch to… This is a brief discussion of its use with MathViews.

Restore

Returns the window to the default size and position.

Move

Changes the position of the application window on the screen.

An easy way to accomplish the same action is to click on the title bar and drag the window to the desired location.

Size

Resizes the window.

An easy way to accomplish the save action is to move to the lower right corner of the window, until the mouse cursor becomes a bi-directional arrow; then click on the mouse and drag until the desired size is reached.

Minimize

Reduces the window into an icon.

Maximize

Enlarges the window to its maximum size.

Close

Closes the window of the current application. In order words, quit the application.

Switch to ...

Switches between the applications currently running under Windows.

Maximize/Minimize Button

Functions in exactly the same manner as the Maximize/Minimize item in the control menu.

3

Tutorial

Lets Try Some Examples

Click on the Interactive Window, and then enter the following commands:

```
x = [ 0 0]
y = [ 0 3.14; -imag(log(-1)) 3.14; x]
z = cos(y)
zz := sin(y)
x(1) = -imag(log(-1)) / 2
```

Notice that the variable `zz` is updated automatically. The dependency operator, :=, assigns the value of `sin(y)` to `zz`. Whenever the value of `y` is changed, either directly or indirectly, the value of `zz` is updated.

Now try:

```
exp(x)
x->exp
```

The MathViews pipe operator, `->`, is an alternative method for entering an expression.

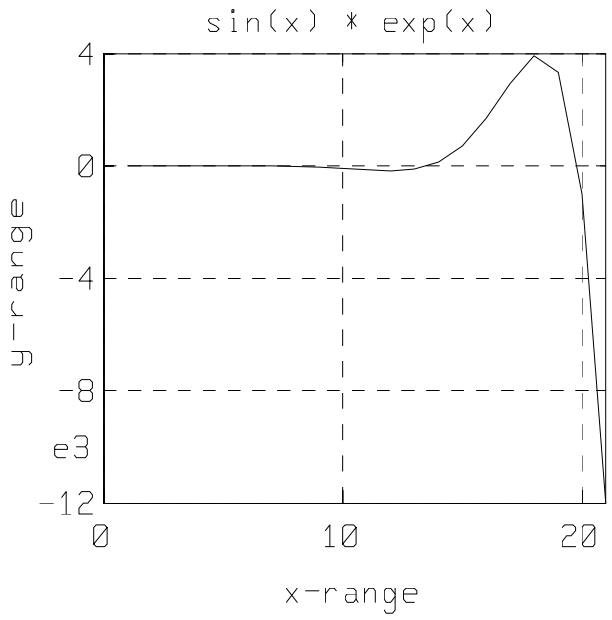
Now try entering Matrices:

```
A = [ 1 2 3; 4 5 6; 7 8 9]
A'
A += A
A -= A
```

MathViews offers a number of method for creating matrices and performing operations on them.

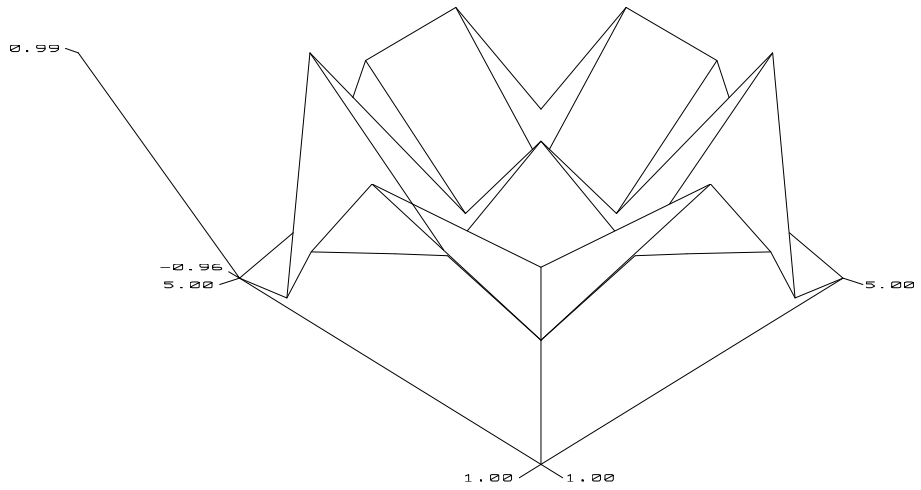
MathViews lets you visualize your data with its extensive set of graphical plotting functions. Lets try plotting some data using a linear X-Y plot:

```
x = 0:.5: 10
plot(sin(x).* exp(x))
grid
title('sin(x) * exp(x)')
xlabel('x-range')
ylabel('y-range')
```

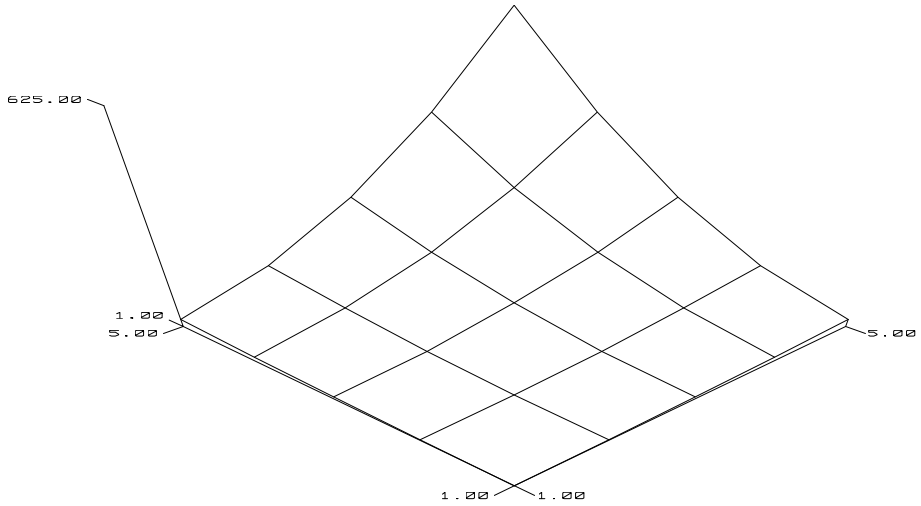


Now lets try plotting some data using a 3-dimensional hidden line removal plots:

```
x = 1:5
y = 1:5
z = x'*y
fsin = sin(z)
fxsq = z .* z
mesh(fsin), mesh(fxsq)
```



plot of fsin



plot of fxsq

MathViews lets you quickly zoom into any section of your plots. With the mouse, try zooming in and out of any of the plots you have just created by selecting (highlighting) an area within the graphics plot to zoom on that area. Try selecting another area. Click anywhere in the window to return to the original scale.

Running an Example M-file

Click on the FILE menu in the Application window, and choose OPEN.

Locate the MATHVIEW directory, and select `MVDEMO.M`.

At this point, you have four options:

- Run the complete M-file.
- Select a portion of the M-file to execute.
- Step through the M-file.
- Step through a selected statements of the M-file.

From the Application menu, choose the Program option, then choose Run. You will see the statements highlighted as they run through the interpreter.

You can also step through the M-file statements by choosing the Step item under the Program option.

Lastly, try selectively executing statements in `MVDEMO.M`.

Use the mouse to select any statements you want to run, and choose the Run item again. You will observe that only the highlighted statements are executed.

AutoAssign Demonstration

Load the `AUTOASGN.M` file into the program window.

Here is the listing of the demonstration M-file:

```
x=1:16
n=1
y1:=autoplt(x,x.^2,3,'x.^2');
y2:=autoplt(x,x.^3,4,'x.^3');
y3:=autoplt(x,abs(fft(x)),5,'abs(fft(x))');
y4:=autoplt(x,sin(6*n*x/max(x)),6,'sin(x)');
```

Choose the Run item to execute the `AUTOASGN.M` M-file statements.

Let the M-file run to completion. You will see four plots created on the screen.

Now, type

```
x = 1:32
```

Notice the graphs are automatically replotted to reflect the change in `x`.

This example demonstrates how easy it is to create a graphical spreadsheet using MathViews' AutoAssign technology.

The remainder of this tutorial gives more detailed descriptions on how to perform operations in the MathViews environment.

Fundamental Concepts

Operations in the MathViews environment are based on statements. Statements are of the form:

variable = expression

or

expression

An expression is a sequence of operators, operands and punctuators that specifies a computation. Expressions are used to assign values to variables and to carry out a specific computation. MathViews uses the internal variable '`ans`' if a variable is not given.

The principal data type in MathViews is a matrix of either real or complex values. Matrices of a single row or column can be thought of as vectors. MathViews has a number of functions and operators that manipulate matrices or vectors. You can also manipulate matrices and vectors on an element by element basis.

Variables

Results of computations are kept in variables. As in most programming languages, a variable occupies memory space and stores a value assigned to it. Assigning a value to a variable also specifies its type. For example,

```
x = 2
  x =
    2
```

```
sq = sin(x)^2 + cos(x)^2
  sq =
    1
```

MathViews automatically displays the result of evaluating the expression in the **Output** window. You can suppress the display in the **Output** window by placing a semicolon at the end of each assignment statement. For example,

```
angle = cos(3*pi);
```

Multiple assignment statements can be placed on a line by using the semicolon between statements. For example,

```
x = 2; sq = sin(x)^2 + cos(x)^2;
```

A MathViews variable can also be an ASCII string. An ASCII string must be enclosed in single quotation marks. Text strings are assigned in the following manner:

```
str = 'MathViews is Powerful';
```

Complex Variables

Complex variables store complex (real and imaginary parts) rather than real numbers. The individual parts of a complex variable can be extracted using the `real(z)` and `imag(z)` MathViews functions. A complex variable can be assigned in one of the following ways:

```
z = 2 + 4j
z = 2 + 4i
z = 2 + i * 4
z = 2 + 4 * sqrt(-1)
```

```
eio = cos(2) + i * sin(2)
exp(2i)
```

You can use either `i` or j to specify complex values. MathViews uses `j` to represent the imaginary part when it prints complex values. The multiplication symbol "`*`" is not required if the `i` or `j` is used after the real number representing the imaginary part. If the `i` or `j` is used ahead of the real number the multiplication symbol must be entered.

```
z = 2 + 4j;
```

Valid assignment statement.

```
z = 2 + j4;
```

Invalid assignment statement, must use "".*

Values in radians and degrees are also expressed in terms of complex numbers.

Vectors and Matrices

Vectors and Matrices can be thought of as arrays of values. Vectors are matrices with either one row or one column. From here on, the terms vector and matrix will be used interchangeably except where the distinction is meaningful. Matrices can have elements that are either complex or real. A matrix is considered to be complex if any element of the matrix is complex.

A MathViews matrix can be entered in the following manner:

```
A = [ 2 3 1; 4 3 2; 5 2 1]
```

```
A =
  2  3  1
  4  3  2
  5  2  1
```

```
A = [sqrt(-1) cos(0) sin(pi/2); exp(3) 2^2 5i]
```

```
A =
    1j          1                1
 20.0855        4                5j
```

Row elements can either be separated by a comma or spaces. A semicolon is used to separate rows. Matrix assignments must be enclosed in square brackets, `'['']'`.

Elements of a matrix can be accessed using the format `A(index)`, where `index` begins at one. Elements are indexed in a columnwise fashion. For example,

```
A =
  2   3   1
  4   3   2
  5   2   1
```

```
A(4)
```

```
ans =
    3
```

```
A(3)
```

```
ans =
    5
```

```
A(2)
```

```
ans =
    4
```

You can also assign values to matrix elements that have not yet been allocated. For example,

```
x = [ 2   3   1 ]
```

```
x =
    2   3   1
```

```
x(5) = 4
```

```
x =
     2   3   1   0   4
```

Assigning a value to the nonexistent fifth element in the `x` array causes an automatic expansion of the matrix to five elements and assignment of zero to the fourth element. The variable `x` must first exist for the assignment to be performed successfully.

Matrix and Array Operations

MathViews makes a distinction between performing matrix type operations on a matrix and operating on a matrix on an element-by-element basis. The term *array* is used to refer to a matrix on which elemental operations are performed. Operations on matrices are performed with the matrices as a whole, whereas operations on arrays are performed on an element-by-element basis.

Operations

Transpose.

MathViews uses the `'` (apostrophe) character to denote transposition. An example will best explain the operator's function:

Let

```
A =                          x =
   2   3   1                      1   2   3   4
   4   3   2
   5   2   1
```

Then

```
x'
ans =
   1
   2
   3
   4
```

```
A'
```

```
ans =
  2  4  5
  3  3  2
  1  2  1
```

Addition and Subtraction.

Both addition and subtraction are performed on an element by element basis. The dimension of both operands must be equal, except in the case where one is a scalar. MathViews will generate an error message if incompatible operands are used. For example,

```
x=(x' + 5)'
```

```
x =
  6  7  8  9
```

```
A + A
```

```
ans =
   4    6    2
   8    6    4
  10    4    2
```

```
A= [ 1 2 3; 1 2 3] + [3 4; 2 3]
```

This is an error, since the dimensions of the first operand are not equal to the dimensions of the second operand.

Matrix Multiplication and Division.

Matrix multiplication requires that the second dimension of the first operand be the same as the first dimension of the second operand. For example, if the first operand is `m x 3`, then the second operand must be `3 x n`, and the resultant matrix will have dimensions of `m x n`. The relationship among the numbers of columns and rows of the operands and resultant matrix is as follows:

```
   A     *     B     =     C
 mxn         nxp         mxp
```

For example,

```
A =                          B =
   3 -1   4                      1   0 -3 -2
   2   0   1                    -2   4   5 -1
                                 3 -1   0   6
```

```
A * B
ans =
  17   -8   -14   19
   5   -1    -6    2
```

MathViews provides two matrix division operators, `/` and `\`. These correspond to right-hand and left-hand division, respectively. Matrix divisions follow the general rule:

`A/B` is equivalent to `A*inv(B)`

`A\B` is equivalent to `inv(A)*B`

Array Multiplication and Division.

MathViews performs array multiplication and division on an element by element basis. Array multiplication and division are denoted by placing a period (`.`) ahead of the `*`, `/`, and `\` operators. Array multiplication and division are performed as follows:

`C = A.*B` is equivalent to $C_i = A_i * B_i$

`C = A./B` is equivalent to $C_i = A_i / B_i$

`C = A.\B` is equivalent to $C_i = B_i / A_i$

For example,

```
A =
   2   3   1
   4   3   2
   5   2   1
```

```
A .* A
```

```
ans =
    4     9     1
   16     9     4
   25     4     1
```

```
A ./ A
```

```
ans =
   1   1   1
   1   1   1
   1   1   1
```

Matrix Manipulations

MathViews offers many methods for manipulating the data elements of a matrix. We will explore some of the more fundamental procedures for manipulating matrix elements in this section. The best way for learning how to manipulate matrices is to simply experiment with the operations discussed here.

Colon Operator.

The colon operator, '**:**', plays a very important role in matrix manipulation. Its simplest function is to assign a range of elements to a variable. For example,

```
A = [ 1:4; 2:5 ]
```

```
A =
   1   2   3   4
   2   3   4   5
```

The general form of the colon operator is `start:increment:end`, corresponding to a starting value, an increment, and an ending value. The increment is assumed to be 1 if it is not explicitly specified. The colon operator is a simple way to generate a row vector covering a specified range of values.

```
x = 0.5:0.5:2
```

```
x =
   0.5   1   1.5   2
```

Notice square brackets are not necessary when creating vectors using the colon operator.

```
x = 2:-0.5:0.5
```

```
x =
   2   1.5   1   0.5
```

Negative numbers are also allowed for the increment value.

```
x = sqrt(1:4)
```

```
x =
   1   1.41421   1.73205   2
```

The colon operator can also be used to pass a range of values to a function.

```
1 + 1:5
```

```
ans =
   2   3   4   5
```

```
1 + (1:5)
```

```
ans =
   2   3   4   5   6
```

The precedence of operators must be taken into consideration when creating expressions. Notice here the + operator has higher precedence than the colon operator. Care must be used when multiple operators are involved.

Creating Matrices.

Large matrices can be easily created using a combination of vectors and the colon operator. For example,

```
x = [ 1 2 3]
x = [ x, 4 5 6]
```

```
x =
  1  2  3  4  5  6
```

```
A = [ x; 5:10]
```

```
A =
    1    2    3    4    5    6
    5    6    7    8    9   10
```

Matrix Subscripting:

Matrix subscripting allows you to get access to individual matrix elements. The general form is as follows:

```
A( rV, cV)
```

The `rV` and `cV` arguments are subscripts and can be either scalars or vectors. When using subscripting you must ensure that the `A` matrix exists and that the ranges of the subscripts are subsets of the ranges of `A`. The elements of `rV` can not have values greater than the number of row of `A` and the elements of `cV` can not have values greater than the number of columns of `A`. The MathViews `exist()` function can be used to test for the existence of a variable.

If the subscript variable is a vector, then only those elements of the matrix whose indices correspond to the values in the vector are accessed. This method is best illustrated by examples:

```
A =
  1  2  3  4
  1  2  3  4
  1  2  3  4
  1  2  3  4
```

```
A(1:2, 1:2)
```

```
ans =
  1  2
  1  2
```

```
A(2,:)
```

```
ans =
  1  2  3  4
```

In the previous example, the colon operator was used with no values to select the third row of matrix `A`. Using the colon operator by itself corresponds to selecting all elements of the specified row or column. The second column of `A` could have been accessed by simply swapping the `2` and the `:`, i.e. `A(:,2)`.

```
A(:,2:3) = A(:, 3:-1:2)
```

```
A =
  1  3  2  4
  1  3  2  4
  1  3  2  4
  1  3  2  4
```

This example switches columns 2 and 3.

```
A(2:3, [1 1 2 2])
```

```
ans =
  1  1  3  3
  1  1  3  3
```

This example selects from row 2 and 3, the first column twice and the second column twice.

Matrix Reshaping:

The colon operator can also be used on the left-hand side of an assignment statement to change the dimensioning or size of the matrix. For example,

```
B =
  1  2  3
  2  3  4
```

```
B(:)
```

```
ans =
  1
  2
  2
  3
  3
  4
```

This example changes the dimensions of B from a 2-by-3 matrix to a column vector whose elements are the columns of the previous B matrix.

```
B(:) = 21:26
```

```
B =
  21  23  25
  22  24  26
```

This example changes the values of the B matrix to the elements of a vector with values from 21 to 26. Notice that B retains its previous dimensioning.

```
B(:) = [ 21 24 26 3 4 5]
```

```
B =
  21  26  4
  24   3  5
```

This example changes the values of the B matrix to the elements of the specified vector.

The matrix must already exist prior to using the colon operator to reshape matrices. The matrix must exist to maintain the same size, otherwise MathViews will create a column vector.

Matrix Shifting.

Columns and rows of a matrix can be shifted to the left and right. Shifting can be either circular or linear. In a circular shift, as elements are shifted off the end they are wrapped back to the beginning. In a linear shift, as elements are pushed off the end, a zero value is added in place of the missing element.

Here are some examples of performing linear and circular shifts:

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

```
A >> [ 2 -2]
ans =
  0   0   0   0
  0   0   0   0
  3   4   0   0
  7   8   0   0
```

This example shows a linear shift of `A` by two elements down and two elements right. Notice that MathViews zero fills the remainder of `A`.

```
A <> [ 2 -2]
ans =
  11   12    9   10
  15   16   13   14
   3    4    1    2
   7    8    5    6
```

In this example we perform a linear shift on the A matrix. Notice that in the circular case the elements that were shifted off are wrapped back around. The row shifts occur first, then the column shifts are performed.

Deleting Matrix Rows/Columns.

You can use the empty matrix notation, `A = []` to easily delete matrix elements, columns, or rows. The empty matrix `A` is defined as a 0-by-0 matrix. A column or row is deleted in the following manner:

```
A =
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
```

```
A(:, [ 1 3 5]) = [ ]
```

```
A =
   2   4   6   7   8
   2   4   6   7   8
   2   4   6   7   8
   2   4   6   7   8
```

This statement deletes column 1, 3, and 5.

```
A([2 4],:) = [ ]
```

```
A =
   2   4   6   7   8
   2   4   6   7   8
```

This statement deletes rows 2 and 4.

Logical Vectors.

Matrices and vectors can be manipulated and reshaped using logical vectors. Logical vectors are vectors whose elements are either zero or one. A logical vector must have the same number of elements as the rows or columns they are referencing. The following examples illustrate the use of logical vectors:

```
A =
   1    2    3    4
   5    6    7    8
   9   10   11   12
x =
   1   0   1   0
y =
   0   1   0
```

Notice that the matrix `A` has three rows and four columns. The vector `y` has three elements and will be used to access the rows of matrix A. The vector `x` has four elements and will be used to access the columns of `A`.

```
A(y,x)
```

```
ans =
   5   7
```

The resultant matrix is those rows and columns of `A` for which the logical vectors, `x` and `y`, have a value of one.

```
x =
   1   1   1   1
y =
   0   1   0
```

```
A(y,x)
```

```
ans =
  5   6   7   8
```

In this case the entire second row of `A` is returned since the logical vector `y` is all ones and the logical vector `x` has a one only in the second position.

```
x =
  1   1   1
```

```
A(:,x)
```

```
ans =
    1     1     1
    5     5     5
    9     9     9
```

In this example `x` is not treated as a logical vector for selecting columns from `A` since the number of elements of `x` is not equal to the number of columns of `A`. In this case, vector `x` is used as a subscript to select column one of `A` three times.

Piping

MathViews provides the pipe operator (->) to make the task of writing nested statements easier. The pipe operator allows you to send the output from one function into another function. This can eliminate the need for lots of parentheses and reduces the likelihood of making errors due too improper matching of parenthesis. For example,

```
x =
  1   2   3
```

```
abs( sqrt( cos( exp(x))))
```

This expression can be converted into a simpler form with more readability using the pipe operator,

```
exp(x)->cos->sqrt->abs
```

```
ans =
   0.954848   0.669594   0.573232
```

Dependence Assignments

MathViews provides a powerful facility for allowing you to set up automatic recalculations by assigning dependencies between variables. With the MathViews Auto-Assign™ technology you use the dependence assignment operator, `:=`, to build relationships among variables. Relationships can be nested many levels deep. MathViews will automatically update all dependent variables. For example,

```
x = 1
y = 1
z = 2
```

```
A := abs(x + y)
```

```
A =
  2
```

This statement sets up the relationship that `A` is dependent on the value of the variables `x` and `y`.

```
B := sqrt(z + A)
```

```
B =
  2
```

This statement sets up the relationship that `B` is dependent on the value of variables `z` and `A`. `B` is indirectly dependent on the variables `x` and `y` since `A` is also dependent on the value of variables `x` and `y`.

```
_list(0)
```

```
Your relations are :
Level 1 : A := x , y
Level 2 : B := A , z
```

The MathViews function `_list()` can be used to display the relationships you have set up.

```
x = 4
```

```
x =
   4

A =
   5

B =
    2.64575

Variable Changed :B =
    2.64575

Variable Changed :A =
    5
```

Changing the value of `x` to 4 results in an automatic recalculation of the variables `A` and `B`. Recall that `A` is directly dependent on `x` and that `B` is indirectly dependent on `x`. Notice that all the appropriate related lower levels are updated before higher levels are updated.

Auto-Assign™ technology and the dependent operator easily allow you to set up what-if type calculations. You can immediately see the effect changing a single value has on a set of equations.

Graphic Visualization

Operating in the highly graphical Windows environment, MathViews offers a number of functions for generating high-quality graphic displays of your data. MathViews graph types include various x-y plots, polar plots, contour plots, and three-dimensional plots. The set of plotting functions in MathViews includes:

Plotting Function	Description
`plot`	linear x-y plot
`semilogx`	log-x, linear-y plot
`semilogy`	linear-x, log-y plot
`loglog`	log-x, log-y plot
`polar`	polar plot
`contour`	contour plot
`mesh`	three-dimensional plot

A number of functions are available for annotating the graphs once they are created. These include functions for titling, axis labeling, annotation text, and grids. The following functions are available in MathViews for annotating graphs:

Annotation Function	Description
`text`	places text on graphics window
`title`	adds title to graphics window
`xlabel`	adds x-axis label
`ylabel`	adds y-axis label
`zlabel`	adds z-axis label
`grid`	places grid lines on plot

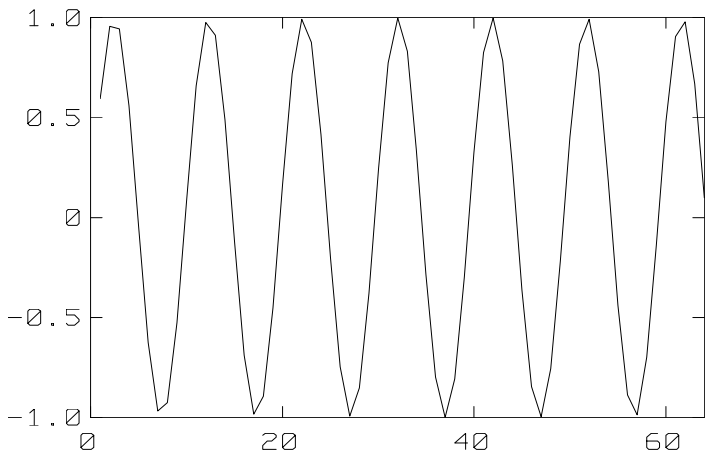
MathViews also includes a number of additional functions for manipulating and printing graphics. The following additional graphics functions are provided:

Graphics Function	Description
`print`	print the graphics window
`meta`	create Windows meta-file
`hold`	hold plot for multiple traces
`subplot`	create subplot windows
`axis`	allow manual axis scaling
`gwclr`	clear the graphics window

Creating X-Y Plots

The `plot()` function is the primary means of creating x-y plots in MathViews. In its simplest form, if `plot` is called with a vector, it automatically plots the data in a graphics window. MathViews automatically scales the plot to match the range of data. If plot is called with two vectors, it uses the first vector for the x-axis and the second vector for the y-axis. The following plot shows this case:

```
t = 1:64
plot(t, sin(2*t/pi))
```

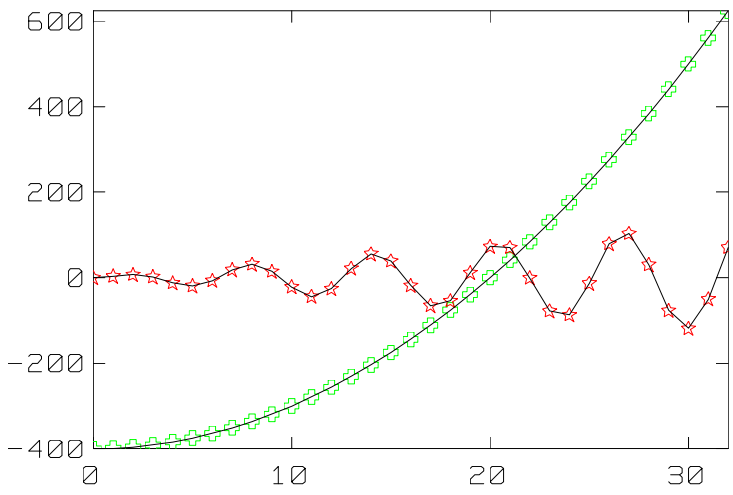


Multiple lines can be plotted on one graph by using multiple arguments to the `plot()` function. You can also specify line style, symbol type and color. The following example plots two lines on the same graph window. The first line is plotted

in green and uses the + symbol and the second line is plotted in red and uses the * symbol. Notice also that we use the `hold` function to hold the plot while we redraw the lines using solid lines with no symbols.

```
t = 0:32
t2 =-400 +  t^2;
t4 = 4*t.*sin(t);

plot(t,t2,'+g', t, t4, '*r')
hold on
plot(t,t2,t,t4)
hold off
```

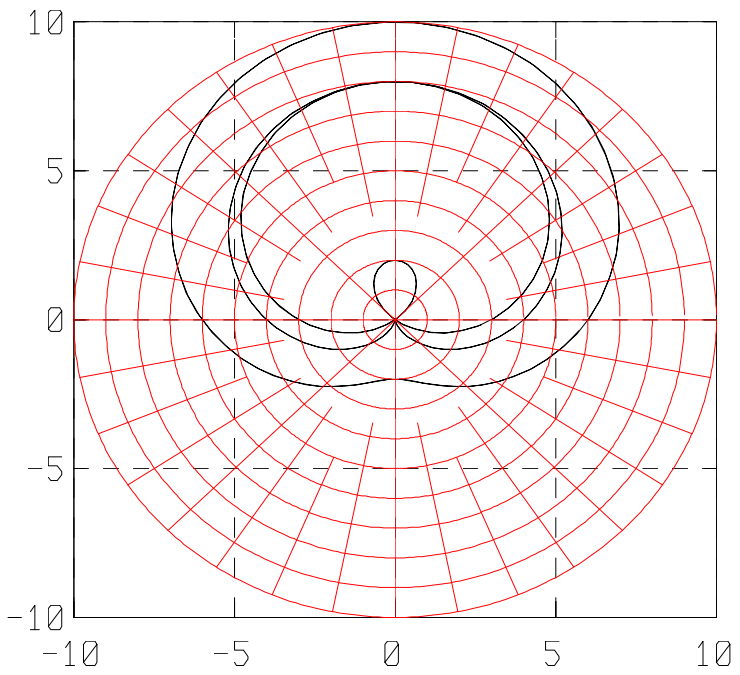


Other MathViews functions can be used to generate x-y plots with different x-y axis scaling. The `loglog()`, `semilogx()`, and `semilogy()` functions allow you to generate various logarithmic scalings.

Creating Polar Plots

The `polar()` function is used to create polar plots in MathViews. This function takes two arguments, the first specifies the angle and the second specifies the length of the vector at the corresponding angles. The following example shows how to create a polar plot. Notice again the use of the `hold` function to allow multiple plots to be place into one graphics window. The `grid` function places a polar grid on the window.

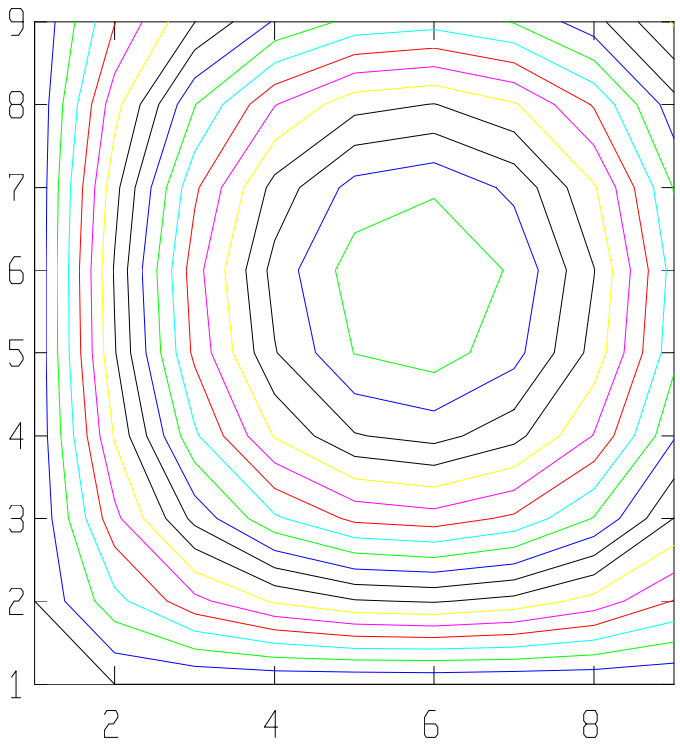
```
pi = acos(-1);
theta=0:.1:4*pi;
r=4+4*sin(theta);
r1=3+5*sin(theta);
r2 = 6+4*sin(theta);
polar(theta,r2);
hold on
polar(theta,r1);
polar(theta,r);
grid
```



Creating Contour Plots

The `contour()` function is used to create contour plots in MathViews. In its simplest form, this function takes a 2-dimensional array of real values and creates a contour with 10 levels. The following example shows this case:

```
n=8;
x=(0:n)/n;
y=x.*(1.2-x);
y /= max(y);
z=y'*y;
contour(z)
```

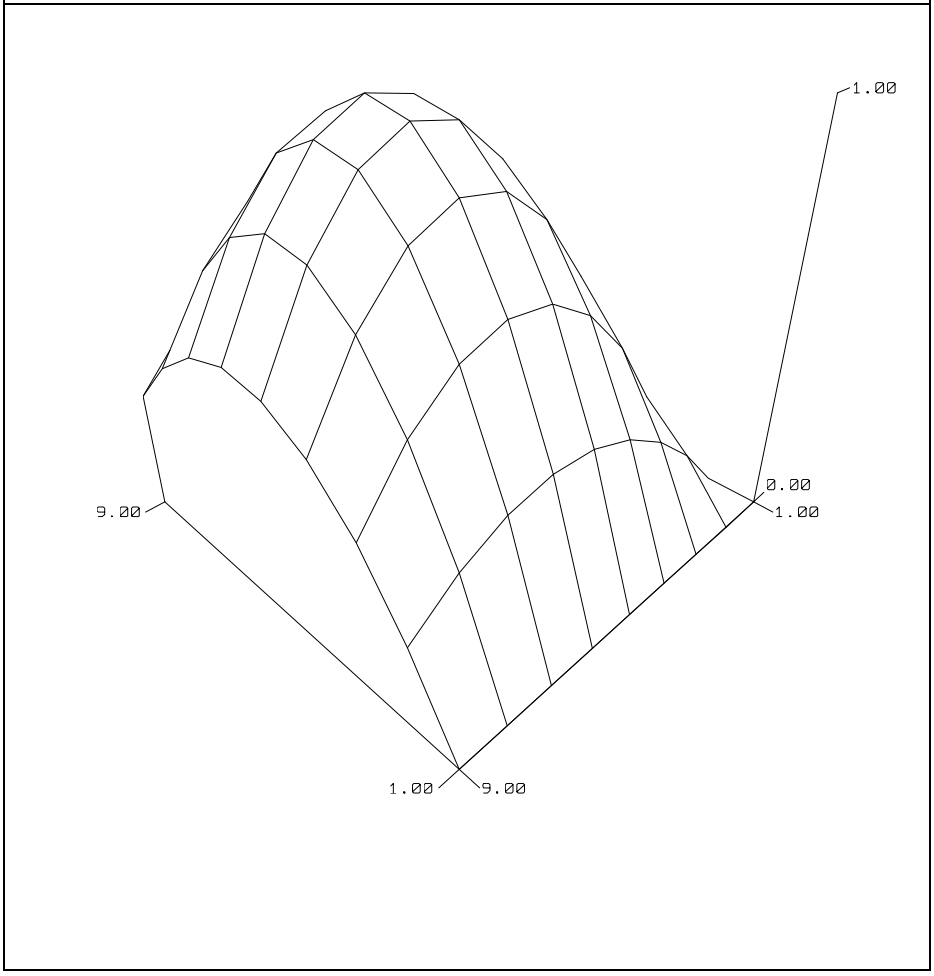


Additional arguments to the `contour` function allow you to specify the number of levels or the specific levels at which you wish to plot contours.

Creating Three-Dimensional Plots

The `mesh()` function is used to create three-dimensional plots in MathViews. In its simplest form, this function takes a two-dimensional array of real values that indicates the height, Z-axis value, above the X-Y plane. The following example illustrates this case:

```
n=8;
x=(0:n)/n;
y=x.*(1.2-x);
y /= max(y);
z=y'*y;
mesh(z)
```



Additional parameters to the mesh function allow you to change the viewing angle and distance.

Relational And Logical Operations

Relational and logical operators are used to test the validity of two expressions based on a specified relationship. They operate on matrices on an element-by-element basis. The results of both relational and logical operations are 0 or 1.

Relational Operators

The set of relational operators in MathViews includes:

Relational Operator	Meaning
==	equal
<=	less than or equal
<	less than
>=	greater than or equal
>	greater than
~=	not equal

A MathViews relation has the form:

{ expr } (relational operator) { expr }

The relational operator is applied to the matrices resulting from the expressions on an element-by-element basis. The resultant matrices must be of equal dimensions. The resultant value is a 1 if the relationship holds; otherwise it is a 0. For example,

```
A =                          B =
     1    2    3    4            9    2   45    4
     5    6    7    8            4    6    8    8
     9   10   11   12            9   20   11   12
```

```
x = (A >= 5)
x =
   0   0   0   0
   1   1   1   1
   1   1   1   1
```

This relationship returns matrix `x` with a one for each element of matrix `A` that has a value greater than or equal to 5.

```
x = A == B
```

```
x =
   0   1   0   1
   0   1   0   1
   1   0   1   1
```

This relationship returns a matrix `x` with a one for each element of matrix `A` that is also in matrix B.

Logical Operators

The set of logical operators in MathViews includes:

~	not
&	and
|	or

A logical comparison has the form:

{ expr } (logical operator) { expr }

The result of logical operators is a 1 if the logical condition holds and a 0 if the logical condition fails. The binary logical operators (`&` and `|`) *must* have operands whose dimensions are equal. The not logical operator (~) is unary. Logical operators function on elements of matrices, usually with *logical* matrices. The following examples illustrate the concept:

```
A =
   1   0   1
   0   1   0
```

```
B =
   1   1   1
   0   1   1
```

A | B	A & B
ans = 1 1 1 0 1 1	ans = 1 0 1 0 1 0

Loops And Conditionals

Like most programming languages, MathViews has statements to control the flow of execution of statements. Looping constructs, such as while and for loops, can be used to repeat a set of statements either until a condition is met or a set number of times. Conditional statements can be used to execute a set of statements based on specified conditions.

Conditional Statements

The **if** statement is used allow conditional execution, based on the evaluation of an expression. An **if** statement must terminate with the keyword **end**. The general form is as follows:

if *expression*

statements

end

or

if *expression*

statements

else[if *expression***]**

statements

end

The expression evaluates to *true* if it has a value, or elements with value, greater or equal to 1; otherwise it is *false*. The **else** and **elseif** conditionals are optional. The condition expression resultant must be a scalar.

The following examples illustrate the use of **if** statements:

```
A =
  1 0 1
  0 1 0
B =
  1 1 1
  0 1 1
C =
  1 1 1
  0 1 1
```

```
if ( all(all(A == B)))
    disp('B equal to A')
else
    disp('B not equal to A')
end
```

```
B not equal to A
```

```
if ( all(all(C == A)))
    disp('C is equal to A')
elseif all(all(C == B))
    disp('C is equal to B')
end
```

```
C is equal to B
```

The MathViews `all()` function is used to compare all elements of the matrix it is passed to see if they are greater than or equal to 1.

Iteration Statements

While Loops.

The **while** construction executes a series of statements in a loop as long as a given condition is true. The general form is as follows:

`while` *condition*

statement

`end`

The condition is usually a scalar result of a relational operation. As with the **if** statement, you could use the `all()` function to obtain a scalar condition from a condition matrix. The following example illustrates the use of a **while** loop:

```
x = 0
while( x < 5 )
    disp(x++)
end
```

This example displays the numbers from 0 to 4. The **while** loop stops executing when the value of x is no longer less than five.

For Loops.

The **for** loop repeats a group of instructions for a specified number of times. The general form is as follows:

`for` `n =` *expression*

statements

`end`

The **for** loop expression in MathViews is a matrix. The statements in the **for** loop get executed once for each column of the expression matrix. The **for** loop variable, n, is assigned the value of the current column of the expression matrix. The expression matrix is typically a vector created with the colon operator, i.e. `for n = 1:5`, would cause the statements to be executed five times, with n taking on the integer values from 1 to 5.

The following example shows the use of nested **for** loops and the `size()` function. The `size()` function is used to determine the number of elements in a matrix. The

matrix dimensions are used to specify the number of times to execute the statements in the **for** loop.

```
A =
  1  2  3  4
  1  2  3  4
  1  2  3  4
  1  2  3  4
```

```
[m, n] = size(A);

for i = 1:m
    for j = 1:n
        if ( i ~= j)
            A(i,j) = A(j,i);
        end
    end
end
A
```

```
A
ans =
  1  1  1  1
  1  2  2  2
  1  2  3  3
  1  2  3  4
```

Break Statement

The **break** statement is used to terminate from within an iteration statement. **Break** can be used in the body of **while** or **for** looping constructs. When used within nested looping constructs, **break** transfers control to the loop that is nested one level above the loop in which it occurs. For example,

```
A =
   1   1   1   1   1   1   1   1   1
   1   2   2   2   2   2   2   2   2
   1   2   3   3   3   3   3   3   3
   1   2   3   4   4   4   4   4   4
   1   2   3   4   5   5   5   5   5
   1   2   3   4   5   6   6   6   6
   1   2   3   4   5   6   7   7   7
   1   2   3   4   5   6   7   8   8
   1   2   3   4   5   6   7   8   9
```

```
for i = 1:m
    for j = 1:n
        A(i,j) = i * j;
        if ( (i == 2) & (j ==2))
            break
        elseif ( (i == 4) & (j == 4))
            break
        elseif( (i==6) & (j==6))
            break
        end
    end
end
A
```

```
A

ans =

    1    2    3    4    5    6    7    8    9
    2    4    2    2    2    2    2    2    2
    3    6    9   12   15   18   21   24   27
    4    8   12   16    4    4    4    4    4
    5   10   15   20   25   30   35   40   45
    6   12   18   24   30   36    6    6    6
    7   14   21   28   35   42   49   56   63
    8   16   24   32   40   48   56   64   72
    9   18   27   36   45   54   63   72   81
```

Creating Programs and Functions

MathViews can be used in an interactive mode to provide solutions to your problems. However, MathViews can also be programmed to control repetitive tasks or to extend the MathViews language itself. MathViews, as was discussed in the previous section, provides statements that control the flow of execution of your statements.

MathViews programs are written in units called M-files. M-files are stored on the disk in files with extensions of "`.M`". There are two types of M-files:

> **Script** M-files contain a set of statements that are executed when the script file is invoked.
>
> **Function** M-files are similar to script files. In addition to containing a set of statements that are executed, they can also be passed parameters and return results. Only one function can be specified per M-file.

MathViews also supports another type of file that is extremely useful for maintaining your functions when developing a large project. L-files serve as modules that can contain multiple functions. This overcomes the limitation of only having one function per file as in M-files.

Scripts

Script M-files contain sequences of statements. The script M-file is invoked by calling the name of the M-file. For example, suppose the following script is stored in the file `maxval.m`. This script file finds and prints the maximum value contained in the matrix A. Executing the script by entering `maxval` causes the maximum value in matrix A to be printed. The matrix A must exist prior to executing this M-file.

```
%
% Script M-file to find the maximum value
% in the matrix A.
%
[m,n] = size(A);
max = -1e300;
for x = 1:m
   for y = 1:n
      if A(x,y) > max
         max = A(x,y);
      end
   end
end
max
```

Script files are executed as if you had entered them in interactive mode. Any variables created in the script file are left in the MathViews environment upon completion of the script.

Functions

Functions defined in a function M-file can be used to extend the MathViews language. Function files are similar to script files in that they both contain a set of statements to be executed, however, there are a few important differences. The first line of a function file must contain the word `function`. Functions can also be passed arguments and can return values. Also, unlike script M-files, variables created in a function M-file only exist during the execution of the function. They are not left in the MathViews environment upon completion of the function. Only one function can be defined per M-file.

The following example performs the same operations as the script file discussed above, that is, finding the maximum value of a matrix. However, the function we create will be passed the matrix for which we wish to find the maximum and it will return the maximum value as a variable.

```
function max = maxval(a)
%
% Function M-file to find the maximum value
% in the matrix a.
%
[m,n] = size(a);
max = -1e300;
for x = 1:a
   for y = 1:a
      if a(x,y) > max
         max = a(x,y);
      end
   end
end
```

The first line of the function M-file defines the input and output parameters for the function. There is one input variable, `a`, and one output variable, `max`. If we store this function in an M-file named `maxval.m`, we can invoke it using the form:

```
maximum = maxval(A)
```

In this case, we pass the matrix A as an argument to maxval. The maxval function M-file operates on the matrix A and returns the maximum value.

Modules

MathViews library module files overcome the limitations of only allowing a single function per M-file. A library module file, L-file, contains only functions in the form discussed above for function M-files. Functions can be grouped according to the specific application they were designed for. This simplifies management of the functions you create by placing all related functions into one location.

MathViews lets you easily load a function library file into memory by executing the `_lcompile` statement.

4

Function Listing

This section provides a list of most of the functions that are available in MathViews.

Function	Description
abs	absolute value function
acos	arccosine function
acosh	hyperbolic arc cosine
all	logical test for arrays
angle	compute phase
any	logical test for arrays
asin	arcsine function
asinh	hyperbolic arc sine
atan	arctangent function
atan2	arc tangent function
atanh	hyperbolic arc tangent
axis	scaling axis on plots
balance	balance a matrix to improve its conditioning
blackman	creates a Blackman function window
casesen	toggle case sensitivity
ceil	he ceiling function
chdir	file manipulation command to change directory
chol	Cholesky matrix factorization
cla	compatibility function (clears output window)
clc	clear output window

Function	Description
clear	removes items from MathViews memory
cntrlb	contour plot with level readings
compan	the companion matrix
computer	returns 'WIN' (a compatibility function)
cond	the condition number of a square matrix

conj	complex conjugate function
contour	graphics contour plot
conv	convolution
cos	cosine function
cosh	hyperbolic cosine function
cosw	sinusoidal waveform
cov	covariance matrices
cumprod	cumulative product of an array
cumsum	cumulative sum of an array
delete	file manipulation command to delete files
det	matrix determinant
dft	Discrete Fourier Transform
diag	diagonal elements
diary	record the current input session on disk file
diff	compute the difference to approximate derivatives
dir	list the files and subdirectories in a directory
disp	display a text string or a matrix
echo	print M-files statements during execution
edit	invokes the MathPad editor
eig	eigenvalues/eigenvectors of a real symmetric matrix
error	echos a message to output window
etime	computes the elapsed time
eval	text macro utility
exist	test existence of variable

Function	Description
exit, quit	quit MathViews session
exp	exponential function
expm	matrix exponential
expr_sel	ternary conditional selection
eye	get identity matrix
fac	factorial
feval	function evaluation
fft	Fast Fourier Transform 1-D and 2-D
filter	filter data
find	find nonzero elements
fix	rounding function
flip	mirror image of an array
fliplr	flip matrix left/right
flipud	flip matrix up/down
floor	rounding function
format	number display format
fprintf	file output conversion

`grid`	overlap plot with a grid
`gwclr,clg`	clear MathViews graphics window
`gwinit`	initialize MathViews graphics windows environment
`gwsel`	select MathViews graphics window
`hamming`	creates a Hamming function window
`hanning`	creates a Hanning function window
`help`	help command
`hilb`	Hilbert matrix
`hold`	hold plot on graphical screen
`home`	place text cursor at top of interactive window
`ifft`	Inverse Fast Fourier Transform
`imag`	extract imaginary component of a complex number

Function	Description
`input`	prompt for input
`int2str`	convert integer number to string format
`inv`	matrix inversion
`invhilb`	inverse Hilbert matrix
`iscmplx`	test for complex elements
`iscol`	test for column vector
`isempty`	test for empty matrix
`isreal`	test for real (scalar) number
`isrow`	test for row vector
`isscalar`	test for scalar
`isstr`	test for string
`isvector`	test for vector
`keyboard`	pass control to the interactive window from an M-file
`length`	test for length
`load`	load variables from disk
`log`	natural logarithm function
`log2`	base-2 logarithm function
`log10`	base-10 logarithm function
`loglog`	plot using log-log scales
`ltifr`	linear time invariant frequency response
`ltitr`	linear time invariant time response
`max`	find largest element of array(s)
`mean`	compute the mean
`median`	compute the median
`mesh`	create 3-D hiddeln line removal graph
`meshdom`	create mesh domain
`meta`	create a Windows metafile
`min`	find smallest element of array(s)
`norm`	matrix norm

Function	Description
num2str	convert number to a string
ones	get an all 1's vector or matrix
pack	compact memory
pause	pause until key pressed
pi	returns the value pi as a function
pinv	pseudo-inverse
play	display a Windows metafile (wmf)
plot	line plots
polar	generate polar plot
poly	characteristic polynomial
polyfit	polynomial fitting
polyval	polynomial evaluation
print, prtsc	print the current graphics windows
prod	compute the product
rand	random number generator
rank	rank of a matrix
rcond	compute condition number of a matrix
real	extract the real component of a complex number
rem	compute the remainder
reshape	reshape an array
roots	polynomial roots
round	rounding function
save	save variable(s) to disk file
semilogx	plot using semi-log scales in x
semilogy	plot using semi-log scales in y
setstr	string handling function
sign	sign function
sin	sine function
sinh	hyperbolic sine function

Function	Description
sinw	sinusoidal waveform
size	dimension of variable
sort	sort an array
sprintf	string conversion function
sqrt	square root function
startup	start up script file
strcmp	string comparison function, extended to string arrays.
subplot	Create subplot windows within the graphics window
sum	sum of an array

svd	singular value decomposition
tan	tangent function
tanh	hyperbolic tangent function
text	plot text onto graphics window
title	insert plot title
trace	trace of a matrix
tril	lower triangular matrix
triu	upper triangular matrix
type	display the contents of a file
wbartlet	creates a triangle window
who, whos	list variables
wmerit	figures of merit for any window function
wrect	creates a rectangle/Dirichlet window
wsina	creates a family of sine windows
x01	returns an n-point vector linearly spaced from 0 to 1
x101	returns an n-point vector spaced linearly from -1 to 1
xlabel	label x is
ylabel	label y is
zlabel	label z is
zeros	0 matrix

Function	Description
_compile	compile M-file
_lcompile	library compilation
_list	list dependence variables
_loadlib	load precompiled library
_savelib	saves the current compile library
_ver	get version number