Matlab, Introduction

Resources: intro
2. Matlab on Athena (MIT computer services web page).
   medium level
   more advanced
   also
7. Lecture notes: 10.001 web page.

October-November 2002
Main Features of Matlab

• Matlab = matrix laboratory, matrix oriented programming language + desktop environment.
• Any variable is an array by default, thus almost no declarations. All variables are by default double.
• Translator - interpreter: line after line, no .exe files.
• High level language:
  (i) quick and easy coding
  (ii) tools assembled into toolboxes (Spectral Analysis, Image Processing, Signal Processing, Symbolic Math etc.)
  (iii) relatively slow.
• All Matlab functions are precompiled.
• One may add extra functions by creating M-files and modify the existing ones.
Comparison with C.

- Syntax is similar
- Language structure is similar to C:
  - MATLAB supports variables, arrays, structures, subroutines, files
  - MATLAB does NOT support pointers and does not require variable declarations
  - MATLAB has extra built-in tools to work with matrices/arrays
Matlab, Getting Started

1. Accessing Matlab on Athena:
add matlab
matlab &
2. Log out: quit or exit

MATLAB desktop (version 6):
1) Command window
2) Launch Pad / Workspace window
3) Command history / Current Directory window
Useful Hints & Commands

- input: variable_name  \rightarrow output: variable_value
- semicolon at the end will suppress the output
- command history: upper & lower arrows, also command name guess:
  (i) type abc
  (ii) hit “upper arrow” key  \rightarrow get the last command starting from abc
- `format compact` - no blank lines in the output
  `format loose` - back to default
- `help commandname` - info on commandname
Workspace Maintenance

- all the assigned variables “reside” in the workspace
- `clear all` - clears all the memory (workspace)
- `clear xyz` - removes `xyz` from the memory
- `who` - lists all the variables from the workspace
- `whos` - also gives the details

```plaintext
>> who
Your variables are:
ans    c1    c2
>> whos
Name     Size   Bytes   Class
ans  1x1     8      double array
c1   1x1    16      double array(complex)
c2   2x2    64      double array(complex)
```
Workspace Maintenance

- **save** saves all workspace variables on disk in file `matlab.mat`

- **save filename x y z** - `x, y, z` are saved in file `filename.mat`

- **load filename** - loads contents of the `filename.mat` to the workspace

- **load filename x y z** - loads only `x, y, z` from `filename.mat` to the workspace

- Each array requires a continuous chunk of memory; use `pack` for memory defragmentation.
Built in Constants & Functions

- pi – π number
- i & j stand for “imaginary one” (i = -1^{1/2}), however may be redefined
- Trigonometric: sin, cos, tan, sec, cot
- Inverse trig.: asin, acos, atan, asec, acot
- Exponential: log, log2, log10, exp
- Complex: abs – abs. value, angle – phase angle, conj – conjugate transpose, imag – imaginary and real- real part
**Linear Algebra**

**Vector**: an ordered set of real or complex numbers arranged in a row or column.

\[
x = \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_m
\end{bmatrix}
\]
m-element column vector \((n \times 1)\)

\[
y = \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{bmatrix}
\]
n-element row-vector \((1 \times n)\)

October-November 2002
Vector Operations

Addition/subtraction (element-wise, array operation:
\[ c = a + b \rightarrow c_i = a_i + b_i, \quad i = 1, \ldots, n \]
\[ d = a - b \rightarrow c_i = a_i - b_i, \quad i = 1, \ldots, n \]

Multiplication/division by a scalar:
\[ b = \alpha a \rightarrow b_i = \alpha a_i \]
\[ b = a/\alpha \rightarrow b_i = a_i/\alpha \]

Vector transpose, turns row into column and vise versa:
\[ x = [1, 2, 3] \rightarrow x^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (x^T)^T = x \]
Vector Operations

Vector inner (dot, scalar) product (vector/matrix operation):

\[ a = x \cdot y \quad \Rightarrow \quad a = \sum_{i=1}^{n} x_i y_i \quad a = y^T \cdot x^T \]

x is a row vector

y is a column vector

The dimensions of x and y must agree.

NB General rule for vector/matrix multiplication:
“row times column” – take i-th row of the left multiplier and multiply by the j-th column of the right multiplier
Vector Operations

Outer (tensor) product:

\[ M = yx = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} y_1x_1 & y_1x_2 & y_1x_3 & y_1x_4 \\ y_2x_1 & y_2x_2 & y_2x_3 & y_2x_4 \\ y_3x_1 & y_3x_2 & y_3x_3 & y_3x_4 \\ y_4x_1 & y_4x_2 & y_4x_3 & y_4x_4 \end{bmatrix} \]

\[ M_{ij} = x_i y_j \]
Vector Norms

To compare two vectors a vector norm (analogous to length, size) is introduced:

\[ \|x\| > \|y\| \rightarrow \text{“norm of } x \text{ is greater than norm of } y\” \]

Euclidian norm, length in nD (also called L₂ norm):

\[ \|x\|_2 = (x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2)^{1/2} \]

\[ \|x\|_1 = |x_1| + |x_2| + |x_3| + \ldots + |x_n| \]  \hspace{1cm} (L₁)

\[ \|x\|_{\text{inf.}} = \max(|x_1|, |x_2|, |x_3|, \ldots |x_n|) \]

\[ \|x\|_p = (x_1^p + x_2^p + x_3^p + \ldots + x_n^p)^{1/p} \]  \hspace{1cm} (Lₚ)

October-November 2002
Dealing with Vectors/Matrices

Entering matrices by explicit list of elements:

\[ A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \]

\[ A = \begin{bmatrix} 1; & 2; & 3 \end{bmatrix} \]

Spaces separate the elements, semicolons and "new line" symbols separate the rows.
Dealing with Matrices

Complex matrices:
either \( A=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}+i\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \)

or \( A=\begin{bmatrix} 1+5i & 2+6i \\ 3+7i & 4+8i \end{bmatrix} \)

No blank spaces, i or j stands for “imaginary one”.

Matrix and array operations, classification.

\[
\begin{align*}
+ & \quad \text{element-wise (array operations)} \\
- & \quad \text{array or matrix operations} \\
\cdot & \quad \text{conjugate transpose} \\
\backslash & \quad \text{left division} \\
/ & \quad \text{right division}
\end{align*}
\]

only matrix operations
Matrix Multiplication

Product of the two matrices $A(n \times m)$ and $B(m \times l)$ is matrix $C(n \times l)$:

$$ C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \ldots + A_{im}B_{2jm} $$

The dimensions of $A$ and $B$ must agree.

$C_{ij}$ – is a dot product of $i$-th row of $A$ and $j$-th column of $B$.

Again “row (on the left) times column (on the right)”.

If $A$ or $B$ is a vector, we will have either

“row vector times matrix = column vector” or

“matrix times column vector = row vector”
Dealing with Matrices

In Matlab *left* and *right division* are inverse to multiplication by column and row vectors correspondingly:

\[ A \times x = b \Rightarrow x = A \div b \ (\text{left}) \ A - \text{matrix } m \times n, \ b - \text{row vector } 1 \times n \]

\[ x \times A = b \Rightarrow x = b \div A \ (\text{right}) \ b - \text{column vector } m \times 1 \]

*Conjugate transpose*: swaps the indices and changes the sign of imaginary part of each element.

\[ C = A^\prime \]

\[ C(i,j) = \text{real}(A(j,i)) - i \times \text{imag}(A(j,i)) \]
Dealing with Matrices, Examples

```
>> C = A + B;
C(k,l) = A(k,l) + B(k,l)

>> C = A*B;
C(k,l) = A(k,m) * B(m,l)
```

Matrix multiplication, summation over the repeating index is implied.

```
>> C = A.*B
C(k,l) = A(k,l)*B(k,l)
```

Element-wise (array) operation, imposed by ".".

```
>> C = A^alpha;
>> C = A.^alpha;
C(k,l) = A(k,l)^alpha
```

Matrix A to the power alpha
Each element of A to the power alpha
Dealing with Matrices

*Standard math. functions of matrices* operate in array sense:
\[\exp(A), \sin(A), \sqrt{A} = A.\text{^0.5}\]

\[
\begin{align*}
\text{>> } & B = \exp(A) \quad \Rightarrow \quad B(i,j) = \exp(A(i,j)) \\
\end{align*}
\]

*Colon notation* is used:
(i) to construct vectors of equally spaced elements:
\[
\begin{align*}
\text{>> } & a = 1:6 \quad \Rightarrow \quad a = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\
\text{>> } & b = 1:2:7 \quad \Rightarrow \quad b = 1 \quad 3 \quad 5 \quad 7 \\
\end{align*}
\]
(ii) to access submatrices:
\[A(1:4, \ 3)\] - column vector, first 4 elements of the 3-d column of A.
\[A( : , 3 )\] - the 3-d column of A
(iii) in “for” loops
Relational & Logical Operators & Functions

• R&L operations are needed for computer to make decisions and take actions once some conditions are satisfied.

Example – while loops

• Argument of R&L operations is true if it is non-zero and false if it is zero; output is 1 for true and zero for false.

• Relational: <, <=, >, >=, ==, ~=.

Operate on matrices in elementwise fashion.

```matlab
>> A = 1:9, B = 9 - A
A =  1  2  3  4  5  6  7  8  9
B =  8  7  6  5  4  3  2  1  0
>> tf = A > 4
tf = 0 0 0 0 1 1 1 1 1
>> tf = (A==B)
0 0 0 0 0 0 0 0 0
```
Relational & Logical Operators & Functions

• Logical:  & AND;  |  OR;  ~  NOT.

>> tf = ~(A>4)

\[ \text{tf} = 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \]

>> tf = (A>2) & (A<6)

\[ \text{tf} = 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \]

• Functions: \( x \oplus y \) - exclusive OR, true if either \( x \) or \( y \) is non-zero, false of both are true or false.

isempty - true for empty matrix

isreal, isequal, isfinite,...
Flow of Control

For loops. Syntax:

```matlab
for x = array
  (commands)
end
```

Example:
```
>> for n = 1:10
  x(n) = sin(n*pi/10);
end
```
Nested loops, decrement loop.

```matlab
>> for n = 1:5
    for m = 5:-1:1
        A(n,m) = n^2 + m^2;
    end
end
```

Alternative: *vectorized* solution, much faster: assigns memory for `x` only once.

```matlab
>> n = 1:10;
>> x = sin(n*pi/10)
```
Flow of Control

*While loops*. Syntax:

```plaintext
while expression
    (commands)
end
```

(commands) will be executed as long as all the elements of expression are true.

Example: search for the smallest number EPS which if added to 1 will give the result greater than 1.
Flow of Control

```matlab
>> num = 0; EPS = 1;
>> while (1+EPS)>1
    EPS = EPS/2;
    num = num+1;
end
>> num
num = 53
>> EPS = 2*EPS
EPS = 2.2204e-16
```
Flow of Control

If-Else-End constructions. Syntax:

if expression1
    (commands1: if expr-n1 is true)
elseif expression2
    (commands2: if expr-n2 is true)
elseif expression3
    (commands3: if expr-n3 is true)
    ...
else
    (commands: if 1,2,...,n are false)
end
Flow of Control

Breaking out of the loop:

```matlab
>> EPS = 1;
>> for num = 1:1000
    EPS = EPS/2;
    if (1+EPS)<=1
        EPS = EPS*2
        break
    end
end
EPS = 2.2204e-16
```
M-files

Script files & Function files

*Script files:* contain a set of Matlab commands - programs. To execute the file: enter the file name.

```matlab
% script M-file example.m
erasers = 4; pads = 6; tapes = 2;
items = erasers + pads + tapes
cost = erasers*25 + pads*52 + tapes*99
average_cost = cost/items

>>example
items =  12
cost = 610
average_cost = 50.833
```

October-November 2002
M-files

Interpreter actions while processing `example` statement:
1. Is `example` a current Matlab variable?
2. Is `example` a built-in Matlab command?
3. Is `example` an M-file?
4. Opens the file and evaluates commands as if they were entered from the command line.

(i) all workspace variables are accessible to the commands form the M-file.
(ii) all variables created by M-file will become a part of the workspace if declared global.
**M-files**

*Function files*
- Analogous to functions in C.
- Communicate with the workspace only through variables passed to it and the output variables it creates. All internal variables are invisible to the main workspace.
- M-file’s name = function’s name.
- The first line - *function-declaration line*

```
function s=area(a,b,alpha)
```

- **‘function’ keyword**
- **output variable**
- **function name**
- **expected input arguments**
function s=area(a,b,alpha)
% AREA calculates triangle’s area given 2 sides & angle between them
% AREA reads in two sides of the triangle and the angle between them
% (in radians) and returns the area of the triangle.

if a < 0 | b<0
    error(‘a and b can not be negative.’)
end
s = a*b*sin(alpha)/2;

Terminates execution of the M-file

searched and displayed by the lookfor command
searched and displayed by the help command
Function M-files

- Function M-files may call script files or other (sub)functions, the script file/subfunction being evaluated in the function’s workspace.
- Function M-files may have zero input and output arguments.
- Functions may share variables. The variable must be declared as `global` in each desired workspace.
Some Helpful Commands

\[ [n,m] = \text{size}(A) \] – dimensions of matrix A
\[ n = \text{length}(B) \] - the length of vector B
\[ \text{zeros}(m,n) \] – creates \( m \times n \) matrix of zeros
\[ \text{ones}(m,n) \] – creates \( m \times n \) matrix of ones
\[ \text{eye}(n) \] – \( n \times n \) matrix, ones on the diagonal, zeroes elsewhere
\[ x = \text{linspace}(s,f,n) \] - x-vector of \( n \) equally spaced elements form \( s \) up (down) to \( f \), similar to
\[ x = s : ((f-s)/(n-1)) : f \]
Graphics

- Each graph is created in a figure window
- By default only one figure window can be opened, thus the second graph will replace the first one

To create a graph you run:

- Management functions (arranging the figure window(s))
- Graph generation functions
- Annotation functions (formatting the graphs, optional)
## Graphics

### Plotting functions: 3 categories

<table>
<thead>
<tr>
<th>Management</th>
<th>Generation</th>
<th>Annotation &amp; characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>figure</td>
<td>plot, polar, fill, plotyy</td>
<td>xlabel, ylabel, zlabel, text, title, legend, box, set, grid</td>
</tr>
<tr>
<td>subplot</td>
<td>plot3, surf, surf3, mesh, meshz</td>
<td></td>
</tr>
<tr>
<td>zoom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>view</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rotated</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graphics management

- **figure(n)** – opens figure window number n, also makes window n default window; the new **plot()** command will replace the plot in the default window
- **hold on** – holds the current window active, you may add curves using **plot()** command
- **hold off** – “releases” the current window
- **subplot(i,j,k)** – divides figure window into i x j array of sectors for plots; k – number of the sector to put the plot in
Two-Dimensional Graphics:

• “join-the-dots” x-y plot

```
>> x = [1.2 2.3 3.7 4.1 5.0 7.3];
>> y = [2.6 2.8 3.7 4.0 4.3 5.6];
>> plot(x,y)
```

Syntax: `plot(x,y,string)`. 
String (optional) stands for color, marker and plot style. Example: `r*--` -red, asterisk at each data point, dashed line. Colors: r, g, b, c, m, y, k, w. Line styles: – solid, -- dashed, : dotted, -. dash-dot.
Plotting many curves:
```
plot(x, y, 'r-', a, b, 'g--', ....)
```

Some other control strings:
```
'LineWidth', 2, 'MarkerSize', 7, 'MarkeFaceColor', 'r', ...
```
```
plot() -> loglog() 
```
changes lin-lin plot to log-log one.
Graphics

Labels and title:
\texttt{xlabel('concentration')}\\
\texttt{ylabel('viscosity')}\\
\texttt{title('C(\eta)\ plot,\ PEO - H_2O\ solution.' )}

Axes:
\texttt{axis([xmin\ xmax\ ymin\ ymax])},\\
\texttt{xlim([xmin\ xmax]),axis\ tight,\ grid\ on},\\
\texttt{axis\ square,\ ......}

Also go to “edit” option of the plot window.
Adding text box at the position (x,y):
{
    text(x,y,'here is the text');
}

Multiple plots in a Figure:
{
    subplot(k,m,1), plot(......)
    subplot(k,m,2), plot(......)
    subplot(k,m,k*m), plot(......)
}

k, m - number of lines and columns in the array of plots, 1,2,...k*m - number of the plots in the array.
plot([x_1, x_2], [y_1, y_2]) – plots a straight line form (x_1, y_1) to (y_1, y_2)

Let’s plot a set of straight lines: connecting (x_{1j}, y_{1j}) and (x_{2j}, y_{2j}) for j=1,...,n. The plot instruction will be:

plot([x_1; x_2], [y_1; y_2]).

Say, x_1=[1 3 5]; x_2=x_1; y_1=[0 0 0]; y_2=[3 6 2]; - 3 vertical lines will be plotted.
Three-dimensional Graphics

The 3D version of plot is:

\[
\text{plot3}(x_1, y_1, z_1, S_1, x_2, y_2, z_2, S_2, \ldots)
\]

3 coordinates, control string, 3 coordinates...

Example: \(\sin(x), \cos(x), x\). Plots function of a function.

\[
\text{plot3}([x_1; x_2], [y_1; y_2], [z_1; z_2])
\]

Arguments – vectors of \(n\) elements each. \(x_1, x_2\) store \(x\)-coordinates of the points, where lines 1, \(\ldots, n\) begin and end correspondingly,

\(y_1, y_2\) and \(z_1, z_2\) do the same for \(y\) and \(z\) coordinates.
Three-dimensional Graphics

3D, scalar functions of 2 variables, mesh plots:

\[ z = f(x,y) \]

Plot of \( f(x,y) \) - surface in 3-d.

1. Create a mesh in x-y plane:

\[
\begin{align*}
&>> ~ x = x0:x1, \ y = y0:y1 \\
&>> [X, Y] = \text{meshgrid}(x,y)
\end{align*}
\]

\( x \) has \( m \) and \( y \) has \( n \) elements, \( X \) & \( Y \) - matrices nxm,
\( X \) consists of \( n \) row vectors \( x \), \( Y \) of \( m \) column vectors \( y \).
Each pair \( X(i,j) \) & \( Y(i,j) \) gives coordinates in x-y space.
Three-dimensional Graphics

X & Y may be treated as matrices or arrays.
If \( z = f(x,y) = 3(x^2+y)^3 \):

\[
>> Z = 3 \times (X.\,^2 + Y) .\,^3 \quad \% \text{Matrix Z is created}
\]

\[
>> \text{mesh}(X, Y, Z) \quad \% \text{Draws mesh plot of f(x,y)}
\]

\text{meshc} - draws the underlying contour plot

\text{meshz} - meshplot with zero plane

\text{surf}(X, Y, Z) - surface plot: surface between the mesh points is filled in; \text{surf}(x, y, Z) and \text{surf}(Z) also work, same is true for “mesh” command.

\[
>> Z = X.\,^4 + 3 \times X.\,^2 \times Y \times X.\,^2 + 6 \times 2 \times X - 2 \times Y \times X.\,^2 + X.\,^2 - 2 \times Y;
\]
Contour Plots etc.

Contour plots:

```matlab
>>contour(X,Y,Z,20) %Draws contour plot of f(x,y) with 20 contour lines.
>>contourf(X,Y,Z,10) %Filled contour plot with 10 contour lines.
```

```matlab
>>[X,Y,Z]=cylinder(y,N) % y(x) sets the shape of the cylinder along the x-axis, N – number of points around the cylinder surface.
```

```matlab
>>mesh(X,Y,Z) will plot the cylinder surface in 3D
```

```matlab
>>[X,Y,Z]=cylinder(1.2+sin(linspace(0, 2*pi,100)),20)
```

October-November 2002