Differential programming

Sergey Barseghyan barseghyan@phystech.edu

16.08.2022

Outline

Levenberg-Marquad algorithm - is an optimization algorithm which is basically combination of two objective minimization algorithms: **Gauss-Newton** and **Gradient Descent** It is very very well suited for the optimization problemsswhere model is **non-linear** in its parameters

Non-linear least squares

$$
\begin{aligned}
\hat{y}(t; \boldsymbol{p}) & \quad \text{- fitted model} \\
t \in \mathbb{R} & \quad \text{- independent variable} \\
\{t_i, y_i\}_{i=1}^{m} & \quad \text{- observed data points} \\
W_{ij} & \quad \text{- inverse covariance matrix} \\
\boldsymbol{p} \in \mathbb{R}^n & \quad \text{model parameters}
\end{aligned}
$$

Optimized objective is nothing else but chi-squared statistic

$$
\begin{aligned}
\chi^2(\boldsymbol{p}) &= \sum_{i=1}^{m} \left[\frac{y(t_i) - \hat{y}(t_i; \boldsymbol{p})}{\sigma_{y_i}} \right]^2 \\
&= (\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))^\top \boldsymbol{W} (\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))
\end{aligned}
$$

Gradient Descent and Jacobian update

Gradient Descent step update

$$\frac{\partial}{\partial \boldsymbol{p}}\chi^2 = -2(\boldsymbol{y} - \hat{\boldsymbol{y}})^\top \boldsymbol{W}\boldsymbol{J}$$

$$\boldsymbol{h}_{\mathrm{gd}} = \alpha \boldsymbol{J}^\top \boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$

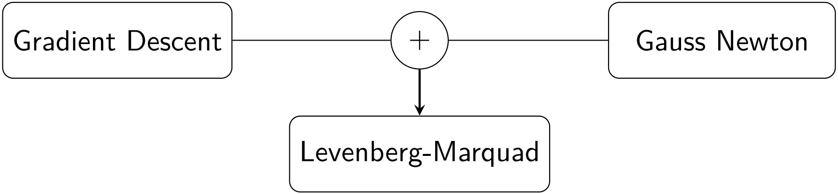
Gauss-Newton step update

$$\hat{\boldsymbol{y}}(\boldsymbol{p}+\boldsymbol{h}) \approx \hat{\boldsymbol{y}}(\boldsymbol{p}) + \left[\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{p}}\right]\boldsymbol{h} = \hat{\boldsymbol{y}} + \boldsymbol{J}\boldsymbol{h}$$

$$\frac{\partial}{\partial \boldsymbol{h}}\chi^2(\boldsymbol{p}+\boldsymbol{h}) \approx -2(\boldsymbol{y} - \hat{\boldsymbol{y}})^\top \boldsymbol{W}\boldsymbol{J} + 2\boldsymbol{h}^\top \boldsymbol{J}^\top \boldsymbol{W}\boldsymbol{J}$$

$$\left[\boldsymbol{J}^\top \boldsymbol{W}\boldsymbol{J}\right]\boldsymbol{h}_{\mathrm{gn}} = \boldsymbol{J}^\top \boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$

Levenberg-Marquad update



$$\left[\boldsymbol{J}^\top \boldsymbol{W}\boldsymbol{J} + \lambda\,\mathrm{diag}\left(\boldsymbol{J}^\top \boldsymbol{W}\boldsymbol{J}\right)\right]\boldsymbol{h}_{\mathrm{lm}} = \boldsymbol{J}^\top \boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$

λ is dumping parameter

Algorithm 1 LM algorithm parameter update

$\lambda_0 = \lambda_o$
Calculate $\boldsymbol{h_{lm}}$ and ρ
if $\rho_i(\boldsymbol{h_{lm}}) > \epsilon_4$ **then**
    $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h_{lm}}$;
    $\lambda_{i+1} = \max\left[\lambda_i / L_{down}, 10^{-7}\right]$
else
    $\lambda_{i+1} = \min\left[\lambda_i L_{up}, 10^{7}\right]$
end if

Where

$$\rho_i(\boldsymbol{h_{lm}}) = \frac{\chi^2(\boldsymbol{p}) - \chi^2\left(\boldsymbol{p} + \boldsymbol{h}_{\mathrm{lm}}\right)}{\boldsymbol{h}_{\mathrm{lm}}^{\top}\left(\lambda_i \operatorname{diag}\left(\boldsymbol{J}^{\top}\boldsymbol{W}\boldsymbol{J}\right)\boldsymbol{h}_{\mathrm{lm}} + \boldsymbol{J}^{\top}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)}$$

Jacobian update

For every $2n$ interation where $\chi^2(\boldsymbol{p}+\boldsymbol{h}) > \chi^2(\boldsymbol{p})$ forward or central difference is calculated for Jacobian

$$J_{ij} = \frac{\partial \hat{y}_i}{\partial p_j} = \frac{\hat{y}\left(t_i; \boldsymbol{p} + \delta \boldsymbol{p}_j\right) - \hat{y}\left(t_i; \boldsymbol{p}\right)}{\left\|\delta \boldsymbol{p}_j\right\|}$$

$$J_{ij} = \frac{\partial \hat{y}_i}{\partial p_j} = \frac{\hat{y}\left(t_i; \boldsymbol{p} + \delta \boldsymbol{p}_j\right) - \hat{y}\left(t_i; \boldsymbol{p} - \delta \boldsymbol{p}_j\right)}{2\left\|\delta \boldsymbol{p}_j\right\|}$$

For intermediate iterations Broyden's rank-1 update formula is used in order not to make expensive calculations on every interation.

$$\boldsymbol{J}^{i+1} = \boldsymbol{J}^{i} + \frac{(\hat{\boldsymbol{y}}(\boldsymbol{p}+\boldsymbol{h}) - \hat{\boldsymbol{y}}(\boldsymbol{p}) - \boldsymbol{J}^{i}\boldsymbol{h})\boldsymbol{h}^{\top}}{\boldsymbol{h}^{\top}\boldsymbol{h}}$$

```python
@torch.no_grad()
def broyden_jacobian_update(self):
    '''
    Broyden 1-rank Jacobian update
    '''
    df = self.func(self.p + self.dp) - self.func(self.p)
    self.J += torch.outer(df - torch.mv(self.J, self.dp),
                          self.dp) \
                      .div(torch.linalg.norm(self.dp, ord=2))
@torch.no_grad()
def torch_jacobian_update(self, p):
    '''
    Finite-difference Jacobian update
    '''
    self.J = torch.autograd.functional.jacobian(self.func, p)

```

```python
@torch.no_grad()
def solve_for_dp(self):
    '''
    Solver for optimizer step
    '''
    self.JTW = torch.matmul(torch.transpose(self.J, 0, 1),
                            self.W)
    self.JTWJ =  torch.matmul(self.JTW, self.J)

    dy =  self.y_data - self.func(self.p)
    self.dp = torch.linalg.solve(self.JTWJ
                            + self.lambda_lm
                            * torch.diag(
                                torch.diagonal(self.JTWJ)
                                    ),
                            torch.mv(self.JTW , dy)
                            )
```

Motivation for the model

Model Definition

Exact solution