From linear algebra to non-linear weighted least squares optimization

Comments	
Dates Taught	@September 4, 2020 → September 11, 2020
Lecture No.	L7 L8 L9
Links of Videos	L7, L8 (Sep 8), L9
Module	SLAM: Smoothing
Related to All Questions (Property)	

> Note-1: Please read this note first.
> Note-2: While this page has the basic theoretical content, more elaborate explanations/visualizations will be added in sometime. Will notify on Moodle after adding.

> Page Author: Shubodh Sai

Using this Notion page

(click on triangle shaped thing to toggle)

- Read this to know how to use this page the right way.
  1. Any logged-in Notion user can comment on this page. This is a wonderful way to collaborate. Whether you have doubts or a better way to explain a concept or even correct a typo, feel free to comment. **There are existing** `TODO` **places where you can contribute.**
  2. Whenever you're studying this page, try to think critically and spend enough time wherever a question is asked before you "toggle" for the answer.
  3. Enable dark mode is a good idea.

Resources

- Resources: Books are referred to by their codenames in this page.

  Books are referred to by their codenames in this page.
  1. Convex Optimisation: Stephen Boyd (referred to as **Book - CO** below)
  2. Applied Linear Algebra: Stephen Boyd (**Book ALA**)
  3. Multi-view Geometry: Zisserman — Appendix 4, 5 & 6 (**Book MVG**)
  4. Numerical Optimization, Jorge Nocedal and Steve Wright (**Book NO**) — Chapter 10
  5. Linear Algebra: Gilbert Strang (especially **Lecture 15**)

Introduction

- Be it Machine Learning or Computer Vision or Robotics problems, a common approach to formulating them is as optimization problems.
  - Formulate an objective function first.
  - Then, one of the most common ways is to pose it as **Minimizing** $squared$ error.
  - Other ways include **Minimizing** *cross-entropy* or **Maximizing** *log-likelihood* (Ex: Deep Learning), **Maximizing** *discounted sum of rewards* (Ex: Reinforcement Learning).
- We look at **Minimizing** $squared$ error.

Linear Least Squares

Linear Algebra: The Overdetermined System

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

Note: We will be considering only full rank matrices. There are special cases like a low rank (like enforcing in F matrix computation), unknown rank systems etc which we won't be covering in this lecture.

In practice, it is full rank (noise) and overdetermined system (data). Assumed throughout this lecture.

▼ r=n (<m): it will span some space but not the whole m space. (overdetermined) — 0 or 1 solution. (Consider for example, m = 3, n = 2)

- there are no free variables so if b lies in column space of A, then 1 solution if it exists or else 0 solution.
  - ▼ (Reasoning - THINK FIRST! and then toggle this)
    1. Think in terms of spanning the space. Does Ax span the entire m-dimensional space?
    2. Another way of reasoning: After row reduction, think of last rows of A — the bottom-most *reduced* rows are all zeros but if the last value of *reduced* b isn't, you'll have 0 solutions. But if solvability condition is satisfied, a solution exists.

Posing it as Least Squares

So what you do is you say, you'd rather minimize the norm which represents the error:

$$||Ax - b||_2$$

▼ But why use ℓ^2- norm in the first place?

Try attempting yourself. Will add more details in some time. TODO

Linear Algebra Perspective:

Just a quick intuition is described here. For deeper understanding, watch this.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cb448965-78ac-4fd2-af53-5aed148ce378/lin_alg_perspective_of_least_squares.mp4

For a quick Linear Algebra intuition, watch this short animation.

▼ Animation — Linear Algebra perspective: Orthogonality Principle

(For the sake of this simple explanation, think of column space of A as a plane)

b doesn't always lie in the column space of A i.e. $Ax = b$ doesn't always have a solution. So, what is the vector ($p = A\hat{x}$) in the column space of A that is closest to b? The projection of b on A!

Say $A = \begin{bmatrix} a_1^T \\ a_2^T \end{bmatrix}$ where a_1 and a_2 are linearly independent vectors. From animation, since

$$e = (b - p) = (b - A\hat{x})$$

is perpendicular to both a_1 and a_2, we have $a_1^T \cdot (b - A\hat{x}) = 0$ and so is the case for a_2. Which means $A^T(b - A\hat{x}) = 0$.

Now see this.

We will be taking the Calculus route in this lecture and not going deeper into the Linear Algebra way.

Calculus Perspective:

Minimizing the error:

$$||Ax - b||_2$$

Ah, but $||Ax - b||_2$ is not differentiable. So I cannot say much about its convexity from this directly.

▼ But what I *can* do, is to square it, as both least-norm and least-square norm are equivalent problems. See this for more details.

(For those who want to go deep) Note: This statement is not false but not really the actual reason why we use "squared error". Will add more details in some time. TODO

So let's minimize

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

where r is known as

$$residual.$$

Expanding

$$x^T(A^TA)x - 2(A^Tb)^Tx + b^Tb$$

Matrix Calculus Basics

> What really is **Linear** here?

Solution

> We solve it analytically.

Pseudo inverse

This is a typical convex quadratic minimization problem.

▼ Why is it convex

▼ (THINK FIRST!)

💬 Why is linear least-squares a convex function?

$$\begin{aligned}
&\Longrightarrow \nabla = 0 \\
&\Longrightarrow \frac{\partial S}{\partial x} = -2A^Tb + 2A^TA\hat{x} = 0 \\
&\Longrightarrow A^TA\hat{x} = A^Tb
\end{aligned}$$

Which are called **normal equations** of the least-squares problem.

$$\begin{aligned}
\hat{x} &= \left(A^TA\right)^{-1}A^Tb \\
&= A^\dagger b \qquad\qquad (A^\dagger \text{ is pseudo inverse.})
\end{aligned}$$

▼ Why A^TA is invertible

▼ (THINK FIRST!)

🕐 A^T A is invertible

▼ Test your understanding

Does $\hat{x} = A^\dagger b$ generally **satisfy** $A\hat{x} = b$?

▼ Answer

No.

Computing using SVD

Objective

Find the least-squares solution to the $m \times n$ set of equations $\mathtt{A}\mathbf{x} = \mathbf{b}$, where $m > n$ and rank $\mathtt{A} = n$.

Algorithm

(i) Find the SVD $\mathtt{A} = \mathtt{UDV}^\mathsf{T}$.
(ii) Set $\mathbf{b}' = \mathtt{U}^\mathsf{T}\mathbf{b}$.
(iii) Find the vector $\mathbf{y}$ defined by $y_i = b'_i/d_i$, where d_i is the i-th diagonal entry of $\mathtt{D}$.
(iv) The solution is $\mathbf{x} = \mathtt{V}\mathbf{y}$.

Book MVG (SVD in detail: Will add more details in some time.)

Computing using QR

Algorithm 12.1 LEAST SQUARES VIA QR FACTORIZATION

given an $m \times n$ matrix A with linearly independent columns and an m-vector b.

1. *QR factorization.* Compute the QR factorization $A = QR$.
2. Compute $Q^T b$.
3. *Back substitution.* Solve the triangular equation $R\hat{x} = Q^T b$.

Book ALA

Orthogonal matrix Q and an upper triangular matrix R.

Non-Linear Least Squares

Problem Definition

$$\text{minimize} \quad \|\mathbf{f}(\mathbf{x})\|^2$$

> What is non-linear here?

$$\implies \text{minimize} \quad F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$$

Solution

> We solve it numerically (heuristics)

All nonlinear optimization algorithms follow the following procedure - Iterative Minimization:

1. Start from ***initial*** *estimate*
2. Each interation - **update** step Δ is calculated
3. **Next** estimate $x^{t+1} = x^t + \Delta$ is obtained.

Stop when convergence criteria is reached (say Δ falls below a small threshold).

Algorithm 9.1 *General descent method.*

given a starting point $x \in \mathbf{dom}\, f$.

repeat

1. Determine a descent direction Δx.
2. *Line search.* Choose a step size $t > 0$.
3. *Update.* $x := x + t\Delta x$.

until stopping criterion is satisfied.

Book A - CO

The search direction in any descent method must satisfy:

$$\nabla F\left(x^{(k)}\right)^T \Delta x^{(k)} < 0$$

Acute angle with the negative gradient

Two important assumptions:

1. Needs good initialization
2. f is locally linear

Table I: Iterative Optimization Methods

Method	Step $\Delta\mathbf{x}$
Gradient Descent	$\Delta\mathbf{x} = -\alpha\mathbf{J_f}^\top\mathbf{f}(\mathbf{x})$
Newton's Method	$\left(\mathbf{H}^\top\mathbf{f} + \mathbf{J}^\top\mathbf{J}\right)\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$
Gauss-Newton	$\mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$
Levenberg-Marquardt	$\left(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I}\right)\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$

J & H are corresponding to f, not F.

Using Gradient Descent

Moving in the opposite direction of the gradient. A natural choice.

$$\begin{aligned}\Delta x &= -\nabla F(x)\\ &= -\mathbf{J_F}\end{aligned} \qquad \text{minimize} \quad F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top\mathbf{f}(\mathbf{x})$$

Matrix Calculus Basics:

🚄 Matrix Calculus

See below page for a solved example on how to compute Jacobian.

Non-Linear LS using GD: Solved example

> Click on this page on next line ⤵️

🏓 Non-Linear Least Squares — Solved example: Computing Jacobian for a Gaussian & Gradient Descent

Using Newton's Method

For $x \in \operatorname{dom} f$, the vector

$$\Delta x_{\mathrm{nt}} = -\nabla^2 F(x)^{-1}\nabla F(x)$$

is called the Newton step (for f, at x).

Newton method's interpretation:

$$\widehat{F}(x+v) = F(x) + \nabla F(x)^T v + \frac{1}{2}v^T\nabla^2 F(x)v$$

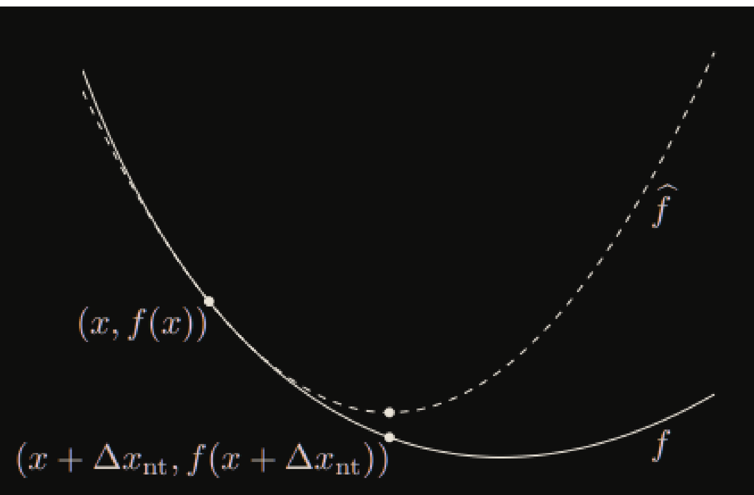

Figure 9.16 The function f (shown solid) and its second-order approximation $\widehat{f}$ at x (dashed). The Newton step Δx_{nt} is what must be added to x to give the minimizer of $\widehat{f}$.

Not to get confused, **"f" is this figure is F in our case**. Book A - CO

Positive definiteness of $\nabla^2 f(x)$ implies that

$$\nabla f\left(x^{(k)}\right)^T \Delta x^{(k)} < 0$$

Acute angle with

$$\nabla F(x)^T \Delta x_{\mathrm{nt}} = -\nabla F(x)^T \nabla^2 F(x)^{-1} \nabla F(x) < 0 \qquad \textbf{the negative gradient}$$

unless $\nabla f(x) = 0$, so **the Newton step is a descent direction (unless x is optimal)!**

$$\begin{aligned} \mathbf{H}_F \Delta \mathbf{x} &= -\mathbf{J}_F \\ \left(\mathbf{H}_{\mathrm{f}}^\top \mathbf{f}(\mathbf{x}) + \mathbf{J}_{\mathrm{f}}^\top \mathbf{J}_{\mathbf{f}}\right) \Delta \mathbf{x} &= -\mathbf{J}_{\mathrm{f}}^\top \mathbf{f}(\mathbf{x}) \end{aligned}$$

Using Gauss Newton

> Linearize f

$$\text{As } F(\mathbf{x}) = \frac{1}{2} \mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}), \text{ we have}$$

$$\begin{aligned} F(\mathbf{x} + \Delta \mathbf{x}) \approx L_{\mathbf{x}}(\Delta \mathbf{x}) &= \frac{1}{2} (\mathbf{f}(\mathbf{x}) + \mathbf{J} \Delta \mathbf{x})^\top \cdot (\mathbf{f}(\mathbf{x}) + \mathbf{J} \Delta \mathbf{x}) \\ &= \frac{1}{2} \mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{x})^\top \mathbf{J} \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^\top \mathbf{J}^\top \mathbf{J} \Delta \mathbf{x} \\ &= F(\mathbf{x}) + \mathbf{f}(\mathbf{x})^\top \mathbf{J} \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^\top \mathbf{J}^\top \mathbf{J} \Delta \mathbf{x} \end{aligned}$$

$$\begin{aligned} 0 = L'(\Delta \mathbf{x}) &= \mathbf{f}(\mathbf{x})^\top \mathbf{J} + \mathbf{J}^\top \mathbf{J} \Delta \mathbf{x} \\ \mathbf{J}^\top \mathbf{J} \Delta \mathbf{x} &= -\mathbf{f}(\mathbf{x})^\top \mathbf{J} \\ \mathbf{J}^\top \mathbf{J} \Delta \mathbf{x} &= -\mathbf{J}^\top \mathbf{f}(\mathbf{x}) \end{aligned}$$

Algorithm 18.1 BASIC GAUSS–NEWTON ALGORITHM FOR NONLINEAR LEAST SQUARES

given a differentiable function $f : \mathbf{R}^n \to \mathbf{R}^m$, an initial point $x^{(1)}$.

For $k = 1, 2, \ldots, k^{\max}$

1. *Form affine approximation at current iterate using calculus.* Evaluate the Jacobian $Df(x^{(k)})$ and define

$$\hat{f}(x; x^{(k)}) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)}).$$

2. *Update iterate using linear least squares.* Set $x^{(k+1)}$ as the minimizer of $\|\hat{f}(x; x^{(k)})\|^2$,

$$x^{(k+1)} = x^{(k)} - \left(Df(x^{(k)})^T Df(x^{(k)})\right)^{-1} Df(x^{(k)})^T f(x^{(k)}).$$

Book ALA

Levenberg Marquardt

> Combination of Gradient Descent and Gauss Newton.

$$\left(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I}\right) \Delta \mathbf{x} = -\mathbf{J}^\top \mathbf{f}(\mathbf{x})$$

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

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

Levenberg–Marquardt algorithm. The ideas above can be formalized as the algorithm given below.

Algorithm 18.3 LEVENBERG–MARQUARDT ALGORITHM FOR NONLINEAR LEAST SQUARES

given a differentiable function $f : \mathbf{R}^n \to \mathbf{R}^m$, an initial point $x^{(1)}$, an initial trust parameter $\lambda^{(1)} > 0$.

For $k = 1, 2, \ldots, k^{\max}$

1. *Form affine approximation at current iterate.* Evaluate the Jacobian $Df(x^{(k)})$ and define
$$\hat{f}(x; x^{(k)}) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)}).$$
2. *Compute tentative iterate.* Set $x^{(k+1)}$ as minimizer of
$$\|\hat{f}(x; x^{(k)})\|^2 + \lambda^{(k)}\|x - x^{(k)}\|^2.$$
3. *Check tentative iterate.*
If $\|f(x^{(k+1)})\|^2 < \|f(x^{(k)})\|^2$, accept iterate and reduce λ: $\lambda^{(k+1)} = 0.8\lambda^{(k)}$.
Otherwise, increase λ and do not update x: $\lambda^{(k+1)} = 2\lambda^{(k)}$ and $x^{(k+1)} = x^{(k)}$.

Book ALA: The factor is different here

Method	Step $\Delta\mathbf{x}$
Gradient Descent	$\Delta\mathbf{x} = -\alpha\mathbf{J}^\top\mathbf{f}(\mathbf{x})$
Newton's Method	$\left(\mathbf{H}^\top\mathbf{f} + \mathbf{J}^\top\mathbf{J}\right)\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$
Gauss-Newton	$\mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$
Levenberg-Marquardt	$\left(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I}\right)\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{f}(\mathbf{x})$

J & H are for f, not F.

Non-Linear Weighted Least Squares

Problem Definition

$$\begin{aligned} F &= \frac{1}{2}\sum_i \|f_i\|^2_{\Sigma_i} = \frac{1}{2}\sum_i \|\mathbf{x}_i - \mathbf{y}_i\|^2_{\mathbf{\Sigma}_i} \\ &= \frac{1}{2}\mathbf{f}^\top\mathbf{\Sigma}^{-1}\mathbf{f} \\ &= \frac{1}{2}\mathbf{f}^\top\mathbf{\Omega}\mathbf{f} \end{aligned}$$

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\ F(\mathbf{x}) = \underset{\mathbf{x}}{\operatorname{argmin}}\ \frac{1}{2}\mathbf{f}(\mathbf{x})^\top\mathbf{\Omega}\mathbf{f}(\mathbf{x})$$

Solution

Gauss Newton/Levenberg Marquardt for Weighted Nonlinear LS

$$\begin{aligned} F(\mathbf{x} + \Delta\mathbf{x}) \approx L_{\mathbf{x}}(\Delta\mathbf{x}) &= \frac{1}{2}(\mathbf{f}(\mathbf{x}) + \mathbf{J}\Delta\mathbf{x})^\top\mathbf{\Omega}(\mathbf{f}(\mathbf{x}) + \mathbf{J}\Delta\mathbf{x}) \\ &= \frac{1}{2}\mathbf{f}(\mathbf{x})^\top\mathbf{\Omega}\mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{x})^\top\mathbf{\Omega}\mathbf{J}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{\Omega}\mathbf{J}\Delta\mathbf{x} \\ &= F(\mathbf{x}) + \mathbf{f}(\mathbf{x})^\top\mathbf{\Omega}\mathbf{J}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{J}^\top\mathbf{\Omega}\mathbf{J}\Delta\mathbf{x} \end{aligned}$$

$$
\begin{aligned}
0 = L'(\Delta \mathbf{x}) &= \mathbf{f}(\mathbf{x})^\top \mathbf{\Omega} \mathbf{J} + \mathbf{J}^\top \mathbf{\Omega} \mathbf{J} \Delta \mathbf{x} \\
\mathbf{J}^\top \mathbf{\Omega} \mathbf{J} \Delta \mathbf{x} &= -\mathbf{f}(\mathbf{x})^\top \mathbf{\Omega} \mathbf{J} \\
\mathbf{J}^\top \mathbf{\Omega} \mathbf{J} \Delta \mathbf{x} &= -\mathbf{J}^\top \mathbf{\Omega}^\top \mathbf{f}(\mathbf{x})
\end{aligned}
$$

GN solution is what we just ended up. LM variant of the above is as follows:

$$
\left(\mathbf{J}^\top \mathbf{\Omega} \mathbf{J} + \lambda \mathbf{I}\right) \Delta \mathbf{x} = -\mathbf{J}^\top \mathbf{\Omega}^\top \mathbf{f}(\mathbf{x})
$$