# B

# Matlab

## B.1 Matlab basics

This section contains some basics of Matlab. Matlab has a lot of powerful and painless toolboxes, but we will try to do most things by translating mathematics into matlab code. Though less painless, this path leads to much greater ability to develop new ideas and test them out. There is a somewhat parallel tutorial to this one on the Mathworks website.

### B.1.1 Self help

There are two ways to get help in matlab. The first is command line, which is useful if you know the name of the command but you are not sure what parameters the command takes.

```
>> help command_name;
```

The second way is to go to the menu bar at the top of the matlab window and click on help. This gives you a searchable database of information. There are a couple of things about the main matlab window that you should know. You may or may not get a few panels in the left part of the window when you start matlab. These panels include *command history*, *current directory*, and *workspace*. You can manipulate which panels are displayed using the *desktop* drop down menu.

A few words about these. The workspace contains variables that are loaded into matlab working memory. If you are working with very large data structures then you can run out of memory. So you will want to get familiar with manipulating the workspace and with the type of data *structures* (such as `*.mat` files) that matlab knows how to read automatically.

```
>> help struct;
```

For properly formatted data structures (in *.mat files) it is possible to drag and drop them into the workspace. Right clicking on a workspace variable allows you to perform certain appropriate manipulations including clearing the variable, plotting, etcetera.

The command history is a useful way of saving effort, particularly if you want to execute various modifications of a given command over and over at the command line. There are two ways that you can do this. The first is by pointing on a command in the command history and double clicking (or copying and pasting into the workspace); the second is by using the up arrow at the keyboard which toggles through all previously executed commands in order of most recent. If you type part of the command, *e.g.*

```
>> elf;
```

then hit the up arrow, you will toggle through the commands that start with the string ”elf”. You can find other shortcuts by typing ”shortcut” in the matlab help screen.

### B.1.2 Vectors and Matrices

Matlab stands for **Mat**rix **Lab**oratory. The most basic structures are matrices. Any calculus is done at the *discrete* level, although one can employ symbolic computation engines. As with any matrix, you have to specify its size.

```
>> help size;
```

You can specify the elements of a matrix manually by hand, for example,

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

produces the matrix

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

Matlab thinks of vectors as matrices with a single row or column.

```
>> size(A)
```

returns the $1 \times 2$ array $[3,3]$. In general, if $A$ is an $m \times n$ matrix then $\text{size}\,(A) = [m,n]$. Sometimes it is convenient to produce a large matrix with all zeros or all ones. For example,

```
>> Z=ones(5);
```

produces a $5 \times 5$ matrix $Z$ with all ones whereas

```
>> O=zeros(5,3);
```

produces a $5 \times 3$ matrix with all zeros. If you want to produce a row vector with all zeros then you have to do something like

```
>>N=100;
>> V=zeros(1,N);
```

Matlab also deals with higher dimensional data structures called *arrays*. For example,

```
>> A=zeros(3,3,3);
```

provides an array that can be visualized as a cube and each of the three layers contains a $3 \times 3$ matrix.

### B.1.3 Arithmetic

Matlab can perform arithmetic on arrays having the same dimensionality. In particular, addition is done componentwise. If $A, B$ are a pair of $m \times n \times k$ arrays then

```
>> A+B;
>> A-B;
```

is their componentwise sum and difference, respectively.

Multiplication in matlab is a little more interesting. One form of multiplication is componentwise multiplication. This is very useful for multiplying *discrete functions*. You can multiply arrays componentwise if they have the same dimensions.

```
>> A.*B;
>> A./B;
```

produces the componentwise product and componentwise quotient respectively. If $B$ has any zeros $A./B$ will produce a *Warning: divide by zero* statement. You can use ordinary matrix multiplication on numbers and matrices with the symbol "$*$". (You cannot multiply arrays of higher order this way). For "$*$" the usual rules of matrix multiplication apply. For example $3 * 5$ is fine because 3 and 5 are both treated at $1 \times 1$ matrices.

You can also compute powers of a square matrix. For example, if $A = [1,1;\ 1,1]$ then

```
>> A.^2
>> A^2;
```

produce $[1,1;\ 1,1]$ and $[2,2;\ 2,2]$ respectively. For more help see the matlab help for arithmetic operators.

### B.1.4 Built-in functions

Matlab has a number of built in functions. For example, you can compute 20! by

```
>>N=factorial(20)
```

Matlab will return the number $2.432902008176640e+018$ which, of course, is only an approximation. If you try

```
>>N=factorial(2.5)
```

then you will get an error, but

```
>>N=gamma(2.5)
```

returns 1.3293. The most basic functions for us, besides polynomials, are trigonometric and exponential functions. Trig functions are computed in radians.

Though it is possible to perform symbolic functions in Matlab (with a Maple kernel), it is simplest first to define a vector or array of sample points at which the function is to be evaluated, then “plug in” the sample array.

### B.1.5 Comments on graphics

By default, matlab interpolates values when it plots graphs. For example,

```
>>plot([1,3,4,2,0,1]);title('plot of a vector of points');
```

produces the plot in Figure B.1 while

```
>>plot([1,3,4,2,0,1],'*');title('plot of a vector of sample points');
```

produces the plot in Figure B.2 After rendering a plot you can edit the figure in various ways using the pull down menus in the figure window.

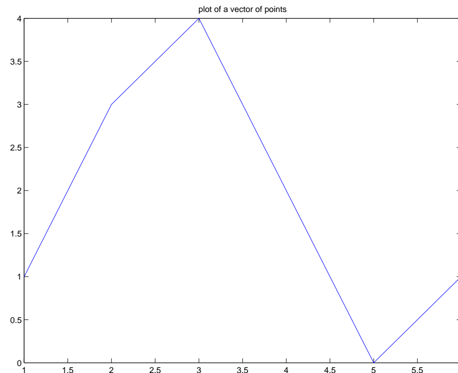


**Fig. B.1.** A plot of a vector of points.

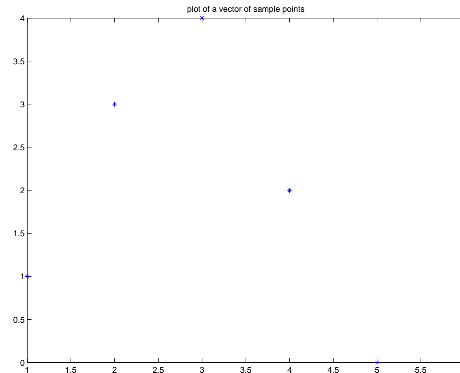


**Fig. B.2.** A plot of the corresponding sample vector.

Now let’s try to plot sample values of a function. first we have to define the samples. For example,

```
>>t=0:.5:2*pi;
>>plot(sin(t));
>>title('samples of sin x');
```

You can see that the picture is jagged as matlab linearly interpolates the samples. You can get a smoother looking estimation of the sin curve by using more samples. The interpolation is still linear, but the line segments are very short. Notice that the $x$-axis is labelled by sample index, not by the actual value of $t$.

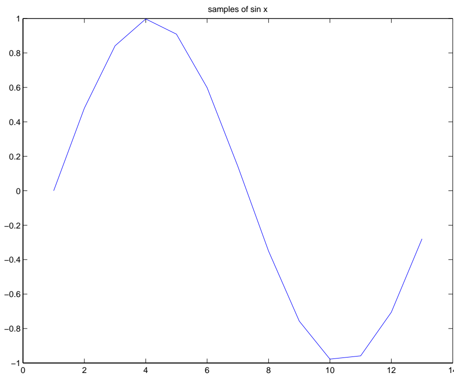


**Fig. B.3.** Interpolated samples of the sine function.

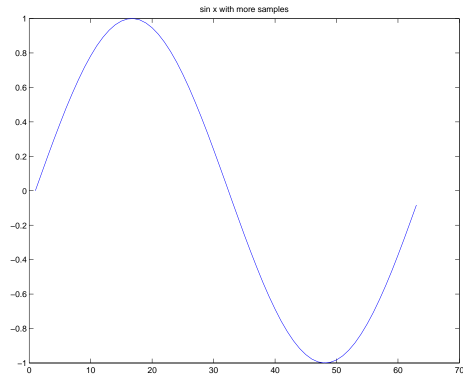


**Fig. B.4.** Interpolation of sine using more samples.

```
>>t=0:.1:2*pi;
>>plot(sin(t));
>>hold on;
>>plot(cos(t));
```

If you want to label the $x$-axis with the variable $t$ then you need to specify this as a first argument in the plot command:

```
>>hold off
>>t=0:.1:2*pi;
>>plot(t,sin(t));
```

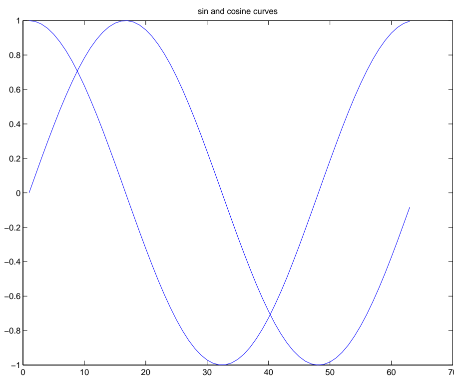


**Fig. B.5.** Interpolated sine and cosine

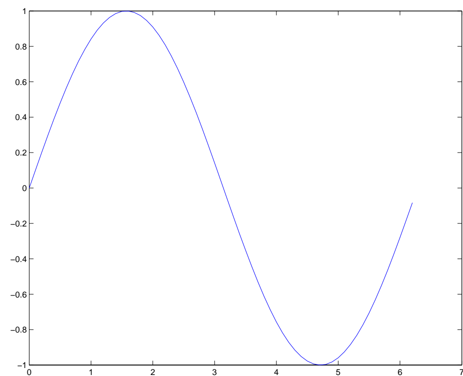

**Fig. B.6.** $\sin t$ versus $t$.

You can also plot the pairs $(\cos t, \sin t)$. Again, matlab automatically interpolates them.

```
>>plot(cos(t),sin(t));
```

### B.1.6 flow control

Can matlab compute factorials accurately? Here is an experiment.

```
>> N=100;
>> factorial(N)
>> k=1;
>> a=1;
 >> for k=1:N
```

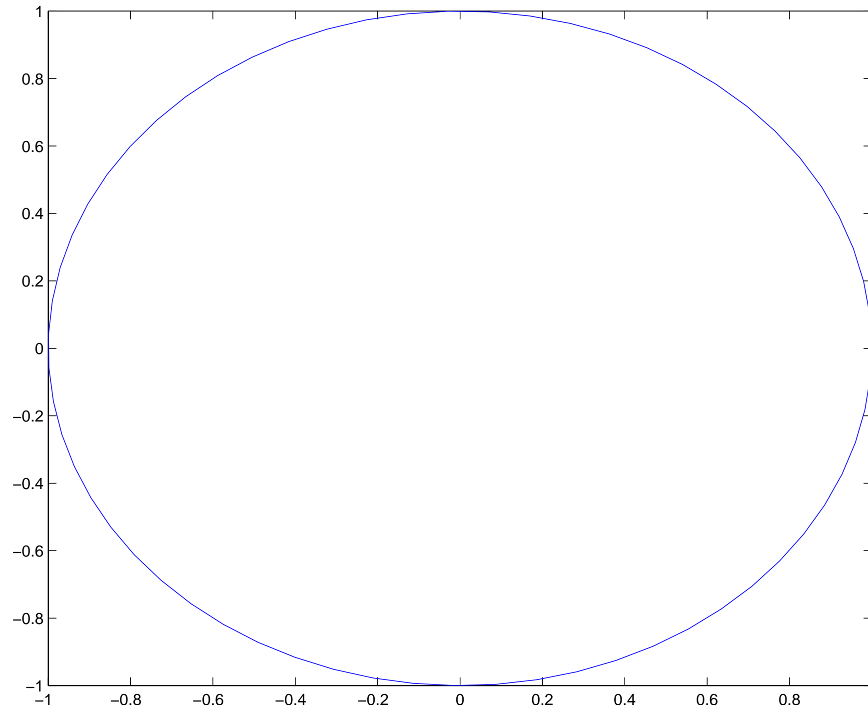


**Fig. B.7.** $\sin t$ versus $\cos t$

```
    a=k*a;
    end
>> a
```

Try larger values of $N$. What happens?

Let's consider a more analytical problem. Suppose that we wish to compute the partial sums of the series $\sum_{k=1}^{\infty} \frac{\sin \pi 4^k t}{3^k t}$. We have to be a little bit clever here because we need enough sample points in order that the summing terms make sense. For example, if we take $t$ to be the vector $[1, 2, \ldots, 15]/16$ then the vector $4 * t = [1, 2, \ldots, 15]/4$ and $4^2 * t = [1, 2, \ldots, 15]$ so that the values of $\sin(\pi * 4^2 * t)$ will all vanish. Suppose that we want partial sums up to $k = 10$. If we want nontrivial values all the way up then we should take $t = [1, \ldots, 2^{11} - 1]/2^{10}$. However, we also might want a choice that takes advantage of the periodicity of the sine function. In this case, one possible fix is to take all of the sample points to be irrational numbers. To do this, we can take, for example, $t = [1, \ldots, 2^5 - 1]/\sqrt{2}$.

Let's try a couple of these. Recursion comes into play when we add the terms of the partial sum. Here's how it works.

```
>> N=3;
>> t=0:1/4^N:1;
>> ps=zeros(1,length(t));
>> for k=1:N
term=sin(pi*4^k*t)./3^k;
ps=ps+term;
end
>>plot(t,ps);
```

Try plotting the partial sums in this manner for a few values of $N$ (like $N = 3, 5, 7$). They should look something like Figure B.8.

You should refer to Matlab help for syntax on other Matlab control loops variables including `for` and `while` loops and `if ... then ... else` loops.

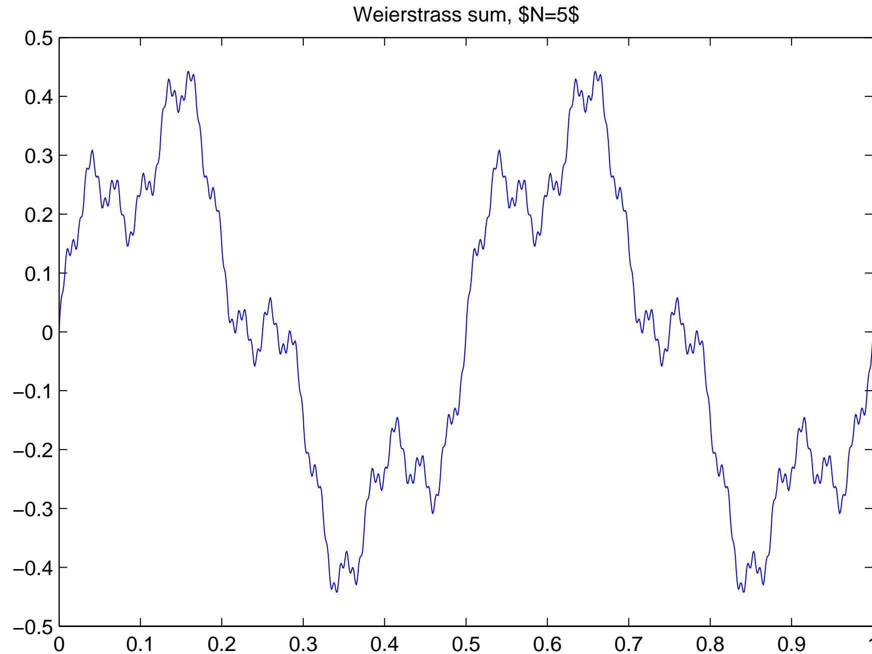


**Fig. B.8.** Partial sum ($N = 5$ of Weierstrass nowhere differentiable function

### B.1.7 Matlab scripts

If you wish to carry out a complex sequence of matlab commands that can be modified and tested then it is best to store the so-called script in an M-file. This is simply a file with a `.m` suffix. One of the benefits of a script is that you can control parameters for purposes of numerical experimentation. A second benefit is that you can have scripts call other scripts. A script that is intended to take certain inputs and produce corresponding outputs is called a *function*.

Suppose for example that we wish to investigate the behavior of the partial sums of the Weierstrass function. Let's write several commands for carrying out the script and store them in a file called `weierstrass_psum.m`. The contents are as follows:

```
function [w, wprev] =weierstrass_psum(N);

%% create a sample vector for the (N-1)-st partial sum of the Weierstrass
%% nowhere differentiable function on [0,1]
%% and compare the (N-2)-nd and (N-1)-st partial sums.

% Input: N: number of partial sums to compute
% Outputs: w: (N-1)-st partial sum vector
%           wprev: (N-2)-nd partial sum vector

t=0:1/4^N:1;
ps = zeros(1,length(t));

for k=1:N-1
    term = sin(pi*4^k*t)./3^k;
    ps = ps+term;
end
w=ps;
wprev=w-term;
plot(t,wprev);
hold on
```

```
plot(t,w,'r');
```

When calling the function you would type something like

```
>> [fine, course] =weierstrass_psum(6);
```

Then `fine` will be a vector in the matlab workspace with the values of the fifth weierstrass partial sum samples and `course` will be contain the fourth order samples. The initial lines starting with "%" signs are considered documentation lines. The symbol "%" tells matlab to ignore the remaining contents of the line. What follows % is called a "comment." The first set of commented lines are returned when one types the help command. Specifically,

```
>>help weierstrass_psum
```

returns

```
 % create a sample vector for the (N-1)-st partial sum of the Weierstrass
 % nowhere differentiable function on [0,1]
 % and compare the (N-2)-nd and (N-1)-st partial sums.
```

Only the first three lines are returned because of the intervening space between the next set of lines. It is good practice to document any scripts. In this case and in general, it would be preferable also to included discussion of the inputs and outputs in the "help" lines.

### B.1.8 Animation

Making movies in matlab is a snap. One thing to keep track of is that, when different frames of a movie are to be rendered on the same axes, one should fix the axis as can be done by specifying the range of values to be viewed at the outset, then using `hold on` to keep them fixed.

Here is a simple example that illustrates a point moving around the unit circle.

```
>> clf
>> axis([-1.1,1.1,-1.1,1.1]);
>> hold on
>> t=0;
>> for i=1:100;
    t=cat(1,t,max(t)+.063);
    plot(cos(t),sin(t));
    M(i)=getframe;
   end
>> movie(M);
```

Here, the command `cat` concatenates the next sample input to the previous list of sample inputs. Thus, each consecutive frame plots the coordinates $)\cos t, \sin t)$ with an additional sample point. The `movie` command plays back the frames sequentially. There are a few options regarding frame rate and repeated playback. In addition, it is possible to export