Aalto University
School of Electrical Engineering

ELEC-E8740 — Static Nonlinear Models, Gradient Descent, and Gauss–Newton

Simo Särkkä

Aalto University

October 2, 2020

Contents

Intended Learning Outcomes

After this lecture, you will be able to:

- Identify the need for non-linear models;
- understand the principles of gradient decent and Gauss–Newton methods;
- apply gradient decent and Gauss–Newton methods to nonlinear sensor fusion problems.

Recap (1)

- The general linear model is given by

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}, \; \mathrm{E}\{\mathbf{r}\} = 0, \; \mathrm{Cov}\{\mathbf{r}\} = \mathbf{R}$$

- Affine models can be tackled by rewriting

$$\begin{aligned} \mathbf{y} &= \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}, \\ \underbrace{\mathbf{y} - \mathbf{b}}_{\tilde{\mathbf{y}}} &= \mathbf{G}\mathbf{x} + \mathbf{r}. \end{aligned}$$

- Different least squares estimators:

$$\begin{aligned} \hat{\mathbf{x}}_{\mathrm{LS}} &= (\mathbf{G}^\mathsf{T}\mathbf{G})^{-1}\mathbf{G}^\mathsf{T}\mathbf{y}, \\ \hat{\mathbf{x}}_{WLS} &= (\mathbf{G}^\mathsf{T}\mathbf{R}^{-1}\mathbf{G})^{-1}\mathbf{G}^\mathsf{T}\mathbf{R}^{-1}\mathbf{y}, \\ \hat{\mathbf{x}}_{\mathrm{ReLS}} &= (\mathbf{G}^\mathsf{T}\mathbf{R}^{-1}\mathbf{G} + \mathbf{P}^{-1})^{-1}(\mathbf{G}^\mathsf{T}\mathbf{R}^{-1}\mathbf{y} + \mathbf{P}^{-1}\mathbf{m}). \end{aligned}$$

- We also computed their expectations and covariances.

Recap (2)

- Alternative form of regularized least squares estimator:

$$
\begin{aligned}
\mathbf{K} &= \mathbf{P}\mathbf{G}^\mathsf{T}(\mathbf{G}\mathbf{P}\mathbf{G}^\mathsf{T} + \mathbf{R})^{-1}, \\
\hat{\mathbf{x}}_{\text{ReLS}} &= \mathbf{m} + \mathbf{K}(\mathbf{y} - \mathbf{G}\mathbf{m}), \\
\text{Cov}\{\hat{\mathbf{x}}_{\text{ReLS}}\} &= \mathbf{P} - \mathbf{K}(\mathbf{G}\mathbf{P}\mathbf{G}^\mathsf{T} + \mathbf{R})\mathbf{K}^\mathsf{T}.
\end{aligned}
$$

- Sequential (weighted/regularized) least squares estimator:

$$
\begin{aligned}
\mathbf{K}_n &= \mathbf{P}_{n-1}\mathbf{G}_n^\mathsf{T}(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^\mathsf{T} + \mathbf{R}_n)^{-1}, \\
\hat{\mathbf{x}}_n &= \hat{\mathbf{x}}_{n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{G}_n\hat{\mathbf{x}}_{n-1}), \\
\mathbf{P}_n &= \mathbf{P}_{n-1} - \mathbf{K}_n(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^\mathsf{T} + \mathbf{R}_n)\mathbf{K}_n^\mathsf{T}.
\end{aligned}
$$

Static Nonlinear Models

- Linear models have closed form solutions, but are limited in many cases
- General nonlinear model has the form:

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

- General cost function that we consider:

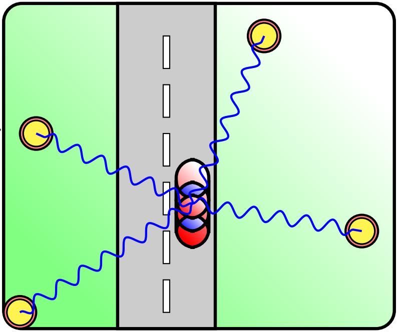
$$J_{\mathrm{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- For some models, closed form solutions do exist – but for most they do not.
- Regularized cost functions can be handled with an augmentation trick – we will come back to that.

Nonlinear Model of an Autonomous Car

- 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

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

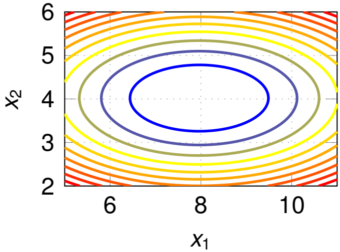
Numerical Optimization

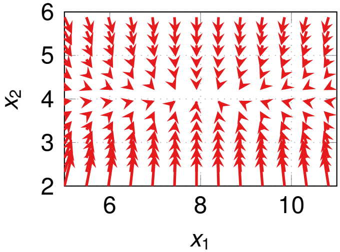
- Iterative algorithms can be used to find the minima of a cost function.
- Generally find local minima $\Rightarrow$ require good initialization.
- Today, we will look at two approaches:
  1. Gradient descent method
  2. the Gauss–Newton algorithm

Gradient Descent: Formulation

- The gradient of $J(\mathbf{x})$ w.r.t. $\mathbf{x}$ points to the direction where $J(\mathbf{x})$ increases as a function of $\mathbf{x}$
- Changing $\mathbf{x}$ in the opposite direction of the gradient decreases $J(\mathbf{x})$
- If the function to minimize is $J_{\text{WLS}}(\mathbf{x})$, the cost is decreased by the iteration

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} - \gamma \left. \frac{\partial J_{\text{WLS}}(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\hat{\mathbf{x}}^{(i)}},$$





Gradient Descent: Derivation (1/3)

- Let us start by considering scalar LS cost function

$$J_{\mathrm{LS}}(\mathbf{x}) = \sum_{n=1}^{N} (y_n - g_n(\mathbf{x}))^2$$

- The gradient is

$$\begin{aligned}\frac{\partial J_{\mathrm{LS}}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} \sum_{n=1}^{N} (y_n - g_n(\mathbf{x}))^2 \\ &= \sum_{n=1}^{N} -2(y_n - g_n(\mathbf{x})) \frac{\partial g_n(\mathbf{x})}{\partial \mathbf{x}}.\end{aligned}$$

Gradient Descent: Derivation (2/3)

- Vector form:

$$
\begin{aligned}
\frac{\partial J_{\mathrm{LS}}(\mathbf{x})}{\partial \mathbf{x}} &= \sum_{n=1}^{N} -2(y_n - g_n(\mathbf{x}))\frac{\partial g_n(\mathbf{x})}{\partial \mathbf{x}} \\
&= -2 \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial \mathbf{x}} & \frac{\partial g_2(\mathbf{x})}{\partial \mathbf{x}} & \cdots & \frac{\partial g_N(\mathbf{x})}{\partial \mathbf{x}} \end{bmatrix} \left(\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_N(\mathbf{x}) \end{bmatrix} \right) \\
&= -2 \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_1} & \frac{\partial g_2(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_N(\mathbf{x})}{\partial x_1} \\ \frac{\partial g_1(\mathbf{x})}{\partial x_2} & \frac{\partial g_2(\mathbf{x})}{\partial x_2} & & \vdots \\ \vdots & & \ddots & \frac{\partial g_N(\mathbf{x})}{\partial x_{K-1}} \\ \frac{\partial g_1(\mathbf{x})}{\partial x_K} & \cdots & \frac{\partial g_{N-1}(\mathbf{x})}{\partial x_K} & \frac{\partial g_N(\mathbf{x})}{\partial x_K} \end{bmatrix} (\mathbf{y} - \mathbf{g}(\mathbf{x})) \\
&= -2\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\mathbf{x})\,(\mathbf{y} - \mathbf{g}(\mathbf{x})).
\end{aligned}
$$

- $\mathbf{G}_{\mathbf{x}}$ is the Jacobian matrix of $\mathbf{g}(\mathbf{x})$

Gradient Descent: Derivation (3/3)

- Generalization to WLS cost function:

$$\begin{aligned}\frac{\partial J_{\mathrm{WLS}}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} (\mathbf{y} - \mathbf{g}(\mathbf{x}))^{\mathsf{T}} \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) \\ &= -2 \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\mathbf{x})\, \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).\end{aligned}$$

- The direction of negative gradient is $\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\mathbf{x})\, \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x}))$.
- The parameter update becomes:

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

- We have absorbed the factor 2 into the constant γ.

Gradient Descent: Step Size

- How long the step length γ should be chosen?
  - Choosing γ too large may cause the cost function to increase.
  - Too small steps might cause unnecessarily slow convergence.
- A typical strategy is to simply choose it small enough so that the cost decreases at every step.
- Advisable to change the step length during the iterations in one way or another.
- One way is to use a line search – but we come back to that next time.

Gradient Descent: Algorithm

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

Example: Localizing a Car (1)

- We have

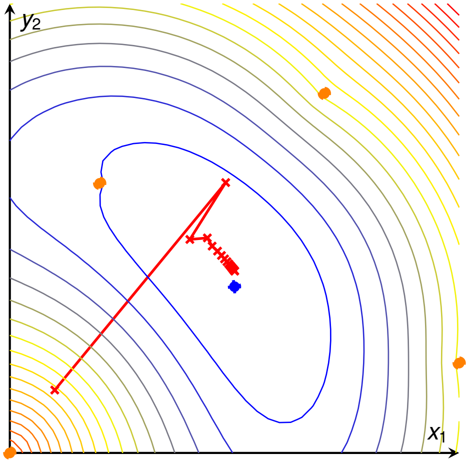
$$y_n^R = \underbrace{\sqrt{(s_n^x - p^x)^2 + (s_n^y - p^y)^2}}_{g_n(\mathbf{x})} + r_n^R, \quad \mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_M(\mathbf{x}) \end{bmatrix}$$

- The Jacobian is

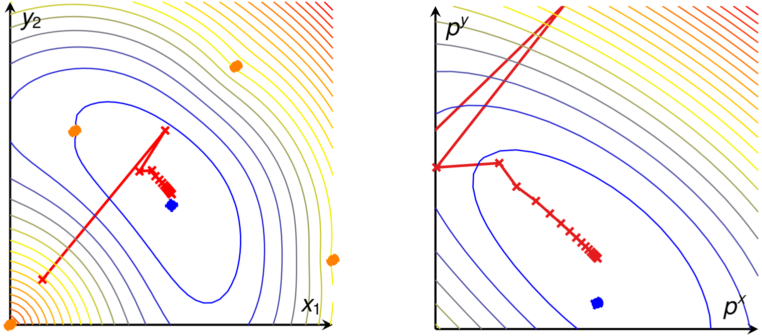
$$\mathbf{G_x}(\mathbf{x}) = \begin{bmatrix} \frac{-(s_1^x - p^x)}{\sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2}} & \frac{-(s_1^y - p^y)}{\sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2}} \\ \vdots & \vdots \\ \frac{-(s_M^x - p^x)}{\sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2}} & \frac{-(s_M^y - p^y)}{\sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2}} \end{bmatrix}$$

- *Tip:* always, always check the Jacobian numerically!

Example: Localizing a Car (2)



Example: Localizing a Car (3)



Gauss–Newton Algorithm: Derivation (1/2)

- Idea: Given $\hat{\mathbf{x}}^{(i)}$, we can linearize the nonlinear measurement model around that point
- Linearized measurement model:

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\hat{\mathbf{x}}^{(i)}) + \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{x} - \hat{\mathbf{x}}^{(i)})$$

- Cost function approximation:

$$\begin{aligned} J_{\mathrm{WLS}}(\mathbf{x}) &\approx \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right)^{\mathsf{T}} \mathbf{R}^{-1} \\ &\quad \times \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right) \\ &= \left(\mathbf{e}^{(i)} - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right)^{\mathsf{T}} \mathbf{R}^{-1} \\ &\quad \times \left(\mathbf{e}^{(i)} - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})\,(\mathbf{x} - \hat{\mathbf{x}}^{(i)})\right) \end{aligned}$$

- This can now be minimized w.r.t. $\mathbf{x}$ in the same way as linear models.

Gauss–Newton Algorithm: Derivation (2/2)

- Gradient of the cost function approximation w.r.t. $\mathbf{x}$:

$$
\begin{aligned}
\frac{\partial J_{\mathrm{WLS}}(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{\partial}{\partial \mathbf{x}} \Big(& (\mathbf{e}^{(i)})^{\mathsf{T}} \mathbf{R}^{-1} \mathbf{e}^{(i)} - (\mathbf{e}^{(i)})^{\mathsf{T}} \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) \, (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \\
& - (\mathbf{x} - \hat{\mathbf{x}}^{(i)})^{\mathsf{T}} \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)}) \, \mathbf{R}^{-1} \mathbf{e}^{(i)} \\
& + (\mathbf{x} - \hat{\mathbf{x}}^{(i)})^{\mathsf{T}} \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)}) \, \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) \, (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \Big) \\
= -2 & \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)}) \, \mathbf{R}^{-1} \mathbf{e}^{(i)} + 2 \mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}^{(i)}) \, \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) \, (\mathbf{x} - \hat{\mathbf{x}}^{(i)})
\end{aligned}
$$

- Setting to zero and solving for $\mathbf{x}$ gives:

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

- The Gauss–Newton method we used the above solution $\mathbf{x}$ as the next iterate $\hat{\mathbf{x}}^{(i+1)}$.

Gauss–Newton Algorithm: Algorithm

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 $\hat{\mathbf{x}}_{\mathrm{WLS}}$

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 $\hat{\mathbf{x}}_{\mathrm{WLS}} = \hat{\mathbf{x}}^{(i)}$

Gauss–Newton Algorithm: Covariance of the Estimate

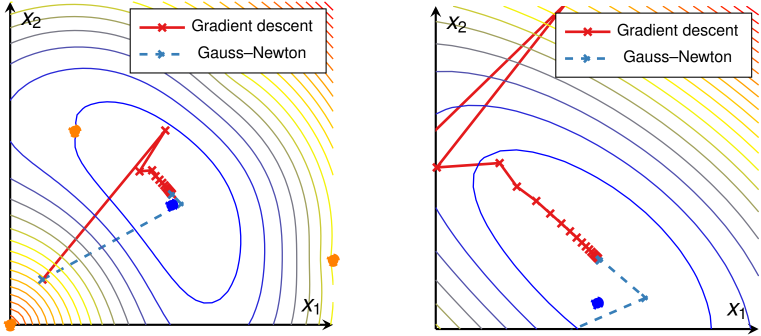
- The covariance of the estimate is hard to compute in the non-linear case.
- However, we can use the linearization approximation:

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\hat{\mathbf{x}}_{\mathrm{WLS}}) + \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}_{\mathrm{WLS}})\,(\mathbf{x} - \hat{\mathbf{x}}_{\mathrm{WLS}})$$

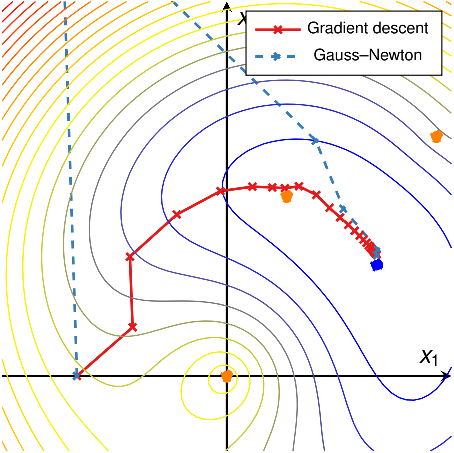
- The covariance for this leariar model can be used as approximation for the covariance:

$$\mathrm{Cov}\{\hat{\mathbf{x}}_{\mathrm{WLS}}\} \approx (\mathbf{G}_{\mathbf{x}}^{\mathsf{T}}(\hat{\mathbf{x}}_{\mathrm{WLS}})\,\mathbf{R}^{-1}\mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}_{\mathrm{WLS}}))^{-1}.$$

Example: Localizing a Car (4)



Example: Localizing a Car (5)



Gauss–Newton Algorithm: Challenges

- When the initial point is far away, the convergence can fail.
  - ✓ We can try several starting points or select them cleverly.
- The full step using the linearized model might go too far:
  - ✓ Line search can be used to select a good step length.
  - ✓ We could use regularized solution for the linearized model.
- The latter two methods will be presented next time.

Summary

- 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:

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

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