Least Squares

A Quick Recap of SLAM

What is Least-Squares?

- Optimization is very popular in Robotics, CV, and ML domains.
- Optimization problems involve an objective function that we aim to minimize (or maximize) and a set of constraints within which we are expected to obtain the solution.

Example:

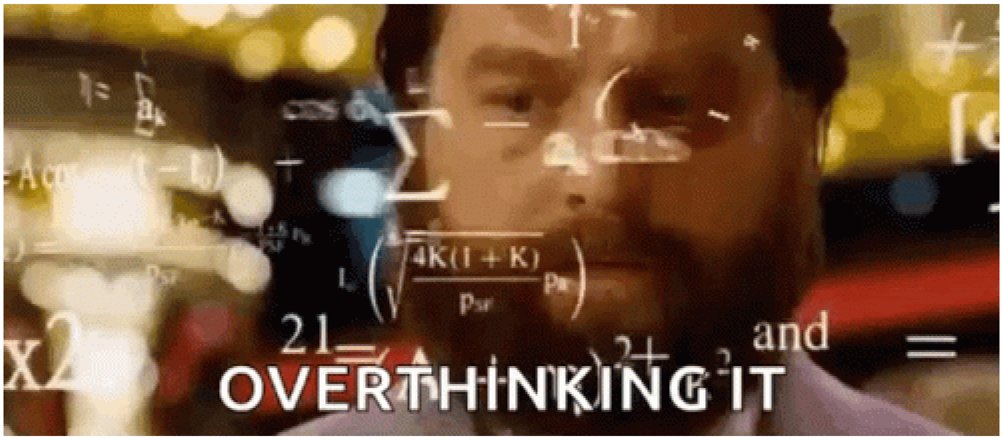
$$(R, \mathbf{t}) = \underset{R \in SO(d),\ \mathbf{t} \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2 ,$$

▼ Does this equation ring a bell do you remember seeing this somewhere recently?

ICP: We wish to find a rigid transformation that optimally aligns the two sets in the least-squares sense, i.e., we seek a rotation **R** and a translation vector **t** such that the point clouds defined by **p** and **q** align.

There are multiple ways to formulate and solve an optimization problem, but for now, we shall focus on one of the most popular methods called **Non-Linear Least Squares** which is a procedure to find the **best-fit state** for a set of data points by **minimizing** the sum of **residuals**.

Non-linear least squares is the form of least squares analysis used to fit a set of *m* observations with a model that is non-linear in *n* unknown parameters ($m \geq n$). It is used in some forms of nonlinear regression. The basis of the method is to approximate the model by a linear one and to refine the parameters by successive iterations.



Let us consider a simple example to understand the above sentence

Overdetermined System = number of observations > number of parameters to be estimated

$$Ax = b$$
$$A\, is\, m \times n \text{ matrix where } m > n.$$

A: Matrix that maps the input state to a given observation

b: is the set of observations

x: is the unknown state that we are interested in estimating

Posing the above problem as least squares

$$S(x) = \arg\min_x \|Ax - b\|_2^2 = \|r\|^2$$

▼ Is there an analytical solution to this example?

Yes, **Pseudo inverse method**

$$\hat{x} = \left(A^T A\right)^{-1} A^T b$$
$$= A^\dagger b \qquad (A^\dagger \text{ is pseudo inverse.})$$

expand S(x) and differentiate it w.r.t x and equate to 0 to obtain this equation.

There are other methods based on SVD and QR decomposition, try them out yourself.

Unfortunately, a closed-form solution exists only for simple problems and we have to resort to iterative methods otherwise.

Optimization Methods

- **Problem Definition**

- Given a system described by a set of n observation functions $\{f_i(\mathbf{x})\}_{i=1:n}$
- Let
  - $\mathbf{x}$ be the state vector
  - $\mathbf{z}_i$ be a measurement of the state $\mathbf{x}$
  - $\hat{\mathbf{z}}_i = f_i(\mathbf{x})$ be a function which maps $\mathbf{x}$ to a predicted measurement $\hat{\mathbf{z}}_i$
- Given n noisy measurements $\mathbf{z}_{1:n}$ about the state $\mathbf{x}$

Goal: Estimate the state $\mathbf{x}$ which bests explains the measurements $\mathbf{z}_{1:n}$

- **Further Explanation:**

$$\mathbf{x} \quad \begin{array}{cc} f_1(\mathbf{x}) = \hat{\mathbf{z}}_1 & \mathbf{z}_1 \\ f_2(\mathbf{x}) = \hat{\mathbf{z}}_2 & \mathbf{z}_2 \\ \ldots & \\ f_n(\mathbf{x}) = \hat{\mathbf{z}}_n & \mathbf{z}_n \end{array}$$

state (unknown)	predicted measurements	real measurements

Can you guess what could be the state, prediction model and the observation (real measurements) for the examples given below

- Bundle Adjustment

  State: Camera Pose and 3D point location

  Prediction Model: Camera model

  Observation: 2D image coordinates

- PNP

  State: Camera Pose

  Prediction Model: Camera Model

  Observation: 2D image Coordinates

- ICP

  State: Relative Transformation

  Prediction Model: Rigid Transformation Model

  Observation: The Target Point Cloud

- **Residual or the Error Function**

- Error $\mathbf{e}_i$ is typically the **difference** between **actual and predicted** measurement

$$\mathbf{e}_i(\mathbf{x}) \quad = \quad \mathbf{z}_i - f_i(\mathbf{x})$$

- We assume that the error has **zero mean** and is **normally distributed**
- Gaussian error with information matrix $\mathbf{\Omega}_i$
- The squared error of a measurement depends only on the state and is a scalar

$$e_i(\mathbf{x}) \quad = \quad \mathbf{e}_i(\mathbf{x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x})$$

- **The goal is to find the minimum value x***

- Find the state $\mathbf{x}^*$ which minimizes the error given all measurements

$$
\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}}\, F(\mathbf{x}) \qquad \boxed{\text{global error (scalar)}} \\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i e_i(\mathbf{x}) \qquad \boxed{\text{squared error terms (scalar)}} \\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{e}_i^T(\mathbf{x})\,\boldsymbol{\Omega}_i\,\mathbf{e}_i(\mathbf{x})
\end{aligned}
$$

$\boxed{\text{error terms (vector)}}$

Optimization Procedure

- Step 1: linearize the error function about the initial guess

- Approximate the error functions around an initial guess $\mathbf{x}$ via Taylor expansion

$$\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \simeq \underbrace{\mathbf{e}_i(\mathbf{x})}_{\mathbf{e}_i} + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x}$$

- Reminder: Jacobian

$$\mathbf{J}_f(x) = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{pmatrix}$$

▼ Step 2: Substitute the linearized error into the objective function

$$\begin{aligned} e_i(\mathbf{x} + \Delta\mathbf{x}) &= \mathbf{e}_i^T(\mathbf{x} + \Delta\mathbf{x})\boldsymbol{\Omega}_i\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \\ &\simeq (\mathbf{e}_i + \mathbf{J}_i\Delta\mathbf{x})^T\boldsymbol{\Omega}_i(\mathbf{e}_i + \mathbf{J}_i\Delta\mathbf{x}) \end{aligned}$$

▼ Step 3: Expand the result in step 2 and group similar terms

$$
\begin{aligned}
e_i(\mathbf{x}+\mathbf{\Delta x}) &= \mathbf{e}_i^T(\mathbf{x}+\mathbf{\Delta x})\mathbf{\Omega}_i\mathbf{e}_i(\mathbf{x}+\mathbf{\Delta x}) \\
&\simeq (\mathbf{e}_i+\mathbf{J}_i\mathbf{\Delta x})^T\mathbf{\Omega}_i(\mathbf{e}_i+\mathbf{J}_i\mathbf{\Delta x}) \\
&= \mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{e}_i + \\
&\quad \mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{J}_i\mathbf{\Delta x} + \mathbf{\Delta x}^T\mathbf{J}_i^T\mathbf{\Omega}_i\mathbf{e}_i + \\
&\quad \mathbf{\Delta x}^T\mathbf{J}_i^T\mathbf{\Omega}_i\mathbf{J}_i\mathbf{\Delta x}
\end{aligned}
$$

$$
\begin{aligned}
&e_i(\mathbf{x}+\mathbf{\Delta x}) \\
&\quad \simeq \mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{e}_i + \\
&\quad\quad \mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{J}_i\mathbf{\Delta x} + \mathbf{\Delta x}^T\mathbf{J}_i^T\mathbf{\Omega}_i\mathbf{e}_i + \\
&\quad\quad \mathbf{\Delta x}^T\mathbf{J}_i^T\mathbf{\Omega}_i\mathbf{J}_i\mathbf{\Delta x} \\
&\quad = \underbrace{\mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{e}_i}_{c_i} + 2\underbrace{\mathbf{e}_i^T\mathbf{\Omega}_i\mathbf{J}_i}_{\mathbf{b}_i^T}\mathbf{\Delta x} + \mathbf{\Delta x}^T\underbrace{\mathbf{J}_i^T\mathbf{\Omega}_i\mathbf{J}_i}_{\mathbf{H}_i}\mathbf{\Delta x} \\
&\quad = c_i + 2\mathbf{b}_i^T\mathbf{\Delta x} + \mathbf{\Delta x}^T\mathbf{H}_i\mathbf{\Delta x}
\end{aligned}
$$

▼ Step 4: Generalize the result of Step 3 for the global error and obtain the Quadratic form

$$
\begin{aligned}
F(\mathbf{x}+\mathbf{\Delta x}) &\simeq \sum_i \left(c_i + \mathbf{b}_i^T\mathbf{\Delta x} + \mathbf{\Delta x}^T\mathbf{H}_i\mathbf{\Delta x}\right) \\
&= \sum_i c_i + 2(\sum_i \mathbf{b}_i^T)\mathbf{\Delta x} + \mathbf{\Delta x}^T(\sum_i \mathbf{H}_i)\mathbf{\Delta x}
\end{aligned}
$$

$$
\begin{aligned}
F(\mathbf{x}+\Delta\mathbf{x}) &\simeq \sum_i \left(c_i + \mathbf{b}_i^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_i\Delta\mathbf{x}\right) \\
&= \underbrace{\sum_i c_i}_{c} + 2\,\underbrace{(\sum_i \mathbf{b}_i^T)}_{\mathbf{b}^T}\,\Delta\mathbf{x} + \Delta\mathbf{x}^T\,\underbrace{(\sum_i \mathbf{H}_i)}_{\mathbf{H}}\,\Delta\mathbf{x} \\
&= c + 2\mathbf{b}^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}
\end{aligned}
$$

with

$$
\begin{aligned}
\mathbf{b}^T &= \sum_i \mathbf{e}_i^T\boldsymbol{\Omega}_i\mathbf{J}_i \\
\mathbf{H} &= \sum_i \mathbf{J}_i^T\boldsymbol{\Omega}\mathbf{J}_i
\end{aligned}
$$

$$
F(\mathbf{x}+\Delta\mathbf{x}) \simeq c + 2\mathbf{b}^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}
$$

▼ Step 5: Minimizing the Quadratic form

- Derivative of $F(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x})$

$$\frac{\partial F(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}} \simeq 2\mathbf{b} + 2\mathbf{H}\boldsymbol{\Delta}\mathbf{x}$$

- Setting it to zero leads to

$$0 = 2\mathbf{b} + 2\mathbf{H}\boldsymbol{\Delta}\mathbf{x}$$

- Which leads to the linear system

$$\mathbf{H}\boldsymbol{\Delta}\mathbf{x} = -\mathbf{b}$$

- The solution for the increment $\boldsymbol{\Delta}\mathbf{x}^*$ is

$$\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$$

▼ **Step 6: The Update Step**

1. **Gradient Descent**

$$\begin{aligned} \Delta x &= -\nabla F(x) \\ &= -\mathbf{J_F} \end{aligned}$$

2. **Gauss-Newton**

- Compute the terms for the linear system $\mathbf{b}^T = \sum_i \mathbf{e}_i^T \boldsymbol{\Omega}_i \mathbf{J}_i \qquad \mathbf{H} = \sum_i \mathbf{J}_i^T \boldsymbol{\Omega}_i \mathbf{J}_i$
- Solve the linear system

$$\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$$

- Updating state $\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\Delta}\mathbf{x}^*$

28

3. **Levenberg Marquardt**

Levenberg Marquardt lies somewhere between Gauss-Newton and Gradient Descent algorithms by blending the two formulations. As a result, when at a steep cliff, LM takes small steps to avoid overshooting, and when on a gentle slope, LM takes bigger steps:

$$\delta x = -(J^T J + \lambda I)^{-1} J^T f(x)$$

- Reduction in error → lambda divided by a factor of 10 & make the update.
- Increase in error → lambda multiplied by a factor of 10 & reject the update.

When lambda is too small, it is essentially the same as Gauss-Newton — will converge faster.

Solved Examples

https://colab.research.google.com/drive/1j2nBqBJep3GiNXgQ2yrhuCUfVNFPsvAP?usp=sharing