**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 16, 2024

# Contents

# 1 Abstract

This project report presents a comprehensive comparison of two parameter estimation methods, the Gauss-Newton method with Marquardt modification and the Nelder-Mead algorithm, as applied to two specific algebraic models in chemical engineering. These models relate to the adsorption of 1,2-Dichloropropane on Activated Carbon and the kinetics of MnO2-catalyzed Acetic Acid oxidation in supercritical water. The objective is to evaluate and compare the effectiveness of these optimization methods in accurately estimating the parameters of these models based on their performance, including the reliability of the estimated parameters as expressed through confidence intervals. The methods were implemented in Python, and parameter estimation was performed on experimental data from referenced studies. The study highlights the challenges and effectiveness of each method under the constraints of the complex algebraic models used in chemical engineering, providing insights into their practical applications and limitations.

# 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 models.

## 2.1 Adsorption of 1,2-Dichloropropane on Activated Carbon

The algebraic model is given by

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

where $p$ is the input variable, $q$ is the output variable, and $q_{\text{sat}}, k, t$ 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

$$W_{F_A 0} = \frac{(1 + k_3 \cdot [O_2])^2 \cdot (k_2 \cdot [HOAC] \cdot X - \ln(1 - X))}{k_1 \cdot [HOAC] \cdot [O_2]} \tag{2}$$

where $W_{F_A0}$ is the input variable, $X$ is the output variable, and $k1, k2, k3$ 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 $\beta$ 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 $\beta$, 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 $\lambda$ 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 $\lambda$ adjusts dynamically during iterations: when $\lambda$ is high, the algorithm behaves more like gradient descent, taking smaller steps; when $\lambda$ 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 $\lambda$ and the parameters $\beta$ 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 relibility 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. Notice that in the implementation, pesudoinverse is applied when neccessery to avoid ill-conditioned matrix at the optimum point or away from optimum point.

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 $q_{\text{obs}}$ and the model predictions.

3. **Jacobian Matrix Calculation:** The Jacobian matrix of the model with respect to the parameters $q_{\text{sat}}, k, t$ 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.

8. **Special Treatment for Problem 2:** 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 (even in a *Log*), 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.

```
        def inverse_model_function_bounded(W_F_A0, k1, k2, k3):
        """
        Numerically solve for X given W_F_A0 using a bounded solver.
        """
        # Define the equation to solve
        def equation(X):
            return ((1 + k3 * O2)**2) * (k2 * HOAC * X - np.log(1 - X)) /
                (k1 * HOAC * O2) - W_F_A0

        # Use brentq for solving within bounds, providing more stability
            near edges
        try:
            X_solution = brentq(equation, 0.0001, 0.9999) # Avoiding exact
                0 or 1 to prevent log issues
            return X_solution
        except ValueError as e:
            print(f"No solution found for W_F_A0={W_F_A0} within the
                bounds: {e}")
            return np.nan # Return NaN if no solution is found
```

# 4 Result

The experiment data that the code will intake as input as provided in the reference paper, and they are presented in the Appendix.

## 4.1 Adsorption of 1,2-Dichloropropane on Activated Carbon

Table 1: Results of Parameter Estimation for Problem 1

| Method | Cases($T$) | $q_{sat}$ | $k$ | $t$ |
|---|---|---|---|---|
| GNM | 303 | 4.31 ± 0.19 | 17.73 ± 5.73 | 0.46 ± 0.05 |
| | 338 | 5.2 ± 0.56 | 7.99 ± 2.6 | 0.37 ± 0.05 |
| | 373 | 4.56 ± 1.09 | 1.72 ± 0.41 | 0.38 ± 0.07 |
| | 423 | 6.77 ± 6.75 | 0.8 ± 0.23 | 0.28 ± 0.13 |
| | 473 | 4.41 ± 12.2 | 0.27 ± 0.17 | 0.3 ± 0.32 |
| NMA | 303 | 4.31[4.25 4.47] | 17.81[16.21 20.28] | 0.45[0.43 0.47] |
| | 338 | 5.58[5.21 5.29] | 9.99[7.82 9.2 ] | 0.34[0.33 0.36] |
| | 373 | 5.68[5.65 5.68] | 2.1[1.98 2.21] | 0.33[0.32 0.33] |
| | 423 | 5.77[5.38 5.85] | 0.74[0.73 0.78] | 0.31[0.3 0.31] |
| | 473 | 2.92[2.82 2.99] | 0.3[0.26 0.32] | 0.35[0.34 0.37] |

Note that "GNM" refers to Gauss-Newton Method with Marquardt Modification and "NMA" refers to Nelder-Mead Algorithm. And the bracket following the values in NMA is the 95% confidence interval since the uncertainties are not symmetric.

As presented in table 1, the estimated parameter values from two methods as listed, notice for both methods, same initial guess of parameters are used to ensure the comparability. Comparing with the values reported by the reference (shown in figure 1 below), GNM estimate the parameters perfectly for the first three cases (303T, 338T, and 373T), the estimated values are exactly the same as reported in the reference with little larger 95% confidence intervals which is acceptable since they are generally reasonable. For the last two cases, GNM returns the parameters slightly different from the reference values with much larger 95% confidence intervals which are not reasonable. One reason could be the number of data points are not large enough. Especially for $q_{sat}$, the uncertainties are even larger than the value of the parameters itself. Contrastively, NMA predicts the parameters for the last two cases better even though some error still exist but the smaller 95% confidence intervals indicates its reliability. NMA also successfully estimates the parameter in first three case with some error appears. Comparison between experiment data and estimated data are plotted in figure 2 and 3. In conclusion, GNM and NMA both estimate the parameter values rationally, but their performance might depend on the input data structure/distribution. One method could be better over the other one in some scenarios just as what we observed in the problem 1.

Regarding the complexity of implementation, they are competitive. GNM requires careful treatment of the derivatives/Jacobian Matrix especially when the algebraic model is complex, whereas NMA requires the implementation of additional method for extracting the confidence interval information. Computation time demanded by both method are less than one minute to estimate the parameters, however, since NMA requires the Bootstrap method to compute the confidence interval which involves resampling the data up to the chosen iteration, it might require much more time if the iteration limit is large.

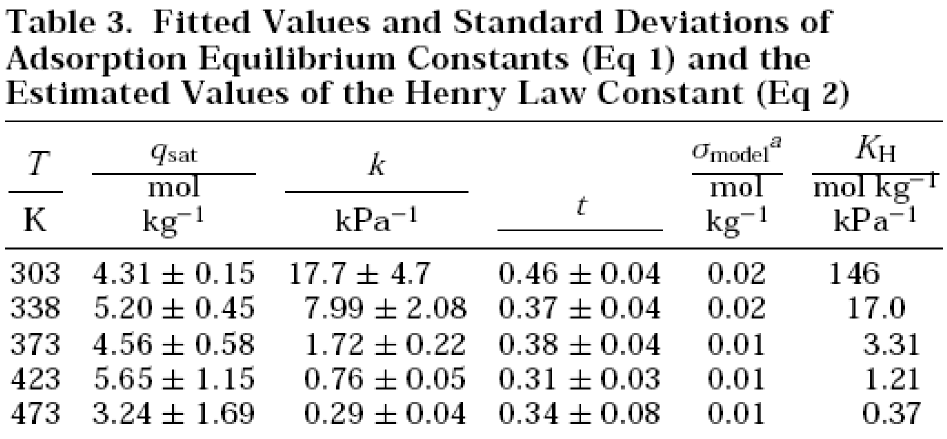

**Table 3. Fitted Values and Standard Deviations of Adsorption Equilibrium Constants (Eq 1) and the Estimated Values of the Henry Law Constant (Eq 2)**

| $T$ / K | $q_{sat}$ / mol kg$^{-1}$ | $k$ / kPa$^{-1}$ | $t$ | $\sigma_{model}$[a] / mol kg$^{-1}$ | $K_H$ / mol kg$^{-1}$ kPa$^{-1}$ |
|---|---|---|---|---|---|
| 303 | $4.31 \pm 0.15$ | $17.7 \pm 4.7$ | $0.46 \pm 0.04$ | 0.02 | 146 |
| 338 | $5.20 \pm 0.45$ | $7.99 \pm 2.08$ | $0.37 \pm 0.04$ | 0.02 | 17.0 |
| 373 | $4.56 \pm 0.58$ | $1.72 \pm 0.22$ | $0.38 \pm 0.04$ | 0.01 | 3.31 |
| 423 | $5.65 \pm 1.15$ | $0.76 \pm 0.05$ | $0.31 \pm 0.03$ | 0.01 | 1.21 |
| 473 | $3.24 \pm 1.69$ | $0.29 \pm 0.04$ | $0.34 \pm 0.08$ | 0.01 | 0.37 |

Figure 1: Parameter Values Reported in Reference for Problem 1[1]

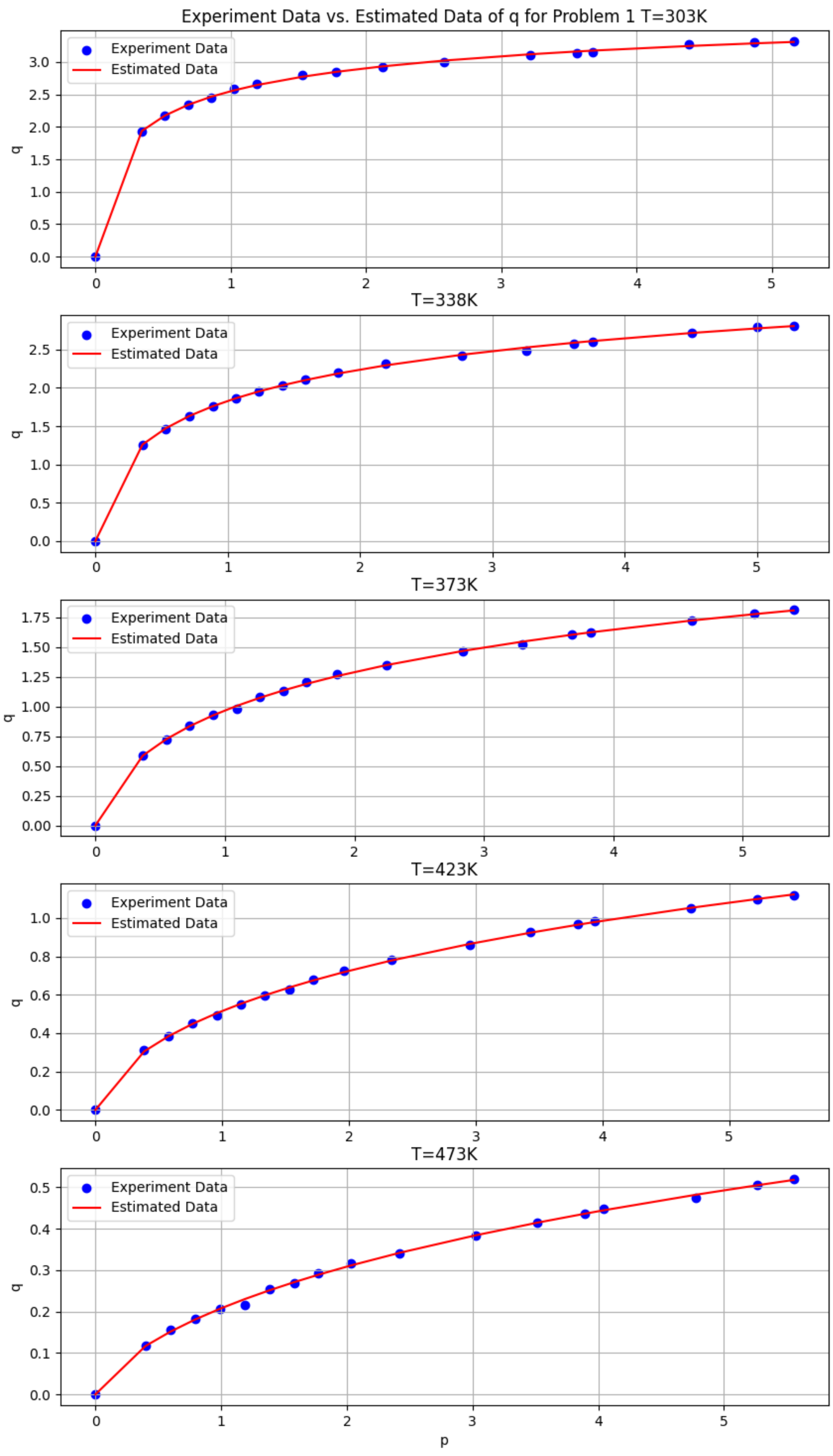

Figure 2: Experiment Data vs. Estimated Data of q for Problem 1 by GNM

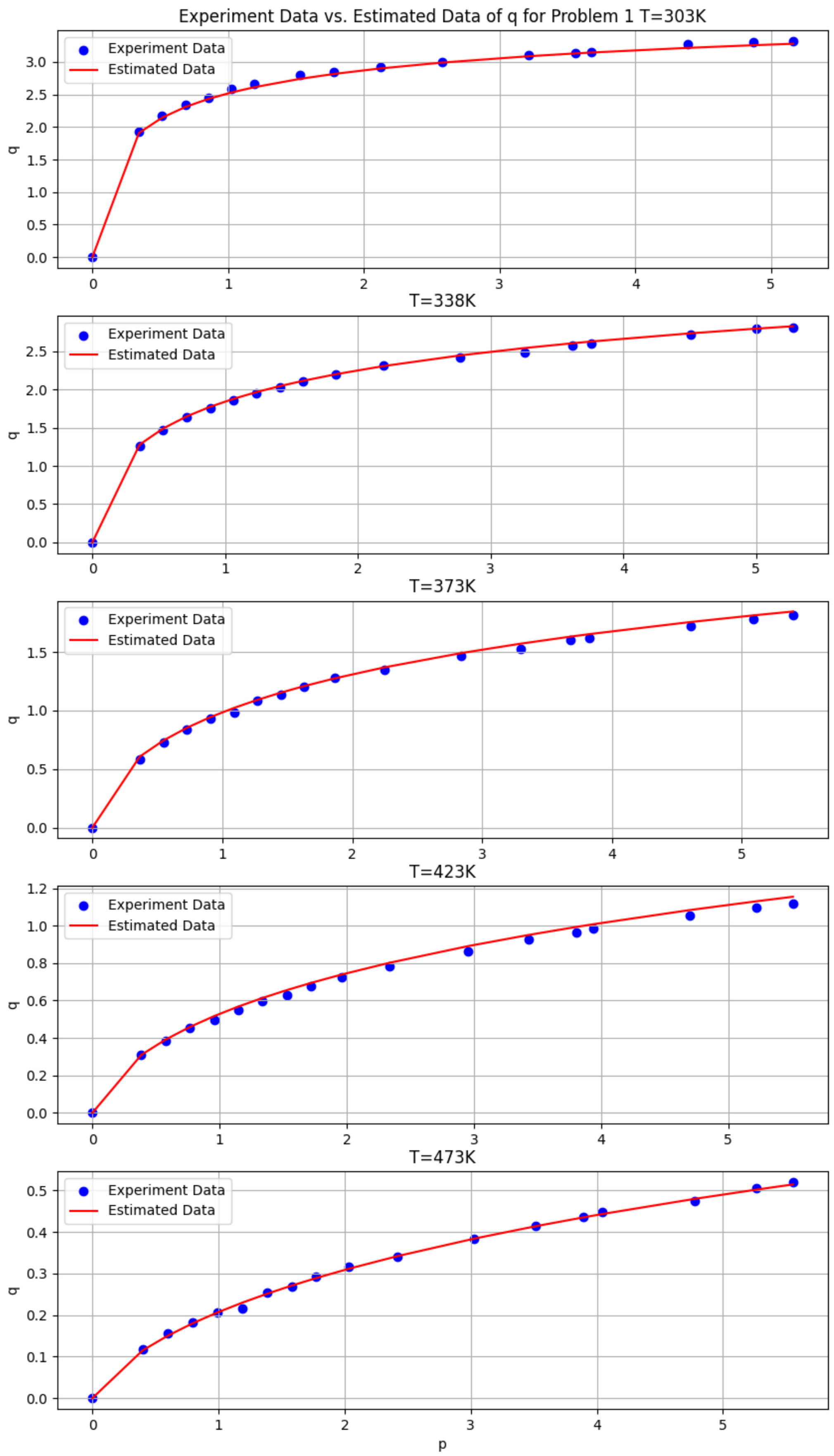

Figure 3: Experiment Data vs. Estimated Data of q for Problem 1 by NMA

## 4.2 Kinetics of MnO2-Catalyzed Acetic Acid Oxidation in Supercritical Water

As shown in figure 8 in Appendix, the experiment data varies a lot in terms of reaction temperature, pressure, and concentrations of species. Since the model itself is complex enough and for the performance of the code, only experiment data for $\frac{W}{F_{A0}}$and $X$ (HOAC convert percentage) in the cases where$[HOAC]$ = 3.2 and $[O_2]$ = 24 are used. Table 2 below presents the estimated values of parameters, and one can observe that they do not generally agree across methods. Figure 4 shows the reference values for those parameters which vary in 4 degree of magnitude, which I could not reasonably interpret. Thus, the validation is conducted by comparing the experiment data with the estimated data (as shown in figure 5 and 6).

Table 2: Results of Parameter Estimation for Problem 2

| Method | Cases | $k_1$ | $k_2$ | $k_3$ |
|---|---|---|---|---|
| GNM | $[HOAC]$ = 3.2 $[O_2]$ = 24 | 2.224 ± 1.019 | 1.392 ± 0.781 | -0.604 ± 1.321 |
| NMA | $[HOAC]$ = 3.2 $[O_2]$ = 24 | 1.247[0.252 2.476] | 0.803[0.162 2.283] | 0.431[0.43 0.47] |

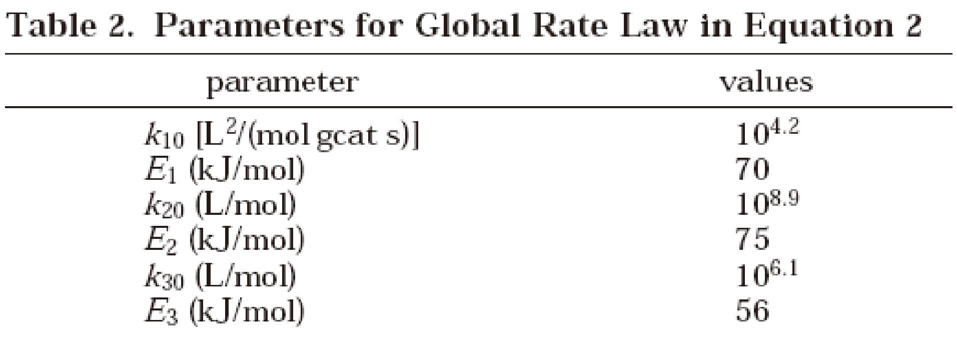

**Table 2. Parameters for Global Rate Law in Equation 2**

| parameter | values |
|---|---|
| $k_{10}$ [L$^2$/(mol gcat s)] | $10^{4.2}$ |
| $E_1$ (kJ/mol) | 70 |
| $k_{20}$ (L/mol) | $10^{8.9}$ |
| $E_2$ (kJ/mol) | 75 |
| $k_{30}$ (L/mol) | $10^{6.1}$ |
| $E_3$ (kJ/mol) | 56 |

Figure 4: Parameter Values Reported in Reference for Problem 2[2]

By observing figure 5 and 6, one could conclude that GNM and NMA are not predicting the parameters reasonably. Even though for some cases, the estimated X values agree with the experiment data, large error remains in other cases. The reason could be the complexity of the model and the inability of the code to solve numerically when applying those optimization methods.

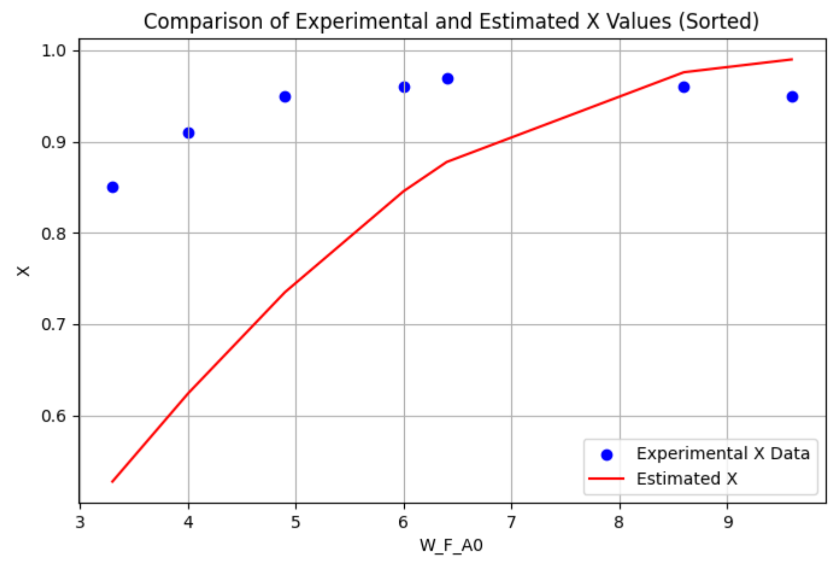

Figure 5: Experiment Data vs. Estimated Data of q for Problem 2 by GNM

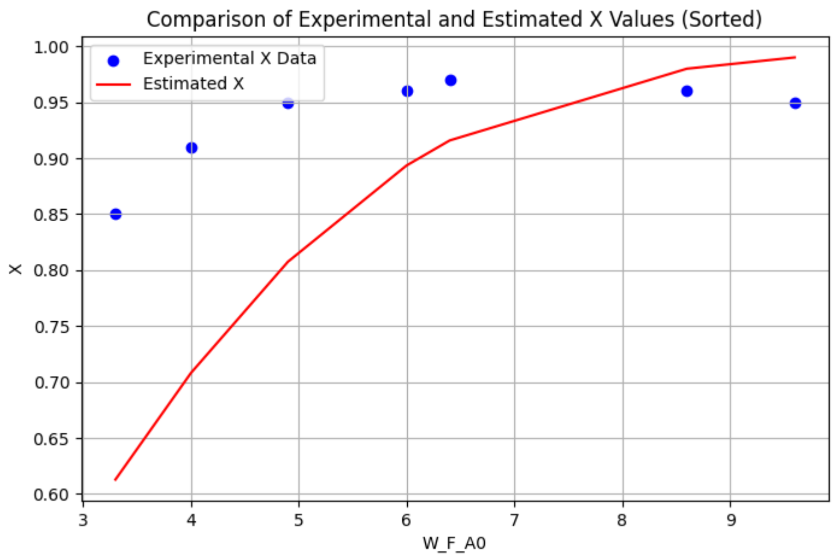

Figure 6: Experiment Data vs. Estimated Data of q for Problem 1 by NMA

## 5 Conclusion

The investigation into the Gauss-Newton method with Marquardt modification and the Nelder-Mead algorithm revealed distinct differences in their applicability and performance in the context of parameter estimation for complex chemical engineering processes. The Gauss-Newton method, enhanced by Marquardt's modification, demonstrated high accuracy in parameter estimation for simpler model scenarios but was less effective in handling more complex models with higher parameter uncertainty. On the other hand, the Nelder-Mead algorithm, while generally more robust in dealing with non-linear problems without derivative information, showed limitations in precision and confidence interval estimation, which was mitigated by employing the bootstrap method. Both methods showcased their respective strengths and weaknesses, indicating the necessity for careful selection of optimization techniques based on the model characteristics and data quality. Future work should focus on enhancing these methods for higher complexity models and integrating more advanced statistical tools to improve the reliability and accuracy of parameter estimations in scientific research and industrial applications.

## 6 Reference

1. Zhang et al., "Adsorption of 1,2-Dichloropropane on Activated Carbon", J Chem Eng Data, 2001, 46, 662-664.

2. Yu and Savage, "Kinetics of MnO2-catalysed acetic acid oxidation in supercritical water", Ind. Eng. Chem. Res. 2000, 39, 4014-4019.

## 7 Appendix

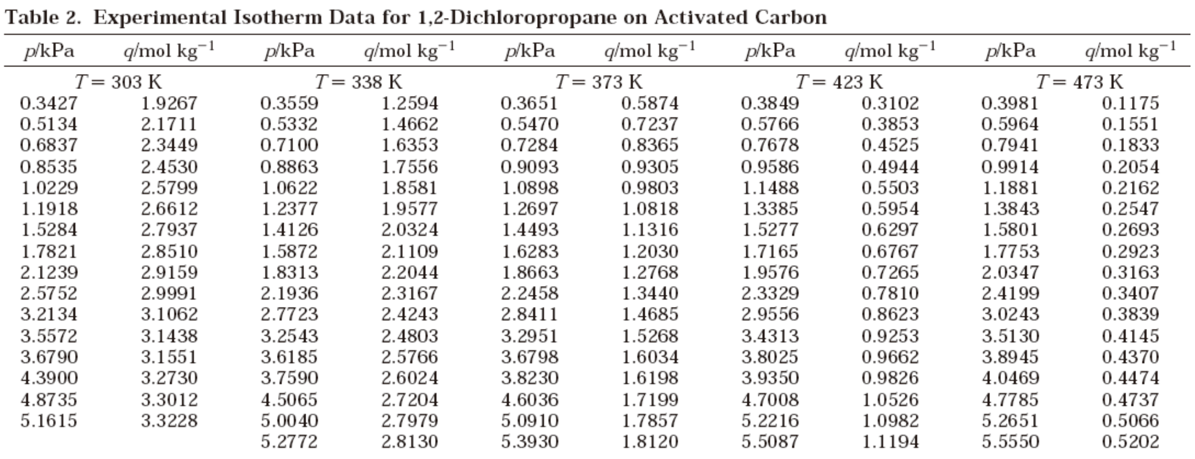

**Table 2. Experimental Isotherm Data for 1,2-Dichloropropane on Activated Carbon**

| $p$/kPa | $q$/mol kg$^{-1}$ | $p$/kPa | $q$/mol kg$^{-1}$ | $p$/kPa | $q$/mol kg$^{-1}$ | $p$/kPa | $q$/mol kg$^{-1}$ | $p$/kPa | $q$/mol kg$^{-1}$ |
|---|---|---|---|---|---|---|---|---|---|
| $T$ = 303 K | | $T$ = 338 K | | $T$ = 373 K | | $T$ = 423 K | | $T$ = 473 K | |
| 0.3427 | 1.9267 | 0.3559 | 1.2594 | 0.3651 | 0.5874 | 0.3849 | 0.3102 | 0.3981 | 0.1175 |
| 0.5134 | 2.1711 | 0.5332 | 1.4662 | 0.5470 | 0.7237 | 0.5766 | 0.3853 | 0.5964 | 0.1551 |
| 0.6837 | 2.3449 | 0.7100 | 1.6353 | 0.7284 | 0.8365 | 0.7678 | 0.4525 | 0.7941 | 0.1833 |
| 0.8535 | 2.4530 | 0.8863 | 1.7556 | 0.9093 | 0.9305 | 0.9586 | 0.4944 | 0.9914 | 0.2054 |
| 1.0229 | 2.5799 | 1.0622 | 1.8581 | 1.0898 | 0.9803 | 1.1488 | 0.5503 | 1.1881 | 0.2162 |
| 1.1918 | 2.6612 | 1.2377 | 1.9577 | 1.2697 | 1.0818 | 1.3385 | 0.5954 | 1.3843 | 0.2547 |
| 1.5284 | 2.7937 | 1.4126 | 2.0324 | 1.4493 | 1.1316 | 1.5277 | 0.6297 | 1.5801 | 0.2693 |
| 1.7821 | 2.8510 | 1.5872 | 2.1109 | 1.6283 | 1.2030 | 1.7165 | 0.6767 | 1.7753 | 0.2923 |
| 2.1239 | 2.9159 | 1.8313 | 2.2044 | 1.8663 | 1.2768 | 1.9576 | 0.7265 | 2.0347 | 0.3163 |
| 2.5752 | 2.9991 | 2.1936 | 2.3167 | 2.2458 | 1.3440 | 2.3329 | 0.7810 | 2.4199 | 0.3407 |
| 3.2134 | 3.1062 | 2.7723 | 2.4243 | 2.8411 | 1.4685 | 2.9556 | 0.8623 | 3.0243 | 0.3839 |
| 3.5572 | 3.1438 | 3.2543 | 2.4803 | 3.2951 | 1.5268 | 3.4313 | 0.9253 | 3.5130 | 0.4145 |
| 3.6790 | 3.1551 | 3.6185 | 2.5766 | 3.6798 | 1.6034 | 3.8025 | 0.9662 | 3.8945 | 0.4370 |
| 4.3900 | 3.2730 | 3.7590 | 2.6024 | 3.8230 | 1.6198 | 3.9350 | 0.9826 | 4.0469 | 0.4474 |
| 4.8735 | 3.3012 | 4.5065 | 2.7204 | 4.6036 | 1.7199 | 4.7008 | 1.0526 | 4.7785 | 0.4737 |
| 5.1615 | 3.3228 | 5.0040 | 2.7979 | 5.0910 | 1.7857 | 5.2216 | 1.0982 | 5.2651 | 0.5066 |
| | | 5.2772 | 2.8130 | 5.3930 | 1.8120 | 5.5087 | 1.1194 | 5.5550 | 0.5202 |

Figure 7: Experiment Data for Problem 1[1]

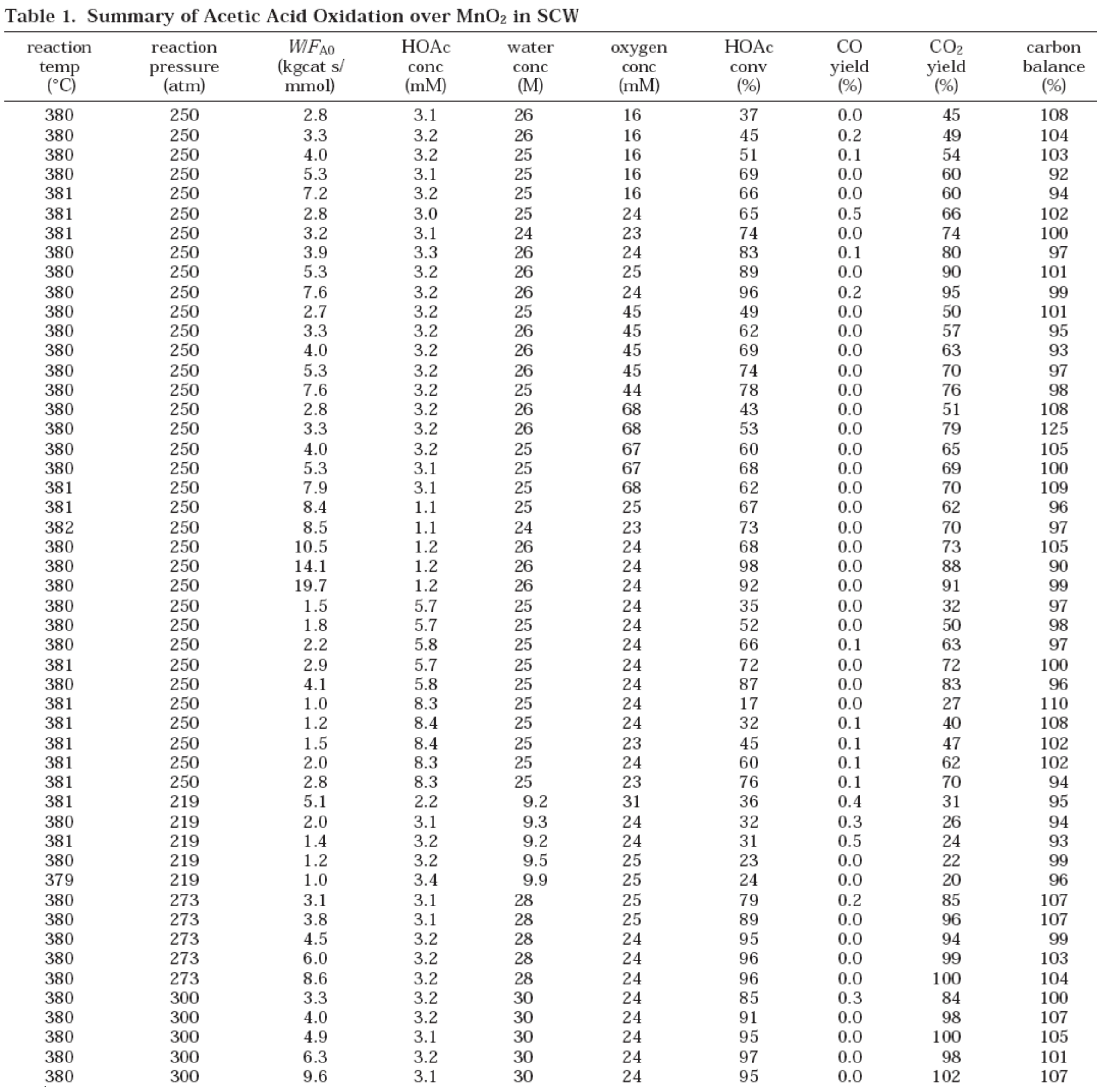

**Table 1. Summary of Acetic Acid Oxidation over $MnO_2$ in SCW**

| reaction temp (°C) | reaction pressure (atm) | $W/F_{A0}$ (kgcat s/ mmol) | HOAc conc (mM) | water conc (M) | oxygen conc (mM) | HOAc conv (%) | CO yield (%) | $CO_2$ yield (%) | carbon balance (%) |
|---|---|---|---|---|---|---|---|---|---|
| 380 | 250 | 2.8 | 3.1 | 26 | 16 | 37 | 0.0 | 45 | 108 |
| 380 | 250 | 3.3 | 3.2 | 26 | 16 | 45 | 0.2 | 49 | 104 |
| 380 | 250 | 4.0 | 3.2 | 25 | 16 | 51 | 0.1 | 54 | 103 |
| 380 | 250 | 5.3 | 3.1 | 25 | 16 | 69 | 0.0 | 60 | 92 |
| 381 | 250 | 7.2 | 3.2 | 25 | 16 | 66 | 0.0 | 60 | 94 |
| 381 | 250 | 2.8 | 3.0 | 25 | 24 | 65 | 0.5 | 66 | 102 |
| 381 | 250 | 3.2 | 3.1 | 24 | 23 | 74 | 0.0 | 74 | 100 |
| 380 | 250 | 3.9 | 3.3 | 26 | 24 | 83 | 0.1 | 80 | 97 |
| 380 | 250 | 5.3 | 3.2 | 26 | 25 | 89 | 0.0 | 90 | 101 |
| 380 | 250 | 7.6 | 3.2 | 26 | 24 | 96 | 0.2 | 95 | 99 |
| 380 | 250 | 2.7 | 3.2 | 25 | 45 | 49 | 0.0 | 50 | 101 |
| 380 | 250 | 3.3 | 3.2 | 26 | 45 | 62 | 0.0 | 57 | 95 |
| 380 | 250 | 4.0 | 3.2 | 26 | 45 | 69 | 0.0 | 63 | 93 |
| 380 | 250 | 5.3 | 3.2 | 26 | 45 | 74 | 0.0 | 70 | 97 |
| 380 | 250 | 7.6 | 3.2 | 25 | 44 | 78 | 0.0 | 76 | 98 |
| 380 | 250 | 2.8 | 3.2 | 26 | 68 | 43 | 0.0 | 51 | 108 |
| 380 | 250 | 3.3 | 3.2 | 26 | 68 | 53 | 0.0 | 79 | 125 |
| 380 | 250 | 4.0 | 3.2 | 25 | 67 | 60 | 0.0 | 65 | 105 |
| 380 | 250 | 5.3 | 3.1 | 25 | 67 | 68 | 0.0 | 69 | 100 |
| 381 | 250 | 7.9 | 3.1 | 25 | 68 | 62 | 0.0 | 70 | 109 |
| 381 | 250 | 8.4 | 1.1 | 25 | 25 | 67 | 0.0 | 62 | 96 |
| 382 | 250 | 8.5 | 1.1 | 24 | 23 | 73 | 0.0 | 70 | 97 |
| 380 | 250 | 10.5 | 1.2 | 26 | 24 | 68 | 0.0 | 73 | 105 |
| 380 | 250 | 14.1 | 1.2 | 26 | 24 | 98 | 0.0 | 88 | 90 |
| 380 | 250 | 19.7 | 1.2 | 26 | 24 | 92 | 0.0 | 91 | 99 |
| 380 | 250 | 1.5 | 5.7 | 25 | 24 | 35 | 0.0 | 32 | 97 |
| 380 | 250 | 1.8 | 5.7 | 25 | 24 | 52 | 0.0 | 50 | 98 |
| 380 | 250 | 2.2 | 5.8 | 25 | 24 | 66 | 0.1 | 63 | 97 |
| 381 | 250 | 2.9 | 5.7 | 25 | 24 | 72 | 0.0 | 72 | 100 |
| 380 | 250 | 4.1 | 5.8 | 25 | 24 | 87 | 0.0 | 83 | 96 |
| 381 | 250 | 1.0 | 8.3 | 25 | 24 | 17 | 0.0 | 27 | 110 |
| 381 | 250 | 1.2 | 8.4 | 25 | 24 | 32 | 0.1 | 40 | 108 |
| 381 | 250 | 1.5 | 8.4 | 25 | 23 | 45 | 0.1 | 47 | 102 |
| 381 | 250 | 2.0 | 8.3 | 25 | 24 | 60 | 0.1 | 62 | 102 |
| 381 | 250 | 2.8 | 8.3 | 25 | 23 | 76 | 0.1 | 70 | 94 |
| 381 | 219 | 5.1 | 2.2 | 9.2 | 31 | 36 | 0.4 | 31 | 95 |
| 380 | 219 | 2.0 | 3.1 | 9.3 | 24 | 32 | 0.3 | 26 | 94 |
| 381 | 219 | 1.4 | 3.2 | 9.2 | 24 | 31 | 0.5 | 24 | 93 |
| 380 | 219 | 1.2 | 3.2 | 9.5 | 25 | 23 | 0.0 | 22 | 99 |
| 379 | 219 | 1.0 | 3.4 | 9.9 | 25 | 24 | 0.0 | 20 | 96 |
| 380 | 273 | 3.1 | 3.1 | 28 | 25 | 79 | 0.2 | 85 | 107 |
| 380 | 273 | 3.8 | 3.1 | 28 | 25 | 89 | 0.0 | 96 | 107 |
| 380 | 273 | 4.5 | 3.2 | 28 | 24 | 95 | 0.0 | 94 | 99 |
| 380 | 273 | 6.0 | 3.2 | 28 | 24 | 96 | 0.0 | 99 | 103 |
| 380 | 273 | 8.6 | 3.2 | 28 | 24 | 96 | 0.0 | 100 | 104 |
| 380 | 300 | 3.3 | 3.2 | 30 | 24 | 85 | 0.3 | 84 | 100 |
| 380 | 300 | 4.0 | 3.2 | 30 | 24 | 91 | 0.0 | 98 | 107 |
| 380 | 300 | 4.9 | 3.1 | 30 | 24 | 95 | 0.0 | 100 | 105 |
| 380 | 300 | 6.3 | 3.2 | 30 | 24 | 97 | 0.0 | 98 | 101 |
| 380 | 300 | 9.6 | 3.1 | 30 | 24 | 95 | 0.0 | 102 | 107 |

Figure 8: Experiment Data for Problem 2[2]