Introduction

MATLAB

MATLAB = MATrix LABoratory

Interactive system.

Basic data element is an array that does not require dimensioning.

"Efficient" computation of matrix and vector formulations (in terms of writing code - it is interpreted so looses efficiency there) relative to scalar non-interactive language such as C or Fortran.

The 5 parts

-

1 - Desktop Tools and Development

2 - Mathematical Functions

3 - The Language

4 - Graphics

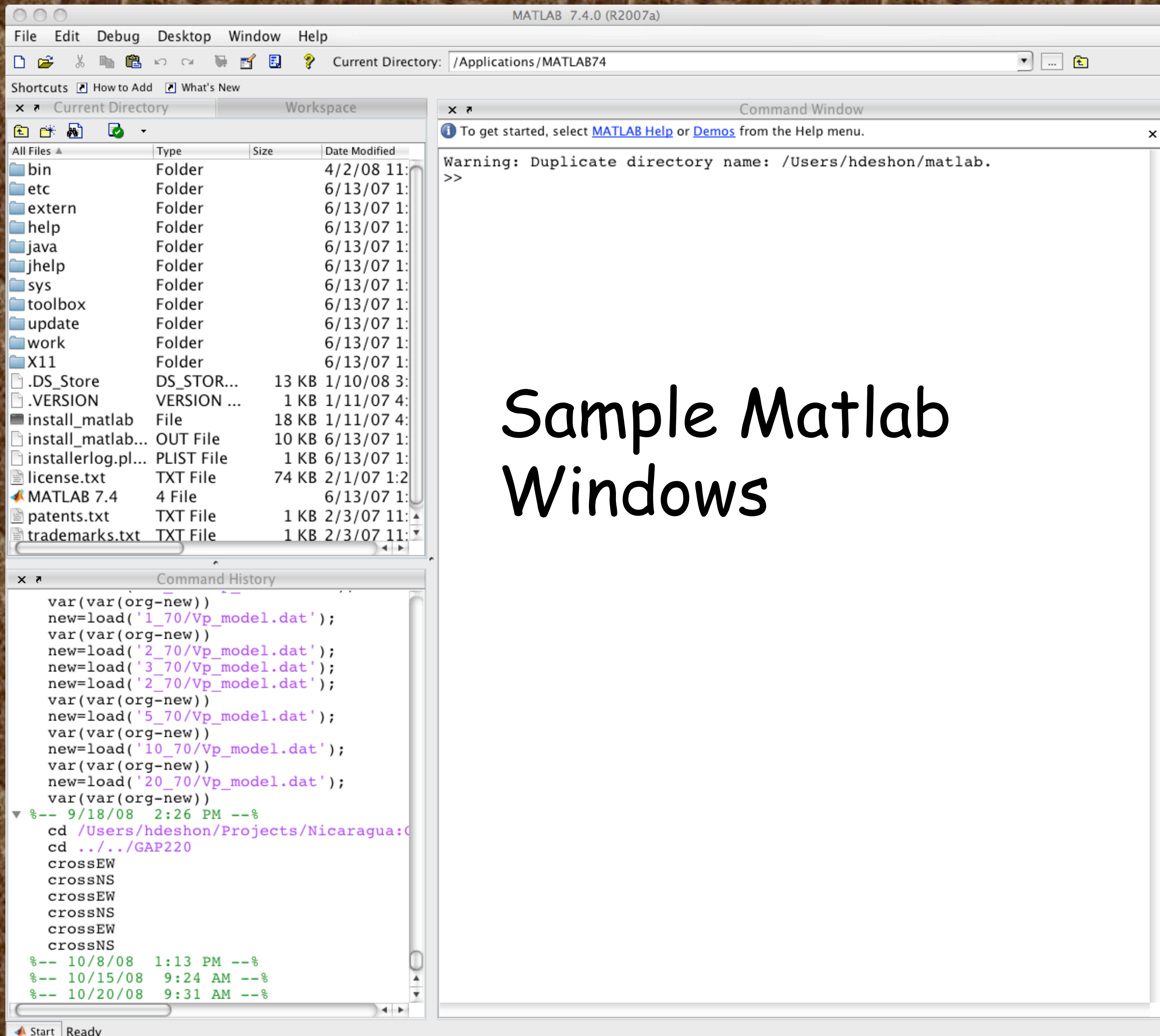
5 - External Interfaces

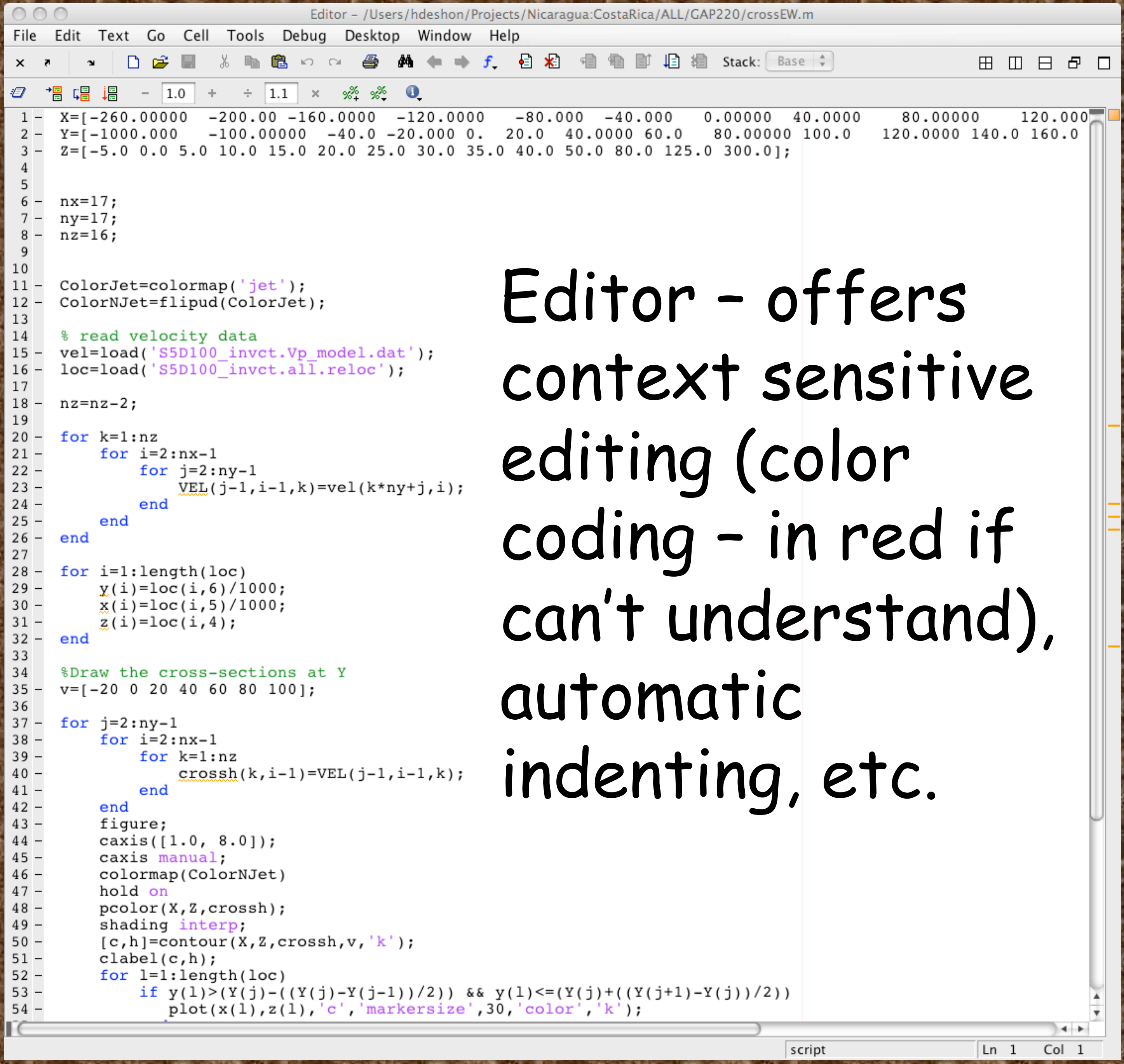
Desktop Tools & Development

Graphical user interfaces:

- MATLAB desktop and Command Window
- Command history window
- Editor and debugger
- A code analyzer and other reports
- Browsers for viewing help, the workspace, files, and the search path.

Sample Matlab Windows





Editor - offers context sensitive editing (color coding - in red if can't understand), automatic indenting, etc.

Mathematical Functions

Large collection of computational algorithms including but not limited to:

Elementary functions, like sum, sine, cosine

Complex arithmetic

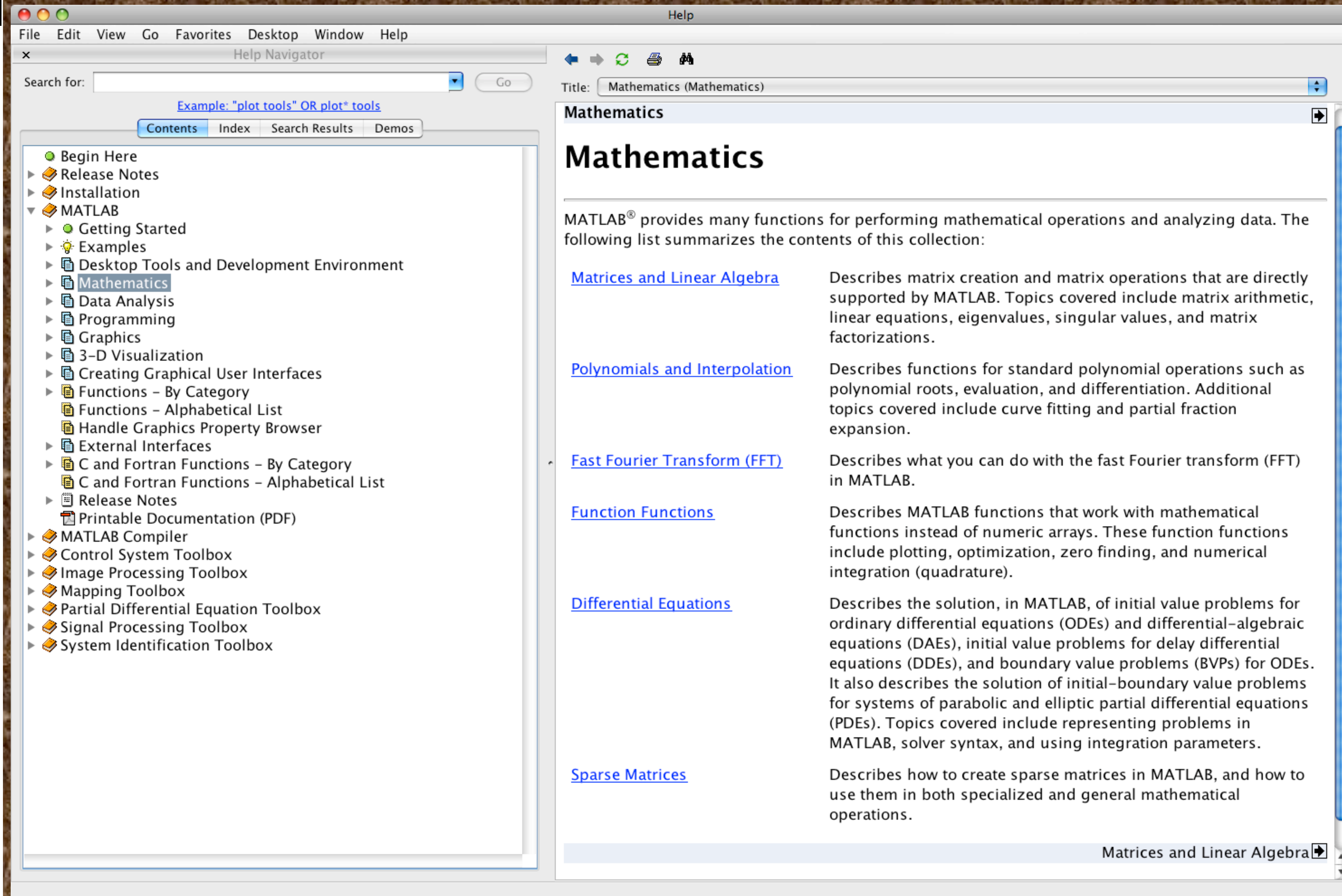
Matrix math – inverse, eigenvalues/vectors, etc.

Fast Fourier transforms

Bessel functions

etc.

Interactive help and documentation.



Biggest resource

GOOGLE/WEB

There are trillions of matlab tutorials, program exchanges, discussions, “toolboxes”, etc., on the web.

The Language
High-level matrix/array language

Includes control flow statements, functions, data structures, input/output, and object-oriented programming features

It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create large and complex application programs.

Graphics:

Two-dimensional and three-dimensional data visualization.

Image processing.

Animation.

Presentation graphics.

Graphics:

It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete GUIs for your own applications.

External Interfaces

Library that allows you to write C and Fortran programs that interact with MATLAB.

It includes facilities for calling routines from MATLAB (dynamic linking), for calling MATLAB as a computational engine, and for reading and writing MAT-files.

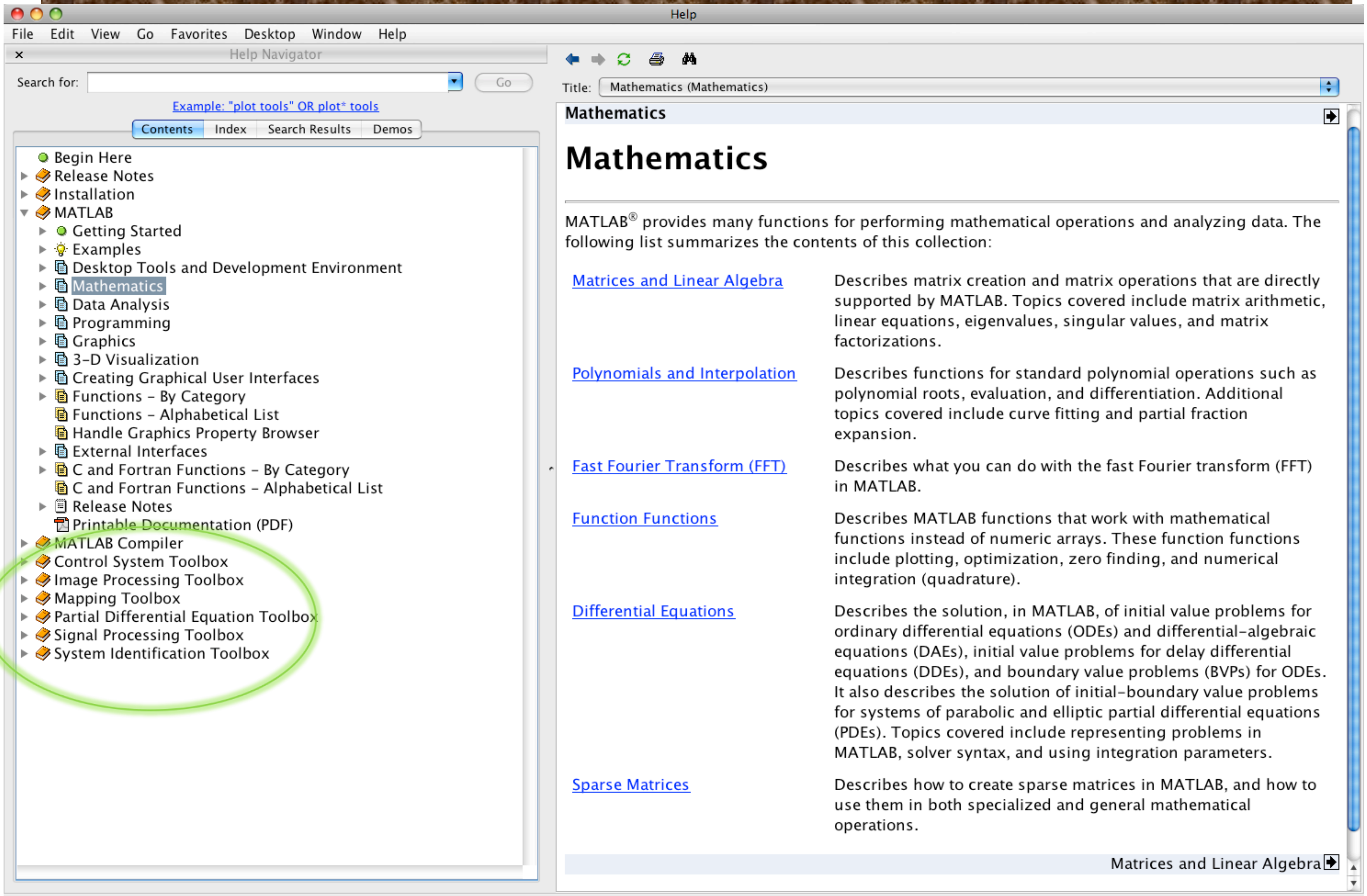
Toolboxes

Add-on application-specific solutions

Comprehensive collections of MATLAB functions (M-files) to solve particular classes of problems.

Examples include:

- Signal processing
- Image processing
- Partial differential equations
- Mapping
- Statistics


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Search for:
Go
Example: "plot tools" OR plot* tools
Contents Index Search Results Demos
Begin Here
Release Notes
Installation
MATLAB
Getting Started
Examples
Desktop Tools and Development Environment
Mathematics
Data Analysis
Programming
Graphics
3-D Visualization
Creating Graphical User Interfaces
Functions – By Category
Functions – Alphabetical List
Handle Graphics Property Browser
External Interfaces
C and Fortran Functions – By Category
C and Fortran Functions – Alphabetical List
Release Notes
Printable Documentation (PDF)
MATLAB Compiler
Control System Toolbox
Image Processing Toolbox
Mapping Toolbox
Partial Differential Equation Toolbox
Signal Processing Toolbox
System Identification Toolbox
Title: Mathematics (Mathematics)
Mathematics
Mathematics
MATLAB® provides many functions for performing mathematical operations and analyzing data. The following list summarizes the contents of this collection:
Matrices and Linear Algebra
Describes matrix creation and matrix operations that are directly supported by MATLAB. Topics covered include matrix arithmetic, linear equations, eigenvalues, singular values, and matrix factorizations.
Polynomials and Interpolation
Describes functions for standard polynomial operations such as polynomial roots, evaluation, and differentiation. Additional topics covered include curve fitting and partial fraction expansion.
Fast Fourier Transform (FFT)
Describes what you can do with the fast Fourier transform (FFT) in MATLAB.
Function Functions
Describes MATLAB functions that work with mathematical functions instead of numeric arrays. These function functions include plotting, optimization, zero finding, and numerical integration (quadrature).
Differential Equations
Describes the solution, in MATLAB, of initial value problems for ordinary differential equations (ODEs) and differential-algebraic equations (DAEs), initial value problems for delay differential equations (DDEs), and boundary value problems (BVPs) for ODEs. It also describes the solution of initial-boundary value problems for systems of parabolic and elliptic partial differential equations (PDEs). Topics covered include representing problems in MATLAB, solver syntax, and using integration parameters.
Sparse Matrices
Describes how to create sparse matrices in MATLAB, and how to use them in both specialized and general mathematical operations.
Matrices and Linear Algebra

Starting MATLAB

Runs on SUNS, MACS, PC's - same interface.

From CERI unix machines, just type

```
%matlab
```

On a PC/Mac, double-click the Matlab icon.

Starting MATLAB

In an X11 window (assuming it is in your path), type

```
%matlab
```

Useful trick from remote machines

```
%matlab –nojvm
```

or

```
%matlab -nodesktop -nosplash
```

turns off the graphical interface - which is SLOW and buggy over net.

the Matrix

A <u>matrix</u> is a rectangular array of numbers

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

<u>Vectors</u> are matrices with only one row or column

16	3	2	13

<u>Scalars</u> can be thought of as 1-by-1 matrices

16

Matlab basically thinks of everything as a matrix.

Handles math operations on

Scalars
Vectors
2-D matricies

With ease

Gets ugly with higher dimension matrices - as there are no mathematical rules to follow.

Entering Matrices

- Enter an explicit list of elements.

- Load from external data files.

- Generate using built-in functions

-Create with your own functions in M-files

(matlab's name for a file containing a matlab program. Same as shell script, sac macro, batch file, commnad file, etc. but for matlab.)

Entering a matrix from the command line:

Separate the elements (columns) of a row with blanks or commas.

Use a semicolon, ";" , to indicate the end of each row.

Surround the entire list of elements with square brackets, [ ].

```
>> A44 = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A44 =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> A14 = [16 3 2 13]
A14 =
    16     3     2    13
>> A41 = [16; 5; 9; 4]
A41 =
    16
     5
     9
     4
>> whos
  Name          Size                 Bytes  Class     Attributes
  A14           1x4                     32  double
  A41           4x1                     32  double
  A44           4x4                    128  double
>>
```

whos - reports what is in memory

Matrices indexed the same as math (row, column)

Suppressing Output

If you simply type a statement and press Return or Enter, MATLAB automatically displays the results on screen.

If you end the line with a semicolon, MATLAB performs the computation but does not display any output. This is particularly useful when you generate large matrices.

Matlab normally prints out results – to stop printout, end line with semi-colon ";" (this is general rule).

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

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

The load function

reads binary files containing matrices (generated by earlier MATLAB sessions), or text files containing numeric data.

The text file should be organized as a rectangular table of numbers, separated by blanks, with one row per line, and an equal number of elements in each row.

```
%cat magik.dat
16.0    3.0     2.0     13.0
5.0     10.0    11.0    8.0
9.0     6.0     7.0     12.0
4.0     15.0    14.0    1.0
>>A=load('magik.dat') #places matrix in variable A
>> load magik.dat   #places matrix in variable magik
```

Matlab is particularly difficult with data files that do not fit this format.

Matlab is also particularly difficult with processing character data.

Generate matrices using built-in functions.

Complicated way of saying "run commands" and send output to new matrices.

Also does matrix operations (e.g. - transpose).

```
>>magik'  #transpose matrix magik
ans =
    16    5   9   4
    3  10     6   15
    2     11      7   14
    13    8   12      1
```

M-Files

Text files with MATLAB code (instructions). Use MATLAB Editor (or any text editor) to create files containing the same statements you would type at the MATLAB command line.

Save the file with a name that ends in .m

```
%vim magik.m
i
A = [ 16.0 3.0 2.0 13.0
5.0 10.0 11.0 8.0
9.0 6.0 7.0 12.0
4.0 15.0 14.0 1.0 ];
(esc)wq
```

in matlab

```
>>magik    #places matrix in A
```

Entering long statements

If a statement does not fit on one line, use an ellipsis (three periods), "...", followed by "Carriage Return" or "Enter" to indicate that the statement continues on the next line.

```
>>s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
        - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Subscripts

Matrices consists of rows and columns. The element in row i and column j of A is denoted by $A(i,j)$ (same as math).

Example: $A(4,2)=15.0$

$$
\begin{array}{c|cccc}
i \backslash j & 1 & 2 & 3 & 4 \\
\hline
1 & 16.0 & 3.0 & 2.0 & 13.0 \\
2 & 5.0 & 10.0 & 11.0 & 8.0 \\
3 & 9.0 & 6.0 & 7.0 & 12.0 \\
4 & 4.0 & \boxed{15.0} & 14.0 & 1.0
\end{array}
$$

4^{th} row, 2^{nd} column.

If you store a value in an element outside of the current size of a matrix, the size increases to accommodate the newcomer:

```
>>A = [ 16.0 3.0 2.0 13.0
5.0 10.0 11.0 8.0
9.0 6.0 7.0 12.0
4.0 15.0 14.0 1.0 ];
>>X = A;
>>X(4,5) = 17
X =
16   3    2    13   0
5    10   11   8    0
9    6    7    12   0
4    15   14   1    17
>>
```

You can also access the element of a matrix by referring to it as a single number.

This is because computer memory is addressed linearly – a single line of bytes (or words).

There are therefore (at least) two ways to organize a two dimensional array in memory – by row or by column (and both are/have been used of course).

MATLAB (and Fortran) store the elements by columns (called column major order).

```
>>A = [ 16.0 3.0 2.0 13.0
5.0 10.0 11.0 8.0
9.0 6.0 7.0 12.0
4.0 15.0 14.0 1.0 ]
A=
16  3   2   13
5   10      11      8
9   6   7   12
4  15   14      1
```

The elements are stored in memory by column.

```
16, 5, 9, 4, 3, 10, 6, 15, 2, 11, 7, 14, 13, 8, 12, 1.
(1)(2)(3)(4)(5) (6)(7) (8)(9)(10)(11)(12)(13)(14)(15)(16)
```

So A(11)=7.

A(i,j)

i varies most rapidly

j varies least rapidly

For 4x4 matrix

(1,1), (2,1), (3,1), (4,1), (1,2), (2,2)...(3,4), (4,4,)

(1) (2) (3) (4) (5) (6) (15) (16)

This may be important when reading and writing very large matrices – one wants the data file to have the same storage order as memory to minimize time lost to page faulting.

When you go to 3 dimensions, order of subscript variation is maintained (1st to last)

$A(i,j,k)$

i varies most rapidly

j varies next most rapidly

k varies least rapidly

For 3x2x2 matrix

(1,1,1), (2,1,1), (3,1,1),
(1,2,1), (2,2,1), (3,2,1),
(1,1,2), (2,1,2), (3,1,2),
(1,2,2), (2,2,2,), (3,2,2),
…

C uses row major order (stores by row).

If mixing Matlab and Fortran there is no problem as both use column major order.

If mixing Matlab or Fortran and C - one has to take the array storage order into account.

(one also has to deal with how information is passed

- by reference [the address of the information in memory - Fortran]

- or value [a copy of the information - C].)

The Colon Operator

The colon, ":", is one of the most important MATLAB operators

It can be used to

- Create a list of numbers
- Collapse trailing dimensions (right- or left-hand side)
- Create a column vector (right-hand side behavior related to reshape)
- Retain an array shape during assignment (left-hand side behavior)
- Work with all entries in specified dimensions

Creating a List of Numbers

You can use the “:” operator to create a vector of evenly-spaced numbers.

Here are the integers from -3 to 3.

```
>>list1 = -3:3
list1 =
    -3    -2    -1     0     1     2     3
```

Creating a List of Numbers

Here are the first few odd positive integers.

```
>>list2 = 1:2:10
list2 =
     1     3     5     7     9
```

Negative increment

```
>>100:-7:51
100 93 86 79     72     65     58     51
```

syntax for this use of color operator -
start:[increment:]end
(default increment = 1)

Creating a List of Numbers

Here's how to divide the interval between 0 and pi (Matlab knows about pi) into equally spaced samples.

```
>>nsamp = 5;
>>sliceOfPi = (0:1/(nsamp-1):1)*pi
sliceOfPi =
         0    0.7854    1.5708    2.3562    3.1416
```

(Note – can also define single dimension row matrix with colon operator by ()'s or no delimiters rather than []'s. Does not work when try to use ";" for another row or by specifying elements.)

```
a=(1:3)
a =
     1     2     3
a=1:3
a =
     1     2     3
```

Aside – for languages that (unlike Matlab) don't have PI predefined, how can one get the "best" representation of pi (most precise on that computer)?

Collapsing Trailing Dimensions

Suppose have the following 4-dimensional array.

```
>> b=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
b =
   1   2   3   4   5   6   7   8   9  1    11   12   13   14   15   16
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
>>
```

1-d vector

2-d matrix

3-d stack of 2-d matrices

>3-d something hard to visualize - but fine mathematically (4-d is 2-d matrix with each element itself a matrix)

```
>> x=[1 2 3]
x =
     1     2     3
>> sum(x)
ans =
     6
>> xt=[1;2;3]
xt =
     1
     2
     3
>> sum(x)
ans =
     6
>> y=[1 2; 4 4]
y =
     1     2
     4     4
>> sum(y)
ans =
     5     6
>> sum(sum(y))
ans =
    11
>>
```

Matlab "sum" command.

Sums elements in vector (row or column) - result is a scalar.

For a matrix, sums elements by column (the order stored in memory) - result is a vector of the column sums.

To sum whole matrix, call twice (once to sum columns, then second time to sum resulting vector) - result is a scalar.

```
>> b=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
b =
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
>> sum(b4d(:,:,1,1))
ans =
     3     7
>> sum(b4d(:,:,2,1))
ans =
    11    15
>>
```

Summing parts of the 4-d matrix.

Same as summing on the 2-d matrices.

```
b11=
     1     3
     2     4
b21=
     5     7
     6     8
```

```
b =
   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
>> b4d(1,1,:)
ans(:,:,1) =
     1
ans(:,:,2) =
     5
ans(:,:,3) =
     9
ans(:,:,4) =
    13
>>
```

Colon gives us

- Full range of index

- At end of list it "compresses" all the remaining indices into a single index (indexed as in memory – by single subscript - linearly). This is called "collapsing" trailing dimensions.

```
>> b4d(1,1,:)
ans(:,:,1) =
     1
ans(:,:,2) =
     5
ans(:,:,3) =
     9
ans(:,:,4) =
    13
>> sum(b4d(1,1,:))
ans =
    28
>>
```

Sum - adds them.

```
b =
   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
>> b4d(:,1,1)
ans =
     1
     2
>> b4d(1,:,1,1)
ans =
     1     3
>>
```

Works differently from front or in middle.

```
>>b4d(1,1,1,:)
ans(:,:,1,1) =
     1
ans(:,:,1,2) =
     9
>> b4d(1,1,:)
ans(:,:,1) =
     1
ans(:,:,2) =
     5
ans(:,:,3) =
     9
ans(:,:,4) =
    13
>> b4d(1,:,1)
ans =
     1     3
```

Works differently from front or in middle.

```
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
```

```
>> b4d(1,1,:,:)
ans(:,:,1,1) =
     1
ans(:,:,2,1) =
     5
ans(:,:,1,2) =
     9
ans(:,:,2,2) =
    13
>> b4d(1,1,:)
ans(:,:,1) =
     1
ans(:,:,2) =
     5
ans(:,:,3) =
     9
ans(:,:,4) =
    13
>>
```

Are equivalent

```
>> b4d=reshape(b,2,2,2,2)
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
```

```
>> b4d(1,:,:,1)
ans(:,:,1) =
     1     3
ans(:,:,2) =
     5     7
>> b4d(:,1,1,:)
ans(:,:,1,1) =
     1
     2
ans(:,:,1,2) =
     9
    10
>> b4d(1,:,1,:)
ans(:,:,1,1) =
     1     3
ans(:,:,1,2) =
     9    11
>>b4d(:,1,:,1)
ans(:,:,1) =
     1
     2
ans(:,:,2) =
     5
     6
>>
```

Get 4 elements back on each reference with two colons. May be 1 row or column vector, or two row or column vectors.

```
>> a=[1 2 3 4]
a =
     1     2     3     4
>> at=a(:)
at =
     1
     2
     3
     4
>> a22=[1 2; 3 4]
a22 =
     1     2
     3     4
>> a22c=a22(:)
a22c =
     1
     3
     2
     4
>>
```

Creating a column vector from another vector or matrix. (note first example would usually be done using transpose operator at=a')

Retaining Array Shape During Assignment – color operator is on left side = "pours" value into elements defined on lhs.

```
>> b4d
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    13    15
    14    16
```

```
>> b4d(:,:,2,2)=20
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    10    12
b4d(:,:,2,2) =
    20    20
    20    20
>>
```

```
>> b4d(2,:,:,2)=21
b4d(:,:,1,1) =
     1     3
     2     4
b4d(:,:,2,1) =
     5     7
     6     8
b4d(:,:,1,2) =
     9    11
    21    21
b4d(:,:,2,2) =
    20    20
    21    21
>>
```

Working with All the Entries in Specified Dimensions

To manipulate values in some specific dimensions, use the ":" operator to specify the dimensions.

A ":" by itself indicates all elements of that index position (usually rows or columns)

```
>>a(:,1)
```

Means "all rows, in column 1"

Refers to range of values for indices (portions) of a matrix

```
>>k=2;
>>a(1:k,1)
```

'rows 1 through 2, and column 1'

Same as

```
>>a(1:2,1)
```

Can be pretty tricky. For example, suppose I want to perform a left shift on the values in the second dimension of my 3-D array. Let me first create an array for illustration.

```
a3 = zeros(2,3,2);
a3(:) = 1:numel(a3)
a3(:,:,1) =
     1     3     5
     2     4     6
a3(:,:,2) =
     7     9    11
     8    10    12
```

```
a3 = zeros(2,3,2);
a3(:) = 1:numel(a3)
a3(:,:,1) =
     1     3     5
     2     4     6
a3(:,:,2) =
     7     9    11
     8    10    12
```

Now shift columns all over to the left, and have the left-most one "wrap" to become the right most column. Columns are dimension 2. Here's a way (there are others) to do it.

```
a3r1 = a3(:,[2:size(a3,2) 1],:)
a3r1(:,:,1) =
     3     5     1
     4     6     2
a3r1(:,:,2) =
     9    11     7
    10    12     8
```

For all rows, put columns 2 to end (get from 2nd element of size - the middle dimension), then column 1, for all "planes" (2-d matrices in 3rd dimension).

```
a3r1 = a3(:,[2:size(a3,2) 1],:)
a3r1(:,:,1) =
     3     5     1
     4     6     2
a3r1(:,:,2) =
     9    11     7
    10    12     8
```

Variables

MATLAB does not require any type declarations

(actually all variables are double precision floating point - you can declare them to be other things if needed - however many/most Matlab routines [such at FFT, filtering, etc.] will not work with anything other than double precision floating point data)

or dimension statements.

Variables

When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage.

If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage.

MATLAB is case sensitive. ("A" is not the same as "a")

Concatenation

You can concatenate using the square brackets, [] (same as making a matrix, but using other matrices as the elements)

```
>>B = [A      A+32;   A+48             A+16]
B =
16      3   2   13      48      35      34      45
5   10      11      8   37      42      43      40
9   6   7   12      41      38      39      44
4   15      14      1   36      47      46      33
64      51      50      61      32      19      18      29
53      58      59      56      21      26      27      24
57      54      55      60      25      22      23      28
52      63      62      49      20      31      30      17
```

Deleting rows and columns

You can also use [] to remove rows, columns, or elements (again – variation on theme of assigning elements in a matrix – have a syntax rule and read it like a lawyer for all possible interpretations and implications.)

e.g. Remove the second column

```
>>X=A;
>>X(:,2) = [];
```

Create vector from X; removes every 2nd element from 2 to 10

```
>>X(2:2:10) = []
X =
16      9   2   7   13      12      1
```

Stuff you will need for homework:

FOR loop – matlab syntax

```
for cnt=1:2
    Stuff
end
```

To plot – use plot command.

To find out how to use the plot command, use help

```
help plot
```