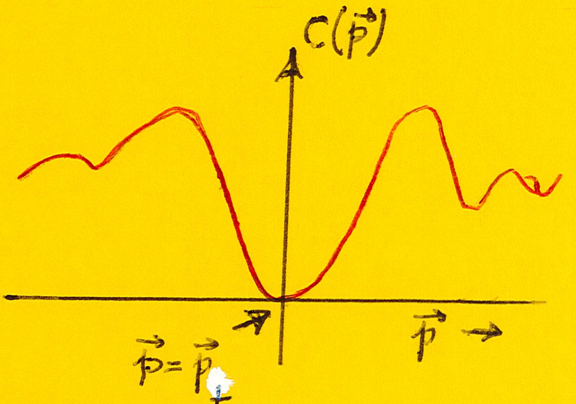
ECE 661
Purdue U.
Avi Kak
Lecture 13

Refining Homographies with Nonlinear Least-Squares Minimization (Gradient-Descent, Levenberg-Marquardt, & DogLeg)

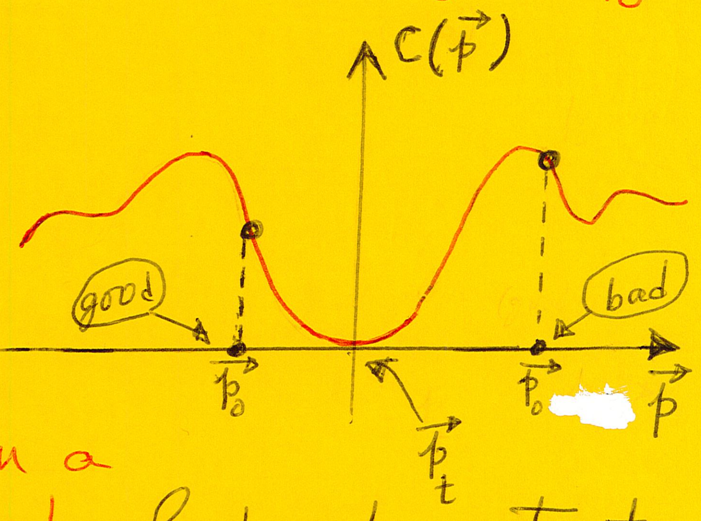
- Recall the least-squares minimization we carried out in Lecture 11 for estimating a homography from noisy correspondences. What we minimized there was $\|A\vec{h}\|$, which is usuall referred to as the $\boxed{\text{algebraic distance}}$. As you saw there, this minimization can be carried out by manipulations of the matrix A of known values. For that reason, we called the approach of Lecture as "linear".
- We will now carry out a different kind of least-square minimization — that is capable of producing more precise homographies than the linear approach of Lecture 11 (provided you start with a good guess for the final solution). For reasons that will soon be clear, we will refer to the new approach as "Nonlinear Least-Squares Minimization."
- Given a correspondence (x, x') in homogeneous coordinates, let the same correspondence in physical coordinates be $(\tilde{x}, \tilde{x}')$ with $\tilde{x} = \binom{x}{y}$ and $\tilde{x}' = \binom{x'}{y'}$. For a homography H, we can express the mapped point Hx as $\boxed{\begin{pmatrix} f_1(h_{11}, h_{12}, \ldots, h_{33}) \\ f_2(h_{11}, h_{12}, \ldots, h_{33}) \end{pmatrix}}$ as a 2-vector in physical coordinates in the range plane. Obviously, we have $\boxed{f_1(h_{11}, \ldots, h_{33}) = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}}$ and $\boxed{f_2(h_{11}, \ldots, h_{33}) = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}}$. Going forward, we will express the mapped point as $\begin{pmatrix} f_1(\vec{p}) \\ f_2(\vec{p}) \end{pmatrix}$ where the vector $\vec{p}$ represents the unknown 9 elements of the homography (or only 8 if you want to assume $h_{33} = 1$). So the square of the error between the actually observed $\tilde{x}' = \binom{x'}{y'}$ and the mapped $\begin{pmatrix} f_1(\vec{p}) \\ f_2(\vec{p}) \end{pmatrix}$ can be expressed as $\|\tilde{x}' - \vec{f}(\vec{p})\|^2$ where $\vec{f}$ is a 2-vector and $\vec{p}$ a 9-vector or an 8-vector) of the unknowns. (Recall that the $\boxed{\text{vector norm}}$ $\|\vec{A}\|$ for a vector $\vec{A}$ is given by the relation $\boxed{\|\vec{A}\|^2 = \vec{A}^T\vec{A}}$.)
- Let's now consider N correspondences between a pair of images. Given a homography H in the form of a vector $\vec{p}$ of unknowns, the square of the physical error for the i^{th} correspondence is $\boxed{\|\tilde{x}'_i - \vec{f}^i(\vec{p})\|^2}$ where $\vec{f}^i$ is again 2-vector that is specific to the i^{th} correspondence. This expression is a sum of two squares, one for the difference along the x-coordinate and the other for the difference along the y-coordinate.
- Let's now consider the summation $\boxed{\sum_{i=1}^{N} \|\tilde{x}' - \vec{f}^i(\vec{p})\|^2}$ as our overall geometrical error in matching the domain points with the range points. Because the x-coordinate difference and the y-coordinate difference appear as two separate squares in what $\|\tilde{x}'_i - \vec{f}^i(\vec{p})\|^2$ stands for, we can re-express the summation $\sum_{i=1}^{N} \|\tilde{x}' - \vec{f}^i(\vec{p})\|^2$ as $\|\vec{X} - \vec{f}(\vec{p})\|^2$ where $\vec{X}$ is

a 2N-vector given by $(x_1', y_1', x_2', y_2', \ldots, x_N', y_N')^T$ and $\vec{f}$ a 2N-vector given by $(f_1^1, f_2^1, f_1^2, f_2^2, \ldots, f_1^N, f_2^N)^T$.

- Our goal is obviously to find the unknown $\vec{p}$ so that the overall geometrical error $\|\vec{X} - \vec{f}(\vec{p})\|^2$ is minimized. We will refer to this error as our cost function $\boxed{C(\vec{p}) = \|\vec{X} - \vec{f}(\vec{p})\|^2}$ in which $\vec{X}$ is the vector of observations and $\vec{f}(\vec{p})$ a vector function of the unknown $\vec{p}$. Obviously, if $\vec{p}$ can be estimated without error, $C(\vec{p}) = 0$. Since we also know that $C(\vec{p}) \geq 0$ for all $\vec{p}$, in the vicinity of the true value $\vec{p} = \vec{p}_t$, we can expect $C(\vec{p})$ to look what is shown at right.



- The reason as to why $C(\vec{p})$ may possess both a global minimum and several local minima is to allow for $\vec{f}(\vec{p})$ to be a non-linear function of $\vec{p}$ – which is indeed the case for homography estimation.
- Since $\vec{f}(\vec{p})$ is, in general, a nonlinear function of $\vec{p}$, we refer to solving the problem $\boxed{\min_{\vec{p}} \|\vec{X} - \vec{f}(\vec{p})\|^2}$ as an exercise in nonlinear least-squares minimization.
- As already mentioned, the cost function $C(\vec{p})$ is the geometric distance between the true solution $\vec{p}_t$ and some solution $\vec{p}$. So, whereas the Linear Least-Squares Minimization of Lecture 11 minimizes the algebraic distance $\|A\vec{h}\|$, a Nonlinear Least-Squares Minimization minimizes the more intuitive geometric distance between the actually observed points in the range plane and their mappings from the domain plane.
- All nonlinear least-squares minimization methods start with some guess $\vec{p}_0$ for the solution and then try to descend down to the true solution along the surface of $C(\vec{p})$. For this to work, the guess $\vec{p}_0$ for the starting point must be sufficiently close to the true solution so that any descent from $\vec{p}_0$ will terminate close to $\vec{p}_t$.



- Therefore making a good starting guess $\vec{p}_0$ is critical to getting a decent result from a nonlinear least-squares minimization method. So how to construct a good guess for $\vec{p}_0$ becomes a big question in the use of the nonlinear least-squares minimization. ▶ For homography estimation, we use the solution returned by linear least-squares minimization as the starting guess $\vec{p}_0$ for a refinement of the solution with nonlinear least-squares minimization.

- In the rest of this lecture, we'll assume that the observation vector $\vec{X}$ in the cost function $C(\vec{p})$ is m-dimensional. We will also assume that the vector $\vec{p}$ of unknowns is n-dimensional. Since $\vec{X}$ is m-dimensional, $\vec{f}(\vec{p})$ must also be m-dimensional. Since $\vec{p}$ is n-dimensional, $C(\vec{p})$ defines a surface in an $(n+1)$ dimensional space. Our goal is to descend down this surface starting from the point that corresponds to $\vec{p}_0$ until we hit the bottom of the valley.
- In the rest of this lecture, I'll go over the following methods for descending down the surface defined by $C(\vec{p})$: ① Gradient-Descent (GD), ② Gauss-Newton (GN), ③ Levenberg-Marquardt (LM), and ④ DogLeg (DL). I'll briefly review the pros and cons of each.

The Gradient-Descent Method (GD)

- Think of $C(\vec{p})$ as a surface defined over an n-dimensional domain. We start at point $\vec{p}_0$ in the domain where the height of the surface is $C(\vec{p}_0)$ and our goal is to take steps from $\vec{p}_0$ to $\vec{p}_1$ to $\vec{p}_2$, and so on, until we reach a point in the domain where the height of the surface is the least.
- In GD, given that we are currently at the domain point $\vec{p}_k$, where the height of the surface is $C(\vec{p}_k)$, we take our next step to the domain point $\boxed{\vec{p}_{k+1} = \vec{p}_k - \gamma_k \nabla C\big|_{\vec{p}=\vec{p}_k}}$ where ∇C is the gradient of the surface at $\vec{p}_k$.

  The negative sign is to take care of the fact that the direction of the gradient is along the steepest ascent on the surface while we want our step to be along the direction of steepest descent. The multiplier γ_k is known as step size controller. In simple implementations of GD, γ_k remains the same for all steps and we stop descending down the surface when the step size $\|\vec{p}_{k-1} - \vec{p}_k\|$ falls below some threshold.

- In more sophisticated implementations, we also compare the height of the surface $C(\vec{p}_{k+1})$ with the previous height $C(\vec{p}_k)$. If the new height is heigher than the previous height, that means we took too large a step in going from $\vec{p}_k$ to $\vec{p}_{k+1}$ in the underlying domain. So, instead of actually taking the step, we halve the value of γ_k and recalculate the step.
- To make the calculation of $\vec{p}_{k+1}$ from $\vec{p}_k$ a bit more explicit, let's rewrite the cost function as the following form: $\boxed{C(\vec{p}) = \vec{\epsilon}^T(\vec{p})\,\vec{\epsilon}(\vec{p})}$ where $\boxed{\vec{\epsilon}(\vec{p}) = \vec{X} - \vec{f}(\vec{p})}$. We can therefore write $\boxed{\nabla C(\vec{p}) = 2 J_{\vec{\epsilon}}^T(\vec{p})\,\vec{\epsilon}(\vec{p})}$

where $J_{\vec{\epsilon}(\vec{p})}$ is the Jacobian of the vector function $\vec{\epsilon}(\vec{p})$. Since the vector $\vec{X}$ in $\vec{\epsilon}(\vec{p}) = \vec{X} - \vec{f}(\vec{p})$ is a constant, we have $\boxed{J_{\vec{\epsilon}(\vec{p})} = -J_{\vec{f}(\vec{p})}}$ where the Jacobian $J_{\vec{f}}$ is given by:

$$\boxed{J_{\vec{f}} = \begin{bmatrix} \frac{\partial f_1}{\partial p_1} & \cdots & \frac{\partial f_1}{\partial p_n} \\ \vdots & & \\ \frac{\partial f_m}{\partial p_1} & \cdots & \frac{\partial f_m}{\partial p_n} \end{bmatrix}}$$

Substituting these in our original formula for $\vec{p}_{k+1}$, we can write

$$\boxed{\vec{p}_{k+1} = \vec{p}_k + 2\gamma_k J^T_{\vec{f}}(\vec{p}_k) \cdot \vec{\epsilon}(\vec{p}_k)}$$

- While GD is easy to implement, it has a major shortcoming that is a direct consequence of the role played by the gradient ∇C in $\boxed{\vec{p}_{k+1} = \vec{p}_k - \gamma_k \nabla C(\vec{p}_k)}$: As you approach the minimum — meaning as ∇C becomes smaller and smaller — your step size $\vec{p}_{k+1} - \vec{p}_k$ will also become smaller and smaller. It is like trying to reach a target point on a straight line by taking discrete steps, with the length of each step being half the distance to the target. At least theoretically, you will never reach the target. Whether or not this becomes a source of inefficiency in a given implementation of GD depends on the choice of the step size controller γ_k.

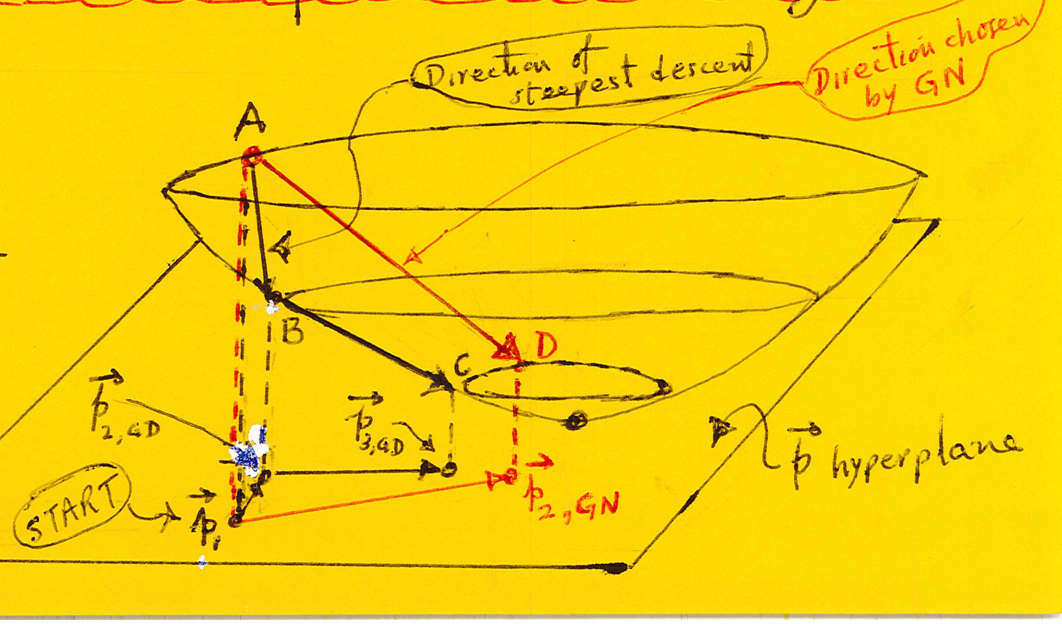
- One final point about GD: Although it is more "fun" to imagine a GD algorithm as descending downhill to the bottom of the valley, from an algorithmic perspective it is more accurate to imagine GD as taking a walk in the "flat" (hyper)plane spanned by $\vec{p}$ as you are looking straight up at the surface $C(\vec{p})$. From the surface properties gleaned from just around the point you see up, you need to decide what direction to take the next step in and what the size of your step should be.

The Gauss-Newton Method (GN)

- GN gives us the beautiful insight that, perhaps contrary to intuition, the quickest way to the minimum is NOT always along the direction of steepest descent.

- However, for GN to work, your solution point must already be close to the minimum.

- To visualize why the direction of steepest descent may not give you the quickest way to the minimum, imagine the minimum at the bottom of a bowl with a very elongated elliptical mouth.



If your starting point in the hyperplane, $\vec{p}_1$, corresponds to point A on the rim of the bowl as shown, the direction of the steepest descent will bring you straight down to the "spine" of the bowl, as shown with the arrow from A to B, and subsequent GD iterations will be along the spine of the bowl. On the other hand, the GN estimate for $\vec{p}_2$ in the $\vec{p}$-hyperplane will correspond to a direct descent down to point D (and possibly to the minimum itself).

- Assuming that you are currently at point $\vec{p}$ in the hyperplane and that your next step will be to $\boxed{\vec{p}+\vec{\delta}_p}$, GN bases its calculation of $\vec{\delta}_p$ on the hope that this step will take you $\boxed{\text{directly}}$ to the minimum. In other words, at any position $\vec{p}$, GN tries to find the $\boxed{\text{best}}$ $\vec{\delta}_p$ for getting directly to the minimum. In practice, it takes multiple such steps to actually get to the minimum. [As you'll see, the "best" will be in the linear least-squares sense.]

- Assuming that your current position $\vec{p}$ is already close to the minimum, we can say that the measurement vector $\vec{X}$ is close in value to $\vec{f}(\vec{p})$ since the optimum $\vec{p}$ is supposed to minimize $\|\vec{X}-\vec{f}(\vec{p})\|^2$. In the same spirit, since the $\vec{\delta}_p$ step is supposed to take us to an even better position vis-a-vis the minimum, $\vec{f}(\vec{p}+\vec{\delta}_p)$ must be an even better approximation to $\vec{X}$. This implies $\boxed{\vec{X} \approx \vec{f}(\vec{p}+\vec{\delta}_p) = \vec{f}(\vec{p}) + J_{\vec{f}} \cdot \vec{\delta}_p}$ where $J_{\vec{f}}$ is the Jocobian of the vector function $\vec{f}(\vec{p})$, as shown at the top on page 13-4. This means that $\vec{\delta}_p$ must be a solution of the system of linear inhomogeneous equations given by $\boxed{J_{\vec{f}} \cdot \vec{\delta}_p = \vec{\epsilon}(\vec{p})}$ where $\vec{\epsilon}(\vec{p})$ is a column vector of known values: $\vec{\epsilon}(\vec{p}) = \vec{X} - \vec{f}(\vec{p})$

- Recall from Lecture 10 that the solution to $A\vec{x} = \vec{b}$ that minimizes $\|A\vec{x}-\vec{b}\|^2$ is given by the pseudoinverse $\boxed{\vec{x} = A^+\vec{b} = (A^TA)^{-1}A^T\vec{b}}$. Recognizing that $A = J_{\vec{f}}$, $\vec{x} = \vec{\delta}_p$, and $\vec{b} = \vec{\epsilon}(\vec{p})$, our best solution for $\vec{\delta}_p$ is $\boxed{\vec{\delta}_p = (J_{\vec{f}}^T J_{\vec{f}})^{-1} J_{\vec{f}}^T \vec{\epsilon}(\vec{p})}$

- Obviously, then, when $J_{\vec{f}}^T J_{\vec{f}}$ is diagonal, the GD and the GN solutions become identical with regard to the direction of the step $\vec{\delta}_p$. The matrix product $J_{\vec{f}}^T J_{\vec{f}}$ becomes diagonal for the case when the cross-derivatives in $J_{\vec{f}}$ are zero, meaning when $\frac{\partial f_i}{\partial p_j}$, $i \neq j$, are zero.

- In addition to the constraint that you must already be close to the final solution, the other constraint on GN is that $J_{\vec{f}}$ cannot be rank deficient, since otherwise $J_{\vec{f}}^T J_{\vec{f}}$ will become singular. Additionally, if you fire up GN when the starting point $\vec{p}$ is still far from the final solution, the value of $\vec{\delta}_p$ as calculated by GN could be meaningless.

The Levenberg-Marquardt Method (LM)

- LM tries to combine the best of GD with the best of GN. If the starting point is far from the minimum, LM behaves like GD. But, then, as the solution point approaches the minimum, LM acts like GN.

- To understand how LM can switch from GN to GD, let's reexamine the GN solution $\boxed{\vec{\delta}_p = (J_{\vec f}^T J_{\vec f})^{-1} J_{\vec f}^T \vec{\epsilon}(\vec p)}$. We will re-write this solution as $\boxed{(J_{\vec f}^T J_{\vec f})\vec{\delta}_p = J_{\vec f}^T \vec{\epsilon}(\vec p)}$. So far we are still exclusively with GN. Focusing on the fact that, if $J_{\vec f}^T J_{\vec f}$ were purely diagonal, the solution returned by GN would be the same as by GD, let's now heuristically extend the last equation so that it can "admit" both GN and GD solutions: $$\boxed{(\underbrace{J_{\vec f}^T J_{\vec f}}_{n\times n} + \underset{\text{damping coefficient}}{\mu} \underbrace{I}_{\substack{n\times n\\ \text{diagonal}}})\underbrace{\vec{\delta}_p}_{n\times 1} = \underbrace{J_{\vec f}^T}_{n\times m}\underbrace{\vec{\epsilon}(\vec p_k)}_{m\times 1}}$$ Augmented Normal Equation for solving for $\vec{\delta}_p$
where we refer to the modification of $J_{\vec f}^T J_{\vec f}$ by the addition of μI as damping.

- In our new system of equations for $\vec{\delta}_p$ if the damping coefficient μ is much larger than the diagonal elements of $J_{\vec f}^T J_{\vec f}$ (and also much larger than the other elements of $J_{\vec f}^T J_{\vec f}$), the solution returned by the Augmented Normal Equations will be close to the GD. On the other hand, if $\mu = 0$, the solution returned will be the same as GN.

- Starting with an initial guess $\vec p_0$, LM at each iteration k consists of first setting a value for the damping coefficient μ_k and then setting setting $\vec{\delta}_p$ to $\boxed{\vec{\delta}_p = (J_{\vec f}^T J_{\vec f} + \mu I)^{-1} J_{\vec f}^T \vec{\epsilon}(\vec p_k)}$. The new value for the solution would then become $\vec p_{k+1} = \vec p_k + \vec{\delta}_p$.

- The main challenge in using LM is to figure out how to set the damping coefficient at each iteration.

- On the face of it, there is a bit of circular reasoning involved in the calculation of the best μ to use in going from $\vec p_k$ to $\vec p_{k+1}$. The $\vec{\delta}_p$ increment obviously depends on μ as shown above. However, the best μ to use depends on $\vec{\delta}_p$ as you will soon see. So we proceed as follows:

- When we are at $\vec p_k$, we assume we have already calculated μ_k. We use this μ_k to calucate $\boxed{\vec{\delta}_p = (J_{\vec f}^T J_{\vec f} + \mu_k I)^{-1} J_{\vec f}^T \vec{\epsilon}(\vec p_k)}$, and tentatively we set $\vec p_{k+1} = \vec p_k + \vec{\delta}_p$. Next, we proceed to evaluate the "quality" of this $\vec{\delta}_p$ as described below. Should the quality test not pass, we revert the solution point back to $\vec p_k$. That is, we set $\vec p_{k+1} = \vec p_k$ and, at the same time, we set $\mu_{k+1} = 2\mu_k$. In this manner, we re-compute the step size at $\vec p_k$ with a value for the damping coefficient that is twice the previous value. As you know, making the damping coefficient larger means putting greater faith in GD.

- The quality of a computed $\vec{\delta}_p$ is tested primarily by the sign of $\boxed{C(\vec{p}_k) - C(\vec{p}_{k+1})}$. If the sign is negative, we have probably "jumped" over the minimum with the step direction we used at $\vec{p}_k$. So we should go back to $\vec{p}_k$ and try a step direction that is more aligned with GD than with GN. (Remember, at the beginning of the iterations, we start out by being primarily aligned with GD in our steps. So, presumably, we got into trouble as we steered too far from GD to GN.

- More precisely, the quality of a computed $\vec{\delta}_p$ is tested using the ratio of the actual change in the cost function (meaning $C(\vec{p}_k) - C(\vec{p}_{k+1})$) to the change in the cost function predicted by a particular choice for μ_k. The denominator of this ratio requires us to express $C(\vec{p}_k) - C(\vec{p}_{k+1})$ in terms of μ_k. Using for the denominator the result derived in the box at right, we write the following formula for the ratio:

$$\rho^{LM}_{k+1} = \frac{C(\vec{p}_k) - C(\vec{p}_{k+1})}{\vec{\delta}_p^T J_f^T \vec{\epsilon}(\vec{p}_k) + \vec{\delta}_p^T \mu_k I \vec{\delta}_p}$$

We note that $C(\vec{p}) = \vec{\epsilon}^T(\vec{p})\,\vec{\epsilon}(\vec{p})$. Therefore,

$$C(\vec{p}_k) - C(\vec{p}_{k+1}) = \vec{\epsilon}^T(\vec{p}_k)\vec{\epsilon}(\vec{p}_k) - \vec{\epsilon}^T(\vec{p}_{k+1})\vec{\epsilon}(\vec{p}_{k+1})$$

However,

$$\vec{\epsilon}(\vec{p}_{k+1}) = \vec{\epsilon}(\vec{p}_k) + J_{\vec{\epsilon}}\,\vec{\delta}_p = \vec{\epsilon}(\vec{p}_k) - J_{\vec{f}}\,\vec{\delta}_p$$

Therefore, by substitution we can show

$$C(\vec{p}_k) - C(\vec{p}_{k+1}) = 2\vec{\delta}_p^T J_f^T \vec{\epsilon}(\vec{p}_k) - \vec{\delta}_p^T J_f^T J_f \vec{\delta}_p$$

Next, we add and subtract the term $\vec{\delta}_p^T \mu_k I \vec{\delta}_p$ from the RHS to get

$$C(\vec{p}_k) - C(\vec{p}_{k+1}) = 2\vec{\delta}_p^T J_f^T \vec{\epsilon}(\vec{p}_k) - \vec{\delta}_p^T (J_f^T J_f + \mu_k I)\vec{\delta}_p + \vec{\delta}_p^T \mu_k I \vec{\delta}_p$$

From the LM formula for estimating $\vec{\delta}_p$, we note $(J_f^T J_f + \mu_k I)\vec{\delta}_p = J_f^T \vec{\epsilon}(\vec{p}_k)$

Substituting this in the equation for the cost function increment, we get

$$C(\vec{p}_k) - C(\vec{p}_{k+1}) = \vec{\delta}_p^T J_f^T \vec{\epsilon}(\vec{p}_k) + \vec{\delta}_p^T \mu_k I \vec{\delta}_p$$

The value of ρ^{LM}_{k+1} thus obtained is used to calculate the damping coefficient for the next iteration through the following formula:

$$\boxed{\mu_{k+1} = \mu_k \cdot \max\left\{\frac{1}{3},\ 1 - (2\rho^{LM}_{k+1} - 1)^3\right\}}$$

The important point to note here is to see as to what happens when ρ^{LM}_{k+1} is negative and when it is positive. When ρ^{LM}_{k+1} is negative, $2\rho^{LM}_{k+1} - 1$ is negative and its cubed power remains negative, with its magnitudes being greater than 1. As a result, the second arg to 'max' will exceed 2. Consequently, we will have $\mu_{k+1} \geq 2\mu_k$, which implies a strong steer in the direction of GD. Let's now consider the case when ρ^{LM}_{k+1} is strongly positive. Say $\rho^{LM}_{k+1} = 2$. In this case, the second arg to 'max' will be -26. Consequently, $\mu_{k+1} = \frac{\mu_k}{3}$, implying a conservative steer of the solution toward GN. You will notice that for small positive values of ρ^{LM}_{k+1}, $0 \leq \rho^{LM}_{k+1} \leq 0.5$, we have $1 - (2\rho^{LM}_{k+1} - 1)^3 \geq 1$, implying $\mu_{k+1} \geq \mu_k$. What that means is that even when we know we are descending downhill, if the numerator of ρ^{LM}_{k+1} is not comparable to the denominator, we continue to bet on the numerical safety of GD.

- That leaves us with just the issue of the initialization of the sequence μ_k

of damping coefficients. The very first value, μ_0, is estimated by setting it to $\boxed{\mu_0 = \tau \cdot \max\{\text{diag}(J_{\vec{f}}^T J_{\vec{f}})\}}$ for some $0 < \tau \leq 1$. You can think of the largest value on the diagonal of $J_{\vec{f}}^T J_{\vec{f}}$ as a scaling parameter. The diagonal elements of $J_{\vec{f}}^T J_{\vec{f}}$ only involve the straight derivatives of $\vec{f}(\vec{p})$ with respect to $p_1, p_2, \ldots, p_n$. The largest value on the diagonal corresponds to that cardinal direction along which the cost function $C(\vec{p}) = \vec{\epsilon}^T(\vec{p})\vec{\epsilon}(\vec{p})$ has the steepest descent.

- Finally, note that we steer the path toward GD whenever we suspect LM is not doing a good enough job. That is because GD has safer convergence, meaning that it is guaranteed to get you to the minimum even if takes a long time to get there (note we are NOT using the "step-size controller"). GN has must faster convergence near the minimum, but it can behave erratically if you get its parameters wrong. LM is stopped when either $\|J\|$ drops below some threshold, or when $\|\vec{\delta}_p\|$ drops below a threshold, or when max iterations reached.

The DogLeg Method (DL)

- DL uses a "formal" framework for combining the best of GD with the best of GN.
- DL uses the notion of a trust region of radius r_k around the point $\vec{p}_k$ where we are currently in the $\vec{p}$-hyperplane. The trust region is supposed to answer the question of how far we can step away from $\vec{p}_k$ before the error between the actual values of $C(\vec{p})$ and the values estimated with the first-order derivatives at $\vec{p}_k$ becomes too large.
- For DL, while we are at $\vec{p}_k$, we first calculate the following two different possibilities for the step to the next point $\vec{p}_{k+1}$:

$$\boxed{\vec{\delta}_{p,GD} = \frac{\|J_{\vec{f}}^T \vec{\epsilon}(\vec{p}_k)\|}{\|J_{\vec{f}} J_{\vec{f}}^T \vec{\epsilon}(\vec{p}_k)\|} J_{\vec{f}}^T \vec{\epsilon}(\vec{p}_k)} \qquad \boxed{\vec{\delta}_{p,GN} = \frac{1}{J_{\vec{f}}^T J_{\vec{f}} + \mu_k I} J_{\vec{f}}^T \vec{\epsilon}(\vec{p}_k)}$$

- Assuming we know the value of the radius of trust r_k around $\vec{p}_k$, we next use the logic shown at right to set $\vec{p}_{k+1}$. Of the three cases shown, in the first case, if the GN jump falls within the region of trust, we go ahead and take it (and hopefully that will be the end of the iterations. The second case applies if the GD jump is within the trust region, but the GN jump is outside. Now we first move to the point where the GD jump takes us and we make a dogleg in the direction of the GN jump but only through a distance that stops at the perimeter of the trust region. The parameter β is to ensure that we stop at the perimeter. For the third case, both GN and GD are outside the trust radius; so we fall back on GD but stop at the perimeter of trust.

$$\vec{p}_{k+1} = \vec{p}_k + \begin{cases} \vec{\delta}_{p,GN} & \text{if } \|\vec{\delta}_{p,GN}\| < r_k \\ \vec{\delta}_{p,GD} + \beta(\vec{\delta}_{p,GN} - \vec{\delta}_{p,GD}) & \text{if } \|\vec{\delta}_{p,GD}\| < r_k < \|\vec{\delta}_{p,GN}\| \\ \frac{r_k}{\|\vec{\delta}_{p,GD}\|} \cdot \vec{\delta}_{p,GD} & \text{otherwise} \end{cases}$$

- To obtain β, we want $\boxed{\|\vec{\delta}_{p,GD} + \beta(\vec{\delta}_{p,GN} - \vec{\delta}_{p,GD})\|^2 = r_k^2}$ which reduces to the quadratic form: $\|\vec{\delta}_{p,GD}\|^2 + \beta^2 \|\vec{\delta}_{p,GN} - \vec{\delta}_{p,GD}\|^2 + \beta\, 2\vec{\delta}_{p,GD}^T(\vec{\delta}_{p,GN} - \vec{\delta}_{p,GD}) = r_k^2$
- Finally, to set r_{k+1} from r_k, we first calculate the ratio $\boxed{\rho^{DL} = \frac{C(\vec{p}_k) - C(\vec{p}_{k+1})}{2\vec{\delta}_p^T J_{\vec{f}}^T \vec{\epsilon}(\vec{p}_k) - \vec{\delta}_p^T J_{\vec{f}}^T J_{\vec{f}} \vec{\delta}_p}}$ ← Denom. in 6th line of box prev. page

We set $\boxed{r_{k+1} = \begin{cases} r_k/4 & \text{if } \rho^{DL} < \frac{1}{4} \\ r_k & \text{if } \frac{1}{4} < \rho^{DL} \leq 3/4 \\ 2r_k & \text{otherwise} \end{cases}}$ If $\rho^{DL} \leq 0$ we have hopped over the minimum. So $\vec{p}_{k+1} = \vec{p}_k$ and $r_{k+1} = r_k/2$.

No hard and fast rules for the initialization of r_0. set $r_0 = 1$ and let the algo take it from there.