

UNIVERSITY OF MINNESOTA

 *Supercomputing Institute*
for Digital Simulation and Advanced Computation

Computation with Matlab

Shuxia Zhang

Supercomputing Institute

For Digital Simulation and Advanced Computation

e-mail: szhang@msi.umn.edu, help@msi.umn.edu

Tel: 612-624-8858 (direct) , 612-626-0802(help)

Outline

Introduction

Basic Math Operations

Input and Output

Solving $A X = B$

Symbolic Math

M-files

Submit Matlab jobs to the queues

Hands-on

Reference

Introduction

- MATLAB handles a range of computing tasks in engineering and science, from data acquisition and analysis to application development.
- Focus on high-level technical concepts and ignore programming details.
- One can interactively run line by line command. One can also write a MATLAB code (referred as M-file) and run it in a batch mode.
- Interactive language and programming environment. M-files require no compiling or linking, so you can edit and debug an M-file and test the changes immediately.

How to Start Matlab

Type:

```
module load matlab  
matlab -nojvm
```

Search information, type the followings on MATLAB window

```
>> help known-name  
>> lookfor string
```

Demos:

```
>> demo
```

Always type lowercase for the name or string with online help.

“>>” marks the commands that one can type on MATLAB window.

MATLAB Toolboxes

Bioinformatics Toolbox
Communications Toolbox
Control System Toolbox
Curve Fitting Toolbox
Data Acquisition Toolbox
Database Toolbox
Datafeed Toolbox
Excel Link
Filter Design Toolbox
Financial Toolbox
Financial Derivatives Toolbox
Financial Time Series Toolbox
Fixed-Income Toolbox
Fuzzy Logic Toolbox
GARCH Toolbox
Genetic Algorithm Toolbox
Image Acquisition Toolbox
Image Processing Toolbox
Instrument Control Toolbox

LMI Control Toolbox
Mapping Toolbox
Model-Based Calibration Toolbox
Model Predictive Control Toolbox
**Mu-Analysis and Synthesis
Toolbox**
Neural Network Toolbox
Optimization Toolbox
**Partial Differential Equation
(PDE) Toolbox**
Robust Control Toolbox
Signal Processing Toolbox
Spline Toolbox
Statistics Toolbox
Symbolic Math Toolbox
System Identification Toolbox
Virtual Reality Toolbox
Wavelet Toolbox

Elementary Math and Matrix Manipulation Functions

Elementary math functions

Trigonometric.

- sin - Sine.
- cos - Cosine.
- tan - Tangent.
- sec - Secant.

Exponential.

- exp - Exponential.
- log - Natural logarithm.
- log10 - Common (base 10) logarithm.
- log2 - Base 2 logarithm
- sqrt - Square root

Complex.

- abs - Absolute value.
- complex - Construct complex data
- conj - Complex conjugate.
- imag - Complex imaginary part.
- real - Complex real part.

Matrix initialization.

- zeros - Zeros array.
- ones - Ones array.
- eye - Identity matrix.
- rand - Uniformly distributed random
- randn - Normally distributed random numbers.
- linspace - Linearly spaced vector.
- logspace - Logarithmically spaced vector.

```
>> help logspace
```

Basic array information.

- size - Size of matrix.
- length - Length of vector.
- ndims - Number of dimensions.
- disp - Display matrix or text
- spy - View the matrix structure

```
>> lookfor key_word
```

Basic operations

Create an array or an vector:

```
>> a = [1 2 3 4 5 6 7 8 9]
```

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

plus:

```
>> b = a+2
```

```
b =  
 3 4 5 6 7 8 9 10 11
```

Creating a matrix is as easy as making a vector

```
>> A = [1 2 0; 2 5 -1; 4 10 1]
```

```
A =  
 1 2 0  
 2 5 -1  
 4 10 1
```

Initialization functions

ZEROS(M,N) is an M-by-N matrix of zeros.

```
>> zeros(2,3)
```

```
ans =
```

```
0 0 0
0 0 0
```

ONES(M,N) is an M-by-N matrix of ones.

```
>> ones(3,2)
```

```
ans =
```

```
1 1
1 1
1 1
```

EYE(M,N) is an M-by-N matrix with 1's on the diagonal and zeros elsewhere.

```
>> eye(4,4)
```

```
ans =
```

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

Inner products

dotprod is the dot product weight function.

Inputs:

W – input (row) 1x n matrix

P - input (n columns) vectors.

dotprod(W,P) returns the dot product of W and P.

```
>> w=[1 3 5 7];
```

```
>> p=[2; 4; 6; 8]
```

```
>> dotprod(w,p)
```

```
ans =
```

```
100
```

```
>> dotprod(p,w)
```

```
ans =
```

```
2 6 10 14
```

```
4 12 20 28
```

```
6 18 30 42
```

```
8 24 40 56
```

NORM: Matrix or vector norm

For matrices...

$\text{NORM}(X)$ is the largest singular value of X , $\max(\text{svd}(X))$.

$\text{NORM}(X,2)$ is the same as $\text{NORM}(X)$.

$\text{NORM}(X,1)$ is the 1-norm of X , the largest column sum,
 $= \max(\text{sum}(\text{abs}((X))))$.

$\text{NORM}(X,\text{inf})$ is the infinity norm of X , the largest row sum,
 $= \max(\text{sum}(\text{abs}((X'))))$.

$\text{NORM}(X,\text{'fro'})$ is the Frobenius norm, $\sqrt{\text{sum}(\text{diag}(X'*X))}$.

$\text{NORM}(X,P)$ is available for matrix X only if P is 1, 2, inf or 'fro'.

For vectors...

$\text{NORM}(V,P) = \text{sum}(\text{abs}(V).^P)^{(1/P)}$.

$\text{NORM}(V) = \text{norm}(V,2)$.

$\text{NORM}(V,\text{inf}) = \max(\text{abs}(V))$.

$\text{NORM}(V,-\text{inf}) = \min(\text{abs}(V))$.

SPARSE

SPARSE(X) converts a sparse matrix to sparse form by squeezing out any zero elements. Save memory
But it may use more memory if the matrix is dense.

Example:

```
>> p=rand(10,20);
>> for i=1:10
    for j=1:20
        if (p(i,j) <0.5 ) T(i,j) = 0;
        else
            T(i,j) = p(i,j);
        end;end;end
>> sparse(T);
>> whos
```

Name	Size	Bytes	Class
T	10x20	1356	sparse array
i	1x1	8	double array
j	1x1	8	double array
p	10x20	1600	double array

SPARSE

Create Sparse Matrix:

`S = sparse(i,j,s,m,n,nzmax)`

S: created m-by-n sparse matrix

nzmax: allocated zeros

i and j: integer index vectors

s: real or complex vector (non-zeros);

i,j, and s all have the same length

`m=max(i)`

`n=max(j)`

Example:

```
>> s=rand(10,1);
```

```
>> i=[1,3,4,6,8,9,14,18,24,26];
```

```
>> j=[1,2,5,7,14,11,13,9,18,26];
```

```
>> m=max(i);
```

```
>> n=max(j);
```

```
>> S=sparse(i,j,s,m,n) %It will create a 26-by-26 sparse matrix.
```

```
>> spy(S) % will show the structure of the sparse matrix
```

Basic matrix operations

Given: $A = [1 \ 2 \ 0; 2 \ 5 \ -1; 4 \ 10 \ 1];$

Transpose of the matrix A

```
>>B = A'
B =
     1     2     4
     2     5    10
     0    -1     1
```

Matrix multiplication

```
>>C=A*B
C =
     5    12    24
    12    30    57
    24    57   117
```

Multiply the corresponding elements of two matrices or vectors

```
>>C = A.*B
C =
     1     4     0
     4    25   -10
     0   -10     1
```

Basic matrix operations

INV(X) is the inverse of the square matrix X

Given: $A = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 5 & -1 \\ 4 & 10 & 1 \end{bmatrix}$;

```
>> X = inv(A)
```

```
X =
```

```
5.0000 -0.6667 -0.6667  
-2.0000 0.3333 0.3333  
0 -0.6667 0.3333
```

```
>> I = inv(A)*A
```

```
I =
```

```
1 0 0  
0 1 0  
0 0 1
```

Basic matrix operations

$E = \text{eig}(X)$ -- a vector containing the eigenvalues of a square matrix **X**
 $[V,D] = \text{eig}(X)$ -- a diagonal matrix **D** of eigenvalues and a full matrix **V**,
 whose columns are the corresponding eigenvectors so
 that **$X*V = V*D$** .

Example:

Given: $A = [1 \ 2 \ 0; 2 \ 5 \ -1; 4 \ 10 \ 1];$

$\gg E = \text{eig}(A)$

$E =$

0.1911

$3.4045 + 2.0270i$

$3.4045 - 2.0270i$

$\gg [V,D] = \text{eig}(A)$

$V =$

-0.9258 $-0.1555 + 0.0304i$ $-0.1555 - 0.0304i$

0.3745 $-0.2178 - 0.1212i$ $-0.2178 + 0.1212i$

-0.0510 $-0.9041 + 0.3088i$ $-0.9041 - 0.3088i$

$D =$

0.1911 0 0

0 $3.4045 + 2.0270i$ 0

0 0 $3.4045 - 2.0270i$

Input and Output

Save: saves workspace or variables to disk:

```
>> save          % saves all variables to the default binary file "matlab.mat"
>> save fname    % saves all variables to the file with the given name.
>> save fname X Y % save only variables X and Y.
>> save fname X Y -append % adds the variables to an existing mat-file.
>> save fname -ascii      % uses 8-digit ASCII form instead of binary.
>> save fname -ascii -double % uses 16-digit ASCII form.
```

To read in the mat-files, one need to use the load command, i.e.,

```
>> load fname
```

To delete a variable in the memory

```
>> clear X
```

Input and Output

I/O formats ---- compatible with other computer languages.

The commonly used ones include:

fopen - Open file.

fclose - Close file.

Input:

fread - Read binary data from file.

textread - Read formatted data from text file.

fscanf - Read formatted data from file.

load - Load workspace from MAT-file or ASCII file.

Output

save - Save workspace variables to disk

fwrite - Write binary data to file.

fprintf - Write formatted data to file.

I/O examples

```
fread:          >> fid = fopen('input.dat','r');
                >> [A, COUNT] = fread(fid,size,precision,skip)
```

size -- optional; if not specified, the entire file is read; else it can be:

N read N elements into a column vector.

inf read to the end of the file.

[M,N] read elements to fill an M-by-N matrix, in column order.
N can be inf, but M can't.

precision-- type of data that MATLAB supports, like schar, int8, single, double, etc,...

skip -- the number of bytes or bits to skip, dependent on the data type.

fprintf :

```
>> x = 0:.1:1; y = [x; exp(x)];
>> fid = fopen('exp.txt','w');
>> fprintf(fid,'%6.2f %12.8f\n',y);
>> fclose(fid);
```

Hilbert Matrix

`hilb(N)` produces the N by N matrix with elements $1/(i+j-1)$.

```
>> n = 12;
>> A = hilb(n);
>> b = A * ones(n,1);
>> x = A \ b;
>> err = ones(n,1)-x
err=
    0.0000
   -0.0000
    0.0001
   -0.0017
    0.0129
   -0.0585
    0.1671
   -0.3088
    0.3687
   -0.2743
    0.1157
   -0.0211
```

Solving $A X = B$

```
>>A = [1.0000  0.5000  0.3333  0.2500  0.2000  0.1667  0.1429;  
       0.5000  0.3333  0.2500  0.2000  0.1667  0.1429  0.1250;  
       0.3333  0.2500  0.2000  0.1667  0.1429  0.1250  0.1111;  
       0.2500  0.2000  0.1667  0.1429  0.1250  0.1111  0.1000;  
       0.2000  0.1667  0.1429  0.1250  0.1111  0.1000  0.0909;  
       0.1667  0.1429  0.1250  0.1111  0.1000  0.0909  0.0833;  
       0.1429  0.1250  0.1111  0.1000  0.0909  0.0833  0.0769]
```

```
>> B= [2.5929 1.7179 1.3290 1.0956 0.9365 0.8199 0.7301]';
```

To get a solution for X

```
>> C = inv (A);
```

```
>> X = C*B
```

What is X ?

Solving $A X = B$ iteratively

<http://www.siam.org/books/kelley/kellcode.htm>

Linear Equations:

pcgsol.m: Preconditioned CG

Gmres.m: Brute force GMRES

gmres.m: GMRES -- requires givapp.m .

bicgstab.m : Bi-CGSTAB

tfqmr.m : TFQMR

Nonlinear Equations

nsol.m : Basic Newton-Shamanskii solver, difference
Jacobian, LU factorization.

brsol.m: Locally convergent Broyden solver

nsolgm.m: Newton-GMRES solver.

A lot of more!

Symbolic Math

- Integrates powerful symbolic and variable precision computing
- Symbolic math support is not yet for every platform
 - Solaris systems -- Yes
 - SGI Irix -- Yes
 - Linux (l1, l2) -- No
- Symbolic variables must be defined via the **sym** function
- One defined symbolic variable can be assigned to a new undefined variable

Example

```
>> a = sym('a'); t = sym('t'); x = sym('x'); y = sym('y');
```

or

```
>> syms a t x y
```

```
>> b=a; % now b is a symbolic variable
```

Symbolic Math

Symbolic variables in expressions and as arguments to many different functions.

```
>> r = x^2 + y^2
>> theta = atan(y/x)
>> e = exp(i*pi*t)
```

Derivatives and integrals are computed by the **diff** and **int** functions

```
>> diff(e)
ans = i*pi*exp(i*pi*t)
>> diff(x^3)
ans = 3*x^2
>> int(x^3)
ans = 1/4*x^4
>> int(exp(-t^2))
ans = 1/2*pi^(1/2)*erf(t)
```

Symbolic Math

MATLAB's vector and matrix notation extends to symbolic variables.

```
>> n = 4;  
>> A = x.^((0:n)'*(0:n))  
A =  
[ 1, 1, 1, 1, 1]  
[ 1, x, x^2, x^3, x^4]  
[ 1, x^2, x^4, x^6, x^8]  
[ 1, x^3, x^6, x^9, x^12]  
[ 1, x^4, x^8, x^12, x^16]
```

```
>> D = diff(log(A))  
D =  
[ 0, 0, 0, 0, 0]  
[ 0, 1/x, 2/x, 3/x, 4/x]  
[ 0, 2/x, 4/x, 6/x, 8/x]  
[ 0, 3/x, 6/x, 9/x, 12/x]  
[ 0, 4/x, 8/x, 12/x, 16/x]
```

Symbolic Math

The **solve** and **dsolve** functions seek analytic solutions to algebraic and ordinary differential equations.

One can either find the zeros of a symbolic expression, without quotes:

```
>> syms a b c x
```

```
>> x = solve(a*x^2 + b*x + c); <- no longer working
```

To find the roots of an equation, given in quotes:

```
>> x = solve('a*x^2 + b*x + c = 0');
```

```
x =
```

```
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
```

```
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

Symbolic Math

dsolve('eqn1','eqn2', ...) accepts symbolic equations representing ordinary differential equations and initial conditions. Several equations or initial conditions may be grouped together, separated by commas, in a single input argument.

```
>> y = dsolve('Dy = -a*y')  
y = C1*exp(-a*t)
```

Specify an initial condition.

```
>> y = dsolve('Dy = -a*y','y(0) = 1')  
y =  
exp(-a*t)
```

Symbolic Math

The second derivative is denoted by "D2".

```
>> y = dsolve('D2y = -a^2*y', 'y(0) = 1, Dy(pi/a) = 0')
```

```
y =  
    cos(a*t)
```

A nonlinear equation produces two solutions in a vector.

```
>> y = dsolve('(Dy)^2 + y^2 = 1','y(0) = 0')
```

```
y =  
    [ sin(t)  
    [-sin(t)]
```

Symbolic Math

simple(S) does not display intermediate simplifications, but returns the shortest found

simplify(S) simplifies each element of the symbolic matrix S

pretty(S) prints the symbolic expression S in a format that resembles type-set mathematics.

Example 1:

```
>> syms x y positive
>> r=simple(log(x) + log(y));
r = log(x*y)
>> r=simplify(log(x) + log(y));
r = log(x)+log(y)
```

Example 2:

```
>> syms a b c; alpha=a; beta=b;
>> simplify(exp(c*log(sqrt(alpha+beta))))
ans = (a+b)^(1/2*c)
>> pretty(exp(c*log(sqrt(alpha+beta))))
ans=
          1/2
exp(c log((a + b)  ))
```

M-file

The M-files let you capture your command-line explorations as permanent, reusable MATLAB functions.

Suppose you have a M-file, or you saved the command-line operations into a M-file, named as `matlab_test.m`, to run this code interactively, just type:

```
module add matlab  
matlab < matlab_test.m > output
```

Note: `quit` should be the last command in the M-file.

One can also submit the MATLAB jobs to the queue.

Submit jobs to the queue

An example of LoadLeveler script file

```
#!/bin/csh
#@ initialdir = /homes/sp9/szhang
#@ output    = matlab.out
#@ error     = matlab.err
#@ requirements = (Feature == "Ethernet")
#@ wall_clock_limit = 10:00:00
# to run the matlab for 10 hours
#@ queue

module add matlab
matlab -nojvm -nodisplay <small.m > std.out
```

An example of PBS script file

```
#PBS -l ncpus=1,mem=1gb,walltime=30:00
#PBS -l arch=r14k
#PBS -m abe
cd /home/smpb/szhang/matlab_batch

module add matlab
matlab -nojvm -nodisplay < particles.m
```

Hands On

- 1) Create a Hilbert Matrix A for dimension $i=j=12$, a vector

$$B=[1, 0.5, 0.4, -2.0, -4.6, -1.2, 0.9, 1.0, 0.0, 0.0, 0.0, 1.0]T$$

and solve the equation for unknown X

$$AX = B$$

- 2). Practice the examples presented in the lecture.
- 3). Learn to use the PDE solvers

www.msi.umn.edu/software/matlab/tutorials/computation/PDE.html

Reference

On-line help

www.mathworks.com/products/matlab

Need help? Send e-mail to:

szhang@msi.umn.edu

help@msi.umn.edu

Or call at

612-624-8858 (direct)

612-626-0802

**This workstation room is reserved
for a workshop on April . 7, 2005
(Thursday) from 1:00pm to 3:00pm**

Thank you much for your cooperation