

CHBE 552 Project Report

**Comparison of Gauss-Newton Method with Marquardt Modification with
Nelder-Mead Algorithm for Parameter Estimation in Algebraic Models**

by

Jincong Li

M.Eng, The University of British Columbia, 2024

April 13, 2024

© Jincong Li, 2024

Contents

1	Abstract	2
2	Introduction	2
2.1	Adsorption of 1,2-Dichloropropane on Activated Carbon	2
2.2	Kinetics of MnO ₂ -Catalyzed Acetic Acid Oxidation in Supercritical Water . . .	2
2.3	The Gauss-Newton Method with Marquardt Modification	3
2.3.1	Gauss-Newton Method	3
2.3.2	Formulation	3
2.3.3	Marquardt's Modification (Levenberg-Marquardt Algorithm)	3
2.3.4	Application	3
2.4	Computing Confidence Intervals for Estimated Parameters	4
2.4.1	Steps to Compute the Confidence Intervals	4
2.5	Nelder-Mead Algorithm	4
2.6	Bootstrap Method for Computing Confidence Intervals	5
2.6.1	Steps in Bootstrap Method	5
2.6.2	Advantages of the Bootstrap Method	6
2.6.3	Limitations	6
3	Methodology	6
4	Result	9
5	Conclusion	9
6	Reference	9

1 Abstract

asd

2 Introduction

Parameter estimation plays a crucial role in modeling and understanding various processes across engineering and science disciplines. Accurate estimation of model parameters allows for the prediction and optimization of system behavior. This project aims to compare the effectiveness of two widely used optimization methods, Gauss-Newton method with Marquardt modification and Nelder-Mead Algorithm, in estimating parameters of algebraic models. The comparison will be based on solving two distinct chemical engineering problems: 1) Adsorption of 1,2-Dichloropropane on Activated Carbon[1], and 2) Kinetics of MnO₂-Catalyzed Acetic Acid Oxidation in Supercritical Water[2]. Note that the chemical nature of these two problems is not the main focus of this project, so the meaning of each variable is omitted in the following sections. The problems are treated as pure algebraic model.

2.1 Adsorption of 1,2-Dichloropropane on Activated Carbon

The algebraic model is given by

$$q = \frac{q_{\text{sat}} \cdot k \cdot p}{(1 + (k \cdot p)^t)^{\frac{1}{t}}} \quad (1)$$

where p is the input variable, q is the output variable, and q_{sat}, k, t are parameters of the model.

2.2 Kinetics of MnO₂-Catalyzed Acetic Acid Oxidation in Supercritical Water

The algebraic model for the second problem is given by

$$W_{FA0} = \frac{(1 + k_3 \cdot [O_2])^2 \cdot (k_2 \cdot [HOAC] \cdot X - \ln(1 - X))}{k_1 \cdot [HOAC] \cdot [O_2]} \quad (2)$$

where W_{FA0} is the input variable, X is the output variable, and k_1, k_2, k_3 are parameters of the model.

As mentioned, this work will investigate into comparing two different optimization method in parameter estimation for algebraic models over two well-studied cases. One method is chosen from the gradient-based methods, that is the Gauss-Newton method with Marquardt modification. The other method is the Nelder-Mead algorithm which is a kind of Simplex methods from the direct search field. They will be briefly reviewed here as well as the methods for computing the confidence intervals.

2.3 The Gauss-Newton Method with Marquardt Modification

2.3.1 Gauss-Newton Method

The Gauss-Newton method is an algorithm used for solving non-linear least squares problems, which are often encountered in parameter estimation for algebraic models. This method iteratively refines parameter estimates to minimize the sum of squared differences between observed and model-predicted values.

2.3.2 Formulation

Given a set of observations y_i and a non-linear model $f(x_i, \beta)$, where x_i are the independent variables and β represents the parameters to be estimated, the objective is to minimize the sum of squared residuals $S(\beta)$ defined as:

$$S(\beta) = \sum_{i=1}^n (y_i - f(x_i, \beta))^2$$

The parameter updates in each iteration are calculated using:

$$\beta_{\text{new}} = \beta_{\text{old}} + (J^T J)^{-1} J^T r$$

Here, J is the Jacobian matrix of partial derivatives of the model functions with respect to the parameters β , and r is the vector of residuals between the observed and predicted values.

2.3.3 Marquardt's Modification (Levenberg-Marquardt Algorithm)

The Levenberg-Marquardt algorithm modifies the Gauss-Newton method by introducing a damping factor λ to improve the convergence properties, especially in situations where the Jacobian matrix $J^T J$ is not invertible or near singular. The updated formula for parameter adjustment becomes:

$$\beta_{\text{new}} = \beta_{\text{old}} + (J^T J + \lambda I)^{-1} J^T r$$

Here, I is the identity matrix. The value of λ adjusts dynamically during iterations: when λ is high, the algorithm behaves more like gradient descent, taking smaller steps; when λ is low, it behaves more like the Gauss-Newton method, taking larger, more aggressive steps.

2.3.4 Application

The essence of Marquardt's modification is to combine the strengths of both the Gauss-Newton method and gradient descent, allowing for a more versatile and robust approach to parameter estimation in non-linear models. The algorithm iteratively adjusts λ and the parameters β until convergence is reached, typically when the change in the sum of squared residuals is below a certain threshold.

2.4 Computing Confidence Intervals for Estimated Parameters

The confidence interval information of the estimated parameters is also required in this work, since it could indicate the reliability of the parameter values estimated by optimization method, such as the Gauss-Newton method with Marquardt modification. The confidence intervals for the estimated parameters are computed based on the statistical properties of the least squares estimators, using their standard errors and the assumption that they follow a t-distribution, especially in the context of small sample sizes.

2.4.1 Steps to Compute the Confidence Intervals

1. Covariance Matrix Calculation:

- After obtaining the parameter estimates through an optimization process such as the Gauss-Newton method (modified by Marquardt), calculate the covariance matrix of the parameter estimates.
- This matrix is derived from the inverse of the Hessian matrix, which in the context of the Gauss-Newton method, is approximated by $J^T J$, where J is the Jacobian matrix of the partial derivatives of the residuals with respect to the parameters.
- If $J^T J$ is singular or near-singular, indicating potential issues with model identifiability or data collinearity, use the pseudo-inverse for stability.

2. Standard Errors of the Parameter Estimates:

- The diagonal elements of the covariance matrix provide the variances of the parameter estimates.
- The standard errors of the estimates are obtained by taking the square root of these diagonal elements.

3. Confidence Interval Calculation:

- With the standard errors, compute the confidence intervals for the parameters.
- For a specified confidence level (e.g., 95%), determine the critical value from the t-distribution, considering the degrees of freedom, which typically depend on the number of data points and the number of parameters.
- The confidence interval for each parameter is calculated as:

$$\text{Parameter Estimate} \pm (t\text{-critical value} \times \text{Standard Error})$$

2.5 Nelder-Mead Algorithm

The Nelder-Mead algorithm, also known as the Downhill Simplex Method or Amoeba Method, is an optimization method used to find the minimum or maximum of an objective function in a multidimensional space. It is particularly suitable for problems where the objective function does not have a derivative or is not smooth.

- **Initialization:** Start with a simplex, which in two dimensions is a triangle and in three dimensions is a tetrahedron. This simplex is formed by $n + 1$ vertices for an n -dimensional problem, where each vertex represents a potential solution.
- **Evaluation:** Calculate the objective function at each vertex.
- **Transformation:** Perform one of four actions based on the values of the objective function at the vertices:
 1. **Reflection:** Creates a new point by reflecting the worst point of the simplex across the opposite face of the simplex.
 2. **Expansion:** If the reflection results in a better point, the algorithm tries to take a larger step in the same direction to see if the improvement continues.
 3. **Contraction:** If reflection does not improve the solution, the algorithm takes a smaller step from the worst point towards the simplex.
 4. **Shrinkage:** If contraction still does not yield a better point, the algorithm reduces the size of the simplex towards the best current point.
- **Iteration:** Repeat the evaluation and transformation steps until a stopping criterion is met, such as a minimum change in the objective function value or a maximum number of iterations.

Note: The Nelder-Mead method does not require derivatives of the objective function, making it useful for optimization problems where derivatives are difficult or impossible to calculate. However, it is not guaranteed to find the global minimum, especially in problems with multiple local minima.

2.6 Bootstrap Method for Computing Confidence Intervals

However, in the Simplex family method, the derivatives are not available, which means the covariance matrix method for computing the confidence interval is not applicable. Thus, we need a new method in this case to capture the confidence interval information. The bootstrap method is a powerful statistical tool used to estimate the distribution of a statistic by resampling with replacement from the original data. It is particularly useful in situations where the theoretical distribution of the statistic is unknown or complex.

2.6.1 Steps in Bootstrap Method

1. Data Resampling:

- Generate a large number of bootstrap samples from the original dataset by resampling with replacement. Each bootstrap sample is the same size as the original dataset.

2. Statistic Calculation:

- Calculate the statistic of interest (e.g., mean, median) for each bootstrap sample. This process creates a distribution of the statistic known as bootstrap replicates.

3. Confidence Interval Estimation:

- **Percentile Method:** For a 95% confidence interval, use the 2.5th and 97.5th percentiles of the bootstrap statistics.
- **Bias-Corrected and Accelerated (BCa) Method:** This more complex method adjusts for both bias and skewness in the bootstrap distribution, providing more accurate confidence intervals.

4. Interpretation:

- The resulting confidence interval estimates where the true parameter lies with a specified level of confidence.

2.6.2 Advantages of the Bootstrap Method

- Flexible and applicable to various statistics.
- Simple to implement with modern computing power.
- Nonparametric and does not assume a specific parametric distribution.

2.6.3 Limitations

- Requires a large sample size to be effective.
- Computationally demanding due to the need to generate thousands of bootstrap samples.
- May not perform well with very small sample sizes or highly skewed data.

3 Methodology

The two problems and two optimization methods mentioned in the introduction method will be implemented in Python. Key algorithm procedures are described as followings: note that in each step, the first problem is used as example, the procedure for the second problem will be analogical.

1. **Model Function:** The algebraic model of the problem will be converted to the model function that could be called by other functions. For instance:

```
def model(p, q_sat, k, t):  
    # Ensure all operations are element-wise  
    p = np.array(p, dtype=float)  
    term = (1 + (k * p) ** t)  
    return q_sat * k * p / term ** (1 / t)
```

2. **Residual Function:** For the Levenberg-Marquardt algorithm, this function calculates the residual between the estimated output and the experiment output.

```
def residual(params, p, q_obs):
    q_sat, k, t = params
    return q_obs - model(p, q_sat, k, t)
```

This function calculates the difference between observed values q_{obs} and the model predictions.

3. **Jacobian Matrix Calculation:** The Jacobian matrix of the model with respect to the parameters q_{sat}, k, t is calculated for optimization. Detailed implementation is omitted here since the partial derivatives are too long to fit in.
4. **Levenberg-Marquardt Algorithm/Nelder-Mead Algorithm:** Levenberg-Marquardt Algorithm is shown here as an example, the implementation for Nelder-Mead Algorithm will be analogical.

```
def levenberg_marquardt(p, q, params_guess, max_iter=100,
                        lambda_inc=10, lambda_dec=10, lambda_init=0.01):
    params = np.array(params_guess, dtype=float)
    n = len(q)
    iteration_details = []

    # Initial calculation
    r = residual(params, p, q)
    J = jacobian(p, params)

    A = J.T @ J
    g = J.T @ r
    lambda_ = lambda_init * np.max(np.diag(A))
    objective = np.sum(r**2)

    for i in range(max_iter):
        # Solve for parameter update
        try:
            delta = np.linalg.solve(A + lambda_ * np.eye(len(params)),
                                    g)
        except np.linalg.LinAlgError:
            print(f"Iteration {i}: Singular matrix encountered.
                  Adjusting lambda.")
            lambda_ *= lambda_inc
            continue
        # Check if improvement
        new_params = params + delta
        new_r = residual(new_params, p, q)
        new_objective = np.sum(new_r**2)
```

5. **Parameter Adjustment and Convergence Checking:** Parameters are updated based on improvement in the objective function, iterating until convergence criteria are met.
-


```

if new_objective < objective:
    # Update is successful, decrease lambda (move towards
    Gauss-Newton method)
    lambda_ /= lambda_dec
    params = new_params
    objective = new_objective
    r = new_r
    J = jacobian(p, params)
    A = J.T @ J
    g = J.T @ r
    #print(f"Iteration {i}: Success - Parameters updated to:
    {params} with objective: {objective}")
else:
    # Update is not successful, increase lambda (move towards
    gradient descent)
    lambda_ *= lambda_inc
    #print(f"Iteration {i}: No improvement - Lambda adjusted to:
    {lambda_}")

```

6. **Error Estimation (Confidence Interval Computation):** Post-fitting, the covariance matrix of the parameters is calculated using the Jacobian matrix from the final iteration to estimate their standard errors for Levenberg-Marquardt Algorithm. And the bootstrap method applied for Nelder-Mead Algorithm is shown here as an example:

```

def bootstrap_CI(data, model_func, optimization_func,
    initial_guesses, n_iterations=1000, ci=95):

    np.random.seed(42) # For reproducibility
    p_data, q_data = data
    estimates = []

    for _ in range(n_iterations):
        # Bootstrap sample
        bs_indices = np.random.choice(range(len(p_data)),
            size=len(p_data), replace=True)
        bs_p_data = p_data[bs_indices]
        bs_q_data = q_data[bs_indices]

        # Update objective function for bootstrap sample
        def bs_objective_function(params):
            predicted_q = model_func(bs_p_data, *params)
            return np.sum((bs_q_data - predicted_q) ** 2)

        # Estimate parameters for bootstrap sample
        bs_estimates = optimization_func(bs_objective_function,
            initial_guesses)
        estimates.append(bs_estimates)

    # Calculate confidence intervals
    estimates = np.array(estimates)

```

```
lower_bounds = np.percentile(estimates, (100 - ci) / 2, axis=0)
upper_bounds = np.percentile(estimates, 100 - (100 - ci) / 2, axis=0)

return list(zip(lower_bounds, upper_bounds))
```

7. **Handling Multiple Data Sets & Execution & Post-Processing:** The implementation is set up to handle multiple data sets for different scenarios, each characterized by different initial parameters and data points. Then the algorithm functions will be called by specifying corresponding initial guess and experiment data, and the result of estimated values of parameters will be returned as well as the confidence interval information. Finally, those results will be printed to the console and the comparison between experiment data and estimated data will be plotted as well.

4 Result

Iteration0 : Success-Parameters updated to : [4.1894683319.010307570.31511711] with objective : 20.

using the same initial guess, we can see that the ref values generally stay in the 95 CI, which means the Parameters are generally ok. In this problem, as we can see in the definition of the model, the output variable X is included in the RHS of the model, which means that it could not be solved explicitly, and this fact requires the code to solve for X numerically and the numerical solver package "scipy.optimize.brentq" is applied.

5 Conclusion

6 Reference

- Zhang et al., "Adsorption of 1,2-Dichloropropane on Activated Carbon", J Chem Eng Data, 2001, 46, 662-664.
- Yu and Savage, "Kinetics of MnO₂-catalysed acetic acid oxidation in supercritical water", Ind. Eng. Chem. Res. 2000, 39, 4014-4019.