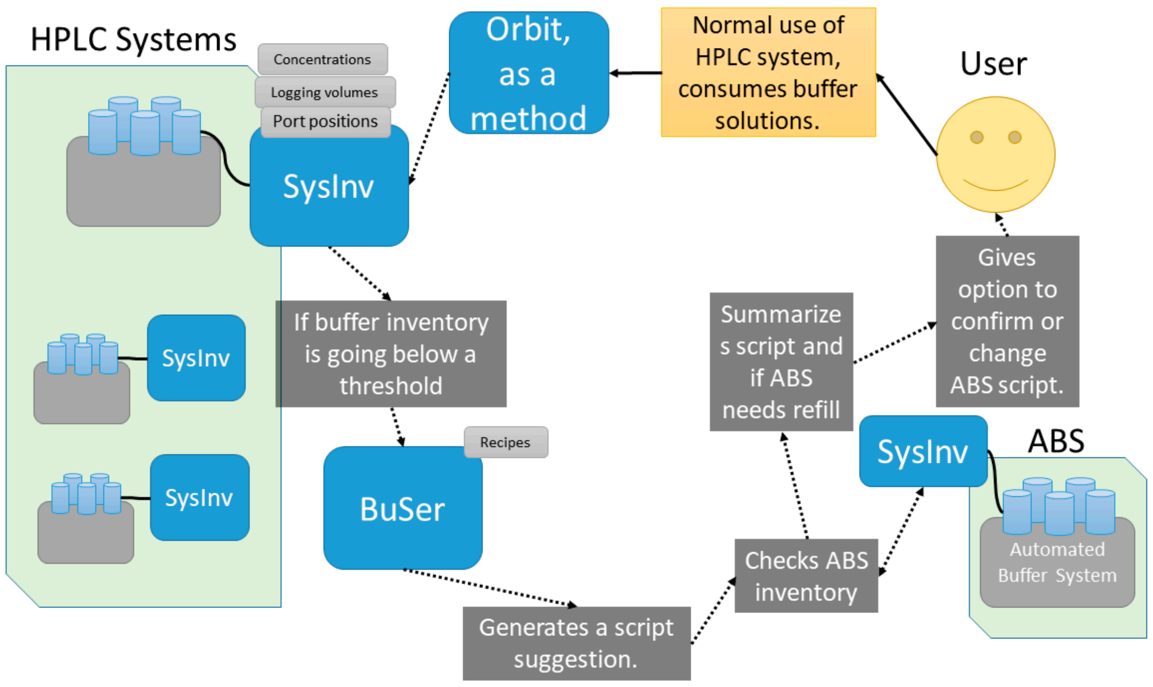
Autonomous Buffer Preparation



HPLC Systems
Concentrations
Logging volumes
Port positions
SysInv
SysInv
SysInv
Orbit, as a method
Normal use of HPLC system, consumes buffer solutions.
User
If buffer inventory is going below a threshold
Recipes
BuSer
Generates a script suggestion.
Checks ABS inventory
Summarizes script and if ABS needs refill
Gives option to confirm or change ABS script.
SysInv
ABS
Automated Buffer System

Lukas Önnestam

Department of Chemical Engineering

Master Thesis 2020

LUND UNIVERSITY

Autonomous Buffer Preparation

Master Thesis

Author: Lukas Önnestam

Tutor: Niklas Andersson
Examinator: Bernt Nilsson

Department of Chemical Engineering, Chromatography Group
Lund University
Sweden
September 1, 2020

Abstract

Every laboratory involved with liquid chromatography (LC) has to solve the issue of a supply of mobile phase. In the case of reversed-phase liquid chromatography the mobile phase is polar in nature and most commonly aqueous solutions. With a source of distilled water in-house the possibility of creating a mobile phase supply is within reach. A pair of retired LC systems is being repurposed for an automated mobile phase production, and these solutions should have buffering capacity. This thesis will be dedicated to deliver buffer solutions by mixing conjugating acid/base salt stock solutions with distilled water by automating said systems. The automation is made possible by the Python-based Orbit library developed at Department of Chemical Engineering, Lund University.

Both experimental setup and programming had to be employed for this endeavour. An signal called "accumulated volume" was experimentally investigated for reliability and accuracy, to be used as a run criteria in automation programming. It was found to be able to deliver volumes within 1.5% difference to target volume (7 ml) at high flowrates (100 ml/min).

Programming work spawned new programs with the tasks of keeping track of inventory of a system as well as receiving requests of buffer solution and construct scripts in order to deliver.

Conclusively, the current state of the material involved in this project is not a finished automated buffer system. However it is significantly progressed and promising for being an addition to the plethora of tools available for the LC laboratory.

Sammanfattning

Varje laboratorie som använder sig av vätskekromatografi måste lösa frågan kring försörjning av mobilfas. Om man huvudsakligen använder sig av omvänd-fas vätskekromatografi så är den mobila fasen polär och består för det mesta av vattenlösningar. Har man en källa till destillerat vatten så är det inte långt till att kunna producera sina egna lösningar.

Ett par av pensionerade vätskekromatografi system ska omanvändas till ett automatiserat system för produktion av mobilfaslösningar. Det är ett krav att dessa lösningar skall ha en buffrande förmåga och tillräcklig kapacitet. Denna uppsats kommer handla om att visa hur bufferlösningar kan göras genom att blanda konjugerande syra/bas/salt stamlösningar med destillerat vatten genom att automatisera ovan beskrivet system. Denna automatisering är möjlig genom det Python-baserade biblioteket Orbit, som är utvecklat hos avdelningen i kemiteknik vid Lunds Universitet.

Både experimentella uppställningar och programmering användes som tillvägagångssätt för att undersöka det här fallet. En signal vid namn *ackumulerad volym* var experimentellt testade för pålitlighet och träffsäkerhet, för att kunna användas som ett kriterium i automatiseringsprogrammen. Det visade sig kunna generera volymer inom 1.5% skillnad av målvolymen (7 ml) vid höga flöden (100 ml/min).

Programmeringsarbetet gav upphov till nya program som skall hålla reda på vilket material varje system har tillgängligt, men också ta emot beställningar av bufferlösningar och konstruera skript för att leverera.

Sammanfattningsvis, det nuvarande tillståndet av materialet som har tagits fram i detta projekt är inte ett färdigt automatiserat system. Däremot så är det på god väg och visar på lovande förmågor att kunna användas till bufferlösningsproduktion.

Acknowledgement

The end of a journey like for so many students before I have quite a few to show my utmost gratitude.

First of, Niklas Andersson who has been my tutor this spring simply put; you are great! Your enthusiasm for the work that you do is admirable, and the effort in supporting me is immensely appreciated.

My examinator and initial person of contact for this thesis project Bernt Nilsson, I thank you wholeheartedly for the openness and welcoming environment you create.

Nervous as anyone new at a unfamiliar place is, I came to the Department of Chemical Engineering and felt accepted incredibly fast. Everyone of you that I shared time with in the lab, fika room and corridors, thank you.

The path to get this project initiated and underway, you know who you are; Helena A, Görel E, Anna K, for making me see the world more straight than I could on my own.

Also a huge thank you to the leaders of Theatre Without Borders the fall of 2019; Magnus, Daniel and Tony - you guys taught me things I did not think I needed to know.

Oskar H, I got to know you within the walls of Katte high school and you have been with me ever since. Without your support through encouraging words or a Happy Meal with toy included, I do not know what degree of a wreck I would be. For all the wondrous memories till here and forward, sincerely thank you.

Finally, but not the very least, my family; father, mother, Elin, Amanda and the dog Olle. I would have not made it this far without everyone of you in my heart.

Contents

1 Introduction

Delving into science and its intricate categories of fields one might find oneself in a laboratory. Such a place is equipped with instruments that could be used to reveal more of the physical world we live in. One such instrument is the liquid chromatography (LC), an established method in the analytical chemistry field.

Operated with the proper equipment and parameters one could distinguish between proteins of different shape. But in order to have such an instrument running a consumable solution with specific quality and properties is required. Namely, a single or multiple buffer solutions to bring the protein through the tubing, pumps, sensors and column.

Nevertheless, buffer solutions are a crucial component of a working LC system. Of course, producing these solutions oneself is favorable for a control of error but also in the economic department.

This project will explore the possibility to employ a pair of ÄKTA-systems (high-pressure LC, or HPLC) for an automated product of buffer solutions. This will relieve workload, increasing time designated for research using the HPLC systems.

2 Background

2.1 Chromatography

Despite that this project involves pair of chromatography units the work involves no actual chromatography processes. However, the product for this project is a system that produce buffer solution, which fills a multitude of crucial functions in chromatography. So it makes sense to briefly cover chromatography here.

Chromatography is analytical separation method based on differences in chemical compounds attraction to a secondary phase. This secondary phase is a solid porous or packed material in which the primary mobile phase is flowing through, this is called liquid chromatography. During analysis the mobile phase consists of two parts, the sample and a buffer solution. Often the sample is containing multiple types of chemical compounds which again, interact with the solid phase, and thus require different time to travel through the solid phase. After the separation via the solid phase the mobile phase is led into sensor equipment, like UV, conductivity and pH. These sensors are what the user sees as output and presented in a connected computer software. Since the mobile phase consists of buffer in majority it allows for stable chemical conditions, most notable pH, throughout the process run. [2]

2.2 Buffer Preparation

So what is a buffer solution and how do you make one? A buffer solution main property is to resist changes in pH. This chemical property is mathematically defined as the inverse logarithm of the hydrogen ion concentration and denoted $[H^+]$ in units of mole per cubic decimeter, see Equation 1. This denotation is actually a shorthand for $[H_3O^+]$ which forms when a Brønsted-Lowry acid donates a proton, that H^+ essentially is, and a water molecule accepts it.

$$pH = -log[H^+] = log\frac{1}{[H^+]} \tag{1}$$

The reaction formula for this proton donation for a weak acid (HA) is described in Equation 2. What is important about the acid being "weak" is that this reaction is instead an equilibrium reaction, effectively only a portion of all the weak acid molecules react with water. This reaction can also be reversed, with increased levels of H_3O^+ it reacts with the conjugate base (A^-) into water and weak acid again.

$$HA + H_2O \rightleftharpoons H_3O^+ + A^- \tag{2}$$

This protonation/deprotonation equilibrium reaction is described in math by Equation 3.

$$K_a = \frac{[H_3O^+][A^-]}{[HA][H_2O]} \tag{3}$$

The value of K_a is chemical specie specific and is also useful to convert to its logarithmic counterpart.

$$pK_a = -logK_a = log\frac{1}{K_a} \tag{4}$$

Equation 4 combined with Equation 3 and Equation 1 displays a relation between this chemical specific property pK_a and pH.

$$pK_a = pH + log\frac{[A^-]}{[HA]} = pH + log\frac{[conjugate\,base]}{[weak\,acid]} \tag{5}$$

If you have a chemical specie that has a pK_a value within the conventional pH range of 0 to 14 and mixing equal amount of the specie and its conjugating base you end up with Equation 6.

$$pH = pK_a \tag{6}$$

This is the case for weak acids and weak bases as well which coincidentally have their pK_a take on these values. If you would attempt to increase or decrease the pH it would require more resources to do so compared to a water system without this weak acid/conjugating base present. [1]

For a chromatography system this behaviour is desired for a mobile phase [3]. Buffer solutions which they are called is mixed by combining a conjugating acid/base pair in an aqueous solution. Example of such a pair would be Acetic acid $CH_3COOH(l)$, Sodium Acetate $CH_3COONa(s)$ each diluted in water and then mixed together in an equimolar fashion you get buffer solution.

2.2.1 Market Products

Since chromatography is a very robust and widely used technique in a life science setting [3] a couple of products exists to provide a system producing buffer solutions of high quality.

A company by the name Sartorius offer solutions aimed at life sciences and one such product promises a wide range of uses one being buffer preparation, given the consumer chose this configuration. Named Flexact® BP is the Buffer Preparation variant and has a capacity of producing from 50 L to 3000 L in an automated process. [11]

Another example of a market available buffer preparation solution is from Cytiva called BioProcess IC System, this company is the same which currently owns the ÄKTA-line discussed later. BioProcess IC System is a separate system to buffer production, it also operates on an automated basis with single component stock solutions as a feed material. This systems design of being a separate ultimately means that it could provide buffer to other system types other than chromatography. [10]

A second option to their dedicated system is an actual integrated buffer preparation system in a chromatography unit named ÄKTA Avant. This variant is adjusted to being part of the chromatography unit and therefore more space effective than previous product. [9]

2.3 Overview of Orbit

Orbit is an in-house code library developed in Python, designed to control primarily chromatography processes via communication protocols by the Chromatography Group at the Department of Chemical Engineering, Kemicentrum, Lund University [4]. It allows for very complex run operations of chromatography systems, increasing extent of separation and thereby analysis.

In essence, Orbit consist of a tenfold text files that work together to retrieve and send information to Unicorn which in turn allows for running the ÄKTA system through a command prompt. The user has to only create three distinct types of files for their specific need, lets say in case of a complex chromatography run. See 1 for a simplified overview of Orbits structure.

Firstly, write a system-file that depicts the ÄKTA system and all its units and tubing, i.e. pumps, valves and sensors. One notable Orbit file that is really useful when writing the system-file is the Unit_library.py-file. In it a lot of the specific units in a couple of ÄKTA systems has been described, and for the user it is only a matter of importing the file and referring to the correct units. An important property of the valve units in the system-file is to declare where each valve port leads to, this is referred to as ”maps”.

Secondly, create a process-file for the specific experiment that is to be run. Here common operations could be defined that is going to be repeated throughout the run.

Lastly, create a script-file which calls for the system and process-files specific operations, in which order they happen, under which operating parameters, which signals to be logged and then order the run command prompt. This sends all the users self declared information to Orbit which in turn runs Unicorn and the ÄKTA system. The script file is the most frequently changing file of all the user-files related to Orbit, which means a new script is produced between runs with few parts remaining the same.

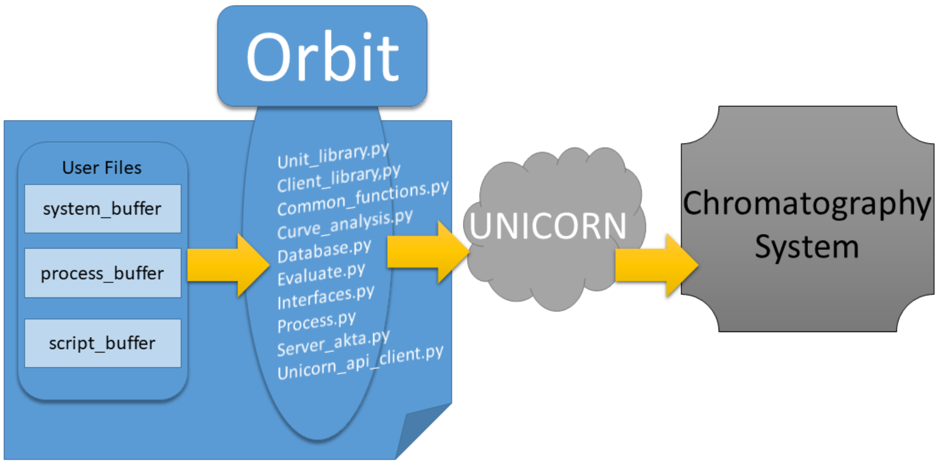


Figure 1: General overview of information structure revolving Orbit, Unicorn and ÄKTA.

3 Method and Materials

3.1 System Flowcharts

Two flowcharts will be presented here in this section. The first describes the system at the beginning of the project and the second is for the final configuration of the same system. Later the laborative experimental part will have their own highlighted variant of the system to test that specific part that the experiment is designed for.

3.1.1 System at the start of project

At the very beginning of the project the dual ÄKTA Explorer setup was as describe in figure 2. Remember the different valves are named based on their original purpose as a chromatography process system, so the name ”Sample Valve” do not make any sense compared to what its current use actually is.

At the very top of the flowchart eight positions for possible stock solutions is represented by *Stock 1*, *Stock 2* incremental up to *Stock 7* and finally (distilled) *Water*. These are connected to a valve with eight ports and one outlet leading to a pump denoted *Pump A1*. The outlet of the pump is connected to the *Sample Valve*, this is the main path this valve will deliver fluid to the *Mixing Flask* and there is secondary paths to *Waste* and *Next System*. The route described between the *Stock* units to the *Mixing Flask* is how the actual mixing of buffer process begins.

When the desired stock solutions has been added an operation of pH adjustment is performed. The pH is monitored by having the primary inlet of *Pump B1* in the *Mixing Flask* delivering the buffer solution in making to the *pH* sensor (ÄKTA Explorer in-House sensor). When exiting the sensor and into the *Outlet Valve* the primary path is back into the *Mixing Flask*, with secondary paths to *Waste* and *Next System*. This route creates a loop in order to continuously monitor the pH during a run.

The actual addition of base or acid to the buffer solution in making is what System 2 is dedicated to. *Pump A2* with a 5M hydrochloric acid (HCl) at the inlet and outlet leading into the *Mixing Flask*. *Pump B2* connected to a 4M sodium hydroxide (NaOH) and outlet in the *Mixing Flask*.

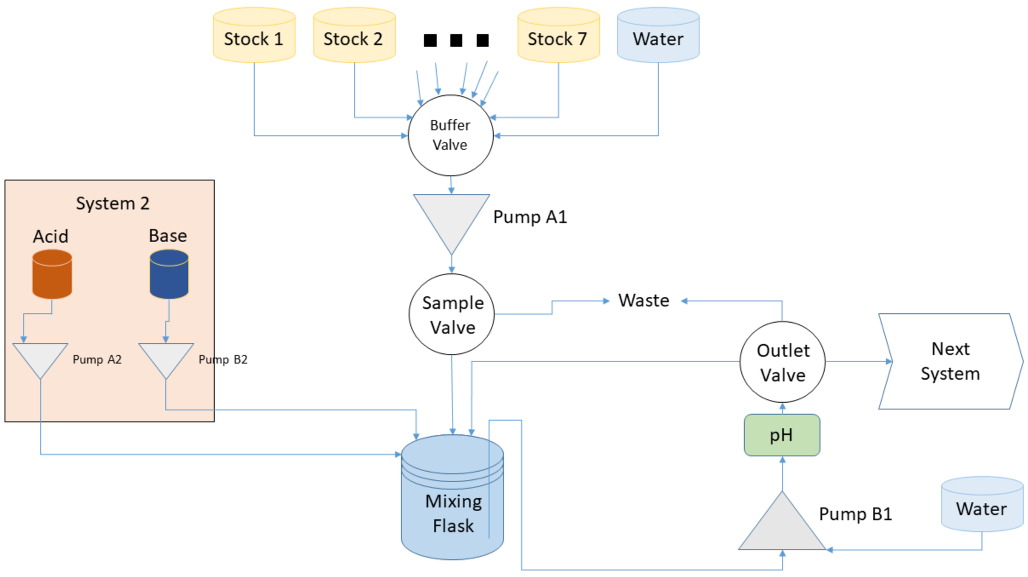


Figure 2: Initial configuration of the dual ÄKTA Explorer system at the beginning of project.

3.1.2 Final configuration

As with all things natural, changes will be made. Although for the physical setup a lot of Fridas original work has been expanded upon rather than remade. The figure below is an illustration of the final flowchart representing the system, see figure 3 . No indication of conductivity and UV sensor being included previously, so these were added as an extension of the pH loop and now an option in orbit to include or exclude for a particular run. A water source is available for rinsing the Melander system (S2). *Pump A1* had no connection for its second inlet which now is connected to the *Mixing Flask*, now it is possible to empty the *Mixing Flask* with both pumps at same time if one wishes.

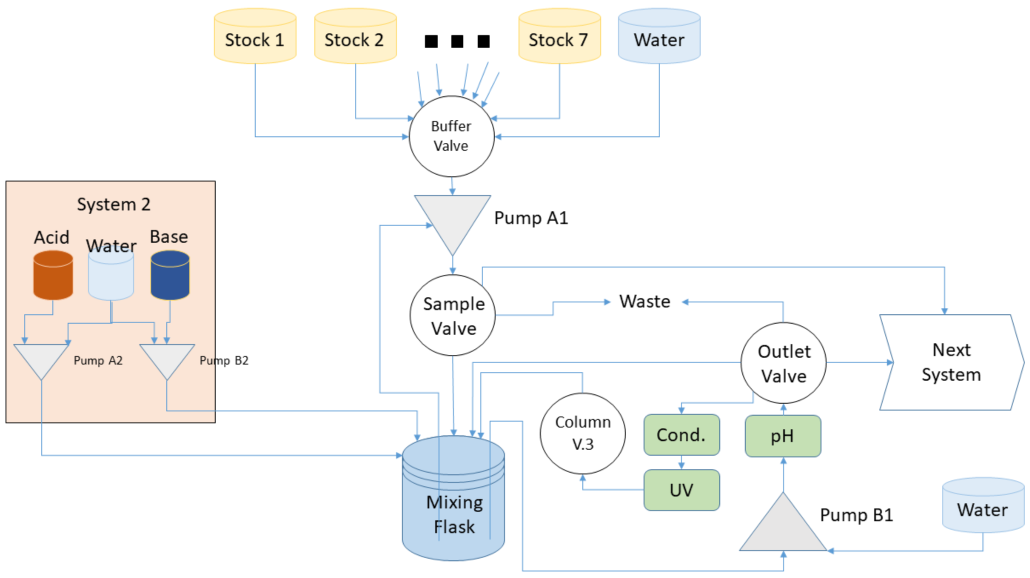


Figure 3: Updated flowchart for the buffer system. Notable additions is a water source in System 2, possibility to include conductivity and UV sensors in the pH loop. Minor inclusion is the added tubing from mixer flask to pump A1.

Physical setup of Horvath and Melander ÄKTA Explorer chromatography units see Figure 4 below.

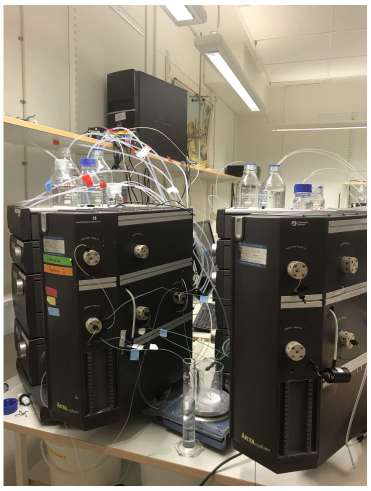


Figure 4: Physical ÄKTA dual Explorer system. Horvath (S1, system 1) on the left and to the right Melander (S2, system 2). Mixing flask inbetween and waste bucket below table.

3.2 Experimental work.

Executing code on the computer, investigating whether the machinery performs the tasks that the code represents. How the investigative part is done is usually through data sampling, either from Unicorn's own generated values, or taking physical measurements like total volume pumped through the system. Some simple analytical sampling was done with pH-instrument, but more so for verifying the chromatography machine's in house pH-instrument.

3.2.1 Preparation of stock solution

Of course, preparation of stock solutions that is used to prepare buffers was a part of the laborative work. Most stock solutions consists of a single ingredient of a relative high concentration to that of the finished buffer solutions, example is 500 mM NaAc compared to the buffer with 31.9 mM NaAc. Sodium acetate trihydrate (NaAc * 3H2O), sodium chloride (NaCl) and sodium hydroxide (NaOH) are prepared into stock solutions by weight measurement of each salt and diluted in a volumetric flask. NaCl and NaOH in salt form were anhydrous. Acetic acid on the other hand was prepared from a pure concentration liquid (100% HAc) and simply diluted with distilled water.

All calculations for preparation of stock solutions the excess volume effect of partial molar properties was ignored and assumed ideal interactions instead. Nevertheless, all solutions are diluted with distilled water, by the in-house production, and no non-aqueous based solutions were included for production of buffer.

3.2.2 Case 1: Setup of "Accumulated Volume" test

Case 1 is a test designed to show difference or limitations of valve switching as a method to produce desired volumes from the pumps each system is equipped with. A secondary purpose was to explore if a Unicorn parameter "Accumulated Volume" could be used as criteria for the valve switching. As opposed of using the Python-side clock to determine when to stop flow. Each pump has a viable range of flowrate between 0.01 ml/min to 100 ml/min. Expecting to encountering difficulties at high flowrates and small volumes, 50 and 100 ml/min and target volume 7 ml was decided. The 7 ml target volume is from experience producing buffer solutions of 150 ml required this volume from the stock solutions. A volumetric flask was installed pump A of system S1. The script ordered 12 repeats of above run parameters for a target total of 84 ml as the volumetric flask had a range of 10 to 100 ml markings. All stock solutions are expected to behave similar to water as working fluids.

In the figure shown below, it demonstrates the simple rearrangement of the outlet of pump A in system 1 from the mixing flask to a 10 to 100 ml volumetric flask.

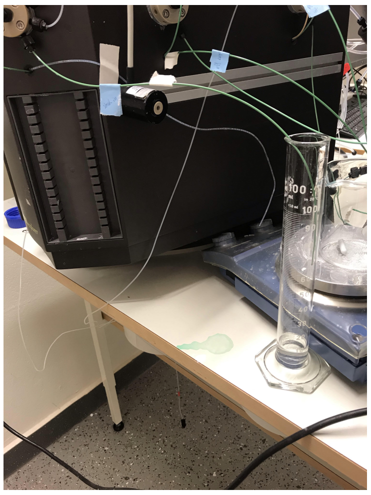


Figure 5: Physical setup, outlet from pump A to a volumetric flask.

Here is a crop out of the flowchart figure highlighting with a green glow which path is run through this experiment case, see figure 6

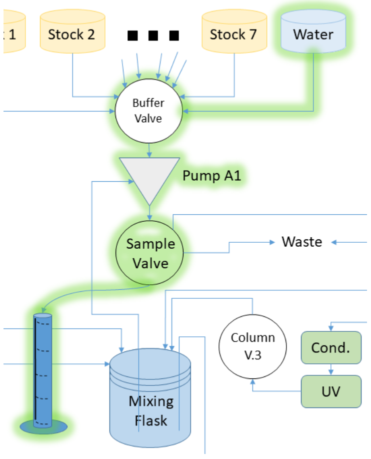


Figure 6: Part of flowchart included in this experiment case, outlet from pump A to a volumetric flask.

3.2.3 Case 2: Setup of "Man vs Machine" test

This test was an early test to let the system produce a buffer of three components, $NaAc$ 31.9 mM, $NaCl$ 50mM and HAc 18.1 mM with pH 5 at a final volume of 150 ml. Log pH and conductivity measurements and try to mix a buffer solution by hand as close to the same recipe as possible. Run the hand-made solution through the same instruments the system use and log pH and conductivity. Additionally, same stock solutions to produce the machine buffer is used for the hand-made one.

Below, Table 1 depicts the target volumes for each specie with the used stock solutions concentrations for a 150 ml final volume, where the remainder is distilled water. Since these volumes are limited to the value accuracy of the pipettes available, tenth of a millilitre being the smallest increment and hence no smaller order decimal values.

Table 1: Target volumes from stock solutions for machine- and human-made buffer solutions for a final volume 150 ml.

Specie	*Stock Conc. [mM]*	*Volume [ml]*
$NaAc$	499	9.4
$NaCl$	2500	2.9
HAc	477	2.6
H_2O	—	135.1

Procedure of hand-made buffer solution; the three stock solution species were carefully pipetted to a volumetric flask of 100 ml, and subsequently filled with stock distilled water to the 100 ml mark. Solution transferred to final flask container, another batch of 50 ml stock distilled water measured up in the same volumetric flask and lastly these 50 ml too were transferred to the final flask for a total volume of 150 ml.

Initially, the final flask was connected to the pH loop in the ÄKTA dual Explorer system, run at different flowrates for a couple minutes each and making sure no pH fluctuations was present. Thereafter, a constant flowrate was issued and manual pipettation of 5M hydrochloric acid or 4M sodium hydroxide to the hand-made solution during the run. Until the pH was within the same limits (pH = 5.00 +/- 0.05) as for the machine-made solution.

3.3 Coding in Python.

Divided into two parts; The coding language itself and the other is; Orbit which has its rules and structures but it is written in Python. The coding and running said code if it executes properly is a very time consuming part of the whole project. A lot of trial, error and learning is behind this section, as the projectee had no prior experience of Python and only 7.5 credits of Java. For Orbit, there is reports available explaining its structure and logic as Orbit itself has been a continuous project the last couple of years by the Chromatography group of the Chemical Engineeering Department [4, 5]. It is by all means a good introduction, although good understanding comes from using already functioning material like the buffer-file bundle by Frida Heskebeck. A brief overview is included later in this section.

As for the gritty coding in the Python language a lot of online resources was applied as lexicons, to name a few w3schools, docs.python.org, programiz and stackoverflow. They are all great to look up built-in functions in python and common coding issues. Python is a text-based language which basically means you could open most files in Notepad on Windows, higher flexibility and quality of life is available in Spyder and Anaconda. Spyder is the substitute for Notepad and provides numerous tools to make coding easier and faster to create and troubleshoot. Anaconda is replacing the function of Run command prompt. Both available at the Chemical Engineering departments computers.

3.4 ÄKTA-system

The ÄKTA system originates from a company called Pharmacia back in 1994 and was developed for primarily protein purification. The product series has been inherited and is presently owned by Cytiva. Production of protein by ÄKTA systems is in the scale of micrograms to tens of grams, i.e. laboratory scale. The system version in question is ÄKTA Explorer 100 (code nbr: 24450377) which is discontinued and succeeded to ÄKTA avant 150 or ÄKTA pure versions.

The system allows for automatic control of flow and monitoring of sensor signals like UV, conductivity and pH as a means of following the purification process. Features that allow for different elution methods, such as automatic, simple and step-gradient by utilizing a pair of pumps to change the composition of buffer mixture during the process. The system is outfitted with multiple six-port valves that allows for selection of buffer solution, flow direction through column, allowing sample into system and open for washing of equipment, to name a few [7]. Placement of these valves in relation to column, sensors and pumps opens up for high configurability of the system.

An entire protein purification can be preprogrammed in its accompanied software Unicorn, although depending on choice of technique for injection of protein sample some manual interaction is involved. Unicorn software reflects the system units and during run also the configuration of valves, which ports that are open to where et.c.. [8]

3.5 Buffer program and experimental setup structure

The part of Frida Heskebecks master thesis work [6] describes a proof of concept for an automation of a pair of ÄKTA Explorer chromatography units. Code-side consists of a file bundle; system_buffer.py, process_buffer.py and script_buffer.py, which allows the user to produce a 50 mM $NaAc$ buffer solution. Physical-side is of course the system itself, the pair of chromatography machines which is rewired to fit the code and work towards a separate magnetic-mixing flask. When buffer is finished it is produced towards either the waste or "nextSystem" representing a to-be delivery unit or storage unit. However, when resuming this work nextSystem consists of a secondary flask. The original flowchart is included in the flowchart section, see figure 2.

3.5.1 Orbit code initial overview of the user files

Mentioned earlier in subsection 2.3 system_buffer, process_buffer and script_buffer is the user files for this particular setup to produce buffer solutions. In this section the core functionalities and apparent missing functionalities will be covered.

First of, the user file system_buffer.py contain written code declaring valves, pH sensor and pumps and relevant mapping reflected in the flowchart Figure 2. A breakdown of the initial version of system_buffer.py is shown in Figure 7. Mapping here refers to what each unit different connections leads to, example: Sample Valve port 5 is paired with a text string like "flask". On initial inspection functionality like tubing declaration, conductivity & UV sensor and stock solution flasks were missing.

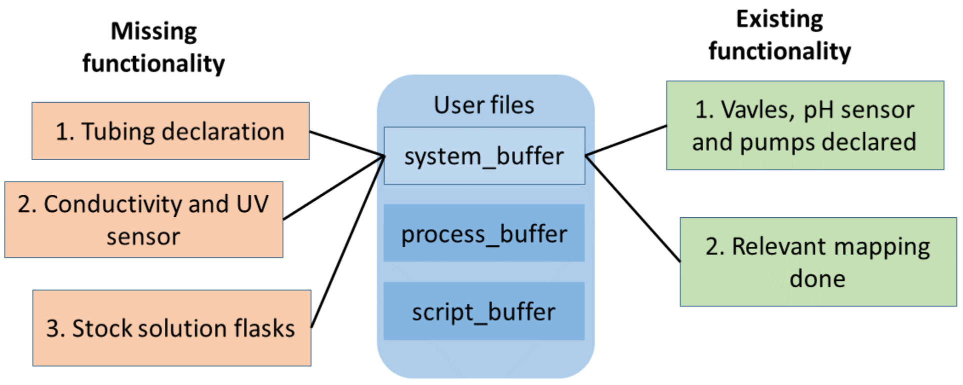


Figure 7: User file: system_buffer.py. Existing and missing functionality overview.

The process_buffer.py file was missing a dedicated wash program of the entire systems equipment as well as a function to transfer finished product from mixing flask to a destination. But existing functionality being pH adjustment event, loading of stock solution to mixing flask and minor Python functions like "set pump flowrate". For the visual breakdown of the functionality see Figure 8.

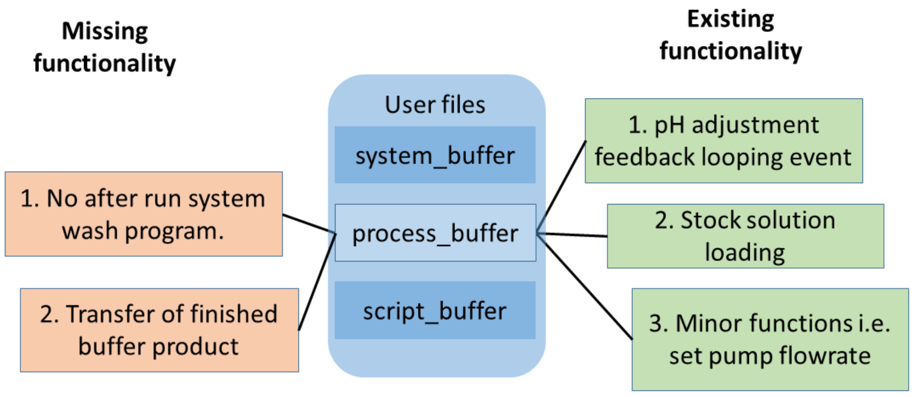

Missing functionality
1. No after run system wash program.
2. Transfer of finished buffer product
User files
system_buffer
process_buffer
script_buffer
Existing functionality
1. pH adjustment feedback looping event
2. Stock solution loading
3. Minor functions i.e. set pump flowrate

Figure 8: User file: process_buffer.py. Existing and missing functionality overview.

Lastly, the script_buffer.py file is able to be run in both test mode (where pH changes was simulated) and real mode producing a 50 mM $NaAc$ buffer solution. A lot of the parameters were explicitly defined with values (i.e.: flow = 50. % float value) which is of course needed at some point. However, some variables could be redefined to changeable attributes, such as the concentration of a stock solution being imported from a database. This way attempting to increase the portion of recyclable parts of the script. Remember, as stated in 2.3 the script is the code that most frequently has to be changed between runs. So construction of a "script producer" function with the help of a template could be of use towards automation. Finally a visual breakdown could be viewed in Figure 9

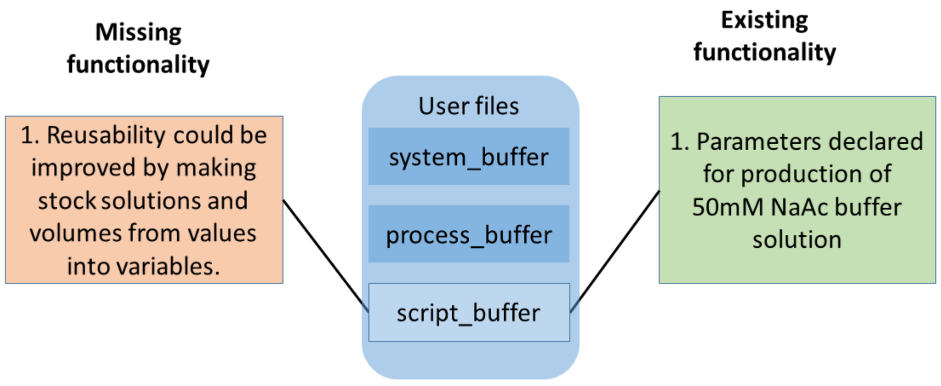

Missing functionality
1. Reusability could be improved by making stock solutions and volumes from values into variables.
User files
system_buffer
process_buffer
script_buffer
Existing functionality
1. Parameters declared for production of 50mM NaAc buffer solution

Figure 9: User file: script_buffer.py. Existing and missing functionality overview.

4 Result and Discussion

4.1 Experiments

A couple of defined experiments is presented below to complement heavy weight on code work in this project.

4.1.1 Case 1. Accumulated Volume

The resulting data is presented here in figure 10. A simple linear regression is included for both series of 50 and 100 ml/min accompanied by their R^2 value and approximated equation. Of course, ideally the expected equation should be y = 7.0x + 10.0, where y is accumulated volume and x is phase number. This data did not warrant worry as in the most deviating serie were 50 ml/min with a slope equal to 6.9011 which is a 1.43% deviation. No further intricate statistical analysis was performed with this in mind.

An apparent limitation of the experiment is the smallest unit displayed on the volumetric flask was 1 ml, example is the reading of a value is either 6 or 7 not 6.5 ml.

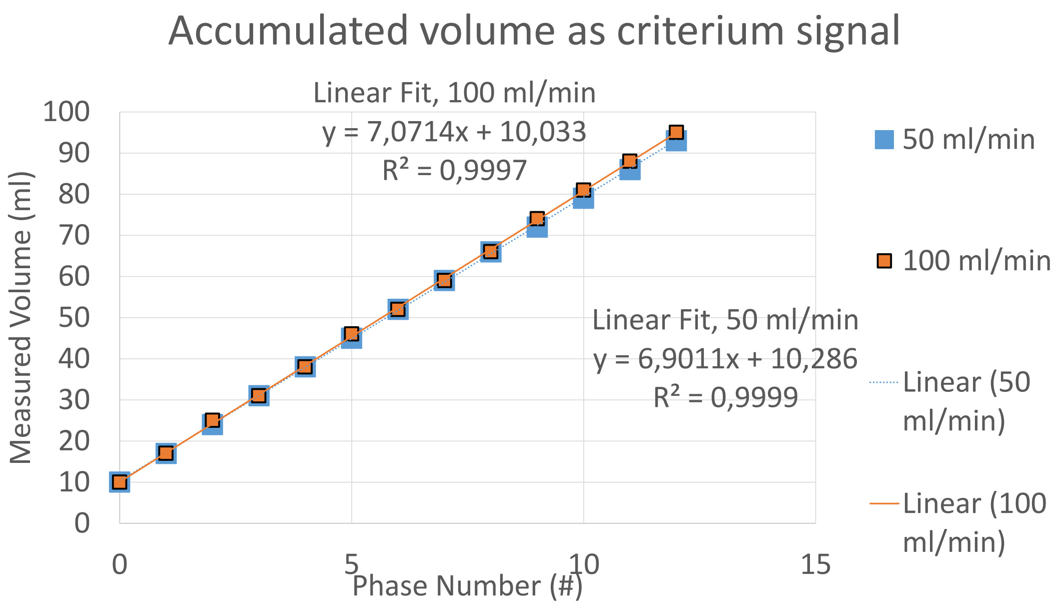


Figure 10: Accumulated volume versus phase number.

Remember, this experiment setup was setup with small final volumes, range of 150 to 200 ml, in mind and 100 ml/min is the highest flow rate according to design for ÄKTA Explorer pumps. Expected reasonable final volume would instead be upwards of 800 to 1000 ml and the added concentrated ingredient solution would increase in proportion.

4.1.2 Case 2. Man- vs Machine-made buffer solution

Starting by presenting the sampled signal for the machine-made buffer solution. Included signals are; flow, pH, accVol (accumulated volume), cond (conductivity), S2flow (flowrate of Melander). A couple of highlighted data points of interest is included in the figures, such as 'final pH', for easier reading of the at times cluttered imagery. Additionally, all the previously mentioned signals are represented in normalised values. An example of this is, maximum pH in figure 11 is 5.26 which is displayed as the value of 1 and the final pH value is adjusted to 5.05 which corresponds to 0.96 in the figure. This allows for all the signals to be viewed together throughout the run. Please refer to the legend on the right hand side in each figure for actual max values for each signal.

Figure 11 shows the system running flows of 50 ml/min, starting and stopping, this is the sequence for producing stock solution volumes to the mixing flask hence all the vertical beige lines between minute zero and five. Once this is done the pH adjustment process is initiated and the flow is put to a constant 50 ml/min and the conductivity with pH changes when the buffer solution runs through the sensor loops. What happens onward is Melander (S2) producing volumes of 0.05 ml with a flow rate (S2flow) of 1 ml/min (its max value) to the mixing flask. The behavior for the sixty minutes of remaining run time is starting, stopping, measuring pH twice and repeat. Eventually at roughly minute 55 the pH start to change and finally reaching the desired pH target of 5.05.

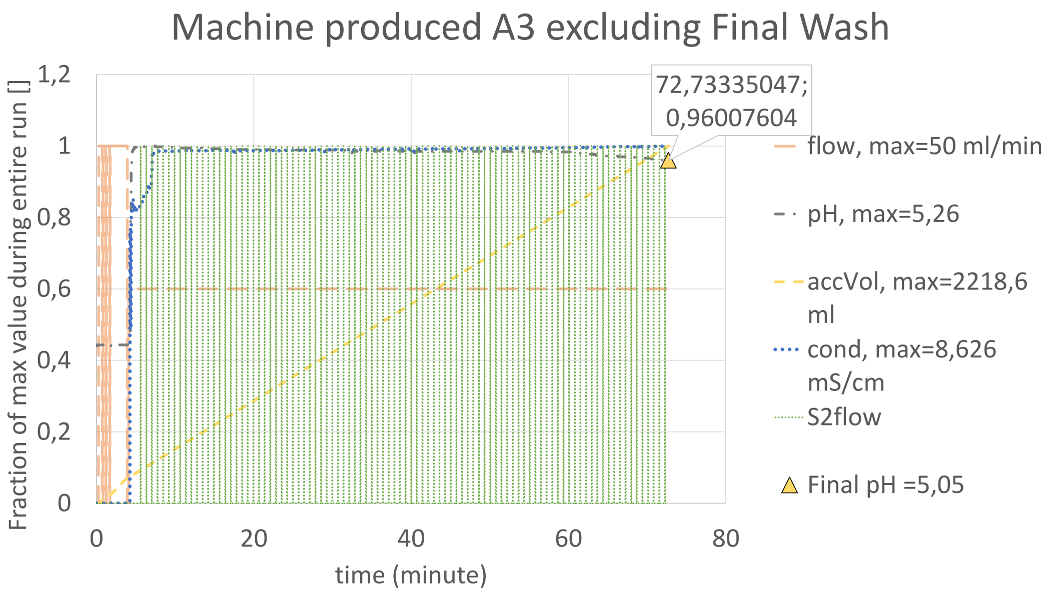


Figure 11: Diagram for buffer system run producing a solution of 31.9mM NaAc, 18.1 mM HAc and 50 mM NaCl. The data is displayed in fractions of their own maximum value during the entire run, in order to be able to plot all the signals together. Note: maximum value is not always the final value, which is important for the marked data point at the end of pH. 96% percent of 5.31 is 5.05 (within threshold of acceptable final pH).

Obviously, the extensive run time of this machine-made buffer solution is long and the issue was a simple one. Previously a washing segment had been added to the acid and base pump and tubing which leaves several millilitres of distilled water, approximation of this volume could be between 3 to 5 ml. This volume had to be pushed out before actual acid was added to the solution and thus the long time period before pH would change. Since each repetition of adding acid and measuring pH would take a little more than a minute each time, the volume added each time is as little as 0.05 ml and therefore taking this long. An obvious flaw in the programming not taking this into account.

Moving on to the human-made buffer solution which got hooked up to the sensor loop. In figure 12 the different flowrates is shown and a stable pH and conductivity reading is displayed.

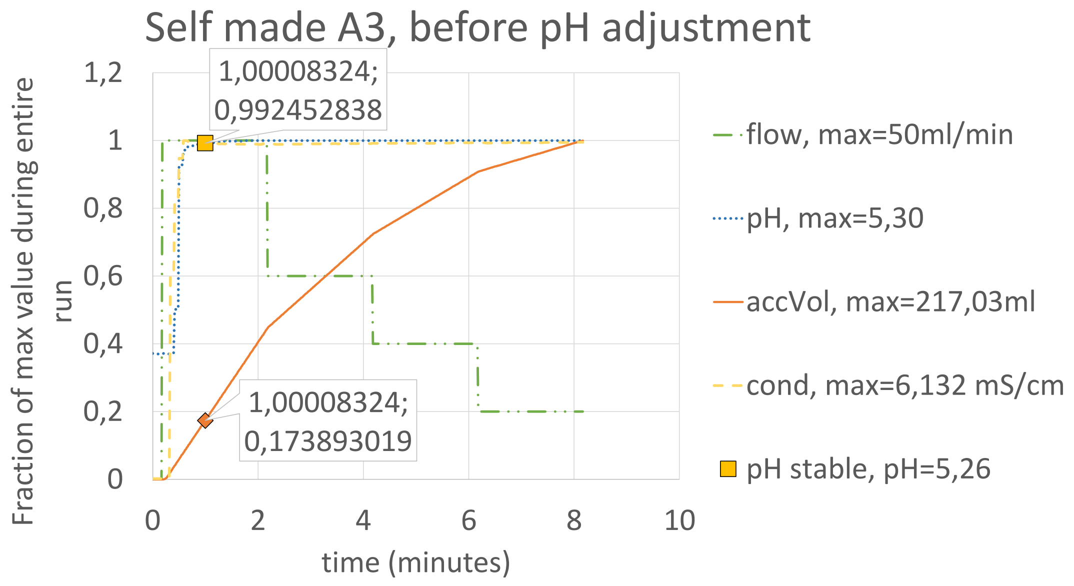


Figure 12: Displaying the run of hand-made buffer solution through the sensor loop, before said solution had its pH adjusted. A stable pH and conductivity reading is occurring roughly one minute in and after a total of 38.74 ml buffer volume pumped through the sensor loop.

Finally, the hand-made solution had a separate run for its pH adjustment at a constant flowrate of 30 ml/min. Since this adjustment of pH was manual, in other words using a 3 ml pipette to add droplets of 5M HCl to the buffer solution meanwhile waiting and monitoring for a change in pH from the sensors. Procedure went as follows; start pumps, verified stable pH reading, added 2 droplets of acid, new reading of pH 5.21, adding 3 droplets of acid, new reading of pH 5.00 and followed by a couple of minutes to confirm stable reading. See figure 13 below for the graphic representation of this procedure.

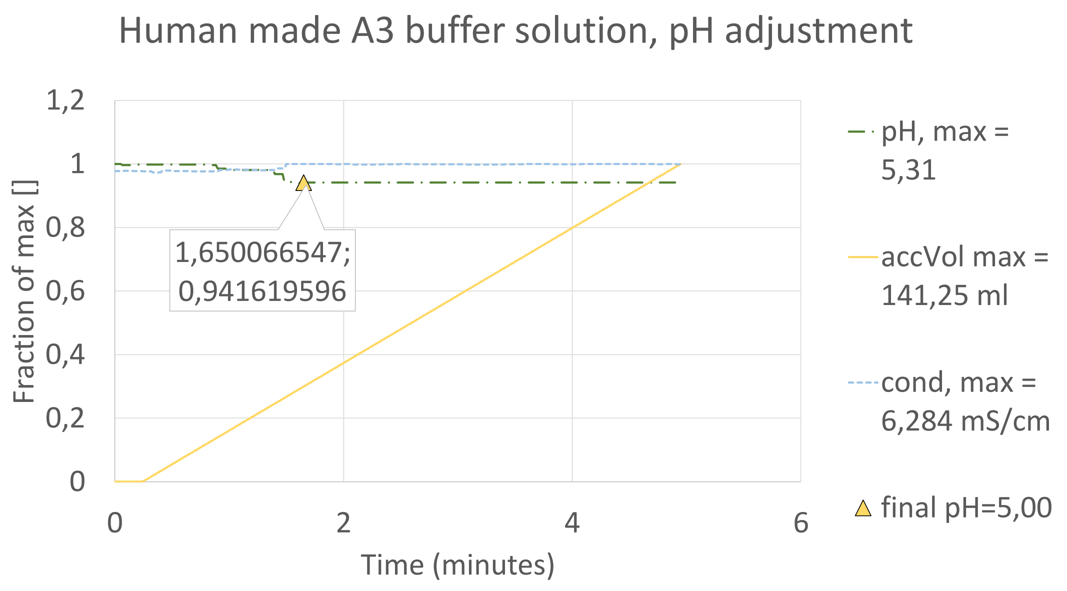


Figure 13: Run of the pH adjustment for hand-made buffer solution at a constant flowrate of 30 ml/min. Additions of two droplets at roughly minute 0.9 and three droplets at minute 1.2. As per same concept as earlier figures, the included signals are based on fractions of maximums for each signal and max values are stated in the legend on the right hand side.

In retrospect, an error in the method of this hand made buffer solution. The pH adjustment displayed was performed on an already 150 ml so these additional volumes of acid is technically exceed the intended final volume. In this case the amount of acid was minimal and thus not exacerbating the potential error.

To conclude, both the human- and machine-made buffer solutions has their issues and would warrant a reiteration of each experiment. The distilled water in acid/base tubing for the machine-made and the human-made technically exceeding the desired volume of 150 ml with the acid droplets during pH adjustment. However, one could argue that both of these issues are negligible as the distilled water is a stock ingredient as well and five droplets compared to 150 ml is relatively small.

This case indicates that the system is well underway to be able to produce sufficient quality buffers for in-house use.

4.2 Code

A few additions made to work with the Orbit library. Most of the code produced comes with creator-made comments to clarify and ease the understanding and use.

4.2.1 Overview, information path

Figure 14 presents a summarization of how the two new class files SysInv.py and BuSer.py works in relation to each other but also existing structures. SysInv is short for SystemInventory.py and BuSer is for BufferService.py.

The user is using Orbit as normal for the use of another HPLC system, an underlying code structure records the use of buffer solution and feeds this information to SysInv.

A threshold is checked if some buffer solution is running low in the users HPLC system. This prompts a call to BuSer which generates a script suggestion and checks its own inventory if it can produce the specific solution. A summarization is the presented to the user which then could change it or confirm and run it.

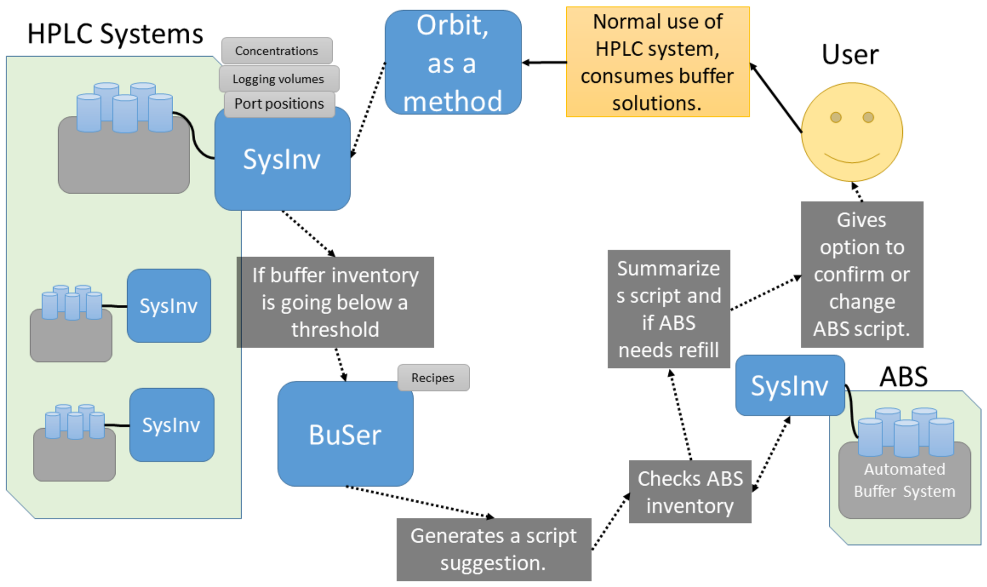

HPLC Systems
Concentrations
Logging volumes
Port positions
SysInv
SysInv
SysInv
Orbit, as a method
Normal use of HPLC system, consumes buffer solutions.
User
If buffer inventory is going below a threshold
Recipes
BuSer
Generates a script suggestion.
Checks ABS inventory
Summarizes script and if ABS needs refill
Gives option to confirm or change ABS script.
SysInv
ABS
Automated Buffer System

Figure 14: Path of code information for the new files in relation to existing code. A good place to start reading this figure is to start at the User, or smiley, and then proceed counter-clockwise.

The following sections will explain in further detail what the different files do and which options is available.

4.2.2 SystemInventory.py

Short connotation is SI or SysInv, is a class designed with simple functions that writes down data on what the system has in store. Python has its own serializing method; Pickle, which in SIs case creates and calls a file SIDLog.pickle where information is stored.

It made sense to create an attribute of the dictionary type where its keys is each specie the user has included. Retrieving the value from a key would grant the user another dictionary, effectively the pickle file is an encrypted nested dictionary. This specie specific dictionary includes properties like: position in system, sum of volume entries, list of volume entries, list of dates for the volume entries and concentration.

This gives the system its own memory and opens up possibility for an automated code-side decision making, like figuring out if the system have enough of said specie to produce a requested buffer. However, that kind of functionality is intended to be inside of a class discussed later, namely BufferService.

Table 2 describes the intended structure of the dictionary described above.

Table 2: Example of typical information available from a single entry of the pickle file. Top row is not part of the file but help defining what each column stands for. SID abbreviation for SystemInventory Dictionary.

Dictionary['specieX']	*keys*	*values*
SID['hac'] = {	'position'	'a4'
—	'concentration'	499
—	'sumvol'	323
—	'vollog'	list(volume entries)
—	'dates'	list(corresponding dates) }

SI has a constructor that allows the user to pass a path to the folder which contains the pickle file, otherwise if no such path is included it will assume the path folder is the current working directory. Constructor will attempt to read the pickle file and loads up a temporary copy of the dictionary and finally an argument can be passed to state what mode to run, 'test' or 'real'. Mode alternatives reflects the modes in orbit 'test', 'real' and 'ask', where the last mentioned will be translated into 'real' for SI.

First function is 'getValue' which allows you to retrieve specific values from a designated specie. An alternative is to not designate a target specie, then the function calls turns into an iterable usable in loops checking for some criteria. This alternative is known in Python coding as a generator function. This generator function is applied to help the system_buffer file to know which Flask units are filled with what and how much.

Following functions are getSpecieList and getSIDict, first one gives the user a list of all species (primary keys) included in the pickle file. Second one, gives you the actual dictionary itself that SI is working with, in case you can not be bothered with SI or need to cross check something.

Moving on, printContent will present the user with the entire content of the pickle file in a structured manner. Sections with specie name in alphabetic order, all single values presented first in each section followed by the extensive volume log with corresponding dates. Note, the volume log could be very long after a while, time constraints disallowed for production of a systematic memory dump of the oldest log entries.

setValue function allows for initializing a new specie to the file as well as changing already existing specific data. updateVolLog function is actually similar to setValue but specifically designed for volume entry like when stock solution is dispensed, negative volumes to appropriate stock is written down. The function also allows for concentration entry alongside the volume in the case of refilling a stock solution where a volume still remains in the stock container. Concentration calculations are made to accommodate for both solutions to a new concentration and also written down. Lastly, OpenWriteAndClose is a simple command to save and overwrite work in the pickle file.

Figure 15 provides a simple overview for the content of SystemInventory.py.

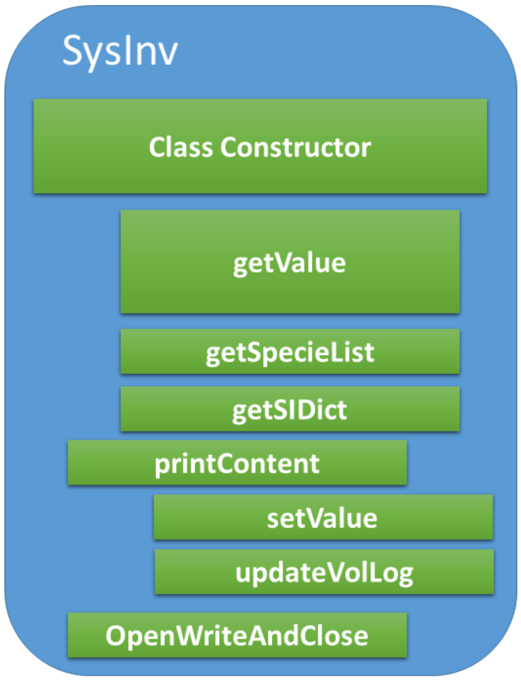


Figure 15: Function overview for SystemInventory class file.

4.2.3 BufferService.py

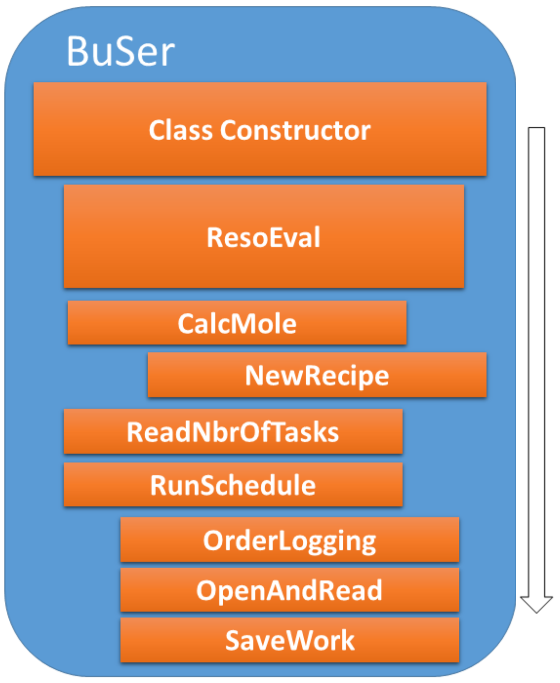
BuSer is most common short, BS is not used for somewhat obvious reasons. This class is intended to be a "One-Call-And-Done" for ordering buffer solutions to be produced, however it should be runable manually too. This class retrieves information on what resources are left from SI and make a determination if it can produce the requested solutions.

The constructor of this class is required to be run, it gives the user the possibility to enter optional arguments such as test/real mode and workpath.

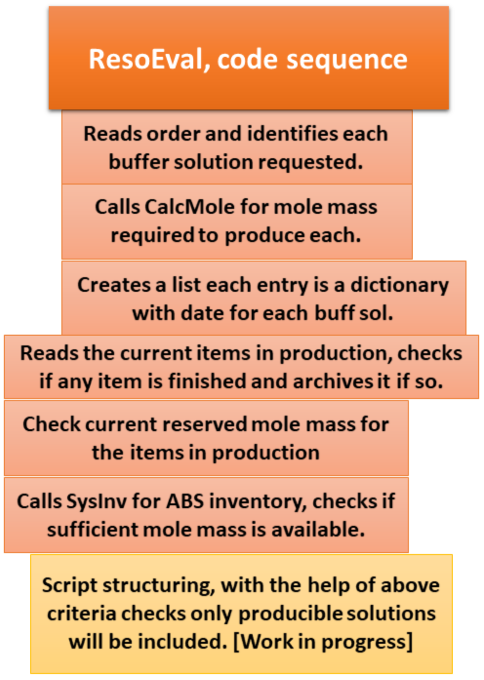
During a real mode run the main function ResoEval (Resource Evaluation) will read the designated pickle files and check if sufficient material (concentration * volume) exist in the system. This function needs a new "order" to be run, whereas an "order" is defined as a single or bundle of "task"s ("buffer solution"s). Similar to ordering food at a restaurant, you can in a single order issue for multiple meals of different kinds. ResoEval primary use is to generate the script to be used later for buffer production.

The intention to have this "order"-level is so that the script algorithm could give priority to tasks that was ordered before recent ones. Or in the case of a certain order given priority over other orders made before and after.

The function ResoEval gets a special figure detailing its operations in Figure 16b below.



(a) Breakdown of functions in the BufferService class file.


ResoEval, code sequence
Reads order and identifies each buffer solution requested.
Calls CalcMole for mole mass required to produce each.
Creates a list each entry is a dictionary with date for each buff sol.
Reads the current items in production, checks if any item is finished and archives it if so.
Check current reserved mole mass for the items in production
Calls SysInv for ABS inventory, checks if sufficient mole mass is available.
Script structuring, with the help of above criteria checks only producible solutions will be included. [Work in progress]

(b) The code sequence overview of ResoEval function in BufferService class.

Figure 16: Overview of BufferService class functions and the code sequence of a particular function called ResoEval.

4.2.4 system_buffer

Some maps of valves and pumps has been adjusted to reflect changes and additions to the physical system. Also an Orbit-side new Unit "Flask" has been added as well as introduction of function calls to SI for information regarding what these contain, as solution in these is expended for buffer production. Aforementioned earlier in SystemInventory.py the getValue generator function allows for the system_buffer file to update itself. Effect of this is that if a user specifies with help of SysInv that a new specie is installed in a new position the system_buffer file does not need further editing to reflect it.

Figure 17 describes the brief overview of what is new compared to what units existed before this project. To clarify, colValve is short for column valve and connects the outlet from the UV and conductivity sensor sequence back to the mixing flask. inlPumpsS2 allows for control of the inlet valves of pumpA2 and pumpB2 in system 2, the secondary inlet is for a water resource used in washing program.

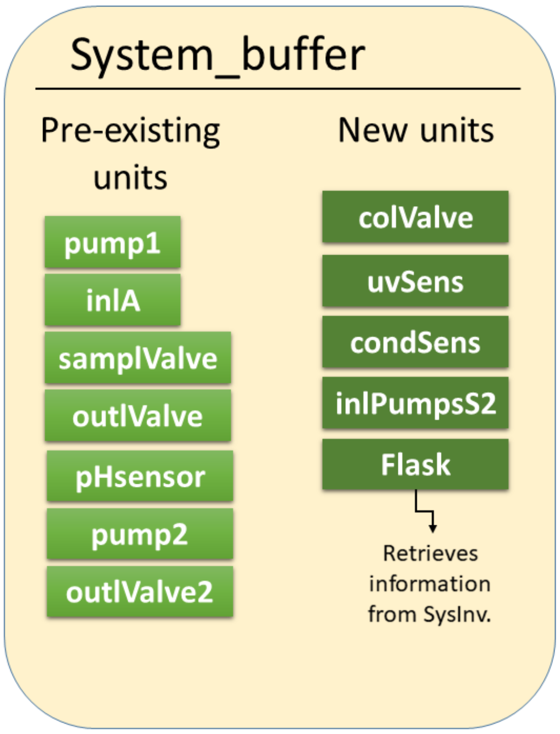


Figure 17: Overview of pre existing and new units for the system_buffer file.

4.2.5 process_buffer

Two new Events (e_Transfer and e_LoadLog), e_pHControl revisited, one PhaseOption o_FinalWash has been added. The contents of process_buffer could be compared to different programs in a dishwasher, selecting a program on a dishwasher makes it perform a series of specific instructions. In the case of process_buffer you yourself could create these programs and what they do for your specific system.

The PhaseOption o_FinalWash creates a sequential but extensive wash operation of all possible volume holding units like; pumps, tubing, analytical instruments and the mixing flask. An optional argument can be passed to specify how many times the mixing flask is rinsed with clean water, default is two. Earlier washing of acid and base pumps was non-existent so avoiding having the pump membranes in constant contact with strong agents like HCl and NaOH, as membrane rupture is not a desired incident.

The so called OnOffControl mimics the behavior in LoadStockAndPumpWash where desired volume is pumped through a valve, then changes from flask to waste and the flowrate is consequently turned off. This control method was explored in the AccVol testing and found to be surprisingly robust, for more details refer to its dedicated subsubsection 4.1.1.

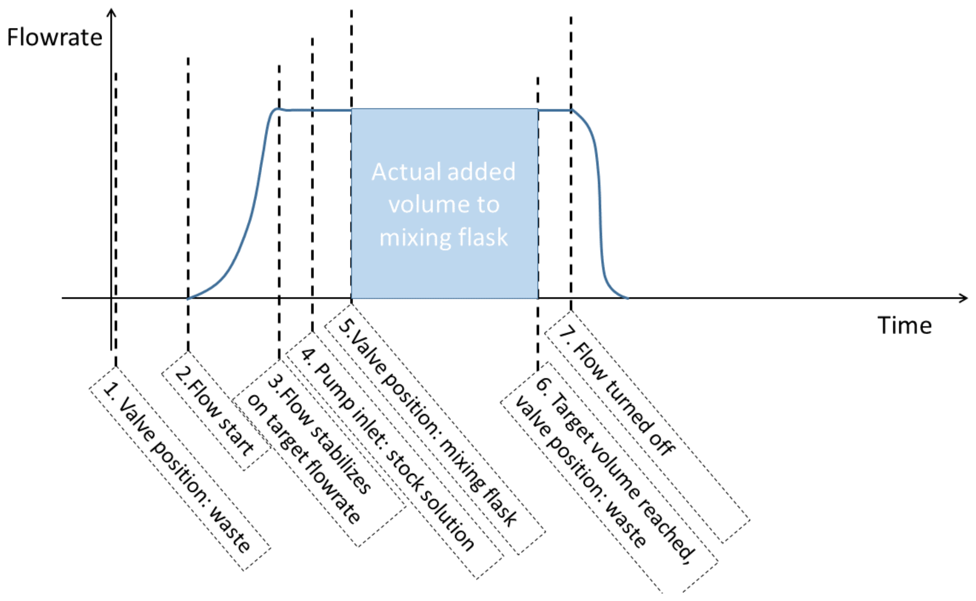

Flowrate
Time
Actual added volume to mixing flask
1. Valve position: waste
2.Flow start
3.Flow stabilizes on target flowrate
4. Pump inlet: stock solution
5.Valve position: mixing flask
6. Target volume reached, valve position: waste
7. Flow turned off

Figure 18: Illustration of OnOffControl-sequence in event LoadLog when loading stock solution to the mixing flask.

Moreover, the change to an Event from PhaseOption allows for easier conditional control, as in Events an accumulated volume signal is repeatedly retrieved from Unicorn. This is then used to check versus the desired volume input, set by the user. For comparison, PhaseOption allows you to state sequentially what should happen and relies on the computer clock for when instructions happen, rather than the actual process parameters.

Furthermore, e_Transfer is a simple event that pumps the solution out of the mixing flask and to a destination, at time of writing it is either waste or nextSystem (an ordinary flask). Additionally, it checks if the previous processes has water yet to be added to the buffer which is finally added to the mixing flask and then as well pumped to the destination. This is intended to work as a water plug forcing the final product to where it should be.

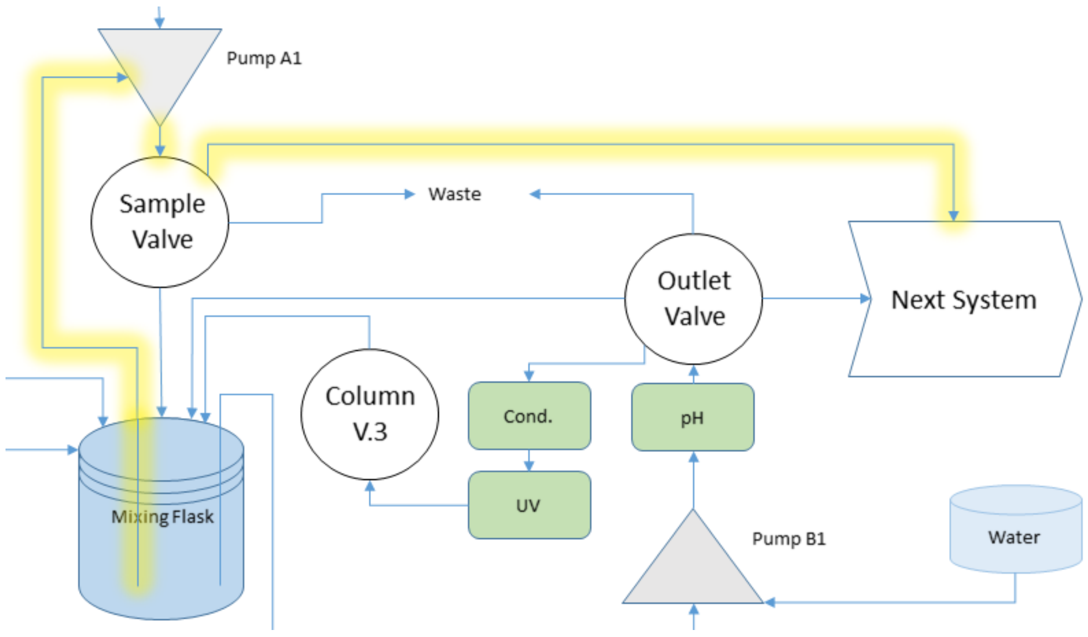


Figure 19: A crop out of the final flowchart configuration. Highlight of the intended path for Transfer event to use when moving solution from mixing flask to nextSys.

5 Conclusion

To sum up this project, the ÄKTA shows promising capability to be able to produce buffer solutions of sufficient quality for use in HPLC systems. However, more setup of digital infrastructure needs to be implemented as well as tested before an automated version could be employed.

6 Future Work

This is the bittersweet part, all the things that could be done or could have been done. The very obvious things for the system is completing the functions that actually makes the automation go around.

6.1 Items in this project that is Work-In-Progress

Firstly, fully flesh out the script generating function *ResoEval* to create priority based scheduling for the buffer system. Such a priority system could look very different depending on what one considers important. One example is; line up buffer solutions of similar ingredient content. Another could be; produce the most urgent solution first, lets say the one with lowest volume or highest rate of consumption. However, one could start with simply apply a "no priorities"-type of approach.

Secondly, small issues here and there but issues nonetheless. Creating a threshold check function in SysInv for initiating a call to BuSer. Apply the same code for checking the ABSs own inventory and finally summarizing this information for the user to confirm.

Process regulation is of course a very enticing concept to implement. Currently the start and stop of flow to and from the ÄKTA systems pumps are controlled by valves, also known as a step response. Control theory or simple manual tuning of a PID controller is possible to implement for the flow control but also the pH adjustment sequence.

Please refer to figure 20 for highlights of these issues.

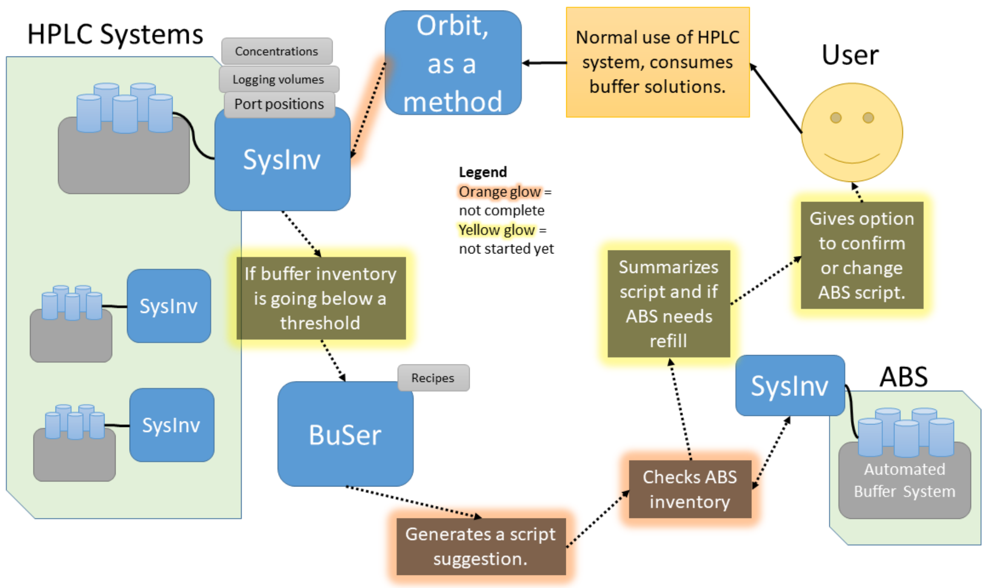

HPLC Systems
Concentrations
Logging volumes
Port positions
SysInv
SysInv
SysInv
Orbit, as a method
Normal use of HPLC system, consumes buffer solutions.
User
Legend
Orange glow = not complete
Yellow glow = not started yet
If buffer inventory is going below a threshold
Recipes
BuSer
Generates a script suggestion.
Checks ABS inventory
Summarizes script and if ABS needs refill
Gives option to confirm or change ABS script.
SysInv
ABS
Automated Buffer System

Figure 20: Highlighted points of Work-In-Progress (WIP) in yellow and orange. Yellow being items with no work started yet and orange is items which has a significant work progress but not finished for testing.

6.2 Sources of error in this project

Suspicion regarding the build up of error in volume documentation, meaning how well will the volume written in code correspond to the real volume in a source flask? Sure, the Accumulated Volume testing could indicate that little worry is warranted for an afternoon of running the system. Although, that test was performed prior to the actual implementation of variable volume as an attribute to code-side objects representing these source flasks. One could consider this as validation work that the writer fret he did not have time to perform.

6.3 Ideas for future work projects

A list of ideas that is beyond the scope of this project and for a later time as an expansion of the system, see below.

- Deterministic statistic function on used material for predictions of how much stock of each specie is required
- Research how solid material could be consistently measured and dissolved into a stock solution.
- Physical setup of "nextSys" in a more elaborate storage of finished buffer solution or actual pump work to system in need of said solution.
- Integrate a buffer filtration unit in the system.
- Create a direct link to a distilled water source, in our case: the in-house tap system.

7 References

[1] Irwin H Segel and Leigh D Segel. "pH and Buffers". In: *elS* (Aug. 21, 2001). DOI: `https://doi.org/10.1038/npg.els.0003117`.

[2] *Ullmann's Encyclopedia of Industrial Chemistry. Basic Principles of Chromatography.* 2011. URL: `https://search.ebscohost.com/login.aspx?direct=true&db=cat07147a&AN=lub.6244115&site=eds-live&scope=site` (visited on 05/29/2020).

[3] Mohammad Azam Mansoor. "Liquid Chromatography". In: *eLS* (June 2015). DOI: `10.1002/9780470015902.a0002679.pub3`.

[4] Niklas Andersson et al. "Design and control of integrated chromatography column sequences". In: *Biotechnology Progress* (Jan. 2017). DOI: `https://doi.org/10.1002/btpr.2434`.

[5] Niklas Andersson. *The Orbit Controller.* Research report. Department Chemical Engineering, Lund University, 2018.

[6] Frida Heskebeck. "*Towards autonomous antibody purification*". Master Thesis. Department of Chemical Engineering, Lund University, 2019. URL: `http://lup.lub.lu.se/student-papers/record/8981399` (visited on 05/15/2020).

[7] *Sample Injection - Valco 6 port valve.* ValcoInstruments Co. Inc. 2020. URL: `https://www.vici.com/support/app/app11j.php` (visited on 07/21/2020).

[8] *ÄKTA^{TM} Laboratory-scale Chromatography Systems, Instrument Management Handbook.* URL: `https://cdn.gelifesciences.com/dmm3bwsv3/AssetStream.aspx?mediaformatid=10061&destinationid=10016&assetid=16189` (visited on 07/17/2020).

[9] *ÄKTA Avant Chromatography System.* Cytiva. URL: `https://www.cytivalifesciences.com/en/us/shop/chromatography/chromatography-systems/akta-avant-p-06264` (visited on 08/20/2020).

[10] *BioProcess IC System.* Cytiva. URL: `https://www.cytivalifesciences.com/en/us/shop/chromatography/buffer-preparation-systems/bioprocess-ic-system-for-large-scale-buffer-management-p-03728#related-documents` (visited on 07/24/2020).

[11] *FlexAct® BP.* Sartorius. URL: `https://www.sartorius.com/en/products/process-filtration/flexact-single-use-automated-solutions` (visited on 07/24/2020).

[12] *Python Documentation 3.8.3.* Python Software Foundation. URL: `https://docs.python.org/3/` (visited on 06/05/2020).

8 Appendix

8.1 Basic Python Concepts

A few basic python concepts is recommended to understand some content of this report. This entire section dedicated to Python is based on experience and documentation available at their documentation website for version 3.8.3. [12]

8.1.1 Attributes, or Common data types (list, dictionary, string, float)

Starting of, in Python coding there is different ways to allocate computer memory to values like numbers, text et.c. Those ways mentioned here are string, float, list and dictionary. Reading Python documentation a collective name for these value types is: attributes.

- String; is one single attribute containing text, an example would be: string1 = "Hello World"
- Float; is also a single attribute but contain a number with decimals, i.e. : number1 = 3.14.
- List; can contain multiple attributes of different kinds of attributes, a list could even be empty with no amount of attributes. An important characteristic of a list is that the order in which attributes appear is always the same, until changed. ListExample1 = [400, 1, "second", 3], each time you use this list the number 400 will appear first in order and the text value "second" will appear as the third value.
- Dictionary; similar to a list it can contain none, single or multiple values. However, these appear in no particular order but instead is connected to a so-called "key", which is a string unique to that value it is assigned in said dictionary. DictEx1 = {'food' : 'avocado', 'not food' : 3.14}, if you call for this dictionary like this: DictEx1['food'] it will give you the string 'avocado'

These are only a few attribute types, but hopefully the reader could attain some confidence approaching the material ahead.

8.1.2 def

For those familiar with programming in languages like Java, this equals to a function. For those unfamiliar, this is effectively assigning a piece of code to a specific name with the option to give it input parameters. Very useful if you know a piece of code will likely need to be used at multiple points in a larger project, then a function is highly recommended.

Example would be to writing code that calculates the value of an equation, lets say Einsteins $E = mc^2$. You suspect that the mass will vary from case to case, and therefore allow the mass m to be passed as an input argument to a function involving this equation. Every time you call this function it will return the value of energy, assuming we allow the light of speed to be a constant within the function.

8.1.3 class

A class is effectively the super ordinate of a function, a class can hold many functions and attributes. This is an effective way to keep track of a high number of functions and what each can do, especially when multiple classes of the same base class is created with slightly different parameter inputs.

A simplistic example would be two pumps of the same design could be expected to behave similar, thus you could code that is reusable for that design and in turn for both pumps. But in reality these pumps do different tasks, one pumps fluid from a reservoir to a reactor tank and the other empties fluid from the reactor tank to a storage container. In code, a distinction between these two pumps with a base class of the same type but with different parameters which could explain how they are connected, when to pump and at what flow rate.

8.1.4 Pickle, a serializing tool

A straightforward way to write down information with a purpose to be used later is having your program write text in a .txt-file. However, this method has a couple of issues; one being security, as these files can be read and altered by anyone having access to said .txt-file. A serialised file offer some protection from intrusion.

Nevertheless, the main purpose is to change a complex data structure into a compact byte sized unit and thus increasing storage efficiency.

The reader might reminisce seeing files with extensions like xml and json, these are information storage files with a serialisation unique to each type of file. With the appropriate serialization tool these files can be read and ability reconstruct their complex data counter part.

Pickle is the tool built in Python and is a good choice if information with Python specific data types is desired to be saved for later use. Data types in the case of this project is dictionaries but also some functions.