Aalto University
School of Electrical
Engineering

ELEC-E8740 — Gauss-Newton with Line Search and Levenberg-Marquardt Algorithm

Simo Särkkä

Aalto University

October 2, 2019

Contents

Intended Learning Outcomes

After this lecture, you will be able to:

- understand the principle of line search in Gauss–Newton method;
- understand the principle of Levenberg–Marquardt algorithm;
- apply Gauss–Newton method with line search and Levenberg–Marquardt algorithm to sensor fusion problems.

Recap (1)

- Sensor fusion problems are often nonlinear.
- General nonlinear model has the form:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r},$$

- General cost function that we considered:

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- Gradient descent algorithm takes steps towards the direction of negative gradient.
- Gauss–Newton iteratively linearizes the model and solves the linear optimization problem.

Recap (2)

Algorithm 1 Gradient Descent

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data $\mathbf{y}$, function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}(\mathbf{x})$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{\text{WLS}}$

1: Set $i \leftarrow 0$

2: **repeat**

3: Calculate the update direction

$$\Delta\mathbf{x}^{(i+1)} = \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\,\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

4: Select a suitable $\gamma^{(i+1)}$

5: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma^{(i+1)}\Delta\mathbf{x}^{(i+1)}$$

6: Set $i \leftarrow i + 1$

7: **until** Converged

Recap (3)

Algorithm 2 Gauss–Newton Algorithm

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data $\mathbf{y}$, function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_\mathbf{x}$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{\text{WLS}}$

1: Set $i \leftarrow 0$

2: **repeat**

3: Calculate the update direction

$$\Delta\mathbf{x}^{(i+1)} = (\mathbf{G}_\mathbf{x}^\mathsf{T}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_\mathbf{x}(\hat{\mathbf{x}}^{(i)}))^{-1}\mathbf{G}_\mathbf{x}^\mathsf{T}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

4: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta\mathbf{x}^{(i+1)}$$

5: Set $i \leftarrow i + 1$

6: **until** Converged

7: Set $\hat{\mathbf{x}}_{\text{WLS}} = \hat{\mathbf{x}}^{(i)}$

Gauss–Newton Algorithm with Line Search: Motivation

- Gauss–Newton uses linearization to determine the next iterate.
- As linearization is a local approximation, taking the full step might over/undershoot.
- Instead, we can use scaled Gauss–Newton step:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \Delta\hat{\mathbf{x}}^{(i+1)},$$
$$\Delta\hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}))^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})),$$

- Here γ is the scaling factor – typically $\gamma \in [0, 1]$.
- How should we select the scaling factor?

Gauss–Newton Algorithm with Line Search: Derivation

- One way to select the scaling parameter γ is to use a line search.
- Given the Gauss–Newton increment $\Delta\hat{\mathbf{x}}^{(i+1)}$, we can consider cost as function of the scale parameter:

$$J_{\mathrm{WLS}}^{(i)}(\gamma) = J_{\mathrm{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma\Delta\hat{\mathbf{x}}^{(i+1)}\right).$$

- Then the idea of line search is to locally optimize the above function in the range $[0, 1]$.
- In the exact line search we simply find the minimum e.g. by evaluating it in grid.
- We could also use bisection algorithm or interpolation methods to find the minimum.

Exact Line Search on Grid

Algorithm 3 Line Search on Grid

Require: Previous iterate $\hat{\mathbf{x}}^{(i)}$, the update direction $\Delta\hat{\mathbf{x}}^{(i+1)}$, the cost function $J_{\text{WLS}}(\mathbf{x})$, and the grid size N_g.

Ensure: Optimal step size γ^*.

1: Set $\gamma^* \leftarrow 0$ and $J^* \leftarrow J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)}\right)$
2: **for** $j \in \{1, 2, \ldots, N_g\}$ **do**
3: Set $\gamma \leftarrow j/N_g$
4: **if** $J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma\Delta\hat{\mathbf{x}}^{(i+1)}\right) < J^*$ **then**
5: Set $\gamma^* \leftarrow \gamma$
6: **end if**
7: **end for**

Inexact Line Search (1/2)

- The line search does not need to be exact to guarantee to find the minimum.
- In backtracking we decrease the parameter γ until it provides a sufficient decrease in the cost.
- One way is to halve the step size until the cost decreases.
- In Armijo backtracking we demand that the cost is decreased at least with an amount that is predicted by a first order Taylor series expansion.
- The first order Taylor series expansion for the cost as function of scale parameter gives:

$$J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta\hat{\mathbf{x}}^{(i+1)}\right) \approx J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)}\right) - 2\gamma[\Delta\hat{\mathbf{x}}^{(i+1)}]^{\mathsf{T}}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

Inexact Line Search (2/2)

- Then we demand that the cost decrease should satisfy the Armijo condition

$$J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta\hat{\mathbf{x}}^{(i+1)}\right) - J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)}\right) \leq -2\beta\,\gamma [\Delta\hat{\mathbf{x}}^{(i+1)}]^{\mathsf{T}} \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- Here β is a parameter that we can choose freely on range $[0, 1)$ (e.g. $\beta = 0.1$).
- We then decrease γ by multiplying it with a parameter τ (e.g., $\tau = 0.5$) on the range $(0, 1)$ until the condition is satisfied:

$$\gamma \leftarrow \tau\,\gamma.$$

Line Search with Armijo Backtracking

Algorithm 4 Line Search with Armijo Backtracking

Require: Previous iterate $\hat{\mathbf{x}}^{(i)}$, the update direction $\Delta\hat{\mathbf{x}}^{(i+1)}$, the cost function $J_{\text{WLS}}(\mathbf{x})$, and parameters β and τ.

Ensure: Suitable step size γ.

1: Set $\gamma \leftarrow 1$ and $J_0 \leftarrow J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)}\right)$.
2: Set $d \leftarrow -2\beta\,[\Delta\hat{\mathbf{x}}^{(i+1)}]^{\mathsf{T}}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$
3: **while** $J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma\Delta\hat{\mathbf{x}}^{(i+1)}\right) > J_0 + \gamma\, d$ **do**
4: Set $\gamma \leftarrow \tau\,\gamma$
5: **end while**

Gauss–Newton Algorithm with Line Search

Algorithm 5 Gauss–Newton Algorithm with Line Search

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data $\mathbf{y}$, function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}(\mathbf{x})$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{\text{WLS}}$

1: Set $i \leftarrow 0$

2: **repeat**

3: Calculate the update direction

$$\Delta\mathbf{x}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}))^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

4: Compute optimal $\gamma^{(i+1)}$ with line search (Algorithm 3 or 4)

5: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma^{(i+1)}\Delta\mathbf{x}^{(i+1)}$$

6: Set $i \leftarrow i + 1$

7: **until** Converged

8: Set $\hat{\mathbf{x}}_{\text{WLS}} = \hat{\mathbf{x}}^{(i)}$

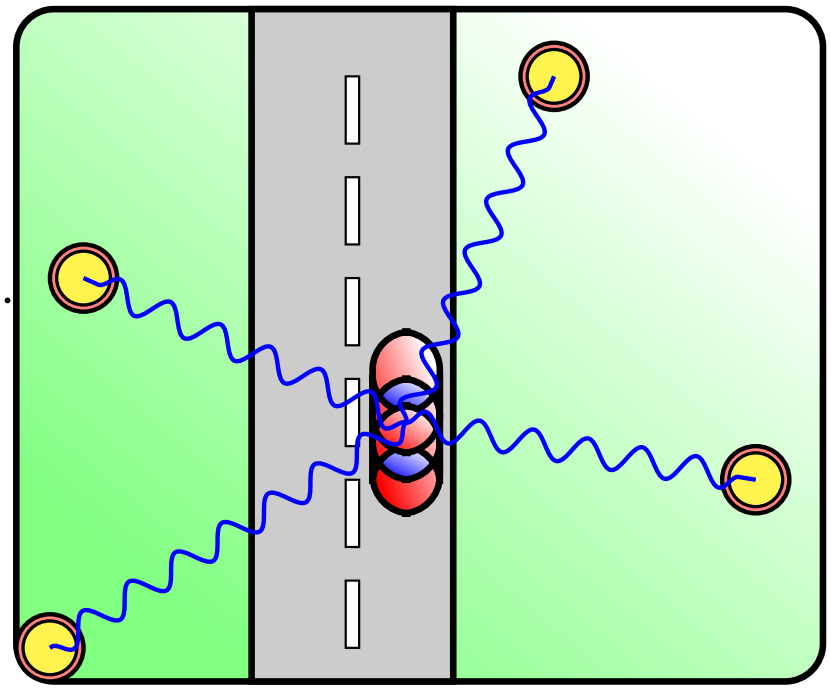
Gauss–Newton Algorithm with Line Search: Example (1/3)

- We measure the range to each landmark:

$$y_1^R = \sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2} + r_1^R,$$

$$\vdots$$

$$y_M^R = \sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2} + r_M^R.$$



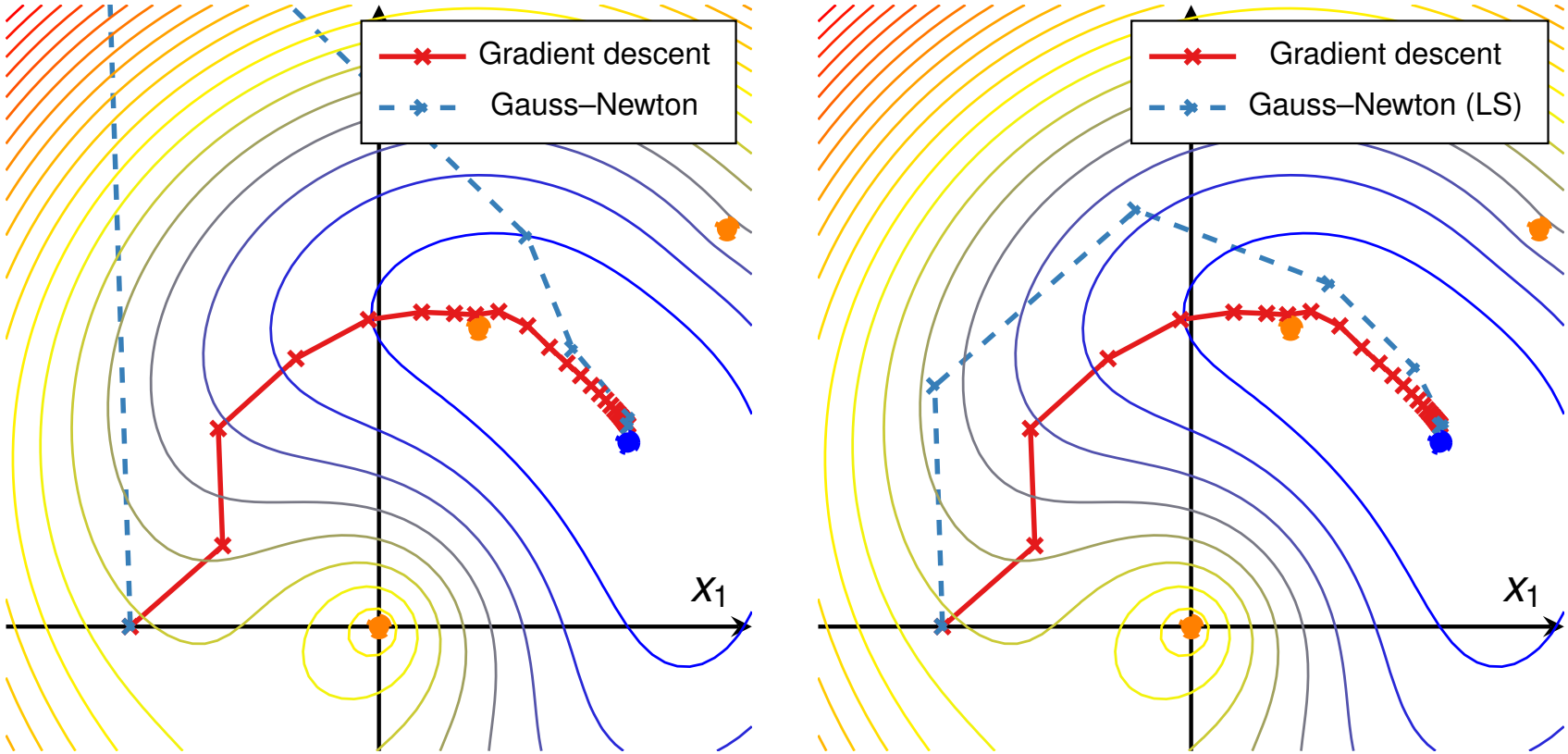
- This is a non-linear model

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

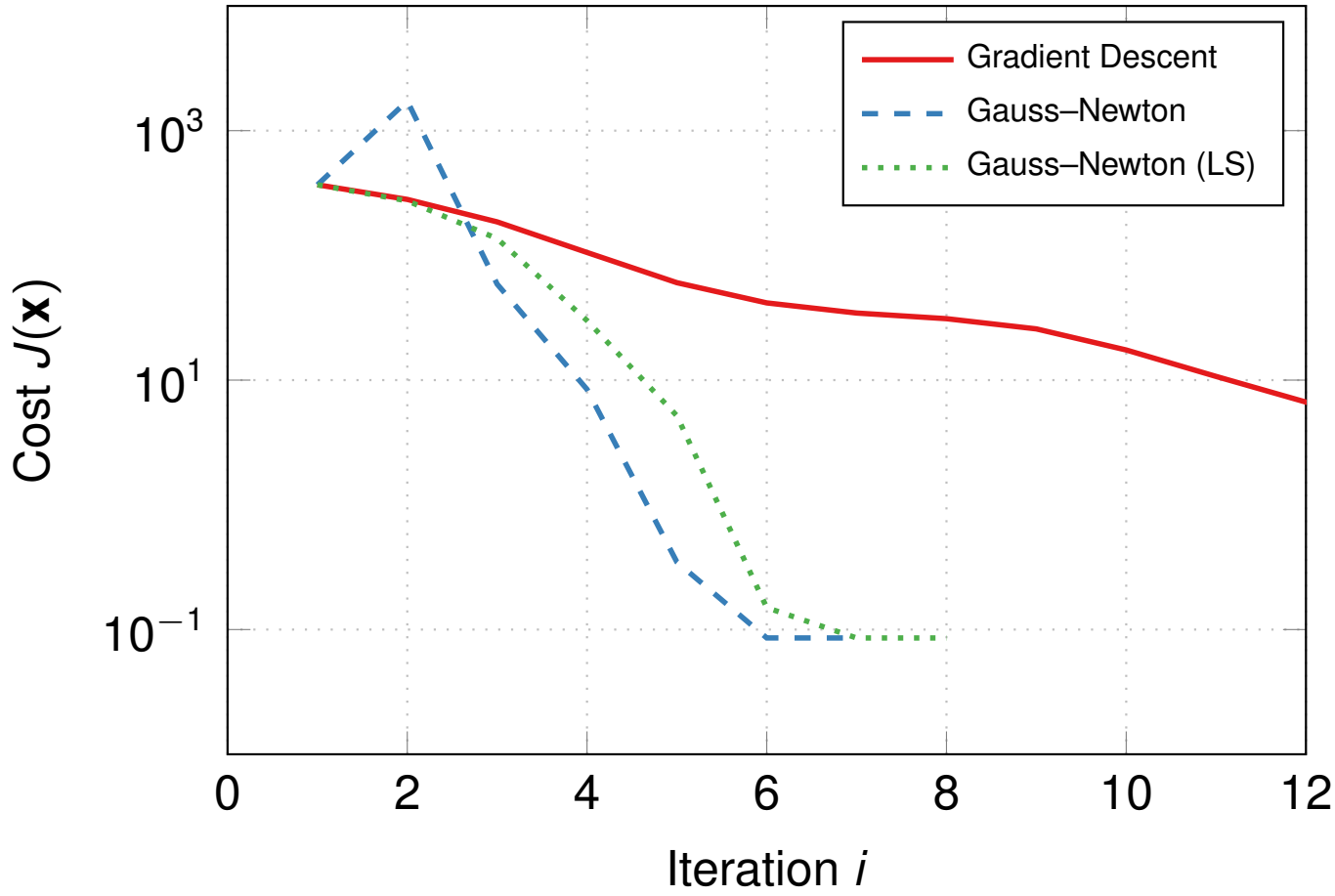
- We can find $\mathbf{x} = (p^x, p^y)$ by minimizing the cost function

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

Gauss–Newton Algorithm with Line Search: Example (2/3)



Gauss–Newton Algorithm with Line Search: Example (3/3)



Levenberg–Marquardt Algorithm: Motivation

- Gradient descent: Quickly moves to low cost area, creeps to minimum
- Gauss–Newton: Straight to minimum, may take a detour
- Can we have the best of both worlds?
- . . . kind of, the Levenberg–Marquardt algorithm.

Levenberg–Marquardt Algorithm: Derivation

- The Levenberg–Marquardt algorithm can be seen as a regularized version of the Gauss–Newton algorithm.
- Cost function approximation:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) \approx & \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right)^{\mathsf{T}} \\ & \times \mathbf{R}^{-1}\left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right) \\ & + \lambda(\mathbf{x} - \hat{\mathbf{x}}^{(i)})^{\mathsf{T}}(\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \end{aligned}$$

- We can now minimize this as a linear regularized problem:

$$\mathbf{x} = \hat{\mathbf{x}}^{(i)} + (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\mathbf{I})^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- Using this as the next iterate gives the iteration:

$$\begin{aligned} \hat{\mathbf{x}}^{(i+1)} &= \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)}, \\ \Delta\hat{\mathbf{x}}^{(i+1)} &= (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\mathbf{I})^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})). \end{aligned}$$

Levenberg–Marquardt Algorithm: Damping (1/2)

- The Levenberg–Marquardt update:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)},$$

$$\Delta\hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\mathbf{I})^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- How should the damping parameter λ be chosen?
- If $\lambda \to 0$:

$$\Delta\hat{\mathbf{x}}^{(i+1)} \to (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}})^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- If $\lambda \to \infty$:

$$\Delta\hat{\mathbf{x}}^{(i+1)} \to \frac{1}{\lambda}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- Ideally, in flat regions of the cost function where the linear approximation is good, λ should be chosen small, whereas in steep regions, it should be chosen large.

Levenberg–Marquardt Algorithm: Damping (2/2)

- A simple strategy is to start from some damping value $\lambda^{(0)}$ (e.g. $\lambda^{(0)} = 10^{-2}$) and select a fixed factor ν (e.g. $\nu = 10$).
- Then at each step we do the following:
  - First compute a candidate $\hat{\mathbf{x}}^{(i+1)}$ using the previous parameter value $\lambda^{(i-1)}$. Then proceed as follows:
    - If $J_{\text{WLS}}(\hat{\mathbf{x}}^{(i+1)}) < J_{\text{WLS}}(\hat{\mathbf{x}}^{(i)})$ then accept $\hat{\mathbf{x}}^{(i+1)}$ and decrease the damping parameter by $\lambda^{(i)} = \lambda^{(i-1)}/\nu$.
    - Otherwise continue with $\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)}$ and increase the damping parameter by $\lambda^{(i)} = \nu\,\lambda^{(i-1)}$.
- This idea appears already in the original article of Marquadt (1963).
- More sophisticated adaptation schemes can also be found in literature.

Levenberg–Marquardt Algorithm: Scaling

- The resulting algorithm is not scale invariant.
- To make it scale invariant we can normalize the regularized cost function approximation by using the diagonal values of $\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})$.
- This is equivalent to replacing the regularization term $\lambda\mathbf{I}$ with $\lambda\,\mathrm{diag}(\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}))$.
- The scaled iteration then becomes:

$$\begin{aligned}
\hat{\mathbf{x}}^{(i+1)} &= \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)}, \\
\Delta\hat{\mathbf{x}}^{(i+1)} &= (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\,\mathrm{diag}(\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})))^{-1} \\
&\qquad \times \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).
\end{aligned}$$

Levenberg–Marquardt Algorithm: Algorithm

Algorithm 4.6 Levenberg–Marquardt Algorithm with simple adaptation

Input: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data $\mathbf{y}$, function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}$, initial damping $\lambda^{(0)}$, and parameter ν.

Output: Parameter estimate $\hat{\mathbf{x}}_{\text{WLS}}$

1: Set $i \leftarrow 0$ and $\lambda \leftarrow \lambda^{(0)}$.
2: **repeat**
3: Compute the (candidate) parameter update:
4: **if** using scaled version **then**

$$\Delta\hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\,\mathrm{diag}(\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})))^{-1} \times \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

5: **else**

$$\Delta\hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda\mathbf{I})^{-1}\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

6: **end if**
7: **if** $J_{\text{WLS}}(\hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)}) < J_{\text{WLS}}(\hat{\mathbf{x}}^{(i)})$ **then**
8: Accept the candidate and decrease λ:

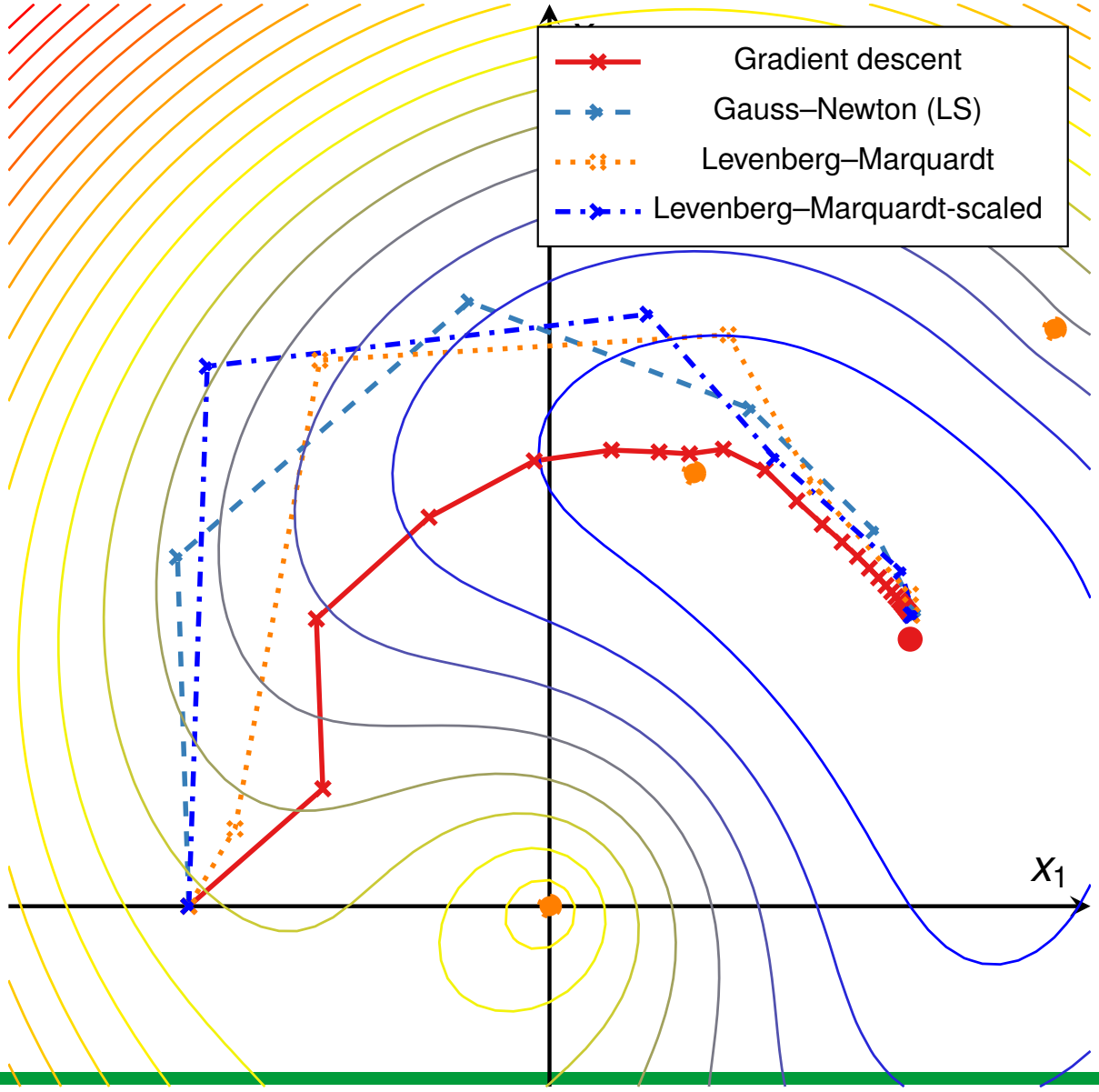
$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)}$$
$$\lambda \leftarrow \lambda/\nu$$

9: Set $i \leftarrow i + 1$
10: **else**
11: Reject the candidate and increase λ:

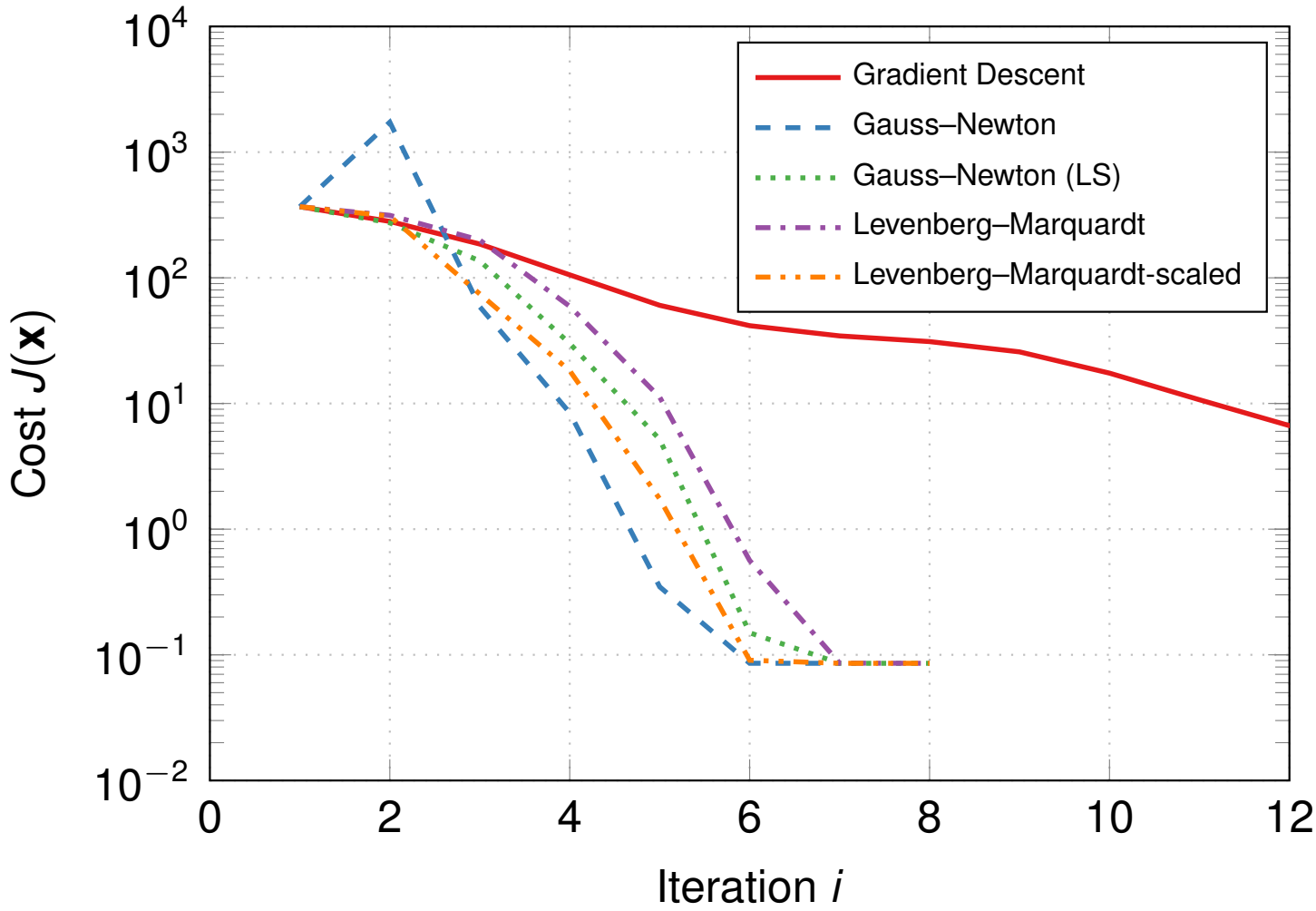
$$\lambda \leftarrow \nu\,\lambda$$

12: **end if**
13: **until** Converged
14: Set $\hat{\mathbf{x}}_{\text{WLS}} = \hat{\mathbf{x}}^{(i)}$

Levenberg–Marquardt Algorithm: Example



Decrease in Cost



Regularized Non-Linear Models

- The cost function that we considered:

$$J_{\mathrm{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- However, sometimes we are interested in regularized cost functions.
- Luckily, we can use the following simple trick:

$$\begin{aligned} J_{\mathrm{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{m} - \mathbf{x})^{\mathsf{T}} \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right)^{\mathsf{T}} \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right). \end{aligned}$$

- This is now a non-regularized cost function and hence all the algorithms presented are applicable.

Quasi-Newton Methods (1/2)

- Here we have only concentrated on least squares problems.
- There also exists a wider class of quasi-Newton methods.
- Let us consider a generic cost function $J(\mathbf{x})$ which we wish to minimize.
- Assume that our current guess for the minimum is $\mathbf{x}^{(i)}$ – we can now Taylor expand the cost function as follows:

$$J(\mathbf{x}) \approx J(\mathbf{x}^{(i)}) + \left[\frac{\partial J(\mathbf{x})}{\partial \mathbf{x}}\right]^{\mathsf{T}}\Bigg|_{\mathbf{x}=\mathbf{x}^{(i)}} (\mathbf{x} - \mathbf{x}^{(i)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(i)})^{\mathsf{T}} \left[\frac{\partial^2 J(\mathbf{x})}{\partial \mathbf{x}^2}\right]\Bigg|_{\mathbf{x}=\mathbf{x}^{(i)}} (\mathbf{x} - \mathbf{x}^{(i)}).$$

- We can now minimize the right hand side with respect to $\mathbf{x}$ and use the result as the next guess.

Quasi-Newton Methods (2/2)

- The resulting Newton's method takes the following form:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left[\frac{\partial^2 J(\mathbf{x})}{\partial \mathbf{x}^2}\right]^{-1} \left.\frac{\partial J(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^{(i)}}.$$

- However, computation of the Hessian is often not desirable.
- In so called quasi-Newton methods the Hessian is approximated in various ways.
- The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is a famous method for this.
- Gauss–Newton method can also be seen as a quasi–Newton method where we approximate the Hessian by $2\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}$.
- The line search procedure is typically an essential part of quasi-Newton methods.

Convergence Criteria

- When should we terminate iterations in optimization method?
- Various criteria and their combinations are used:
  - The absolute or relative change in the parameter estimate falls below a threshold, e.g.:

$$\|\Delta \mathbf{x}^{(i)}\| < \epsilon_p$$

.

  - The absolute or relative change in the cost falls below a certain threshold, e.g.:

$$\left| \frac{(J(\mathbf{x}^i) - J(\mathbf{x}^{(i+1)}))}{J(\mathbf{x}^i)} \right| < \epsilon_c$$

.

  - A maximum number of iterations is reached.

Summary (1/2)

- The Gauss–Newton update can be scaled with additional parameter γ:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}.$$

- The parameter can be found via line search that minimizes

$$J_{\text{WLS}}^{(i)}(\gamma) = J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}\right).$$

- We can also use inexact line search which ensures that the cost is decreased a sufficient amount.
- In Levenberg–Marquardt (LM) algorithm we replace the linear approximation in Gauss–Newton with its regularized version.
- In LM algorithm, we find a suitable regularization parameter λ via an iterative procedure.

Summary (2/2)

- We can also consider regularized nonlinear problems with a simple trick:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{m} - \mathbf{x})^{\mathsf{T}} \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right)^{\mathsf{T}} \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right). \end{aligned}$$

- Quasi-Newton methods are more general optimization methods that approximate the Hessian in Newton's method.
- Various convergence criteria are available for terminating iterative optimization methods.