PRA1004 Scientific Computing 2013

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Lab Assignments Week 3

Overview Lab 3

In this lab session, you will...

- ...get to enjoy the wonderful world of matrices.
- ...use different *interpolation* methods on a data set of the population of The Netherlands in the previous century as well as random data, that you generate yourself.
- ...see that making an exact fit is not always the best thing to do. Especially in the presence of sensor noise, it may be preferable to fitting lines and polynomials using *least-squares regression*.

As the name MATrix LABoratory implies. Matlab is for a big part based on working with matrices. This does take a bit of learning in the beginning, and some of the early exercises may not be the most interesting ones. However, you will quickly get the hang of it and then you will also see how matrices and matrix operations can make certain complex operations really easy to write.

For this lab, you will need to download the lab kit from my website.

General Instructions

- Save your work for this lab in a script. That is, create a big script file with the name `week3_YOURNAME.m` in which you put all the commands that were used in answering the assignments.
- Leave out any code that was not directly useful.
- Use `%` to make comments in your code. In particular use it to make 'sections' in your script. E.g.:

```
% Scientific Computing - week 3 - John Doe
%
% Assignment 1
% here we should ...
<ACTUAL CODE>

% Assignment 2
% ...
```

- Make sure it looks relatively nice, next week you will need to show it.
- Make sure that if you run your script, you get the results that are asked for. Also, make sure that you can easily copy/paste the code for certain (parts of) assignments from your script, in case I want you to show that your code actually works.

1 Getting Help

In Matlab, there are 2 main ways of getting help:

1. type `help COMMANDNAME`, where `COMMANDNAME` is the name of the command you want help about.
2. open the helpdesk: either type `helpdesk`, or select "start->help".

2 Creating Matrices

In this lab you will start working with vectors and matrices. These can be created by using square brackets. For instance, try typing

- `X1=[1,2,3]` to create a row vector with 3 elements (i.e., a 1-by-3 matrix).

As you can infer the comma is used to separate the entries in different columns. To separate rows, Matlab uses the semicolon. For instance:

- `Y=[1;2;3]` creates a column vector with 3 elements.

In order to create matrices, we can combine the two:

- `A = [1,2;3,4;5,6]`

Using the so-called *transpose* operator (indicated with a quote ''), you can transform a column vector into a row vector and vice versa. This also works on matrices: all rows are converted to columns (and vice versa).

- E.g., try `Ytrans=Y'`, `Atrans = A'`.

Matlab also has a number of commands that will generate entire matrices for you. For instance, try the following commands:

- `zeros(3)`, `zeros(2,4)`, `ones(3,4)`, `rand(4,2)`.
- Can you explain what they do? Can you generate a random column vector with 10 elements?

Matrices can also be created by putting together, or *concatenating*, two (or more) previous matrices. For instance, try

- `X2 = [4,5,6]`
- `X = [X1;X2]`
- `Y2 = [4;5;6]`
- `Y = [Y1,Y2]`

  (Can you explain the difference between the use of ';' and ',' in these?)

Another thing that you should know, is that it is possible to have an empty matrix

- `X = []`

This is often used when you want to incrementally build up a matrix using concatenation; you can add something to the empty matrix. Try

- `ans=[]`
- `[ans, X2]`
- `[ans, X2]`
  (yes, type it twice!)

Appending two empty matrices just gives you an empty matrix. (How can you test this in Matlab?)

Finally, there are two common ways to create vectors that contain a range of numbers. First, the colon ':' operator creates a range of integers (e.g., `R1 = 1:9`) or numbers with a fixed spacing (e.g., `R2 = 1:0.5:9`).

- Try out these command.

The second way of specifying ranges is using `linspace`. E.g., try

- `R3 = linspace(1,9,10)`
- Do you understand the difference between the colon operator and linspace? (what is the argument that is different?)

3 Element-wise Functions

Of course, matrices would not be useful if we couldn't manipulate them. Here, we first consider so-called *element-wise functions*. These work on individual elements of matrices and vectors. First there are 'matrix-scalar' operations that take as one argument a matrix (or vector) and as the other a scalar (i.e., a single number). For intance:

- `X = [1:3]`
- `X2 = X * 3`
- `X3 = X - 2`
- `X4 = X .^ 2`

Note the use of the period (`.`) in the last example! (Try the command without out it, what is the error message? The explanation is that the period is required to denote 'element-wise' power. In contrast `X^2` is interpreted as `X*X`, which is matrix multiplication, which we will encounter later. See also `help ^` and `help .^`).

A similar issue happens with elementwise matrix-matrix (or vector-vector) operations. For instance, you can verify that

- ```
  X = [1:3];Y = [4:6];
  X+Y
  ```

gives you the element-wise addition. However, you will notice that `X * Y` does not work. Instead, try

- `XY = X .* Y`

(again the period indicates that we want the vector-wise operation).

Now, we can use the learned operations to do something a bit more useful. For instance, we can create a 'look up table' to convert inches to cm in the following way:

- ```
  inches = [0:5:50]
  cm = inches * 2.54
  table = [inches', cm']
  ```
- Verify this. Make sure you understand every line!

Make a plot of the function $y(x) = x^2 + 2x + 4$, over the 'interval' `X=[-4:0.5:4]`.

- first define the range, then compute the vector of Y-values, then plot it.
- To make the plot a bit nicer, we wil plot both points and lines using:
  ```
  plot(X,Y,'r+')
  hold on
  plot(X,Y,'b-')
  ```

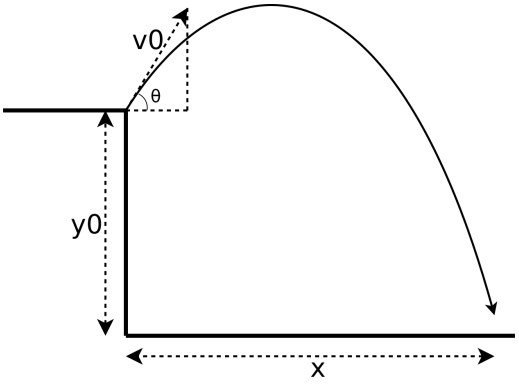

Figure 1: Flight of a projectile without friction.

Note that the strings `'r+'` and `'b-'` are line specifications. For all options try:

- `help plot`, or
- look at the 'LineSpec (Line Specification)' entry in the help desk.

Practice some more with

- Ch.2 ex. 31-33, 35, 37, 39

4 (Hand-in) The Flight of a Projectile Revisited

Instructions. This is a hand-in assignment. You should put your code for this assignment in a separate script called `handin_w3_YOURNAME.m`. You should also generate a report (in pdf) that includes the required plots. Remember to follow the instructions for hand-in assignments!

- **NOTE:** this is the first part of the hand-in assignment. (Clearly indicate that using comments in your code).

Assignment. Remember that in the first lab we investigated how far a thrown projectile would fly by plotting its trajectory. The problem is illustrates in Fig. 1 and the equation that governs the height is

$$y(x) = y_0 + x \cdot tan(\theta) - \frac{g \cdot x^2}{2(v_0 \cdot cos(\theta))^2}$$

The considered constants are $g = 9.81$, $\theta = .2\pi$, $v_0 = 10$, $y_0 = 1.8$. Here we will investigate the problem in Matlab.

- First, similar to the inch-to-cm example, create a table that maps 10 evenly spaced points in the interval $[0, 12]$ to their heights.
  (Hint: use `linspace` to generate the interval, and then apply elementwise operators to create the y values. Another hint: do ex. 37 first!).
- Now, we can recreate the trajectory plot. Make sure that you set variables `x,y` to contain your interval and the heights respectively, and create the plot using:
  - `plot(x,h, 'r+-', 'Markersize',10);`
  - Also add a title (use the `title` command), and name the axes using `xlabel('distance'); ylabel('height')` .
- Think of a way to make the plot smoother and execute it. Include the final, smooth, figure in your report (along with a description of the problem, and important parts of the code of course... see the requirements!)

5 Indexing

An important operation is to access elements or parts of a matrix you might be working with. This is called *indexing*. Indexing is done using round braces `()`. For instance, we can get the element from row 4, column 2 using:

- `table(4,2)`

(For these commands to work, make sure you still have the variable `table`—see inch-to-cm above—defined.)

It is also possible to select entire rows or columns. To do this, use the colon ':' as a wild card (i.e., it denotes 'all rows' or 'all columns'). For instance

- `table(:,1)`

does specify the column index (1), but uses the colon for the rows, meaning 'all rows'. The result is that you get back the entire first column.

- Select the 3rd row of `table`.

Indexing is also important to process the results of certain functions. For instance

- `result = size(table)`

will give you a 1x2 matrix, in which the first element denotes the number of rows, and the second denotes the number of columns.

- use the indexing operator on `result` to set two variables: `num_rows,num_cols`

As a final remark, in cases (like `size`) where you know the number of elements you will get back from the function, it is also possible to use the following short hand notation: `[rows, cols] = size(X)`.

- Practice your matrix skills, do Ch.2 ex. 40–44.

6 Inner Products

The 'inner product', also called 'dot product' is a particular way of multiplying the elements of two vectors, and then adding them up. In particular, if we have two vector $\langle 1, 2, 3\rangle$ and $\langle 4, 5, 6\rangle$, their inner product is:

$$1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

Expressed in formulas, the inner product of two vectors $a = \langle a_1, a_2, \ldots, a_n\rangle$, $b = \langle b_1, b_2, \ldots, b_n\rangle$ is

$$dot(a, b) = \sum_{i=1}^{n} a_i b_i.$$

This simply means multiply the corresponding elements of the vectors, and then add everything up.

While inner products seem strange at first, they are in fact very convenient. For instance, we can use them to easily compute the amount of calories in our lunch. The number of calories in certain lunch foods are shown in Table 1.

- Store the number of calories for the various items in a row vector called `calories`. Use the same ordering as in the table.
- Create another row vector, `lunch`, that specifies the amount of each item that you think is appropriate for a big lunch. E.g., 1 banana, 4 slices of bread, 1 cup of milk and 2 apples.
- Now we can easily compute the number of calories using `lunch_cals = dot(calories, lunch)`.

	calories	fat(g)	Calcium(mg)	Vitamin C(mg)
banana	200	0.7	11.3	19.6
slice of bread	68.9	1.1	26.8	0
cup of milk	149	8.4	246	2.2
apple	65	0.2	7.5	5.7

Table 1: Nutrition in fooditems.

Similar computations can be used for many things. For instance, to perform similar operations on the other nutrition columns, but also to compute the weight of a collection of items (e.g., to compute the mass of a spacecraft – Example 6.1 in the book [1]).

- Do Ch. 6, ex. 1,2.

As said, similar computation can be used for the other listed nutritions. In fact, by using **vector-matrix multiplication**, we can compute all of them at the same time. If we have a $k \times n$ matrix A, and a $n \times 1$ (i.e., column) vector b, their product is written as

$$Ab = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{k1} & & & a_{kn} \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} \tag{1}$$

the result, a column vector c, has as its entries different inner products. Let us write $A_{row(i)}$ for the i-th row of matrix A. Then

$$c_i = dot(A_{row(i)}, b),$$

that is, the i-th entry of c is the inner product between the i-th row of A and b. This means that we can compute several inner products at the same time, as we will now demonstrate.

- Create a matrix, say `nuts`, containing the all the (data) columns from Table 1.
- Create a variable, say `nuts_T`, that contains the transpose of `nuts`.
- Transform `lunch` to a column vector.
- Compute the total amount of nutritions in `lunch` using: `lunch_nuts = nuts_T*lunch`.

The result should be a (column) vector of which the first entry are the number of calories in the lunch, the second entry denotes the amount of fat, etc.

- Notice what is going on: the first entry is the inner product between `lunch`, and what part of matrix `lunch_T`?

The inner product of two vectors is frequently denoted as a special case of matrix. E.g., when a, b are column vectors, we have that

$$dot(a, b) = [a_1, a_2, \dots, a_n] * \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a^T * b \tag{2}$$

(where T denotes the the transpose). This is also simply written as $a^T b$. Note that this is just a special case of equation (2).

- Define a row vector `a` and column vector `b` and multiply them: `a*b`. Does it match the inner product?

- What happens when you reverse the arguments? (I.e., when you type `b*a`)? (In order to fully understand what is going on, you will need to understand matrix-matrix products. We will treat these next week.)

7 Interpolation

7.1 Population Data & Piecewise Interpolation

- Load the data from `population.txt`, which contains the population (in thousands) of The Netherlands for the indicated years. Make a nice point plot of the data. Add labels to the x and y axis.
- Use the `interp1` function to perform a linear and spline interpolation. In particular, perform the interpolation for all years between 1900 and 2000. Use the `helpdesk` command to understand the exact syntax of the function and have a look at the example. Also see Example 8.2 in the book [1].
- Also perform a 'nearest' interpolation.

Often it is convenient to plot multiple lines in the same figure. This can be accomplished using the commands `hold on` / `hold off` to avoid/enable erasing previous plots. In particular, after giving the `hold on` command, Matlab will leave old lines when issueing new plot commands. To plot a line in a new clear figure, first type `hold off`.

- Plot the nearest, linear and spline interpolations in the same figure. Use line plots. What seems to be the better interpolation for this data?

The lab kit also contains the actual data for the missing years in a file `populationFull.txt`.

- Import the data and plot it alongside the interpolation that you had identified to be best. (So: make a new plot that only contains the interpolation that you thought was best. Then add the data for the missing years to the plot)

7.2 Polynomial Interpolation

A different way of doing interpolation is called *polynomial interpolation*. The idea is that if you have N data points, you can exactly fit a polynomial of degree $N-1$ through the points. Here we will not discuss exactly how such a fitting is found (of course, it can be seen as a particular system of linear equations, and can be solved using techniques of linear algebra...!), but rather demonstrate how to do it in Matlab.

In particular, you will use a self-written function `RandomPolyInterpolation` that looks like this:

```
function RandomPolyInterpolation(num_points)
% This function:
%    RandomPolyInterpolation(num_points)
% generates a number of random data points and performs
% polynomial interpolation.
%
% input arguments:
%   num_points - the number of random point generated
%
% output arguments:
%   none
% create the X-range:
X = 1:num_points;
% create corresponding random Y-values:
Y = rand(1,num_points)*100;
% perform polynomial evaluation
```

```
c = polyfit(X,Y,num_points-1);
% generate finer spaced x-values:
Xfull = linspace(0,num_points+1, num_points*100);
% get their y-values according to the interpolation:
Yfull_p = polyval(c, Xfull);
% plot the functions
hold off;
plot(X,Y,'+r')
hold on;
plot(Xfull, Yfull_p, '-b');
```

- Try running this functions once using `RandomPolyInterpolation(3)`.
- Open up the function using `edit RandomPolyInterpolation.m`.
- Notice the blue color of the word 'function'. This means it is a keyword (special word). In particular, what it does it that it tells Matlab: "watch out, what follows is a function, called RandomPolyInterpolation).
- Try to understand what the other lines do. (you will need some of these functions in Section 8.1). Try executing some of the commands in the command window.
- Try calling the function a couple of times for a small (2–4) number of points. Does it seem logical that it is always possible to fit a polynomial of degree $N-1$ through N points?
- Try calling the function for larger numbers of points (5–10), what do you notice? (You should observe something called *Runge's phenomenon.*)

7.3 Noisy Measurements

1. Load the data from `noisy.m` into a variable called `noisy`, plot the data (use a point plot).
2. What kind of interpolation would you want to use for this data?
3. Do the interpolation.

8 General Curve Fitting & Least Squares Solutions

As might be clear by now, it is not always the best choice to use an *interpolent* (i.e., a function that passes exactly through the data points). The reason is that this may give you a very complex function, especially when your measurements have some noise, while the underlying relation in fact is much simpler.

8.1 Diamond Prices

Here we examine some data about diamond prices.

- `load diamond.dat`

The first column contains the number of carats, while the second column containt the price.

- Plot the data set using `scatter`.
- We can peform a least-squares fit of a line (a polynomial of degree 1) using `polyfit(X,Y,1)`. (You still need to set `X,Y` though!)
- Evaluate the fit on the interval `denseX=[0:0.01:0.4]` using `polyval` and plot the resulting fit in the existing figure (i.e., together with the data points.)
- If needed, you can adjust the range of the axes using `axis([0, 0.4, -300, 1200])` .

What is peculiar about the found relation?

8.2 More Noisy Data

In this section, we will again investigate the noisy data. In contrast to Section 7.3, we will not make a interpolation, but a (least-squares) fit to the data.

- Create a scatter plot of `noisy`. (If necessary, load the data from `noisy.m` again.)
- Open up the basic fitting tool: in the figure, select "Tools->basic fitting". Try some of the different fits. What seems to be the best model?
- Select 'plot residuals', matlab will now show a bar graph of how big the 'errors', the distance from the fit to the actual data point, are.
- Also select 'show norms of residuals'. This will take the square of all the errors (to make them positive), sums them all up, and then takes the square root.[1]
- You can use the norm of residuals to compare different fits. Clearly, a lower norm means that there is less error.
- Make sure that you only select the curve that has the best fit. This should normally be one of the higher order polynomials. (Note, you can only select a polynomial fit, so not one of the 'interpolations').

Now we will save our fit. The following instructions might be difficult to understand without using screen shots. If you have trouble following what to do, open the matlab helpdesk and find the help entry "Example: Using Basic Fitting GUI". Then, an explanation of how to save the fit with screenshots is given under "View and Save the Cubic Fit Parameters".

- In the basic fitting tool, click on the button with the arrow pointing right. This should open up more options.
- Click "Save to workspace", this will allow us to save the variables describing the fit to the workspace. Behind "Save fit as a MATLAB struct named", type `ourfit`. Click ok.
- Now in the command window type `ourfit`, you should see something like

```
>> ourfit
ourfit =
type: 'polynomial degree 10'
coeff: [1x11 double]
```

- The variable `ourfit` is what is called a *struct*. It is a variable that groups together some other variables, called *fields*. This struct has two fields: 'type' and 'coeff'. The latter stores the actual coefficients. To show them type `ourfit.coeff`.

The data was in fact generated from a (fictitious) series of noisy measurements. We will now generate more data and investigate whether the previous fit is still a good fit.

- Run `data2=GenerateNoisyMeasurements` to get a new data set.
- Create a new figure: `figure(2)`. And make a scatter plot of the new data in it.
- Now we will recreate the previously fitted line in the new plot.
  - First define the range over which we want to plot: `[Xrange=50:300]`.
  - Now we evaluate the previous fit on this range using `predictedYs = polyval(ourfit.coeff, Xrange)`.
  - Plot it together with your previous best fit (use `hold on`).

[1] That is, it is the square root of the *sum of squared errors*. A term used frequently in computing.

- Is it still a good fit? Thinking back, was the choice for a higher order polynomial the best?

The problem of selecting what fitting function to use is called *model selection* and has a rich literature in statistics. One rule of thumb is to use the simplest possible model that does a good job of describing the data, a principle referred to as *Occam's razor.* (`http://en.wikipedia.org/wiki/Occam%27s_razor`).

9 (HAND-IN) Golf

Instructions. This is a hand-in assignment. You should put your code for this assignment in a separate script called `handin_w3_YOURNAME.m`. You should also generate a report (in pdf) that includes the required plots. Remember to follow the instructions for hand-in assignments!

- **NOTE:** this is the second part of the hand-in assignment.

Assignment. In this exercise, we will use a data set `golf.dat` with statistics on golf putting [2].

- load the golf data.

The data consists of 3 colums: the distance (in feet), the number of attempts, and the number of successes.

- Compute the success rate for each distance (this can be done with just 1 line).
- Make a scatter plot showing the distance vs. the succes rate.
- Use the curve fitting tool to fit a curve through the (distance, success rate) data. What seems to be the best model?

References

[1] Delores .M. Etter. *Introduction to Matlab.* Pearson Education, second edition, 2011.

[2] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis.* Texts in Statistical Science. Chapman and Hall/CRC, 2 edition, July 2003.