Linear Regression

With Python 3.11 and Sage 8.9 [or somesuch]

Made by Gal Lebel and Alex Baucke on May 26th, 2023

> Contents

Problem Statement

- What / why are we approximating?
- Definitions
- The error function

Deriving a Solution

- Via calculus
- Via matrix/vector multiplication
- Geometric implications

Example

- Obtain a line through 3 points
- Verify the solution

Other Aspects

- Further examples
- Pyplot and stylesheets

Problem Statement

Problem Statement (1/3) - Finding the perfect fit

Given **n** points of data: $P_i := (x_i, y_i)$ $0 \dots i \dots n$

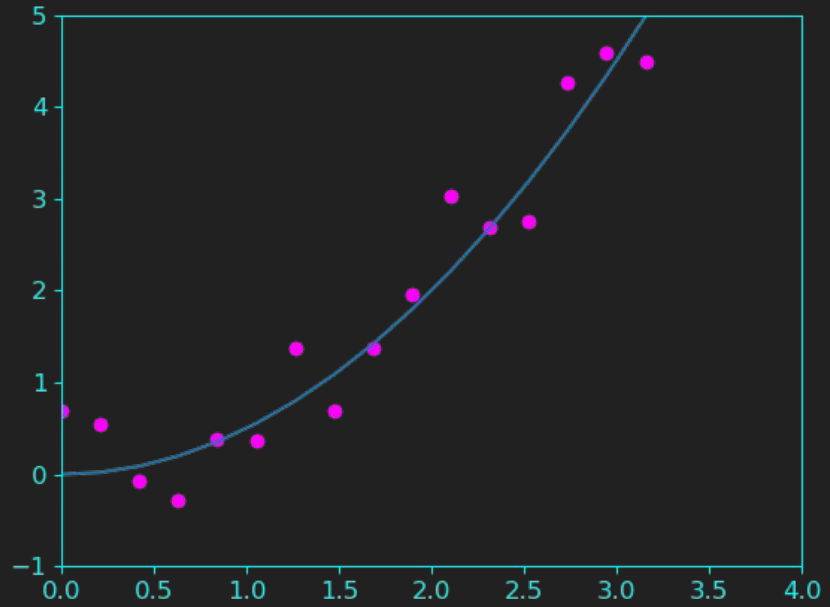
Find a function that best reflects the data.

The naive idea:

Find an $f(x)$, such that $f(x_i) = y_i$

=> This is interpolation

Why is interpolation a terrible idea for this?



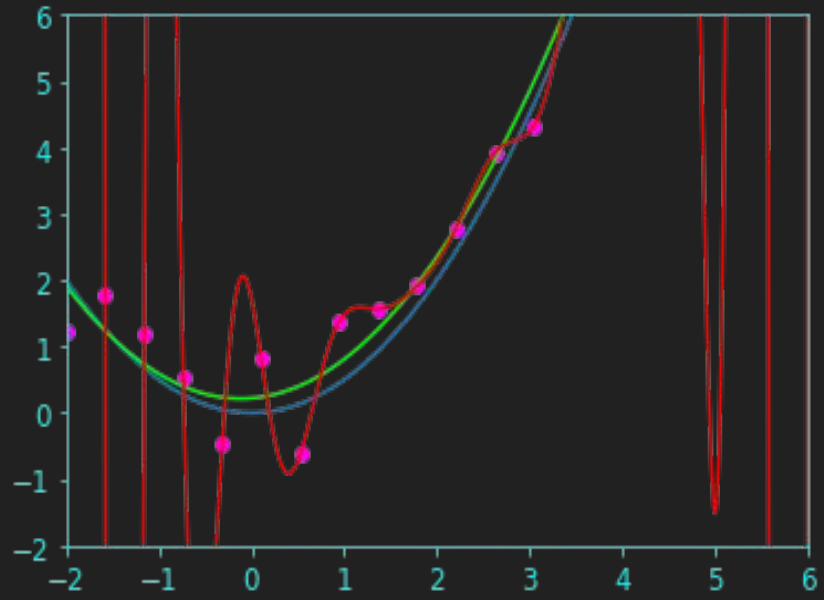
Seen on this graph: The stem function (blue) with errors on the y values added to equally spaced points (purple)

=> The purple dots are our P_i

Problem Statement (2/3) - Interpolation vs. Approximation

Let's assume a **sample size of 20** (or 20k...?)

=> **#P = 20** and **deg(f (x)) = 19**



Interpolation will not work here:

- Numerical stability
- Performance
- Usefulness for prediction
- "Model mismatch"

This also means that there cannot be a perfect solution - an **approximation** is needed.

So we are looking for **f (x) ∈ $\mathcal{P}_m \subset \mathcal{P}_n$** , with $\mathcal{P}_n$ denoting the polynomials of **degree <= n**

Problem Statement (3/3) - Finding the imperfect fit

We need a measure of "quality" for the approx.

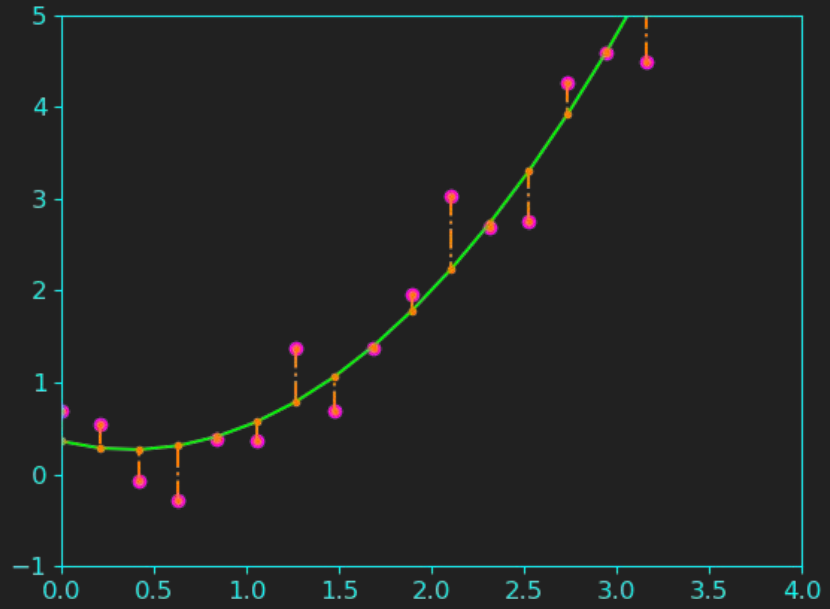
This usually means a function like this:

$$Err(f, P) = \sum_{i..n} (y_i - f(x_i))^2 \qquad (x_i, y_i) \in P_i$$

Therefore, we can write:

$$f := \operatorname{argmin}_{f \in \mathcal{P}_m} (Err(f, P))$$

for our approximation space $\mathcal{P}_m$



And call $f \in \mathcal{P}_m$ the **Least Squares Approximation** of P

Problem Statement - Recap

- Interpolating the stem function in every $P_{1...n}$ with $f(x) \in \mathcal{P}_n$ is not feasible
- Instead, we look for a "good" approximation with $f(x) \in \mathcal{P}_m$ where $m << n$
- This means that we will have to allow some "error": $Err(f, P)$

Finding the $f(x) \in \mathcal{P}_m$ that minimizes this error is our solution.

Such an $f(x)$ could look like this: $f(x) = c_0 + c_1 x \quad f(x) \in \mathcal{P}_1$

(If written in the monomial basis)

The next chapters are about how to obtain these c_i !

Deriving a Solution

The Calculus Way (1/5)

We know that a solution to our problem takes the form:

$$f(x)=c_0+c_1x+c_2x^2+\dots \quad , \quad f(x) \in \mathcal{P}_m \quad , \quad m < n$$

For simplicity, we assume $m = 2$, so $f \in \mathcal{P}_2$ is linear.

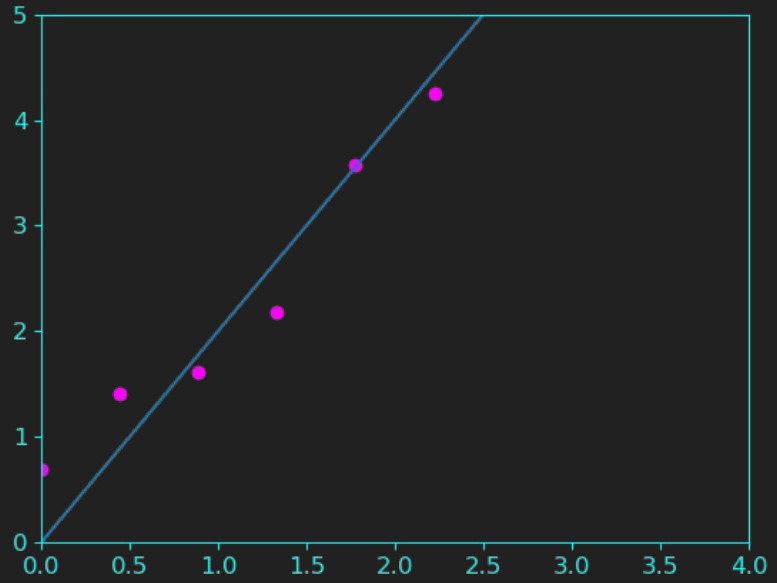
Given n data points and c_0, c_1 as our unknowns, we have

$$c_0 + c_1x_1 = y_1$$

$$\dots$$

$$c_0 + c_1x_n = y_n$$

This system of $n > 2$ equations and 2 unknowns is overdetermined and has no exact solution.



The blue line is the assumed linear relation.
The points deviate by some noise from the line.

The Calculus Way (2/5)

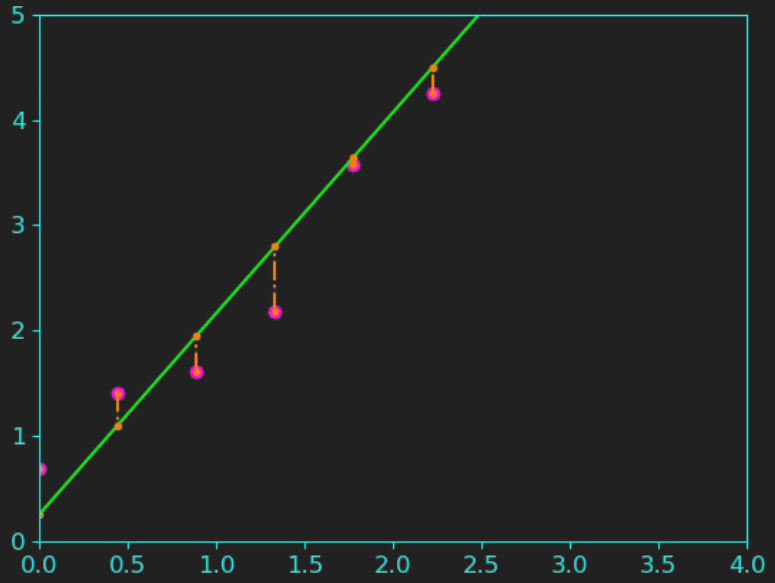
To make solving the system possible, we introduce residuals r_i given by:

$$r_i := y_i - f(x_i) = y_i - (c_0 + c_1 x_i)$$

By allowing **errors** into our **approximation**, we have turned our problem into a system of **n** equations and **n** unknowns, thus the system has an exact solution!

We also want to square and sum the residuals:

$$S(c_0, c_1) = r_1^2 + r_2^2 + \ldots + r_m^2 = (y_i - (c_0 + c_1 x_i))^2 + \ldots = Err(f, P)$$



Green line is a possible imperfect fit line.
The residuals are colored orange.

The Matrix Way (3/5)

How do we minimize the error function **S**?

Lets make our life easier with some matrices, assuming a system of equations of the form

Ax = b

A is a **n x m** Matrix, where the columns vectors are the monomial basis of the space $\mathcal{P}_m$

x is a **N x 1** vector of our coefficients which we want to determine

b is a **Nx1** vector of the **n** observation points (the y_i's)

A (tall-matrix : M < N) x b

$$\begin{bmatrix} (x_1)^0 & (x_1)^1 & \dots & (x_1)^m \\ (x_2)^0 & (x_2)^1 & \dots & (x_2)^m \\ \vdots & \vdots & \vdots & \vdots \\ (x_n)^0 & (x_n)^1 & \dots & (x_n)^m \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Dimensions: NxM N N

Where are the residuals?

Aren't we looking for an approximation?

The Matrix Way (4/5)

Considering our residuals from earlier, we allow for some error, thus we have:

$$Ax + r = b \quad \rightarrow \quad r_i = b_i - A_i x_i$$

And with error function S:

$$S(x^T) = r_1^2 + \dots r_n^2 = r^T r = (b - Ax)^T (b - Ax)$$

(we have here x as argument since its the coefficients vector)

And through simplification of the right-hand-side:

$$S(x) = b^T b - 2x^T A^T b + A^T x^T A x$$

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} (x_1)^0 & (x_1)^1 & \dots & (x_1)^m \\ (x_2)^0 & (x_2)^1 & \dots & (x_2)^m \\ \vdots & \vdots & \vdots & \vdots \\ (x_n)^0 & (x_n)^1 & \dots & (x_n)^m \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

The Matrix Way (5/5)

Since we're interested in the solution for **x**, we will derive **S(x)** w.r.t. x and set it to 0:

$\partial S(x)/\partial x = 0 = -2A^Tb + 2A^TAx$

Divide by **2** : $= -A^Tb + A^TAx$

Isolate x : $\rightarrow A^TAx = A^Tb$

Multiply by $(A^TA)^{-1}$: $\rightarrow x = (A^TA)^{-1}A^Tb$

Thus we have arrived at the matrix form of the solution!

$$S(x) = b^Tb - 2x^TA^Tb + A^Tx^TAx$$

$\partial S(x)/\partial x\ b^Tb = \underline{0}$

$\partial S(x)/\partial x\ -2x^TA^Tb$ is just like $\partial S(x)/\partial x\ 2x*y$ with $y = A^Tb$
$\rightarrow = \underline{-2A^Tb}$

$\partial S(x)/\partial x\ A^Tx^TAx = \partial S(x)/\partial x\ x^2A^TA = \underline{2A^TAx}$

Equivalently, we could have arrived to this by

$Ax = b$ (multiply by A^T)

$A^TAx = A^Tb$ (assuming $|A^TA| != 0$, multiply by $(A^TA)^{-1}$, otherwise - no solution!)

$x = (A^TA)^{-1}A^Tb$

Which is exactly what we've on the left side!

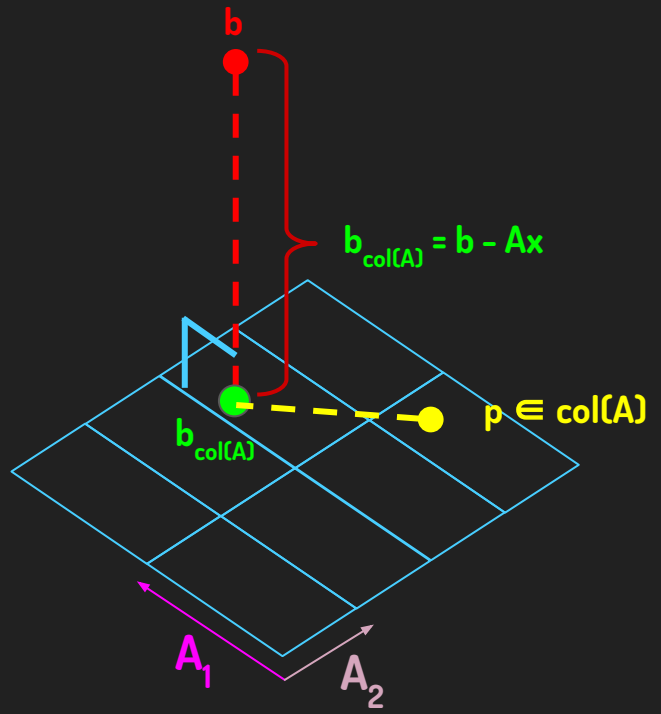
Geometrical Visualization \ Solution (1/2)

Col(A) spans a plane in 3D space

$b \in R^3 \land b \notin col(A)$

A solution would be to get as close to the vector b while staying in the subspace **col(A)**.

The shortest distance from **b** to **col(A)** would be an orthogonal vector going from **b** down to a specific point (call it $\mathbf{b}_{col(A)}$) in **col(A)**.



A1 and **A2** are the columns of **A** - so just 2 columns (for simplicity of visualization)

Geometrical Visualization \ Solution (2/2)

We want a linear combination of the column vectors of **A** to be as close to **b** (which means it should be equal to $\mathbf{b}_{col(A)}$).

By orthogonality we have:
$(b - Ax)A_i = 0$ for every column A_i in **A**.
(note that **b** and **Ax** are both Nx1 vectors)

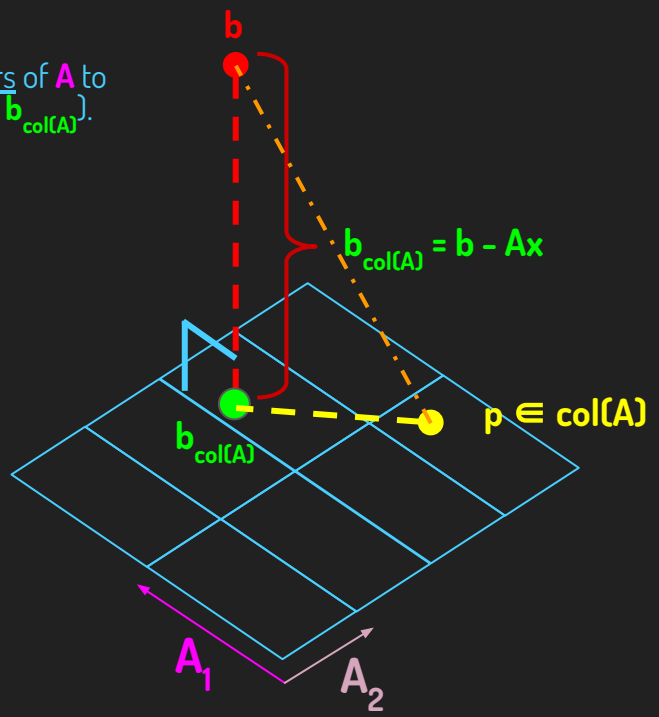
So we have $(b - Ax)A_i = (b-Ax)^T A$.

Then :

$$(b - Ax)^T A = b^T A - A^T Ax$$

$$\rightarrow b^T A = A^T Ax$$

$$\rightarrow x = (A^T A)^{-1} b^T A$$



Two identities:

$(b - Ab_{col(A)}) \cdot (Ab_{col(A)} - Ap) = 0$

$\|(b - Ab_{col(A)})\| <= \|(b - Ap)\|$

Implies that there is one exact solution that yields the best approximation under the given circumstances.

A1 and **A2** are the columns of **A** - so just 2 columns (for simplicity of visualization)

Deriving a solution - Recap

- By orienting ourselves to the premise of the problem statement - we've derived a solution.
- We have shown two different ways (geometry and calculus with some matrices) to reach the same solution!

So what about a short example, to understand and see how this actually works?

The next short chapter will show a very simple example

That demonstrates what've shown so far.

A simple example for approximating 3 points

A simple example for approximating 3 points

Given 3 points of data

(1,1), (2,4), (3,3)

We want to find the line with the best fit.

We will create:

- the matrix **A** with the monomial basis (x_i^0 , x_i^1)
- the vector **x** with the coefficients c_0, c_1
- the vector **b** with the y points

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, x = \begin{bmatrix} c0 \\ c1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$$

The 3 matrices for our system of equations

A simple example for approximating 3 points

We can now use our derived solution for the matrix form: $\mathbf{x = (A^T A)^{-1} A^T b}$

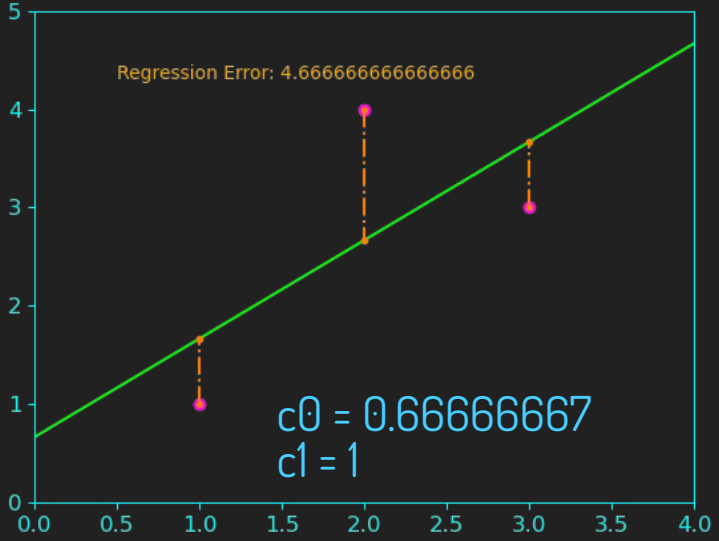
$$x = \begin{bmatrix} c0 \\ c1 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 14/6 & -1 \\ -1 & 1/2 \end{bmatrix} \begin{bmatrix} 8 \\ 18 \end{bmatrix}$$

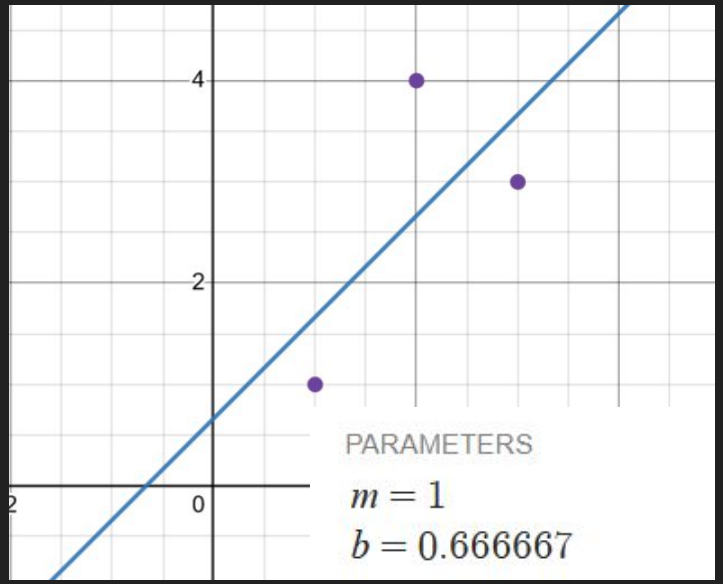
$$= \begin{bmatrix} 4/6 \\ 1 \end{bmatrix}$$

A simple example for approximating 3 points

We can verify this calculation with both desmos and our program



Our Program



Desmos

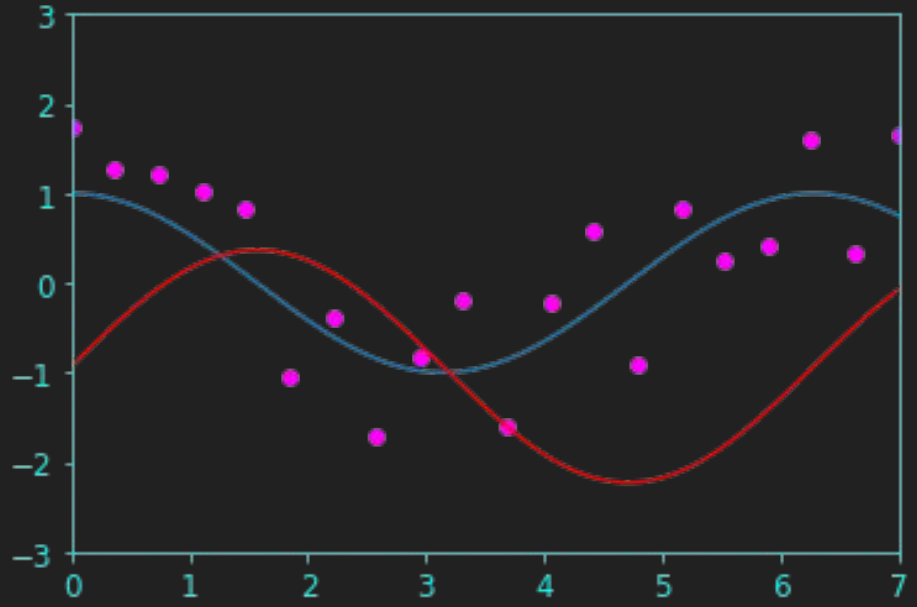
Other Aspects

Limitation of estimating with linear least squares

As the name suggests, linear least squares is best for approximating linear relationships between dependent and independent set of datas.

Assume data with the form:
$f(x) = c_0 * \sin(x)$ (with some noise added).
As we know, the sine function is nonlinear, but the function overall is linear with with respect to **c_0**.

For this function, **c_0** will be adjusting the amplitude of the sine curve, i.e. – if **sin(x)** goes from **0 to 1**, **f(x)** will oscillate from **$-c_0$ to c_0**.



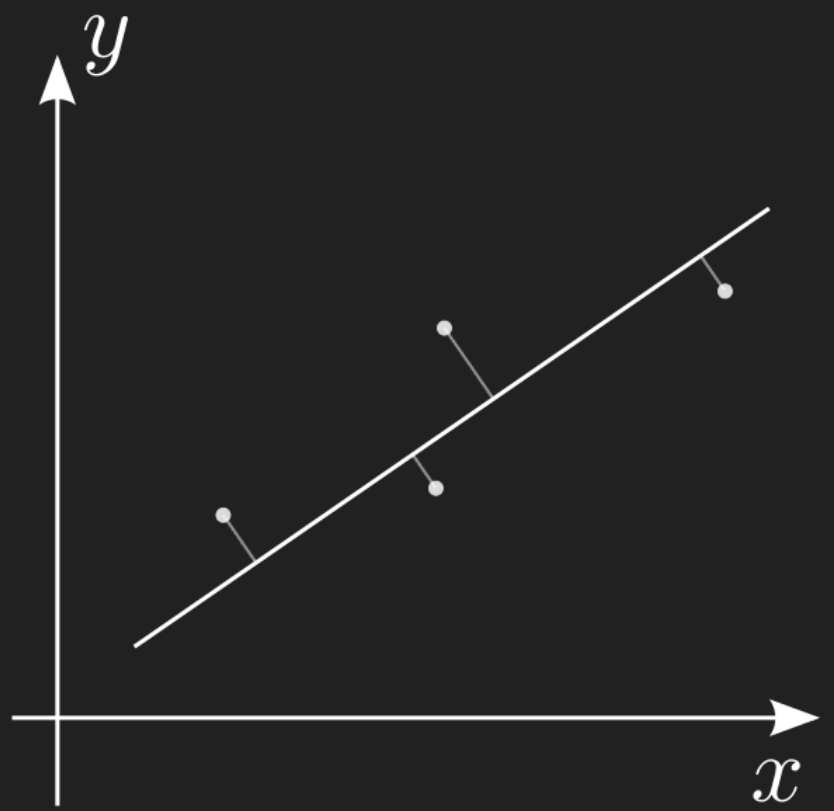
$\sin(x + c_0)$ – the stem function
$a + b * \sin(x)$ – approximation function

Other types of estimation methods

Among the linear least squares, there are more estimation methods with the same goal.

The total least squares is one of them.

The usual assumption for TLS is, that the error that arises through the measurement of the data is equal in both the dependant and independent variables, thus an approximation of the residuals by the distance from the tangent of the line\curve is favoured and more robust to those noises\errors, in comparison to the linear least squares.



Ty for your time 🤡

Any Questions?

Resources for further reading:

- https://en.wikipedia.org/wiki/Linear_least_squares
- https://textbooks.math.gatech.edu/ila/1553/least-squares.html
- https://en.wikipedia.org/wiki/Gram_matrix
- https://jakevdp.github.io/PythonDataScienceHandbook/04.11-settings-and-stylesheets.html

entwurfe

The problem with approximating in this way is, that we have no control over the frequency and the period of the sine function.
This will result in a bad approximation of the data, depending on strong the data diverts from the stem function.

A possible attempt to get a more accurate approximation would be to work with:
$f(x) = c_0 * \sin(c_1 x)$
Which will allow us to adjust the frequency of the sine wave, i.e. how fast it goes from its maximum to its minimum.

By doing that, though, the function is no longer linear with respect to c_1 which means that we cannot use linear least squares to find a best fit.