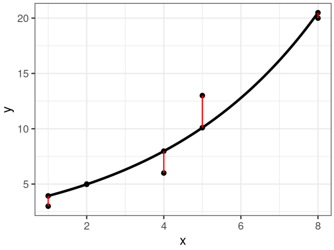
Optimization in Machine Learning

Second order methods
Gauss-Newton



Learning goals

- Least squares
- Gauss-Newton
- Levenberg-Marquardt

LEAST SQUARES PRO BLEM

Consider the problem of minimizing a sum of squares

$$\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}),$$

where

$$g(\boldsymbol{\theta}) = r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta}) = \sum_{i=1}^{n} r_i(\boldsymbol{\theta})^2$$

and

$$\begin{aligned} r : \mathbb{R}^d &\to \mathbb{R}^n \\ \boldsymbol{\theta} &\mapsto (r_1(\boldsymbol{\theta}), \dots, r_n(\boldsymbol{\theta}))^\top \end{aligned}$$

maps parameters $\boldsymbol{\theta}$ to residuals $r(\boldsymbol{\theta})$

NEWTON-RAPHSON IDEA

Approach: Calculate Newton-Raphson update direction by solving:

$$\nabla^2 g(\boldsymbol{\theta}^{[t]})\mathbf{d}^{[t]} = -\nabla g(\boldsymbol{\theta}^{[t]}).$$

Gradient is calculated via chain rule

$$\nabla g(\boldsymbol{\theta}) = \nabla(r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta})) = 2 \cdot J_r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta}),$$

where $J_r(\boldsymbol{\theta})$ is Jacobian of $r(\boldsymbol{\theta})$.

In our example:

$$J_r(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial r_1(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_1(\boldsymbol{\theta})}{\partial \theta_2} \\ \frac{\partial r_2(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_2(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots & \vdots \\ \frac{\partial r_5(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_5(\boldsymbol{\theta})}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} \exp(\theta_2 x^{(1)}) & x^{(1)}\theta_1 \exp(\theta_2 x^{(1)}) \\ \exp(\theta_2 x^{(2)}) & x^{(2)}\theta_1 \exp(\theta_2 x^{(2)}) \\ \exp(\theta_2 x^{(3)}) & x^{(3)}\theta_1 \exp(\theta_2 x^{(3)}) \\ \exp(\theta_2 x^{(4)}) & x^{(4)}\theta_1 \exp(\theta_2 x^{(4)}) \\ \exp(\theta_2 x^{(5)}) & x^{(5)}\theta_1 \exp(\theta_2 x^{(5)}) \end{pmatrix}$$

GAUSS-NEWTON FOR LEAST SQUARES

Gauss-Newton approximates $\mathbf{H}_g$ by dropping its second order part:

$$
\begin{aligned}
H_{jk} &= 2\sum_{i=1}^{n}\left(\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k} + r_i\frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}\right) \\
&\approx 2\sum_{i=1}^{n}\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k} \\
&= 2J_r(\boldsymbol{\theta})^\top J_r(\boldsymbol{\theta})
\end{aligned}
$$

Note: We assume that

$$
\left|\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k}\right| \gg \left|r_i\frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}\right|.
$$

This assumption may be valid if:

- Residuals r_i are small in magnitude <u>**or**</u>
- Functions are only “mildly” nonlinear s.t. $\frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}$ is small.

LEVENBERG-MARQUARDT ALGORITHM

- **Problem:** Gauss-Newton may not decrease g in every iteration but may diverge, especially if starting point is far from minimum
- **Solution:** Choose step size $\alpha > 0$ s.t.

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \alpha \mathbf{d}^{[t]}$$

decreases g (e.g., by satisfying Wolfe conditions)

- However, if α gets too small, an **alternative** method is the

Levenberg-Marquardt algorithm

$$(J_r^\top J_r + \lambda D)\mathbf{d}^{[t]} = -J_r^\top r(\boldsymbol{\theta})$$

- D is a positive diagonal matrix
- $\lambda = \lambda^{[t]} > 0$ is the *Marquardt parameter* and chosen at each step