Sageable Mathematics

February 9, 2019

Sageable Mathematics is a project that introduces SageMath (or Sage) to students and encourages them to program in Sage. Each student (or each small group of students) will pick a mathematics concept and implement the computation in Sage. The goal is to experience programming through Sage and also gain a deeper insight into the concept.

This document is a gallery of students' work in Math316: Practice of Applied Mathematics at National Sun Yat-sen University, in 2019 Spring.

Contents

1 Linear Regression

—by Jephian Lin

Overview. Given a set of N data (x_i, y_i) for $i = 1, \dots, N$, *linear regression* aims to find a line

$$y = ax + b$$

that best describes the data.

That is, the goal is to find two values a and b such that

$$\sum_{i}^{N} (y_i - ax_i - b)^2$$

is minimized.

Algorithm.

1. Create an $N \times 2$ matrix $A = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}$ and a vector $v = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$.

2. Then compute
$$\begin{bmatrix} a \\ b \end{bmatrix} = (A^\top A)^{-1} A^\top v$$

   Note: If $(A^\top A)^{-1}$ does not exist, use the Penrose–Moore pseudo inverse instead.

Explanation. The goal is to solve the equation
$$Ax = v, \text{where} x = \begin{bmatrix} a \\ b \end{bmatrix}$$
for x.

The equation does not always have a solution. If not solvable, we find a vector x such that
$$|Ax - v|^2 = \sum_i^N (y_i - ax_i - b)^2$$
is minimized.

To do so, let v_0 be the orthogonal projection of v onto the column space of A. By the formula of orthogonal projection
$$v_0 = A(A^\top A)^{-1} A^\top v.$$

Now solve $Ax = v_0$ and get $x = (A^\top A)^{-1} A^\top v$. Therefore,
$$\begin{bmatrix} a \\ b \end{bmatrix} = (A^\top A)^{-1} A^\top v.$$

Implimentation.

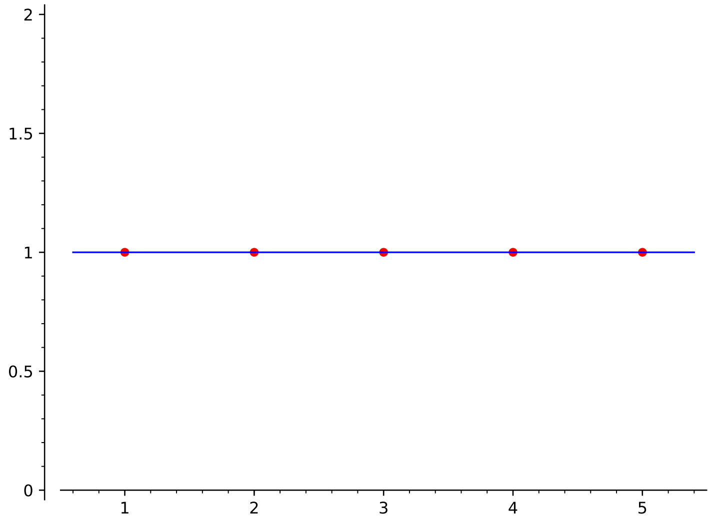
```
def linear_regression(data, draw=False):
    """
    Input:
        data: a list of pairs [(x1,y1), ..., (xN,yN)]
    Output:
        Output [a,b] so that the line y = ax + b
        is the best fitting line for the data.
        When draw == True,
        create a graphical illustration p and return [a,b,p].
    """
    NN = len(data)
    ### x_list = [x1, x2, ..., xN]
```

```
    x_list = [p[0] for p in data]
    ### one_list = [1,1, ..., 1]
    one_list = [1] * NN
    ### y_list = [y1, y2, ..., yN]
    y_list = [p[1] for p in data]
    ### define A and v as described in the algorithm
    A = matrix([x_list, one_list]).transpose()
    v = matrix([y_list]).transpose()
    AT = A.transpose()
    ATA = AT * A
    ATAinv = ATA.pseudoinverse()
    ans = ATAinv * AT * v
    a, b = ans.transpose()[0]

    if draw:
        x_min = min(x_list)
        x_max = max(x_list)
        x_range = x_max - x_min
        x = var('x')
        pic = (a*x + b).plot(xmin=x_min-0.1*x_range,
                             xmax=x_max+0.1*x_range)
        pic += point(data, rgbcolor='red', size=30)

        return [a,b,pic]

    return [a,b]
```

Examples.

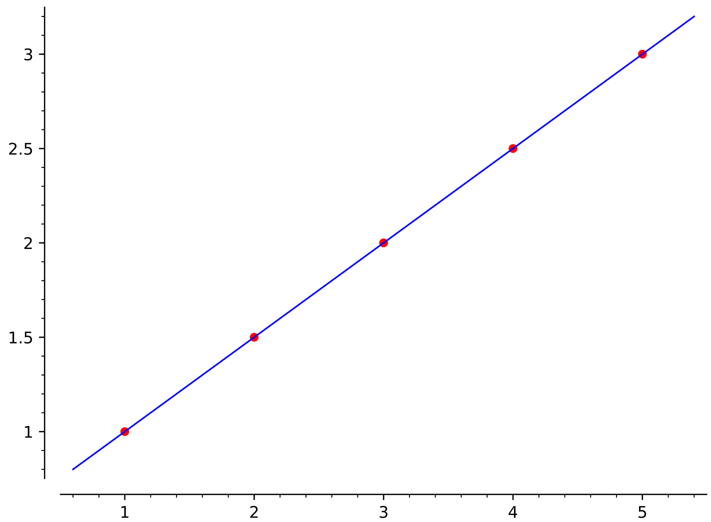
```
### horizontal data
data = [(1,1),(2,1),(3,1),(4,1),(5,1)]

a,b,p = linear_regression(data,True)
```

Then $a = 0$, $b = 1$, and p is the figure below.



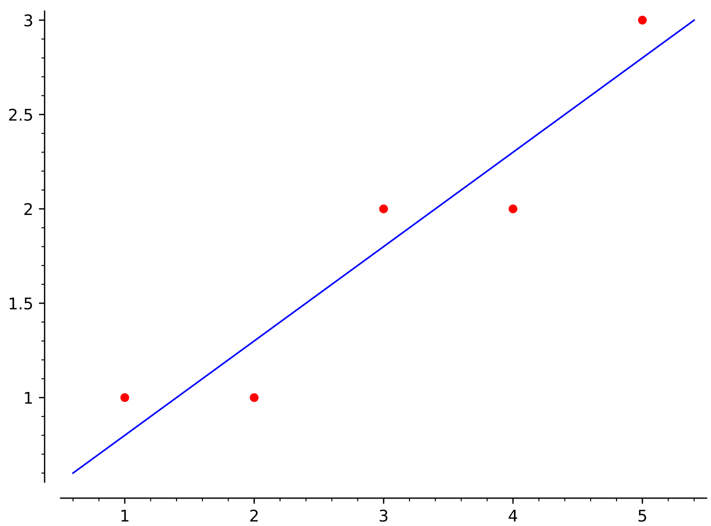

```
### linear data
data = [(1,1),(2,1.5),(3,2),(4,2.5),(5,3)]

a,b,p = linear_regression(data,True)
```

Then $a = 0.500000000000000$, $b = 0.500000000000000$, and p is the figure below.




```
### non-linear data
data = [(1,1),(2,1),(3,2),(4,2),(5,3)]

a,b,p = linear_regression(data,True)
```

Then $a = \frac{1}{2}$, $b = \frac{3}{10}$, and p is the figure below.




```
### almost-linear data
import numpy as np
x = np.linspace(1,5,50)
y = x*0.2 + 1 + 0.1*np.random.randn(50)
data = list(zip(x,y))

a,b,p = linear_regression(data,True)
```

Then $a = 0.2147921121583119$, $b = 0.9506433754276478$, and p is the figure below.