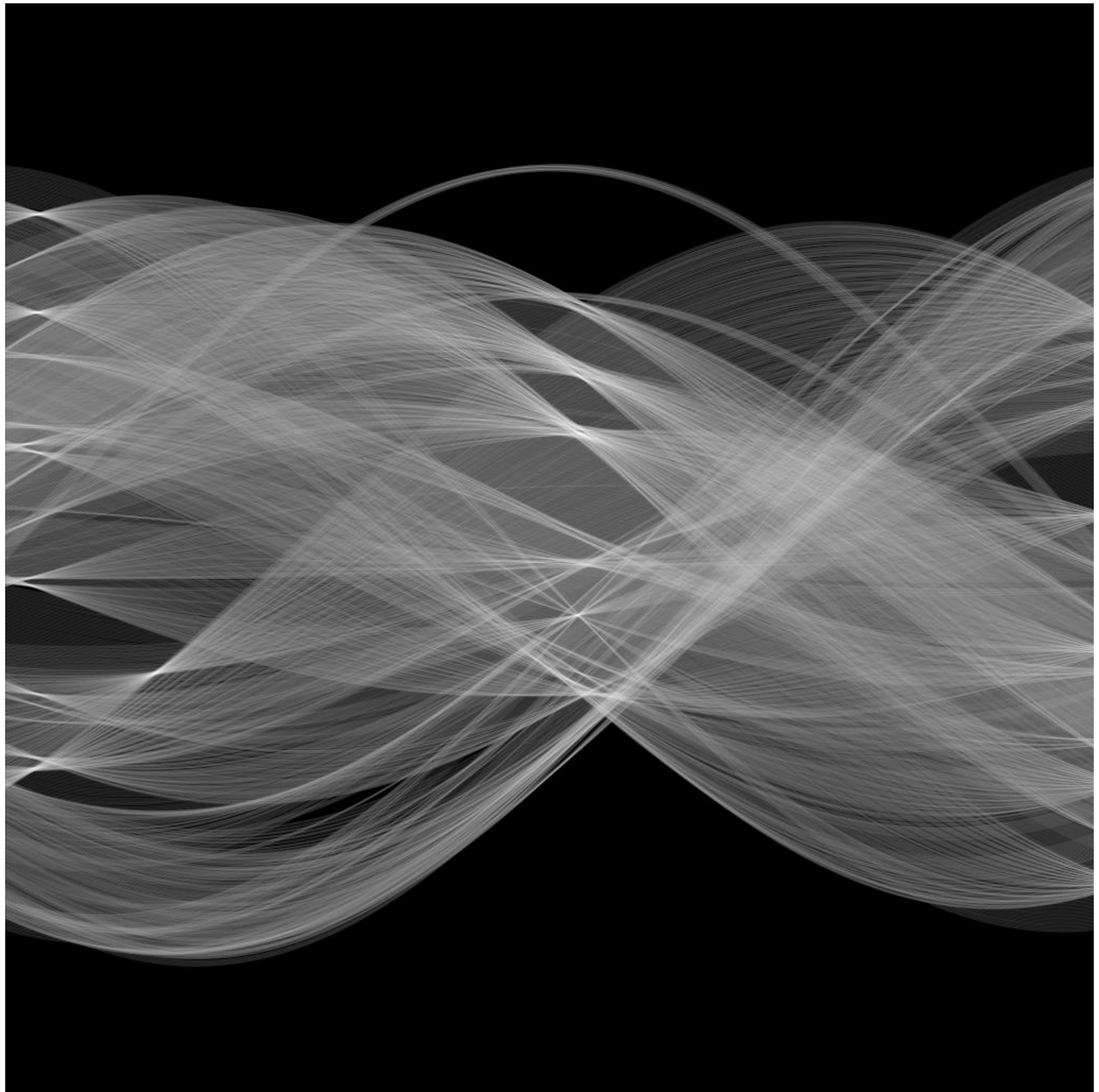
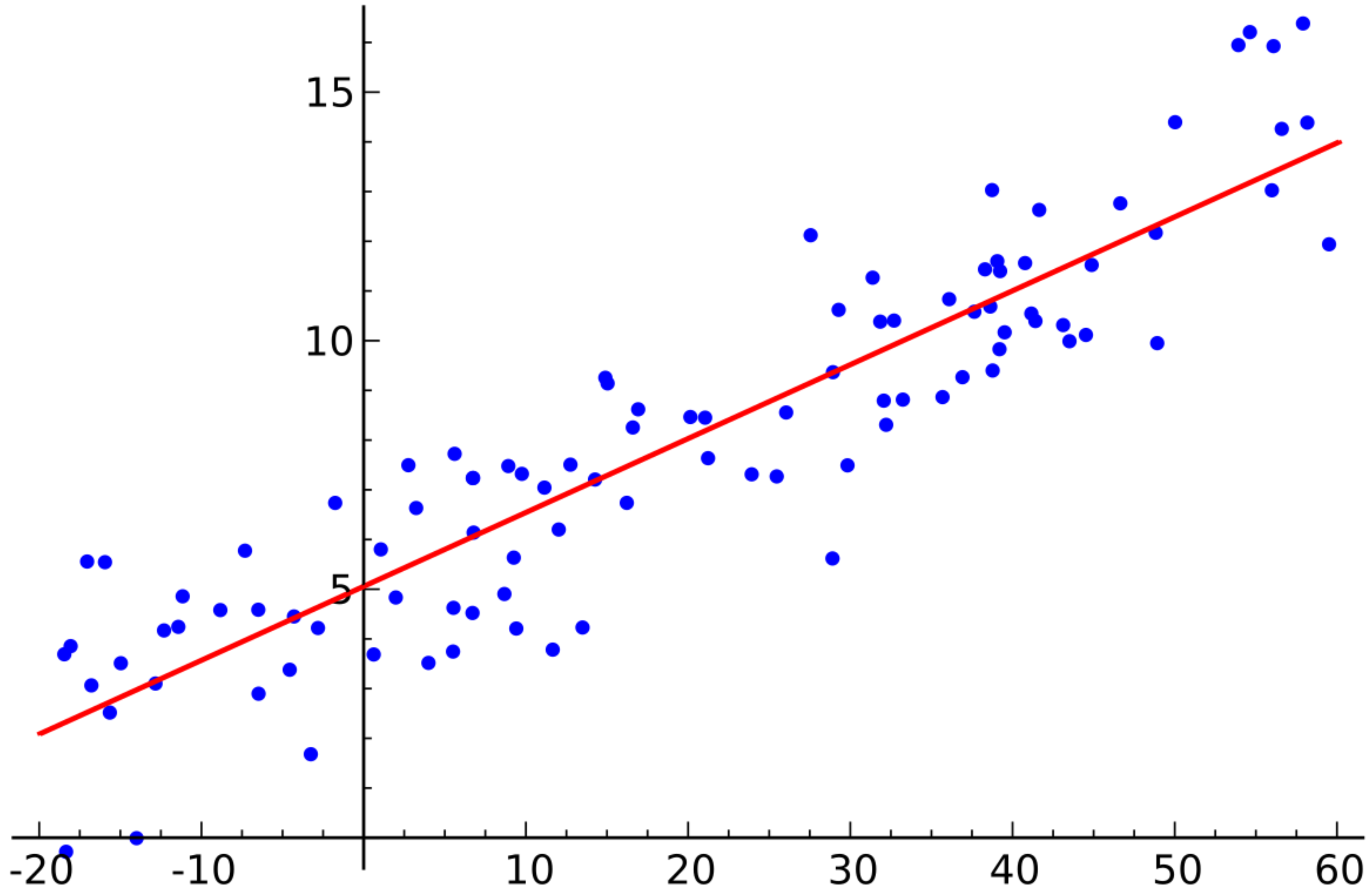
Curve fitting



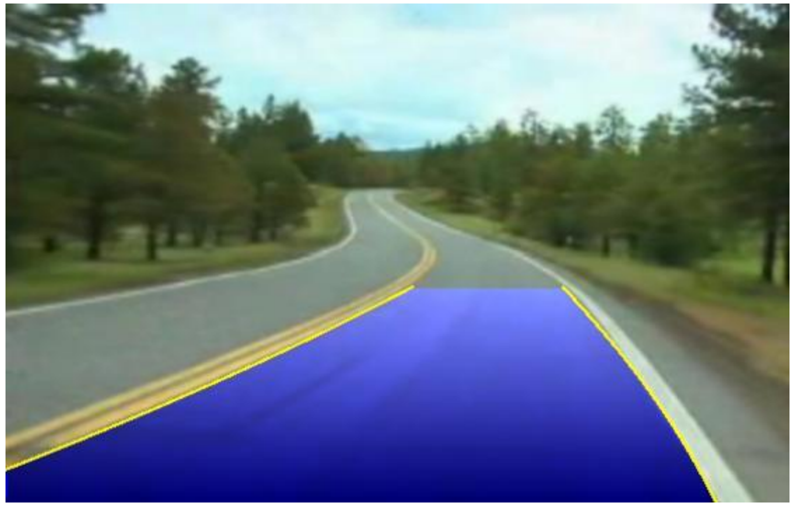
- Line fitting
  - Why
  - LS
  - Total LS
  - RANSAC
- Hough
  - Regular m,n hough
  - Problem with m=inf
  - R,theta hough
  - Circles hough
  - Generalized ?
- Template matching- can be moved to SIFT (feature descriptors and matching) / stereo

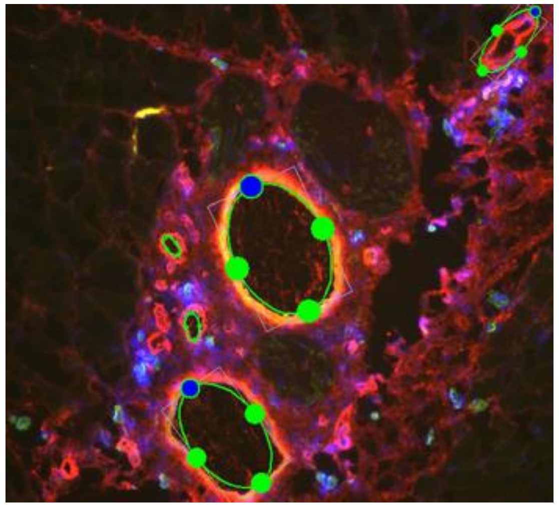
Some motivation

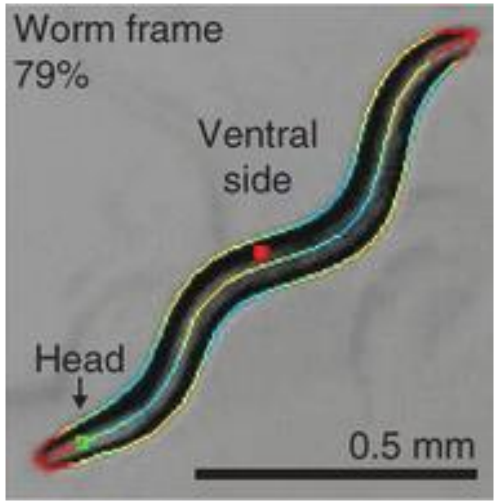
- What you think about when you here “line/curve fitting”



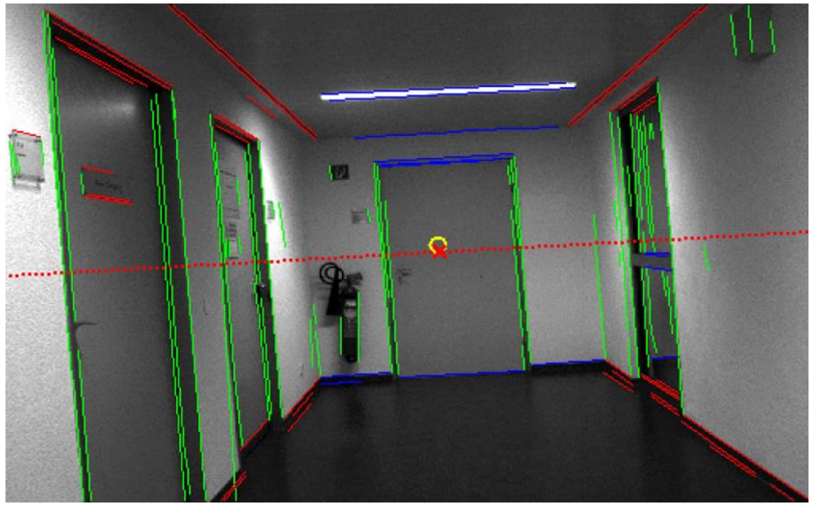
Some motivation

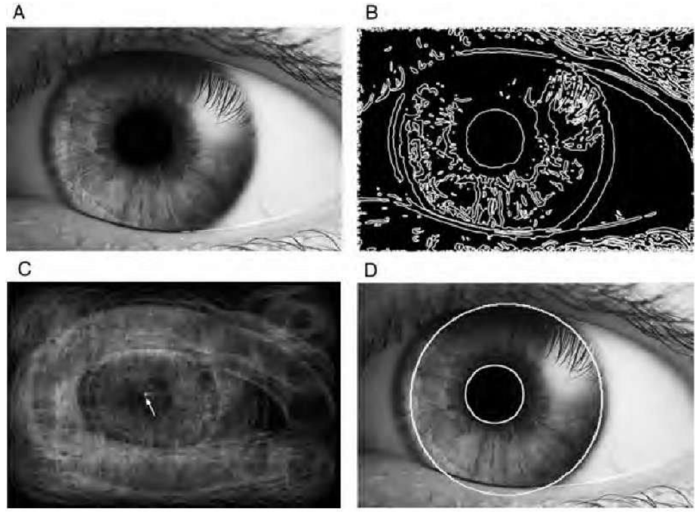

Autonomous Vehicles
(lane line detection)


Bio-medical engineering
(blood vessel counting)



Biology
(earthworm contours)


Robotics
(scene understanding)



Psychology/ Human computer interaction
(eye tracking)

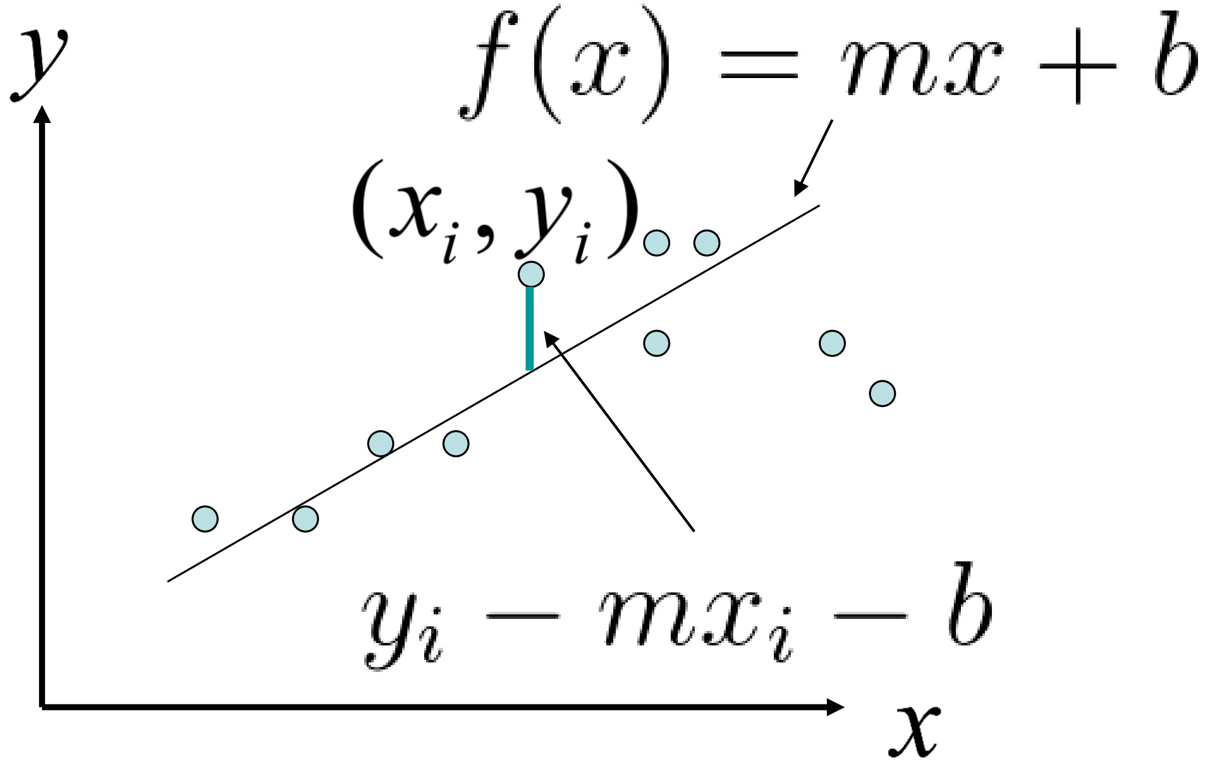
What is curve fitting

- **Curve fitting** is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points. [Wikipedia]
- Input: dataset (e.g.: $\{(x_i, y_i)\}_{i=1,..,N}$ in 2D).
- Output: best representing function (e.g.: $f(x) = y$).

- This problem sometimes also called **Regression.**

Linear least squares (LS): step by step example

- Given: $\{(x_i, y_i)\}_{i=1,\ldots,N}$
  find best line representation: $f(x) = mx + b$
- The best representation (m, b) are the ones that minimizes the total error e.



Side note: error / loss

- **Error** (also known as **loss**) is a method of evaluating how well specific algorithm models the given data.
  - If predictions deviates too much from actual results, error would be high.
- A popular error used a lot in CV and statistics is called **MSE** (mean square error, also known as **L2 loss** or **quadratic loss**).

$$\boxed{e_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$$

- We need to find the set of variables that minimizes the error - **MMSE** (minimum mean square error).

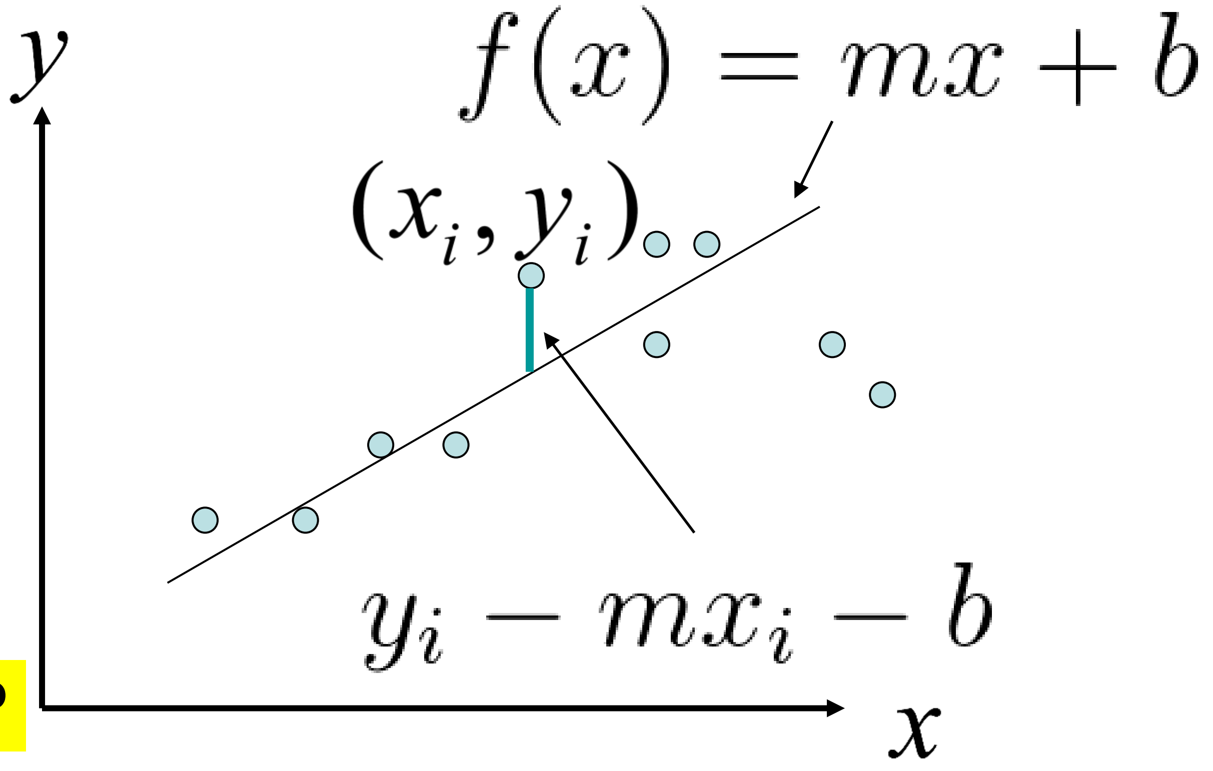
Linear least squares (LS): step by step example

- Given: $\{(x_i, y_i)\}_{i=1,\dots,N}$
  find best line representation: $f(x) = mx + b$
- The best representation (m, b) are the ones that minimizes the total error e.

$$e_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

$$\downarrow$$

$$e = \frac{1}{N}\sum_{i=1}^{N}(y_i - mx_i - b)^2$$



- How do we find this set of variables?

Linear least squares (LS): step by step example

- Find derivatives of e with respect to both variables m, b s.t. (such that) we'll reach the minimum error (partial derivative of both variables equals zero…):

$$e = \frac{1}{N}\sum_{i=1}^{N}(y_i - mx_i - b)^2$$

  - Define: $\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,\ \overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$

$$0 = \frac{\partial e}{\partial n} = \frac{-2}{N}\sum_{i=1}^{N}(y_i - mx_i - b) \mapsto 0 = \frac{1}{N}\sum_{i=1}^{N}(y_i - mx_i - b) \mapsto b = \overline{y} - m\overline{x}$$

$$0 = \frac{\partial e}{\partial m} \mapsto \ldots \mapsto m = \frac{\sum_{i=1}^{N}(y_i - \overline{y})(x_i - \overline{x})}{\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

- Full derivation here.

Linear LS in matrix form

$$e = \frac{1}{N}\sum_{i=1}^{N}(y_i - mx_i - b)^2$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad X = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \quad \beta = \begin{bmatrix} m \\ b \end{bmatrix}$$

$$e = \frac{1}{N}||y - X\beta||^2 = \frac{1}{N}(y - X\beta)^T(y - X\beta) =$$

$$\frac{1}{N}(y^T y - \beta^T X^T y - y^T X\beta + \beta^T X^T X\beta) \overset{\beta^T X^T y = scalar = scalar^T = y^T X\beta}{=}$$

$$\frac{1}{N}(y^T y - 2\beta^T X^T y + \beta^T X^T X\beta)$$

$$0 = \frac{\partial e}{\partial \beta} = 2X^T X\beta - 2X^T y$$

$$X^T X\beta = X^T y$$

$$\boxed{\beta = (X^T X)^{-1} X^T y}$$

Side note: pseudoinverse

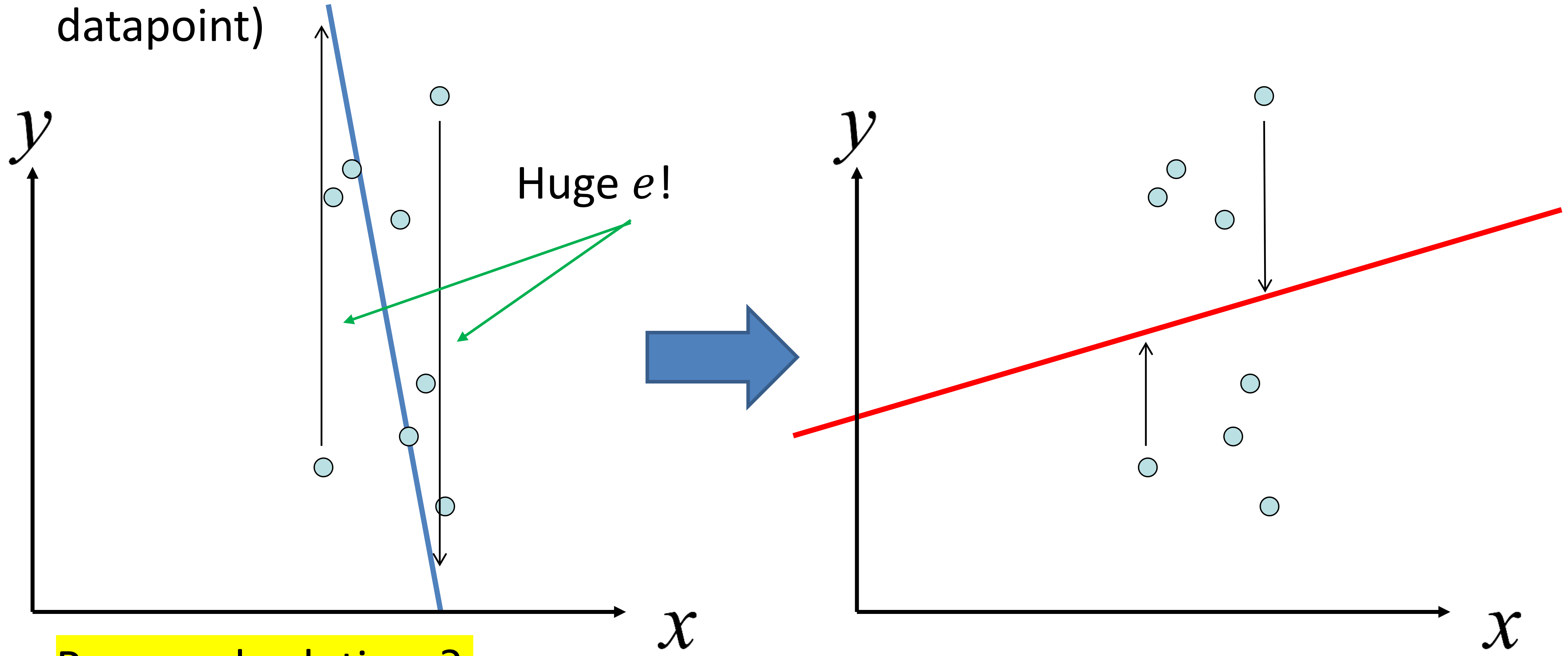
$$\boxed{\beta = \underbrace{(X^T X)^{-1} X^T}_{} y}$$

- This is a known result which is called **the pseudoinverse matrix** of X. It is also known as Moore–Penrose inverse.
- If $X^T X$ is not invertible, the solution will be: $\beta = (X^T X + \epsilon I)^{-1} X^T y$
- The solution above is **a solution for any linear regression problem that is over-determined** (==more equations than unknowns).

- least_squares.ipynb

Problem 1: Linear LS with vertical data

- Near vertical data is hard to fit since the error is computed perpendicular to x axis + the bigger the error, the bigger the error squared! (more weight for this datapoint)



- Proposed solutions?

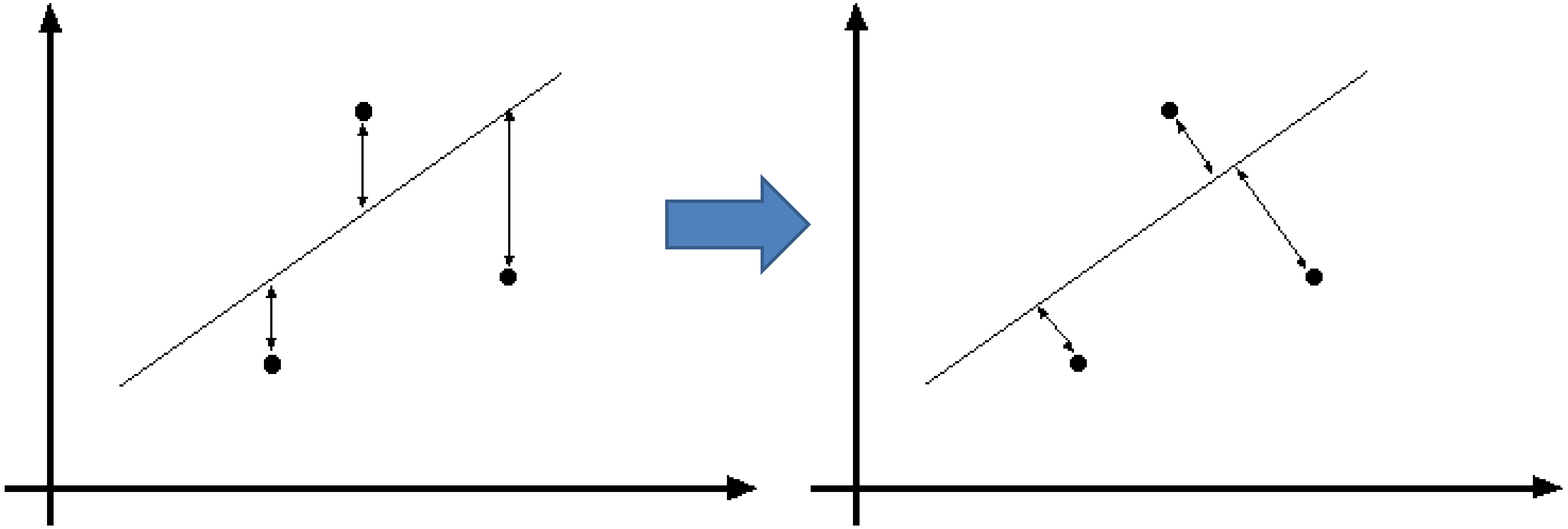
Problem 1: Linear LS with vertical data

- One possible solution is making all errors weigh the same (and not squared). This will make the far points have the same impact on error as closer points. One such error **MAE** (mean absolute error, also known as **L1 loss**) instead of MSE.

$$e_{MAE} = \frac{1}{N}\sum_{i=1}^{N} ||y_i - f(x_i)||$$

  - The derivation of MAE is out of the class scope, but some details can be found here.
- Another possible solution is computing the error distance of each point in a different way- one that takes into account the y data as well. One such algorithm is **linear total least squares.**

Linear total least squares



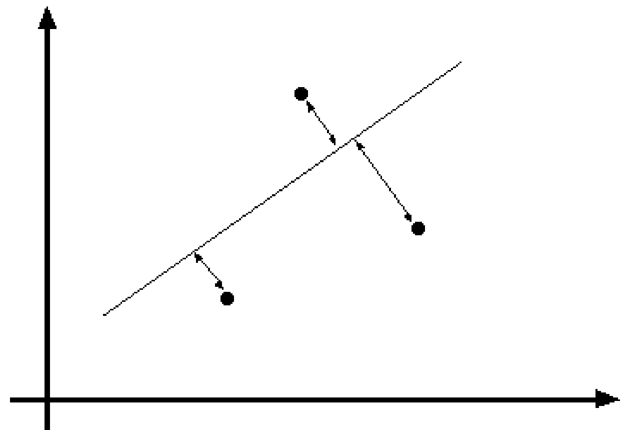
Linear TLS

- Another representation of a line: $ax + by + c = 0$
- Distance between a point (x_i, y_i) and line:

$$d_i = \frac{|ax_i + by_i + c|}{\sqrt{a^2 + b^2}}$$

- The line equation above has 2 degrees-of-freedom (DOFs), so we can decide that, for later purpose, $a^2 + b^2 = 1$.
- The error to be minimized:

$$e_{MSE} = \frac{1}{N}\sum_{i=1}^{N} d_i^2 = \frac{1}{N}\sum_{i=1}^{N}(ax_i + by_i + c)^2$$



Linear TLS

$$e_{MSE} = \frac{1}{N}\sum_{i=1}^{N} d_i^2 = \frac{1}{N}\sum_{i=1}^{N}(ax_i + by_i + c)^2$$

$$0 = \frac{\partial e}{\partial c} = \frac{1}{N}\sum_{i=1}^{N} 2(ax_i + by_i + c)$$

$$c = -\frac{a}{N}\sum_{i=1}^{N} x_i - \frac{b}{N}\sum_{i=1}^{N} y_i = -a\overline{x} - b\overline{y}$$

$$e = \frac{1}{N}\sum_{i=1}^{N}(ax_i+by_i-a\overline{x}-b\overline{y})^2 = \frac{1}{N}\sum_{i=1}^{N}(a(x_i-\overline{x})+b(y_i-\overline{y}))^2 = \frac{1}{N}\left\| \begin{bmatrix} x_1-\overline{x} & y_1-\overline{y} \\ \vdots & \vdots \\ x_N-\overline{x} & y_N-\overline{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 =$$

$$\frac{1}{N}\|X\beta\|^2 = \frac{1}{N}\beta^T X^T X\beta$$

$$\begin{cases} minimize & \frac{1}{N}\beta^T X^T X\beta \\ s.t. & a^2+b^2=1 \end{cases} = \boxed{\begin{cases} minimize & \beta^T X^T X\beta \\ s.t. & \beta^T\beta = 1 \end{cases}}$$

Linear TLS -the minimization problem

- The minimization problem is:

$$\begin{cases} minimize & \beta^T X^T X \beta \\ s.t. & \beta^T \beta = 1 \end{cases}$$

- Recall eigendecomposition: $Av = \lambda v \mapsto v^T A v = \lambda$
  - Also recall that each eigenvector v is normalized ($||v|| = v^T v = 1$).
- The solution to the minimization problem above is the eigenvector corresponding to smallest eigenvalue of $X^T X$.

- **Watch out:** trying to minimize the problem above without the constraint $\beta^T \beta = 1$ will result with the trivial solution of $\beta = 0$.

LS

- What about curve fitting?
- Recall the matrix linear LS result: $\beta = (X^T X)^{-1} X^T y$

- The solution above is a solution for any **linear regression** problem that is over-determined (==more equations than unknowns).
  - Linear regression $\neq$ linear LS
  - Linear regression means only that the unknowns are linearly dependent in the data.
- For example- data set of a parabola:

  $\{(x_i, y_i)\}\ s.t.\ ax_i^2 + bx_i + c = y_i$

- How the matrices X, y, β will look like?

LS

- data set of a parabola:

  $\{(x_i, y_i)\}\ s.t.\ ax_i^2 + bx_i + c = y_i$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad X = \begin{bmatrix} x_1^2 & x_1 & 1 \\ \vdots & \vdots & \vdots \\ x_N^2 & x_N & 1 \end{bmatrix} \quad \beta = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

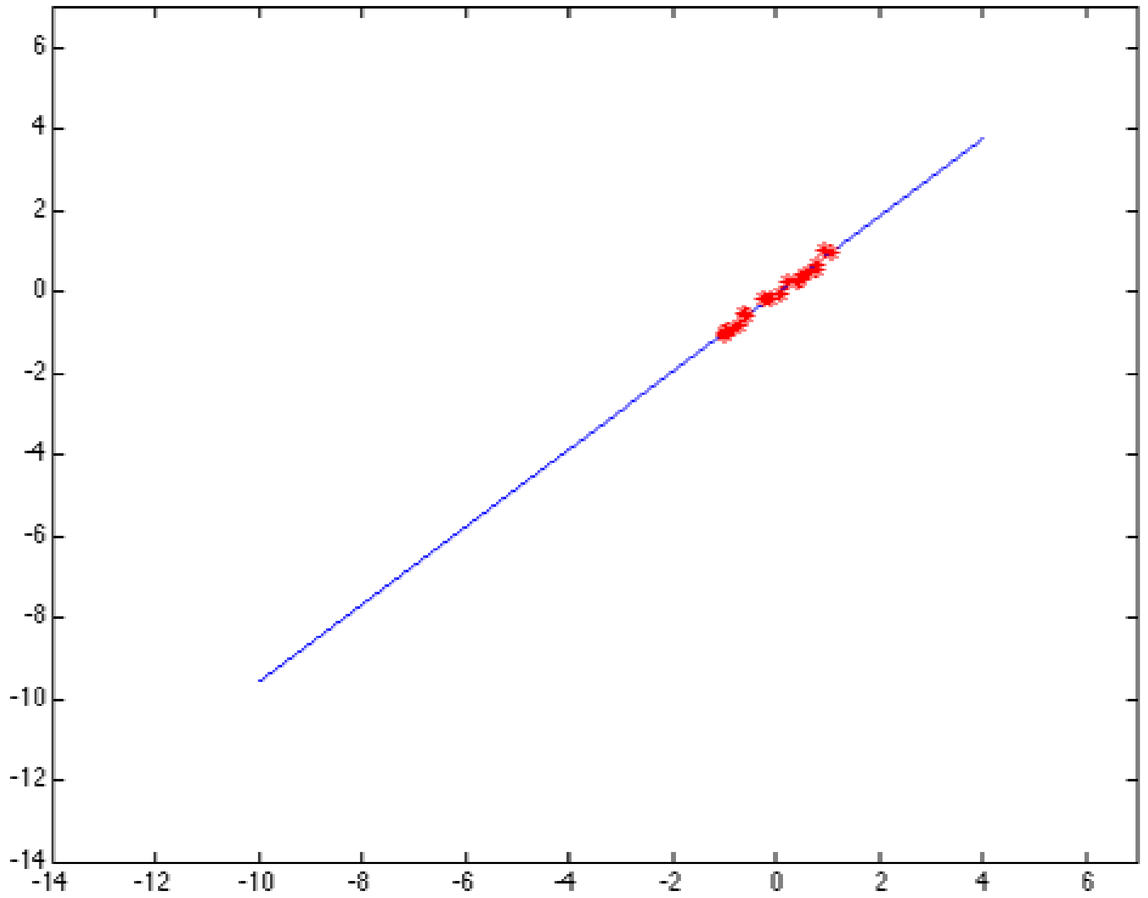
- The solution of this LS problem is the same (in our Derivation we didn't use the assumption that the data is linear…):

$$\beta = (X^T X)^{-1} X^T y$$

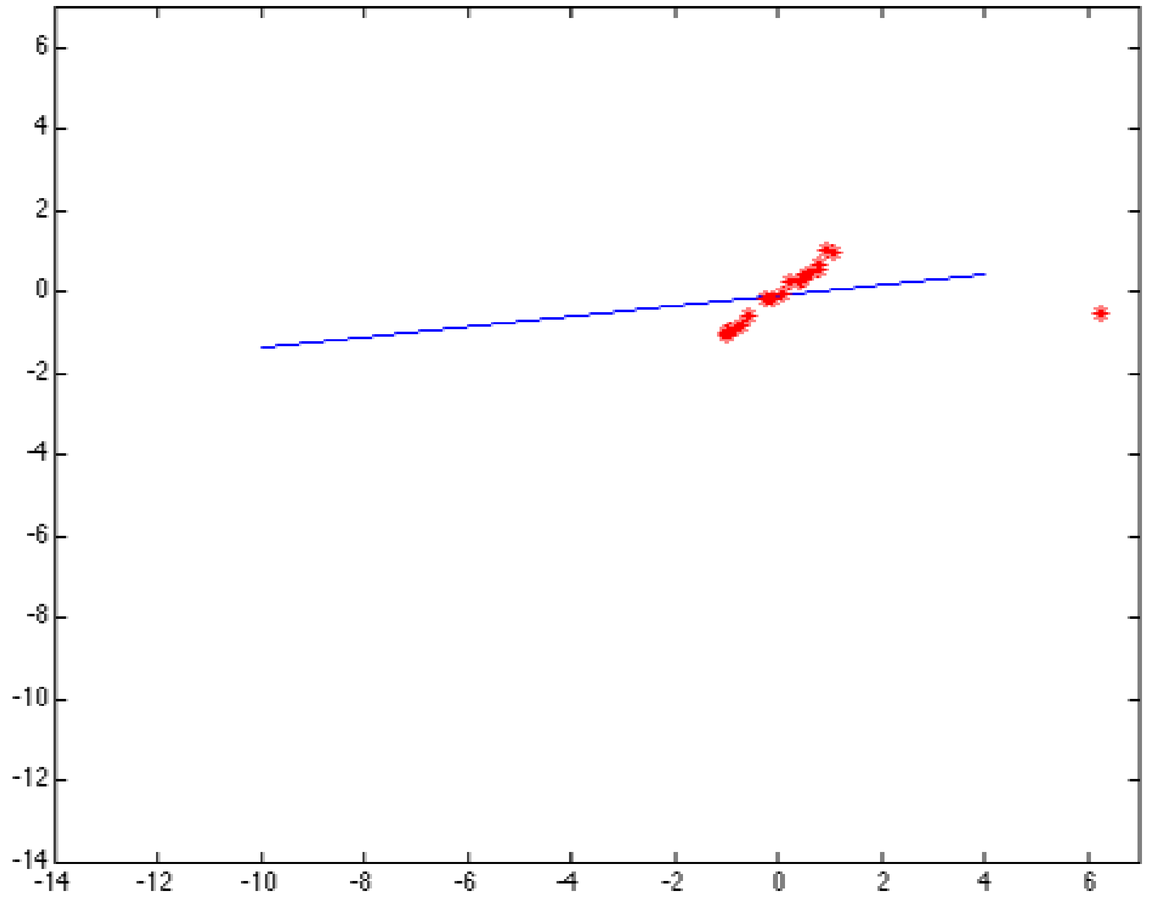
- Non-linear TLS also exist, but this can't be solved as before (the linearity assumption was used). This topic is out of scope- proof and examples here.

Problem 2: LS with outliers datapoints

- **Outlier**: a data point that differs significantly from other observations. [Wikipedia]



Least-squares error fit



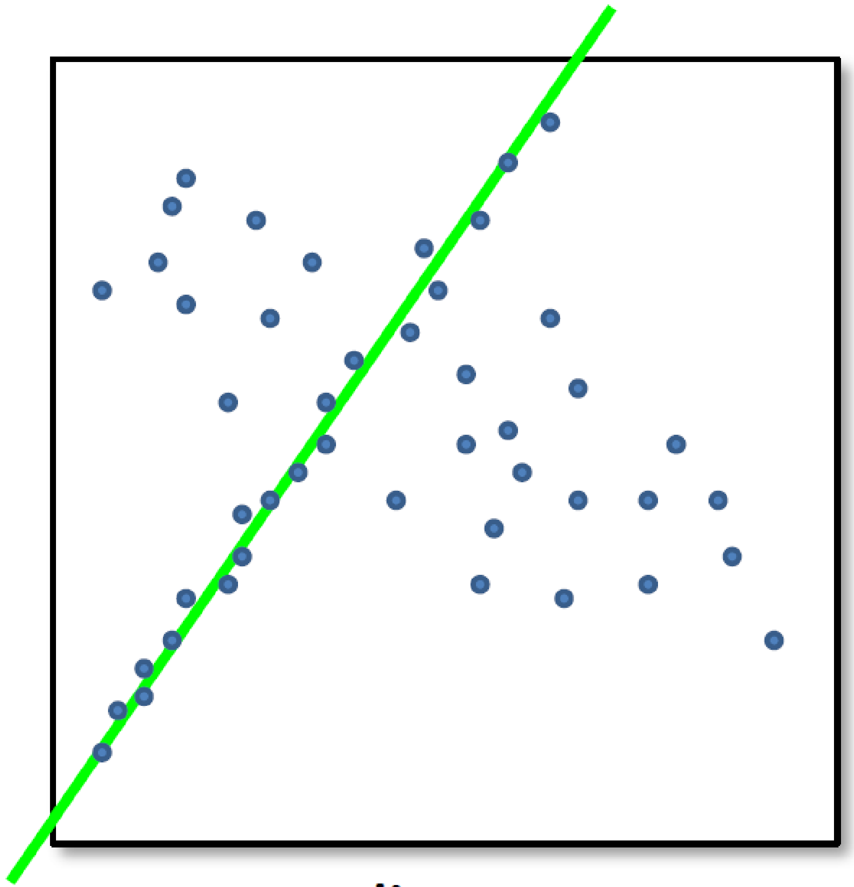
Squared error heavily penalizes outliers

Problem 2: LS with outliers datapoints

- One possible solution: MAE (again)- for less penalization on outliers.
- A better solution: removing the outliers! Possible algorithm to use: **RANSAC**

RANSAC

- **Random sample consensus** (**RANSAC**) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers. [Wikipedia]



Inliers: 20

RANSAC algorithm

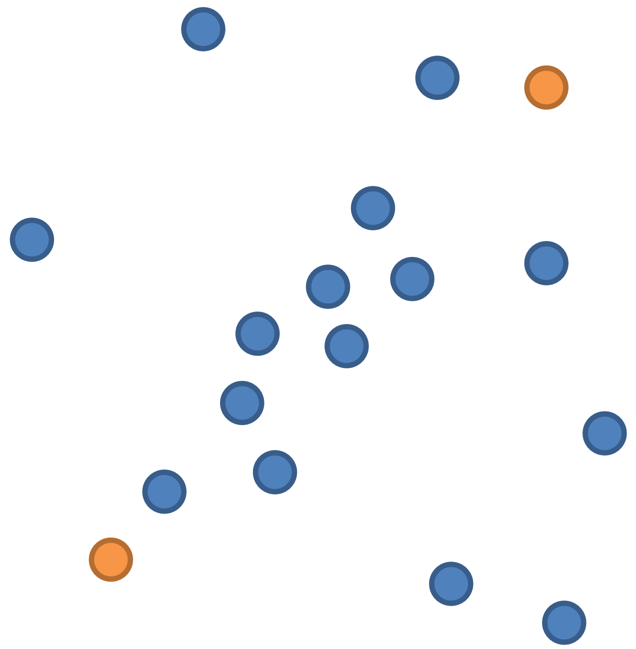
While searching for best model fit:

1. Select a random subset of the original data.
2. Fit a model to the data subset.
3. Test all dataset against the fitted model. Points that fit the estimated model well, according to some chosen loss function, are considered as part of the **consensus set** (also known as **inliers**).
4. Save as best model if number of inliers is bigger the old best model number of inliers.

RANSAC algorithm - step by step - 1

Select a random subset of the original data.

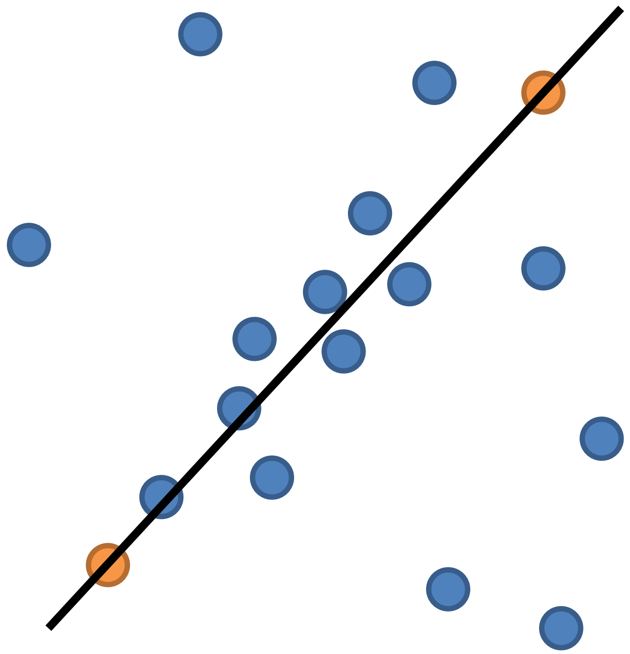
- The number of samples in the subset is the smallest needed to determine the model parameters (== number of unknow variables).
- Examples:
  - For line fit- only 2 datapoints.
  - For parabola fit- 3 datapoints.



RANSAC algorithm - step by step - 2

Fit a model to the data subset.

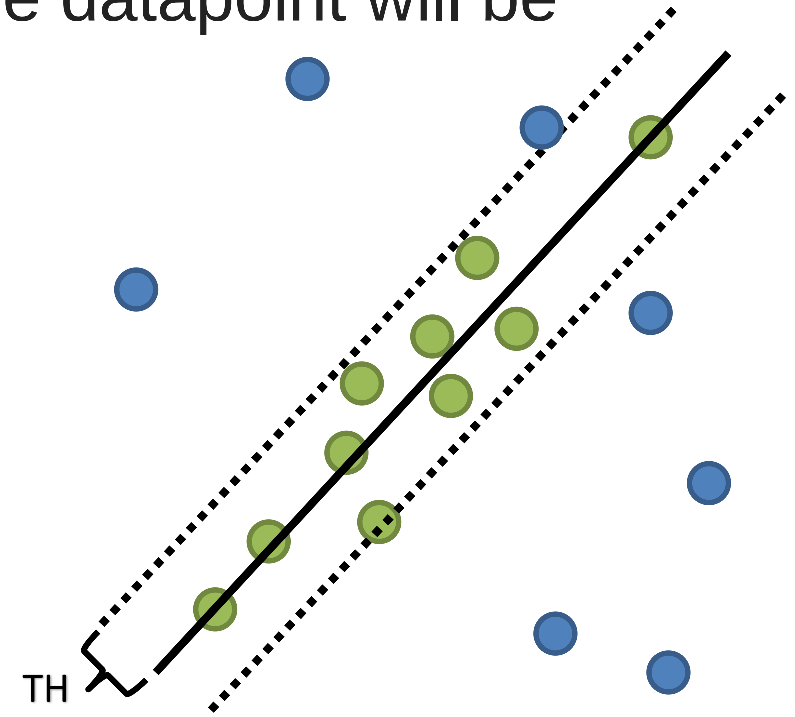
- Can be done using a chosen algorithm (for example LS).



RANSAC algorithm - step by step - 3

Test all dataset against the fitted model. Points that fit the estimated model well, according to some chosen loss function, are considered as part of the consensus set.

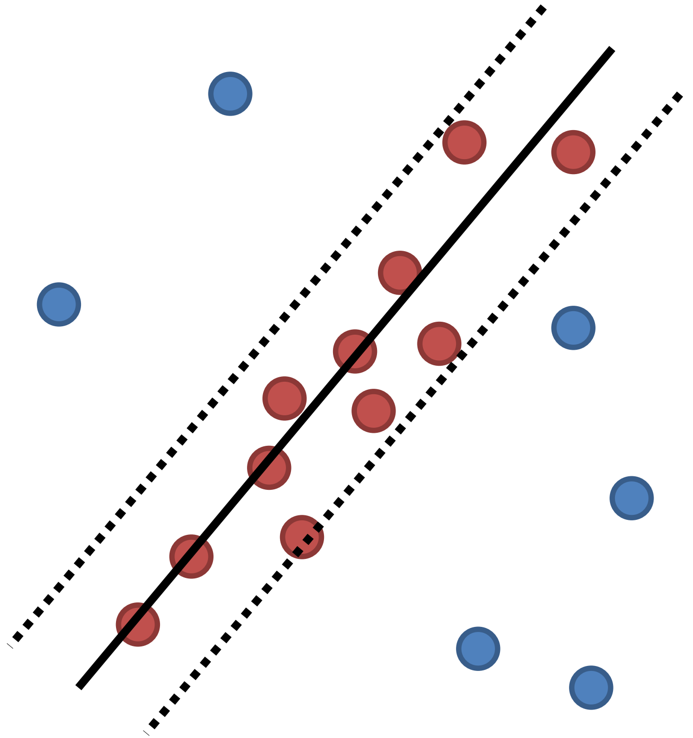
- A possible loss function is MSE of the distances (same as TLS).
  - Choose a threshold for the error: below this TH the datapoint will be consider as an inlier.



RANSAC algorithm - step by step - 4

Save as best model if number of inliers is bigger the old best model number of inliers.

- Trivial…
- An improvement to the final step can be iteratively re-fitting the model with all inliers to find a better describing model.



RANSAC algorithm - step by step - 0

While searching for best model fit.

- How do we know when to stop? How much we need to iterate before getting the best model?
- Let’s look at a possible statistical model for this question.

RANSAC convergence

- Denote

$\omega = \frac{\#\, inliers}{\#\, total\ datapoints}$:

  - getting ω ratio will be considered as reaching the best model.
  - This ratio is usually user specified according to dataset properties (or educated guess).

k: number of iterations (will be calculated).

p: percentage of achieving best model in chosen k iterations (also user specified).

n: number of initially sampled subset datapoints.

RANSAC convergence

- ω^n is the probability that all n points are inliers.
- $1 - \omega^n$ is the probability that at least one of the n points is an outlier.
- $1 - p = (1 - \omega^n)^k$ is the probability that in k iterations we will always have at least one outlier (meaning we didn’t get best match).
- $\boxed{k = \frac{\log(1-p)}{\log(1-\omega^n)}}$