Matlab programming, plotting and data handling

Andreas C. Kapourani

(Credit: Steve Renals & Iain Murray)

1 Introduction

In this lab session, we will continue with some more sophisticated matrix operations, we will learn how to program in MATLAB using scripts and functions. Also, we will go through data visualization using plots and how to handle text files.

2 Turning maths in matrix operations: Vectorization

If you write MATLAB code with lots of explicit loops (even worse, lots of nested loops), then your code is unlikely to be taking advantage of the facilities for efficient matrix-vector computation (**vectorization**). Writing your code in terms of matrix-vector operations will result in code that is easier to read and that runs an order of magnitude faster. Given a mathematical expression like:

$$R = \sum_{i=1}^{I} fn(x_i, x_j) val(z_i) \tag{1}$$

The results of `fn` could be put in an IxJ matrix, and the results of `val` in a $Ix1$ vector. The sum is then a matrix-vector multiply:

```
>> fn = [2 3; 4 6; 7 8];   % fn is a 3x2 matrix
>> val = [4; 7; 8];        % val is a 3x1 matrix
>> R = fn' * val           % R has values: [92; 118]
```

Exercises

1. Write your code in `for` loop to compute the above mathematical expression.

2. Given another expression:

$$R = \sum_{i=1}^{I} fn(x_i, x_j) weights(z_i, z_j) \tag{2}$$

   use `vectorization` to compute the result vector (**hint**: first do the multiplication inside the sum for all i and j at once, and then sum out the index i).

Sometimes `repmat` or `bsxfun` is required to make the dimensions of matrices match. Check online or Matlab help to understand what they do.

In general, if you see code that involves tricks you don't understand, try to write a for-loop version based on what you think it should do, and see if you get the same answer. If you are writing code

with lots of for loops, keep going, but then afterwards try to replace parts of it with a `vectorized` version, checking that your answers don't change. There are more references in the Matlab notes at:

http://homepages.inf.ed.ac.uk/imurray2/compnotes/matlab_octave_efficiency.html

3 Programming in MATLAB

You can get lots done in MATLAB using it as an interactive environment, issuing commands one at a time, while keeping data and variables in the workspace. However you will soon want to write *functions* or *scripts*.

A MATLAB function or script is written as an M-file (a text file with the extension `.m`)

Functions A function file should contain a *single* function definition., Function `func` should reside in a file called `func.m`. Functions can take input arguments and return output values, and variables within them are local.

Scripts A MATLAB script file is a sequence of MATLAB commands (and no `function` heading) stored in an M-File. When a script executes (called by typing its name without the `.m`) the commands are executed, and variables mentioned in the script reside in the MATLAB workspace (and are available when the script ends).

3.1 Examples

On the upper-left side of the MATLAB GUI click on the `New` button. This will open the MATLAB editor, and we are ready for creating our first scripts and functions.

Copy the following code in your script, save the script and then on the MATLAB `Command Window` type `myscript`, where `myscript` is the name of your script (Make sure that the script is saved in the same directory from where you are running MATLAB). Our running example, will be to perform operations on a matrices, such as finding the mean of each column, centering the columns, etc.

```
% Create a 5x3 matrix
A = [ 1  2  3;
      4  5  6;
      7  8  9;
     10 11 12;
     13 14 15];
% Printing command
fprintf('\nComputing column means: \n');
[I, J] = size(A);
mu = zeros(1, J); % allocate space for answer
for i = 1:I
    for j = 1:J
        mu(j) = mu(j) + A(i,j);
    end
end

mu = mu / I  % result should be: 7 8 9
```

Of course, we could get the exact same result if we used the MATLAB built in function `mean`.

```
>> mean(A, 1)   % 1 indicates to apply the mean per column
ans =
     7     8     9
```

In most of the cases we want to repeat a sequence of commands, e.g. finding the mean of a matrix or vector, but we want to be able to do it with any given matrix or vector. This can be done using `functions` (also called subroutines). Functions are the same as executable files, but we can pass them parameters.

Functions should start with the following line:

```
function [x] = function_name(param_a, param_b)
```

where `x` is the returned variable (if we want to return more than one variables we include the list in the brackets with commas in between the variable names). `param_a` and `param_b` are the input parameters to the function. This function should be saved in a file named `function_name.m`.

To create a function file, we click on `New` in the upper left side of MATLAB GUI and then choose `function`. This function will center the columns of the matrix to its mean (i.e. zero-mean column matrix). We will write different versions of this function gradually, so as to understand how we can write efficient MATLAB code.

To make each column zero-mean, we could write the following function:

```
function [ A_shift ] = mean_shift_1( A )
%MEAN_SHIFT_1 Laborious centering of each column using for loop

    [I, J] = size(A);
    mu = mean(A, 1); % compute the mean of each column
    A_shift = A;     % (lazily) creates a copy
    for i = 1:I      % for each row
        for j = 1:J  % for each column
            A_shift(i,j) = A_shift(i,j) - mu(j);
        end
    end
end
```

Then we call the function as follows:

```
>> A_shift = mean_shift_1(A)
A_shift =
    -6    -6    -6
    -3    -3    -3
     0     0     0
     3     3     3
     6     6     6

>> mean(A_shift, 1) % check the columns are centered
ans =
     0     0     0
```

A faster way is subtract the mean row of each row in one operation per row:

```
function [ A_shift ] = mean_shift_2( A )
%MEAN_SHIFT_2 We can take the mean row off each row
```

```
% in one operation per row.

    [I, J] = size(A);
    mu = mean(A, 1); % compute the mean of each column
    A_shift = A;
    for i = 1:I       % for each row
        A_shift(i,:) = A_shift(i,:) - mu;
    end
end
```

The result is of course the same:

```
>> A_shift = mean_shift_2(A);
>> mean(A_shift, 1)
ans =
     0     0     0
```

We could do this even faster (and without using `for` loop at all) by using the `repmat` function.

```
>> mu = mean(A, 1); % compute the mean of each column
>> A_shift = A - repmat(mu, size(A,1), 1);
>> mean(A_shift, 1)
ans =
     0     0     0
```

`repmat` function replicates the vector `mu` with Nx1 copies of `mu`, where N is the number of rows of matrix A. The generated matrix is then subtracted from the original matrix A. Check individually each command of the above code and look at `mu` and `repmat(mu, size(A,1), 1)` so as to understand what the code does.

Finally, MATLAB also supports `broadcasting`, but through a regrettably ugly function called `bsxfun`. Any size of length 1 is implicitly expanded, so that the sizes match without having to call the `repmat` function.

```
>> mu = mean(A, 1); % compute the mean of each column
>> A_shift = bsxfun(@minus, A, mu);
>> mean(A_shift, 1)
ans =
     0     0     0
```

4 Plotting

4.1 2-D line plots

To create 2-dimensional line plots, use the `plot` function. For example, we can plot the `sin` function evaluated from 0 to 5 and multiplied with π as follows:

```
>> x = (1:0.01:5);     % create some points
>> plot(x, sin(x*pi)) % a new window will open with the plot
```

We can add an optional title for the plot and label the axes as follows:

```
>> title('Sine function plot')
>> xlabel('x')
>> ylabel('sin(x*pi)')
```

Now close the plotting window.

To plot several curves on the same graph we use the command:

```
>> hold on
```

to prevent previous plots being erased. We can now type:

```
>> plot(x, sin(x*pi))          % plot sine function
>> plot(x, cos(x*pi), 'r--')  % plot cosine function
>> legend('sin','cos', 'Location','northwest') % add legend
```

The second argument defines the line colour (e.g. type `plot(x,y,'r')`). You can also change the line style, etc. Use the Help window to find out more. It also possible to add axis labels, a title, tick marks, etc. to a plot, either through additional arguments (see Help), or interactively using the menus on the plot window.

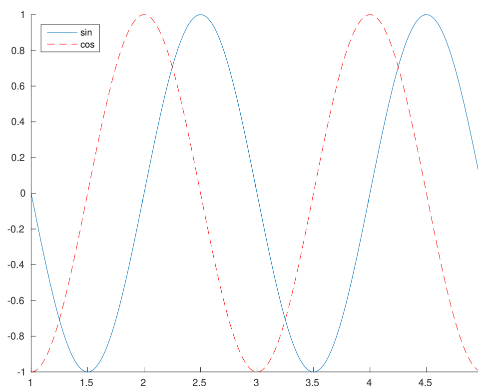

Figure 1: *2D line plot.*

4.2 Drawing contours

A contour line of a function of two variables is a curve along which the function has a constant value. A plot of several contour lines is called a **contour plot**. MATLAB provides a `contour` function for drawing contour plots.

First, we need to create a surface defined by a function in two variables, $z = f(x, y)$. Thus, in order to evaluate z we need to generate a set of (x, y) points. This can be done using the `meshgrid` built-in function in MATLAB.

```
>> figure     % figure opens a new plot window

% generate matrix of elements that give the range over x and y,
   that is: -10 < x < 10 and -5 < y < 5
>> [x y] = meshgrid(-10:0.1:10, -5:0.1:5);
>> z = x.^2 + y.^2;   % define function z = x^2 + y^2
>> contour(x, y, z)
```

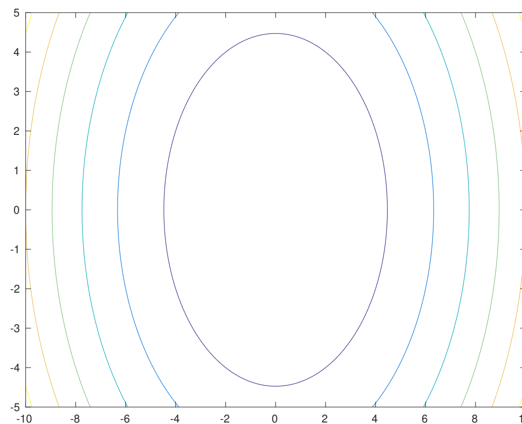

Figure 2: *Contour plot.*

4.3 3-D plots

3-dimensional plots display a surface defined by a function in two variables , $z = f(x, y)$. Hence, as we demonstrated before with `contour` plots, we need to create a set of (x, y) points using `meshgrid`, then define the function z, and finally call the function `surf`, instead of `contour`, in order to create the surface plot.

```
>> figure
>> [x y] = meshgrid(-10:0.1:10, -5:0.1:5);
>> z = x.*exp(-x.^2 - y.^2);
>> surf(x, y, z)                    % call the surf function
```

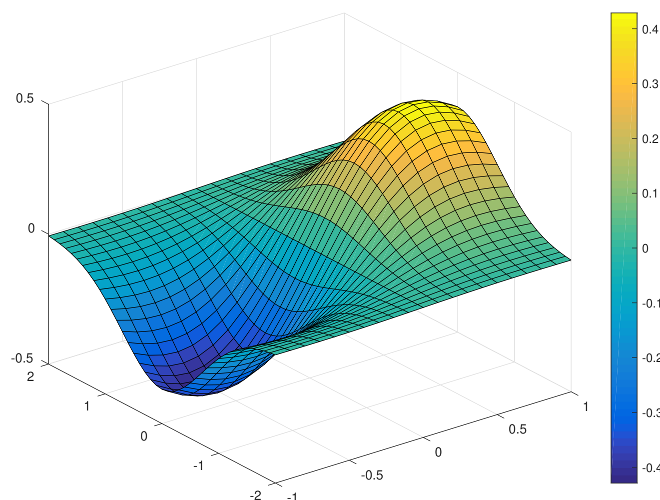

Figure 3: *Example 1, surface plot.*

Another example using a different function:

```
>> figure
>> [x,y] = meshgrid(-1:0.1:1,-1:0.1:1);
>> z = sqrt(x.*x + y.*y);
>> surf(x, y, z)
```

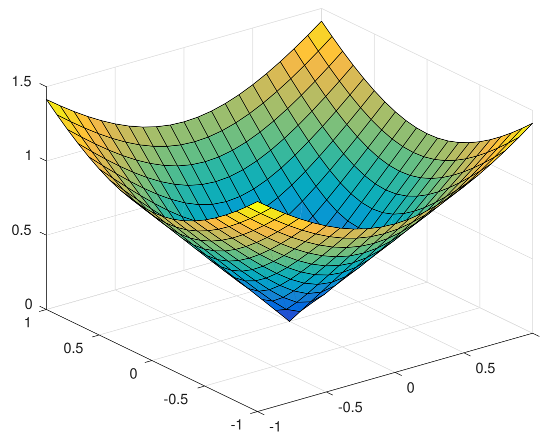

Figure 4: *Example 2, surface plot.*

MATLAB has many demos, several to do with plotting. Either type `demo` or click Help→Demos.

We can save the figure plots as follows:

```
>> close all  % close all opened figures
>> x = (1:0.01:5);
>> plot(x, sin(x*pi));
>> print('sinPlot','-dpng')  % save the plot as sinPlot.png
% or we could save it as Vector Graphics File, etc.
>> print('sinPlot','-depsc') % save the plot as sinPlot.eps
```

4.4 Exercises

1. On the same graph in the range $(0, 2\pi)$ plot $\sin(x)$ and $\cos(x)$.
2. In a new figure plot, on the same graph, $\lg(x)$, x, $x\lg(x)$, and x^2. (Note that `log` gives $\log_e$ and `log10` gives $\log_{10}$ in MATLAB. lg is not built in.)
3. Plot $\sqrt{x}$ and $x^{\sin(x)}$ on the same axes.
4. Plot x^3 and 1.01^x on the same axes.

Make sure the axes are appropriately scaled (see `axis` command for scaling the x and y axes) and labeled, that the graph has a title, that the individual curves are titled in the legend etc.

5 Handling Data and Files

5.1 Saving and Loading Data

In order to save our work and store our data locally in a file we can use the `save` command. For example:

```
>> test_data = magic(7)  % creates a 7x7 'magic' matrix
% store the test_data variable in test.mat file
>> save test.mat test_data
```

If we wanted to save all the variables that we have in our workspace we could simply write:

```
% store all workspace variables in test.mat file
>> save test.mat
```

Now we can load them back using the `load` command as follows:

```
>> clear          % clear all the workspace variables
>> load test.mat % load test.mat file
```

5.2 Handling text files

In many cases we need to read our data from text files, assuming that they have a specific format, e.g each entry is tab delimited. We can handle these files in various ways in MATLAB.

Let's see an example. First we need to create a file, we can do these by writing the `test_data` matrix in the data.txt file using a tab delimited format for each line. This can be easily done using the following command:

```
% First we write our matrix 'test_data' in the data.txt file
>> dlmwrite('data.txt', test_data, 'delimiter', '\t')
```

In you folder you can see the `data.txt` file. Open in and see if everything is written as expected.

Now, we can read the data of this file using the following code:

```
>> filename = 'data.txt';  % name of the file
>> delimiterIn = '\t';     % split data at specified delimiter
>> headerlinesIn = 0;      % read numeric data starting from line
   headerlinesIn+1
>> A = importdata(filename,delimiterIn,headerlinesIn);
```

Now check that matrix `A` is the same as the `test_data` matrix.