

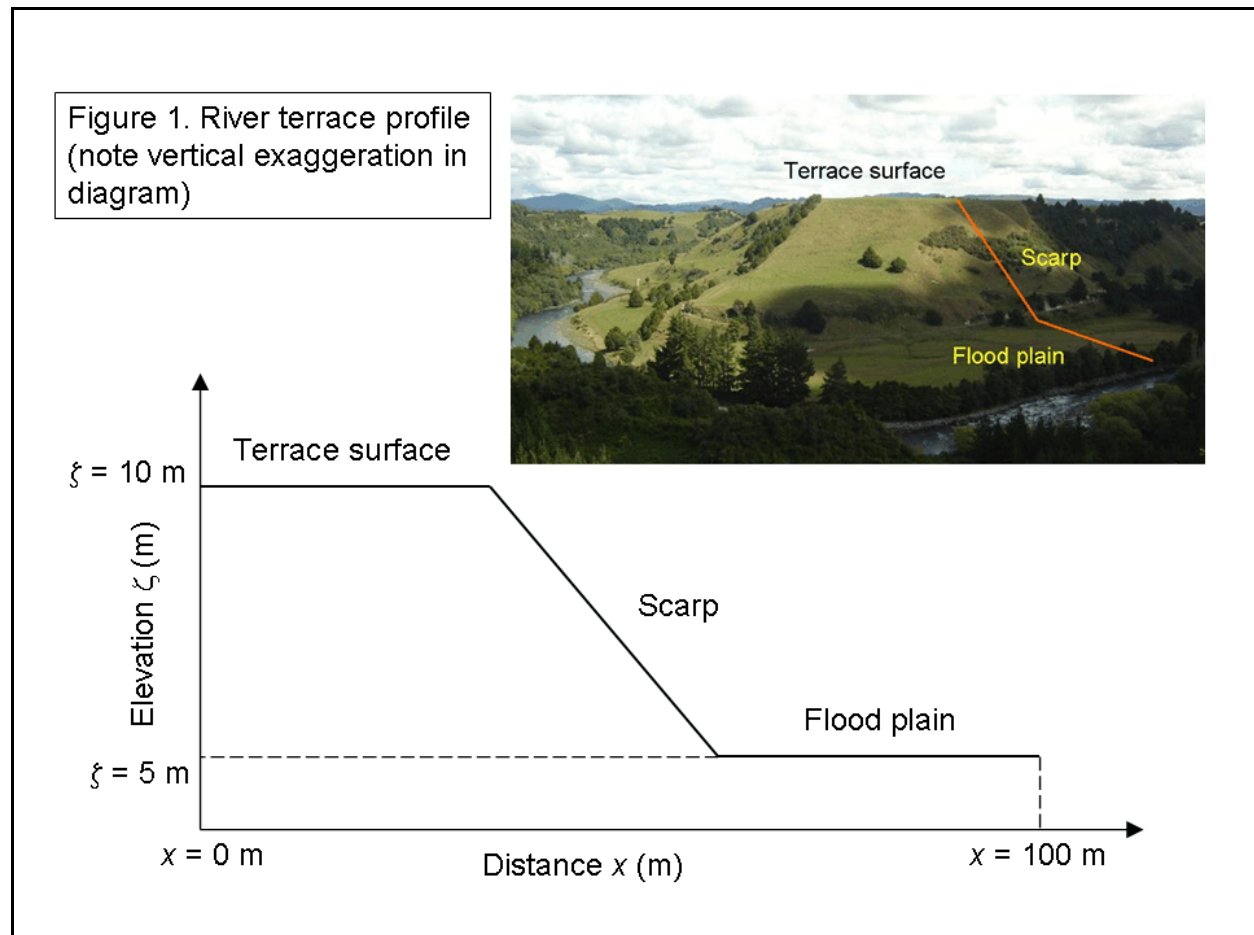
# MATLAB Exercise 1: Matrices, For Loops, and M-Files

## Getting Started in MATLAB

Before starting MATLAB, create a suitable working folder in your directory. For example, on my laptop I currently have a piddle-around folder named “MLab” with the directory path: C:\djf\MLab. We’ll help you set up this folder as needed, and you can always change it later.

Now, double click on the MATLAB 7.1 icon to open it. A main MATLAB window partitioned into three parts will appear. (Depending on who used this program last in your windows workspace, other windows might also appear, notably the MATLAB Help window. *Note: Because others will be using MATLAB on the same machine, please don’t customize any MATLAB settings yet.*) The three partitions include the Command Window, the Command History, and the Current Directory.

Go to the Current Directory drop-down mini-window at the top of the MATLAB window (not the Current Directory partition) and either type in the path for your working folder or browse for it using the “...” button. Once you’ve entered the path for your working folder in the drop-down mini-window, the Current Directory partition should refresh itself, showing no files present in your working folder. We’ll return to this later.



Click the Edit button on the upper left of the MATLAB window. On the drop-down panel, click Clear Workspace and then click OK on the Confirm Delete pop-up. We'll explain the significance of this below.

### Matrices and a River Terrace Profile

Consider the profile of a river terrace and floodplain (Figure 1). The upper part of the profile crosses the terrace surface and is flat; the central part of the profile crosses the terrace scarp; and the lower part of the profile crosses the floodplain normal to the direction of the stream channel and is flat.

By definition this profile represents the elevation of the land surface along a coordinate line  $x$ . Let's call the local elevation  $\zeta$  (the Greek letter zeta). Then, any point on the profile has the coordinates  $(x, \zeta)$ . Moreover, we may say that the elevation  $\zeta$  is a function of the coordinate position  $x$ , that is,  $\zeta = f(x)$ . Moreover, let's specify the  $x$ -coordinate position of the profile on the terrace farthest from the stream as  $x = 0$  m (the left boundary), and the  $x$ -coordinate of the floodplain at the edge of the stream as  $x = 100$  m (the right boundary). The elevation at the left boundary is  $\zeta = 10$  m, and the elevation at the right boundary is  $\zeta = 5$  m.

Our first practical task in using MATLAB is to tell it how to register (and remember) values of the coordinate  $x$  and the elevation  $\zeta$ . Later, for example, we will want MATLAB to plot a figure of the profile  $\zeta = f(x)$ . But recall from your math that the coordinate  $x$  can take on an infinite set of values between 0 and 100. Similarly, there are infinite values of the elevation  $\zeta$  between 5 and 10. Herein is a problem. Computers have limited memory. So what we will do is ask MATLAB to register (and remember) only selected values of  $x$  and  $\zeta = f(x)$ .

Let's select values spaced at intervals of 10 m along the  $x$ -coordinate axis, including both boundaries. This means a total of 11  $x$ - $\zeta$  pairs of values. There are several ways to do this in MATLAB. (In the description below, "enter" means "type and then press Enter/Return.")

In the Command Window enter

```
x = [0 10 20 30 40 50 60 70 80 90 100]
```

(Note that MATLAB is case-sensitive, so for now type things in exactly as indicated.) The result is a list of the 11 entries (elements) in this *matrix* named  $x$ . Now enter

```
x
```

The same list appears, which merely illustrates that MATLAB remembers what you placed in the matrix  $x$ .

Now enter (note the semicolon at the end)

```
z = [10 10 10 10 10 7.5 5 5 5 5 5];
```

Only the prompt ">>" shows. The effect of the semicolon is to suppress the screen printing of  $z$ . Now enter

`z`

The matrix  $z$  is listed. So far we have entered two matrices,  $x$  and  $z$ , each containing 11 elements.

Now enter

```
plot (x,z)
```

The plot that appears, besides being quick and nice, illustrates that the order of the elements in the two matrices,  $x$  and  $z$ , is important, and that MATLAB associates the first element in  $x$  (i.e. 0) with the first element in  $z$  (i.e. 10), the second element in  $x$  (10) with the second element in  $z$  (10), and so forth.

Notice that when we entered  $x$  and  $z$  above to return the lists of elements in these matrices, each of the lists appeared as a single row of 11 elements. MATLAB is treating each of these as a  $1 \times 11$  matrix (a “row vector”), that is, a matrix with one row and 11 columns.

Now enter

```
x'
```

The apostrophe mean to “transpose” the matrix  $x$ , in this case meaning to convert the  $1 \times 11$  “row matrix” into an  $11 \times 1$  “column matrix” (or “column vector”).

Now type (on the same line) and enter

```
xzdata = [0 10; 10 10; 20 10; 30 10; 40 10; 50 7.5; 60 5; 70 5; 80 5; 90 5; 100 5]
```

which is an  $11 \times 2$  matrix. Let’s “query” this matrix. Enter

```
xzdata(6,2)
```

which returns the number 7.5. That is, we asked for the element appearing in the 6<sup>th</sup> row and 2<sup>nd</sup> column. The “6” and “2” in this context are indices of the  $11 \times 2$  matrix  $xzdata$ . In general the element in row  $i$  and column  $j$  of  $xzdata$  is denoted by  $xzdata(i, j)$ . Thus,  $xzdata(6, 2) = 7.5$ ,  $xzdata(1, 1) = 0$ ,  $xzdata(11, 2) = 5$  and so forth. Notice, then, that

```
xzdata'
```

transposes the  $11 \times 2$  matrix into a  $2 \times 11$  matrix.

Now type the following four lines, pressing Enter/Return after each:

```
for i = 1:11
    xdata(i) = xzdata(i,1);
    zdata(i) = xzdata(i,2);
```

```
end
```

Then enter

```
plot(xdata,zdata,'LineWidth',2)
```

We just performed a simple version of the singularly most powerful (and important) algorithm in all of computing — a “for loop” (also known as a “do loop” in other programming languages). Let’s see exactly what it did. Here is the “code” with block comments following each line describing what the line does:

```
for i = 1:11
%{
This sets the index  $i$  to 1, with the understanding that  $i$  will successively take on the integer values 2, 3, 4, and so on to a maximum value of  $i = 11$ . Furthermore, this line, starting with the word “for,” tells the computer to perform subsequent lines until it reaches a matching “end” line.
%}
    xdata(i) = xzdata(i,1);
%{
This defines a matrix  $xdata$  and assigns to its  $i$ th index position the value of  $xzdata$  contained in its  $(i, 1)$  index position. So, for example, during the first iteration of this for loop, this line is equivalent to  $xdata(1) = xzdata(1, 1)$ . Thus,  $xdata(1)$  is assigned the value of 0.
%}
    zdata(i) = xzdata(i,2);
%{
This defines a matrix  $zdata$  and assigns to its  $i$ th index position the values of  $xzdata$  contained in its  $(i, 2)$  index position. So, for example, during the first iteration of the for loop, this line is equivalent to  $zdata(1) = xzdata(1, 2)$ . Thus,  $zdata(1)$  is assigned the value of 10.
%}
end
%{
This is the “end” line that matches the first “for” line. This “end” tells the computer to return to its matching “for” line, add 1 to  $i$  and, so long as  $i \leq 11$ , iterate the embedded operations (lines) again. So, after the first iteration ( $i = 1$ ) as described above, this “end” sends the computer back to the “for” line, and  $i$  takes on the value 2. Then, because  $2 \leq 11$ , the loop performs its task again. Now, in this second iteration ( $i = 2$ ), the second line is equivalent to  $xdata(2) = xzdata(2, 1)$  and the third line is equivalent to  $zdata(2) = xzdata(2, 2)$ . Thus,  $xdata(2)$  is assigned the value of 10, and  $zdata(2)$  is assigned the value of 10. During the third iteration ( $i = 3$ ),  $xdata(3) = xzdata(3, 1) = 20$  and  $zdata(3) = xzdata(3, 2) = 10$ ... and so on to the 11th iteration ( $i = 11$ ) wherein  $xdata(11) = xzdata(11, 1) = 100$  and  $zdata(11) = xzdata(11, 2) = 5$ . At the end of this 11th iteration, the “end” line once again sends the computer back to the matching “for” line and adds 1 to  $i$ . However, because now  $i = 12$  does not satisfy the inequality,  $i \leq 11$ , the computer knows to “jump out” of the for loop and commence with any commands that follow the loop. Cool, huh..?
%}

```

### More on Loops, and M-Files

Type the following lines, pressing Enter/Return after each:

```

Xdata = (0:2*pi/10:2*pi);
for j = 1:3
    for i = 1:11
        Zdata(i,j) = cos(Xdata(i) - 0.5*(j-1));
    end
end

```

We just performed a “nested” for loop. Notice that this algorithm created an  $11 \times 3$  matrix named *Zdata*. Before examining this further, enter the following loop whose purpose is to separate the three columns of *Zdata* into three separate matrices, *Z1*, *Z2* and *Z3*, much like we used a loop above to separate the matrix *xzdata* into two matrices, *xdata* and *zdata*.

```

for i = 1:11
    Z1(i) = Zdata(i,1);
    Z2(i) = Zdata(i,2);
    Z3(i) = Zdata(i,3);
end

```

Then enter

```

plot(Xdata,Z1,Xdata,Z2,Xdata,Z3,'LineWidth',2)

```

With this plot in place let’s have a closer look at the individual lines in the “nested” loop above.

```

Xdata = (0:2*pi/10:2*pi);
%{
This is a quick way to fill the matrix Xdata with 11 elements, the first being 0 and the last being  $2\pi = 6.2832$  (rounded off). Specifically, this command tells MATLAB to take the range from 0 to  $2\pi$  and divide it into 10 equal intervals (thus 11 total values) starting with 0 and ending with  $2\pi$ . Entering Xdata in the Command Window indeed reveals that  $Xdata(1) = 0$ ,  $Xdata(2) = 0.6283$ ,  $Xdata(3) = 1.2566$ , ...  $Xdata(11) = 6.2832$ .
%}

```

```

    for j = 1:3
%{
As in the previous example of a “for loop” this line sets the index j to a value of 1 initially, with the understanding that it will change to 2 with a second iteration, then to 3 with a third (final) iteration.
%}

```

```

        for i = 1:11
%{
This sets the index i to a value of 1 initially, with the understanding that it will increase by one with each successive iteration to a maximum value of 11. Note that the value of j remains as assigned by the previous “for” line.
%}

```

```

            Zdata(i,j) = cos(Xdata(i) - 0.5*(j-1));
%{
This line defines the matrix Zdata, and assigns to its ith row and jth column a value determined by the cosine function. So, for example, during the first iteration of the “outer” j loop (i.e.  $j = 1$ ), the

```

“inner”  $i$  loop marches through  $i = 1, 2, 3, \dots, 11$ . The line above is successively equivalent to:

```
Zdata(1, 1) = cos(Xdata(1) - 0.5*0);
Zdata(2, 1) = cos(Xdata(2) - 0.5*0);
Zdata(3, 1) = cos(Xdata(3) - 0.5*0);
:
Zdata(11, 1) = cos(Xdata(11) - 0.5*0);
```

During the second iteration of the outer  $j$  loop (i.e.  $j = 2$ ), the inner  $i$  loop again marches through  $i = 1, 2, 3, \dots, 11$ , and the line above is successively equivalent to:

```
Zdata(1, 2) = cos(Xdata(1) - 0.5*1);
Zdata(2, 2) = cos(Xdata(2) - 0.5*1);
Zdata(3, 2) = cos(Xdata(3) - 0.5*1);
:
Zdata(11, 2) = cos(Xdata(11) - 0.5*1);
```

The third iteration of the  $j$  loop completes the assignment of values in the third column of  $Zdata$ . Notice that each of the 11 elements in  $Xdata$  is used three times in the cosine function, and that the subtracted term (the “phase”) in the cosine function contributes to the calculation only during the second and third iteration of the  $j$  loop. (During the first iteration  $j = 1$ , so  $(j - 1) = 0$  and  $0.5*(j - 1) = 0$ .)

```
%}
    end
%{
This is the matching “end” for the inner  $i$  loop.
%}
end
%{
This is the matching “end” for the outer  $j$  loop.
%}
```

Notice that the previous Figure shows three (rough) cosine curves, two of which are displaced horizontally, perhaps reminiscent of the motion of an ocean wave. Wouldn't it be cool to animate this motion? (We'll get to simple animations in a future exercise.)

---

We suspect you've inferred that once you enter the “for loop” lines in the Command Window and the loop “runs,” to do it again requires reentering all the lines. The Command Window is great for performing simple operations that you don't necessarily want to do over and over. But for tasks that you might want to readily edit and/or redo numerous times, there is a much better way: the M-file.

Go to the lower left of the MATLAB window and click on the Start button. In the drop-down (or rise-up?!) panel go to Desktop Tools and then click on the Editor. This opens a simple line editor that looks a lot like Microsoft Notepad. Type the code below and, when you are finished, save it with a name that ends with the extension “.m”. This extension is important. Choose something

simple like “terrace.m” knowing that you can always change it later. (*Note: MATLAB doesn't pay much attention to spacing, including blank lines. Nonetheless, choose spacing and indentation that is appealing and easy to read. Others will be examining your code. We can offer hints.*)

```
%{
This is a MATLAB code prepared as an M-file. It plots the land-
surface-elevation profile of a river terrace and soil thickness.
The profile coordinates are specified using “for loops,” which can
be edited to create different profiles.
%}

xmin = 0;
xmax = 50;

imax = 101;

dx = imax/(xmax - 1);
soilthick = 0.5;

for i = 1:imax
    X(i) = (i - 1)*dx;
end

for i = 1:46
    Z(i) = 20;
end

for i = 47:55
    Z(i) = Z(i-1) - 1.0;
end

for i = 56:imax
    Z(i) = 10;
end

for i = 1:imax
    S(i) = Z(i) - soilthick;
end

plot(X,Z,X,S,'LineWidth',1.5)
axis([xmin xmax 5 25])
```

Once you have saved the file, leave the Editor open. (Drag the Editor to a position where you can also view the Command Window partition in the MATLAB window.) At the top of the Editor you will see a button that looks like a page with a downward-pointing arrow. This is the Run (or Save and Run) button. Click it. The program “runs” and then produces a plot of the land-surface and the underlying base of the soil.

You can readily edit lines in the M-file code and rerun it. For example, change the parameter *xmax*

(the maximum distance of the profile) and/or the parameter *soilthick* (soil thickness). Changing the land-surface profile in this code, however, is more tricky. The loop with  $i = 1:46$  sets the elevation of the terrace surface. The loop with  $i = 47:55$  sets the changing elevations down the scarp. The loop with  $i = 56:\text{imax}$  sets the elevation of the floodplain. To change the elevation/shape of the profile will require thinking through the loop limits (the starting and stopping points) and the change in elevation down the scarp.

Close the Figure, then click on the Command Window to make the cursor active in this partition. Enter the name of your M-file, for example

```
terrace
```

Thus, you can run your M-file from the Command Window. Moreover, MATLAB remembers each of the  $1 \times 101$  matrices  $X$ ,  $Z$  and  $S$  that your M-file created. So, close the Figure again, and in the Command Window enter

```
plot(X,Z)
```

The land-surface profile is plotted once again. As described above, you can query the  $X$ ,  $Z$  and  $S$  matrices. For example, entering

```
Z(89)
```

returns the value 10.

---

One final lesson, for now, on using MATLAB. In the Command Window enter

```
Q = [1 2 3 4 5 6 7 8 9 10]
```

which creates a  $1 \times 10$  matrix called  $Q$ . Suppose that later in your session you change your mind. For example, enter

```
Q = [1 1 1 1 1]
```

So now  $Q$  is a  $1 \times 5$  matrix. MATLAB in this example has properly accepted your change in the size (and element values) of the matrix  $Q$ . There are situations, however, where this type of change is not guaranteed, and you should be familiar with two more MATLAB commands.

Enter

```
clear Q
```

Then enter

```
Q
```

MATLAB tells you that  $Q$  is no longer defined. The “clear” command is thus handy for erasing or “resetting” your work. There is a “global” equivalent to this. If you click on the Edit button in the upper left corner of the MATLAB window and then click on Clear Workspace (as we did initially), all variables and functions that you previously defined in your session will be cleared from memory. The Clear Command Window option clears all of your previous entries in the Command Window.

### Exercises

1) Add brief comments for each line of your “terrace” M-file explaining what the line does. You can apply block commenting as used above (with `%{` and `%}` in the lines above and below your comments), or by preceding any comment with the character “%.” For example

```
for i = 1:10 % This starts the loop that computes..., etc.
```

or

```
% This M-file plots the land-surface-elevation profile of a river  
% terrace and soil thickness. The profile coordinates are  
% specified using “for loops,” which can be edited to  
% create different profiles.
```

Be complete, but succinct. The aim is to provide clear documentation of what the program does, including definition of variables and parameters, so that someone who is not familiar with the problem can read your comments and understand the program.

2) Create an M-file containing the nested loop and plotting algorithm for the cosine waves as introduced above, but increase the resolution of the plots by increasing the size of  $Z1$ ,  $Z2$  and  $Z3$  from 11 elements each to 21 elements each. Add comments that (briefly) explain each line in the algorithm.

3) Submit your two M-files to us as attachments in e-mail by the time/date specified in the Course Schedule. Clearly indicate in your e-mail which computer you used, and the directory path for your M-files. Make certain that your M-files “run” to your satisfaction without errors before submitting them.