Lecture 10:

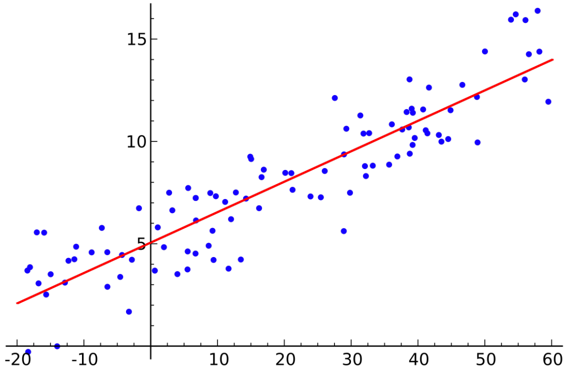
Least Squares Method

CS 111: Intro to Computational Science

Spring 2023

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB



Administrative

- New homework due Monday
- Lab tomorrow

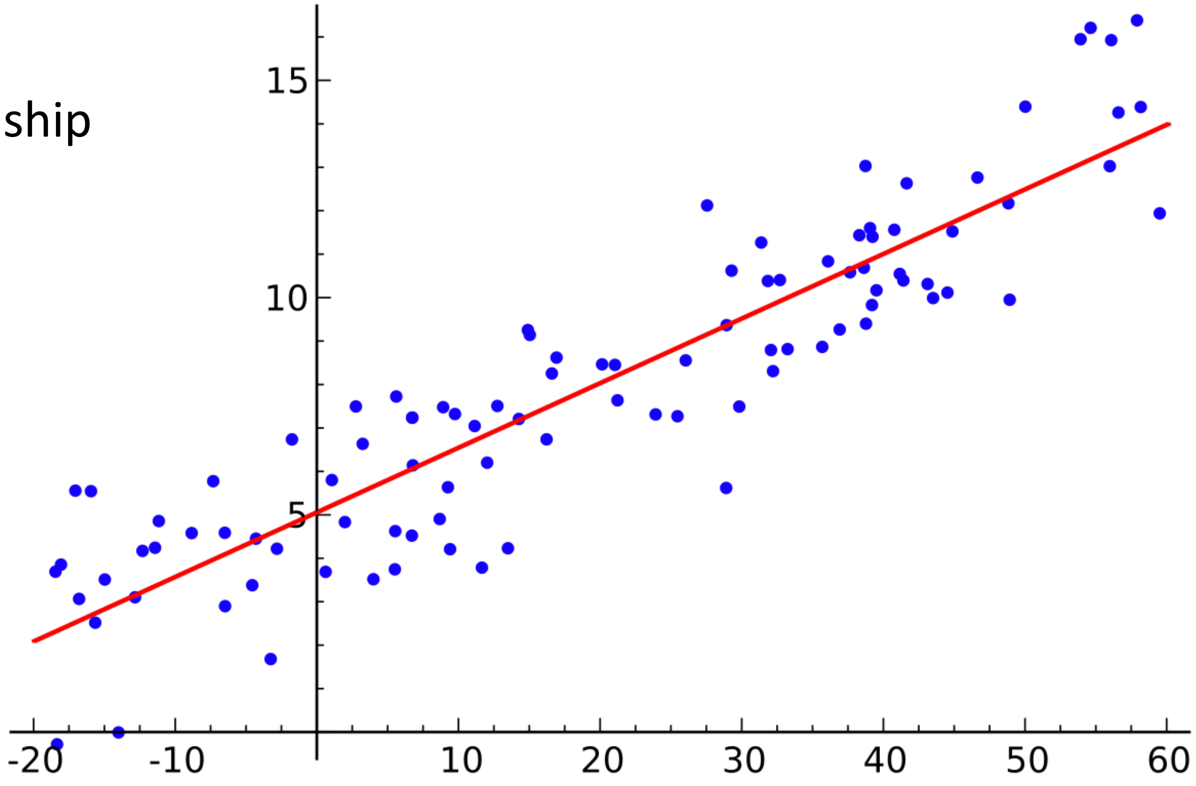
Least Squares Method

A form of mathematical ***regression analysis*** that finds the ***line of best fit*** for a ***set of data***

- Provides a visual demonstration of the relationship between the data points

- Often we refer to the
  x-axis as the Independent Variables
  & y-axis as the Dependent Variables

- Can be used to help forecast data



Example…

Her survey reveals:

- $x_1 = 1237$ ft.
- $x_2 = 1941$ ft.
- $x_3 = 2417$ ft.

To confirm these, she climbs each

- $x_2 - x_1 = 711$ ft.
- $x_3 - x_1 = 1177$ ft.
- $x_3 - x_2 = 475$ ft.

Ax ≅ b

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cong \begin{pmatrix} 1237 \\ 1941 \\ 2417 \\ 711 \\ 1177 \\ 475 \end{pmatrix}$$

This is an OVERDETERMINED SYSTEM!

Can we use `npla.solve()` to find **x**?

Data Fitting

- Given data points (t_i, y_i) (where i= 1 to m)
  - So, **t** is the horizontal axis

- We want to find a vector **x** of parameters that give us the "best fit" to the data by the model function ***f*(t, x)** as

$$\min \sum_{i=1}^{m} (y_i - f(t_i - x))^2$$

Data Fitting

- We will assume the model is **linear**
  (i.e. a *linear combination of parameters*), i.e:

$$\underbrace{f(\mathbf{t}, x) = x_0 + x_1\mathbf{t}}\ + x_2\mathbf{t}^2 + x_3\mathbf{t}^3 + \ldots + x_n\mathbf{t}^n$$

Called the general linear model

- If we think we can fit our data into a model that's a straight-line, then all we need are the **first 2 components, x_0 and x_1**
  - So: $f(\mathbf{t}, x) = x_0 + x_1\mathbf{t}$
  - This means: $x_2, x_3, \ldots x_n$ are all equal **0**

Data Fitting

- We will assume the model is **linear**
  (i.e. a *linear combination of parameters*), i.e:

$$\underbrace{f(\mathbf{t}, x) = x_0 + x_1\mathbf{t} + x_2\mathbf{t}^2}\, + x_3\mathbf{t}^3 + \ldots + x_n\mathbf{t}^n$$

- If instead, we want to fit data into a parabola, then we need the **first 3 components**, etc..
  - So: $f(\mathbf{t}, x) = x_0 + x_1\mathbf{t} + x_2\mathbf{t}^2$
  - This means: $x_3, x_4, \ldots x_n$ are all equal **0**

Revelation!

We can express these kinds of models with a linear system of equations using **matrices**.

LSQ Method

- We're trying to fit a line **$f(t, x) = x_0 + x_1t + x_2t^2 + x_3t^3 + \ldots + x_nt^n$** thru a set of points
- Of course, the line is likely to miss some of the points
- These missed points (collectively called *deviations*) can be represented as a sum of squares:

$$R^2 = \sum_{i=0}^{n} [y_i - f(t_i, x_i)]^2$$

- The idea is to minimize these deviations, so we'll solve for $\frac{\partial R^2}{\partial x_i} = 0$
- As an example, for a straight line ($y = x_0 + x_1t$) fitting:

$$x_0 = \frac{\bar{y}(\sum_{i=1}^{n} x_i^2) - \bar{x}(\sum_{i=1}^{n} x_i y_i)}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \qquad x_1 = \frac{(\sum_{i=1}^{n} x_i y_i) - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

- Luckily, we can just use a built-in **numpy** function! 😉

LSQ Setup

Example: If you want to fit to a STRAIGHT LINE (i.e. *1st order polynomial*):

- You want to fit to the model: $f(t, x) = x_0 + x_1 t$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \cong \begin{pmatrix} 2.9 \\ 2.7 \\ 4.8 \\ 5.3 \\ 7.1 \\ 7.6 \\ 7.7 \\ 8.0 \\ 9.4 \end{pmatrix}$$

Indicates the y-axis values

Indicates the t-axis values

LSQ Setup

Example: If you want to fit to a STRAIGHT LINE (i.e. *1st order polynomial*):

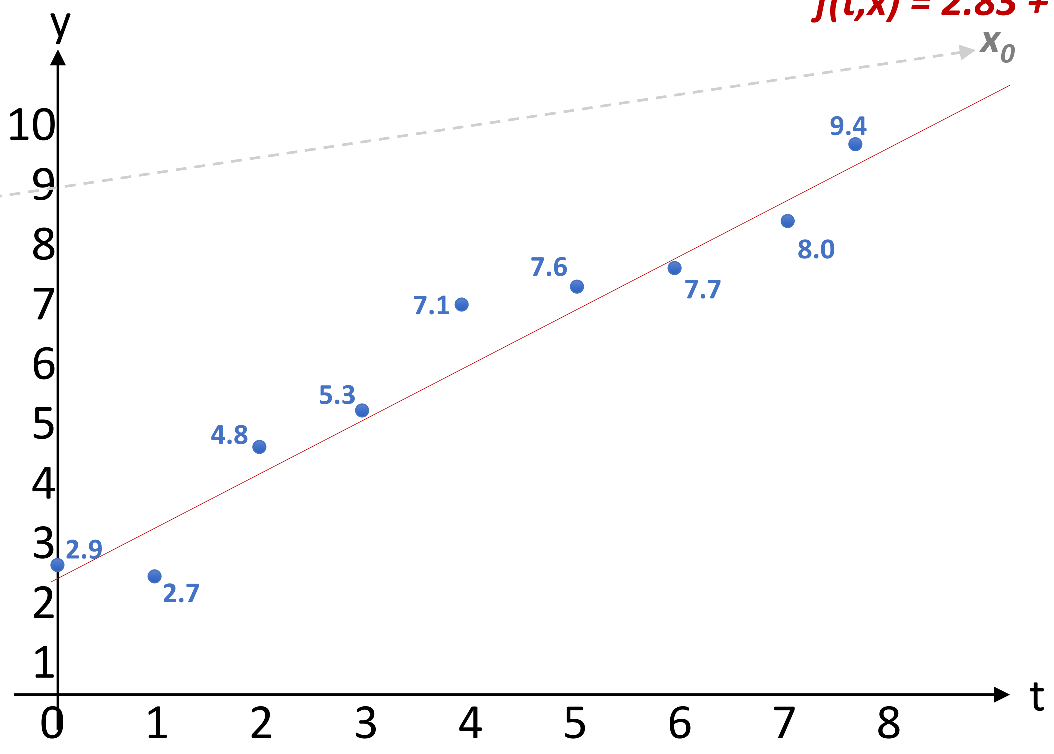
- You want to fit to the model: $f(t, x) = x_0 + x_1 t$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \cong \begin{pmatrix} 2.9 \\ 2.7 \\ 4.8 \\ 5.3 \\ 7.1 \\ 7.6 \\ 7.7 \\ 8.0 \\ 9.4 \end{pmatrix}$$

Solved using LSQ:

f(t,x) = 2.83 + 0.83 t

x_0 x_1



Computationally:

Example: If you want to fit to a STRAIGHT LINE (i.e. *1st order polynomial*): $f(t, x) = x_0 + x_1 t$

- You create matrix **A** to be 2 columns
  - Column 0 is the number **1** (that's t^0, the coefficients of x_0)
  - Column 1 is the number **t** (that's the t^1 coefficients of x_1)
- You create vector **b** (single column matrix)
  - Contains the observed values (i.e. the data ≡ the y-axis)
- You run LSQ method (`lstq()` function) to solve for **x** in **Ax ≅ b**
  - You can also get a **residual vector** (same size as **b**)

```
A:              b:              r:
 [[ 1.   0.]    2.9             -0.75
  [ 1.   1.]    2.7             -1.53
  [ 1.   2.]    4.8             -0.00
  [ 1.   3.]    5.3             -0.08
  [ 1.   4.]    7.1             1.14
  [ 1.   5.]    7.6     x:      1.07
  [ 1.   6.]    7.7      3.6485 0.59
  [ 1.   7.]    7.6      0.5771 -0.09
  [ 1.   8.]    9.4             1.13
  [ 1.   9.]    9.0             0.16
  [ 1.  10.]    9.6             0.18
  [ 1.  11.]    10.0            0.00
  [ 1.  12.]    10.2            -0.37
  [ 1.  13.]]   9.7             -1.45
```

Solving the **LSQ Problem** using **QR Factorization**

Consider: $A\boldsymbol{x} \approx b$

- ➔ $A^TA\boldsymbol{x} \approx A^Tb$
- ➔ $(A^TA)^{-1}(A^TA)\,\boldsymbol{x} \approx (A^TA)^{-1}(A^Tb)$
- ➔ $\boldsymbol{x} \approx (A^TA)^{-1}(A^Tb)$

This is called the "normal equation" and is not always recommended because you cannot guarantee numerical stability
(or that A is non-singular)

Solving the **LSQ Problem** using **QR Factorization**

$$\boldsymbol{x} \approx (A^TA)^{-1}(A^Tb)$$

- However, by forming the product A^TA, you square the condition number!
- Since $\text{Cond}(A) = \text{Cond}(A^T) \Rightarrow \text{Cond}(A^TA) = [\ \text{Cond}(A)\ ]^2$

- So, if you use the QR decomposition on matrix A, you can get a better least-squares estimate than the Normal Equations in terms of solution quality
  - Let’s see how…

Solving the **LSQ Problem** using **QR Factorization**

$$\boldsymbol{x} \approx (A^TA)^{-1}(A^Tb)$$

If A = QR, then:

$$\boldsymbol{x} \approx ((QR)^T(QR))^{-1}((QR)^Tb)$$

$\rightarrow \quad \boldsymbol{x} \approx (R^T\cancel{Q^T}\cancel{Q}R)^{-1}(R^TQ^Tb)$ since $Q^TQ = I$

$\rightarrow \quad \boldsymbol{x} \approx (R^TR)^{-1}(R^TQ^Tb)$

$\rightarrow \quad \boldsymbol{x} \approx R^{-1}R^{-1T}(R^TQ^Tb)$

$\rightarrow \quad \boldsymbol{x} \approx R^{-1}\cancel{R^{-1T}}(\cancel{R^T}Q^Tb)$ since $R^{-1T}R^T = I$

$\rightarrow \quad \boldsymbol{x} \approx R^{-1}Q^Tb$

Python Implementation

- Using `npla.lstsq()`

- OR!! We can also use `npla.qr()`

Quick! To the Python-mobile!

Your TO DOs!

- Finish new assignment by Monday
- Lab tomorrow!

</LECTURE>