Multi-Component Distillation Column Simulator: CLL261 Project Report

Aditya Vishwakarma (2018CH10191)

This report outlines the development and implementation of a C++ simulation of a multi-component distillation column. Through this project, I have gained a deeper understanding of the behavior and performance of distillation processes, particularly the application of material and heat balances, equilibrium relations, and numerical methods to solve complex systems of equations.

1. Introduction

Distillation is a fundamental separation process in chemical engineering. It separates components of a liquid mixture based on their relative volatilities or boiling points. In this project, I developed a simulation of a multi-component distillation column using C++. The primary focus was to simulate the column based on theoretical principles and numerical methods, providing a detailed understanding of the behavior of the column under different conditions.

2. Theoretical Background

2.1 MESH Equations

At the core of my simulation are the MESH (Material balance, Equilibrium, Summation, and Heat balance) equations. These equations ensure that mass and energy are conserved across each stage of the distillation column.

- **Material Balance (M):** For each component *i* on stage *j*, the material balance equation is:

  F*z_i + L_(j+1)*x_(i,j+1)* + *V_(j+1)*y_(i,j+1)* = *L_j*x_(i,j)* + *V_j*y_(i,j)

  Here, *F* represents the feed flow rate, *L* and *V* are the liquid and vapor flow rates, and *x* and *y* are the liquid and vapor compositions, respectively. Each of these parameters is calculated for each stage *j*. The liquid comes from the stage above (*L_(j+1)*) and vapor comes from the stage below (*V_(j+1)*).

- **Equilibrium (E):** The equilibrium relation for each component *i* on stage *j* is given by:

  y_(i,j) = K_(i,j) * x_(i,j)

  This equation represents the vapor-liquid equilibrium, where *K_(i,j)* is the equilibrium constant (K-value) at stage *j* for component *i*. The

equilibrium constant is calculated based on the chosen vapor-liquid equilibrium (VLE) model.

- **Summation (S):** The mole fractions in both the liquid and vapor phases on each stage must sum to unity:

  Σ x_(i,j) = 1.0
  Σ y_(i,j) = 1.0

- **Heat Balance (H):** The heat balance equation for each stage *j* is:

  F*h_F + L_(j+1)*h_(L,j+1)* + *V_(j+1)h_(V,j+1)* = *L_jh_(L,j)* + *V_j*h_(V,j) + Q_j

  Where *h_F* is the feed enthalpy, *h_(L,j+1)* and *h_(V,j+1)* are the liquid and vapor enthalpies from the stages above and below, respectively, and *Q_j* represents the heat added or removed at stage *j*.

2.2 Vapor-Liquid Equilibrium (VLE) Models

Accurate VLE calculations are essential for simulating the distillation process. I considered two models in my simulation:

- **Raoult's Law**: For ideal mixtures, I used Raoult's Law to calculate the equilibrium constant *K_i*:

  K_i = P_i^sat / P

  Where *P_i^sat* is the saturation pressure of component *i* and *P* is the total pressure.

- **NRTL/Wilson Models**: For non-ideal mixtures, I explored the NRTL and Wilson models, which are more complex but provide more accurate results for real-world systems.

2.3 Data Structures

In my implementation, I designed several C++ classes to represent the components, stages, and the overall distillation column.

- **Component Class**: This class stores the properties of each component in the system, including molecular weight and Antoine coefficients (used to calculate saturation pressure via the Antoine equation). An example of the class structure I used is:

```
class Component {
public:
    std::string name;
    double molecularWeight;
    double A; // Antoine coefficient A
    double B; // Antoine coefficient B
```

```
    double C; // Antoine coefficient C
    // Antoine equation: log(P_sat) = A - B / (T + C)
};
```

- **Stage Class**: Each stage in the column is represented by the Stage class, which contains information about the liquid and vapor compositions, temperature, pressure, and flow rates:
```
class Stage {
public:
    int stageNumber;
    std::vector<double> x; // Liquid composition
    std::vector<double> y; // Vapor composition
    double T; // Temperature
    double P; // Pressure
    double L; // Liquid flow rate
    double V; // Vapor flow rate
};
```

- **Column Class**: The Column class ties together the stages and components, as well as key operational parameters such as the reflux ratio, reboiler duty, and feed conditions:
```
class Column {
public:
    int numberOfStages;
    int feedStage;
    double feedFlowRate;
    std::vector<Component> components;
    std::vector<Stage> stages;
    double refluxRatio;
    double reboilerDuty;
};
```

2.4 Boundary Conditions

At the top and bottom of the column, I defined the following boundary conditions:

- **Reflux Ratio**: The ratio of liquid reflux to distillate flow rate.
- **Reboiler Duty**: The heat supplied to the reboiler.
- **Feed Conditions**: The stage where the feed is introduced, along with its composition, temperature, and flow rate.

3. Coding Implementation

3.1 VLE Solver

I implemented the VLE solver to calculate the K-values at each stage based on the chosen VLE model. This was done within the Stage class or as a separate

class to handle equilibrium calculations.

3.2 Numerical Methods

The MESH equations form a non-linear system that requires numerical methods to solve. I primarily used the Newton-Raphson method, which requires the calculation of the Jacobian matrix.

- **Newton-Raphson Method**: I approximated the Jacobian matrix using finite differences. This method is powerful and commonly used for solving non-linear systems, but it becomes computationally expensive as the number of stages and components increases, due to the size of the Jacobian.
- **Broyden's Method**: I also explored Broyden's method, which is a quasi-Newton method that approximates the Jacobian. This method can be more computationally efficient, although I did not fully implement it in the current project.

3.3 Numerical Solver Implementation

The solver iterates over the system of equations until the error is below a specified tolerance. Here is a simplified version of how I structured the Newton-Raphson iteration:

```
std::vector<double> variables; // Unknowns (x, y, T, L, V...)
int iteration = 0;
while (error > tolerance && iteration < maxIterations) {
    // Calculate residuals, Jacobian, and update variables
    error = calculateError(variables);
    iteration++;
}
if (iteration == maxIterations) {
    // Handle non-convergence
}
```

For larger simulations, I explored the use of sparse matrices to improve computational efficiency. Libraries like **Eigen** or **Armadillo** can significantly reduce the memory usage and computation time when handling large systems.

4. Visualization

Although visualization was not a core requirement, I found it helpful to visualize the concentration and temperature profiles during the simulation. This allowed me to better understand how the system was behaving and aided in debugging. I used **Matplotlib-cpp** for real-time plotting during the simulation, but I also exported the results for further analysis in Python or MATLAB.

5. Testing and Validation

To verify the accuracy of my simulation, I compared the results to analytical solutions for simple cases, such as binary distillation, and referenced published data for more complex systems. This process helped me validate the implementation and ensure that the numerical methods converged to the correct solution.

6. Further Enhancements

While the current simulator is functional, there are several enhancements I would like to explore in the future:

- More complex VLE models, such as the NRTL or Wilson models, could be implemented in greater detail.
- Broyden's method could be implemented fully to improve computational efficiency.
- A graphical user interface (GUI) could make the simulator more user-friendly.
- The inclusion of tray efficiencies and pressure drop calculations would provide a more accurate representation of a real distillation column.

Conclusion

This project has allowed me to gain a strong understanding of multi-component distillation and the numerical methods required to simulate it. By implementing the MESH equations, applying VLE models, and solving the system using iterative methods, I was able to model the behavior of a distillation column. Testing and validation confirmed the accuracy of the simulation, and I look forward to further enhancing the project in the future.