Introduction to Matlab

November 22, 2013

Contents

1 Introduction to Matlab

1.1 What is Matlab

The software program called *Matlab* (short for MATrix LABoratory) is arguably the world standard for engineering- mainly because of its ability to do very quick prototyping. It was

originally introduced as a front end to do linear algebra, but since then has gotten quite user friendly for those with little background in higher mathematics.

1.2 Matlab versus Maple

Maple (and Mathematica, and WolframAlpha) are designed primarily as "computer algebra systems" or CAS. These systems have the ability to manipulate expressions symbolically. Matlab is designed to work numerically.

1.3 Getting Started

Matlab's start button is a three-dimensional surface plot. If you don't see it, go to the top left button, and search for Matlab. Press the icon.

One window with several compartments will open. To the left, you can see the contents of the current folder. In the middle is the main window- The Command Window (we'll type commands here). To the right is a Workspace window (keeps track of all your variables) and below that is a Command History (so you can keep track of what you've done).

1.4 Basic Arithmetic and Assignment

We can use Matlab "live" by typing commands in the Command Window. Here are some to get you started- Think about what you're doing as you type these in, and see if the Matlab output is what you expect.

```
x=pi
y=3*sin(x/2)
w=2^3
z=y+exp(-y)
log(z)
z=(1+3*i)*(5-2*i)
help log
whos
```

Things to notice (especially as a comparison to Maple):

- Matlab uses `=` for assignment (in Maple, this was `:=`). For example, $w = 2$ is defined, while $2 = w$ is not.
- Matlab uses small case π (in Maple, the constant is `Pi`).
- In both Matlab and Maple, the exponential function is `exp(x)` rather than e^x.
- Use the up-arrow key to have Matlab give you a previously typed command. For example, re-do the line `z=y+exp(-y)`, but add a semi-colon at the end. What happens?

- Maple uses a colon to suppress output, Matlab uses a semi-colon (see the previous question).

1.5 Clearing the memory and the command window

To clear the memory and the screen:

```
clear
clc
```

Notice that this does NOT clear your Command History.

2 Arrays and Manipulating Arrays

The way numerical data is typically stored is in a matrix or vector, just as in linear algebra. For example, here is a vector $\mathbf{x}$ and matrix A:

$$\mathbf{x} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 9 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 2 \end{bmatrix}$$

In Matlab, these are entered as:

```
x=[1;-1;0;0];
A=[1 2 3;-1 0 2];
```

In each case, the semicolon inside the array denotes the end of a row (so that $\mathbf{x}$ is a column instead of a row). The semicolon at the end of the line suppresses printing (try leaving it off).

To access elements of the array A, use parentheses around the row and column (Maple uses square brackets). For example, $A(1,3)$ is the number 3, and $A(2,2)$ is 0.

Alternatively, the matrix A could have been entered as the following. We can ask Matlab for the size of the array as well:

```
A=[1 2 3
-1 0 2];
[numrows, numcols]=size(A)
A(2,1)=16
```

For the second command, we ask for the size of the matrix A, and note that there are two outputs for the function- The number of rows is the first output, number of columns is the second output.

In the next line, we show how to change elements of a matrix- In this case, we changed the $(2,1)$ entry to 16.

2.1 Special Commands: The colon operator

There are special arrays that are built-in to Matlab, and some use special notation. The symbol : is used a lot. Here are a couple of examples, then we'll give the general case:

```
x=2:9
x=8:-2:1
x=2:3:10
```

So what does the general command `a:b:c` do?

- `a:b`

  Produces a row vector using the numbers $a, a+1, a+2$, etc., where the last element is less than or equal to b.

- `a:b:c`

  Produces the numbers from a to c by adding b each time. The last element is the largest that is less than or equal to c.

2.1.1 Questions:

1. What will Matlab do when you type: `a=0.5:4`? (Answer before typing it in!)
2. What will Matlab do when you type: `b=0:0.3:0.7`? (Answer before typing it in!)

2.2 Matlab commands associated with Arrays

- `linspace(a,b,c)`

  Produces c numbers evenly spaced from the number a to the number b (inclusive). For example, `x=linspace(2,3.5,40)` produces 40 numbers evenly spaced beginning with 2 and ending with 3.5.

  SHORTCUT: Leaving off the third number c will give you 100 numbers between a and b (That is, $c = 100$ is the default value.)

  Compare this with the colon operator. We would use the colon operator if we want to define the length between numbers, and use `linspace` if we want to define the endpoints.

- Random arrays (handy if you just need some quick data!)

  `A=rand(m,n)` Produces an $m \times n$ array of random numbers (uniformly distributed) between 0 and 1. If you just want a single random number between 0 and 1, just type `rand`

  `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance. If you want a single random number (with a normal distribution), just type `randn`

- `A=zeros(m,n)` Produces an $m \times n$ array of zeros.
- `A=ones(m,n)` Produces an $m \times n$ array of ones.
- `A=eye(n)` Produces an $n \times n$ identity matrix.
- `A=repmat(B,m,n)` Matrix A is constructed from matrix (or vector) B by replicating B m times down and n times across.

  Example: Let $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then `A=repmat(B,2,3)` creates the array:

```
A =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

2.3 Some Operations on Arrays

Linear Algebra

Normal linear algebra functions are observed. Below we can computer vector or matrix addition, matrix-matrix multiplication and matrix-vector multiplication are performed using the multiplication symbol: `*`. Examples:

- You can add arrays of equal size (each has to have the same number of rows and columns). The result is performed elementwise- For example,

```
A=[1 2 3;4 5 6;7,8,9];
B=eye(3);
C=A+B
```

  The result C adds 1 to each diagonal element of A.

- Matrix multiplication is performed by typing `A*B`

An important operator not in linear algebra: The dotted operator

In Matlab, if the dot appears before an operation, it means to do that operation to the matrix element-wise. Here are some examples:

- To raise each element to the same power, use `.^` For example, to square every element in A, type:

```
A.^2
```

- You can multiply elementwise using `.*` For example,

  ```
  C=A.*B
  ```

  The answer is zero everywhere except the diagonal, which has $1, 5, 6$ as entries.

- Most other built-in functions act on array element by element- Here are a couple of examples:

  ```
  sin(A)
  exp(A)
  ```

  Side Remark: The matrix exponential is actually defined as a special type of matrix- There is a separate Matlab command for that. We will only work with it element-wise.

Other operations on matrices

- You can "flip" an array by making the first row the new first column (and so on). This is called **transposition**, and is performed in Matlab using an apostrophe. For example,

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

- We can take the dot product between vectors:

  ```
  x=[1 2 3 4];
  y=[5 6 7 8];
  C=dot(x,y)
  ```

  Of course, we can also write: `x'*y` as well, since that is how the dot product is defined.

2.4 Other Linear Algebra Operations

- `det(A)` is the determinant of matrix A.
- `[V,D]=eig(A)` returns the eigenvectors of A in V, and the eigenvalues in D.
- `X=linsolve(A,B)` solves the linear system $AX = B$ for X. Notice that X, B could be matrices.

2.5 Accessing Submatrices

Let A be an $m \times n$ array of numbers. Then:

The notation:	Yields:
`A(i,j)`	The (i,j)th element
`A(i,:)`	The entire ith row
`A(:,j)`	The entire jth column
`A(:,2:5)`	The 2d to fifth columns, all rows
`A(1:4,2:3)`	A 4×2 submatrix

Example: What kind of an array would the following command produce?

```
A([1,3,6],[2,5])
```

A 3×2 matrix consisting of the elements:

$$\begin{matrix} A(1,2) & A(1,5) \\ A(3,2) & A(3,5) \\ A(6,2) & A(6,5) \end{matrix}$$

Example: Create a 5×5 zero array, and change it to:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Answer:

```
A=zeros(5,5);  %Create the matrix of zeros
b=[1 2 3;4 5 6; 7 8 9];
A(2:4,2:4)=b;
```

Note also the use of the `%` sign. It is used to denote *comments*; that is, Matlab would ignore everything on the same line after the `%` sign.

2.5.1 Adding/Deleting Columns and Rows:

Its straightforward to insert and/or delete rows and columns into a matrix. Before doing it, we define `[ ]` as "the empty array": the array with nothing in it.

In the following, let A be a 4×5 array, let b be a 1×5 row, and c be a 4×1 column.

Examples of use (each of these are independent from the previous):

The command:	Produces:
`A(1,:)=[];`	Delete the first row.
`A([1,3],:)=[];`	Delete rows 1 and 3.
`A(:,3)=[];`	Delete column 3.
`A(:,1:2:5)=[];`	Delete the odd columns.
`A(1,:)=b;`	Put (row vector) b as row 1.
`A(:,6)=c;`	Add (column vector) c as the last column.
`d=[c , A(:,1:3)];`	d is c and columns $1-3$ of A.
`A=[A(:,1), c, A(:,2:5)];`	Insert c as column 2 of A, others shift 1 over.
`A=[A(1,:); b; A(2:4,:)];`	Insert b as row 2 of A, others shifted 1 down.

Example: Matlab comes with some built-in data sets. One such set is the image of a clown. For fun, we'll load the array in, display it, then we'll remove all of the even rows and columns, then re-display it:

```
load clown
whos
image(X);
colormap(map);
X(2:2:200,:)=[];
X(:,2:2:320)=[];
image(X);
```

Once you're done, you may want to clear the memory and the screen:

```
clear
clc
```

Now, suppose we want fill out the matrix again, but now replace the rows and columns we had before with zeros. Here is one way you might do that. The last line prints out the first few numbers so you can see what we did.

```
Y=zeros(200,320):
Y(2:2:200,2:2:320)=X;
image(Y);
Y(1:20,1:10)
```

3 Introduction to Graphics

Here's a quick example to get us started:

```
x=linspace(-pi,3*pi,200);
y=sin(x);
plot(x,y);
```

You'll see that we had to create a domain array and a range array for the function. We then plot the arrays. For example,

```
plot([1,2],[3,4]);
```

will plot a line segment between the points $(1, 3)$ and $(2, 4)$. So, Matlab's plotting feature is drawing small line segments between data points in the plane.

3.1 Examples

1. Matlab can also plot multiple functions on one graph. For example:

   ```
   x1=linspace(-2,2);
   y1=sin(x1);
   y2=x1.^2;
   x2=linspace(-2,1);
   y3=exp(x2);
   plot(x1,y1,x1,y2,x2,y3);
   ```

   produces a single plot with all three functions.

2. `plot(x1,y1,'*-');`

   Plots the function `y1`, and also plots the symbol `*` where the data points are.

3. `plot(x1,y1,'k*-',x2,y2,'r^-');`

   Plots the function `y1` using a black (k) line with the asterisk at each data point, PLUS plots the function `y2` using a red line with red triangles at each data point.

   The following lists all of the built in colors and symbols that Matlab can use in plotting: (NOTE: You can see this list anytime in Matlab by typing: `help plot`)

Code	Color	Symbol	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	−	solid
b	blue	*	star
w	white	:	dotted
k	black	−.	dashdot
		−−	dashed

4. The following sequence of commands also puts on a legend, a title, and relabels the $x-$ and $y-$axes: Try it!

```
x=linspace(-2,2);
y1=sin(x);
y2=x.^2;
plot(x,y1,'g*-',x,y2,'k-.');
title('Example One');
legend('The Sine Function','A Quadratic');
xlabel('Dollars');
ylabel('Sense');
```

5. Other Things: If you look at the plotting window from the last example, you'll see lots of things that you can do. For example, there's a zoom in and a zoom out feature. You can also edit the colors and symbols of your plot, and the title, legend and axis labels. Try them out!

6. For more on plots, in the command window type `doc plot`

4 M-Files: Functions and Scripts

Both functions and scripts are text files that are saved with a `.m` suffix. Scripts are files that simply contain a set of Matlab commands that one wishes to execute. Functions (like mathematical functions) take in some kind of input, does something to that input, and produces an output.

The variables that are produced by a script stay in memory. Functions will produce what are called local variables, in that once the function is finished, those local variables will be gone.

To create either a function or a script, Matlab's built-in editor has some nice features. Type `edit` in the command window to bring up the editor.

Let's look at some examples:

4.1 Scripts

Again, scripts (or script files) are text files with a set of commands for Matlab to execute as if you were typing them in live at the keyboard. If you know Maple, then Matlab scripts take the place of Maple worksheets.

- Example: Open the editor and then type and save the following script as `Script01.m` (be sure it has the .m suffix). The lines that begin with the percent sign are comments (you should be sure to put in comments!)

```
% Script file that performs Newton's Method

f=inline('x-exp(-x)');  df=inline('1+exp(-x)');

x(1)=-1;

for j=1:100
  y(j)=f(x(j));
  dy(j)=df(x(j));
  x(j+1)=x(j)-y(j)/dy(j);

  if abs(y(j))<10^(-6)
     break;
  end

end
```

  To run the script, in the command window, type `Script01` (without the .m suffix). In the workspace window, you should see that all of the variables defined here stay in memory after the script is finished.

- A little more complicated example: A script file can be "published" and printed, and can include plots and things. Here's a longer example (on our class website as well):

```
%% Homework Set 9

%% Problem 1: Script file that performs Newton's Method

f=inline('x-exp(-x)');  df=inline('1+exp(-x)');

x(1)=-1;

for j=1:100
```

```
  y(j)=f(x(j));
  dy(j)=df(x(j));
  x(j+1)=x(j)-y(j)/dy(j);

  if abs(y(j))<10^(-6)
     break;
  end

end

N=length(x);
plot(1:N,x,'^-',1:N-1,y,'*-');

%% Problem 2:  Find the line of best fit using the data below
clear
x=[-1,1,2,3]';  y=[-1,1,1,2]';
A=[x, ones(4,1)];
c=linsolve(A,y)
t=linspace(-1,3);
z=c(1)*t+c(2);
plot(x,y,'*',t,z);
```

Now, save as "Script02.m", and in the editor under `File`, you'll see `Publish Script02.m`. Choose that, and a nicely formatted solution should come up in a web browser.

4.2 Functions

As we know, functions are rules that change input values to output values. In programming, a function can have multiple inputs and multiple outputs.

For example, suppose I want to write a program that will tell me the cost of producing a cylindrical can. The inputs might be the radius of the can, r, the height of the can, h, and the costs of the material- We might have one cost for the top and bottom, and a different cost for the sides. We'll call these C_t, C_s, respectively.

Algebraically, the cost will be:

$$C = C_t(2\pi r^2) + C_s(2\pi rh)$$

I may also want to know the surface area, so I'll have the function output that, too:

$$A = 2\pi r^2 + 2\pi rh$$

Here is how I would write this in Matlab:

```
function [C,A]=canFunction(r,h,Ct,Cs)
% function [C,A]=canFunction(r,h,Ct,Cs)
%  Computes the cost C and surface area A of a can.
%  Input:  radius r, height h, Ct, Cs are costs of
%       top/bottom and sides.
%  Output:  Cost and Surface Area (in that order)

TopBottom=2*pi*r^2;
Sides=2*pi*r*h;

C=Ct*TopBottom+Cs*Sides;
A=TopBottom+Sides;
```

Save this file as the function name with a `.m.` suffix, or, `canFunction.m`. Some things to notice about a function:

- The first line should always begin with the word "function". This is how Matlab distinguishes between a script and a function.
- You should always include remarks that tell you how to use the function. (what are the inputs/what are the outputs?)

Now in the command window, we can type things like:

```
help canFunction
[C,A]=canFunction(3,6,10,15);
```

You should notice that when the function is called, only the output variable names are present- that is, the variables `TopBottom` and `Sides` that the function uses are only present for the function itself (these are called "local variables" in computer programming).

We will write some functions, but we will probably stick mostly with scripts.