Introduction to Matlab

Prof. Basu (basu@oxy.edu), Department of Mathematics

MATLAB is short for *matrix laboratory*, and that describes it pretty well. `MATLAB` is a software package for high performance numerical computation and visualization. `MATLAB`'s built-in functions provide excellent tools for linear algebra computations, data analysis, signal processing, optimization and many other types of scientific computations. Most of these functions use state of the art algorithms.

On this worksheet we learn the basics of `MATLAB`. Login into the classroom computers and access `MATLAB` in groups and start computing!

Lesson 1: Creating and Working with Matrices

Command Window

Simple computations may be carried out in the `Command Window` by entering an instruction at the prompt, similar to what you would do on a calculator.

symbol	operation	MATLAB form
$+$	addition	`a + b`
$-$	subtraction	`a − b`
$*$	multiplication	`a * b`
$/$	right division $\frac{a}{b}$	`a/b`
$\backslash$	left division $\frac{b}{a}$	`a\b`
$\wedge$	exponentiation	`a^b`

MATLAB Variables

Variable names in MATLAB may consist of up to 31 characters, starting with a letter and followed by any combination of letters, digits, and underscores. Variable names are case sensitive. Punctuation marks and spaces may be not be used in a variable name.

MATLAB treats all variables as matrices.

Entering Matrices

There are several ways to enter a matrix in MATLAB.

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

will make the matrix
`A =`

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

A row vector is a 1-by-n matrix and a column vector is a n-by-1 matrix. So, we could make the same matrix by first making the row vector `row1`= [1 2 3] and similarly `row2` and `row3`, and then typing

```
>> A=[ row1; row2; row3]
```

Or, you could set `column1`=[1; 4; 7] with matching `column2` and `column3`, and then

```
>> A=[column1 column2 column3]
```

Note:

1. Entries in a row are separated by spaces (or commas) and the rows are separated by semicolons. So, typing
   `>> B = [row1 row2 row3]`
   creates `B` = [1 2 3 4 5 6 7 8 9].

2. Elements in MATLAB arrays are indexed starting from 1.

Creating Special Matrices

The command:
`eye(n)` creates the $n \times n$ identity matrix
`zeros(n,m)` creates the $n \times m$ zero matrix
`rand(n,m)` creates an $n \times m$ matrix with uniformly distributed random elements between 0 and 1.

Creating Special Vectors

The command

```
>> x=1:10
```

creates a row vector of the integers from 1 to 10. The command

```
>> y=1:2:10
```

creates a row vector of the numbers from 1 to 10 with spacing by 2. You can check the size of a variable by typing in the command

```
>> size(y)
```

If you wish to have a specified number of values, evenly distributed between the first and last element and if the desired spacing is linear, then the MATLAB function `linspace` can be used. For example, the command to create a row vector of 100 points, evenly spaced from 1 to 5, inclusive is

```
>> x=linspace(1,5)
```

You can also specify the number of points (including the first and last values) with the command

```
>> x=linspace(1,5,10)
```

Accessing Elements in a Matrix

You can access individual elements in a matrix by using the subscript notation `A(i,j)`. For example for our matrix typing

```
>> A(1,2)
```

yields the element in the first row and second column of the matrix, 2.
Also, you can extract a smaller matrix from a larger one. If `H` is a 10×10 matrix, then

```
>> K=H(1:5,6:10)
```

will pull out the block from rows 1 through 5 and columns 6 through 10, inclusive making `K` a 5×5 matrix. It works the other way round too. If `K` has already been defined to be a 5×5 matrix, then

```
>> H(1:5,6:10)=K
```

will reset the upper, right hand block of `H` to be all of `K`.

The notation `A(2,:)` is a vector of the elements in the second row of every column of `A`, i.e., the second row of `A`. Similarly, typing `A(:,3)`, creates a column vector, that is the third column of `A`.

Recall that for a matrix `A`

- `A(i,j)` refers to the element in the ith row and jth column
- `A(:,5:20)` refers to the elements in columns 5 through 20 of all the rows of the matrix `A`.
- `A(2,:)` refers to the second row of every column of `A`
- Empty square brackets with no elements between them create a null matrix: `[]`
- the command `A(:)` stacks the columns of `A` into a single column
- the symbol `'` transposes
- the command `round(x)` rounds x to the nearest integer.

Matrix Operations and Functions

Transpose: The apostrophe transposes a matrix.
`>> B = A'`; makes a matrix called `B` that is equal to `A` transpose. The semicolon at the end of a statement tells the MATLAB not to output the answer to the screen.

+ and - : `>> D=A+B-C` is self explanatory; just be sure to remember that matrices being added or subtracted must all have the same dimensions. However, matrix+scalar is also defined with

```
>> D=A+3
```

being used to add three to every element in `A`. Also, you can overwrite `A` with the new `A` like

```
>> A=A+3
```

Multiplication: You can multiply two matrices (provided their dimensions allow it) with

```
>> C=A*B
```

Multiplication by a scalar is also defined.

```
>>C=pi*A
```

Determinant: `det(A)` returns the determinant of the matrix.

Inverse: `inv(A)` returns the inverse of the matrix, provided that it actually has one.

Division: There are two operations, \ and /.

To solve the matrix equation $AX = B$, use

```
>> X=A\B
```

In this case, `X` is equal to `inv(A)*B`; however MATLAB does not actually compute the inverse but uses a faster algorithm.
To solve the matrix equation $XA = B$, use

```
>> X=B/A
```

This corresponds to `X=B*inv(A)`.
You can also use / for matrix-scalar division.

```
>> A=A/4
```

Powers: If `A` is square and `p` is an integer greater than one, you can raise `A` to the `p`-th power like this:

```
>> B=A^p
```

Eigenvalues: `eig(A)` returns a vector containing the eigenvalues of `A`, which may be complex-valued.

Element by Element functions: Several library functions operate on an element-by-element basis. For example, `B=sin(A)` returns a matrix the same size as `A` with `B(i,j)=sin(A(i,j))`.
Other functions like this include:
`cos`–cosine, `tan`–tangent
`asine`–arcsine, `acos`–arccos, `atan`–arctangent
`sinh`–hyperbolic sine, `cosh`–hyperbolic cosine, `tanh`–hyperbolic tangent
`asinh`–hyperbolic arcsine, `acosh`–hyperbolic arccosine, `atanh`–hyperbolic arctangent
`sqrt`–square root, `exp`–exponential base e, `log`–natural logarithm, `abs`–absolute value

Putting a period . before *, / , \, $\wedge$ makes these operators act element-by element. For example, if $u = [1\ 2\ 3]$ and $v = [4\ 5\ 6]$, then

```
>> u.*v=[4 10 18]
>> u.^v=[1 32 729]
```

Commands for Managing the Work Session

command	description
`clc`	clears the command window
`clear`	removes all variables from memory
`clear var1 var2`	removes `var1` and `var2` from memory

M-files

There are two types of programs (known as `m-files` in MATLAB: scripts and functions. To generate an `m-file`, choose `New / m-file` from the `File` menu, enter the desired commands, and save the file. The differences between scripts and functions are summarized next.

Scripts: Scripts provide a set of MATLAB commands, comments definitions of parameter values, plotting commands, and so on. A script that has been created and saved is executed by typing the file name at the MATLAB prompt in the `Command Window`, or using `save and run` or `run` from the `Debug` menu.

Functions: A MATLAB function communicates with the MATLAB workspace through the variables passed into the function, the output variables it creates, and the use of global variables. A function is distinguished from a script by the fact that the first line of a function is of the form

```
function output=functionname(input)
...
end
```

The name of the function and the name of the file in which it is stored must be the same, except that the name of the file ends in ".m".

Now let's look at a sample code.

Define a function in a file named `average.m` that accepts an input vector, calculates the average of the values, and returns a single result.

```
function y = average(x)
if ~isvector(x)
error('Input must be a vector')
end
y = sum(x)/length(x);
end
```

In the `Command Window` we can now specify a vector,

```
>> z=1:99;
>> average(z)
ans=
          50
```

Note:

1. isvector(A) returns logical 1 (true) if size(A) returns $1 \times n$ or $n \times 1$ with a nonnegative integer value n, and logical 0 (false) otherwise.
2. $\sim$ means 'not'

Activity

Try executing the following steps in the `command window`:

1. Create a 3×3 identity matrix: `eye(3)`
2. Create a matrix `B` using submatrices made up of elementary matrices of ones, zeros and the identity matrix of the specified sizes:

   ```
   B=[ones(3) zeros(3,2); zeros(2,3) 4*eye(2)]
   ```
3. Write a command to pull out the diagonal of `B` in a row vector: `diag(B)'`
4. Create vectors `d, d1` and `d2` of length 4, 3, and 2 respectively:

   ```
   d=[2 4 6 8]; d1=[-3 -3 -3]; d2=[-1 -1];
   ```
5. Create a matrix `D` by putting `d` on the main diagonal, `d1` on the first upper diagonal and `d2` on the second lower diagonal:

   ```
   D=diag(d)+diag(d1,1)+diag(d2,-2)
   ```

Lesson 2: A Basic MATLAB Session

Launch MATLAB, do some simple calculations and quit. Learn to add, multiply, and exponentiate numbers; use trigonometric functions and control screen output. The problems below are just for practice not for submission.

Arithmetic Operations

Compute the following quantities in MATLAB. Note that the mathematical quantity $\sqrt{x}$ and π are calculated with `sqrt(x)` and `pi` in MATLAB.

1. $\frac{2^5}{2^5-1}$ ANS: 1.0323
2. $3\frac{\sqrt{5}-1}{(\sqrt{5}+1)^2}-1$ ANS: -0.6459
3. Compute the area of a circle whose radius is given by $r = \pi^{1/3} - 1$. ANS: 0.6781

Exponentials and Logarithms

The mathematical quantities e^x, $\ln x$, and $\log x$ are calculated with `exp(x)`, `log(x)`, and `log10(x)`, respectively. Calculate the following quantities:

1. e^3, $\ln(e^3)$, $\log(e^3)$, $\log(10^5)$ ANS: $20.0855, 3, 1.3029, 5$
2. $e^{\pi\sqrt{163}}$ ANS: $2.6254e+17$

Trigonometry

The basic MATLAB trigonometric functions are `sin`, `cos`, `tan`, `sec`, and `csc`. The inverses are calculated with `asin`, `acos` etc. Calculate the following quantities:

1. $\sin\frac{\pi}{6}$, $\cos\pi$, $\tan\frac{\pi}{2}$ ANS: $0.5000, -1, 1.6331e+16$
2. $\sin^2\frac{\pi}{6}+\cos^2\frac{\pi}{6}$ ANS: 1

Complex Numbers

MATLB recognizes the letters `i` and `j` as the imaginary number $\sqrt{-1}$. A complex number may be input as `2+5*i` in MATLAB. Compute the following quantities:

1. $\frac{1+3i}{1-3i}$ ANS: $-0.8000+0.6000i$
2. $e^{i\frac{\pi}{4}}$ ANS: $0.7071+0.7071i$

Format

The floating point output display is controlled by the `format` command.Try the following commands and see what they do: `format short e` and `format long`.

MATLAB Lesson 3: Relational and Logical Operations

The problems presented in this section are for you to try out. Not to be turned in.

Relational Operators

There are six relational operators in `MATLAB`.
`<` less than
`<=` less than or equal
`>` greater than
`>=` greater than or equal
`==` equal
`∼=` not equal
These operations result in a vector or matrix of the same size as the operands, with 1 where the relation is true and 0 where it is false. For example:
`x=[1 5 3 7]; y=[0 2 8 7];`
`k=x<=y` results in `>>k=[0 0 1 1]` because $x_i \leq y_i$ for $i = 3$ and 4.

Activity

What do the following operations result in:

```
k=x<y
k=x>y
k=x>=y
k=x==y
k=x∼=y
```

Logical Operators

There are four logical operators:
`&` logical AND
`|` logical OR
∼ logical complement (NOT)
`xor` exclusive OR
These operators work in a similar way as the relational operators and produce vectors or matrices of the same size as the operands, with 1 where the condition is true and 0 where false.

Activity

Consider two vectors `x=[0 5 3 7]:` and `y=[0 2 8 7];`. The following operations result in:

```
m=(x>y)&(x>4)
n=x|y
m=∼(x|y)
p=xor(x,y)
```

Matlab Lesson 4: Loops, Branches and Control Flow

`MATLAB` has its own syntax for control flow statements such as `for` loops and `if-elseif-else` branching. It also provides three commands `break, error` and `return`. A description of these functions follows.

For Loops

A `for` loop is used to repeat a statement or a group of statements for a fixed number of times. Nested `for` loops are `for` loops within for loops. Every `for` must be matched with an `end`. Here are two examples:

Example 1
```
for i=1:100
num=1/(i+1);
end
```

Example 2
```
for n=100:-2:0
k=1/(exp(n);
end
```
Note that in the second example `n` goes from 100 to 0 as 100, 98, 96, etc.

While Loops

A `while` loop is used to execute a statement or a group of statements for an indefinite number of item until the condition specified by `while` is no longer satisfied. For example:

```
% let us find all powers of 2 below 10000
v=1; num=1; i=1;
while num<10000
v=[v;num]; i=i+1
num=2^i;
end
v % display v
```
A `while` must have a matching `end`.

If-elseif-else Statements

This construction provides a logical branching for computations.

```
i=6; j=21;
if i>5
   k=i;
elseif (i>1) & (j==20)
   k=5*i+j;
else
```

```
   k=1;
end
```

Break

The command `break` inside a `for` or `while` loop terminates the execution of the loop, even if the condition for execution of the loop is true. For the following example assume that the variable `v` is predefined.

```
for i=1:length(v)
   if v(i)<0 % check for negative v
      break % terminate loop execution
   end
   a=a+v(i); % do something
end
```

Error

The command `error(‘message’)` inside a function or a script aborts the execution, displays the error message `message`, and returns the control to the keyboard.The following code has been executed in a function file.

```
function c=crossprod(a,b)
% crossprod(a,b) calculates the cross product axb
if nargin =2 % if not two input arguments
   error(‘Sorry, need two input vectors’)
end
if length(a)==2 % begin calculations
...
end
```

Matlab Lesson 5: Graphics

Use the `plot` function to create a 2D plot of a set of data. If `y` is a vector, then `plot y` will plot each element of `y` against its index on the x-axis. If `x` and `y` are vectors of the same length, then `plot (x,y)` will create a plot of `x` versus `y`. The following labeling functions may be helpful:

```
title( 'your title')
xlabel('your label')
ylabel('your label')
```

that add a title above the plot, and labels the x-axis and y-axis. By default, `plot(x,y)` will connect the points with straight blue lines. A third argument, a character string (enclosed in single quotes), may be added to format the plot. Use the table below to make your preferred string by combining one color symbol with one line style symbol.

symbol	color	symbol	line style
`y`	yellow	.	*point*
`m`	magenta	$\circ$	*circle*
`c`	cyan	`x`	$x - mark$
`g`	green	$+$	*plus*
`b`	blue	$*$	*star*
`w`	white	:	*dotted*
`k`	black	$-.$	$dash - dot$
`r`	red	$--$	*dashed*

For example, the `plot(x,y,'r+')` will put red pluses at each point of the data, and `plot(x,y,'g-.')` will connect the points with a green dash-dot-dash-dot line.

You can plot more that one set if data on the same coordinate axes. If `z` is another vector of the same length as `x` and `y`, `plot(x,y,':', x,z,'--')` will create a plot with dotted line of x versus y and a dashed line of x versus z. You can add an additional plot to an existing one by using the `hold` command. The last plot can also be made with these commands

```
>> plot(x,y,':')
>> hold on
>> plot(z,y,'--')
>> hold off
```

Example

Suppose we want to plot three sine curves with a small phase shift between each line, namely $\sin(x), \sin(x - 0.25), \sin(x - 0.5)$. If we want a green line with no markers for the first sine curve, a blue dashed line with circle markers for the second sine curve and only cyan star markers for the third sine curve the we could write the following code in `MATLAB`:

```
x = 0:pi/10:2*pi;
y1 = sin(x);
y2 = sin(x-0.25);
y3 = sin(x-0.5);
plot(x,y1,'g',x,y2,'b--o',x,y3,'c*')
```

Part of Homework

1. **Equation of a Straight Line** Compute the y coordinates of a line with slope $m = 0.5$ and the intercept $b = -2$ at the following x coordinates:
   $x = 0, 1.5, 3, 4, 5, 7, 9$ and 10. ANS: $y = [-2.0000, -1.2500, -0.5000, 0, 0.5000, 1.5000, 2.5000, 3.0000]$

2. **Multiply, Divide and Exponentiate Vectors** Create a vector t with 10 elements $1, 2, \cdots 10$. Now compute the following quantities

   (a) $x = t\sin(t)$

   (b) $y = \dfrac{t-1}{t+1}$

   (c) $z = \dfrac{\sin(t^2)}{t^2}$

3. **Points on a Circle** All points with coordinates $x = r\cos\theta$ and $y = r\sin\theta$, where r is a constant, lie on a circle with radius r, that is they satisfy the equation $x^2 + y^2 = r^2$. Create a column vector for θ with the values $0, \frac{pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}$. Take $r = 2$ and compute the column vectors x and y. Also check that x and y indeed satisfy the equation of a circle by computing the radius $r = \sqrt{(x^2 + y^2)}$.

4. **The Geometric Series** By this time you know how to compute x^n element by element for a vector x and a scalar component n. How about computing n^x, and what does it mean? The result, again, is a vector with elements n^{x_1}, n^{x_2}, etc.

   The sum of a geometric series $1 + r + r^2 + r^4 + \cdots + r^n$ approaches the limit $\dfrac{1}{1-r}$ for $r < 1$ as $n \to \infty$.

   Create a vector n of 11 elements 0 to 10. Take $r = 0.5$ and create another vector $x = [r^0 r^1 r^2, \cdots r^n]$ with the term by term exponentiation command. Now take the sum of this vector with the command `s=sum(x)`. (`s` is the sum of the actual series.) Calculate the limit $\dfrac{1}{1-r}$ and compare the computed sum `s`. Repeat the procedure taking n from 0 to 50 and then from 0 to 100.

5. **Manipulating Matrices**

   (a) Create a 4×3 random matrix A.

   (b) Get those elements of A that are located in rows 3 to 4 and columns 2 to 3.

   (c) Add a fourth column to A and set it equal to the first column of A.

   (d) Replace the last 3×3 submatrix of A(rows 2 to 4, columns 2 to 4) by a 3×3 identity matrix.

   (e) Delete the first and third rows of A.

   (f) Round off all entries of A towards the nearest integer.

   (g) String out all elements of A in a row.

6. **Matrices and Vectors**

   (a) Create a vector and a matrix with the following commands: `v=0:0.2:12;` and `M=[sin(v); cos(v)];`. Find the size of `v` and `M` using the `size` command.

   (b) Extract the first 10 elements of each row of the matrix and display them as column vectors.

7. **The Roots of a Quadratic Equation** Write a MATLAB code that computes the **real roots** of a quadratic equation.

8. **Matrix Multiplication** Write a MATLAB code that computes the product $C = A \cdot B$, where A is a $m \times p$ matrix and B is a $p \times n$ matrix using for loops. After writing your code analyze it to see if you can make it more efficient. Remember that writing a code that works isn't enough. The efficiency of your code is just as important as getting the correct answer. (Do NOT use MATLABs built-in function `C=A*B` to do this exercise.)

9. **Plotting** Define `x` as a vector of linearly spaced values between 0 and 2π. Use an increment of $\pi/100$ between the values. Define y as sine values of x. Plot the graph of sine in MATLAB. Now, on the same graph plot cosine. Be sure to provide your graph with a title, label axes and a legend.