MATLAB: AN INTRODUCTION

Rajen K. Sinha
Department of Mathematics

HISTORY

- Created originally as a set of FORTRAN subroutines in the 70’s by Cleve Moler, a scientist in Mathworks Inc.
- Rewritten in C with extended capabilities in the 80’s (including graphics).
- Distributed by Mathworks, Inc. (www.mathworks.com)

ABOUT MATLAB

- Interactive Software used in various areas of engineering and scientific applications.
- Not a computer language though it does most of the work of a computer language.
- Easy to learn.
- Does not require the in depth knowledge of computer programming like compiling and linking. (Can be looked upon as an advantage and also as a disadvantage).
- Plotting
- Interactive mode may reduce computational speed.

Basic Features

COMMAND WINDOW

- **When we run MATLAB by double clicking on MATLAB on the desktop, it creates one or more windows on the computer monitor. Out of these, COMMAND WINDOW is the place where we interact with MATLAB.**
- **The prompt >> is displayed in the command window and when the command window is active, a blinking cursor should appear on the right side of the prompt.**
- **Simple Computations can be performed by typing the command and pressing enter.**
- **Ex. S= 1+2, fun = sin (pi/4)**

Workspace and Help

- **As we work in the Command Window, MATLAB remembers the commands we enter as well as the variables we create. They reside in the Workspace.**
- **The command "whos/ who" on the command window shows all he variables in the current session.**
- **The "help" system in MATLAB. On the command window type**

"help command" to get help on a particular command; say for example "help LU". In case one doesn't remember the entire command, one can type "lookfor *incomplete command name*."

The command "helpwin" opens a new window which gives various help options. For example "helpwin zeros" gives example on zeros.

The command "demo" can be used to demonstrate examples.

Simple Math

- Like in a calculator, MATLAB can do simple math.

  Addition (+), Multiplication(*),
- Subtraction(-), Division (/ or \), Exponentiation (^)
- Expressions are evaluated from left to right with exponentiation operation having highest precedence followed by multiplication, division having equal precedence, followed by subtraction and addition having equal precedence.
- Parentheses ( ) can be used to alter these rules of precedence.
- Variables: case sensitive, 31 characters, start variable name with a letter.
- Special Variables: ans, pi etc.

Script m files

Why is this needed?

Number of commands increases.

- **Re evaluate a number of commands for more than one value of one or more variables.**
- **All M-files should have .m extension.**
- **For example – ex1.m**

Illustration: Creating a script file, saving and executing (running) it.

COMMENTS, PUNCTUATION AND ABORTING EXECUTION

- ***Semicolon* at the end of a command suppresses the printing of computed results.**
- **All text after a *percent sign (%)* is taken as a comment statement.**
- ***Multiple commands* can be placed on one line if they are separated by *commas* or *semicolons.***

Contd..

- *Commas* tell MATLAB to *display* results; *semicolons suppress* printing.

- MATLAB processing can be interrupted at any moment by pressing Ctrl-C.

Mathematical Functions

- sqrt, sin, cos, sinh, asin, acos, exp, log etc.

(Note: Can handle Specialized mathematical functions
(Bessel, legendre, beta function etc))

NUMBER DISPLAY FORMATS

By default,

- if a result is an integer, it is displayed as an integer.
- if real number, four digits to right of decimal place is displayed.

Script m files-revisited

- *echo on and echo off*- for displaying the commands in the command window as they are read and evaluated in the script M file.

- Helpful during debugging.
  - *input command* - When computations for a variety of cases is desired, the input command allows one to prompt for input as the script file is executed.

Array and Array Operations

Simple arrays :

- **Start with a left bracket, enter the desired values separated by spaces (or commas), and then close it using a right bracket.**

- **Colon notation**

x=first:last
x=first:increment:last

- **linspace command**

x=linspace (first, last, n)
Note: These arrays have one row and multiple columns.

Vectors and operations

Column vectors

- Specify it element by element and separate values with semicolons.

Note: Separating elements by spaces or commas specifies elements in different columns, whereas separating elements by semicolons specifies elements in different rows.

b=a' transposes;

length(a) gives length of the vector 'a'

a.*a is for component wise application of the operator a on a; if

a=[1 2 3]; a.*a=[1 4 9]

a.^m gives m th power of components of a

a.\b gives component wise division of elements of a by elements in b

a *b gives dot product of the vectors a and b,

cross(a,b) gives cross product of 2 three dimensional vectors.

unique(a) gives the same values as in array a but without repetitions.

sort(a) sorts a in ascending order, for matrices sorts columns in ascending order.

setdiff(a,b) returns vector of values in a but not in b.

Matrix Operations

Formation of matrices

Commas or spaces are used to separate elements in a specific row and semicolon are used to separate individual rows.

Alternately, pressing return or enter key while entering a matrix also tells MATLAB to start a new row.

Ex: A=[1 2 3 4; 5 6 7 8; 1 1 2 3] denotes the matrix

with first row as 1 2 3 4, second row as 5 6 7 8,

third row as 1 1 2 3.

To extract a submatrix B consititing of row 1 and 3, columns 2 and 4, we use the command B=A([1 3], [2 4])

Matrix Operations(Contd..)

- To interchange rows 1 and 3 of A use the row indices together with colon operator: C=A([3 2 1], :). The colon operator : stands for all columns or all rows.
- A(:) creates a vector version of a matrix A.
- To delete a row(column) use the empty vector operator [ ];

  for ex. A(:, 2)=[].
- To insert the second column back,

  A=[A(:,1) [2 6 1]' A(:,3) A(:,4)]
- Let d= [1 2 3]; D= diag(d) is a diagonal matrix with diagonals entries in d.

Contd..

Scalar Array Mathematics

- Addition, subtraction, multiplication(.*), division by a scalar(./) applied to all elements in the array/ matrix.

Array – Array Mathematics & Matrix functions

- size(A) gives order of the matrix.
- A' gives transpose of A.
- A(:, 3) gives the third column of A.
- A(1, :) gives first row of A.
- A(1,:) +A(3,:) adds first and third row of A.
- C=A+B adds two matrices A and B.
- C=A-B
- C=A*B
- inv(A) gives inverse of A.
- A*inv(A) gives identity matrix.
- chol(A) gives the Cholesky decomposition of a positive definite matrix A.
- det(A) gives determinant of matrix A.
- rank(A) gives rank of matrix A.

Contd..

- eye(n) generates identity matrix of order n.
- TRACE(A) gives summation of diagonal elements of a square matrix A.
- zeros(n,m) produces an n x m zero matrix.
- rand(n,m) defines an ‘n x m’ matrix having random entries.
- If v is a vector of dimension ‘n’, diag(v) produces a diagonal matrix whose elements are components of v.
- The eigenvalue problem of a matrix is defined by $A\phi=\lambda\phi$ where λ is the eigenvalue of the matrix A and ϕ is the corresponding eigenvector. eig(A) gives the eigenvalues of matrix A and
- [V,D]=eig(A) produces V matrix whose columns are eigenvectors and the diagonal matrix D whose values are eigenvalues of the matrix A.
- The eigenvalue-eigenvector decomposition of a diagonalizable matrix A is calculated using A=VDV^{-1}: V*D/V; note that right division is used as inverse is expensive.
- LU decomposition command decomposes a matrix into a combination of upper and lower triangular matrices respectively.
- [L,U]=lu(A).

- QR decomposition command decomposes a matrix into a combination of an orthonormal and upper triangular matrix. [Q,R]=qr(A).
- Though one can write routines to compute adj(A), cofactor (A) for a matrix A, they are hardly used in practical applications involving large scale computations.

Solution of linear equations : Ax=y

Assuming the system has a unique solution, we can obtain x as:

- x=inv(A)*y (computes inverse explicitly)
- x=A\y (uses LU decomposition)

Computing the CPU time (in seconds) :

- *T=cputime;*
- *Operations….*
- *TT=cputime-T;*

Loops and Logical statements

- **for loop**

```
for k=array
commands
end
```

Multiple for loops is also possible.

```
                for i=1:100
                    for j=1:50
                        for k=1:50
                a(i ,j)=b(i,k)*c(k,j)+a(i,j);
            end
          end
       end
```

Loop and logical statements

- **While**

```
while condition
   statements
end
```

- **if, elseif, else, end**

```
if condition #1
statement #1
elseif condition #2
statement #2
else
statement #3
end
```

Preallocating Arrays

for and while loops that incrementally increase the size of a data structure each time through the loop can adversely affect performance and memory use. Repeatedly resizing arrays often requires that MATLAB spend extra time looking for larger contiguous blocks of memory and then moving the array into those blocks. You can often improve on code execution time by preallocating the maximum amount of space that would be required for the array ahead of time. The following code creates a scalar variable x, and then gradually increases the size of x in a for loop instead of preallocating the required amount of memory at the start:

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

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

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

Vectorization

One way to make your MATLAB programs run faster is to vectorize the algorithms you use in constructing the programs. Where other programming languages might use for loops or DO loops, MATLAB can use vector or matrix operations.
A simple example involves creating a table of logarithms.

```
x=0.01;
for k = 1:1001
 y(k) = log10(x);
 x = x + .01;
end
```

A vectorized version of the same code is:

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

For more complicated code, vectorization options are not always so obvious.

Function Subroutines

- Special type of m file.
- Receives input data through input argument list.
- Returns results to the caller through an output argument list.

The general form is as follows:

```
  function[outarg1,outarg2,...]=fname(inarg1,inarg2,....)
%Comment lines
executable code
return
```

Plotting Tools- 2D plots

- Simple plotting
  (***See graph1.m)***

- Subplot can be used for plotting more than one set of axes in a single figure. Subplots are created with the help of the command

  subplot(m,n,p)

  This creates m x n subplots in the current figure, arranged in m rows and n columns, and selects subplot p to receive all current plotting commands.

  For ex, subplot(2,3,4) will create six subplots in the current figure and will make subplot 4 (the lower left one) as the current subplot.

  (See graph2.m)
- Plot can be used for plotting more than curve in the same figure.
  (See graph3.m)
- loglog to plot using log-log scales, semilogx to plot using logscale along x-axis.

Plotting tools –3D

- A simple 3D plot is shown in ***graph4.m***
- Another examples of 3D graph in ***graph5.m*** and ***graph6.m***
- Modification in figures, adding arrows, text etc can be done directly in the .fig file using the editing tools built into the Figure Tool Bar.
- Tex conventions can be used for legends; for instance \pi. \alpha etc.

Polynomials

- The command *polyval* evaluates a polynomial at one or several points.
- *roots(p)* computes the zeros of p(x).
- To check correctness of this result, evaluate p(x) at roots r to get err=polyeval(p,r) close to 0.
- poly(r) reconstructs polynomials from the roots.
- *conv(p1,p2)* gives the coefficients of the polynomial given by the product of two polynomials whose coefficients are contained in the vectors *p1 and p2.*
- *polyder(p)* gives the coefficient of *p'(x).*
- *polyint(p)* gives coefficients of integral of *p(x).*

Symbolic Tools

Tool box *symbolic* provides the commands of

- **diff**
- **int**
- **taylor**

which allows us to obtain the analytical expressions of the derivative, indefinite integral and the Taylor polynomial respectively of a given function.

General Instructions for Numerical Analysis Programs

- **Declare the array variables to zero.**
- **Use clear all statement in the beginning of the code.**
- **Use long double for real arrays.**
- **Declare the matrix as “sparse” in case you are working with a sparse matrix to save memory and computation time. For ex. a sparse matrix of dimension 10* 100 is initialized by A = sparse(10,100). Huge identity matrices can be also declared as sparse matrix by using A=speye(100).**
- **A full matrix A can be transformed to a sparse matrix by using the command A=sparse(A).**
- **In case, a sparse matrix needs to be transformed to a full matrix, use the command A=full(A).**
- **Use vectorization as far as possible.**
- **Assign space in memory for arrays and matrices in the beginning.**

Reference

- 1. Cleve B. Moler, Numerical Computing with MATLAB, SIAM, Philadelphia, 2004.