Assignment8

October 6, 2016

EE1103 Numerical Methods
Assignment 8
Rajat Vadiraj Dwaraknath - EE16B033

We first import numpy and matplotlib using pylab, and increase the size of the plots.

```
In [1]: # Importing matplotlib and numpy directly
        # Python 2.7
        %pylab inline
        rcParams['figure.figsize'] = 15, 12 # Increase size of plots
        rcParams['font.size'] = 20 # Increase font size in plots
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: def plotData(s):
            plt = matplotlib.pyplot
            # Open the file
            fo = open(s, 'r')

            # Read lines ignoring those that begin with '#'
            lines = fo.read().split("\n")[:-1]
            lines = [x.strip() for x in lines if x[0] != '#']

            # Close the file
            fo.close()

            # List of time instants
            t = []

            # List of values
            states = []

            for i in lines[:-1]:

                # Separate the columns
```

```
        values = i.split("\t")

        # First column contains time instants
        t.append(float(values[0]))

        # Get the columns
        states.append([float(i) for i in values[1:]])


    # Calculate slope and intercept of line of best fit
    x0 = t[1]
    x1 = t[2]
    y0 = states[1][1]
    y1 = states[2][1]

    a1 = (y1-y0)/(x1-x0)
    a0 = y0 - a1*x0


    # Plot the data
    plt.scatter(t,[i[0] for i in states])
    line = plt.plot(t,[i[1] for i in states])
    plt.setp(line, color = 'r', linewidth = 2);

    plt.legend(["$y = $"+str(a1)+"*$x + $"+str(a0)+
                "\n$r^{2} = $"+lines[-1]], loc = 2);
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title(s+" Y vs X")
```

0.0.1 Answers to the questions

Please run make before running this notebook as the graphs cannot be plotted without the text files generated by make.

1 The absolute value of offsets function is not differentiable at $x = 0$, which makes the optimization procedure more difficult. The square of the offsets is however differentiable everywhere, which allows one to find a closed-form solution for the optimization problem. This is one of the reasons for using the square offset instead of the absolute value of the offset as the cost function.

Another reason is that the square of offsets is analogous to the Euclidean distance in space, which is more natural than the Manhattan distance which is analogous to the absolute value of offsets.

2 The vertical offsets are used over the perpendicular offsets because the regression is done assuming that the value of y must be predicted from a given value of x. This is called finding the regression line of y on x. If the predictions have to be made for x, then the horizontal distances would be used. The perpendicular distance is used when there is no distinction between x and

y as the predictor and predicted values. The vertical offsets also result in a simpler mathematical form for the solution of the optimzation problem.

3 Given a set of n data points (x_i, y_i), the problem of linear regression can be reduced to finding the two parameters which describe a line, its slope m and the y-intercept c which results in the best fit for those data points. This condition on the parameters can be imposed using a loss(or error) function, which should be minimized. The loss function in the least-squares method is

$$L(m,c) = \sum_{i=1}^{n}(y_i - mx_i - c)^2$$

which is the sum of the squares of the vertical errors. This optimization problem can also be thought of as a projection of a vector onto a subspace. Consider the ideal scenario where each of the data points pass through our line. This yields the following n equations which must be satisfied by m and c.

$$y_i = mx_i + c$$

or

$$mx_i + c = y_i$$

This can be expressed in matrix form as

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

which can be written succinctly as

$$Au = \beta$$

This equation need not always have a solution for u as the data points need not all lie on the same line. If it does have a solution, it means that β is in the column space of A. If it does not, it means that β is not in the column space of A. In this case, a solution can be found by finding the vector in the columnspace which is closest to β. This is the projection of β onto the column space of A. Let us call this projection $\hat{\beta}$. Therefore, we must now solve

$$Au = \hat{\beta}$$

It can be shown easily that minimizing $L(c,m)$ and solving the above matrix equation are equivalent.

Let us now solve this matrix equation. The projection matrix of A onto its column space is given by

$$P = A(A^TA)^{-1})A^T$$

which means

$$\hat{\beta} = P\beta$$

Therefore our main equation becomes

$$\begin{gathered}Au = A(A^TA)^{-1})A^T\beta \\ \implies A^TAu = A^T\beta \\ \implies u = (A^TA)^{-1}A^T\beta\end{gathered}$$

Now, upon performing the matrix multiplications, we get

$$A^TA = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i \\ \Sigma x_i & n \end{bmatrix}$$

and

$$A^T\beta = \begin{bmatrix} \Sigma x_iy_i \\ \Sigma y_i \end{bmatrix}$$

This finally leads to the following equation for finding u

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i \\ \Sigma x_i & n \end{bmatrix}^{-1} \begin{bmatrix} \Sigma x_iy_i \\ \Sigma y_i \end{bmatrix}$$

To explicitly solve this equation, we need to invert A^TA. We first find its determinant as follows

$$det(A^TA) = n\Sigma x_i^2 - (\Sigma x_i)^2 = D(say)$$

Its adjoint can also be easily found

$$adj(A^TA) = \begin{bmatrix} n & -\Sigma x_i \\ -\Sigma x_i & \Sigma x_i^2 \end{bmatrix}$$

Its inverse is therefore

$$(A^TA)^{-1} = \frac{1}{D}adj(A^TA)$$

The solution to the matrix equation is

$$\begin{gathered}\begin{bmatrix} m \\ c \end{bmatrix} = \frac{1}{D}\begin{bmatrix} n & -\Sigma x_i \\ -\Sigma x_i & \Sigma x_i^2 \end{bmatrix}\begin{bmatrix} \Sigma x_iy_i \\ \Sigma y_i \end{bmatrix} \\ \implies \begin{bmatrix} m \\ c \end{bmatrix} = \frac{1}{D}\begin{bmatrix} n\Sigma x_iy_i - \Sigma x_i\Sigma y_i \\ \Sigma x_i^2\Sigma y_i - \Sigma x_i\Sigma x_iy_i \end{bmatrix}\end{gathered}$$

The final result is

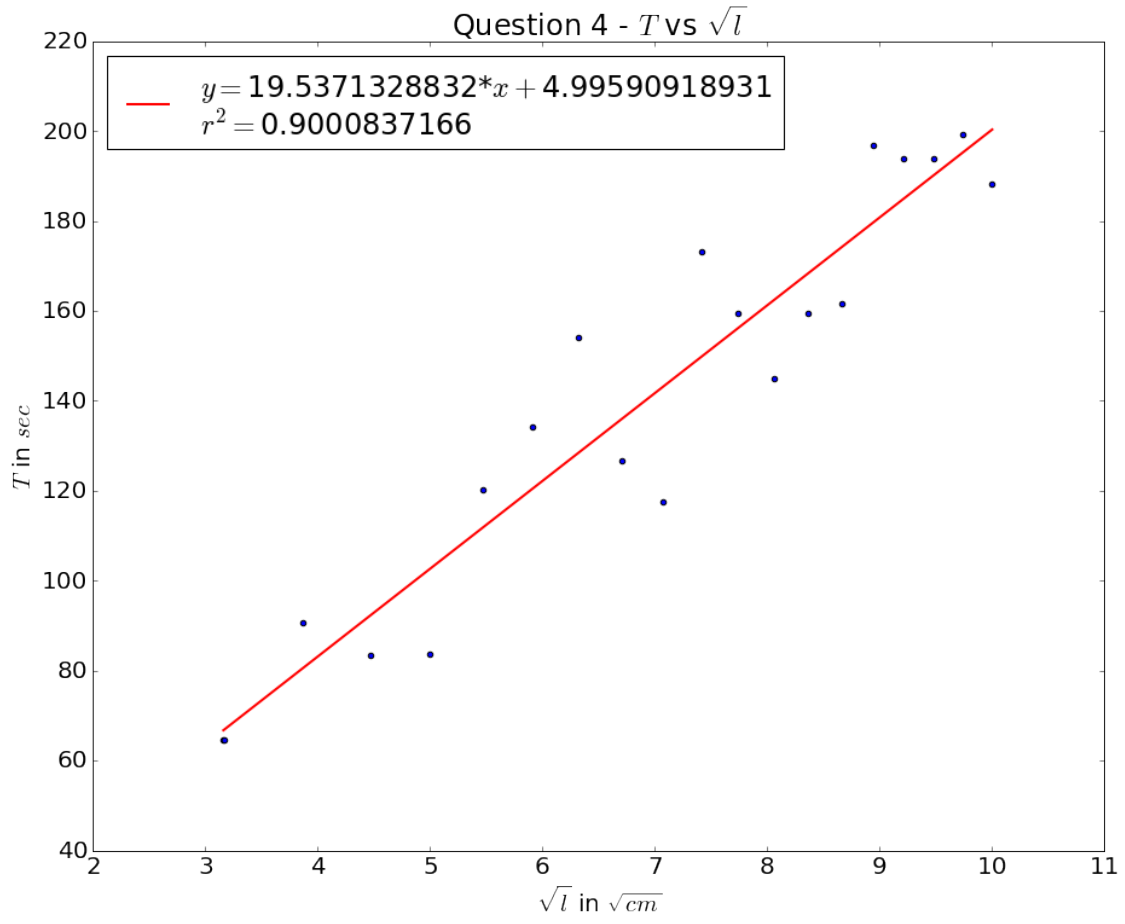
$$m = \frac{n\Sigma x_iy_i - \Sigma x_i\Sigma y_i}{n\Sigma x_i^2 - (\Sigma x_i)^2}$$

and

$$c = \frac{\Sigma y_i}{n} - m\frac{\Sigma x_i}{n}$$

4 The plot of T vs $\sqrt{l}$ where T is the time period of the pendulum and l is the cord length containing the data points and the best fit line is given below:

```
In [3]: plotData("Q4out.txt")
        title("Question 4 - $T$ vs $\sqrt{l}$")
        xlabel("$\sqrt{l}$ in $\sqrt{cm}$")
        ylabel("$T$ in $sec$")
        show()
```



The slope M of the graph between T and $\sqrt{l}$ is

$$M = \frac{2\pi}{\sqrt{g}}$$

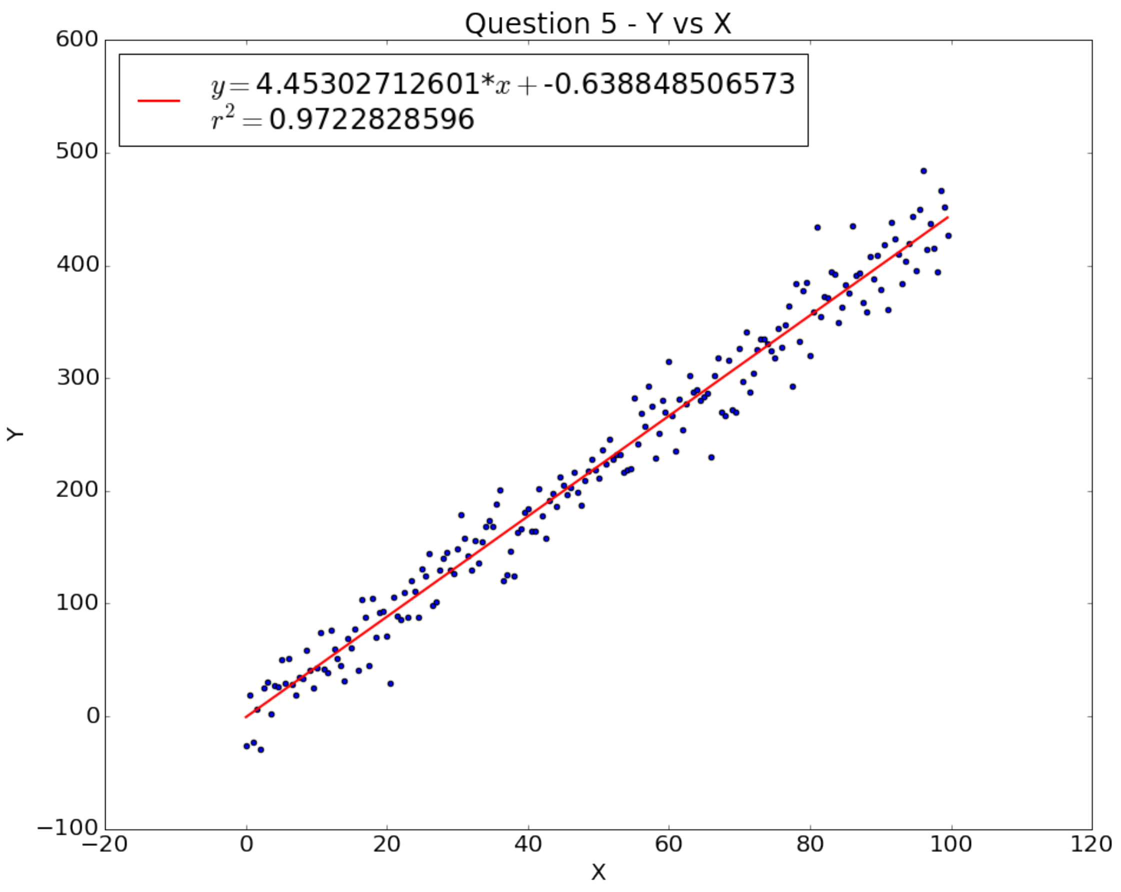
where g is the acceleration due to gravity. Using the value of m from the graph, we obtain

$$g = 0.1034 cm \cdot s^{-2}$$
$$= 10.34 m \cdot s^{-2}$$

which is close to the theoretical value of $9.81 m \cdot s^{-2}$

5 The plot of data and the linear best fit curve is given below:

```
In [4]: plotData("Q5out.txt")
        title("Question 5 - Y vs X")
        show()
```



Source code of C files

leastSquares.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "leastSquares.h"

/**
  Implementation of Least-Squares Linear regression.

Author: Rajat Vadiraj Dwaraknath
Date: 5th October 2016
 */
```

```c
// Perform linear least squares regression on the data points given by x and y,
// and output the slope, intercept, and correlation coefficient into a.
void leastSquares(double *x, double *y, int n, double *a){

    // variables for the sums
    double sumX = 0.0, sumY = 0.0, sumXY = 0.0, sumXSqr = 0.0, sumYSqr = 0.0;
    int i;

    // Calculate the sums
    for(i=0;i<n;i++){
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i]*y[i];
        sumXSqr += x[i]*x[i];
        sumYSqr += y[i]*y[i];
    }

    // Calculate the slope, intercept, and correlation coefficient
    double a1 = (n*sumXY - sumX*sumY)/(n*sumXSqr - pow(sumX,2));
    double a0 = sumY/n - a1*sumX/n;
    double rsqr = (n*sumXY - sumX*sumY)/sqrt((n*sumXSqr - pow(sumX,2))
                *(n*sumYSqr - pow(sumY,2)));
    rsqr = pow(rsqr,2);

    // Put the values into the given pointer
    a[0] = a0;
    a[1] = a1;
    a[2] = rsqr;
}

// Predict the value of y for some x given a regression line described using 'a'.
double predict(double x, double *a){
    return a[0] + a[1]*x;
}
```

assignment8.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "leastSquares.h"

/**
  Using Least-Squares linear regression to solve questions 4 and 5.

Author: Rajat Vadiraj Dwaraknath
Date: 5th October 2016
```

```
 */

int main(int argc, char **argv){
    if(argc!=2){
        printf("Invalid input. Usage: %s <input file path>\n", argv[0]);
        return 0;
    }

    // Reading a file
    char buff[255];
    int n = 0;

    FILE *infile;
    infile = fopen(argv[1],"r");

    // Get the number of lines
    while(fgets(buff,255,infile)){
        n++;
    }

    // Create arrays for x and y values
    double *x = (double*)malloc(sizeof(double)*n),
           *y = (double*)malloc(sizeof(double)*n);

    fclose(infile);
    infile = fopen(argv[1],"r");

    // Get the values from each line
    int i=0;
    while(fgets(buff,255,infile)){
        sscanf(buff,"%lf\t%lf",&x[i],&y[i]);
        i++;
    }

    // Linear regression
    double a[3] = {0.0, 0.0, 0.0};
    leastSquares(x,y,n,a);

    // Output data
    for(i=0;i<n;i++){
        printf("%.4f\t%.13f\t%.13f\n",x[i],y[i],predict(x[i],a));
    }
    printf("%.10f\n",a[2]);

    // Free dynamically allocated space
    free(x);
    free(y);
    return 0;
}
```