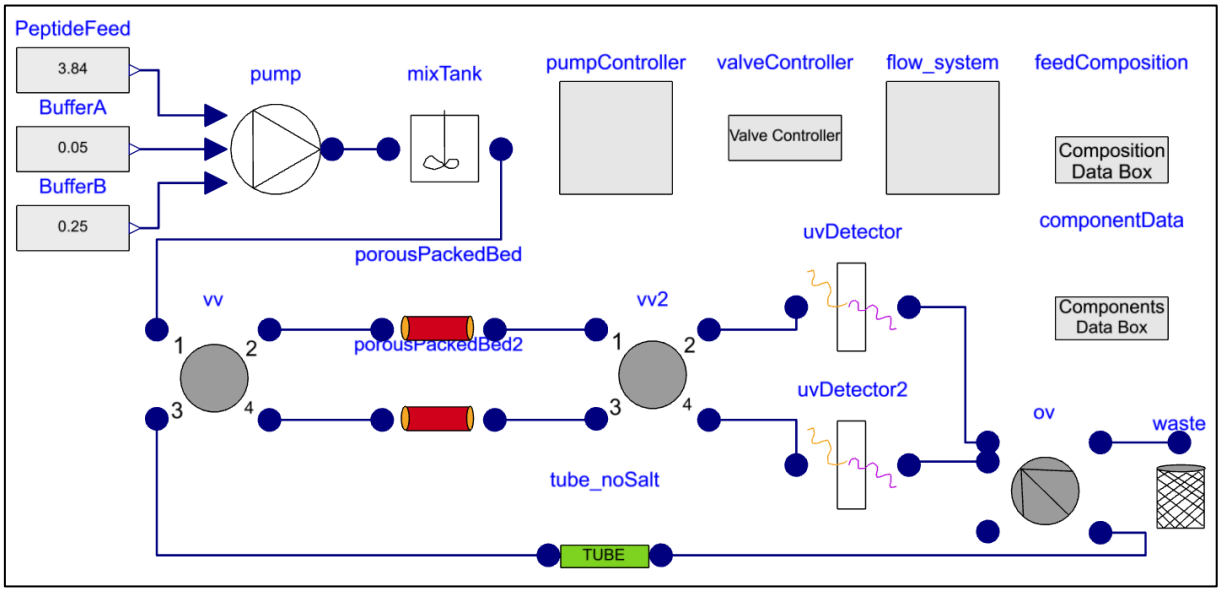
Modeling and simulation of chromatographic separation in Modelon Impact



by

Oliver Thelander

Department of Chemical Engineering
Lund University

June 2022

Supervisor: **Niklas Andersson**
Examiner: **Professor Bernt Nilsson**

Preface

I would like to thank my examiner professor Bernt Nilsson for swiftly granting me the opportunity for this master thesis and his guidance on where to take this project. I would also like to thank my supervisor Niklas Andersson for helping me throughout this project with troubleshooting, guidance on where to take this project and his patience with me.

I would like to thank Mathias Strandberg at Modelon for supplying me with a Modelon Impact license.

Summary

The aim of this master thesis is to implement a previously modeled chromatographic adsorption process from MatLab into the Modelon Impact, a simulation software. Modelon Impact is using Modelica as language syntax that is useful for creating a large system of interconnected models that has the potential to work in different configurations. The model is successfully implemented and done in a step-wise manner. The comparison between the two simulations are near flawless, but some variations can be seen as load time of the column increases. While not conclusively deduced, the problem most likely lies in how the two different solvers work.

Another aim is to use the functionality of Modelica and create different models in order to create a more accurate setup seen during laborative trials. These models range from the column and pump that have previously been modeled in the original MatLab code to controllers, valves and tubing that reflect real experimental setups. The interconnectivity between models are successful, but due to variable overlap some setups had to be altered in order to complete the simulations.

While Impact has proved to be more than capable to simulate chromatographic processes and handle user-created sub-models, there is still room for investigation of it as a tool for finding parameters or handling more complex models.

Sammanfattning

Syftet med denna masteruppsats är att implementera en tidigare modellerad kromatografisk adsorptionsprocess från MatLab till Modelon Impact, en simuleringsmjukvara. Modelon Impact använder sig av Modelica som syntax som i sin tur är användbart för skapandet av stora system av sammankopplade modeller som kan fungera i olika konfigurationer. Implementeringen är lyckad och gjord stegvis under projektet. Jämförelse mellan de två modellerna är nästan perfekt men mindre variationer kan observeras vid höga laddningstider av kolonnen. Medan det inte är helt fastställt ligger troligtvis skillnaden i resultat i hur de olika lösarna fungerar.

Ett annat syfte är att använda Modelicas funktionalitet och skapa olika modeller för att skapa ett mer rättvist system som återspeglar laborativa försök. Dessa modeller sträcker sig från kolonn och pump som redan var modellerade i MatLab till kontroller, ventiler och rör för att efterlikna verkligheten. Sammankopplingen mellan alla modeller är överlag lyckad, men på grund av variabelöverlapp behövdes några system justeras för att fullständigt kunna simuleras.

Medan Impact har visat sig vara mer är kapabel för att simulera kromatografiska processer och hantera självskapade modeller finns det fortfarande utrymma för vidare arbete som ett verktyg för parameteranpassning eller hantering av mer komplexa modeller

Popular scientific summary

Modeling and simulation of a chromatographic process in Modelon Impact

by Oliver Thelander

Modelon Impact is a powerful modeling and simulation software primarily used in the energy, automotive and aerospace industry. Would it be equally useful in the simulation of chromatographic processes compared to conventional tools?

Adsorption chromatography and the simulation of it is one of the many focuses at the department of chemical engineering at Lund University. During adsorption chromatography, a column filled with small porous beads, called the stationary phase, is loaded with a feed containing adsorbing components, called the mobile phase. The components in the feed adsorb onto the stationary phase and are eluted with a rising salt concentration fed through the column, resulting in a separation between the components based on their binding strength.

This process is currently modeled and simulated in MatLab and Python, but this thesis aims to model the same process in Modelon Impact, a simulation software using the language Modelica. A feature of Modelon Impact that will be examined is the creation, connection and reuse of models, where models can be linked together in a visually accessible environment. And finally, is there a future use case of Modelon Impact for chromatographic processes.

The model implementation and the possibility to create and link many models to mimic a laboratory setup were both successful. While the simulation was near flawless between the old simulations and the new in Modelon Impact, there arose some small differences that were more pronounced at high parameter values regarding column load time and starting salt concentration. This variation might have been a result of how the internal solvers work for each simulation environment, ode15s in MatLab and CVode in Modelon Impact.

Both models show similar simulation times between 1-5 seconds where Modelon Impact was consistently longer than MatLab. While this looks bad for Modelon Impact, MatLab requires sparse matrices that would otherwise drastically increase simulation time. Modelon Impact does not require these sparse matrices at all. The discretization size is also a factor for simulation time. While kept relatively low in this thesis, this is also something that increase more for MatLab than for Modelon Impact as a result of Modelon Impact only have to compile the model once, and if changes that do not affect the compiled structure is done to Modelon Impact then the simulation can be done in more rapid successions.

Modelon Impact has shown to be more than capable to model and simulate chromatographic processes but discovery within the software is not done. The software has *experimentation mode* where parameters can be temporarily changed and allow for parameter sweeps, the inStream operator that handles flow-specific variables and evolved models that might otherwise drastically increase simulation time in other programming environments.

Populärvetenskaplig sammanfattning

Modellering och simulering av kromatografisk separation i Modelon Impact

av Oliver Thelander

Modelon Impact är ett kraftfullt modellerings- och simularingsmjukvara främst använd inom energi-, bil- och luftfartsindustrin. Skulle den kunna vara lika användbart för simulering av kromatografiska processer jämfört med konventionella metoder?

Adsorptionskromatografi och simulering därav är ett av de många fokus på kemitekniska institutionen vid Lunds Universitet. Vid adsorptionskromatografi används en kolonn fylld med små porösa kulor, kallat den stationära fasen, som laddas med en ingående ström innehållande adsorberande komponenter, kallat den mobila fasen. Komponenterna i strömmen adsorberas på den stationära fasen och är därefter eluerad med en ökande ingående saltkoncentration genom kolonnen som resulterar i en separation mellan komponenterna baserat på bindningsstyrkor.

Denna process modelleras och simuleras för tillfället i MatLab och Python, men denna uppsats mål är att modellera samma process i Modelon Impact, en simuleringsmjukvara i språket Modelica. En funktion hos Modelon Impact som ska undersökas är skapandet, sammankopplandet och återanvändandet av modeller, där modeller kan kopplas samman i en visuellt lättillgänglig miljö. Slutligen, finns det framtida användningsområden för Modelon Impact gällande kromatografiska processer?

Modellimplementeringen och möjligheten att skapa och koppla samman flera modeller för att efterlikna en laboratorieuppställning var båda lyckade. Medan simuleringen var näst intill perfekt mellan den gamla simuleringen i MatLab och nya simuleringen i Modelon Impact uppstod små skillnader som blev tydligare vid höga parametervärden för laddningstid och startsaltkoncentrationen. Denna skillnad kan ha varit ett resultat av hur de interna lösarna fungerar för respektive simuleringsmiljö, ode15s i MatLab och CVode i Modelon Impact.

Båda modellerna uppvisar snarlika simuleringstider mellan 1-5 sekunder där Modelon Impact var konsekvent långsammare än MatLab. Medan detta kan framstå som dåligt för Modelon Impact behöver MatLab använda sig av sparsamma matriser som i övriga fall skulle drastiskt öka simuleringstiden. Modelon Impact behöver inte dessa sparsamma matriser överhuvudtaget. Diskretiseringsstorleken är också en faktor för simuleringstiden. Medan den är lågt satt under denna uppsats är det något som påverkar MatLab mer än Modelon Impact som ett resultat av att Modelon Impact endast kompilera modellen en gång och så länge den kompilerade strukturen inte förändras kan Modelon Impact simulera snabbare mellan försöken.

Modelon Impact har visat sig vara mer än kapabel att modellera och simulera kromatografiska processer men funktionaliteterna i mjukvaran är inte alla funna än. Mjukvaran har *experimentation mode* där parametrar kan temporärt förändras och tillåta parametersvepningar, inStream-operatorn som hanterar flödesspecifika variabler och mer utvecklade modeller som annars drastiskt ökar simuleringstid i andra programmeringsmiljöer.

Table Of Contents

1 Introduction

Purification processes are important across the chemical, medical and food industries; a product with a high purity and low on contaminants often fetch a higher price and, more importantly, be allowed to be used in other processes. Throughout the years, many purification processes have been developed. From varieties of centrifugation techniques to chromatography and filtration techniques. One chromatography method, adsorption chromatography, is a common tool in purification where a column filled with a stationary phase is used.

While purification processes are important tools, they are also often expensive to use. The tools in-and-of-themselves are expensive to produce and might be prone to break or the feed solution is already difficult to produce leaving no economical room for a wasted batch. To minimize cost and maximize product yield, simulation is a safe tool to use before real purification is carried out. Simulating a process can help evaluate and optimize it before scaling up. This includes the physical parameters of the system such as size, operating parameters such as temperature and flow and chemical parameters such as a product's adsorption capacity. Furthermore, simulations are quick and easy to repeat and the time for changing a parameter or simulate the system that would otherwise take hours of real time can be done in a manner of seconds.

Simulation of purification processes can be done in a variety of digital environments. While the possibility lies in many places, some are more suitable than others. This thesis implements a model previously modeled and simulated in MatLab and Python. While both MatLab and Python work they both also require the use of sparse matrices to keep simulation time low. Modelon Impact is a simulation software using the programming language Modelica, both created with model simulations in mind.[1]

1.1 Aim

The aim is to create several models that were previously one model script MatLab in Modelon Impact. These models will strive to be as general as possible to be reused in different configurations such as single column, recirculation and double column. Possibilities, usefulness and performance of Modelon Impact will also be discussed.

2 Background

2.1 Adsorption chromatography

Adsorption chromatography is a widely used separation technique where a mobile phase containing soluble molecules is separated by passing through a column containing a stationary phase in which the soluble molecules bind to the stationary phase. The bound molecules are later released using an eluent with a higher binding power as a mobile phase, resulting in a more purified solution if phases, eluent, etc. are correctly chosen for the task. The eluent is fed through the column after the solute feed has been loaded onto the column and afterwards acts as a carrier stream, increasing the eluent concentration over time in order to elute all solubles.[3]

2.2 Experimental setup

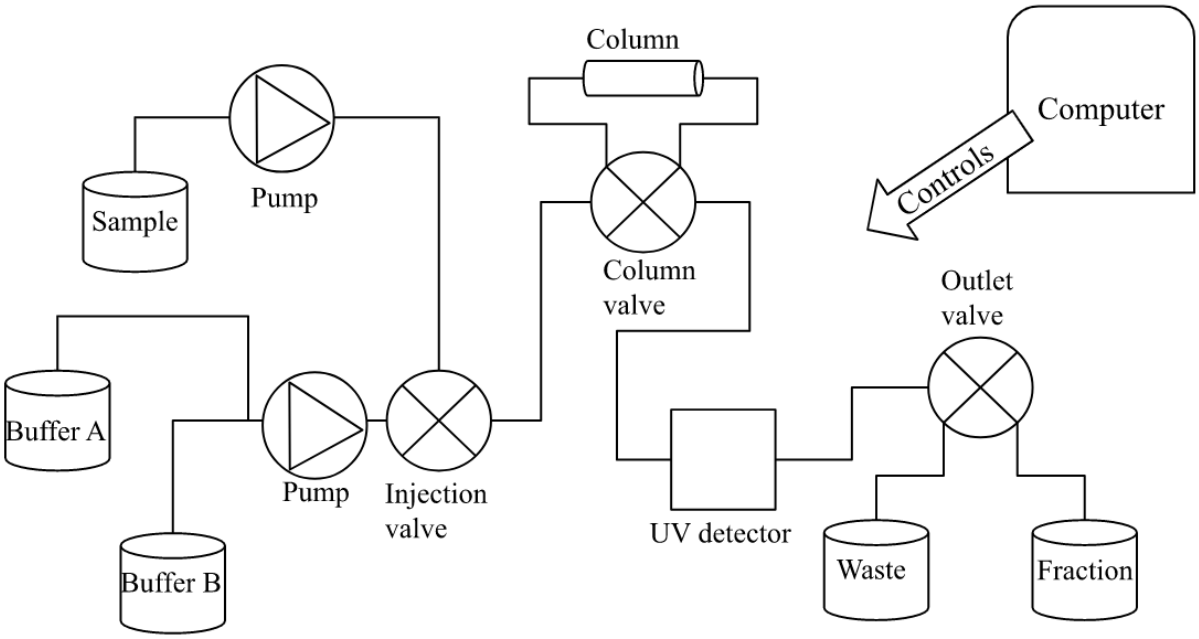


Figure 1: Simplified view of the experimental setup.

A view of the experimental setup is shown in figure 1. Sample is being pumped through the injection valve through the column valve and into the column where it loads the column for a certain amount of column volumes. A stream of a mixture between buffer A and B is then pumped the same way and elutes the components in the column, which are passed through a UV detector, outlet valve and depending on the UV detection to either the waste or the fraction that is being collected. The pumps and valves are controlled with a computer and run a predetermined set of instructions with each run. The resulting chromatogram of the concentration profiles is sent back to the computer from the UV detector.

2.3 The simulated model

The model that is aimed to be implemented has already been done in both MatLab and Python as a sub-project of the advanced course Process Simulation at Lund University. The previously implemented model is similar to figure 1 with some notable exceptions: only within the confines of the column is simulated and thus all other parts are baked into the model and reduced as much as possible such as no volume outside of the column. The previous model implementation will here on out only be referred to as the MatLab model.

2.3.1 FVMtools

To simulate the column, it must be discretized which were done with FVMtools, implemented by the department of chemical engineering at Lund University. FVM stands for Finite Volume Method, and FVMtools is a script that produce a discretization based on some input parameters: first order derivative approximation, second order derivative approximation and boundary approximations.

2.4 Modelica

Modelica is an object-based programming language where the programmer can create reusable models for use in larger complex physical systems. Models have connectors that enable input and output signals between the model components, often creating a larger block diagram of a system, for example an electrical circuit. The model components are written using Modelica syntax with an equation-based description. Time is a built-in variable and all variables are a function of this inherent variable. Modelica has several keyword operators built into it and der() is one which is the time derivative that takes advantage of the built-in time variable.[4]

2.5 Modelon Impact

Modelon Impact is a system simulation software created by Modelon in Lund, Sweden and is an evolution of the previous project JModelica. It uses the Modelica programming standard as syntax and is a browser-based cloud-native software where users can easily create models, simulate systems and share workspaces with others for more accessible collaborations. Modelon also provides a vast library of ready-to-use models, however non except interface connectors and real number signal models were used in this project as the library lacked models relating to concentration flows. [1]

2.5.1 Workspace

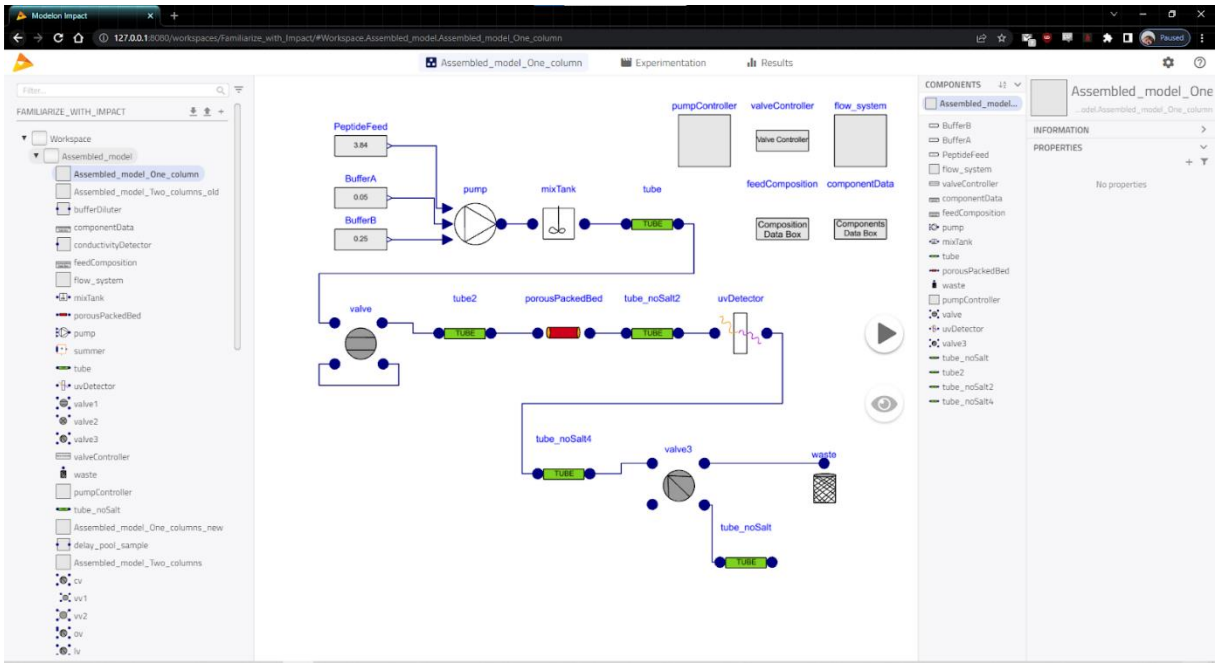


Figure 2: Overview of the Modelon Impact interface

The Impact workspace consists mainly of three parts (see figure 2): the workspace itself (middle), the library (left) and the model properties (right). In the workspace area one can either enter the current sub-model and code its structure, for example how the pump should work, or one can connect sub-models together by dragging them from the library onto the workspace and then click and drag the mouse cursor to connect models. The library contains both user-created models and pre-made models by Modelon. The right side shows the properties of the currently selected model. This includes declared parameters, constants and variables that can easily be changed without going into the model code. This also makes models reusable as changing a value in a large connected model won't change the code of the original sub-model. [2]

2.5.2 Modes

Impact can be further changed into different modes (see top middle in figure 2): model, experimentation, and results. In model mode, see figure 3, one can change the code freely but needs to be recompiled after change. In experimentation mode, see figure 4, one can change parameters without changing the model itself that could be useful for making a parameter sweep. Experimentation mode is also used to set the experiment time, the solver used, and tolerance used. Result mode, see figure 5, stores the simulation data for the current set of parameters and experiment time. The resulting input and output of each sub-model over time is easily accessible by dragging and dropping the interesting component of the system. [2]

The model is run by pressing the large 'play'-button in the center right of the workspace. This starts the simulation, and is compiled if it is the first time, otherwise not. If the simulation is unsuccessful, you get an error message and the result up until failure is saved, for example if a physical value goes to infinity midway through the simulation time. If successful, the browser automatically changes to the results tab and the new results. [2]

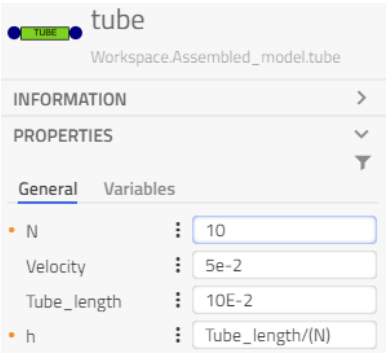


Figure 3: View of a model in model mode. The parameters can be changed and would result in a change in the code. The orange dots indicates that a change would require a recompilation.

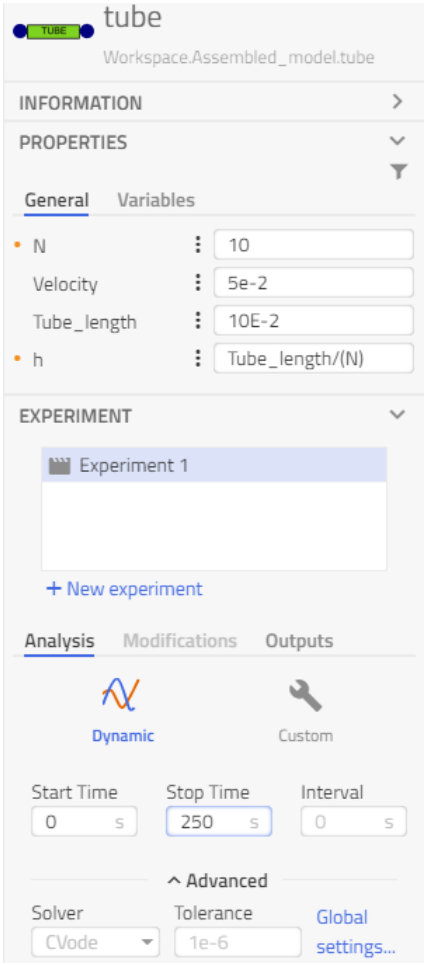


Figure 4: View of a model in experimentation mode. Changing a parameter is saved as a separate experiment set without changing the model code. For each experiment set several global parameters can be set such as time, interval size, solver type and tolerance. If left blank default values are used.

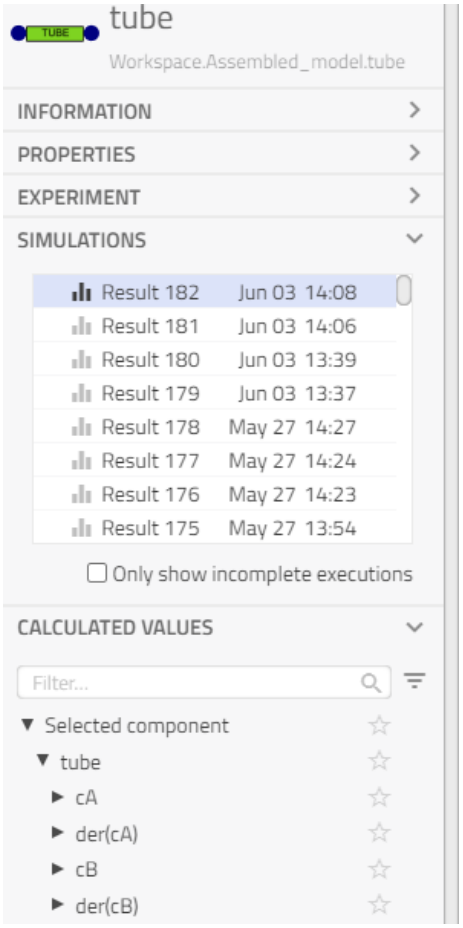


Figure 5: View of a model in results mode. The compilation and simulation log can be accessed here along with the resulting output for each variable and parameter in the model as a function of time. The output results can easily be dragged and dropped onto the workspace canvas for resulting graphs, exemplified several times later in this report.

2.5.3 Impact scripting

```
1  model uvDetector
2      .Modelica.Blocks.Interfaces.RealVectorInput u[4] annotation(...);
3      .Modelica.Blocks.Interfaces.RealVectorOutput Concentration_Flow[4] annotation(...);
4
5      Real Absorbance_A;                        //Abs
6      Real Absorbance_B;
7      Real Absorbance_C;
8      Real Absorbance_D;
9
10     parameter Real light_path_length = 1;     //cm
11
12     parameter Real Absorptivity_A = 1;        //L/g/cm
13     parameter Real Absorptivity_B = 1;
14     parameter Real Absorptivity_C = 1;
15     parameter Real Absorptivity_D = 1;
16
17     Real UV_detection[4];
18     Real UV_c_tot;
19
20 equation
21     Absorbance_A = Absorptivity_A * u[1] * light_path_length;
22     Absorbance_B = Absorptivity_B * u[2] * light_path_length;
23     Absorbance_C = Absorptivity_C * u[3] * light_path_length;
24     Absorbance_D = Absorptivity_D * u[4] * light_path_length;
25
26     Concentration_Flow = {u[1],u[2],u[3],u[4]};
27     UV_detection = {Absorbance_A,Absorbance_B,Absorbance_C,Absorbance_D};
28
29     UV_c_tot = Absorbance_A+Absorbance_B+Absorbance_C+Absorbance_D;
30     annotation(...);
31 end uvDetector;
```

Figure 6: Editing of model code for a UV detector in Modelon Impact.

Each sub-model consists of two parts: the declaration section and equation section. In the declaration section parts can be added that are necessary for the model. This can include parameters, variables, inputs or outputs. One must define the type of parameters or variables, for example if they are integers, reals, Booleans of strings. Variables are not given any value since they vary with the built-in variable time. Parameters must be given a value in the model, but can be changed in experiment mode without changing the original model. The equation section contains the equations used in the model, for example variables changing with the input signal or time. This section also assigns variable values to outputs if required. [2] In figure 6 a simple UV detector is modeled. Rows 2 and 3 declare the input and output of the model (a concentration inlet and outlet). The detector uses Beer-Lambert's law and thus the variable absorbance for each component is declared on rows 5-8, the light path length b as a parameter on row 10, the absorptivity for each component as parameters on rows 12-15, a variable array for storing all the absorbances on row 17 and a single-value variable for all absorbances on row 18. The equation section starts after row 20 with absorbance equations for each component on rows 21-24, the assignment of the outflow concentrations on row 26, the detector values on row 27 and the summation of all absorbances on row 29. The model ends with annotation containing information of the graphical representation layout in the workspace view.

2.5.4 Model connections

A feature of Modelica, and thus Impact, is the usage of connectors where two models can be connected graphically with a simple connect-equation. This allows information between two or more models to be linked, leaving models more generalized and hopefully graphically easier to interpret.[4]

2.5.5 Solvers

For both Impact and previous implementations of the model, large systems of differential equations are required to be solved and thus solvers that are constructed for these purposes are used. In the MatLab model the ode15s solver is used which is a stiff multistep variable-step variable order ordinary differential equation solver based on the numerical differentiation formulas. In Modelon Impact the default CVode solver is used which is a stiff and non-stiff multistep variable-step variable order ordinary differential equation solver. Both solvers include the backwards differentiation formulas. CVode have a default relative tolerance of 1e-6 and ode15s is using a set relative tolerance of 1e-6. [5][6]

3 Material and method

3.1 The target model

This project is based on a project from the course Process Simulation, KETN01, at Lund University, where a simulation of a porous packed bed adsorption chromatography column was done. This was previously done in MatLab, but has since moved to Python. All reference data used is the same from this previous sub-project and found in Appendix A. The complete old model code from MatLab can be found in appendix B.

The model is a heterogeneous porous packed bed model with adsorption. The component balance for each soluble component in the mobile phase is:

$$\frac{\partial c_i}{\partial t} = D_{ax}\frac{\partial^2 c_i}{\partial z^2} - \frac{v}{\varepsilon}\frac{\partial c_i}{\partial z} - \frac{1-\varepsilon_c}{\varepsilon}r_i \tag{1}$$

Where

$$D_{ax} = \frac{L*D_p}{P_{ec}} \tag{2}$$

and

$$\varepsilon = \varepsilon_c + (1-\varepsilon_c)\varepsilon_p \tag{3}$$

The component balance for each soluble component in the stationary phase is:

$$\frac{\partial q_i}{\partial t} = r_i \tag{4}$$

For active sites occupied and available the Langmuir general competitive adsorption was used:

$$r_i = k_{kin,i}(H_i * c_i\left(1 - \sum_{j=1}^{N}\frac{q_j}{q_{max,j}}\right) - q_i \tag{5}$$

With Henry's modified equation

$$H_i = H_{0,i} * c_{s,j}^{-\beta_i} \tag{6}$$

The boundary conditions with Dirichlet at the inlet and von Neumann at the outlet and thus:

$$c_i(t \leq 0, L) = c_{i,in} \tag{7}$$

$$c_i(t > 0, L) = 0 \tag{8}$$

$$\frac{\partial c(t,L)}{\partial z} = \frac{\partial c_s(t,L)}{\partial z} = 0 \tag{9}$$

The initial values are:

$$c_i(0, z) = 0 \tag{10}$$

$$q_i(0, z) = 0 \tag{11}$$

$$c_s(0, z) = c_{Buffer\ A} \tag{12}$$

The column must be discretized for the solvers before running the simulations. The discretization was derived from FVMtools, and similar to earlier simulation in MatLab a 2-point backwards approximation was used for the first derivative and a 3-point central approximation for the second derivative in equation 1. The discretizations and their derivation is found in Appendix C.

3.1.1 Assumptions

The following assumptions are made to make the equations used and simulations more manageable:

- The mobile phase is incompressible and has both constant density and velocity.
- Salt does not adsorb or desorb onto the column.
- Convection only occurs in the inlet flow direction.
- Dispersion occurs in all directions.
- The mobile phase can only occupy the pores in the particles and the column void.
- Constant temperature that results in constant D_{ax}.

3.2 Model implementation

Since MatLab is different from Modelica syntax, the implementation of the model is done in a step-wise manner. With each successful implementation and simulation, complexity is added until a complete model is implemented. CVode is used as a solver in Impact and a relative tolerance of 1e-6. While Impact simulates in seconds by default, this is converted into column volumes, CV, and used as timescale.

3.2.1 Small simplified tank series

The first step into Modelon Impact and implementing the model would be thought of as a small tank series instead of a discretization of a column. The following component balance over each tank is:

$$Ack = In - Out + Prod \tag{13}$$

$$\frac{\partial Vc}{\partial t} = Fc_0 - Fc_i + V_i r_{i,V} \tag{14}$$

$$\frac{\partial c}{\partial t} = \frac{F}{V}(c_{i-1} - c_i) + \frac{1-\varepsilon}{\varepsilon} r_i \tag{15}$$

$$\frac{\partial q}{\partial t} = r_i = k_{kin}\left(H * c\left(1 - \frac{q_i}{q_{max}}\right) - q_i\right) \quad (16)$$

The amount of tanks (partitions) is kept low at four. Only one component in the feed stream will be simulated as a short pulse of 1 g/L for 0.25 CV. Other relevant data that was used is found in Appendix A where the sole component is component A. Simulation time is 5 CV. Henry's constant, contrary to equation 6, is kept constant since no salt is included yet.

3.2.2 Addition of salt gradient and system phases

The second step of implementation is the addition of salt as a separate component. Since the assumption is that salt neither absorb nor desorb onto the stationary phase, $r_s = 0$, makes the corresponding component balance simplified, see equation 15. With salt added to the model, Henry's modified equation, equation 6, is included in the code. The amount of partitions N is increased to 50.

The salt requires two buffer solutions at 0.05 and 0.25 g/L in which a gradient will be constructed. The system will be split into four phases: a load phase where the feed is fed into the column, a short wash phase where a low salt concentration flows and start to wash out the adsorbed component, a gradient phase where the salt concentration gradually increases over time and a final phase where the max buffer concentration flush out the remaining component in the column. These phases dictate the inlet flow to the column with sample feed only happening in the beginning and is followed by a constant salt flow. The salt concentration inlet and gradient are dictated with if-equations similar to the MatLab code, see Appendix B.

3.2.3 Multiple components, discretization and improved model implementation

To inch further towards the complete model implementation, multiple components are added. The data for each component used are found in Appendix A and all the equations 1-12 is being implemented. As for discretization a 3-point central approximation is used for both the first and second order derivative. The number of partitions are further increased to 100.

3.2.4 Further improving discretization and final implementation

As a result of the implementation in 3.2.3, oscillations of values were detected at the front of the concentration curves and thus was decided to change to a 2-point backwards approximation for the first order derivative. This adjustment finalized the implementation of the MatLab model.

3.3 Model comparison

With a complete model implementation, both will be simulated using the same values as found in appendix A. The models will be compared against each other and against a real experimental run, however the internal comparison between models is the focus. Three comparisons will be done, using different load times and initial buffer concentration for the salt gradient. For the MatLab model, a tic and toc function will be added at the start and end of the script to produce a simulation time. Three simulations for each MatLab simulation are made to make an average.

3.4 Model separation

With Impact being a tool for reusable models in a larger system structure, the complete model implementation will be separated in a similar step-wise manner to smaller individual sub-models. The two parts that are in the complete model are the pump and the column.

3.4.1 Pump and inlet feeds

The code calculating the salt gradient that were added in 3.2.2 will be separated into a pump model. This will also contain all the code for the different phases and feed fractions of each component. To more resemble the setup in figure 1, three real input signals representing feed sample and buffers are added as separate models. The pump model will thus have three input signals and one output vector containing the concentrations of each component. The column model is adjusted to accommodate for the input vector containing the concentrations.

3.5 Addition of model elements

To mimic the physical model in figure 1, additional models were made. These include a mixing tank, a UV detector, tubing, valves and disentangled system data. The system data are models that control the pump or valves that would otherwise be tied to a computer, and component and feed data that is specific for the experiment.

3.5.1 Mixing, tubing and detector

A mixing tank model is created to separate the calculation of the feed fractions. The information containing the feed fraction data is further separated into a data box that will be connected to the mixing model.

Since tubes connect all the parts in figure 1, a general tubing model is created to facilitate the transport between each other model. They would only count as long empty space and the concentration profiles are greatly simplified to:

$$\frac{\partial c_i}{\partial t} = -\frac{v}{e}\frac{\partial c_i}{\partial z} = -\frac{v}{e}\frac{2c_i - 2c_{i-1}}{2h} \tag{17}$$

The first order approximation is 2-point backwards and the discretization size for the tubes is kept low at N=10.

A simple detector model is added using Beer-Lambert's law to calculate the absorbance of each component. An additional variable is added where all the absorbance profiles are added together to produce a more accurate chromatogram.

3.5.2 Valves and control systems

Valves and disentangled control system are created last to set up for more configurability. The valves will be simple models with input vectors and output vectors that change with if-equations controlled by the control systems.

The control systems do not contain any equations but only the control parameters that will be used by other models throughout the system. They are all disassociated from all other models but can be connected using dot-notation. The pump controls contain the phase times and salt gradient information. The valve controls contain valve position timings. Flow system contains the flow throughout the system. Feed composition contains the feed fractions. Component data contains the parameters used in the column and the absorptivity in the detector.

3.6 Setup variations

With all the parts in figure 1 created as separate models, different setups are now made available.

3.6.1 Recirculation

During recirculation a pooled sample of the first column passthrough is looped around and back to the column. The valve switching would be based of a test simulation and observing the chromatogram from the UV detector for a suitable time interval.

With the pooled sample separated, a new model named *delay_pool_sample* is added in which the user can define if the recirculated flow shall be unchanged, and thus separated going into the column again, or well mixed, going into the column as a pulse similar to the first feed injection. A system of integrators is added to determine the new fractions in the pooled sample and have to be manually added into *delay_pool_sample*.

To reduce the number of models in the workspace additional code is added to the pump to produce a second gradient during the recirculation. The if-equation determining the salt concentration gets additional conditions to determine when the recirculation occurs and corresponding control parameters are added to the pump controller.

3.6.2 Two columns

As a result of the recirculation configuration the two-column system looks incredibly similar. The second column has changed internal parameters using the data from another sub-project, see Appendix D. With the results from 3.6.1, the system of integrators was deemed superfluous and removed to improve simulation time.

4 Results

4.1 Small simplified tank series

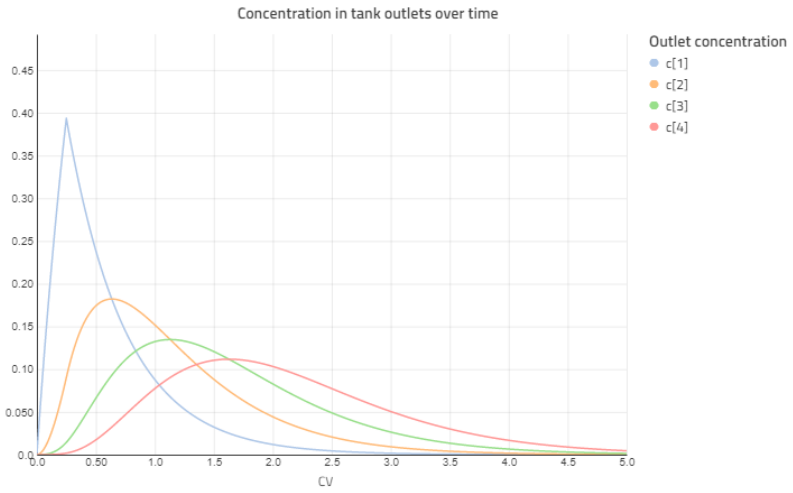


Figure 7: The concentration profile of each of the four tanks over time. A concentration pulse is simulated in the first tank and is dispersed over time in the tank series.

```
der(q[i])=r[i];
r[i] = k_kin * (H*c[i]*(1-(q[i]/qmax))-q[i]);
der(c[i]) = (Flow/V)*(c[i-1]-c[i]) + ((1-e)/e)*r[i];
```

Figure 8: Code describing the general balance over a tank.

The first complete step-wise simulation was a tank series with four tanks and only a single component. The small concentration pulse can be seen in figure 7 as an early spike in line c[1] and then declines over time. Likewise, one can see that with each tank, the concentration tends to reach its highest peak early and then decline over time. This gives an indication of a successful first implementation. Simulation code for a general tank is seen in figure 8.

4.2 Addition of salt gradient and system phases

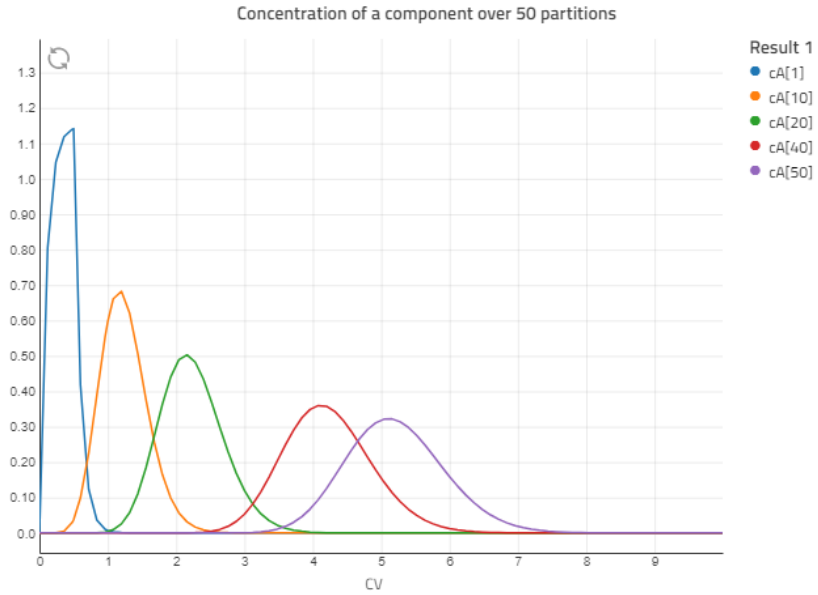


Figure 9: Concentration profile over time in five different partitions in the simulation series: 1st, 10th, 20th, 40th and 50th partition.

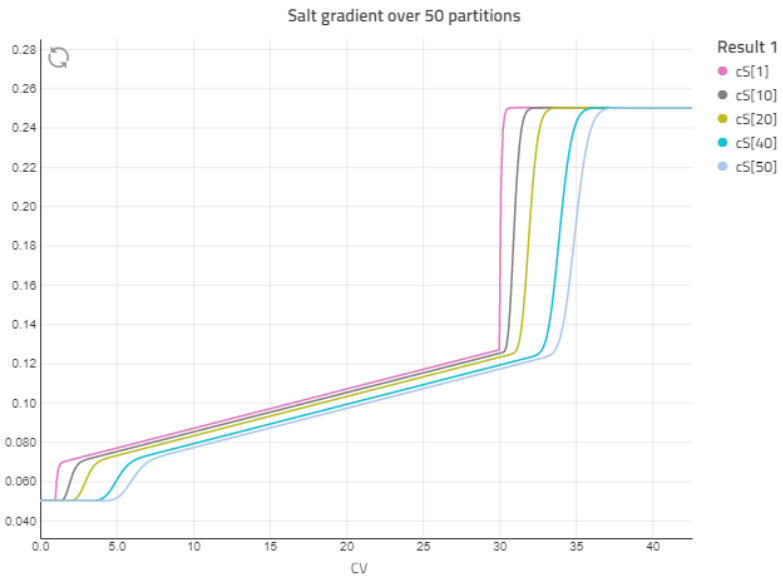


Figure 10: The salt concentration profile over time in five different partitions in the simulation series: 1st, 10th, 20th, 40th and 50th partition.

```
for i in 2:N loop
    der(cS[i]) = (Flow/V)*(cS[i-1]-cS[i]);
    qterm[i] = max(0.01,1-((qA[i])/qmax));
    HA[i] = H0A*cS[i]^(-betaA);
    der(qA[i])=rA[i];
    rA[i] = k_kinA * (HA[i]*cA[i]*qterm[i]-qA[i]);
    der(cA[i]) = (Flow/V)*(cA[i-1]-cA[i]) + ((1-e)/e)*rA[i];
```

Figure 11: Part of code for simulating the concentration profiles in tanks 2 to N.

```
k = if CV >=Time_Wash and CV<Time_Grad then (CV-(1+Time_Load))*k_lut+Bstart else 0;

cS0 = if     CV <Time_Load                  then FeedDil
      elseif CV >=Time_Load and CV<Time_Wash then BufferA
      elseif CV >=Time_Wash and CV<Time_Grad then k*BufferB+(1-k)*BufferA
      else BufferB;
```

Figure 12: The code for simulating the salt gradient.

Looking at figure 9 and 10, the breakthrough curves for the simulated component and salt concentrations is different parts of the column and behave as expected. The phases of the salt gradient can be seen best at the line marked cS[1] and the expected dispersion of the curve is seen over time and between each partition. Part of the code for simulation is seen in figure 11 and 12. The component balances are still simplified in their structure but the salt gradient is fully implemented at this step compared to MatLab code in appendix B.

4.3 Multiple components, discretization and improved model implementation

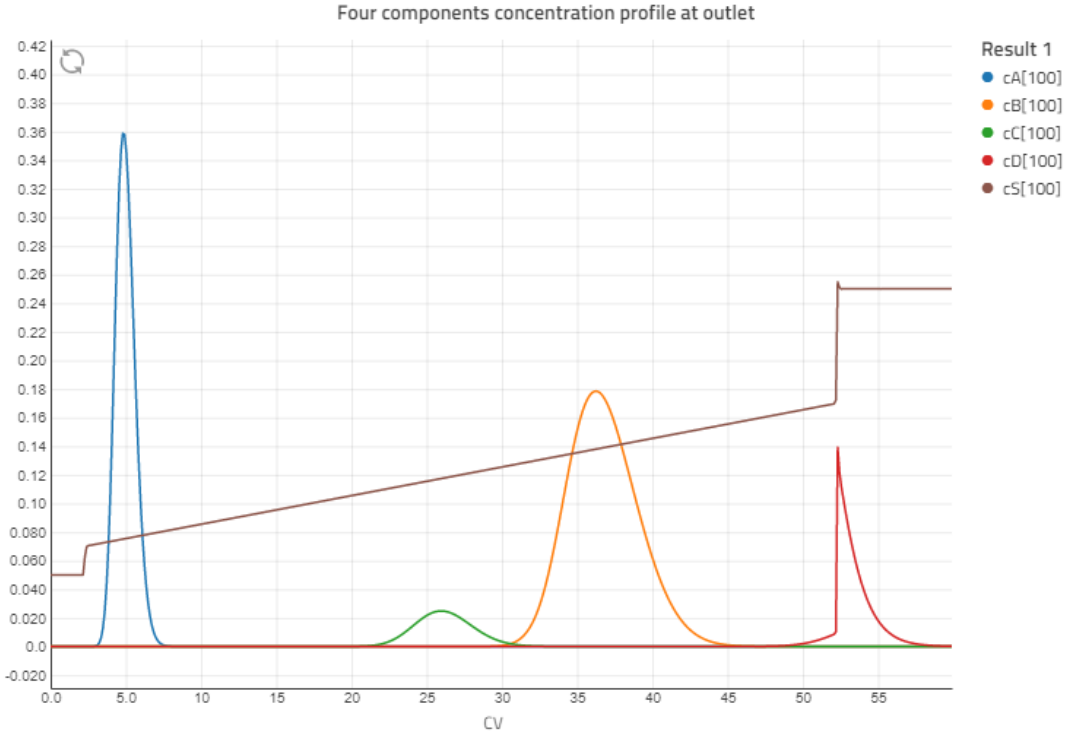


Figure 13: The concentration profiles over time for the salt and four components at the outlet of the column.

```
for i in 2:N-1 loop
//Salt
der(cS[i]) = DAX*(( cS[i+1]-2*cS[i]+cS[i-1] )/(h^2)) - (F_eps_A)*(( cS[i+1]-cS[i-1] )/(2*h));
qterm[i] = max(0.01,1-((qA[i]+qB[i]+qC[i]+qD[i])/qmax));
//Component A
der(cA[i]) = DAX*((cA[i+1]-2*cA[i]+cA[i-1])/(h^2))-(F_eps_A)*((cA[i+1]-cA[i-1])/(2*h))-Langmuir*rA[i];
der(qA[i])=rA[i];
rA[i] = k_kinA * (HA[i]*cA[i]*qterm[i]-qA[i]);
HA[i] = H0A*cS[i]^(-betaA);
```

Figure 14: Part of code for simulating the concentration profiles in tanks 2 to N-1.

In this step the model was further implemented with the diffusion, convection and adsorption terms, a discretization of the first and second order derivative was introduced and multiple (four) components in the feed. While figure 13 looks promising, the use of a 3-point central approximation for the first order derivative as seen in figure 14, shows oscillations at the curve fronts, especially visible on the salt concentration curve around 52 CV in figure 13. The component separations are clearly visible and separated as expected based on previous simulations in MatLab.

4.4 Further improving discretization and final implementation

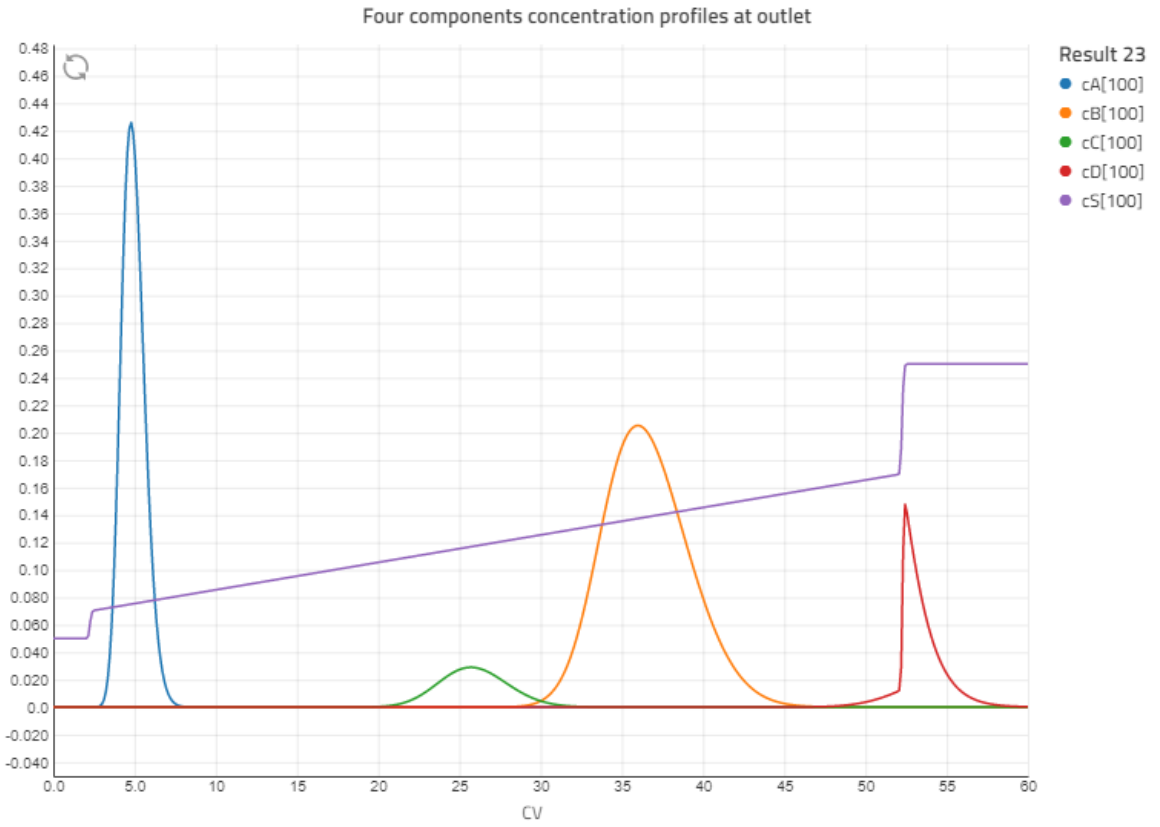


Figure 15: The concentration profiles over time for the salt and four components at the outlet of the column.

```
//First partition
      der(cS[1]) = DAX*(( cS[2]-3*cS[1]+2*cS0 )/(h^2)) - (F_eps_A)*(( 2*cS[1]-2*cS0 )/h);
      qterm[1] = max(0.01,1-((qA[1]+qB[1]+qC[1]+qD[1])/qmax));
      //Component A
      HA[1] = H0A*cS[1]^(-betaA);
      der(qA[1]) = rA[1];
      rA[1] = k_kinA * (HA[1]*cA[1]*qterm[1]-qA[1]);
      der(cA[1]) = DAX*(( cA[2]-3*cA[1]+2*c0A )/(h^2)) - (F_eps_A)*(( 2*cA[1]-2*c0A )/h) - Langmuir*rA[1];
```

Figure 16: Code for the first partition in the column. Only showing the salt and one component.

```
//Second to N-1 partition
  for i in 2:N-1 loop

      der(cS[i]) = DAX*(( cS[i+1]-2*cS[i]+cS[i-1] )/(h^2)) - (F_eps_A)*(( cS[i]-cS[i-1] )/h);
      qterm[i] = max(0.01,1-((qA[i]+qB[i]+qC[i]+qD[i])/qmax));
      //Component A
      der(cA[i]) = DAX*(( cA[i+1]-2*cA[i]+cA[i-1] )/(h^2)) - (F_eps_A)*(( cA[i]-cA[i-1] )/h) - Langmuir*rA[i];
      der(qA[i])=rA[i];
      rA[i] = k_kinA * (HA[i]*cA[i]*qterm[i]-qA[i]);
      HA[i] = H0A*cS[i]^(-betaA);
```

Figure 17: Code for the second to (N-1)th partition of the column. Only showing the salt and one component.

```
//Final partition
      der(cS[N]) = DAX*(( cS[N-1]-cS[N] )/(h^2)) - (F_eps_A)*(( cS[N]-cS[N-1] )/(h));
      qterm[N] = max(0.01,1-((qA[N]+qB[N]+qC[N]+qD[N])/qmax));
      //Component A
      der(cA[N]) = DAX*(( cA[N-1]-cA[N] )/(h^2)) - (F_eps_A)*(( cA[N]-cA[N-1] )/(h)) - Langmuir*rA[N];
      der(qA[N])=rA[N];
      rA[N] = k_kinA * (HA[N]*cA[N]*qterm[N]-qA[N]);
      HA[N] = H0A*cS[N]^(-betaA);
```

Figure 18: Code for the final partition of the column. Only showing the salt and one component.

The final implementation was a changed discretization to the 2-point backwards approximation for the first order derivative and removed the previously seen oscillations, compare figure 13 and 15. The equation code for salt and one component are seen in figures 16-18, showing the first partition (inlet of the column), a general mid partition (middle of the column) and the Nth partition (outlet of the column). The result is now an identical implementation of the MatLab model into Impact.

4.5 Identical simulations in MatLab and Impact

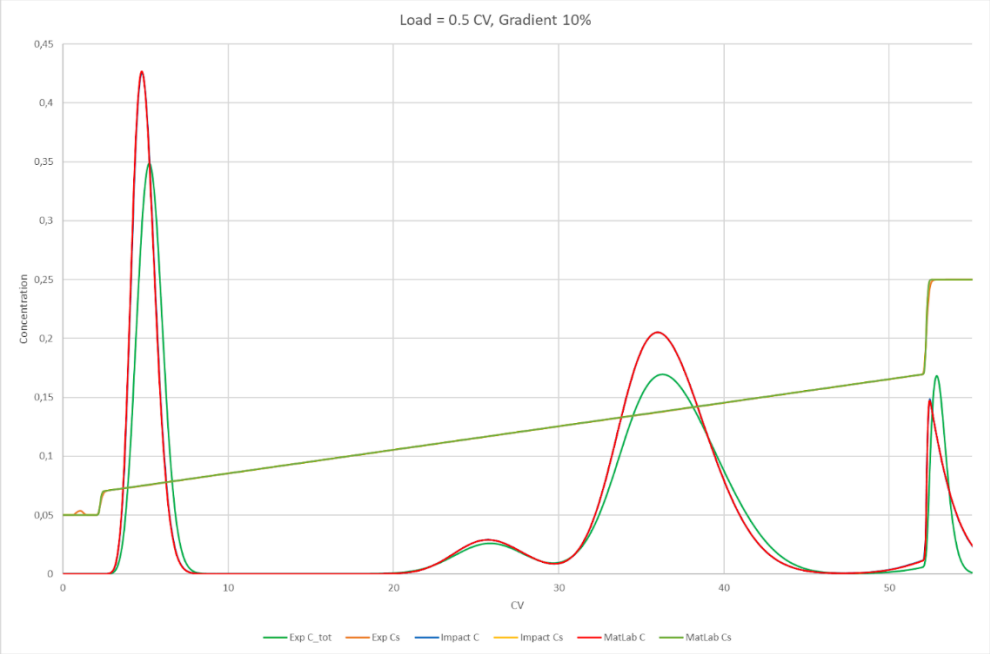


Figure 19.1: Comparison graph between experimental data, Impact model and MatLab model showing both the salt concentration and the cumulative concentration of the individual component concentrations.

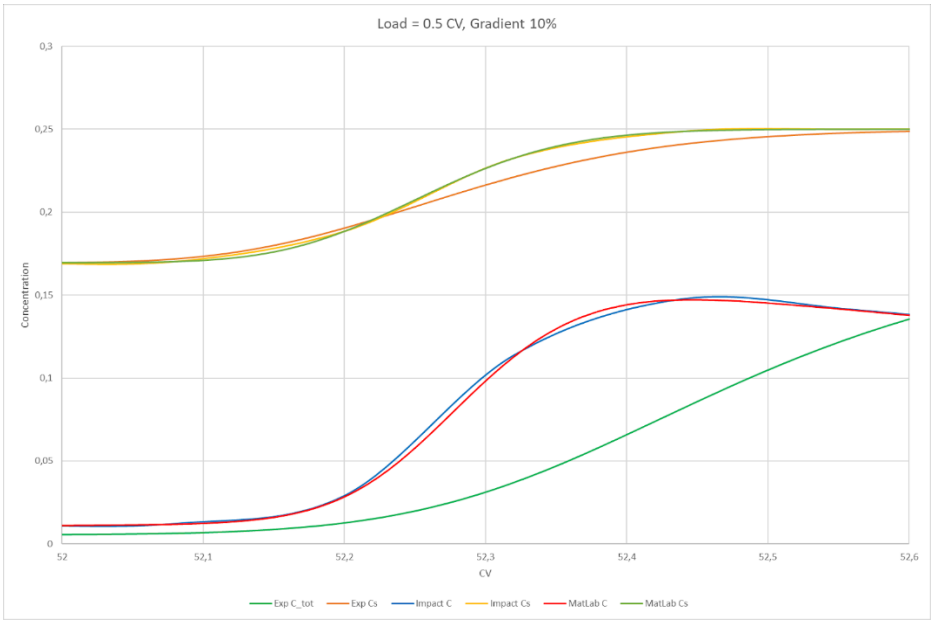


Figure 19.2: Zoomed in part of figure 19.1 highlighting a difference between simulations.

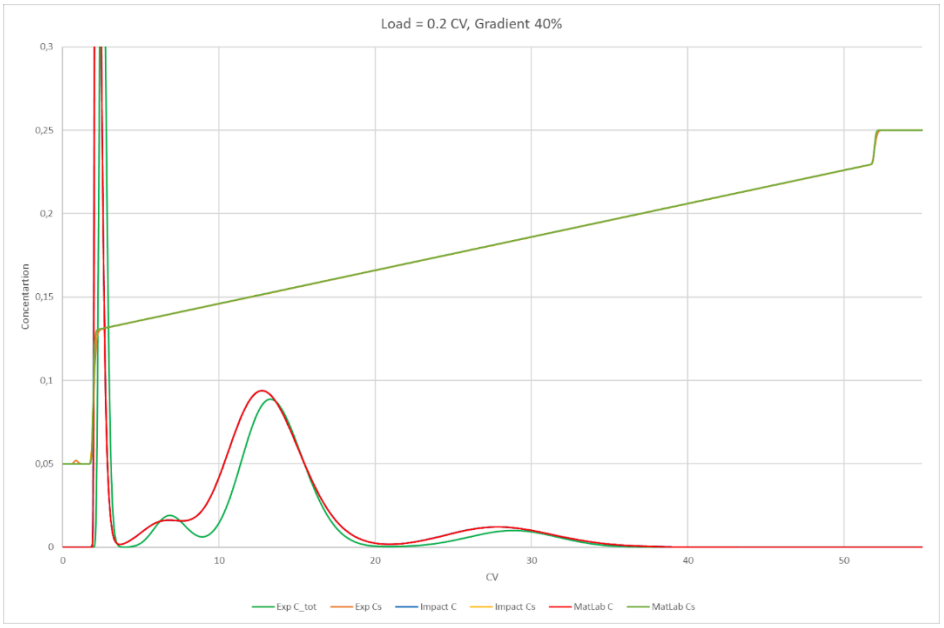


Figure 20.1: Comparison graph between experimental data, Impact model and MatLab model showing both the salt concentration and the cumulative concentration of the individual component concentrations

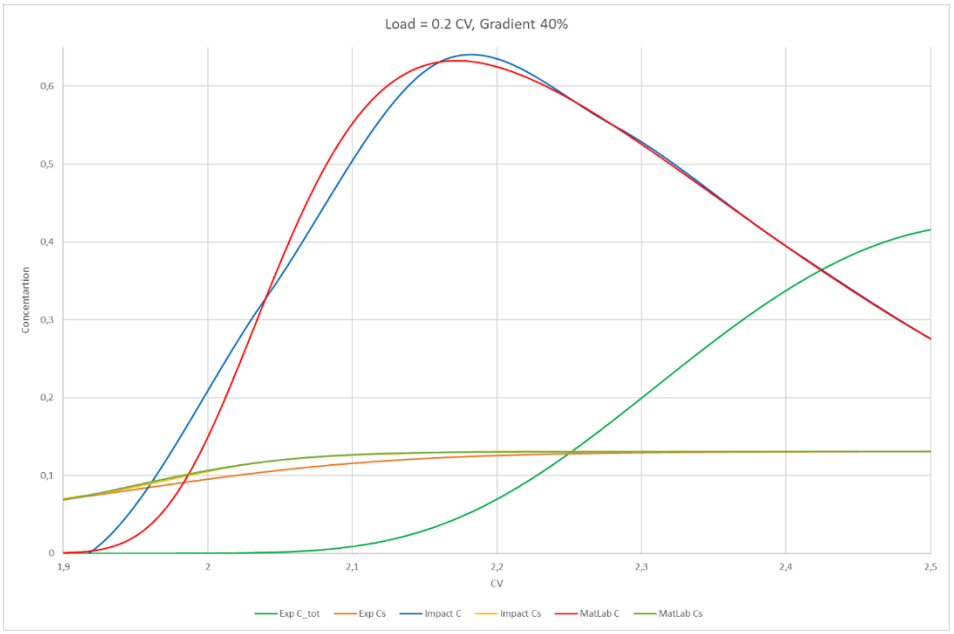


Figure 20.2: Zoomed in part of figure 20.1 highlighting a difference between simulations.

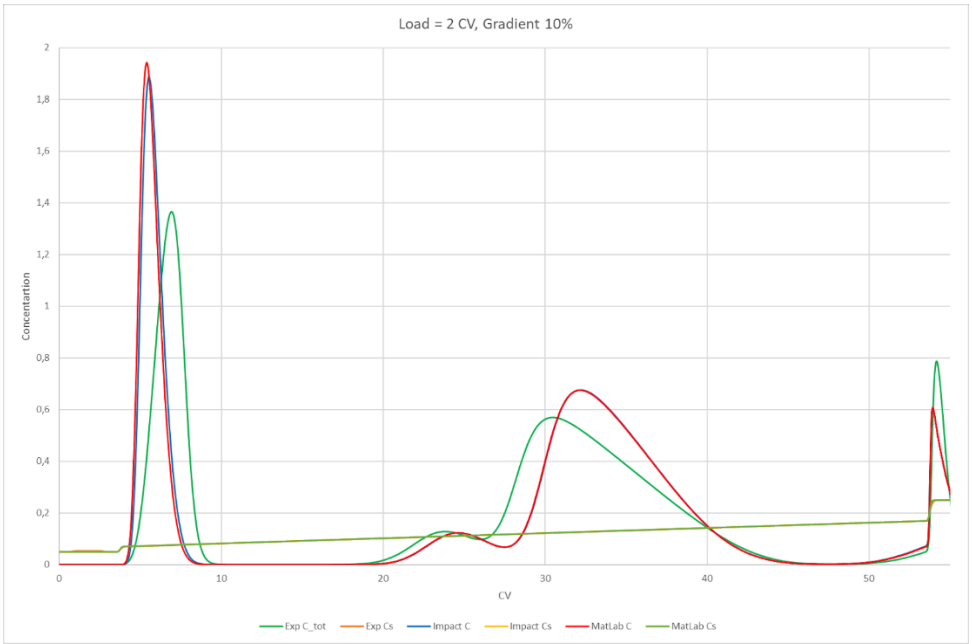


Figure 21.1: Comparison graph between experimental data, Impact model and MatLab model showing both the salt concentration and the cumulative concentration of the individual component concentrations

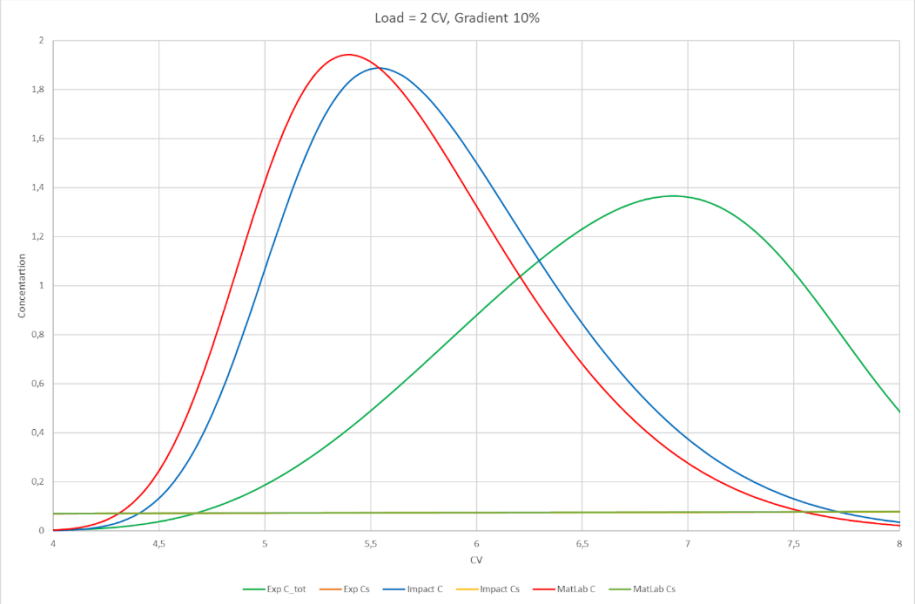


Figure 21.2: Zoomed in part of figure 21.1 highlighting a difference between simulations.

With the model implementation into Impact complete, identical simulations between Impact and MatLab could be done. In figures 19.1, 20.1 and 21.1, three different scenarios were run with varying load times of gradients: one with low load time and gradient percentage, one with low load time and high gradient percentage and one with high load time and low gradient percentage. Both Impact and MatLab were close to the experimental data, but since the component parameters are technically unknown the results are expectedly different.

Table 4: Simulation time in seconds for each environment and their corresponding load time and gradient percentage.

Simulation	Low load Low gradient	Low load High gradient	High load Low gradient
1 (MatLab)	0.9606 s	0.9870 s	1.3454 s
2 (MatLab)	0.9470 s	1.2232 s	2.7928 s
3 (MatLab)	0.9279 s	1.6772 s	2.7429 s
Impact	2.9075 s	2.7529 s	4.8418 s

The simulations times for the results in figures 19.1, 20.1 and 21.1 are found in table 1. Impact had a longer simulation time during all tests but were consistently around the same time (not shown) while MatLab exhibit a larger interval of required time, especially at high load.

There are some visual differences between the two simulated models that are highlighted in figures 19.2, 20.2 and 21.2. Both simulations are most of the time incredibly accurate between each other but at large changes in the concentration derivatives and the higher the feed load is where the largest change can be observed, as seen in figure 21.2. Both models use a variable step size solver, with Impact resulting in 506 steps and 2523 steps in MatLab for the simulation in figure 21.1. Forcing the Impact solver to take close to as many steps at MatLab did not change the results.

4.6 Separation of pump into a new model and connection

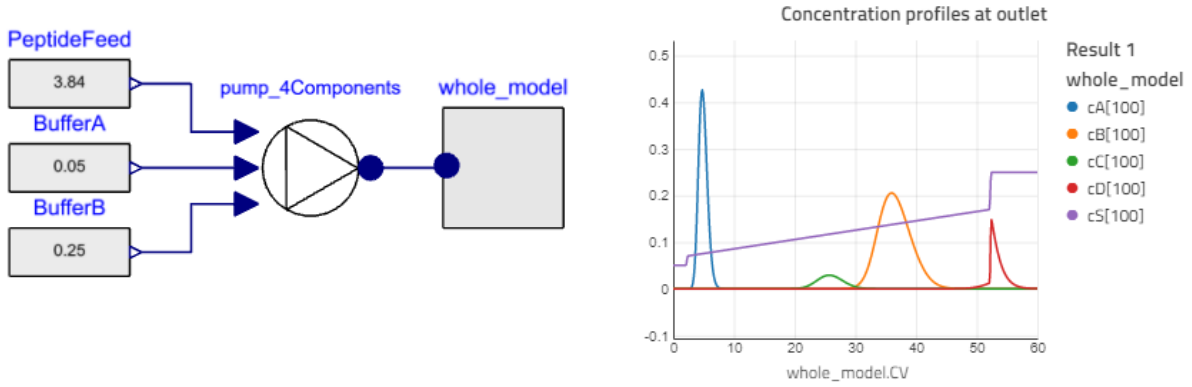


Figure 22: Schematics for the first model separation, separating the pump, buffer and sample from the rest of the model. The concentration profiles for salt and components at the column outlet are simulated and are identical to figure 15.

The first step of separating the model was lifting out “the pump code” into its own model. A vector output connector on the pump model was added that would contain the salt concentration and the feed concentrations. A corresponding vector input connector on the remainder of the model was added and the code was changed to adhere to the new model structure. To further generalize the new pump model, three real value input connectors were added to correspond to the bottle containing the sample feed and the two different buffers. These were in turn connected to single real output blocks and connected to the pump. The complete system and with a simulation result is shown in figure 22. The simulation produced an identical result to that of the unseparated model as previously seen in figure 15.

4.7 Addition of mixing, tubing and detector

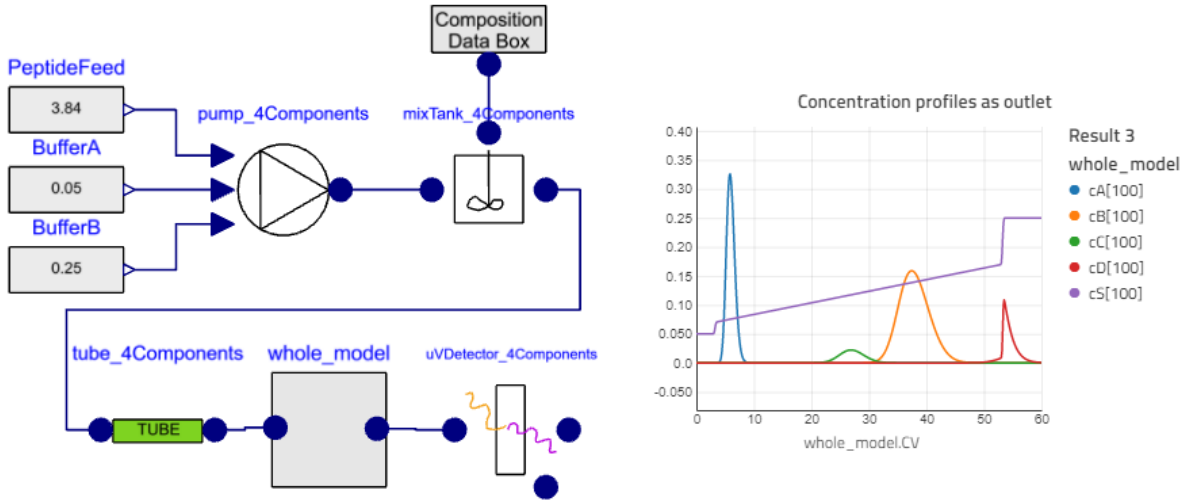


Figure 23: Schematics over a more generalized system. Mixing tank, a separate composition data box containing feed fractions, a tube for liquid transportation and a UV detector are constructed and connected using vector connectors. The concentration profiles for salt and components at the column outlet are simulated and seen on the right.

The mixing tank, data box, tube and UV detector models were implemented smoothly and were all connected using Modelicas vector connectors as shown in figure 23. The simulation continued to work as intended, with a slight change in the behavior of the concentration profiles. This was expected because of the added tubing which dispersed the flow slightly before entering the column simulation in *whole_model*. The composition data box worked as intended, but with the added vector connector the model were not as generalized as it could further be and were later solved in 4.8.

4.8 Addition of valves and separate control systems

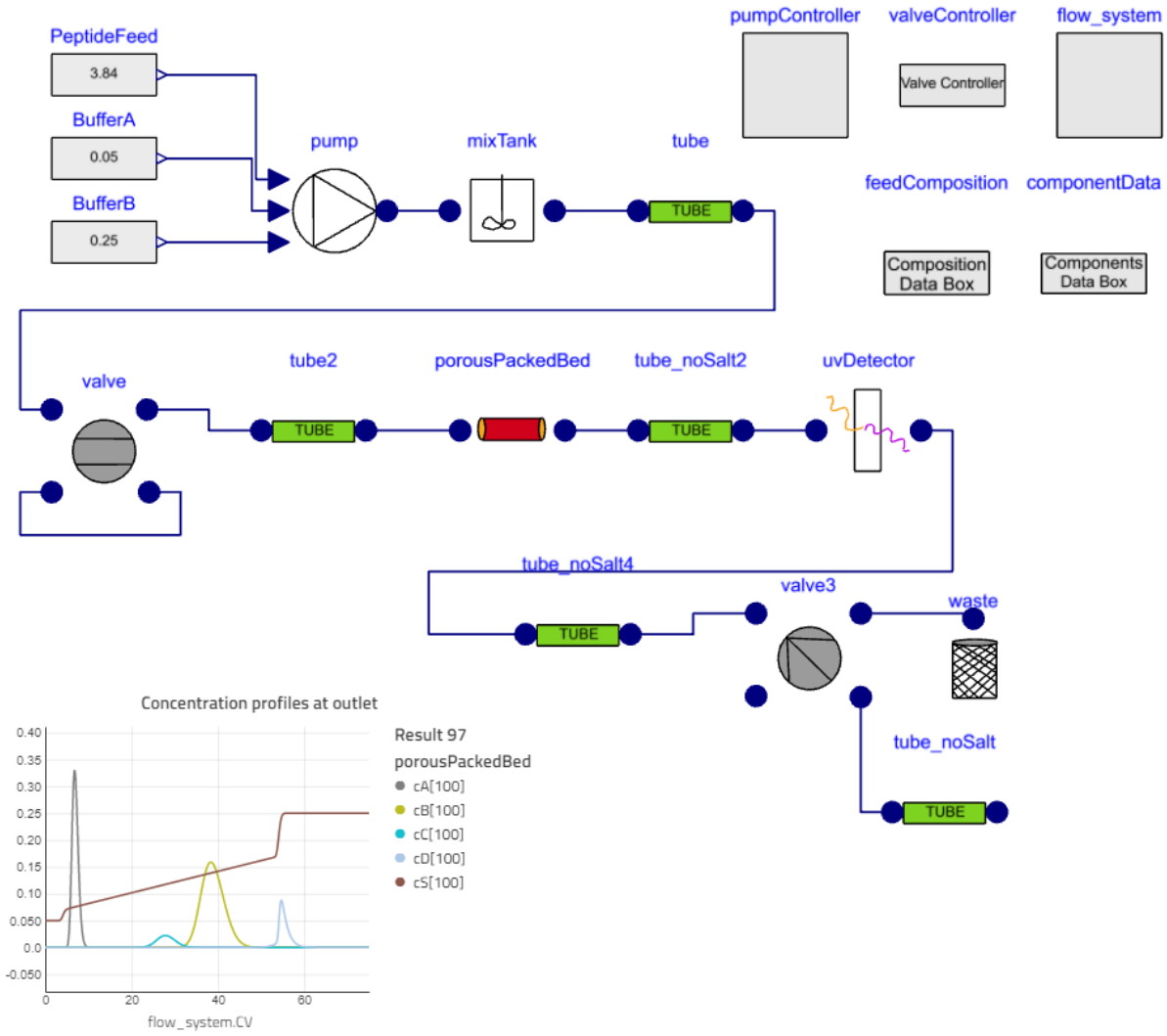


Figure 24: Schematics for a fully separated model with the inclusion of separate controllers, tubing, detector and valves. A concentration profile for all components in the column outflow is also pictured.

This final model schematics shown in figure 24 resembles the one shown in figure 1 with a few differences: the inlet valve is part of the pump model and the column flow does not go back into the column valve. The first difference was deemed mathematically simpler and thus kept pump and injection valve as one model. The second difference could have been connected back to the column valve, but deemed redundant since the flow would be led towards the detector anyways.

The control systems were all disentangled from the other models that now used dot-notation to import the parameters used throughout the system. This improved the useability of the system as parameter were kept at intuitive places. While the simulation is continuedly successful, the increasing amount of tubing in the model start to become more evident with the widening of peaks and longer time for the breakthrough curve peak to be reached.

4.9 Recirculation

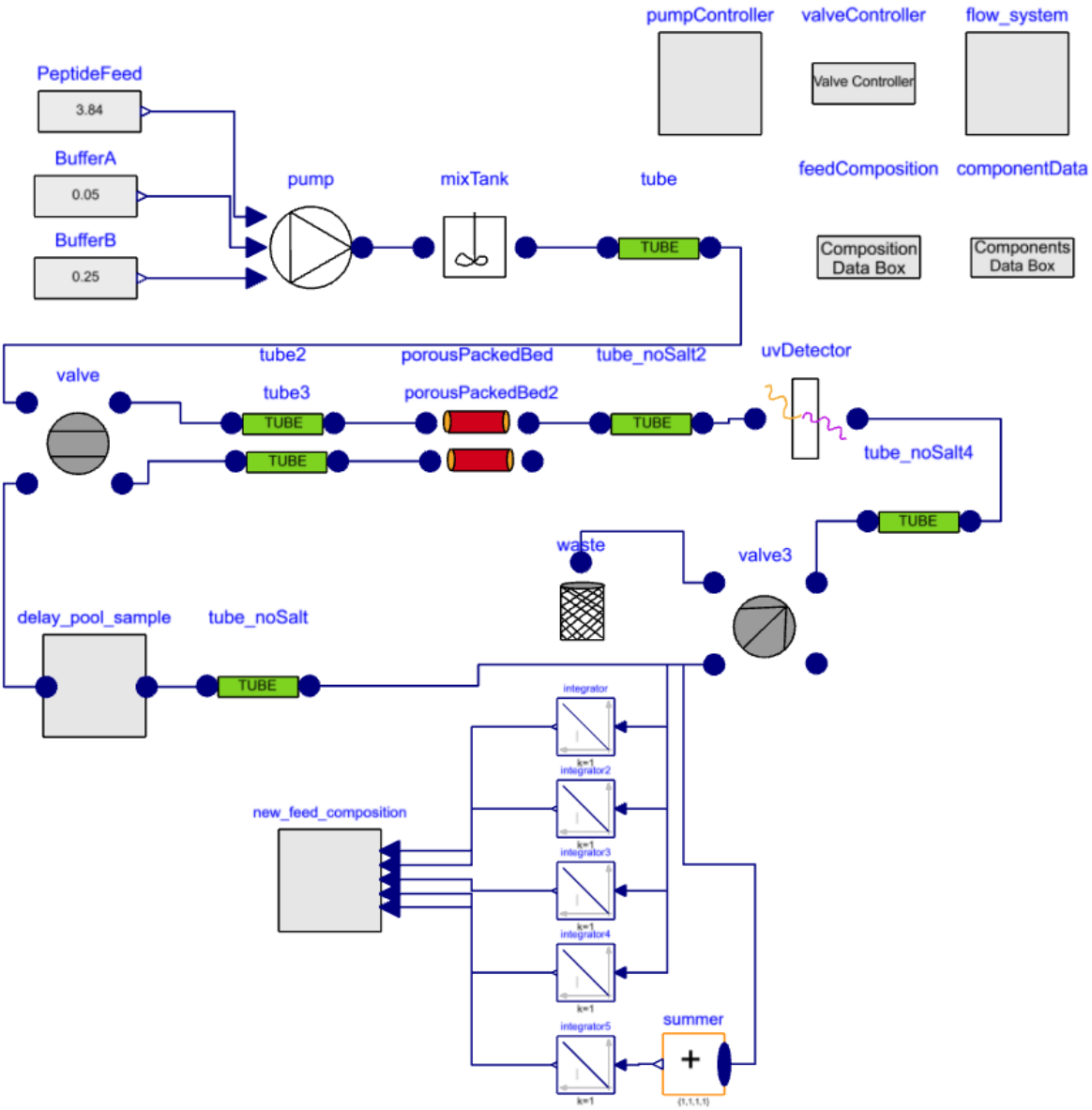


Figure 25: Schematics over the recirculation model. A pooled sample is separated at 'valve3' and fed back to 'valve'. Depending if the recirculated flow would be fully mixed or as separated as the outflow 'valve3', an additional model 'delay_pool_sample' is introduced. Alongside the integrators and the model 'new_feed_composition' the outflow from 'delay_pool_sample' can be determined. A separate column for recirculation is used due to simulation difficulties.

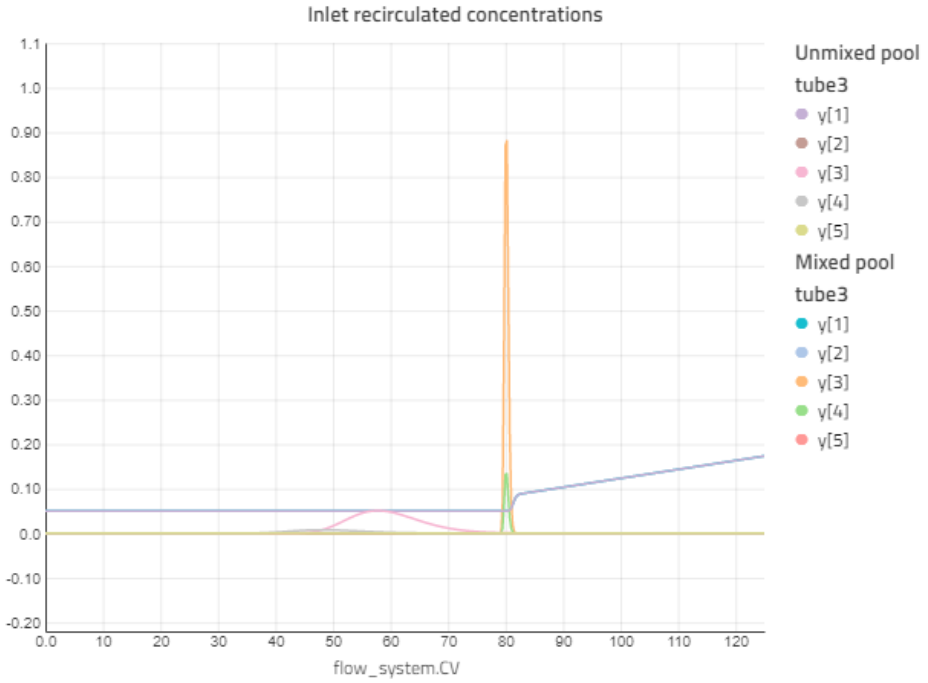


Figure 26: The recirculated pooled sample before recirculating into the column again. One is the unmixed pool sample directly from the first column and one is a well-mixed feed pulse.

Figure 27: The recirculated pooled sample after recirculating into the column again. One is the unmixed pool sample directly from the first column and one is a well-mixed feed pulse, yielding identical results.

With the introduction of recirculation some modifications had to be made. At outlet valve *valve3* in figure 25, there is a change of flows depending on the pooling of the feed. The valve can either be set to switch flow paths depending on time instructions or a UV cutoff. Because UV would indiscriminate between the different component and thus unhelpful for pooling of a

certain sample, the time instructions were used. The times for pooling were determined by running the setup once, look at the chromatogram and decide the start and stop times from there.

To verify that a well-mixed pooled stream would matter or not, an additional system were created. These consisted of integrators that would calculate the fraction of each component in the pooled stream and stored in the new model *new_feed_composition*, as seen at the bottom of figure 25. Another new model, *delay_pool_sample* were also created to test this new system that would use either the unmodified pooled stream or create a new stream based on the fractions in *new_feed_composition*. The looped stream was also delayed to ensure the first run-through of the column is complete. The unchanged and the well-mixed recircled streams in the inlet of the second (recirculation) column can be seen in figure 26 and the outlet streams in figure 27. In figure 26, the profiles are very different, but as seen in figure 27 this new system added nothing but extra manual work.

Because Modelica and Impact does not tolerate variable overlap, the recirculated stream had to be fed through a separate column. All the parameters between the column models were kept the same to resemble a recirculation model.

4.10 Two columns

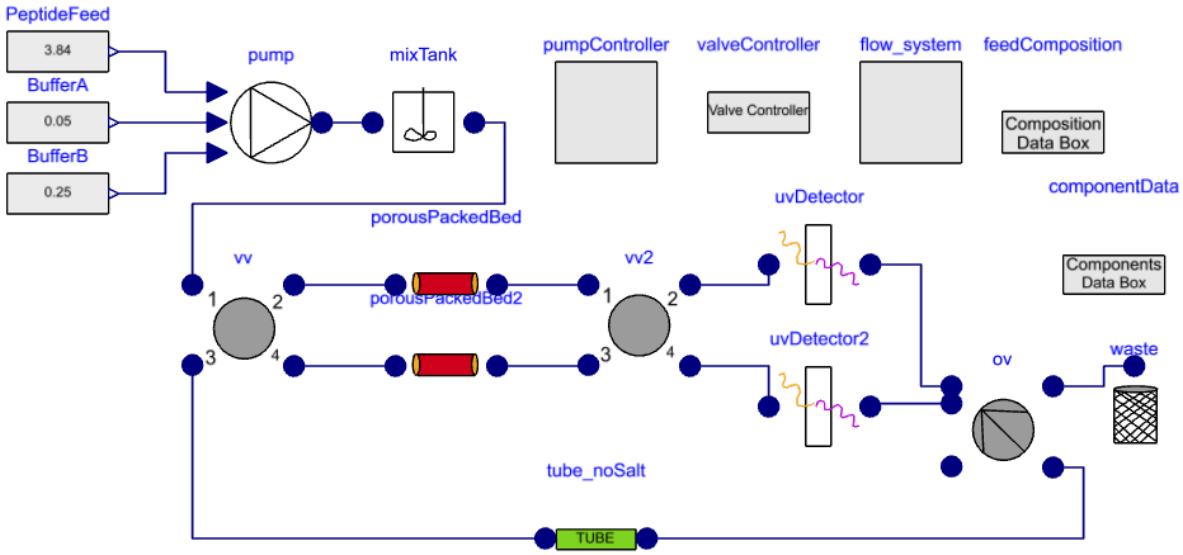


Figure 28: Schematics for a two column setup.

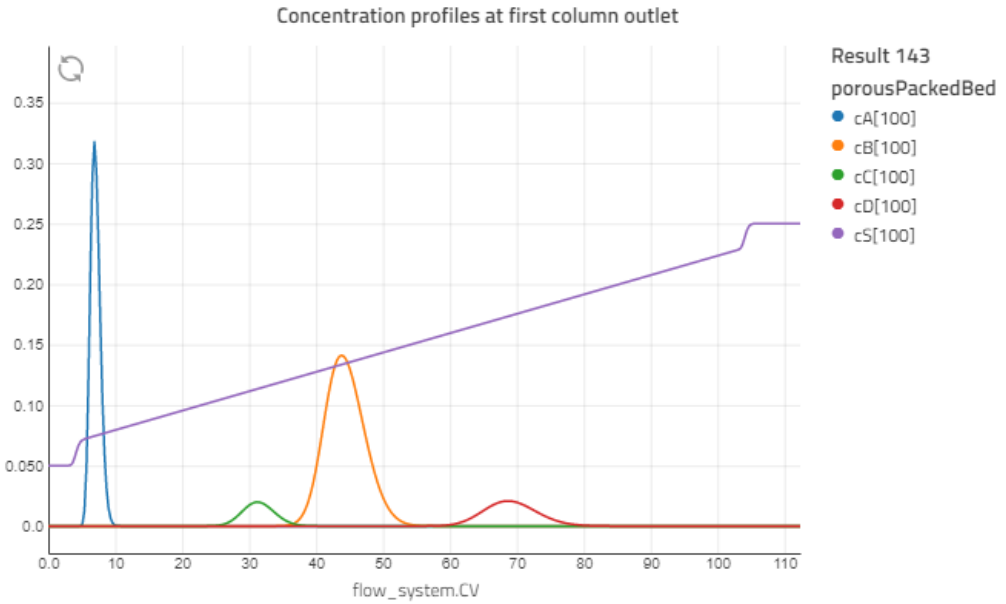


Figure 29: The concentration profiles over time for the salt and four components at the outlet of the first column.

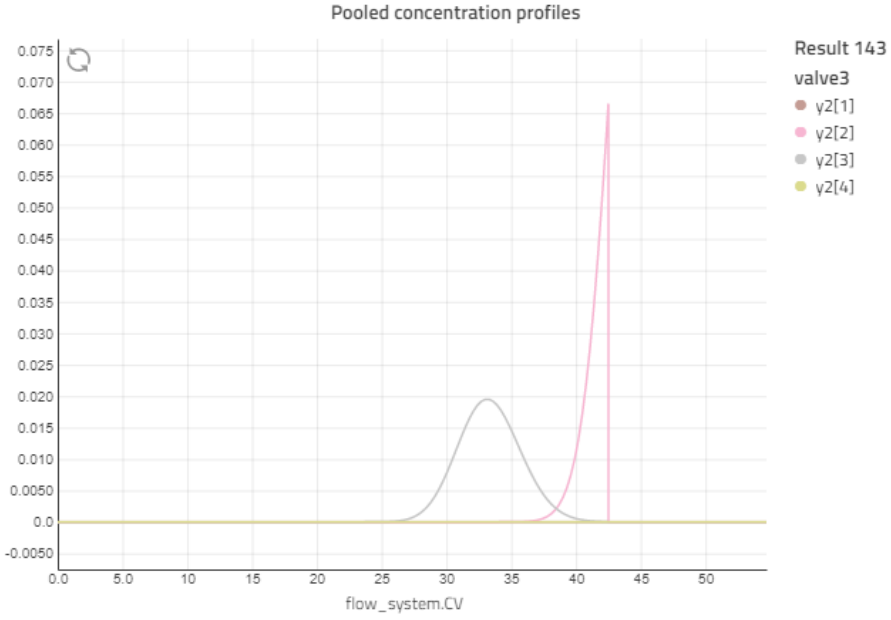


Figure 30: The pooled partition flowing out from valve3 (or ov) based on the profiles in the UV detector between 25 and 42.5 CV.

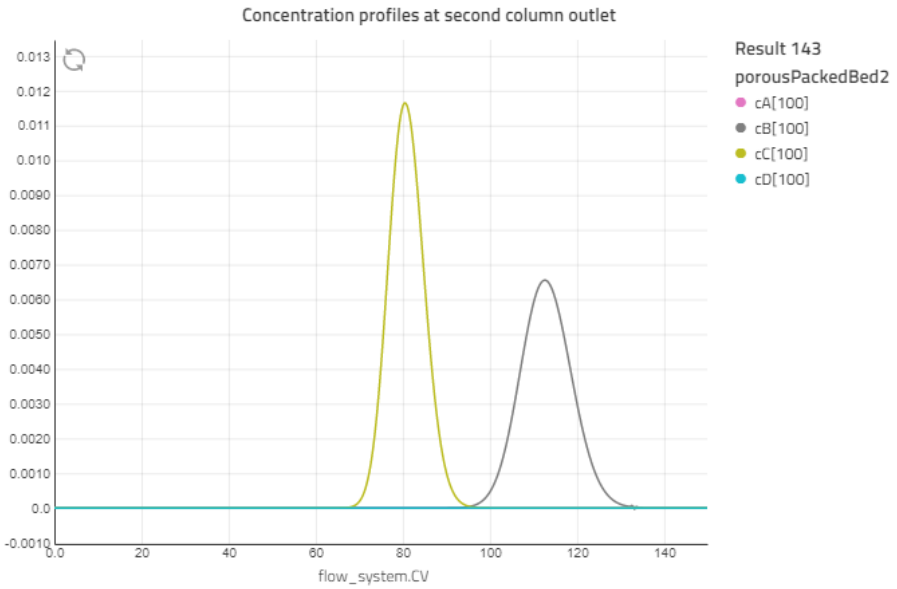


Figure 31: The concentration profiles over time for the four components at the outlet of the second column. Since the pool almost exclusively contained to components, component A and D are almost zero.

The two column schematic is similar to the recirculation as seen in figure 28. The *delay_pool_sample* model, *new_feed_composition* model with its surrounding integrators proved to be useless and removed. Because the tube model had been shown to work as intended, the amount were reduced to one in this system to makes is more legible.

The second column used slightly different parameters and were based on another sub-projects data found in Appendix D. The first column produced the concentration profiles in figure 29 and were pooled between 25 and 42.5 CV as seen in figure 30. The pooled stream was led through the second column and produced the separation seen in figure 31. While the stop time for pooling could have been chosen earlier for no component B in the stream, that would have resulted in loss of component C. A new pool can now be chosen between 60 and 93 CV for almost no loss of component C.

5 Discussion

The implementation had its ups and down. While the step-wise implementation was a wise choice there arose some confusions regarding simulation time. The previous project in MatLab simulated time in CV, which in contrast was done in seconds by default with Impacts built-in time variable. This had to be converted into CV each time an equation was based on time. This, in turn, revealed previously unnoticed errors in the MatLab model, where some parameters were expressed in time but had yet to be converted into CV. These errors and complications came to light after the full implementation had been done in results 4.4. When the comparison between the two models were done there were still differences and the hunt for errors began. The correct implementation is the only visible in Results 4.5 and the code were retroactively changed in previous Impact models to accurately reflect a correct simulation.

The simulation in MatLab and Impact are both near identical as can be seen in figures under Results 4.5, but there were some slight differences when zoomed in. These differences became more pronounced at high load times or with a high salt derivative. A possible solution could be the differences between the step-size between the solvers and the amount of steps taken. MatLab took between 2000-5000 steps for its simulation while Impact only took around 500 steps.

The experimental data were much different from the simulations, but no larger meaning should be considered here. The data provided for the sub-projects in Process Simulation are only approximative and mostly illustrative for the purposes of the exercise.

A big hurdle was the solution to the recirculation implementation. Looping back a pooled sample proved difficult since the pooled sample returned back to the original loop before the first pass-through had finished. I found the easiest solution to be the addition of an identical column next to the original, one taking the first separation and the other taking the pooled sample. At first, a second pump had to be installed in order to generate a new gradient for the second column but eventually the pump was modified to accommodate for a recirculating feed. This solution with two columns meant that recirculation and two columns were more or less the same only that the second columns parameters were changed between the setups. A possible solution might have been to further develop the *delay_pool_sample* model and make the column model wait for the delayed sample to enter.

One aspiration was to make the amount of components in the feed as general as possible, utilizing arrays to better store variables and parameters. Two things were detrimental to this success. First were the vector connectors in Impact. The thought was to have an array generated based on parameter input (however many components were in the feed) and send the arrays throughout the system. This would result in many other arrays being required to be created, most of which would be empty. Second was the requirement to define many more components in advance. Since the parameters surrounding the components are predetermined that would require them to be defined in the code and unless they were used they would not be anything more than superfluous code. Both problems could surely be solved with time, but that might have made the code more illegible.

Modelon Impact is a powerful tool more than capable to simulate chromatographic processes. While this has been a relatively simple model implementation with a couple of components the perks of simulation is not obvious. The simulation time between MatLab and Impact were in most cases in favor of MatLab with around 1-3 second instead of Impacts 2-5 seconds. But this hinges on the use of sparse matrices in MatLab, which without would skyrocket the simulation time. Impact flattens the model and reduces superfluous code for simulating. What could prove beneficial is an increasing complexity of the model without the assumptions that might add many other differential equations for both systems and simulation time.

There are still untapped aspects of Modelica there were not attempted during this thesis project. One such aspect is the inStream operator that is designed to numerically describe quantities carried by matter flow. While the simulation might have been improved with a proper implementation of in the inStream operator, the simulation between the old code and this new implementation might become so different it would have been difficult to decern which simulation is preferred. If the aim of the thesis would have been to most accurately simulate a real concentration profile then inStream might have been a better option. This would probably also require less assumptions and use more complex domain equations.

The experiment mode is something that was rarely used during this thesis project. Since the mode is better tailored to test different parameters without changing the model code, it would be better suited for making parameter sweeps and find the best curve fit.

6 Conclusion

The overall implementation of the MatLab model into Impact went well. Some retroactive corrections had to be implemented into the MatLab code, discovered by simulating the two models and comparing them side by side. While the simulations are not exactly the same as evident in figures 19.2, 20.2 and 21.2, the majority of the curve is spot on and works without using sparse matrices. The separation of the model had varying results, working well in the end but have to accommodate for many different cases. Problem with reuse of column during recirculation make some cases maybe redundant, since it looks identical to the two-column setup, but as long as the parameters are correct then it does not really matter if one or two column models are simulated. Impact is indeed capable of chromatographic simulations and still have potential for further development, both by aspiring master thesis student and Modelon themselves that could implement an available library to all.

7 References

1. Modelon, *What is Modelon Impact*, (2022), accessed 2022-06-13 at: <https://help.modelon.com/latest/home/what_is_modelon_impact/>
2. Modelon, *Modelon Impact overview*, (2022), accessed 2022-06-13 at: <https://help.modelon.com/latest/application_overview/application_overview_long/>
3. Harrison R. Todd P. Rudge S. Petrides D., *BIOSEPARATIONS SCIENCE AND ENGINEERING*, second edition, (2015), pp. 245-247
4. Modelica Association (2021), *Modelica® – A Unified Object-Oriented Language for Systems Modeling Language Specification*, accessed 2022-06-13 at: <https://modelica.org/documents/MLS.pdf>
5. MathWorks Help Center, *Ode15s*, accessed 2022-06-13 at: <https://se.mathworks.com/help/matlab/ref/ode15s.html?s_tid=doc_ta>
6. Andersson C. Führer C. Åkesson J., *Assimulo: A unified framework for ODE solvers*, (2015), accessed 2022-06-13 at <https://jmodelica.org/assimulo/ODE_CVode.html>

8 Appendix

8.1 Appendix A – Data from case study 2A

Experimental conditions

pH 8

Buffer A:	0,05 M	NaCl
Buffer B:	0,25 M	NaCl
Feed (diluted)	0,05 M	NaCl
Feed conc (diluted)	3,84 g/L	(total peptide)
Bed height	10 cm	
flow	30 CV/h	
velocity	5 cm/min	
Resin	Source 30Q	
Particle diameter	30 μm	

Feed comp (g/g)	A	29,9%
	B	53,5%
Feed is diluted 1+9	C	6,1%
with buffer A	D	10,5%

Gradient runs

run no.	Bstart	Bend
1	0	50
2	10	60
3	20	70
4	30	80
5	40	90
6	50	100

Load 0,5 CV
The gradient length if 50 CV
And is preceeded by 1 CV wash with 100% buffer A
Regeneration is 5 CV of 100 % B

Load runs

run no.	Feed volume (CV)
1	0,5
2	1
3	2
4	5
5	10

Gradient start is 10 % B
Gradient end is 60 & B
With a gradient length of 50 CV

PHYSICAL PARAMETERS

Packing parameters: column void, $\varepsilon_c = 0.45$, and packing porosity, $\varepsilon_p = 0.57$, and assume a particle Peclet to be constant, Pe = 0.5.

Adsorption parameters: Langmuir kinetic model with mobile phase modulator of salt conc.

EXPERIMENTAL DATA

Experimental data consists of 6 experiments with a low loading and varying gradients, and 5 with a higher load. The extinction coefficient is 1.0 L/g/cm for all species and the UV cell path is 1 cm B

Para.\Comp.	A	B	C	D	
H_0	0.6e-4	8.5e-4	2.97e-4	9e-4	M^{β}
β	4.37	5.17	5.14	5.93	
q_{max}	70	70	70	70	g/L
k_{kin}	300	75	90	42	h^{-1}

8.2 Appendix B – MatLab code

```
p.Bstart = 0.1;          %   0, 0.1, 0.2, 0.3, 0.4, 0.5   vid tstart=0.5
tstart = -2.0;           %-0.5,  -1,  -2,  -5,  -10       vid Bstart=0.1
p.time.Wash = 1;
p.time.GradL = 50;
p.time.Grad = p.time.GradL + p.time.Wash;
p.col.Flow =30/60;         %CV/min


%mesh and derivative approximation
N = 100;
A1type = '2pb';
A2type = '3pc';


%Component data
p.qmax = 70;


%Data enligt angivelse 2A
p.A.H0 = 0.6e-4;
p.A.beta = 4.37;
p.A.kkin = 300/60/p.col.Flow;
p.B.H0 = 8.5e-4;
p.B.beta = 5.17;
p.B.kkin = 75/60/p.col.Flow;
p.C.H0 = 2.97e-4;
p.C.beta = 5.14;
p.C.kkin = 90/60/p.col.Flow;
p.D.H0 = 9e-4;
p.D.beta = 5.93;
p.D.kkin = 42/60/p.col.Flow;


%Packing parameters
p.pack.ec = 0.45;
p.pack.ep = 0.57;
p.pack.Pe = 0.5;
p.e = p.pack.ec + (1-p.pack.ec)*p.pack.ep;


%exp conditions - pH 8
p.col.BufferA = 0.05;
p.col.BufferB = 0.25;
p.col.FeedDil = 0.05;


%Feed composition
p.col.FeedPep = 5;
p.FeedCompA = 0.299;
p.FeedCompB = 0.535;
p.FeedCompC = 0.061;
p.FeedCompD = 0.105;


p.col.L = 10e-2;
p.col.v = 5e-2/p.col.Flow;
p.col.dp = 30e-6;


p.Dax = p.col.v * p.col.dp / p.pack.Pe;
h = p.col.L/N;
p.k_lut = 0.5/50 * 1.0;
p.lang = (1-p.pack.ec)/p.e;
p.FeA = p.col.v/p.e;


T_pre_matris = toc(T_start)


%Matriser
[A1,A1f]=FVMdisc1st(N,h,A1type,'sparse');
[A2,A2f]=FVMdisc2nd(N,h,A2type,'sparse');
[B1,B0]=FVMdiscBV(N,h,[0 1],[1 -1; 0 0],'sparse');


%Tot-matriser
AT = p.Dax*(A2 + A2f*B1) - p.FeA*(A1+A1f*B1);
BT = p.Dax*A2f*B0 - p.FeA*A1f*B0;


%Sparse Jacobian
S = FVM_Jpattern(@(t,y) colmodel(t,y,N,p,AT,BT),9*N);
option = odeset('JPattern',S,'reltol',1e-6);


%Initial values
yinit = [0.05*ones(N,1);zeros(8*N,1)];
tend = p.time.Grad + 5;
```

```
tspan = [tstart 56];


T_pre_solve = toc(T_start)


% Solve
[t,y] = ode15s(@ (t,y) colmodel(t,y,N,p,AT,BT),tspan,yinit,option);
function dydt = colmodel(t,y,N,p,AT,BT)
% Inputs
cS = y(1:N);
cA = y(N+(1:N));
cB = y(2*N+(1:N));
cC = y(3*N+(1:N));
cD = y(4*N+(1:N));
qA = y(5*N+(1:N));
qB = y(6*N+(1:N));
qC = y(7*N+(1:N));
qD = y(8*N+(1:N));


%Salt gradient
if t < 0
    s = p.col.FeedDil;
    C = p.col.FeedPep;
elseif t < p.time.Wash
    s = p.col.BufferA;
    C = 0;
elseif t < p.time.Grad
    k = (t-1)*p.k_lut + p.Bstart;
    s = k*p.col.BufferB + (1-k)*p.col.BufferA;
    C = 0;
else
    s = p.col.BufferB;
    C = 0;
end
%Salt profile
dcS = AT*cS+BT*s;


%Concentration for each compound
CAin = C * p.FeedCompA; %g/L
CBin = C * p.FeedCompB; %g/L
CCin = C * p.FeedCompC; %g/L
CDin = C * p.FeedCompD; %g/L


%Henry!
HA = p.A.H0 * cS.^(-p.A.beta);
HB = p.B.H0 * cS.^(-p.B.beta);
HC = p.C.H0 * cS.^(-p.C.beta);
HD = p.D.H0 * cS.^(-p.D.beta);


%Rates
qterm = (1-(qA+qB+qC+qD)./p.qmax);
rA = p.A.kkin *(HA.*cA.*qterm-qA);
rB = p.B.kkin *(HB.*cB.*qterm-qB);
rC = p.C.kkin *(HC.*cC.*qterm-qC);
rD = p.D.kkin *(HD.*cD.*qterm-qD);

% Derivatives
dcA = AT*cA+BT*CAin - p.lang * rA;
dcB = AT*cB+BT*CBin - p.lang * rB;
dcC = AT*cC+BT*CCin - p.lang * rC;
dcD = AT*cD+BT*CDin - p.lang * rD;


% Output
dydt = [dcS;dcA;dcB;dcC;dcD;rA;rB;rC;rD];
end
```

8.3 Appendix C – Discretization

2-point-backward approximation for the first derivative: $\frac{dc}{dz} = \frac{c_i - c_{i-1}}{h}$

3-point-central approximation for the second derivative: $\frac{\partial^2 c}{\partial z^2} = \frac{c_{i+1} - 2c_i + c_{i-1}}{h^2}$

8.3.1 First partition

$$c|_{z=0} = c_{in}$$

$$\frac{c_0 + c_1}{2} = c_{in} \Leftrightarrow c_0 = 2c_{in} - c_1$$

$$\frac{dc_1}{dz} = \frac{c_1 - c_0}{h} = \frac{c_1 - (2c_{in} - c_1)}{h} = \frac{2c_1 - 2c_{in}}{h}$$

$$\frac{\partial^2 c_1}{\partial z^2} = \frac{c_2 - 2c_1 + c_0}{h^2} = \frac{c_2 - 2c_1 + (2c_{in} - c_1)}{h^2} = \frac{c_2 - 3c_1 + 2c_{in}}{h^2}$$

Thus…

$$\frac{\partial c_{s,1}}{\partial t} = D_{ax} \frac{\left(c_{s,2} - 3c_{s,1} + 2c_{s,in}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(2c_{s,1} - 2c_{s,in}\right)}{h}$$

$$\frac{\partial c_{i,1}}{\partial t} = D_{ax} \frac{\left(c_{i,2} - 3c_{i,1} + 2c_{i,in}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(2c_{i,1} - 2c_{i,in}\right)}{h} - \frac{1 - \varepsilon_c}{\varepsilon} r_{i,1}$$

8.3.2 2 - Nth-1 partition

$$\frac{\partial c_{s,j}}{\partial t} = D_{ax} \frac{\left(c_{s,j+1} - 2c_{s,j} + c_{s,j-1}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(c_{s,j} - c_{s,j-1}\right)}{h}$$

$$\frac{\partial c_{i,j}}{\partial t} = D_{ax} \frac{\left(c_{i,j+1} - 2c_{i,j} + c_{i,j-1}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(c_{i,j} - c_{i,j-1}\right)}{h} - \frac{1 - \varepsilon_c}{\varepsilon} r_{i,j}$$

8.3.3 Last partition

$$c|_{z>L} = c_N$$

$$\frac{\partial^2 c_N}{\partial z^2} = \frac{c_{N+1} - 2c_N + c_{N-1}}{h^2} = \frac{c_N - 2c_N + c_{N-1}}{h^2} = \frac{c_{N-1} - c_N}{h^2}$$

Thus…

$$\frac{\partial c_{s,N}}{\partial t} = D_{ax} \frac{\left(c_{s,N-1} - c_{s,N}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(c_{s,N} - c_{s,N-1}\right)}{h}$$

$$\frac{\partial c_{i,N}}{\partial t} = D_{ax} \frac{\left(c_{i,N-1} - c_{i,N}\right)}{h^2} - \frac{v}{\varepsilon} \frac{\left(c_{i,N} - c_{i,N-1}\right)}{h} - \frac{1 - \varepsilon_c}{\varepsilon} r_{i,N}$$

8.4 Appendix D – Data from case study 2B

PHYSICAL PARAMETERS

Packing parameters: column void, $\varepsilon_c = 0.42$, and packing porosity, $\varepsilon_p = 0.62$, and assume a particle Peclet to be constant, Pe = 0.5.

Adsorption parameters: Langmuir kinetic model with mobile phase modulator of salt conc.

Experimental conditions

pH 8

Buffer A:	0,05	M	NaCl
Buffer B:	0,35	M	NaCl
Feed (diluted)	0,10	M	NaCl
Feed conc (diluted)	5	g/L (total peptide)	
Bed height	10	cm	
flow	5	CV/h	

Resin	Q Sepharose FF		
Particel diameter	90	µm	

Feed composition	A	0,169	g/g
feed is diluted 1+1 w. buffer A	B/C	0,662	g/g
	D	0,169	g/g