Computional Homework6 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Introduction

This report presents a comparison of two nonlinear least-squares fitting algorithms for source localization: Gauss-Newton (GN) and Levenberg-Marquardt (LM). The problem involves determining the source location (b_0, b_1) using 6 receivers with measurement (t_i, x_i, y_i) where noise is included. The mathematical model is:

$$t = \frac{\sqrt{(x - b_0)^2 + (y - b_1)^2}}{v}$$

where $v = 1500$ m/s is the propagation velocity.

The optimization problem is formulated as minimizing the sum of squared residuals:

$$\min_{b_0, b_1} \sum_{i=1}^{6} \left(t_i^{pred} - t_i^{obs} \right)^2$$

where $t_i^{pred} = \frac{\sqrt{(x_i - b_0)^2 + (y_i - b_1)^2}}{v}$ and t_i^{obs} are the observed arrival times.

1.1 Experimental Configuration

We employ a circular distribution of 6 receivers positioned around the true source location at $(0, 0)$. The receivers are placed at a radius of 9000 meters with angular spacing of $\pi/3$ radians (60 degrees). Figure 1 illustrates the geometric configuration of the receiver network and source location.

The measurement data includes:

- **True travel times:** $[6.0, 6.0, 6.0, 6.0, 6.0, 6.0]$ seconds
- **Noisy observations:** $[6.050, 5.986, 6.065, 6.152, 5.977, 5.977]$ seconds
- **Noise level:** Gaussian noise with $\sigma = 0.1$ seconds

This configuration provides a well-conditioned geometry for source localization, with receivers distributed uniformly around the perimeter to ensure good triangulation capabilities. The relatively high noise level ($\sigma = 0.1$ s) represents a challenging scenario that tests the robustness of both optimization algorithms.

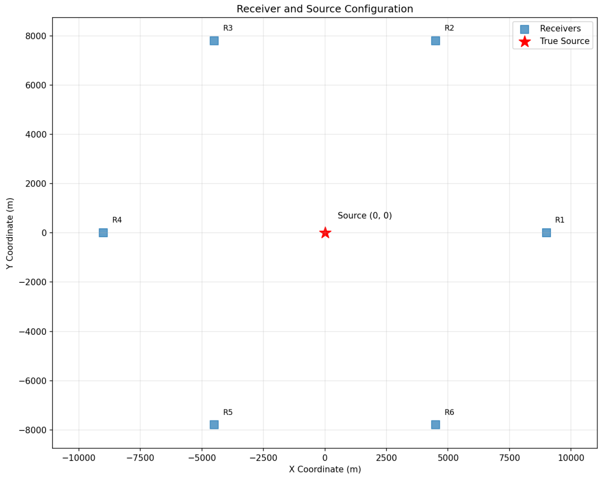

Figure 1 **Receiver and source configuration showing 6 receivers (R1-R6) positioned in a circular pattern with 9000m radius around the true source location at the origin. The uniform angular distribution ensures optimal geometric dilution of precision for localization accuracy.**

```
v = 1500

x0, y0 = 0, 0

def tt(x, y, x0=x0, y0=y0):
    return np.sqrt((x - x0) ** 2 + (y - y0) ** 2) / v

idx = np.linspace(0, 5, 6)
xn = 9000 * np.cos(np.pi / 3 * idx)
yn = 9000 * np.sin(np.pi / 3 * idx)
tn = tt(xn, yn)
t_noise = np.random.normal(0, 0.1, size=idx.shape)
xn = xn
yn = yn
tn = tn + t_noise
```

2 Nonlinear Least Squares Algorithms

2.1 Gauss-Newton Method

The Gauss-Newton method is an iterative algorithm for solving nonlinear least squares problems. We first discuss the Gauss-Newton Algorithm for nonlinear least squares.

Theoretical Derivation:

For the nonlinear least squares problem, we want to minimize the objective function:

$$S(\mathbf{b}) = \frac{1}{2}\|\mathbf{r}(\mathbf{b})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{b})^2$$

where $\mathbf{r}(\mathbf{b}) = [r_1(\mathbf{b}), r_2(\mathbf{b}), \ldots, r_m(\mathbf{b})]^T$ is the residual vector.

Let

$$F(\mathbf{b}) \equiv J_r(\mathbf{b})^T \mathbf{r}(\mathbf{b}) = \nabla S(\mathbf{b})$$

Here, $F(\mathbf{b})$ represents the **gradient of the objective function** $S(\mathbf{b})$.

The fundamental principle of optimization is to find where the gradient equals zero:

$$\nabla S(\mathbf{b}) = 0 \quad \Rightarrow \quad F(\mathbf{b}) = 0$$

Since we want to find $\mathbf{b}^*$ such that $F(\mathbf{b}^*) = 0$, and our current estimate is $\mathbf{b}_k$, we seek an update $\Delta\mathbf{b}$ such that:

$$F(\mathbf{b}_k + \Delta\mathbf{b}) = 0$$

Local Optimization Through Linearization:

Since $F(\mathbf{b})$ is generally nonlinear, we cannot solve $F(\mathbf{b}) = 0$ directly. Instead, we use **local linearization**:

Let

$$\mathbf{b} = \mathbf{b}_k + \Delta\mathbf{b}$$

The Taylor expansion of $F(\mathbf{b})$ with first-order terms gives:

$$F(\mathbf{b}_k + \Delta\mathbf{b}) \approx F(\mathbf{b}_k) + F'(\mathbf{b}_k)\Delta\mathbf{b}$$

Setting $F(\mathbf{b}_k + \Delta\mathbf{b}) = 0$ in the linearized system:

$$F(\mathbf{b}_k) + F'(\mathbf{b}_k)\Delta\mathbf{b} = 0$$

Therefore:

$$\Delta\mathbf{b} = -[F'(\mathbf{b}_k)]^{-1} F(\mathbf{b}_k)$$

Computing the Derivative F'(b):

Since $F(\mathbf{b}) = J_r(\mathbf{b})^T\mathbf{r}(\mathbf{b})$, we have:

$$F'(\mathbf{b}) = \frac{d}{d\mathbf{b}}[J_r(\mathbf{b})^T\mathbf{r}(\mathbf{b})] = J_r(\mathbf{b})^T J_r(\mathbf{b}) + \sum_{i=1}^{m} r_i(\mathbf{b})\nabla^2 r_i(\mathbf{b})$$

The Gauss-Newton approximation neglects the second-order terms (assuming residuals are small at the optimum):

$$F'(\mathbf{b}) \approx J_r(\mathbf{b})^T J_r(\mathbf{b})$$

This leads to the Gauss-Newton update rule:

$$\Delta\mathbf{b} = -[J_r(\mathbf{b}_k)^T J_r(\mathbf{b}_k)]^{-1} J_r(\mathbf{b}_k)^T\mathbf{r}(\mathbf{b}_k)$$

For numerical stability, we add regularization:

$$\Delta\mathbf{b} = -[J_r(\mathbf{b}_k)^T J_r(\mathbf{b}_k) + \epsilon I]^{-1} J_r(\mathbf{b}_k)^T\mathbf{r}(\mathbf{b}_k)$$

where $\epsilon = 10^{-8}$ ensures the matrix is invertible.

2.1.1 Jacobian Matrix

For our source localization problem, the residual function is:

$$r_i(\mathbf{b}) = \frac{\sqrt{(x_i - b_0)^2 + (y_i - b_1)^2}}{v} - t_i$$

The Jacobian matrix $J_r(\mathbf{b})$ contains the partial derivatives:

$$\frac{\partial r_i}{\partial b_0} = -\frac{x_i - b_0}{v\sqrt{(x_i - b_0)^2 + (y_i - b_1)^2}}$$

$$\frac{\partial r_i}{\partial b_1} = -\frac{y_i - b_1}{v\sqrt{(x_i - b_0)^2 + (y_i - b_1)^2}}$$

```
def _jacobian(self, b):
b0, b1 = b
dx = self.xn - b0
dy = self.yn - b1
r = np.sqrt(dx**2 + dy**2)
```

```
    J = np.zeros((len(self.xn), 2))
    J[:, 0] = -dx / (self.v * r)
    J[:, 1] = -dy / (self.v * r)
    return J
```

2.1.2 Residual Function

The residual function computes the difference between predicted and observed arrival times:

$$r_i = t_i^{pred} - t_i^{obs} = \frac{\sqrt{(x_i - b_0)^2 + (y_i - b_1)^2}}{v} - t_i$$

```
    def _f(self, b):
    b0, b1 = b
    return np.sqrt((self.xn - b0) **2 + (self.yn - b1)** 2) / self.v
def _residual(self, b):
    obs = self.tn
    pred = self._f(b)
    return pred - obs
```

2.2 Levenberg-Marquardt Method

The Levenberg-Marquardt (LM) algorithm extends the Gauss-Newton method by introducing an adaptive damping parameter to handle ill-conditioned problems and poor initial guesses.

Motivation for LM:

The Gauss-Newton method can fail when:

- $J_r^T J_r$ is singular or ill-conditioned
- The initial guess is far from the optimum
- The residuals are large (violating the small residual assumption)

LM Strategy:

Starting from the same linearization approach:

$$F(\mathbf{b}_k) + F'(\mathbf{b}_k)\Delta\mathbf{b} = 0$$

Instead of using $F'(\mathbf{b}_k) = J_r(\mathbf{b}_k)^T J_r(\mathbf{b}_k)$, LM uses:

$$F'(\mathbf{b}_k) = J_r(\mathbf{b}_k)^T J_r(\mathbf{b}_k) + \mu I$$

This gives the LM update rule:

$$\Delta \mathbf{b} = -[J_r(\mathbf{b}_k)^T J_r(\mathbf{b}_k) + \mu I]^{-1} J_r(\mathbf{b}_k)^T \mathbf{r}(\mathbf{b}_k)$$

Adaptive Damping Strategy:

- When $\mu \to 0$: approaches Gauss-Newton (fast local convergence)
- When μ is large: approaches steepest descent (robust global behavior)

The parameter μ is adjusted based on the success of each iteration:

$$\text{If loss decreases:} \quad \text{accept step,} \quad \mu \leftarrow \max(\mu/10, 10^{-7}) \tag{1}$$

$$\text{If loss increases:} \quad \text{reject step,} \quad \mu \leftarrow \min(\mu \times 10, 10^{4}) \tag{2}$$

2.2.1 Gauss-Newton Implementation

```python
def fit(self, b_init, method="GN", epochs=100, tol=1e-6, miuk=1e-4):
    b = np.array(b_init, dtype=float)
    p_s = []
    p_s.append(b)
    self.method = method
    losses = []
    r = self._residual(b)
    loss = np.sum(r**2)
    losses.append(loss)
    for ii in range(epochs):
        r = self._residual(b)
        J = self._jacobian(b)
        if method == "GN":
            eps = 1e-8
            delta_b = -np.linalg.solve(J.T @ J + eps * np.eye(J.shape[1]), J.T @ r)
        b_new = b + delta_b
        r_new = self._residual(b_new)
        loss_new = np.sum(r_new**2)
        if method == "GN":
            b = b_new
            r = r_new
        loss = np.sum(r**2)
        losses.append(loss)
        p_s.append(b)
    return p_s, losses
```

2.2.2 Levenberg-Marquardt Implementation

```
def fit(self, b_init, method="GN", epochs=100, tol=1e-6, miuk=1e-4):
    b = np.array(b_init, dtype=float)
    p_s = []
    p_s.append(b)
    self.method = method
    losses = []
    r = self._residual(b)
    loss = np.sum(r**2)
    losses.append(loss)
    for ii in range(epochs):
        r = self._residual(b)
        J = self._jacobian(b)
        elif method == "LM":
            delta_b = -np.linalg.solve(J.T @ J + miuk * np.eye(J.shape[1]), J.T @ r)
        b_new = b + delta_b
        r_new = self._residual(b_new)
        loss_new = np.sum(r_new**2)
        if method == "LM":
            if loss_new < loss:
                b = b_new
                miuk = max(miuk / 10, 1e-7)
            else:
                miuk = min(miuk * 10, 1e4)
        if np.sqrt(np.sum(r**2)) < tol:
            break
        loss = np.sum(r**2)
        losses.append(loss)
        p_s.append(b)
    return p_s, losses
```

3 Results

3.1 Experimental Setup

- Initial guess: $(10000, 5000)$
- Convergence tolerance: 10^{-8}
- Maximum iterations: 10
- LM damping parameter: $\mu_0 = 10^{-3}$

```
Model = NLSF(xn, yn, tn, v)
b_init = [10000, 5000]
method1 = "GN"
method2 = "LM"
```

```python
epochs = 10
tol = 1e-8
miuk = 1e-3
p_s1, losses1 = Model.fit(b_init.copy(), "GN", epochs, tol, miuk)
p_s2, losses2 = Model.fit(b_init.copy(), "LM", epochs, tol, miuk)
```

3.2 Results Analysis

The experimental results show:

- Initial Guess: $(10000, 5000)$
- True Source: $(0, 0)$
- GN Result: $(70.89, -42.68)$, Error: 82.75m, Iterations: 10, Final Loss: 0.022029
- LM Result: $(70.90, -42.68)$, Error: 82.75m, Iterations: 10, Final Loss: 0.022029

Figure 2 shows the convergence behavior of both algorithms. The Gauss-Newton method demonstrates faster initial convergence, while the Levenberg-Marquardt method exhibits slower but more stable convergence due to its adaptive damping mechanism.

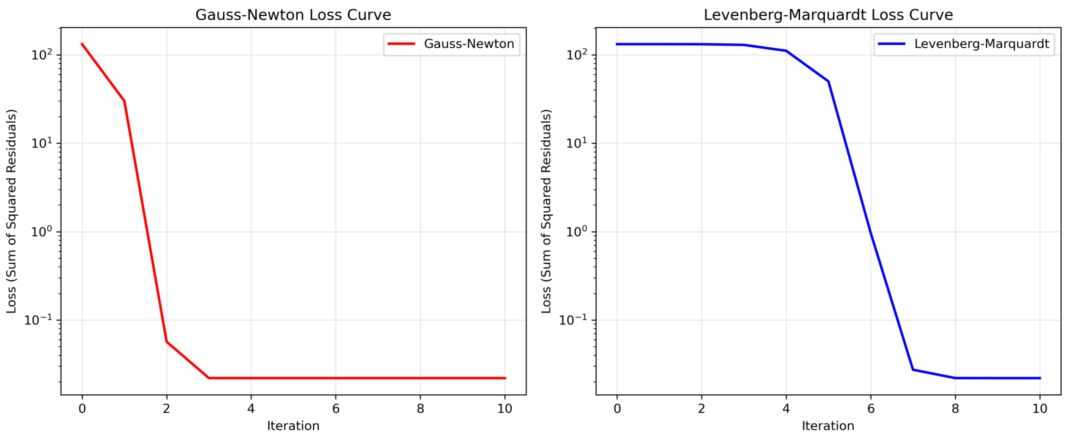

Figure 2 Convergence comparison: GN shows faster initial descent while LM provides more stable convergence.

3.3 Localization Results

Figure 3 presents the final positioning results. Both algorithms achieve similar accuracy despite the challenging initial conditions with the guess being approximately 11180m away from the true source.

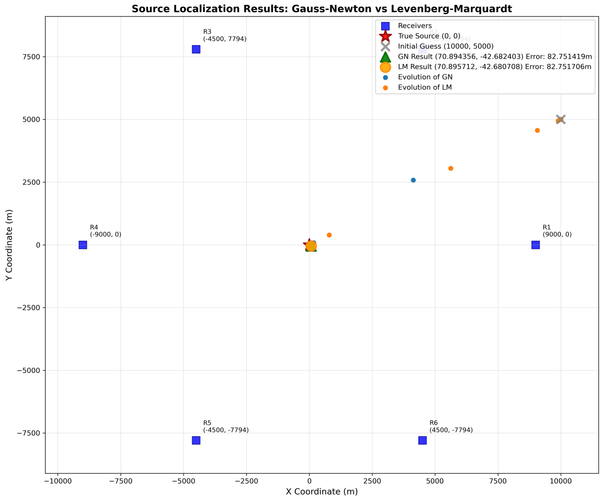

Figure 3 Source localization results showing both methods successfully converge to similar final positions near the true source location.

3.4 Conclusion

Both algorithms successfully solve the source localization problem. The key difference lies in their convergence characteristics: Gauss-Newton offers faster convergence while Levenberg-Marquardt provides more stable and robust optimization through its adaptive damping strategy.