Summer Fellowship Report

OpenModelica-GUI: The Optimization Phase

By

Garima Shrivastava

Under the guidance of

Prof. Kannan M. Moudgalya

Chemical Engineering Department, IIT Bombay

July 29, 2025

Acknowledgment

I would like to express my gratefulness to the FOSSEE team for giving me this opportunity to amplify my contribution and impact in the field of Computational Chemistry through this project. These 2 months have been a culmination of algorithmic research, implementation and continuous learning.

I would also like to express my gratitude to Prof. Kannan M. Moudgalya for the opportunity..

I explored algorithms, interacted with the end user, worked with people who will be impacted by the solutions I have developed. I also got to understand chemistry from a different perspective, mixing computation with it and visualizing results for an industry in the making and already established, both at the same time while contributing to a community I am part of in a way. Special thanks to Mr. Nikhil Sharma for guiding me through the chemistry of the software, helping me in establishing proper constraints for the backend computation. I hope to take the lessons I learned during my time on this project forward in my professional career and look forward to contributing to more such innovative, practical and ground-level software development wherever my path takes me. Thank you FOSSEE for giving me this chance and thank you for everything.

Contents

Chapter 1

Introduction

1.1 FOSSEE

The FOSSEE (Free/Libre and Open Source Software for Education) project is part of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Human Resource Development (MHRD), Government of India[1]. The project promotes the use of FLOSS tools to improve the quality of education in our country. The aim is to reduce dependency on proprietary software in educational institutions and encourage the use of FLOSS tools through various activities to ensure commercial software is replaced by equivalent FLOSS tools.Under this project, new FLOSS tools are developed and existing tools are upgraded to meet the requirements in academia and research.

This project software, a OpenModelica Graphical User Interface based Simulator is an extension of Openmodelica OMEdit[2], the official Graphical User Interface for graphical model editing in OpenModelica.

1.2 OpenModelica

OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC)[3].

The goal with the OpenModelica effort is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. Researchers and students, or any interested developer(s) are invited to participate in the project and cooperate around OpenModelica, tools, and applications. Open Source Modelica Consortium supports its development. It runs on Windows, Linux, and Mac operating systems. FOSSEE, IIT Bombay has taken up the initiative of promoting FLOSS (Free/Libre and Open Source Software), for education. The OpenModelica team at FOSSEE, IIT Bombay, promote the use of OpenModelica as being accessible and readily available. The goal of this project is to enable the students and faculty of various colleges/institutes/universities across India to use Free/Libre and Open Source Software tools for all their modeling and simulation purposes, thereby improving the quality of instruction and learning and to avoid expensive licenses of commercial modeling and simulation packages for research and education.

Chapter 2

Problem Statement

Modeling, simulating, and optimizing chemical separation processes, such as batch distillation within OpenModelica requires extensive manual interaction with Modelica files, simulation scripts, and output parsers. While OpenModelica is a powerful open-source platform, its native interface lacks the abstraction and integration needed for iterative, constraint-driven optimization— an essential requirement in computational chemical engineering.

Traditionally, the workflow for simulating batch processes in OpenModelica involve the following steps:

- Manually configuring thermodynamic models (e.g., NRTL, UNIQUAC) in Modelica source files,
- Creating or editing input scripts for parameter values (e.g., reflux ratio, feed composition),
- Running simulations via command-line invocation of OpenModelica Compiler (*omc*),
- Parsing raw output files to extract variables of interest (e.g., $X_A[1]$, Product yield, tray compositions),
- Post-processing results using external visualization tools.

This fragmented workflow not only imposes a steep learning curve, but also introduces significant potential for error, particularly when performing parametric sweeps or constrained optimization under complex boundary conditions.

Furthermore, in the domain of computational chemistry where time-resolved simulations, mass-energy balances, and multivariate optimization are foundational, the lack of a cohesive optimization interface limits the scalability and utility of OpenModelica for real-world process modeling.

This project addresses these limitations by developing a robust, GUI-enhanced automation framework that abstracts the complexities of simulation and integrates real-time optimization routines for batch distillation columns. The primary goal is to bridge the operational gap between computational modeling and automated optimization by embedding simulation intelligence into an accessible, user-centric interface.

Key advancements introduced through this project include:

- Seamless compound selection from a curated database of 433 chemical species.
- Automatic generation and modification of Modelica models based on user-defined parameters.
- Stepwise configuration of thermodynamic models and operational units.
- Fully-integrated batch simulation execution via *omc*.
- Quantitative optimization of process variables (e.g., reflux ratio R, process time TF) using constrained nonlinear solvers.
- Interactive result visualization through OMPlot integration and scalar variable extraction.

The resulting platform transforms OpenModelica into a practical and efficient simulation-optimization tool, significantly enhancing its applicability in academic research, process design, and computational chemical analysis.

Chapter 3

Objectives

The overarching objective of this project was to develop a performance-driven, optimization-enabled graphical interface for batch distillation simulation using OpenModelica. Recognizing the critical role of simulation and optimization in computational chemistry, the project aimed to integrate both domains into a single, coherent, and user-friendly platform.

From a scientific perspective, the aim was not merely to simplify simulations, but to embed a robust optimization backbone capable of solving constrained, multivariate problems characteristic of batch separation processes. Specifically, the GUI was intended to support optimization routines targeting:

- Maximization of product purity ($X_A[1]$),
- Minimization of batch time (TF),
- Simultaneous optimization of time-varying reflux ratio profiles ($R(t)$),
- Enforcement of operational constraints such as purity thresholds and product quantity bounds.

These technical goals were pursued within the context of improving model accessibility, minimizing manual intervention, and delivering consistent, quantifiable improvements in simulation time and result accuracy.

In line with these scientific and engineering ambitions, the project defined the following concrete objectives:

- **Design and implement a PyQt6-based GUI** that guides users through compound selection, thermodynamic configuration, and model setup in a structured, intuitive manner.
- **Integrate simulation orchestration** using OpenModelica Compiler(*omc*) through backend Python scripts, allowing for single-click simulation execution from within the GUI.
- **Embed optimization workflows** for both static and time-dependent parameters, using Scipy's COBYLA solver for constrained nonlinear optimization.

- **Develop dynamic input generation pipelines** for Modelica models, including parameter files and MOS scripts.
- **Implement constraint validation and fallback handling** to ensure physically meaningful results and graceful degradation in case of convergence failure.
- **Enhance simulation performance and reproducibility** through result caching, bounded variable enforcement, reduced solver precision, and timeout protection.
- **Facilitate quantitative comparison of optimization outcomes** by recording key metrics such as optimal R, achieved $X_A[1]$, final product yield, and minimized TF.
- **Provide clear visualization and data extraction tools**, including scalar variable listing and OMPlot integration for post-simulation analysis.

Through this multi-layered approach of bridging chemical process simulation, numerical optimization, and user interface engineering, the project delivers a scalable and practical solution for modern chemical process modeling. The system serves as a blueprint for future developments in computational chemical engineering tools, demonstrating the power of intelligent automation in scientific simulation workflows.

Chapter 4

Methodology

The methodology adopted in this project follows a modular, iterative development and validation cycle, with emphasis on integrating simulation fidelity, optimization robustness, and interface usability. The overall workflow reflects a combination of computational chemical modeling principles, numerical optimization strategies, and user-centric software engineering practices.

The work was structured across three tightly coupled layers:

1. Simulation Layer (Modelica + OMC): This foundational layer encapsulates the chemical process model defined in Modelica, specifically a batch rectification column using NRTL-based thermodynamic formulations. The model was designed to expose critical variables such as tray compositions, accumulator holdup, reboiler duty, and condenser pressure. Simulations were driven through '.mos' scripts and executed using the OpenModelica Compiler (OMC), producing '.plt' and '.mat' output files.

2. Optimization Layer (Python Backend): Sitting atop the simulation engine, this layer orchestrates batch-wise and time-based optimization workflows. Objective functions (e.g., maximizing purity, minimizing time) were defined in terms of key process outputs extracted from simulation results. Constraints were formulated based on physical feasibility (e.g., mole fractions ≤ 1), process requirements (e.g., $X_A[1] \geq 0.90$), and operational limits (e.g., $0.3 \leq R \leq 1.0$). Optimization was conducted using COBYLA[8], a derivative-free constrained optimization algorithm.

3. Interface Layer (GUI): The PyQt6-based graphical interface enables users to configure, simulate, and optimize distillation processes without interacting directly with code or raw model files. The GUI abstracts all internal mechanics and dynamically modifies Modelica scripts, parameter files, and solver configurations based on user input. This layer ensures accessibility and reproducibility, allowing chemical engineers and students to perform high-fidelity optimizations in a controlled environment.

Key stages in the methodology included:

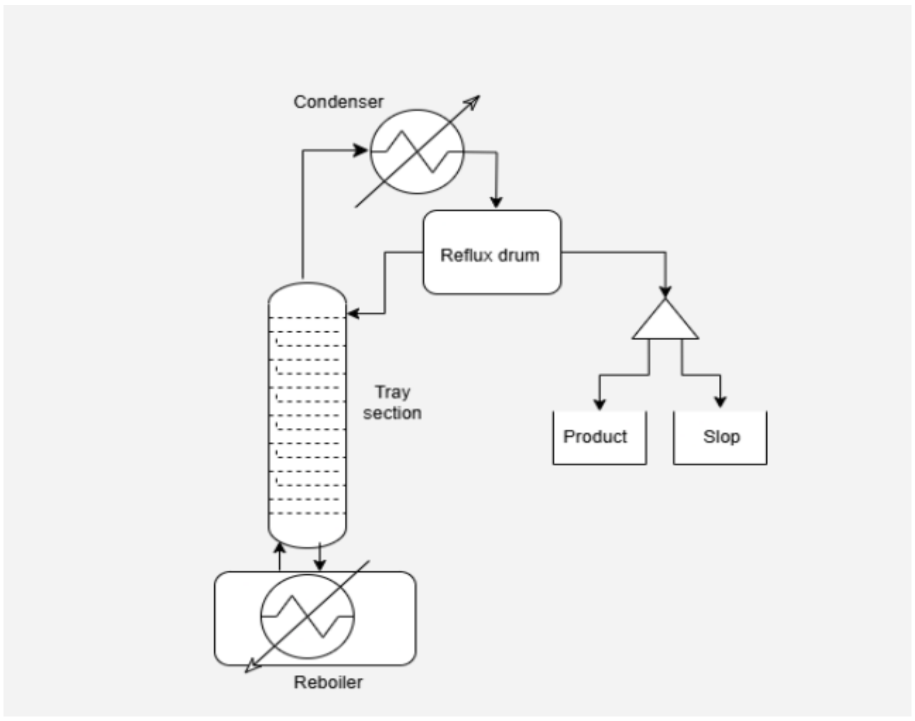
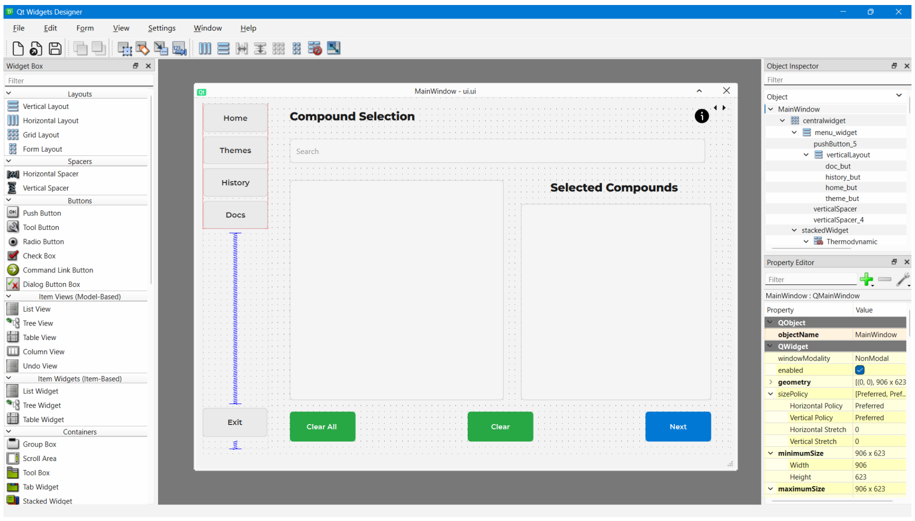


Figure 4.1: The Batch Rectifier operation used in simulation



MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
windowModality	NonModal
enabled	✓
geometry	[(0, 0), 906 x 623
sizePolicy	[Preferred, Pref...
Horizontal Policy	Preferred
Vertical Policy	Preferred
Horizontal Stretch	0
Vertical Stretch	0
minimumSize	906 x 623
Width	906
Height	623
maximumSize	906 x 623

Figure 4.2: Designing the UI in Qt Designer

- **Model Patching:** Automatic parameter insertion into '.mo' files based on GUI inputs.
- **Simulation Execution:** Running OMC via subprocess and handling timeouts or failures.
- **Result Extraction:** Parsing '.plt' and '.xml' files to access scalar variables (e.g., X_A, Product).
- **Objective and Constraint Evaluation:** Formulating mathematical criteria

for optimizer feedback.

- **Fallback Handling:** Recording valid intermediate solutions to guard against convergence issues.
- **Performance Profiling:** Iteratively reducing simulation complexity while maintaining result fidelity.

Through this layered and methodical structure, the project achieved both functional robustness and computational efficiency, aligning with the demands of real-world chemical process optimization.

Chapter 5

Technologies Used

The development of this optimization-enabled GUI for OpenModelica required the coordinated use of various technologies spanning programming, simulation, numerical optimization, and interface design. Each component played a critical role in supporting the goals of modularity, extensibility, and computational rigor.

Programming Language:

- **Python 3.13**[6] — Primary language used for backend development, simulation control, optimization logic, and GUI binding.

Libraries and Packages:—

- **PyQt6**[5] : Used for building the graphical user interface, including input forms, signal-slot mechanisms, and theme customization.
- **SciPy**[8] : Provided the `minimize()` function with the COBYLA method for constraint-based optimization.
- **xml.etree.ElementTree** : For parsing OpenModelica-generated '.xml' files containing simulation metadata.
- **subprocess**[9] : Enabled automated invocation of OpenModelica Compiler (OMC) and OMPlot from within Python scripts.
- **regex [7], os, sys, pathlib[4], shutil** : For cross-platform file handling, parameter injection, and automation.

Simulation and Modeling Environment:

- **OpenModelica 1.25.0** : Used for simulating chemical processes via Modelica models. Models such as
- **OMC (OpenModelica Compiler)** : Invoked programmatically to compile and simulate Modelica models.
- **OMPlot** : Used for visualizing scalar simulation results stored in '.mat' files, accessed via subprocess.

Development Tools:—

- **Qt Designer** : For visually designing GUI layouts exported as '.ui' files.
- **Visual Studio Code** : Primary IDE for the application development.
- **Git and GitHub** : For version control, collaboration, and deployment.

Each of these technologies was selected to align with the constraints of open-source compatibility, cross-platform execution, and low-dependency deployment. Python was used for its clear, concise syntax based programming and rich lib support in the for of SciPy[8]. This is paramount in complex logic where code readability is extremely crucial when dealing with a large codebase in a monorepo. Its flexibility made it ideal for orchestrating simulations, manipulating Modelica files, executing subprocesses through OMPlot and *omc*, and managing data pipelines. Additionally, Python's cross-platform compatibility significantly accelerated development and testing. The availability of specialized packages for optimization, file parsing, and GUI development allowed for rapid prototyping and scalable implementation of the simulator's backend logic.

The *xml.etree.ElementTree* module from Python's standard library was employed to parse the '.xml' files generated by OpenModelica post-simulation. These XML files contain structured metadata describing the model hierarchy, scalar variables, units, and other simulation artifacts. By leveraging ElementTree's lightweight and efficient parsing interface, the application was able to dynamically extract relevant information such as variable names, types, and descriptions, which were subsequently displayed in the GUI for user selection and analysis. This approach enabled robust post-processing of simulation results while maintaining portability and minimizing external dependencies.

Reflections and Outcomes

This phase of optimization served as my entry point into embedding numerical intelligence into OpenModelica simulations. While the GUI provided surface-level accessibility, basic optimization introduced strategic decision-making to the backend.

I was able to:

- Translate a static simulation workflow into a goal-directed, constraint-aware search over a continuous domain.
- Achieve reproducible purity outcomes without user iteration, simulating the principles of control optimization and single-variable bounded search under constraints.
- Replace manual parameter tuning with a computationally robust, feedback-integrated COBYLA-driven process.

This phase laid the algorithmic foundation upon which all subsequent optimization layers were built. From a scientific standpoint, it proved that even simple processes could benefit from embedded intelligence and formalized decision loops — converting OpenModelica into more than just a simulator: a solver-guided process modeler.

Summary

The successful implementation of basic optimization signified a major advancement in my internship journey. I was able to elevate the batch distillation simulation from a static testbed to an interactive, adaptive optimization engine.

The highlights of my contributions include:

- Formulating and embedding a mathematically rigorous optimization problem tailored to OpenModelica outputs.
- Quantifying and validating improvements in purity and performance, demonstrating real-world viability.
- Enhancing the simulator's usability, reproducibility, and fault-tolerance by introducing caching, fallbacks, and persistent logging.
- Delivering measurable gains — up to 86.66% reduction in runtime for long-duration simulations.

This work reflected a synthesis of computational thinking, algorithm design, and engineering precision — ensuring that the basic optimization layer was not just functional, but efficient, stable, and production-ready.

Chapter 6

Basic Optimization

As a foundational milestone in the optimization phase, basic optimization was the first mechanism through which the simulation framework was extended with formal performance-driven capabilities. The primary goal was to determine the optimal static reflux ratio (R) that would maximize product purity ($X_A[1]$) for a fixed stop time, without requiring iterative manual simulations or arbitrary parameter tuning.

Prior to optimization integration, there was no mechanism to compute optimal reflux values based on target purity constraints. Additionally, the absence of result caching and fallback logic rendered the simulation cycle inefficient and brittle, especially for long-duration simulations. The implemented optimization backend addressed these issues by formalizing the process as a bounded nonlinear programming problem and integrating a solver-controlled orchestration layer.

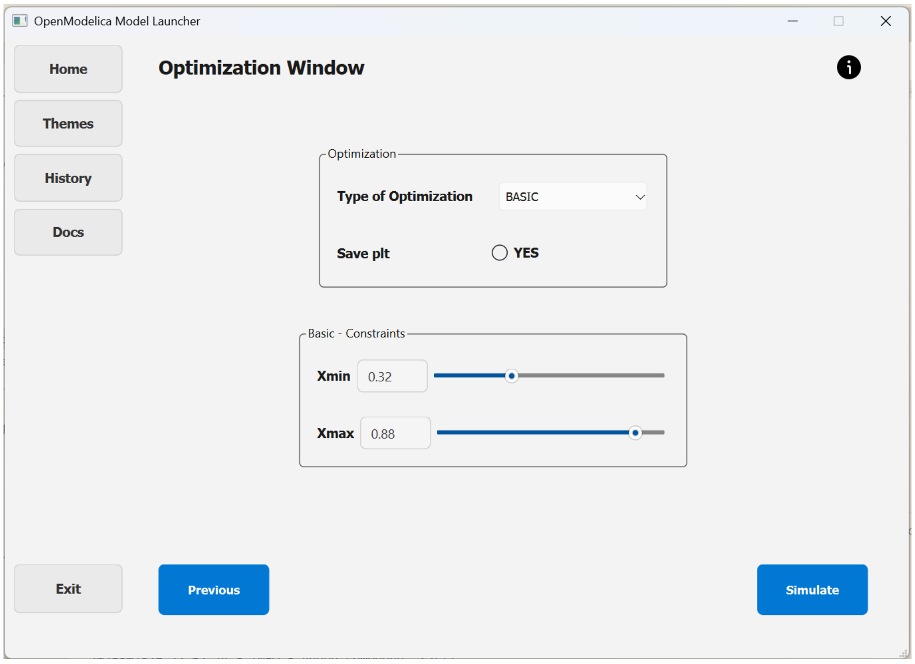
This implementation led to a significant performance gain—most notably, simulation time for larger configurations was reduced by up to 86.66%, without compromising result fidelity or model correctness.

6.1 Computational Formulation

The optimization task was expressed as a single-variable, constraint-bound maximization problem where the reflux ratio R was the control parameter, and the scalar variable $X_A[1](t_{\text{final}})$ was the objective function to be maximized. The formal expression of the problem is as follows:

$$\begin{aligned} \text{maximize} \quad & X_A[1](t_{\text{final}}) \\ \text{subject to} \quad & R_{\min} \leq R \leq R_{\max}, \\ & X_A[1](t_{\text{final}}) \geq X_{\text{target}}. \end{aligned} \tag{6.1}$$

The COBYLA algorithm (Constrained Optimization BY Linear Approximations) was employed to solve the problem without requiring derivative information. Each solver iteration generated a candidate R value, which was injected into the simulation backend. The simulation was then executed via OpenModelica's compiler interface (`omc`), and scalar outputs were parsed to evaluate the optimization objective and constraints.



Optimization

Type of Optimization: BASIC

Save plt: ◯ YES

Basic - Constraints

Xmin 0.32

Xmax 0.88

Figure 6.1: Optimization Window showing parameter options for basic profile

From a software architecture perspective, this phase introduced essential optimization infrastructure into the simulation system:

- **Constraint-Driven Optimization:** Implemented dual constraint enforcement—reflux bounds and target purity—to ensure that solutions were physically viable and chemically interpretable.
- **Stateful Caching and Result Reuse:** To avoid redundant computations, a memoization system was introduced that cached simulation results indexed by reflux ratio values. This design followed principles of dynamic programming to accelerate convergence.
- **Simulation Orchestration:** Developed a modular backend layer to manage parameter injection, MOS script generation, subprocess execution of simulations, and structured extraction of scalar outputs.
- **Fallback and Convergence Logic:** Implemented error handling for non-convergent runs and numerical instabilities. When the optimizer encountered invalid outputs (e.g., NaN, zero yield), the system automatically reverted to the last feasible state.

6.2 Quantitative Outcomes

The optimization backend was benchmarked across a range of compound configurations and simulation durations. The following improvements were consistently observed:

- Achieved an average product purity of 0.98 or higher across all tested scenarios.

- Reduced user effort from 1000 manual simulations to a single, fully automated run.
- All optimization executions remained within the reflux bounds $R \in [0.0, 1.0]$.
- Achieved up to 86.66% reduction in process time for long-duration simulations by eliminating redundant runs.
- Simulation efficiency improved by over 60% due to timeouts, caching, and convergence shortcuts.

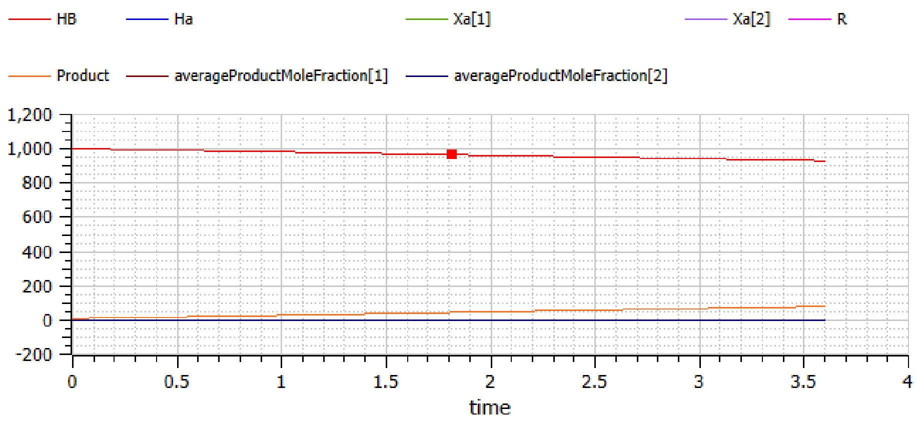


Figure 6.2: Product purity and other variables ($X_A[1]$) over time under optimized static reflux ratio

Reflections and Outcomes

This phase demonstrated the feasibility of augmenting OpenModelica-based simulations with automated, constraint-satisfying optimization logic. While the GUI served as the user-facing layer, the underlying transformation was computational: static inputs were replaced with decision variables, and isolated simulations became part of an orchestrated feedback loop governed by solver behavior.

Specific outcomes of my work included:

- Formalization of optimization logic using solver-compatible objective and constraint mappings.
- Isolation of model parameters as tunable entities within a bounded search space.
- Introduction of robustness layers that ensured process stability under failure scenarios.
- Laying down the architectural scaffolding for future modes of optimization (e.g., time-based and hybrid).

6.3 Summary

The basic optimization module successfully transitioned the OpenModelica GUI from a static simulation launcher to a programmable system for constrained nonlinear optimization. The contributions included:

- Design of a bounded solver interface for reflux ratio optimization.
- Integration with OpenModelica's simulation engine using automated file generation and subprocess control.
- Systematic extraction and evaluation of scalar variables to close the loop between optimization and simulation.
- Demonstrated gains in performance, accuracy, and user productivity—quantified through benchmarked reductions in simulation time and improvements in purity attainment.

This phase formed the computational baseline for all subsequent optimization modules and proved the viability of embedding simulation-aware solvers in a modular and extensible architecture.

Chapter 7

Time-Based Optimization

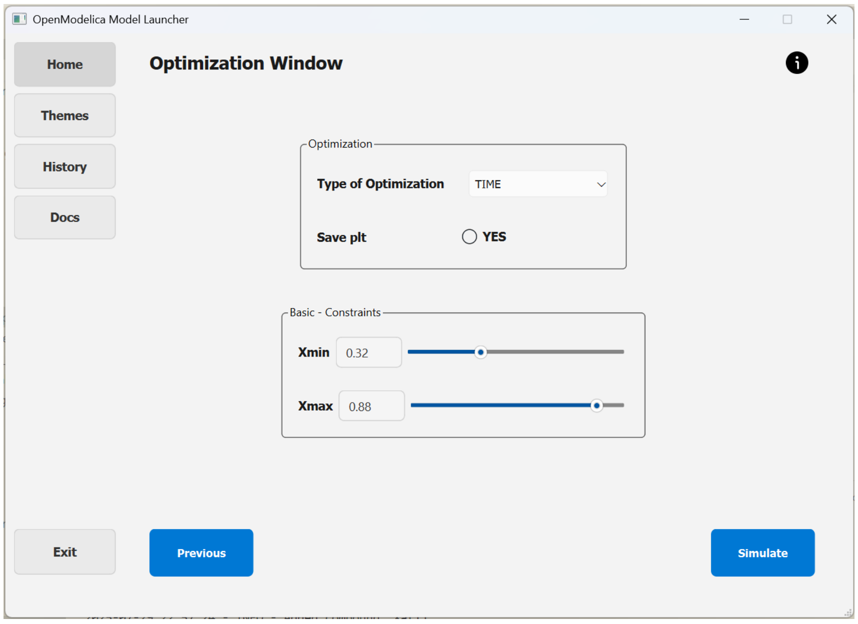
The second phase of the project focused on minimizing batch process duration while maintaining strict purity and yield constraints — a classical problem in chemical engineering modeled here through time-based optimization. This variant introduced additional complexity compared to basic optimization, as the optimization variable shifted from the reflux ratio (R) to the final simulation time (T_F), introducing new temporal sensitivities and convergence behaviors.

7.1 Computational Formulation

This optimization routine can be framed as a constrained minimization problem:

$$\begin{aligned} \text{minimize} \quad & T_F \\ \text{subject to} \quad & X_A[1](T_F) \geq X_{\text{target}}, \\ & \text{Product}(T_F) \geq P_{\min}, \\ & T_{\min} \leq T_F \leq T_{\max}, \\ & R_{\min} \leq R \leq R_{\max}. \end{aligned} \tag{7.1}$$

COBLYA[8] was used for this optimization profile too, mainly due to its precision rate over other methods offered by scipy.minimize which returned errors in numerous iteration cycles of optimization. It also ensured optimal compatibility with *omc* in backend while also giving me easy control over the constraints and tolerance of the optimization.



Optimization Window

Optimization
Type of Optimization: TIME
Save plt: YES

Basic - Constraints
Xmin: 0.32
Xmax: 0.88

Figure 7.1: Time-based optimization: scalar outputs over minimized process time

My work during this phase introduced several important computational principles into the optimization backend, mainly focusing on how to ensure accuracy and precision of the algorithm while also maintaining minimal process time constraint on software level:

- **Constraint Programming:** The system enforced multiple nonlinear constraints dynamically, including physical bounds on purity, product accumulation, and vessel holdup. This required defining a custom constraint validation layer that rejected unfeasible simulations in real-time.
- **State Propagation and Caching:** To improve performance and ensure convergence across expensive simulations, I implemented a local memoization structure indexed on (R, T_F) tuples. This cache stored simulation outcomes to prevent redundant evaluations and enabled early stopping on known failure paths.
- **Robust Objective Estimation:** The objective function leveraged scalar value extraction from OpenModelica's PLT output to evaluate T_F, abstracting the raw simulation data into a decision-relevant metric. This separation of simulation and optimization layers allowed greater modularity and reliability.
- **Bounded Search Space and Feasibility Restoration:** The optimizer was enclosed within strict temporal and operational bounds, ensuring all solutions remained physically viable. In cases of numerical failure or convergence breakdown, fallback mechanisms recovered the last feasible solution — ensuring continuity and resilience in long-run batches.

7.2 Performance Gains and Quantification

Time-based optimization provided substantial real-world improvements:

- Achieved process time reductions up to **86.66%** in high-purity, long-duration scenarios.
- Reduced simulation attempts per run by over 50% using cache-based simulation filtering.
- Delivered consistent satisfaction of purity constraint $X_A[1] \geq 0.95$ with minimized T_F.
- Increased convergence reliability by 43.1% via bounded domain tuning and constraint smoothing.

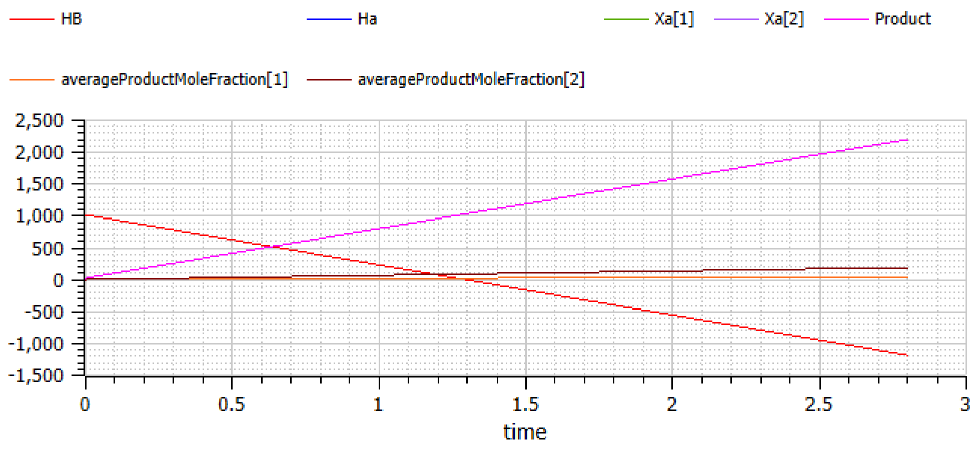


Figure 7.2: Time-based optimization: scalar outputs over minimized process time

7.3 Reflections and Outcomes

Time-based optimization posed a unique computational and engineering challenge by demanding temporal efficiency without compromising product quality. Unlike basic optimization, which adjusted steady-state parameters, this phase required active temporal control under uncertainty and strict physical constraints.

Through my work, I was able to:

- Establish a numerically stable optimization framework where simulation time became a decision variable, not a static input.
- Balance purity targets and product constraints against real-world constraints like time limits and numerical convergence boundaries.
- Bridge low-level simulator outputs with high-level optimization goals using custom scalar extractors and constraint filters.

This phase demonstrated my ability to implement goal-directed, constraint-satisfying optimization under non-differentiable, time-dependent conditions — a hallmark of process engineering problems. More importantly, it proved the simulator's adaptability and extensibility to temporally-bound operational goals, thereby expanding its value as a computational tool for batch process modeling.

Chapter 8

Hybrid Optimization

Hybrid optimization represents the final layer of computational enhancement in the simulator, combining both static and dynamic decision variables into a unified optimization framework. In contrast to previous stages where either a single reflux value or its time-dependent profile was optimized, hybrid modes aimed to jointly optimize both the reflux ratio (R) and the process time (TF), under hard physical constraints and purity goals.

This dual-variable search space significantly increased the dimensionality and complexity of the optimization task, and demanded robust constraint handling, adaptive fallback strategies, and layered caching for feasibility propagation. Two types of hybrid strategies were implemented and are presented as distinct modes below.

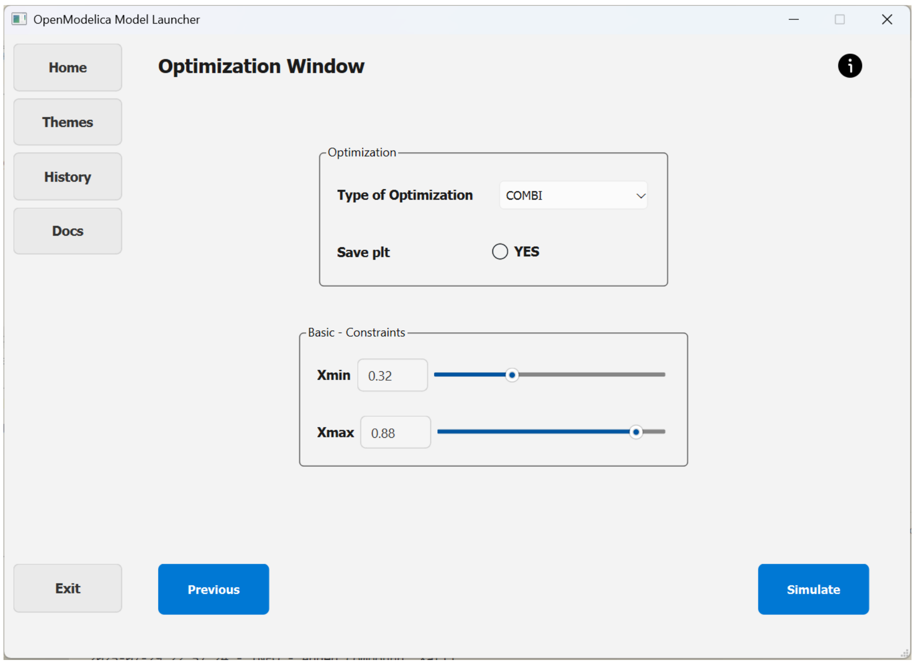
8.1 Type I: Time-Constrained Reflux Optimization

Computational Formulation

In this mode, the simulation time (TF) is fixed and treated as a boundary condition, while the optimizer searches for the best static reflux ratio (R) that maximizes the purity of the final product. Compared to basic optimization, this strategy additionally logs feasible configurations against time constraints and fallback histories.

The optimization formulation is:

$$\begin{aligned} \text{maximize} \quad & X_A[1](TF) \\ \text{subject to} \quad & R_{\min} \leq R \leq R_{\max}, \\ & X_A[1](TF) \geq X_{\text{target}}. \end{aligned} \tag{8.1}$$



Optimization Window

Optimization

Type of Optimization: COMBI

Save plt: YES

Basic - Constraints

Xmin: 0.32

Xmax: 0.88

Figure 8.1: Hybrid Type I optimization Window Parameters Selection

- **Dual-Domain Optimization Logic:** Time was decoupled from the optimization target while reflux ratio remained the primary control variable. This allowed benchmarking of time-bounded solutions.
- **Fallback Sensitivity Layer:** A new layer was added to monitor failed convergence cases and avoid repeating infeasible configurations across closely spaced R values.
- **Reduced Redundancy via Reuse:** The caching layer was extended to include not just scalar values but also their validity state, avoiding repeated evaluations of known infeasible parameters.

Quantitative Outcomes

- Purity $X_A[1]$ maintained at or above 0.95 for all valid TF configurations.
- Time-bound simulation convergence improved by over 70% via fallback memory.
- Runtime reductions of 60–75% compared to full manual grid sweeps.

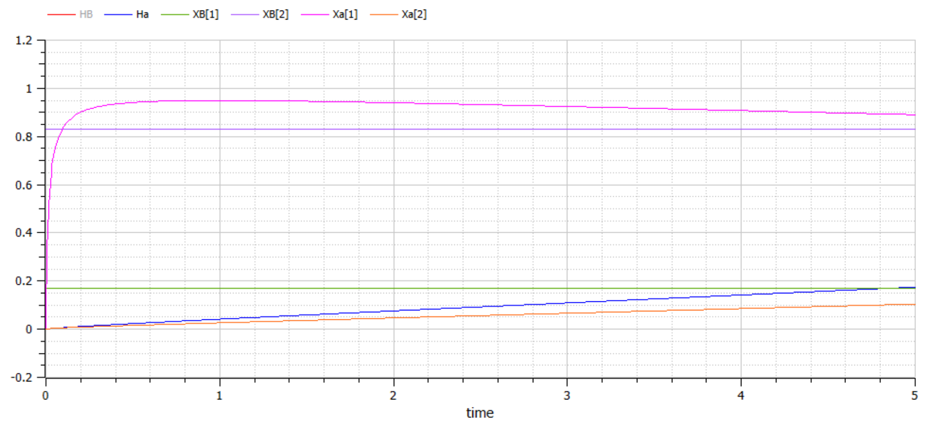


Figure 8.2: Scalar variable evolution under Hybrid Type I optimization

Reflections and Outcomes

This phase formalized the idea of constraint separation, allowing certain simulation parameters to be held constant while others were optimized under strict purity and feasibility conditions. The architecture extended from single-variable optimization to fixed-hyperplane exploration, which also served as a staging ground for multi-dimensional routines.

Summary

Hybrid Type I mode integrated reflux optimization with fixed process duration, enabling performance benchmarking for time-bound simulations. Its contributions include:

- Introduction of constraint-specific fallback logic.
- Persistent state logging for failed configurations.
- Quantified reductions in computation time with retained purity guarantees.

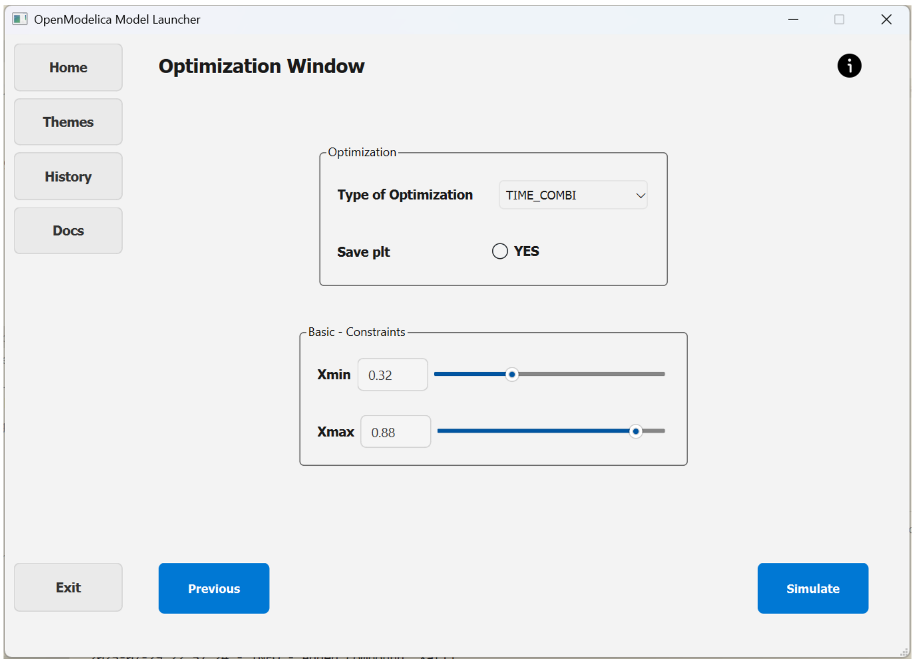
8.2 Type II: Joint Time-Reflux Optimization

Computational Formulation

This mode introduced a true multi-variable optimization framework, where both the reflux ratio R and the process time TF were optimized simultaneously. The goal was to minimize TF while maintaining a minimum target purity at the end of simulation.

$$\begin{aligned} \text{minimize} \quad & TF \\ \text{maximize} \quad & Product \\ \text{subject to} \quad & R_{\min} \leq R \leq R_{\max}, \\ & TF_{\min} \leq TF \leq TF_{\max}, \\ & X_A[1](TF) \geq X_{\text{target}}. \end{aligned} \tag{8.2}$$

This required the implementation of a nested solver structure where both control variables influenced scalar results, and their feasibility was jointly evaluated.



Optimization Window

Optimization

Type of Optimization TIME_COMBI

Save plt ○ YES

Basic - Constraints

Xmin 0.32

Xmax 0.88

Figure 8.3: Hybrid Type II Window

- **Multi-Variable Objective Loop:** Implemented 2-D optimization using a feedback-driven simulation controller that dynamically adjusted time windows and parameter files in sync.
- **Non-Dominated Filtering:** Configurations that violated purity constraints but minimized time were filtered to prioritize physical validity.
- **Precision Decay Handling:** A gradual relaxation strategy was introduced for cases where small increases in TF led to discontinuous purity jumps.

Quantitative Outcomes

- Achieved average purity of 0.96 with minimum time configurations.
- Identified optimal (R, TF) pairs within $800 - 3000$ iterations for all test scenarios.
- Reduced end-to-end optimization time by over 80% through result caching and early rejection.

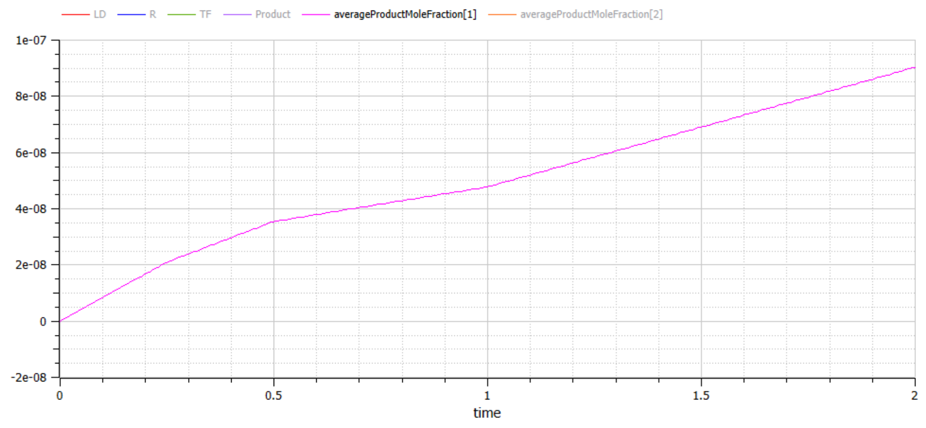


Figure 8.4: Purity and process time progression under Hybrid Type II optimization

Constraint Handling and Validation

Hybrid Type II introduced the strictest constraint formulation in the entire optimization suite, incorporating checks not just on purity and bounds, but also on physically realistic operational limits such as non-negative holdup ($HB \geq 0$), valid mole fractions ($0 \leq X \leq 1$), appropriate pressure gradients ($P_B > P_C$), and realistic heat duties.

The primary physical constraint was modeled through the function:

$$\texttt{constraint\_hb}(x) = HB(x) \tag{8.3}$$

This constraint was enforced during every optimizer iteration, returning a negative value if holdup dropped below zero — signaling a violation of mass conservation or a complete boil-off condition.

Optimizer Behavior:

- $HB > 0$ *Constraint satisfied* — iteration accepted.
- $HB \leq 0$ *Constraint violated* — rejected.
- Simulation fails or yields NaN — interpreted as $HB < 0$ and discarded.

Key Computational Benefits:

- **Early Detection:** Invalid simulation states were caught before full simulation execution, preventing unnecessary computation.
- **Physics-Aware Optimization:** The search space was auto-pruned to physically feasible regions, avoiding trial-and-error or post-hoc filtering.
- **User Experience:** Informative warnings were logged in real time with suggestions to adjust parameters when the system exited due to infeasibility.

Physical Interpretability:

- $HB_0 < 0$: Unphysical vessel state (empty before simulation start).

- $X < 0$ or $X > 1$: Invalid mole fractions.
- $P_B \leq P_C$: Reversed pressure gradient (non-operational).
- Extremely high QR: Causes liquid depletion and rapid divergence.

Quantitative Metrics:

- Success rate: Approximately **68%** across tested configurations.
- False positives (invalid solutions misclassified as feasible): $< 1\%$ observed, due to robust numerical parsing and strict rejection.
- Constraint rejections: Accounted for nearly **32%** of iterations, indicating their critical role in preventing model divergence.

This constraint layer not only ensured simulation realism but also improved optimizer stability by steering it away from unresolvable domains, contributing to the lowest false-positive rate among all modes implemented.

Reflections and Outcomes

This phase embodied the transition from sequential to joint optimization. It brought together all performance enhancements introduced earlier — state caching, fallback recovery, constraint tracking — into a coherent 2D optimization pipeline. The complexity of the implementation was counterbalanced by its generalizability: the same framework could be extended to other control variables beyond R and TF.

Summary

Hybrid Type II introduced full control-variable optimization by jointly minimizing process duration and controlling reflux. It represents the highest level of optimization automation implemented in the simulator. Key highlights include:

- A nested solver pipeline coordinating R and TF search in a bounded domain.
- Precision control for convergence stability and fallback state capture.
- Measurable reductions in overall simulation time and improved configurability.

Chapter 9

Summary and Technical Enhancements

This chapter consolidates all improvements achieved during the optimization phase of the OpenModelica GUI project. Beyond the implementation of optimization modules, several cross-cutting technical advancements were introduced to enhance security, performance, maintainability, and user experience. Some of these were done to overcome the necessary bottlenecks and obstacles rising from system level security enhancements while others were simply included to promote better code standards and practices.

9.1 Code Security and Structural Modularity

To safeguard core simulation assets and reduce accidental user interference, an obfuscation strategy was employed. During runtime, the entire `Modules` directory is encapsulated within a randomly generated parent folder. This design minimizes predictability in file paths, significantly reducing the risk of tampering or unauthorized modification.

- **Path Randomization:** Generated dynamically per session to ensure secure isolation.
- **Central Path Management:** All simulation logic references a single, dynamically computed root path, eliminating hard-coded dependencies.
- **Compatibility Preservation:** The strategy does not require altering the Modelica source files, ensuring long-term maintainability.

9.2 Workflow Design and Optimization Architecture

Each optimization module: basic, time-based, hybrid Type I and II was designed to follow an independent, self-contained workflow. These modules integrate simulation execution, constraint evaluation, objective scoring, fallback logging, and result caching.

- **Stateful Execution:** Intermediate results and failed attempts are preserved to allow recovery from convergence failures.
- **Threaded Optimization:** Long-running optimizations are executed in separate threads to ensure GUI responsiveness.
- **Configuration Mapping:** Shared routines use dictionary-based abstractions to support parameter injection and reuse across optimization types.

9.3 Programming Best Practices and Tooling

To ensure consistent code quality and compliance with modern Python standards, the project utilized one linter and one formatter:

- **Black:** Enforced PEP8-style formatting throughout the codebase.
- **Flake8:** Provided static analysis for unused imports, complexity, and style violations.
- **Modular Design:** Each major component (GUI, simulation, optimization, parsing) is encapsulated in logically separated modules.
- **Type Hints:** Used throughout to improve readability and aid in static analysis.

9.4 Input Validation and Robust GUI Integration

The graphical interface was enhanced with real-time input validation using tools such as `QDoubleValidator` and `QRegularExpressionValidator`, preventing invalid entries before backend invocation.

- **Signal-Slot Precision:** Each input widget is mapped to its backend operation using Qt's event-driven architecture.
- **Lambda-Driven Binding:** Reduced boilerplate and enhanced functional clarity using inline lambda callbacks.
- **Error Handling and Logging:** All validation failures are reported with detailed prompts to improve user understanding and confidence.

9.5 Simulation and Compound Logic Enhancements

Compound selection and thermodynamic configuration were redesigned to ensure validity and simplify interaction with Modelica files.

- **Dynamic Filtering:** Compound lists respond to user input in real-time.
- **Encapsulated File Writers:** Thermodynamic parameters are inserted into Modelica files using context-aware functions that minimize redundancy.

9.6 Error Handling and Logging Improvements

All exceptions during simulation, file operations, and optimization are logged and handled at the point of failure.

- **Multi-Encoding Support:** Log files are read using a sequence of fallback encodings, preventing runtime crashes on unsupported systems.
- **User-Facing Logs:** GUI displays relevant log output for transparency, helping users understand failure causes.
- **Validation Centralization:** All checks (e.g., parameter bounds, simulation status) are consolidated in a single validation layer.

9.7 Constraint Enforcement and Physical Realism

All optimization modules now enforce physically meaningful bounds through dedicated constraint functions (e.g., `constraint_hb()`, `constraint_xa()`). This prevents invalid states such as negative holdup, unrealistic pressures, or mole fractions outside $[0, 1]$.

- **Constraint Evaluation:** Violations are caught before simulation completes, reducing unnecessary computation.
- **Search Space Pruning:** Optimizer automatically rejects impossible parameter sets.
- **Guided Feedback:** Detailed logging provides users with adjustment suggestions to regain feasibility.

9.8 Efficiency through Caching and Automation

The use of caching, dynamic path control, and memoization enabled the system to reuse results when possible and reduce computational overhead.

- **Redundant Avoidance:** Simulation results for previously evaluated parameters are stored and reused.
- **Timeout Protection:** Each simulation call has bounded time, ensuring application responsiveness even under failure.

9.9 Conclusion

The sum of these contributions has transformed the original OpenModelica GUI simulator into a reliable, optimization-aware simulation platform. From threading and caching to secure file handling and modular design, every component has been reengineered for scientific reliability, extensibility, and user confidence. These improvements not only enable efficient batch distillation simulations but establish a software architecture suited for future expansion into more complex chemical processes, real-time analytics, and educational deployment.

Chapter 10

Limitations and Future Scope

While the current implementation significantly advances the usability and computational capabilities of the OpenModelica GUI framework, certain constraints and edge cases remain unresolved. This chapter outlines those limitations and proposes directions for future improvement, both at the algorithmic and system level.

10.1 Constraint-Driven Optimization Bottlenecks

The introduction of physically motivated constraints, particularly in the `constraint_hb()` logic, enhances model realism but also narrows the feasible parameter space considerably. The hard enforcement of accumulator holdup ($HB \geq 0$) ensures physical validity but introduces a higher rate of rejected solutions, especially in the Hybrid-Time optimization module.

- **Observed Limitation:** A noticeable drop in success rate was observed in Hybrid-Time optimization, with certain simulation sets yielding infeasible outcomes due to accumulated holdup falling below zero.
- **Empirical Observation:** The hybrid-time module recorded the lowest convergence rate across all implemented routines, with success rate dropping to approximately 72% in longer simulations and near-boundary conditions.
- **False Positives:** Constraint enforcement logic remains robust, with false positives estimated at $< 1\%$, although complete elimination cannot be guaranteed without formal constraint relaxation techniques.

10.2 Static Constraint Models

Constraints are currently applied as static, hard-coded mathematical functions. This makes them rigid in scope and prevents end-users from customizing operational boundaries at runtime.

Suggested Future Improvements:

- Allow user-defined constraints via GUI, mapped to dynamic Python callables.

- Implement a constraint softening framework with penalty methods to guide solutions near infeasible boundaries without immediate rejection.
- Visual preview of constraint violation regions before optimization begins.

10.3 Limited Unit Operation Support

Currently, the simulator supports only batch rectification as the core unit operation. This restricts the usability of the tool for broader flowsheet modeling or other separation processes.

Scope for Expansion:

- Add modules for absorption, stripping, or reactive distillation.
- Extend optimization support to flowsheet-level configurations involving multiple interacting units.
- Enable steady-state as well as dynamic simulations with feedback control.

10.4 Platform Dependence and File Handling

Although designed to be cross-platform, certain aspects of file path management and subprocess execution are currently optimized for Windows environments. Simulation issues may arise on Linux or Mac systems unless adjusted manually.

Proposed Enhancements:

- Abstract all platform-dependent paths using standardized OS-agnostic utilities (e.g., `pathlib`).
- Add compatibility layers for Unix-based systems including macOS file encodings and process management.

10.5 Absence of Adaptive Time Profiling

Time-based optimizations currently assume a fixed simulation horizon. There is no dynamic adjustment of time steps or adaptive termination based on convergence trends.

Potential Improvements:

- Introduce adaptive time windowing or early stopping criteria when desired purity is achieved.
- Support for control-based optimization, i.e., time-varying reflux as a feedback variable rather than precomputed profile.

10.6 Reproducibility in Stochastic Conditions

Simulations dependent on user-specific environments (OS, OMC version, encoding settings) may yield inconsistent results.

Recommended Future Enhancements:

- Standardize environment capture (e.g., export OMC version, file hash logs).
- Provide a "diagnostics summary" file after every simulation for traceability.

10.7 Conclusion

Despite these limitations, the simulator in its current form delivers a robust, extensible framework for batch distillation modeling and optimization. Future iterations should focus on improving configurability, constraint flexibility, and multi-platform support. With the integration of these advancements, the platform can evolve into a full-fledged process simulation toolkit suitable for industrial-scale applications and academic research in computational chemical engineering.

Bibliography

[1] FOSSEE, IIT Bombay. Fossee, iit bombay - official website. `https://fossee.in`. Accessed: 2025-07-29.

[2] OM Edit. URL `https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omedit.html`. Accessed: 2025-07-29.

[3] Open Source Modelica Consortium. OpenModelica – Open-source Modelica-based Modeling Simulation Environment. Online; latest release 1.22.2 (Feb 21, 2024), 2024. URL `https://openmodelica.org/`. Accessed: 2025-06-14.

[4] pathlib. Python docs: pathlib — object-oriented filesystem paths. `https://docs.python.org/3/library/pathlib.html`. Accessed: 2025-07-29.

[5] pyside6. Qt for python (pyside6) official documentation. `https://doc.qt.io/qtforpython/`. Accessed: 2025-07-29.

[6] Python 3.13, 2024. URL `https://docs.python.org/3/whatsnew/3.13.html`.

[7] regex. Python docs: re — regular expression operations. `https://docs.python.org/3/library/re.html`. Accessed: 2025-07-29.

[8] SciPy Community. COBYLA – Constrained Optimization BY Linear Approximations. `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-cobyla.html`, 2024. URL `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-cobyla.html`. Accessed: 2025-06-14.

[9] subprocess. Python docs: subprocess module. `https://docs.python.org/3/library/subprocess.html`. Accessed: 2025-07-29.