Optimization Algorithms: Newton Method, Gauss-Newton Method, and Levenberg-Marquardt Algorithm

1 Newton Method

The Newton method is an iterative optimization algorithm used to find the stationary points of a function, where the gradient is zero. It uses the first and second derivatives (Hessian) of the function to iteratively update the solution.

1.1 Algorithm

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the Newton method updates the solution $\mathbf{x}$ iteratively as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$$

where $\mathbf{H}_f(\mathbf{x}_k)$ is the Hessian matrix of f at $\mathbf{x}_k$, and $\nabla f(\mathbf{x}_k)$ is the gradient vector of f at $\mathbf{x}_k$.

1.2 Example: Rosenbrock Function

The Rosenbrock function is defined as:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

We apply the Newton method to minimize this function starting from the initial point $\mathbf{x}_0 = [-2, 2]^T$.

```
% Newton Method for Rosenbrock Function
rosenbrock = @(x) 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
rosenbrock_grad = @(x) [-400*x(1)*(x(2)-x(1)^2) - 2*(1-x(1)); 200*(x(2)-x(1)^2)];
rosenbrock_hessian = @(x) [1200*x(1)^2 - 400*x(2) + 2, -400*x(1); -400*x(1), 200];

x = [-2; 2];
epsilon = 1e-4;
max_iter = 10000;
iter = 0;
```

```
while norm(rosenbrock_grad(x)) > epsilon && iter < max_iter
    x = x - rosenbrock_hessian(x) \ rosenbrock_grad(x);
    iter = iter + 1;
end

fprintf('Newton Method for Rosenbrock function:\n');
fprintf('Solution: [%f, %f]\n', x);
fprintf('Iterations: %d\n\n', iter);
```

2 Gauss-Newton Method

The Gauss-Newton method is an optimization algorithm used to solve non-linear least squares problems. It approximates the Hessian matrix using the Jacobian matrix of the residuals.

2.1 Algorithm

Given a residual function $\mathbf{r} : \mathbb{R}^n \to \mathbb{R}^m$, where $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2$, the Gauss-Newton method updates the solution $\mathbf{x}$ iteratively as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\mathbf{J}_r(\mathbf{x}_k)^T \mathbf{J}_r(\mathbf{x}_k)\right)^{-1} \mathbf{J}_r(\mathbf{x}_k)^T \mathbf{r}(\mathbf{x}_k)$$

where $\mathbf{J}_r(\mathbf{x}_k)$ is the Jacobian matrix of $\mathbf{r}$ at $\mathbf{x}_k$.

2.2 Example: Least-Squares Problem

Consider fitting a curve to noisy data:

$$y = 2e^{0.3t} + \epsilon$$

where ϵ is random noise. We define the residual function and its Jacobian, and use the Gauss-Newton method to find the parameters.

```
% Generate noisy data
t = linspace(0, 10, 100)';
y_true = 2 * exp(0.3 * t);
y = y_true + 0.2 * randn(size(t));

% Residual function and Jacobian
residual = @(params) params(1) * exp(params(2) * t) - y;
jacobian = @(params) [exp(params(2) * t), params(1) * t .* exp(params(2) * t)];

% Gauss-Newton Method
params = [1; 0.1]; % Initial guess
max_iter = 100;
```

```
epsilon = 1e-4;

for iter = 1:max_iter
    J = jacobian(params);
    r = residual(params);
    delta = -(J' * J) \ (J' * r);
    params = params + delta;

    if norm(delta) < epsilon
        break;
    end
end

fprintf('Gauss-Newton for least-squares problem:\n');
fprintf('Parameters: [%f, %f]\n', params);
fprintf('Iterations: %d\n\n', iter);
```

3 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm is a modification of the Gauss-Newton method to handle cases where the Hessian matrix might be singular or near-singular. It combines the concepts of the Gauss-Newton method and gradient descent.

3.1 Algorithm

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the Levenberg-Marquardt algorithm updates the solution $\mathbf{x}$ iteratively as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{H}_f(\mathbf{x}_k) + \lambda \mathbf{I})^{-1} \nabla f(\mathbf{x}_k)$$

where λ is a damping parameter and $\mathbf{I}$ is the identity matrix. The parameter λ is adjusted at each iteration to ensure convergence.

3.2 Example: Rosenbrock Function

We apply the Levenberg-Marquardt algorithm to minimize the Rosenbrock function starting from the initial point $\mathbf{x}_0 = [-2, 2]^T$.

```
% Levenberg-Marquardt Method
x = [-2; 2];
lambda = 0.01;

for iter = 1:max_iter
    H = rosenbrock_hessian(x) + lambda * eye(2);
    g = rosenbrock_grad(x);
```

```
    dx = -H \ g;

    if norm(g) < epsilon
        break;
    end

    if rosenbrock(x + dx) < rosenbrock(x)
        x = x + dx;
        lambda = lambda / 10;
    else
        lambda = lambda * 10;
    end
end

fprintf('Levenberg-Marquardt for Rosenbrock function:\n');
fprintf('Solution: [%f, %f]\n', x);
fprintf('Iterations: %d\n\n', iter);
```

3.3 Example: Least-Squares Problem

We use the Levenberg-Marquardt algorithm to solve the least-squares problem of fitting a curve to noisy data.

```
% Levenberg-Marquardt Method for Least-Squares
params = [1; 0.1]; % Initial guess
lambda = 0.01;

for iter = 1:max_iter
    J = jacobian(params);
    r = residual(params);
    H = J' * J + lambda * eye(2);
    g = J' * r;
    delta = -H \ g;

    if norm(g) < epsilon
        break;
    end

    if norm(residual(params + delta)) < norm(r)
        params = params + delta;
        lambda = lambda / 10;
    else
        lambda = lambda * 10;
    end
end
```

```
fprintf('Levenberg-Marquardt for least-squares problem:\n');
fprintf('Parameters: [%f, %f]\n', params);
fprintf('Iterations: %d\n\n', iter);
```

4 Solutions

4.1 Rosenbrock Function

4.1.1 Newton Method

Solution:

$$\begin{bmatrix} 1.000000 \\ 1.000000 \end{bmatrix}$$

Iterations: 5

The Newton method converged to the correct minimum of the Rosenbrock function, $[1, 1]$, in 5 iterations. This is a satisfactory result and demonstrates the efficiency of the Newton method when the Hessian is accurately computed.

4.1.2 Levenberg-Marquardt Algorithm

Solution:

$$\begin{bmatrix} 1.000000 \\ 0.999999 \end{bmatrix}$$

Iterations: 44

The Levenberg-Marquardt algorithm also converged to the correct minimum but required more iterations (44). This is expected because the Levenberg-Marquardt algorithm is designed to handle cases where the Hessian may not be well-conditioned, making it more robust but often slower than the Newton method.

4.2 Least-Squares Problem

4.2.1 Gauss-Newton Method

Warning: Matrix is close to singular or badly scaled.

Parameters:

$$\begin{bmatrix} 0.000000 \\ 26.732431 \end{bmatrix}$$

Iterations: 9

The Gauss-Newton method encountered a numerical issue, indicated by the warning about the matrix being close to singular. The resulting parameters are not satisfactory as the first parameter should be close to 2 and the second should be close to 0.3 based on the generated data. The Gauss-Newton method's reliance on the Jacobian approximation can lead to issues when the problem is ill-conditioned.

4.2.2 Levenberg-Marquardt Algorithm

Parameters:

$$\begin{bmatrix} 2.021720 \\ 0.298689 \end{bmatrix}$$

Iterations: 18

The Levenberg-Marquardt algorithm provided parameters close to the expected values of $[2, 0.3]$, showing it handled the least-squares problem well. The higher number of iterations compared to the Gauss-Newton method is typical for the Levenberg-Marquardt algorithm due to its iterative adjustment of the damping parameter to balance between the Gauss-Newton and gradient descent updates.

4.3 Conclusion

Newton Method: Efficient and accurate for the Rosenbrock function.

Levenberg-Marquardt Algorithm: Robust and accurate for both the Rosenbrock function and the least-squares problem but required more iterations.

Gauss-Newton Method: Failed to produce satisfactory results for the least-squares problem due to numerical instability.

Overall, the Levenberg-Marquardt algorithm's results are the most satisfactory, showing its robustness in handling both well-conditioned and ill-conditioned problems. The Newton method performs well for the Rosenbrock function but is sensitive to the quality of the Hessian matrix. The Gauss-Newton method's failure highlights the importance of robustness in optimization algorithms.