U09M11109 Visual Navigation for Unmanned Aerial Vehicle (VisNav) **Spring 2023**

Lecture 7-2: Least Squares Optimization

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor(s).*

This lecture centers around **unconstrained least-squares optimization** and primarily discusses:

- relevant background in unconstrained *linear* least-squares optimization;
- relevant background in unconstrained *nonlinear* least-squares optimization;
- the Gauss-Newton method for unconstrained nonlinear least-squares optimization;
- the Levenberg-Marquardt method for unconstrained nonlinear least-squares optimization.

As a reminder, our original motivation for performing nonlinear least-squares is to perform state estimation through *maximum likelihood* or *maximum a posteriori* estimation with nonlinear sensor models. Section 2.5 of [1] is an excellent reference for more information on the topics covered in these notes.

Important: While we will briefly mention constraints in Section 18.1, throughout the remainder of this document we will focus on unconstrained optimization. Unless specifically mentioned, the concepts and algorithms in this document will only apply to unconstrained optimization.

7.1 Preliminaries: Minima and Convexity

Consider the following class of optimization problems:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h_i(\mathbf{x}) \leq b_i, \quad i = 1, \ldots, m \end{aligned} \tag{1}$$

where $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ is the *objective function* (or *cost*) we seek to minimize, and $h_i(\mathbf{x}) \leq b_i$ defines the problem constraints.[1] If a given value of $\mathbf{x}$ satisfies all constraints then it is referred to as a *feasible* point. We refer to the set of all values of $\mathbf{x}$ that satisfy the constraints as the *feasible set* of the optimization problem. That is, the feasible set is $\Omega \triangleq \{\mathbf{x} \mid h_i(\mathbf{x}) \leq b_i, \quad i = 1, \ldots, m\}$. A feasible point that attains the minimum of the cost $f(\mathbf{x})$ is called an *optimal* solution or *global minimum* (more about this in the following section).

7.1.1 Minima

Global minima are feasible points such that there are no other feasible points with a lesser cost, i.e., a point $\mathbf{x}^\star$ is a global minimum if $f(\mathbf{x}^\star) \leq f(\mathbf{x})$ for all feasible $\mathbf{x}$. Global minima are exactly the solution(s) we are seeking when solving problem (1).

[1] Note that equality constraints can be written by simultaneously imposing $h_i(\mathbf{x}) \leq b_i$ and $h_i(\mathbf{x}) \geq b_i$, while other authors prefer adding equality constraints more explicitly in the generic formulation (1).

Local minima are feasible points that only achieve the smallest cost in a small neighborhood around them. That is, $\mathbf{x}$ is a local minimum if $f(\mathbf{x}) \leq f(\mathbf{x}+\boldsymbol{\delta})$ for sufficiently small perturbations $\boldsymbol{\delta}$. Note that for local minima, there might still be feasible points (elsewhere in the feasible set) which do have lesser cost. Local minima are often the "trap-doors" of optimization, as it can be difficult to discern whether a given minimum is a local or global minimum, and suboptimal local minima may correspond to poor estimates of the variables we aim to compute in (1).

In the case of unconstrained optimization (i.e., when there are no constraints on the variables), the gradient of $f(\mathbf{x})$ at both local or global minima is equal to zero.

7.1.2 Convex Optimization Problems

An important notion across all of mathematical optimization is the notion of *convexity*. In optimization, a problem is *convex* if both the objective function and the problem constraints are convex;[2] we define what it means for a function and for a set to be convex below.

Important: The importance of convexity lies in the fact that convex problems can be solved in polynomial time using off-the-shelf algorithms and implementations, e.g., [2], while nonconvex problems are typically hard to solve. Hence convexity is an interesting property to characterize problems that are "easy" to solve.[3] In particular, in convex optimization, every local minimum is also a global minimum, and there are no suboptimal local minima our algorithms can get stuck into.

Convex Functions. A function $f(\mathbf{x})$ is convex if for any two points, $\mathbf{x}_1$ and $\mathbf{x}_2$, the line segment between the values at those points, $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$, will be either above or coincide with the function $f(\mathbf{x})$ for all $\mathbf{x} \in [\mathbf{x}_1, \mathbf{x}_2]$. This is maybe more easily understood by the following drawings in Figure 1.

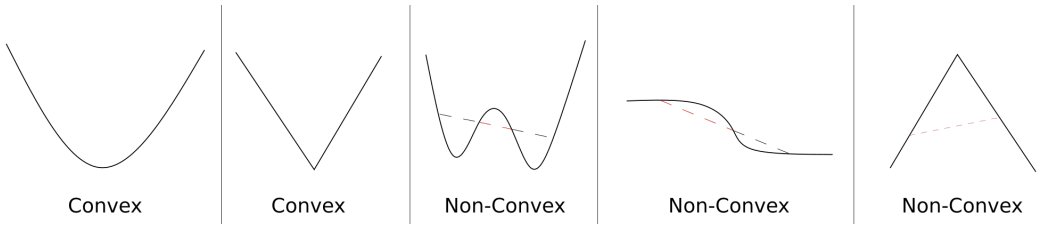

Figure 1: Examples of convex and nonconvex functions, with examples of nonconvexity marked with a red dotted line.

Mathematically, a function $f(\mathbf{x})$ is convex if it satisfies the following relationship in Equation (2) for any two possible values of $\mathbf{x}$ $(\mathbf{x}_1, \mathbf{x}_2)$ and all values of $\lambda \in [0, 1]$:

$$f(\lambda\,\mathbf{x}_1 + (1-\lambda)\,\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \tag{2}$$

Convex (Feasible) Set. A set Ω is convex if for any two points in the set, $\mathbf{x}_1$ and $\mathbf{x}_2$, every point on the straight line connecting those two points is also in Ω. This can be understood by the following diagrams in Figure 2, which show examples of both convex and nonconvex sets.

Mathematically, a set Ω is convex if it satisfies the following relationship in Equation (3) for any two

[2] Saying the constraints are convex is equivalent to saying the feasible set of the optimization problem is convex.

[3] Note that "easy" here refers to the fact that these problems can be solved using well-established polynomial-time algorithms, but the runtime of these algorithms can still be large in some cases.

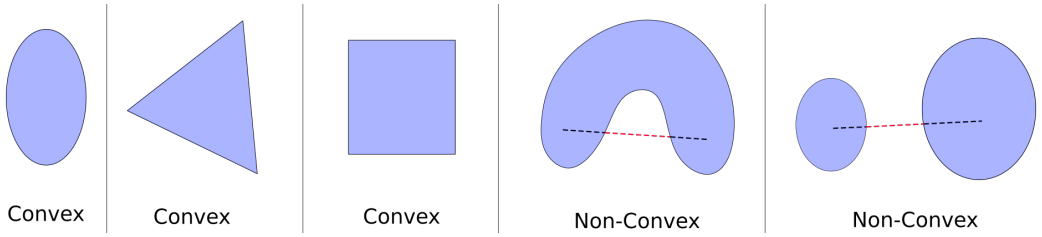

Figure 2: Examples of convex and nonconvex sets, with examples of nonconvexity marked with a red dotted line.

possible values of $\mathbf{x}$ (namely, $\mathbf{x}_1, \mathbf{x}_2$) and all values of $\lambda \in [0, 1]$:

$$\mathbf{x}_1, \mathbf{x}_2 \in \Omega \implies (\lambda\, \mathbf{x}_1 + (1 - \lambda)\, \mathbf{x}_2) \in \Omega \tag{3}$$

7.2 Unconstrained Least-Squares Optimization

Unconstrained least-squares optimization consists in solving an unconstrained optimization problem in which the cost function $f(\mathbf{x})$ can be written as the sum of squared terms:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \quad \sum_{i=1}^{N} \|r_i(\mathbf{x})\|^2 \tag{4}$$

This very general form is generally broken down into two subsets: linear least-squares problems (Section 18.3) and nonlinear least-squares (Section 18.4).

7.3 Linear Least-Squares Optimization

(Unconstrained) Linear least-squares (LLS) optimization is one of the workhorses of modern robotics (and many, many other fields). LLS problems are convex problems and can be written as:

$$\operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{N} \|A_i\, \mathbf{x} \;-\; b_i\|^2 \tag{5}$$

where $A_i \in \mathbb{R}^{m_i \times n}$ and $b_i \in \mathbb{R}^{m_i}$. Let us now stack the A_i matrices into a single matrix, $A \in \mathbb{R}^{m \times n}$ (where $m \triangleq \sum_{i=1}^{N} m_i$), and similarly the b_i vectors into a single vector, $b \in \mathbb{R}^m$:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_N \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_N \end{bmatrix} \tag{6}$$

With this more compact notation we can rewrite (5) more succinctly as:

$$\operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^n} \|A\, \mathbf{x} \;-\; b\|^2 \tag{7}$$

Given the form seen in Equation (7), there is a well-known closed-form solution to LLS problems which we can compute efficiently. This solution to LLS problems comes from solving what are known as the *normal equations*:

$$(A^\top A)\, \mathbf{x} = A^\top b \tag{8}$$

which are easily obtained by setting the gradient of the cost function (7) to zero (recall that in unconstrained optimization minima have zero gradient). The general solution to the normal equations straightforwardly follows:

$$\mathbf{x} = (A^\top A)^{-1} A^\top b \tag{9}$$

The existence of an easily computable closed-form solution is the power of LLS optimization. As we will see in Sections 18.4.3 and 18.4.4, most techniques for solving nonlinear least-squares are actually just solving a sequence of linear least-squares problems.

7.3.1 Practical Notes

Under-constrained and Ill-conditioned Problems. Computing the LLS solution in (9) requires that $A^\top A$ is invertible. In state-estimation problems, $A^\top A$ is invertible only if we have enough measurements to estimate the state of interest ($\mathbf{x}$), a property called *observability*. In GTSAM [1], the optimization library we use in VNAV, a LLS can be formulated as a *GaussianFactorGraph*. When the matrix $A^\top A$ is not invertible, GTSAM will throw an *IndeterminantLinearSystemException*. In addition, even if $A^\top A$ is theoretically invertible, if the ratio between the largest eigenvalue and the smallest eigenvalue is too large this exception may still be thrown, as the system cannot be reliably solved due to numerical issues.

Solving the Normal Equations. In practice, the matrix $(A^\top A)$ should almost **never** be inverted. Instead, the problem is typically solved by linear system solvers via QR or Cholesky decomposition in combination with forward and backward substitution. This is covered in more detail in the robotics context in [1] as well as in the more general context in [3].

Exploiting Sparsity in LLS. In many problems related to robotics, the matrix A in LLS problems is sparse (i.e., many of the entries of the matrix are zero). This is an important observation, since there exist *sparse* linear solvers that can largely reduce the runtime of solving the normal equations if A is sparse. We direct interested readers to [1, 3] for more in-depth explanations.

7.4 Nonlinear Least-Squares Optimization

Nonlinear least-squares (NLS) optimization is ubiquitous in robotic state estimation. NLS assume the following general form:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \|r_i(\mathbf{x})\|^2 \tag{10}$$

where $r_i(\mathbf{x})$ are called the *residual errors* and the objective is the sum of residual errors squared.

In this section we will cover NLS optimization at a high level. For slightly more in depth coverage we also recommend [1] and a great overview given by the Ceres development team[4].

7.4.1 Solving NLS Problems

In general, NLS problems are nonconvex problems which do not admit closed-form solutions and must be solved through iterative (local search) methods. Iterative methods require a starting point (or *initial guess*) for the variables in the NLS problem and will then use some local manipulation of the cost objective function to determine a direction to "step" towards; ideally, after a (hopefully

[4] `http://ceres-solver.org/nnls_solving.html`

small) number of steps, the estimate will converge to a minimum of the NLS problem. If you think of the objective function as defining the "height" over a map of the NLS variables then these methods can be thought of as walking downhill, iteratively taking steps towards the bottom of the "valley" the starting point is within. A general template is given in Algorithm 1.

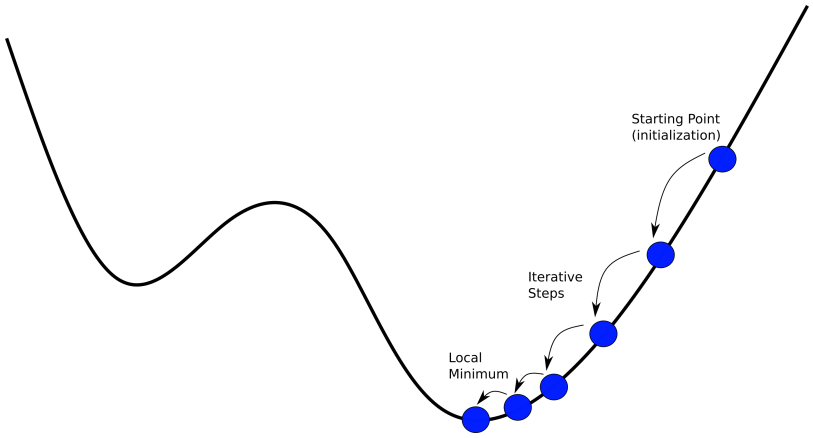

Figure 3: A general local search optimization technique in which a starting point is provided and the algorithm then iteratively takes steps in a descent direction until reaching some local minimum.

Algorithm 1 Descent Methods

1: Given initial guess $\mathbf{x}$
2: **while** Convergence Criteria not Satisfied **do**
3: Choose descent direction: $\boldsymbol{\delta}_x \in \mathbb{R}^n$
4: Choose step size: $t \in \mathbb{R}$
5: $\boldsymbol{\delta} \leftarrow t\boldsymbol{\delta}_x$
6: Update variables: $\mathbf{x} := \mathbf{x} + \boldsymbol{\delta}$

As NLS problems are widely found, much work has gone into developing such iterative solvers. These solvers will commonly differ along two key dimensions: 1) how the step direction is chosen and 2) how the step size is chosen. The NLS approaches we will discuss here will be Newton's method, the Gauss-Newton (GN) method, and the Levenberg-Marquardt (LM) method. Various forms of gradient descent and steepest descent are other common iterative methods for optimization which we do not discuss here but are commonly used for various other problems.

Mathematical Notation. We first begin by briefly defining a few symbols that will reappear throughout these notes. For our given cost function $f(\mathbf{x})$, we will often use a small perturbation, $\boldsymbol{\delta} \in \mathbb{R}^n$, in the variable $\mathbf{x} \in \mathbb{R}^n$ to represent how the cost function will change as $f(\mathbf{x}+\boldsymbol{\delta})$. We will denote the gradient of the cost function at a given $\mathbf{x}$ as $\nabla f(\mathbf{x}) \in \mathbb{R}^n$. Similarly, we will denote the hessian of the cost function (the second derivative) as $\nabla^2 f(\mathbf{x}) \in \mathbb{R}^{n \times n}$.

7.4.2 Newton's Method

At each iteration, Newton's method is simply minimizing the 2nd-order approximation of the cost function $f(\mathbf{x})$, computed at the current estimate of $\mathbf{x}$. Unlike the other approaches we will discuss below, Newton's method applies to any (twice continuously differentiable) real-valued cost function. That is, it is not unique to nonlinear least-squares problems. Given an initial guess $\bar{\mathbf{x}}$, the cost

function $f(\mathbf{x})$ is approximated as:

$$f_{Newton}(\bar{\mathbf{x}} + \boldsymbol{\delta}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top \nabla^2 f(\bar{\mathbf{x}})\boldsymbol{\delta} \tag{11}$$

As this is a quadratic cost function in the variable $\boldsymbol{\delta}$, there is a well-known analytical solution for the $\boldsymbol{\delta}$ to minimize the cost. Effectively, we want to find the $\boldsymbol{\delta}$ such that the derivative of the cost function is equal to zero. By setting the derivative of $f_{Newton}(\bar{\mathbf{x}}+\boldsymbol{\delta})$ to zero we obtain Equation (12)

$$\nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x})\boldsymbol{\delta} = 0 \tag{12}$$

This looks just like the normal equations from our LLS section! From this we can trivially find our optimal $\boldsymbol{\delta}^*$ as described in Section 18.3 (hopefully keeping in mind some of the practical details to make sure the solution is both efficient and numerically stable).

From here the original Newton's method will take a descent step of exactly $\boldsymbol{\delta}^*$, i.e. this determines both descent direction and step size. More common variants will use the direction of $\boldsymbol{\delta}^*$ to define the descent direction but perform what is known as line search to choose a step size in the descent direction.

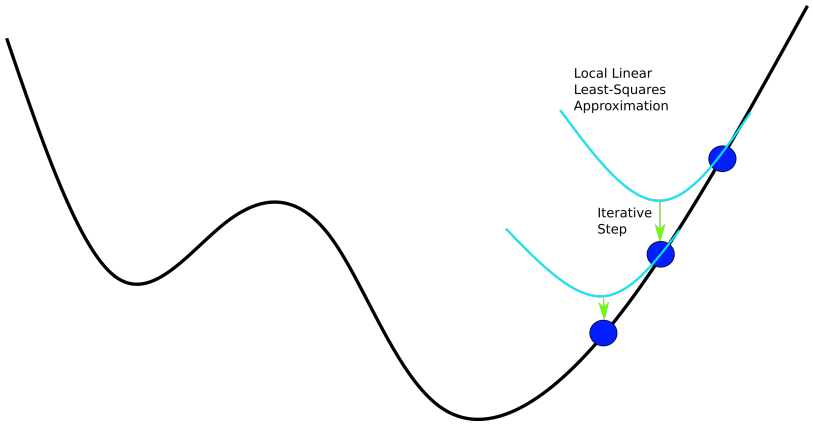

Figure 4: Graphical example of how Newton's method works, by solving iterative quadratic approximations of the cost function.

7.4.3 Gauss-Newton Method

The Gauss-Newton (GN) method can be understood as an approximation of Newton's method in which the Hessian is approximated by just the first-order derivative components. Consider the NLS problem in Equation (10) and assume for simplicity that the residual errors are scalars. In the NLS case, the Hessian can be written as:

$$\nabla^2 f(\mathbf{x}) = \sum_{i=1}^{N} (\nabla r_i(\mathbf{x}) \nabla r_i^\top(\mathbf{x}) + r_i(\mathbf{x}) \nabla^2 r_i(\mathbf{x})) \tag{13}$$

From here the GN method replaces the Hessian by just pruning out the $(r_i(\mathbf{x})\nabla^2 r_i(\mathbf{x}))$ components,

leaving us with an approximation as in Equation (14):

$$\nabla^2 f(\mathbf{x}) \approx \nabla^2_{GN} f(\mathbf{x}) = \sum_{i=1}^{N} (\nabla r_i(\mathbf{x}) \nabla r_i^\top(\mathbf{x})) \tag{14}$$

It is apparent that this approximation is valid under the condition that the pruned components are small. In practice this approximation is generally good if the initial estimate is good (i.e. the residuals are small) and the function is a least-squares problem (i.e. of the form in Equation (10)). It's important to note that the GN method is not guaranteed to converge to a given solution and that a given step in the GN method can actually result in an increase in cost.

An alternative interpretation of the GN method is that at each iteration, GN linearizes the residual errors in Equation (10) around the current estimate $\bar{\mathbf{x}}$:

$$\underset{\boldsymbol{\delta} \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| r_i(\bar{\mathbf{x}}) + \nabla r_i(\bar{\mathbf{x}}) \boldsymbol{\delta} \right\|^2 \tag{15}$$

and then solves the corresponding *linear* least squares problem to compute a step $\boldsymbol{\delta}$. The approach easily generalizes to the case where the residual errors are vector valued, i.e., $r_i(\mathbf{x}) \in \mathbb{R}^{m_i}$, in which case the linear-residual approximation becomes:

$$\underset{\boldsymbol{\delta} \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| r_i(\bar{\mathbf{x}}) + \boldsymbol{J}_i(\bar{\mathbf{x}}) \boldsymbol{\delta} \right\|^2 \tag{16}$$

where the *Jacobian* matrix $\boldsymbol{J}_i \in \mathbb{R}^{m_i \times n}$ stacks (row-wise) the gradients of each entry of $r_i(\mathbf{x})$ with respect to $\mathbf{x}$. If we stack all Jacobians $\boldsymbol{J}_i$ into a single matrix $\boldsymbol{J}$ and all residuals $r_i(\bar{\mathbf{x}})$ into a single vector $r(\bar{\mathbf{x}})$, we can write the previous problem as:

$$\underset{\boldsymbol{\delta} \in \mathbb{R}^n}{\operatorname{argmin}} \left\| \boldsymbol{J}(\bar{\mathbf{x}}) \boldsymbol{\delta} + r(\bar{\mathbf{x}}) \right\|^2 \tag{17}$$

which admits the usual LLS closed-form solution:

$$\boldsymbol{\delta}^\star = - \left(\boldsymbol{J}(\bar{\mathbf{x}})^\top \boldsymbol{J}(\bar{\mathbf{x}}) \right)^{-1} \boldsymbol{J}(\bar{\mathbf{x}})^\top r(\bar{\mathbf{x}}) \tag{18}$$

Algorithm 2 Gauss-Newton Method

1: Given initial guess $\bar{\mathbf{x}}$
2: **while** Convergence Criteria not Satisfied **do**
3: $\quad r_i(\bar{\mathbf{x}}) + \boldsymbol{J}_i(\bar{\mathbf{x}}) \boldsymbol{\delta} \leftarrow$ linearize $r_i(\mathbf{x})$
4: $\quad$ form $\boldsymbol{J}(\bar{\mathbf{x}})$ and $r(\bar{\mathbf{x}})$ by stacking $\boldsymbol{J}_i(\bar{\mathbf{x}})$ and $r_i(\bar{\mathbf{x}})$
5: $\quad \boldsymbol{\delta}^\star \leftarrow \text{solve}\{ \left(\boldsymbol{J}(\bar{\mathbf{x}})^\top \boldsymbol{J}(\bar{\mathbf{x}}) \right) \boldsymbol{\delta} = -\boldsymbol{J}(\bar{\mathbf{x}})^\top r(\bar{\mathbf{x}}) \}$
6: $\quad \bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \boldsymbol{\delta}^\star$

7.4.4 Levenberg-Marquardt Method

The Levenberg-Marquardt (LM) method is another descent method for NLS problems. The LM method is known as a trust-region method as it effectively defines a r adius o ver w hich t he GN approximation is trusted. As the GN approximation is a linear approximation of the cost function, the LM method attempts to account for the accuracy of the linear approximation.

As seen in Algorithm 3, the LM method is effectively solving the GN problem with an additional weighted diagonal component in the linear least-squares step. When this weighted diagonal is substantial (i.e. $\lambda \gg 1$) the LM method is effectively performing gradient descent. When the weighted diagonal component is small (i.e. $\lambda \ll 1$) the LM method is effectively just the Gauss-Newton method. As the weight is adaptively adjusted based on how well the result matches the expectation from the linear approximation, this allows for the trust region to be adaptively controlled while the LM method is being run. This can provide substantial robustness in the solution of NLS problems. Note that there are some details left out of this description, primarily how do we check if the linear approximation is good. This is because there are several different approaches to this and many proposed rules for how to adjust the λ at each step.

Algorithm 3 Levenberg-Marquardt Method

Given initial guess $\bar{\mathbf{x}}$
$\lambda \leftarrow 10^{-3}$
while Convergence Criteria not Satisfied **do**
    $r_i(\bar{\mathbf{x}}) + \boldsymbol{J}_i(\bar{\mathbf{x}})\boldsymbol{\delta} \leftarrow$ linearize $r_i(\mathbf{x})$
    form $\boldsymbol{J}(\bar{\mathbf{x}})$ and $r(\bar{\mathbf{x}})$ by stacking $\boldsymbol{J}_i(\bar{\mathbf{x}})$ and $r_i(\bar{\mathbf{x}})$
    $\boldsymbol{\delta}^\star \leftarrow \text{solve}\{\left(\boldsymbol{J}(\bar{\mathbf{x}})^\top \boldsymbol{J}(\bar{\mathbf{x}}) + \lambda \boldsymbol{I}\right)\boldsymbol{\delta} = -\boldsymbol{J}(\bar{\mathbf{x}})^\top r(\bar{\mathbf{x}})\}$
    if linear approximation is good **then**
        ▷ accept update and increase trust region size
        $\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\delta}^\star$
        $\lambda \leftarrow \lambda/2$
    else
        ▷ decrease trust region size
        $\lambda \leftarrow \lambda * 2$

7.4.5 Practical Notes and Extra Pointers

Local Minima in NLS. Generally, NLS optimization problems will have local minima. This is particularly problematic when we do not have a good initial guess, and we are dealing with high-dimensional functions (both of these are often the case in robotics).

Initialization Procedures for NLS Solvers. The basin of convergence for a given minimum is the set of all variable assignments such that the iterations from those points will converge to that minimum. As the solution quality is effectively determined by which basin the NLS solver is initialized within, there is much interest in techniques to compute high-quality initializations. An excellent overview of some recent initialization approaches for SLAM can be found in [4].

NLS in GTSAM. In GTSAM [1], a NLS can be formulated as a *NonlinearFactorGraph*. The initial guess can be specified by instantiating a suitable *Values* structure. Each term in the objective Equation (10) is referred to as a *factor* in GTSAM.

Under-constrained and Ill-conditioned Problems. As for the linear case, the linear system solved at each iteration of the GN method may not be invertible. This is again due to a lack of observability or to numerical ill-conditioning. For instance, consider a monocular bundle adjustment problem. As we know, the problem can only be solved up to scale, which means that there is an infinite number of solutions that achieve the same optimal cost, resulting in a under-constrained problem. Moreover, in BA we can rotate or translate all the camera poses by the same amount while preserving their relative geometry, which also creates extra ambiguity. Therefore, in these problems it is not uncommon to observe an *IndeterminantLinearSystemException* when using GTSAM.

There are 2 potential solutions to avoid this problem. The first is to add extra measurements (often called "priors") that can remove this ambiguity. For instance, we can put a prior fixing the first pose (which prevents the cost to be invariant to global rotations and translations) and we can put a prior on the relative distance between two camera poses (which removes the scale ambiguity). In alternative, we can use the LM solver which naturally ensures the matrix arising in the normal equations to be invertible by adding the term $\lambda \boldsymbol{I}$ (for some positive λ).

Noise Assumptions in Least-Squares Optimization. Attentive readers may remember that the least-squares problems of interest were originally derived from the assumption of Gaussian additive noise. Interestingly, though the Gaussian noise case is the simplest to derive, many other practical noise models can also be formulated into least-squares problems via maximum *a posteriori* inference and some algebraic cleverness [5].

References

[1] Frank Dellaert and Michael Kaess. "Factor Graphs for Robot Perception". In: *Foundations and Trends in Robotics* 6.1-2 (2017), pp. 1–139. ISSN: 1935-8253. DOI: 10.1561/2300000043.

[2] M. Grant and S. Boyd. *CVX: Matlab software for disciplined convex programming*. URL: http://cvxr.com/cvx.

[3] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Vol. 50. Siam, 1997.

[4] Luca Carlone et al. "Initialization techniques for 3D SLAM: A survey on rotation estimation and its use in pose graph optimization". In: vol. 2015-June. 2015, pp. 4597–4604. ISBN: 9781479969234. DOI: 10.1109/ICRA.2015.7139836.

[5] David M Rosen, Michael Kaess, and John J Leonard. "Robust incremental online inference over sparse factor graphs: Beyond the Gaussian case". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1025–1032.