Miniproject 4

In this miniproject we look at the least squares method in connection with regression.

Read first §6.4 in the book (corresponding to §7.4 in the 2014 edition).

As an example on how we use least squares we perform the calculations from Example 1 on page 404 in the book (this is page 468 in the 2014 edition):

```
>> x = ones(5, 1)

x =

     1
     1
     1
     1
     1

>> w = [260 272 275 267 268]'/100

w =

    2.6000
    2.7200
    2.7500
    2.6700
    2.6800

>> C = [x w]

C =

    1.0000    2.6000
    1.0000    2.7200
    1.0000    2.7500
    1.0000    2.6700
    1.0000    2.6800

>> y = [200 210 210 203 204]'/100

y =

    2.0000
    2.1000
    2.1000
    2.0300
```

```
    2.0400

>> a = inv(C' * C)*C'*y

a =

    0.0555
    0.7446
```

An alternative way to calculate a is with the command `mldivide`:

```
>> a = mldivide(C, y)

a =

    0.0555
    0.7446
```

If the equation $Ca = y$ is inconsistent, `mldivide` still gives us an answer; the answer is a solution in the way mentioned on page 405 in the book (correspondng to page 469 in the 2014 edition).

Besides `mldivide` we can use the following commands in connection with the least squares method.

`polyfit` direct way to get the *coefficients* for the straight line (and in general the polynomial), we wish to describe data with.

For the example above we get the following with `polyfit`:

```
  >> p = polyfit(w, y, 1)

p =

    0.7446    0.0555
```

Notice that the coefficients from `polyfit` are sorted after the power of the independent variable – you can find further info in the documentation.

`polyval` computes values of a polynonial.

Do exercise 1, 11 and 15 on page 409 (page 473 in the 2014 edition) and exercise 8 on page 484 (page 548 in the 2014 edition).

As mentioned on page 406 (page 470 in the 2014 edition) in the book we can easily extend the method from above to find generel approximating polynomials instead of straight lines.

Instead of typing the commands to compute a every time we get new data, we can collect the commands in a function.

In previous exercises we have been collecting commands in m files. The MATLAB terminology for such a collection of commands is a *script*.

The difference between a function and a script is that in a function we do not collect intermediate results, but get only the results we ask for – the *output* from the function.

You can see examples of functions on the homepage with the files `linreg.m` and `linreg2.m` that compute the coefficients for approximating first and second degree polynomials using the least squares method. The file `example.m` and screencast 7 shows how to use the two functions.

Here is also a short introduction to functions:

- A function must have the same name as the file where it is saved.
- The first line of the file with the function `etellerandet` should be

  `function out = etellerandet( in )`

  `out` is the result from the function and `in` is the input – in the same way as `polyfit` above.
- End the file with `end`.
- It is er *always* a good idea to include an explanation in the top of the function that describes how to use the function – next year you have probably forgotten what the function does even though it seems obvious at the time of creation.

In the function `linreg2` is an example of a *loop*. Let us take a look at what the function does and why it makes sense to use a loop.

First we define the matrix C with the command

```
C = ones( size(x, 1), 3 );
```

that makes a `size(x)` $\times 3$ matrix with 1 in all entries. We now replace the second column with $x = [x_1, \ldots, x_n]$ and third column with $x.\hat{}2 = [x_1^2, \ldots, x_n^2]$. This is done with the loop

```
for j = 1:2
    C(:, j+1) = x.^j;
end
```

The loop runs from $j = 1$ to $j = 2$. In the first run j is equal to 1 and the command `C(:, 2) = x.^1 = x` is executed. That is, the second column is replaced with x as desired.

In the second run j is equal to 2 and `C(:, 3) = x.^2` is executed. That is, the third column is replaced with `x.^2` as desired.

If higher powers are desired, corresponding to a matrix with e.g. 4 columns, then we change

```
C = ones( size(x, 1), 3 );
```

to

```
C = ones( size(x, 1), 4 );
```

and `for j = 1:2` to `for j = 1:3`.

Make your own function, for instance with the name `linregn`, that takes the x values, y values and the degree of approximating polynomial n as input and returns the coefficients of the polynomial as output.

Test your function on the data from the homepage in the file `polynomial_data.txt`, by approximating them with a polynomial of degree 3; to load the data into MATLAB you can use the command `dlmread`.

To conclude this miniproject you can calculate the MATLAB exercises 5, 6 and 7 on page 289. In connection with exercises 6 and 7 it is a good idea to take a look at example 3 on page 279-280 (ignore the comment about exercise 98 in section 4.5).