E7: Introduction to Computer Programming for Scientists and Engineers
University of California at Berkeley, Spring 2017
Instructor: Lucas A. J. Bastien

Lab Assignment 09: Least-Squares Linear Regression
Version: release

Due date: Friday March 24th 2017 at 12 pm (noon).

General instructions, guidelines, and comments:

- For each question, you will have to write and submit one or more Matlab functions. We provide a number of test cases that you can use to test your function. The fact that your function works for all test cases provided does not guarantee that it will work for all possible test cases relevant to the question. It is your responsibility to test your function thoroughly, to ensure that it will also work in situations not covered by the test cases provided. During the grading process, your function will be evaluated on a number of test cases, some of which are provided here, some of which are not.
- Submit on bCourses one m-file for each function that you have to write. The name of each file must be the name of the corresponding function, with the suffix `.m` appended to it. For example, if the name of the function is `my_function`, the name of the file that you have to submit is `my_function.m`. **Carefully check the name of each file that you submit.** Do not submit any zip file. If you re-submit a file that you have already submitted, bCourses may rename the file by adding a number to the file's name (*e.g.,* rename `my_function.m` into `my_function-01.m`). This behavior is okay and should be handled seamlessly by our grading system. Do not rename the file yourself as a response to this behavior.
- A number of optional Matlab toolboxes can be installed alongside Matlab to give it more functionality. All the functions that you have to write to complete this assignment can, however, be implemented without the use of any optional Matlab toolboxes. We encourage you to not use optional toolboxes to complete this assignment. All functions of the Matlab base installation will be available to our grading system, but functions from optional toolboxes may not. If one of your function uses a function that is not available to our grading system, you will loose all points allocated to the corresponding part of this assignment. To guarantee that you are not using a Matlab function from an optional toolbox that is not available to our grading system, use one or both of the following methods:
  - ◇ Only use functions from the base installation of Matlab.
  - ◇ Make sure that your function works on the computers of the 1109 Etcheverry Hall computer lab. All the functions available on these computers will be available to our grading system.

- For this assignment, the required submissions are:
  - ◇ `my_regression_metrics.m`
  - ◇ `my_regression_generic.m`
  - ◇ `my_regression_sincos.m`
  - ◇ `my_regression_exp.m`
  - ◇ `my_regression_polynomial.m`
  - ◇ `my_regression_perror.m`

Built-in functions that you may not use

You may not use Matlab's built-in functions `polyval`, `polyvalm`, and `polyfit`. This rule applies to each question of this lab assignment.

Note about the use of \ (backslash) versus the pseudo-inverse

Consider a system of m linear algebraic equations and n unknowns, whose matrix is the $m \times n$ matrix `A`, and whose "right-hand side" vector is the $m \times 1$ column vector `y`. Assume that this system of equations is over-determined (*i.e.* $m > n$) and has zero solution. In Matlab, both `pinv(A)*y` and `A\y` will return the vector `x` that is the approximate solution that minimizes the square error. Therefore, both `pinv(A)*y` and `A\y` should return exactly the same values. There are cases, however, where the values returned by these two commands do not exactly match, as a result of the algorithm used by the backslash operator. In this lab assignment, you should **always** use the approach that relies on `pinv` (*e.g.,* `pinv(A)*y`), and **not** the approach that relies on the backslash operator.

Sample data for the test cases

Sample data are provided to you in the form of the `mat` file named `lab09_sample_data.mat` (available on bCourses). The sample data are used in the published test cases of this lab assignment. Your functions should **not** load this `mat` file.

Where to find information about the methods used in this lab?

Least-squares linear regression was covered in lectures L22 and L23 (March 13$^{\text{th}}$ and 15$^{\text{th}}$ 2017, respectively). Chapter 13 of the textbook also covers this topic.

1. Regression metrics

Write a function with the following header:

```
function [e2, r2] = my_regression_metrics(y_data, y_predicted)
```

where:

- `y_data` is an $m \times 1$ array of class `double` whose elements are different from `NaN`, `Inf`, and `-Inf`. You can assume that $m > 1$.
- `y_predicted` is an $m \times 1$ array of class `double` that represents y-values as predicted by an approximation method (*e.g.,* least-squares linear regression). Each element of `y_predicted` is an estimation of the corresponding element in `y_data` (`y_predicted(i)` corresponds to `y_data(i)`). You can assume that all elements in `y_predicted` are different from `NaN`, `Inf`, and `-Inf`.
- `e2` is a scalar of class `double` that represents the square error E_2 associated with using the values of `y_predicted` in lieu of the values of `y_data`.
- `r2` is a scalar of class `double` that represents the coefficient of determination r^2 associated with using the values of `y_predicted` in lieu of the values of `y_data`.

Given a set of y-data $(y_i, i = \{1, 2, \ldots, m\})$ and a corresponding set of predicted y-values $(y_{\text{predicted,i}}, i = \{1, 2, \ldots, m\})$, the square error E_2 and the coefficient of determination r^2 are given by (respectively):

$$E_2 = \sum_{i=1}^{m} (y_{\text{predicted,i}} - y_i)^2 \tag{1}$$

$$r^2 = 1 - \frac{\sum_{i=1}^{m} (y_i - y_{\text{predicted,i}})^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2} \tag{2}$$

where $\bar{y}$ is the mean of the y-data.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q1_y1 and q1_ypredicted1
>> [e2, r2] = my_regression_metrics(q1_y1, q1_ypredicted1)
e2 =
   1.3782e+05
r2 =
    0.6432

>> % Data set q1_y2 and q1_ypredicted2
>> [e2, r2] = my_regression_metrics(q1_y2, q1_ypredicted2)
e2 =
   4.5853e+03
r2 =
    0.9661
```

2. Generic least-squares linear regression

In this question, you will write a function that performs least-squares linear regression on any function of the form:

$$y = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_n f_n(x) \tag{3}$$

where:

- The a_i's are the coefficients to be determined using least-squares linear regression.
- The f_i's are real-valued functions.
- The f_i's are linearly independent of each other.

More precisely, write a function with the following header:

```
function [coefficients, e2] = my_regression_generic(x_data, y_data, f)
```

where:

- `x_data` and `y_data` are two $m \times 1$ arrays of class `double` that represent two-dimensional data. In other words, these column vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 1$, and that all elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`.
- `f` is a $1 \times n$ `cell` array that contains function handles that represent the f_i's of Equation 3 (`f{i}` represents f_i). Each of these function handles takes a single input argument that is an array of class `double` of any size (and whose elements are different from `NaN`, `Inf`, and `-Inf`), and outputs a single output argument that is an array of class `double` of the same size as the input array (and whose elements are also different from `NaN`, `Inf`, and `-Inf`). You can assume that $n > 0$.
- `coefficients` is a $n \times 1$ array of class `double` that represents the a_i's of Equation 3 (`a(1)` represents a_i), fitted to the x- and y-data represented by `x_data` and `y_data` (respectively), using least-squares linear regression.
- `e2` is the square error associated with the linear regression performed by the function.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q2_x1 and q2_y1 (fit a straight line)
>> f = {@(x) x, @(x) ones(size(x))};
>> [coefficients, e2] = my_regression_generic(q2_x1, q2_y1, f)
coefficients =
  -13.2333
   -9.1484
e2 =
```

```
   8.2453e+04

>> % Data set q2_x2 and q2_y2 (fit a parabola)
>> f = {@(x) x.^2, @(x) x, @(x) ones(size(x))};
>> [coefficients, e2] = my_regression_generic(q2_x2, q2_y2, f)
coefficients =
   1.0e+04 *
    0.5727
    0.3159
   -8.0473
e2 =
   8.6093e+11
```

3. Fitting with sines and cosines

In many situations, functions can be approximated as sums of sine and cosine functions. In this question, you will approximate functions as sums of sine and cosine functions using least-squares linear regression. Given x- and y-data, you will fit a function of the form:

$$f(x) = k + \sum_{i=1}^{n} a_i \sin(ix) + \sum_{i=1}^{n} b_i \cos(ix) \tag{4}$$

where n is an integer that is greater than zero, and where the coefficients to fit are k, $a_i, i = \{1, 2, \ldots, n\}$, and $b_i, i = \{1, 2, \ldots, n\}$.

Write a function with the following header:

```
function [k, a, b] = my_regression_sincos(x_data, y_data, n)
```

where:

- `x_data` and `y_data` are two $m \times 1$ arrays of class `double` that represent two-dimensional data. In other words, these column vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 1$, and that all elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`.
- `n` is a scalar of class `double` that represents n in Equation 4. You can assume that `n` is an integer such that `n > 0`.
- `k` is a scalar of class `double` that represents k in Equation 4.
- `a` and `b` are $n \times 1$ arrays of class `double` such that `a(i)` represents a_i in Equation 4 and `b(i)` represents b_i in Equation 4.

`k`, `a`, and `b` represent the coefficients of Equation 4, fitted to the x- and y-data represented by `x_data` and `y_data` (respectively), using least-squares linear regression.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q3_y1 versus q3_x1
>> [k, a, b] = my_regression_sincos(q3_x1, q3_y1, 2)
k =
    1.2404
a =
    0.7688
   -0.0490
b =
   -0.5525
    0.1406

>> [k, a, b] = my_regression_sincos(q3_x1, q3_y1, 20)
k =
    1.1877
a =
   1.0e+04 *
    0.0001
   -0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0001
    0.0005
    0.0040
    0.1902
   -1.4587
    4.1677
   -5.8100
    3.6291
   -0.1949
    0.1462
   -2.4351
    2.9331
   -1.4440
    0.2717
b =
   1.0e+04 *
   -0.0001
    0.0000
    0.0000
    0.0000
    0.0000
    0.0001
    0.0002
    0.0005
    0.0020
    0.0320
    0.1840
```

```
   -1.8497
    4.9900
   -5.6550
    0.5660
    6.0300
   -7.4902
    4.2312
   -1.1498
    0.1086

>> % Data set q3_y2 versus q3_x2
>> [k, a, b] = my_regression_sincos(q3_x2, q3_y2, 2)
k =
    0.5807
a =
   -0.0325
    0.2035
b =
    0.3768
   -0.3800

>> [k, a, b] = my_regression_sincos(q3_x2, q3_y2, 20)
k =
    0.5875
a =
   -0.0408
    0.1906
    0.1026
    0.0071
    0.1579
    0.0340
   -0.0472
    0.1013
    0.0288
   -0.0026
    0.1142
    0.0483
    0.0124
    0.0639
    0.0201
    0.0132
    0.0379
    0.0383
    0.0466
    0.0188
b =
    0.3874
   -0.3768
   -0.0586
    0.0786
   -0.0759
    0.0043
```

```
 0.0136
-0.0394
-0.0345
-0.0401
 0.0054
-0.0047
-0.0185
 0.0502
-0.0038
-0.0377
 0.0390
-0.0122
-0.0243
 0.0575
```

Figure 1 shows the data used in the test cases above, as well as the corresponding function f fitted for different values of n.

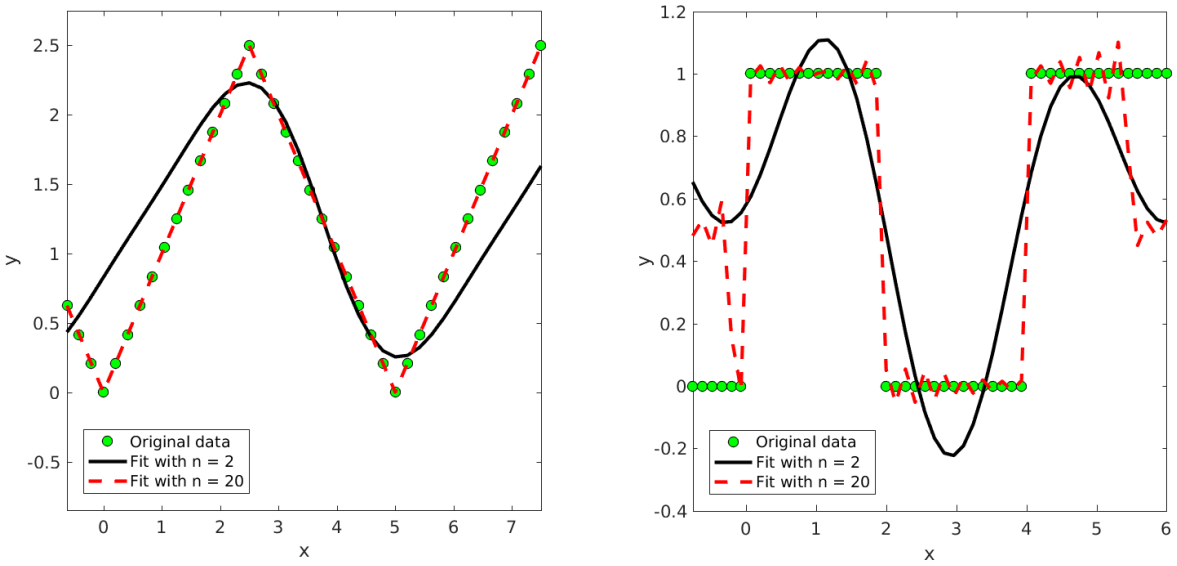

Figure 1: Left: Data set `q3_y1` versus data set `q3_x1` (green dots) and function f fitted with different values of n. See the main text for details. Right: same as the left-hand side, but for the data set `q3_y2` versus `q3_x2` instead

4. Exponential decay

A number of physical processes can be described using a law of the form:

$$y = a \exp\left(\frac{-x}{\lambda}\right) \tag{5}$$

where x and y are two variable quantities, and a and λ are two real constants.

Write a function with the following header:

```
function [a, lambda] = my_regression_exp(x_data, y_data)
```

where:

- `x_data` and `y_data` are two $m \times 1$ arrays of class `double` that represent two-dimensional data. In other words, these column vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 1$, and that all elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`. Additionally, you can assume that all the elements of `y_data` are positive.
- `a` and `lambda` are scalars of class `double` that represent a and λ in Equation 5, respectively, fitted to the x- and y-data represented by `x_data` and `y_data` (respectively), using least-squares linear regression.

To calculate a and λ using linear regression, you will have to:

1. Take the natural log of Equation 5.
2. Calculate the values of $\ln(a)$ and $(-1/\lambda)$ by fitting them using least-squares linear regression on the new equation derived in the previous step.
3. Calculate (from the values of $\ln(a)$ and $(-1/\lambda)$ thus calculated) the values of a and λ.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q4_y1 versus q4_x1
>> [a, lambda] = my_regression_exp(q4_x1, q4_y1)
a =
    2.4567
lambda =
    7.2777

>> % Data set q4_y2 versus q4_x2
>> [a, lambda] = my_regression_exp(q4_x2, q4_y2)
   a =
    6.4740
lambda =
    4.4736
```

5. Fitting polynomials

A real polynomial of degree p is a real-valued function P defined on $\mathbb{R}$ and of the form:

$$P(x) = a_p x^p + a_{p-1} x^{p-1} + \cdots + a_1 x + a_0 \tag{6}$$

where $a_0, a_1, a_2, ..., a_p$ are constant real coefficients and $a_p \neq 0$.

In this question, you will write a function that determines the real polynomial of "reasonable degree" (see below) that best fits a set of two-dimensional data. Note that if you use least-squares linear regression to fit two polynomials to the same data set, one polynomial of degree p and the other one of degree $p+1$, then the square error associated with the fitted polynomial of degree $p+1$ is equal to or smaller than the square error associated with the fitted polynomial of degree p.

Write a function with the following header:

```
function [coefficients] = my_regression_polynomial(x_data, y_data, ...
                                                   degree_min, degree_max)
```

where:

- `x_data` and `y_data` are two $m \times 1$ arrays of class `double` that represent two-dimensional data. In other words, these column vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 1$, and that all elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`.
- `degree_min` and `degree_max` are scalars of class `double` that represent integers that are greater than zero, and such that `degree_min` $\leqslant$ `degree_max`.
- `coefficients` is a $(p+1)\times 1$ array of class `double` such that `degree_min` $\leqslant$ `p` $\leqslant$ `degree_max`. The elements of `coefficients` are the coefficients of a polynomial of degree p (`coefficient(i)` represents a_{i-1} in Equation 6). This polynomial should be the polynomial that best fits, using least-squares linear regression, the x- and y-data data represented by `x_data` and `y_data` (respectively), among all the polynomials of degree `degree_min` to `degree_max` (both boundaries included), and such that:
  - $\diamond$ p is as small as possible; and
  - $\diamond$ the magnitude of the reduction of square error associated with fitting a polynomial of degree $p+1$ instead of a polynomial of degree p is less than 10% of the square error associated with fitting a polynomial of degree p.

  The motivation behind these conditions is to compromise between (i) the degree of the polynomial that is fitted to the data (we want to keep the degree reasonably small), while (ii) keeping the square error reasonably low. If increasing the degree of the polynomial

does not yield a large enough decrease in the square error, then we estimate that increasing the degree of the polynomial is unnecessary. The test cases presented below illustrate the significance of this compromise (see Figure 2).

In this question, if you use least-squares linear regression to fit a polynomial of degree p to the data and the regression yields $a_p = 0$, you can still consider the corresponding polynomial to be of degree p.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q5_y1 versus q5_x1
>> coeffs = my_regression_polynomial(q5_x1, q5_y1, 1, 8)
coeffs =
    6.8734
   -1.8757
   -1.6070
    3.3956

>> % Data set q5_y2 versus q5_x2
>> coeffs = my_regression_polynomial(q5_x2, q5_y2, 3, 8)
coeffs =
   1.0e+04 *
    1.0475
   -0.2903
   -0.3191
    0.0099
    0.0102
```

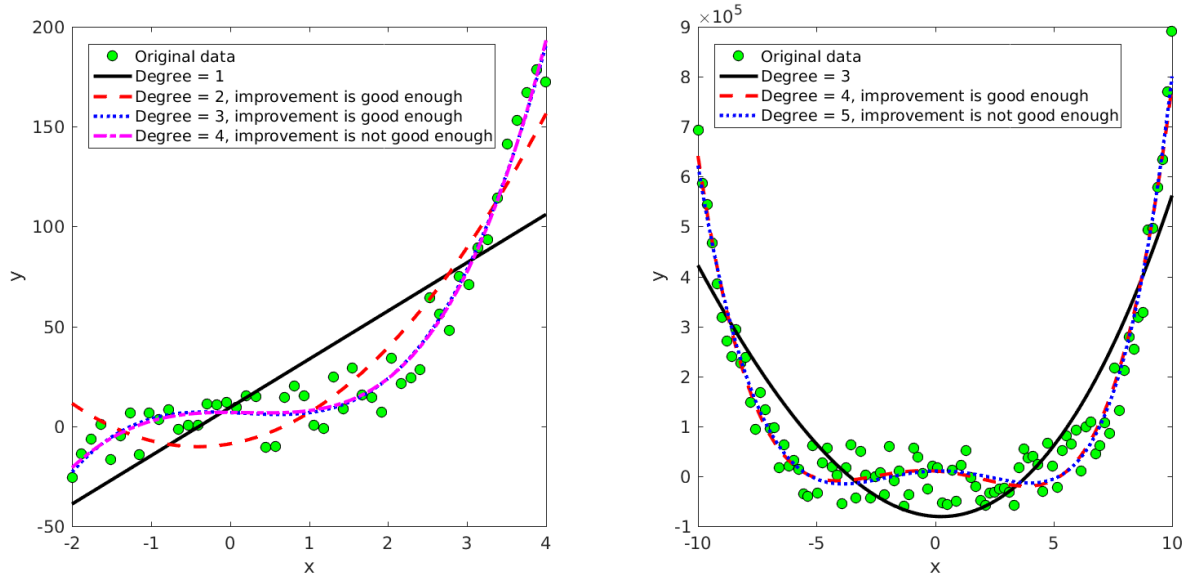

Figure 2: Left: Data set `q5_y1` versus data set `q5_x1` (green dots) and polynomial fits of various degrees: from degree `degree_min` to degree `p+1`, where `p` is the "reasonable degree". The figure illustrates that using polynomials of increasing degrees improves the goodness of the fit, but that beyond a certain degree (here degree `p`), the improvements are marginal. See main text for details, including the significance of `p`. Right: same as the left-hand side, but for the data set `q5_y2` versus `q5_x2` instead.

6. Fitting y = ax with for different error definitions

Imagine that you want to determine the value of the coefficient a such that the line of equation $y = ax$ best fits a set of x- and y-data (x_i and $y_i, i = \{1, 2, \ldots, m\}$). When doing least-squares linear regression, one calculates the value of a that minimizes the square error E_2:

$$E_2 = \sum_{i=1}^{m} (ax_i - y_i)^2 \tag{7}$$

One can generalize the definition of the square error to a "p-error" E_p as follows:

$$E_p = \sum_{i=1}^{m} (ax_i - y_i)^p \tag{8}$$

where p is an **even** integer that is greater than zero. Note that the square error corresponds to $p = 2$. As illustrated by the test cases shown below (see Figure 3), the higher the value of p, the more importance is given to the "outliers" of the data set.

In this question, you will write a function that determines the value of a that, given x- and y-data and a value of p, minimizes the "p-error" E_p. More precisely, write a function with the following header:

```
function [a, p_error] = my_regression_perror(x_data, y_data, p, tolerance)
```

- `x_data` and `y_data` are two $m \times 1$ arrays of class `double` that represent two-dimensional data. In other words, these column vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 1$, and that all elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`. `x_data` and `y_data` represent the x- and y-data (respectively) to which we try to fit the line of equation $y = ax$.
- `p` is a scalar of class `double` that represents p in Equation 8. `p` is an even integer that is greater than zero.
- `tolerance` is a scalar of class `double` that represents a positive number. See below for a more detailed description.
- `a` is a scalar of class `double` that represents the number a such that the `p`-error corresponding to fitting the line of equation $y = ax$ is minimum.
- `p_error` is a scalar of class `double` that represents the p-error E_p corresponding to the value of `a` calculated by your function.

If $p = 2$, your function should calculate a according to the following equation, derived in class (lecture L22):

$$a = \frac{\sum_{i=1}^{m} x_i y_i}{\sum_{i=1}^{m} x_i^2} \tag{9}$$

In any other case, you should use the Newton-Raphson method to determine the correct value of a, using the following approach:

1. Derive, on paper, the expressions of the first and second derivatives of E_p with respect to a: E_p' and E_p'', respectively.
2. Use the Newton-Raphson method to find the value of a that minimizes E_p by solving the equation $E_p'(a) = 0$. You can assume that there is a unique a such that $E_p'(a) = 0$, and that this value of a indeed corresponds to the minimum of E_p. Use the value of a calculated using Equation 9 as the initial guess for the Newton-Raphson method, and **iterate this method until, for two consecutive guesses a_{i-1} and a_i, respectively, the following condition holds: $|a_i - a_{i-1}| \leqslant$ `tolerance`. Use a_i as the value of a.**

To help you with this question, we provide you with the expression of E_p', which holds when

$p \geqslant 2$:

$$E_p'(a) = \sum_{i=1}^{m} p x_i (a x_i - y_i)^{p-1} \tag{10}$$

The derivation of E_p'' is left for you to do.

Test cases:

```
>> % Load the sample data
>> load('lab09_sample_data.mat');

>> % Data set q6_y1 versus q6_x1
>> [a, e] = my_regression_perror(q6_x1, q6_y1, 2, 1e-6)
a =
   20.8767
e =
   1.0256e+05

>> [a, e] = my_regression_perror(q6_x1, q6_y1, 4, 1e-6)
a =
   25.9760
e =
   1.6489e+09

>> [a, e] = my_regression_perror(q6_x1, q6_y1, 8, 1e-6)
a =
   29.1819
e =
   5.8801e+17

>> % Data set q6_y2 versus q6_x2
>> [a, e] = my_regression_perror(q6_x2, q6_y2, 2, 1e-6)
a =
  -13.7061
e =
   8.6205e+04

>> [a, e] = my_regression_perror(q6_x2, q6_y2, 4, 1e-6)
a =
  -17.5203
e =
   7.6355e+08

>> [a, e] = my_regression_perror(q6_x2, q6_y2, 8, 1e-6)
a =
  -18.8711
e =
   8.1935e+16
```

Figure 3 summarizes the data and the fitted coefficients corresponding to the test cases

shown above. This figure illustrates that the higher the value of p, the more the value of the fitted coefficient a is influenced by the few "outliers" of the data set.

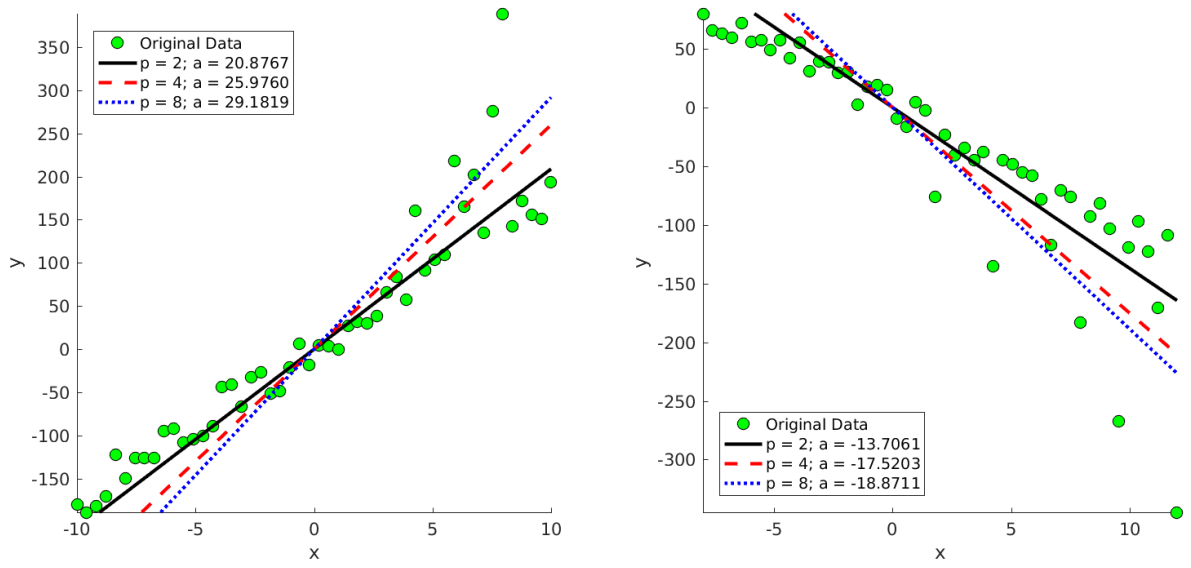

Figure 3: Left: Data set `q6_y1` versus data set `q6_x1` (green dots) and lines of equation $y = ax$ where a was fitted according to different p-errors. See main text for details. Right: same as the left-hand side, but for the data set `q6_y2` versus `q6_x2` instead.