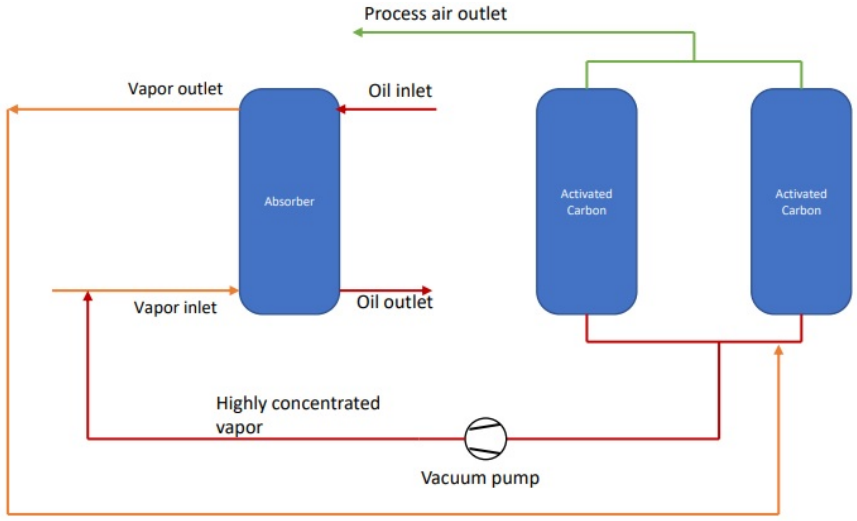
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Sviatoslav Eroshkin

Volatile Organic Components Adsorption on Activated Carbon

Master's thesis in Natural Gas Technology
Supervisor: Even Solbraa
Co-supervisor: Eivind Johannessen
June 2022



NTNU
Norwegian University of Science and Technology

Sviatoslav Eroshkin

Volatile Organic Components Adsorption on Activated Carbon

Master's thesis in Natural Gas Technology
Supervisor: Even Solbraa
Co-supervisor: Eivind Johannessen
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This master thesis was completed at the Norwegian University of Science and Technology in 2022. The master thesis consists of a literature study, theory evaluation, derivation of the main equations, simulation, and validation of volatile organic components adsorption on activated carbon using Python.

The master thesis is done in collaboration with Equinor. I greatly appreciate the work of my supervisors, Even Solbraa and Eivind Johannessen, who guided me through this project.

Summary

Some volatile organic components may cause serious human health and environmental threats. These components are "global warming agents" that contribute to climate change. While the negative impact of methane and carbon dioxide on climate is highly studied, the influence of non-methane volatile organic components is less known. These gases affect the earth-atmosphere energy balance by the production of ozone and a reaction with the hydroxyl radical, which in turn leads to a longer atmospheric lifetime of methane. A more efficient purifying is thus becoming critical for companies that use such components in their processes. Adsorption is one of the most used techniques for this.

The proper development of the adsorption process with high performance involves the design of a model that can describe the dynamic of adsorption, considering all relevant transport phenomena.

The primary purpose of the study is to simulate the adsorption of methane, ethane, propane, butane, nitrogen, and carbon dioxide on an activated carbon bed. The vacuum pressure swing adsorption model in Python is developed. It is based on the Extended Langmuir isotherm, a linear driving force approximation for mass transfer, and mass, momentum, and energy balance equations. One cycle consists of 4 steps: co-current pressurization, adsorption, counter-current blowdown, and counter-current nitrogen purge. The model is validated by comparison with the commercial software Aspen Adsorption and available experimental results. An experimental comparison is implemented with multi-component and two-component adsorption results. Still, some adjustments are required to get a good comparison with experimental data. Furthermore, it was shown that the model gives a good result with adsorption phenomena understanding. The model has been applied to analyze the adsorption dependence on various crucial parameters, such as adsorption and desorption pressure, inlet and purge gas flow rates, adsorption step time.

Table of Contents

List of Figures

List of Tables

Nomenclature

Greek Letters

ϵ	Total porosity	
ϵ_{ext}	External porosity	
ϵ_{int}	Internal porosity	
μ	Dynamic coefficient of viscosity	Pa/s
ρ	Density	kg/m^3

Latin Letters

ΔH	Heat of adsorption	$J/kmol$
A	Adsorbent specific surface area	$1/m$
b	Affinity coefficient	$1/bar$
b_∞	Affinity constant	$1/bar$
C	Molar concentration	$kmol/m^3$
c_p	Specific heat capacity	$J/kmolK$
d	Adsorbent diameter	m
F	Flow rate	$kmol/s$
h	Heat transfer coefficient	W/m^2K
h_W	Heat transfer coefficient through the wall	W/m^2K
IP	Isotherm parameter	W/m^2K
J	Adsorption source term	$kmol/m^3s$
k	Mass transfer coefficient	$kmol/m^3s$
M	Total molar mass in the control volume	$kmol/m^3$
MW	Molecular weight	g/mol
P	Pressure	bar
Q	Isosteric heat of adsorption	$J/kmol$

q_{pump}	Pump volumetric flow rate	m^3/s
q_{purge}	Purge volumetric flow rate	m^3/s
R	Universal gas constant	$J/molK$
r	Radius	m
r_0	Pellet radius	m
S	Cross-sectional area of the adsorber	m^2
T	Temperature	K
u	Velocity	m/s
U_0	Vector of conservative variables	
V	Volume	m^3
w	Concentration in the solid phase	$kmol/kg$

Subscripts

ads	Adsorption
des	Desorption
eq	Equilibrium
i	Particular component
j	Particular component
max	Maximum
PG	Purge
$press$	Pressurization
sup	Superficial

Abbreviations

AC Activated Carbon

EL Extended Langmuir

GAC Granular Activated Carbon

CFD Computational Fluid Dynamics

D-R Dubinin-Radushkevich isotherm

HC Hydrocarbons

LDF Linear Driving Force

PSA Pressure Swing Adsorption

TSA Temperature Swing Adsorption

IAST Ideal Adsorption Solution Theory

VOC Volatile Organic Components

VPSA Vacuum Pressure Swing Adsorption

VRU Vapor Recovery Unit

1 Introduction

Volatile organic components (VOCs) are organic compounds with boiling points less than or equal to 250 $°C$ at standard pressure. VOCs contributes to air pollution, which may cause severe risks to human health. VOCs are also considered "global warming agents" [1]. Figure 1 shows the top 7 countries with the highest VOCs emissions per capita. Norway has significantly reduced VOCs emissions from above 70 kg/capita in 2005 to 28.5 kg/capita in 2019 and is now in 6th place.

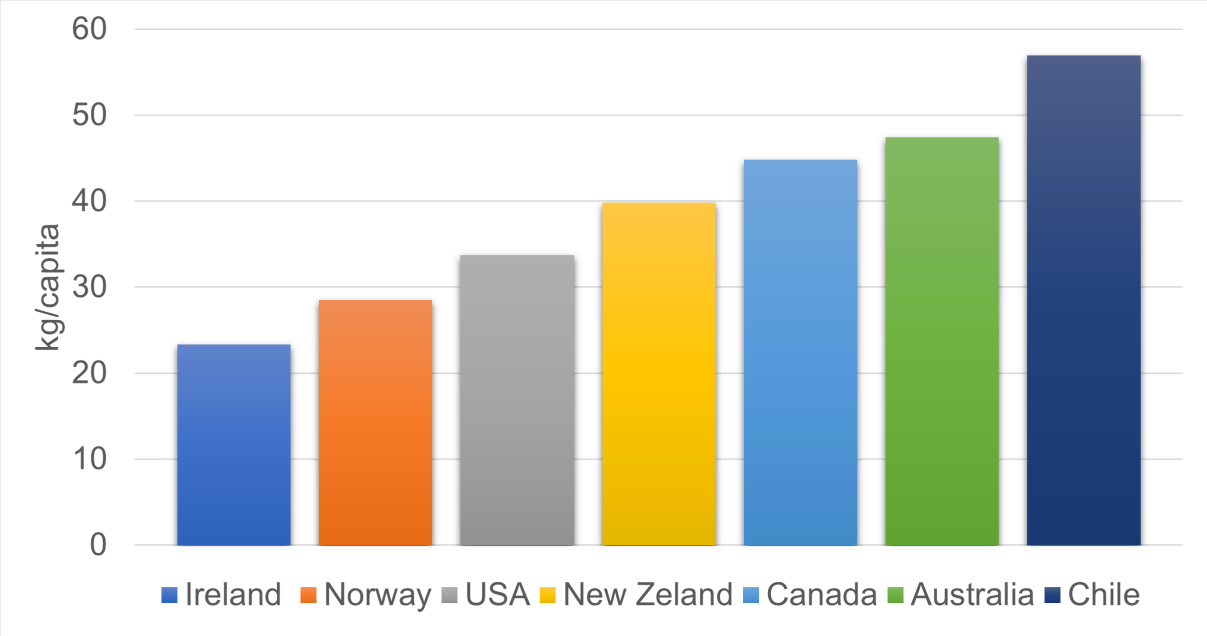


Figure 1: VOCs emissions per capita by country in 2019 [2]

Some components evaporate from crude oil during storage, loading, and unloading operations, and these compounds contribute significantly to VOCs emissions. The composition of crude oil varies, but the majority constitutes hydrocarbons (HC). The lightest components tend to evaporate more (due to their high vapor pressures and low boiling points [1]) such as the first alkane group: methane, ethane, propane, and butane. These components affect human health and air quality by producing photochemical ozone (O_3) and other harmful oxidants, which increase the atmospheric oxidizing. Today, surface O_3 pollution has become a problem in China. This country reported that the photochemical ozone concentrations continuously increased in the last decade [3].

Therefore, the use of vapor recovery units (VRU) to purify vapor during oil operations gained popularity in the 90s, with the principle of adsorption of VOCs on activated carbon (AC) followed by regeneration under vacuum [4]. This technology must be designed and used most efficiently to decrease VOCs emissions in the hydrocarbon industry.

Thus, the master thesis aims to develop a mathematical model in Python to predict the competitive VPSA (Vacuum Pressure Swing Adsorption) of methane, ethane, propane, butane, nitrogen, and carbon dioxide on an AC bed.

This will be achieved by introducing:

1. Literature review of the models for multicomponent adsorption on AC.
2. Theory of the process that includes basic definitions, process description, a multicomponent competitive adsorption isotherm model that can predict equilibrium adsorption of n-component VOCs mixture using pure-component isotherm experimental data, mass transfer model and governing transport phenomena with appropriate boundary and initial conditions.
3. The used methods including the numerical strategy of a model solution in Python as well as the developed model in commercial software Aspen Adsorption.
4. Validation of the model with available experiments (Cavenati and co-authors experiments [5] and Equinor VOCs experiments [6]) and commercial software (Aspen Adsorption) results.
5. Parametric analysis of the developed Python model on various crucial parameters, such as adsorption and desorption pressure, inlet and purge gas flow rates, adsorption step time.
6. Conclusion and proposals of future work.

2 Literature Review

Adsorption of VOCs on a fixed-bed AC is commonly used to reduce emissions [4, 7–9]. The use of VRU has gained popularity in the 1990s. Today 95 % of all new VRU are based on AC adsorption followed by vacuum regeneration [4]. When VOCs are treated using non - destructive methods, such as adsorption, absorption, and condensation, the recycled VOCs can be reused [8]. During this project, the non-destructive method based on AC VPSA will be modeled.

One of the main difficulties in mathematical adsorption modeling is the correct definition of adsorption thermodynamics and mass transfer. The research on the VOCs adsorption equilibria and mass transfer in porous materials ensures accurate VOCs adsorption modeling. Wang et al [9] discussed the adsorption performance of VOCs on AC. The following isotherms have been used for VOCs adsorption equilibrium prediction: Langmuir, Freundlich, Dubinin-Radushevich, Temkin, Flory-Huggins, Hill. The Extended Langmuir (EL) isotherm is widely used due to the model's simplicity. However, the accuracy of this isotherm type is questionable. Even though it has a theoretical basis behind it, the assumptions place a significant restriction on the applicability of this isotherm. The EL model is an explicit model that is preferred over its implicit counterparts, for example, IAST (Ideal Adsorption Solution Theory) [10] , due to the computation complexity of the last one. An essential shortcoming of the EL model is the neglect of the adsorbate size effect [11]. However, the results of studies that use EL are usually coherent with experimental data as in [7, 12, 13]. It should be mentioned that in this master thesis more components will be considered, which may give a negative effect of using simple EL model. The Freundlich isotherm, a well-known empirical model, can describe non-ideal VOCs adsorption but it cannot predict the adsorption equilibrium well under low pressure. Also, the D-R (Dubinin-Radushkevic) model is thermodynamically consistent at medium and high relative pressures, except for low loadings [9].

The main components considered in this project are methane, ethane, propane, butane, carbon dioxide, and nitrogen. The literature was studied to investigate the available VOCs adsorption data. Experimental results for natural gas components equilibrium are available in Esteves et al. work [14]. The adsorption experiments results are presented for the pressure and temperature ranges of 0–9 MPa and 273–325 K. The AC which Esteves et al. used is a coal-based, extruded carbon (2 mm diameter pellets).

The adsorption of VOCs is an exothermic process with high heat release that can be estimated from the temperature dependence of the adsorption isotherm with the assumption of constant heat release over a wide temperature range. Esteves et al. [14] also showed the variation of the heat of adsorption with pressure for carbon dioxide, nitrogen,

and methane. The value of the isosteric heat of adsorption decreases with the loading increase, which indicates that AC is energetically heterogeneous. For example, the heat of carbon dioxide adsorption decreases from 23 kJ/mol in the low-pressure region until leveling off at 20 kJ/mol.

Also, the mass transfer resistance between gas and solid phases plays a great role in adsorption modeling. The adsorption mass transfer model defines the mass accumulation in the solid phase. The typical models include three types: local equilibrium model, linear driving force (LDF) model, and pore diffusion model [15]. The most common is to use the LDF for estimating the mass transfer as in [7, 12, 13, 16]. Kim et al. [12] indicate that the optimal parameters of mass transfer constants should be obtained from the comparison of the numerical results with experimental data. These parameters should be assumed such that the model solution is the closest to the whole range of available experimental data. For example, Vilardi et al. [17] use the result of Cavenati's experiments to get the values of LDF coefficients for methane and carbon dioxide.

Quite a few papers have studied the PSA (Pressure Swing Adsorption) for different components. From the technical point of view, these works are interesting because the mechanism of VOCs VPSA is in many ways similar. In particular, it is interesting to see what has been achieved in optimizing the process and which are the main parameters considered.

The PSA process exploits the change of the adsorption equilibrium with a change in system pressure. The process efficiency depends on many factors such as the number, sequence, and time of PSA steps, flow rate of the gas, gas composition, adsorption pressure, and others. For example, Kim at al. [12] investigated biogas mixture adsorption on an AC molecular sieve with a four-bed and seven-step PSA system. A significant improvement to the PSA process was the implementation of the equalization step: reducing the energy consumption in the pressurization step by means of the purified product from the second adsorber. The experimental setup and numerical model showed that a cyclic steady-state process was obtained after 13 cycles. Furthermore, Kim at al. [12] indicate that optimum conditions for the separation highly depend on such parameters as the adsorption pressure, desorption pressure, purge and feed rates (the ratio of the purge gas flow rate to the flow rate of desirable component feed rate). Ahn et al. [18] also showed the importance of purge to feed ratio, adsorption pressure, feed flow rate, step times, and carbon ratio for layered two- and four-bed 6-step PSA processes for H_2 recovery from coal gas. The layered bed means that the bottom of the adsorber is filled with one adsorbent, which works as the primary separator, while the top, which is filled with another adsorbent, purifies the final stream. The result of the study is that higher purity of the product can be achieved with layered beds; however, less recovery can also be observed. The study also showed

that product purity and adsorbent recovery changes were more significant by varying the operating variables in the two-bed process than in the four-bed process.

In these studies, the bed geometrical characteristics are fixed, and their impact on adsorption efficiency is not properly investigated. Also, the effect of pressure drop and energy balance on PSA efficiency are often considered negligible [19]. However, Vilardi et al. [17] showed that exergy and energy analysis of the biogas upgrading in the PSA process allows proper considerations of the system's costs and environmental impact.

3 Background

3.1 Basic Definitions

The selective mass transfer of components to the bulk or surface of adsorbent is called sorption. Absorption is the process when components are attached to the bulk of the solid. Adsorption is the process when one or more components are attached to the surface of the solid. Adsorption which is driven by interaction between solid and gas phases can be used to remove undesirable components from the product stream. The solid material that adsorbs components is called adsorbent, and solute is usually called adsorbate.

The affinity of molecules to a particular adsorbent depends on molecular characteristics and thermodynamic conditions. The attachment to the solid surface is the result of van der Waals forces, and hydrophobic interactions [20]. The reverse process of adsorption is called desorption.

The first essential parameters that need to be considered in the adsorption process are the types of porosity and density. There are two main types of void fractions: external and internal. As names denote, external (interparticle) voids define the ratio of the space not covered by particles to the total material volume, while internal voids (intraparticle fraction) is the ratio of space in the particles to the total volume. The total void fraction (total porosity) is the sum of interparticle and intraparticle fractions (provided that the porosity is determined in relation to the total volume of the adsorber).

Density is defined as mass per reference volume. There are several meaningful ways to calculate the densities which are summarized in Table 1 [21].

Table 1: Definition of voids and densities

Total system volume	Defined as geometrical volume of bed.
External void volume	Defined as total volume multiplied by external porosity. The volume not covered by particles.
Internal void volume	Defined as total volume multiplied by internal porosity. The free volume inside the particles.
Bulk density	Defined as material mass per total system volume.
Skeletal or solid density	Density if there was a pure solid cube (without voids).
Envelope or particle density	The density is found by drawing a hypothetical envelope around particles and deviding the mass of all particles by the volume of their envelopes.

There are two main parts in AC modeling:

1. Governing transport model that describes gas velocity, temperature, pressure and

other parameters in the adsorber.

2. Source term modeling which includes equilibrium model and mass transfer model. Equilibrium model represent how much of the adsorbate can be adsorbed, and mass transfer model shows how fast equilibrium can be reached.

The above-mentioned models are used in all the four steps of the VPSA system. The main difference between PSA and VPSA is the desorption pressure. For VPSA desorption pressure is less than 1 bara. However, they consist of the same steps:

1. Pressurization. This step is used to increase the adsorber pressure from some value to the required one by pumping gas into the adsorber.

2. Adsorption step. The primary step in the cycle when the outlet product is achieved.

3. Desorption step. The inverse action to pressurization when the pressure in the adsorber is reduced from some value to the required value. This step causes the adsorbate to desorb.

4. Purge step. The purge gas is added to the system to decrease the partial pressure of the adsorbates in the adsorbent and cause them to desorb.

5. Equalization step. This step was not considered in the master thesis but can be well added to improve the adsorption efficiency. This step is used to reduce the energy consumption of a compressor in the pressurization step by utilizing the purified product from another bed.

All these steps can be done in the following ways:

1. Co-current. This means from the adsorber bottom (inlet) to the adsorber top (outlet).

2. Counter-current. This means from the adsorber top (outlet) to the adsorber bottom (inlet).

In the master thesis, co-current pressurization, adsorption, counter-current desorption, and counter-current purge will be considered. However, the choice of either co-current or counter-current ways will affect the adsorption performance and, thus, could be analyzed in the optimization process.

3.2 Activated Carbon

The properties of the material used in a particular adsorption process is crucial. The performance highly depends on how the solid performs in both adsorption equilibrium and mass transfer. For example, an adsorbent with high equilibrium capacity but low mass transfer will have low performance because the required time of adsorbate to reach equilibrium is large. On the other hand, solid with high mass transfer but low adsorption equilibrium will also have low performance because a large amount of adsorbent is needed. Therefore, one should choose the adsorbent that provides a favorable combination of fast mass transfer and high equilibrium capacity. This solid should have a large surface area and a pore network for the transport of molecules.

Among the solids used in industries, AC is used to remove VOCs because of good adsorption equilibrium and mass transfer characteristics to these compounds. The raw material has a certain degree of porosity, but the internal surface is significantly increased by activation (high-temperature oxidation). The outer shape of the activated carbon is usually in two main forms: granular (GAC) or extruded into pellets (PAC). The first form is more economical, while the second form is more efficient (less tendency to form dust and pressure drop is less) [4]. The experiments were performed at Equinor with PAC showed in Figure 2 [6]. The pellets diameter is approximately 2-4 mm which is the same size as the PAC used by Esteves et al. [14]. The data obtained in Esteves et al. [14] study is used in the master thesis to describe the VOCs adsorption equilibrium model.

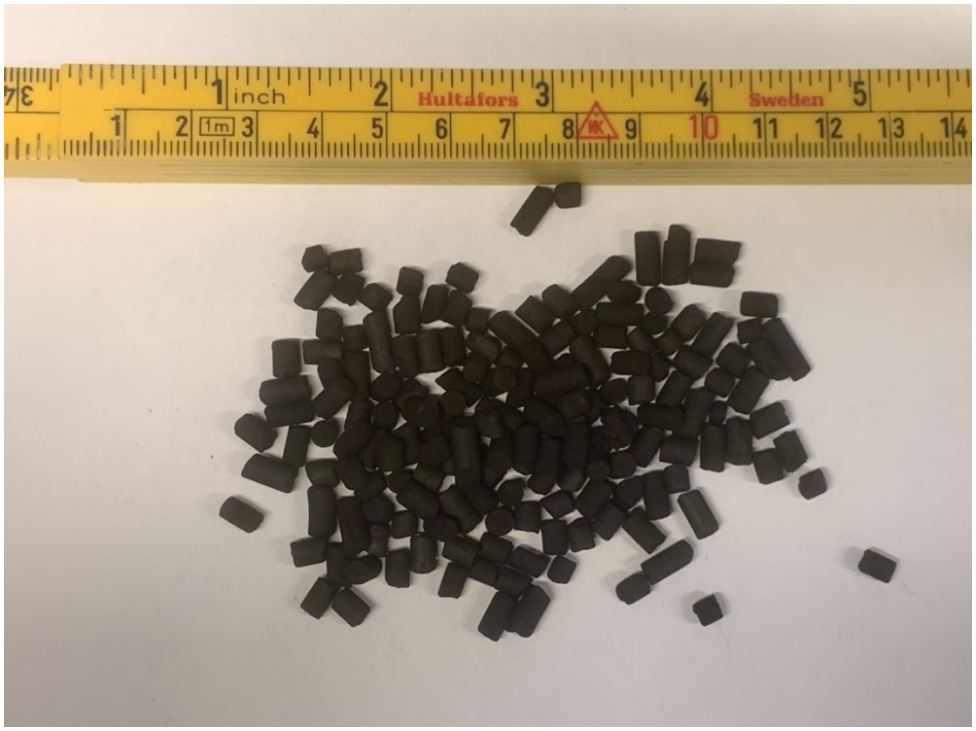


Figure 2: Used PAC at Equinor VOCs experiments

3.3 Process

The VPSA process can be done in different ways (with 2, or more adsorbers, with 3 or more steps with co-current and counter-current ways). This section describes the process closest to the Equinor VOCs experiments [6]. The flow diagram of VOCs adsorption is shown in Figure 3. The process consists of two beds: one in adsorption mode and another in regeneration mode. The adsorber outlet is a pure product during the adsorption process and highly concentrated vapor during the regeneration process. This vapour is then recycled in the absorption system [4].

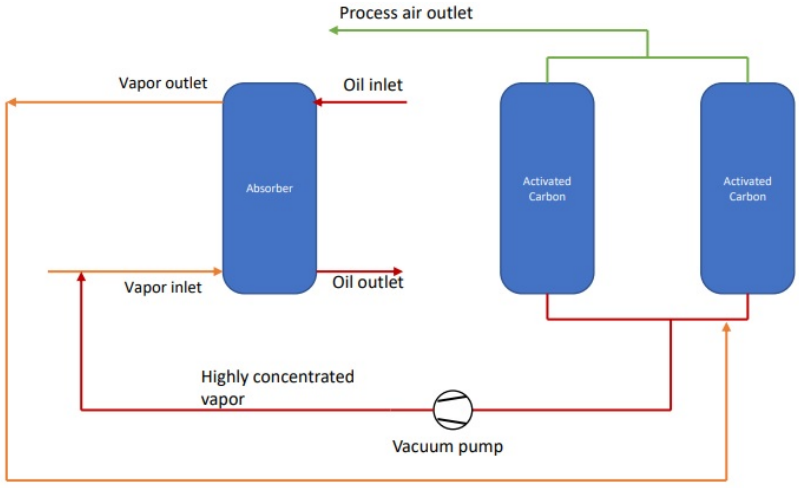


Figure 3: The simplified process flow diagram of VOCs adsorption

The adsorption process with two beds is shown in more detail in Figure 4. The four main steps in the process are pressurization, adsorption, desorption, and purge. The steps are switched by employing valves.

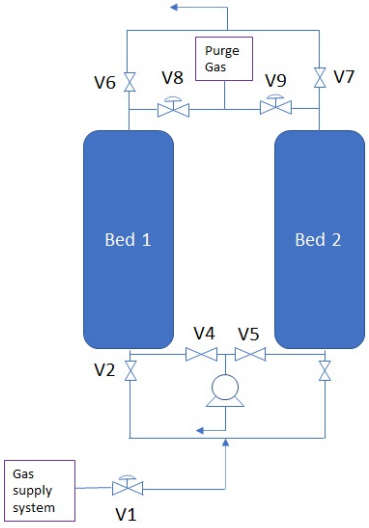


Figure 4: Two bed adsorption scheme

According to the Equinor VOCs experiments [6], the following procedure is used. The top of the adsorber is closed during pressurization and the system's pressure increases (see Figure 5). The pressure increase occurs from a specific low pressure after regeneration to that required for adsorption. Next, the valve at the top of the adsorber opens, and a product is achieved at the outlet. After a certain period, a breakthrough of specific components occurs, and it is necessary to switch the adsorber to the regeneration mode. There is no inflow at the bottom during regeneration. Also, the valve to the vacuum pump opens, and the vacuum pump turns on. After a specific time, inert gas is supplied from above while the pump operates. This step is called "purge". As a result, the adsorbates' partial pressure decreases, which causes them to desorb better. After the purge, the whole cycle repeats. A certain amount of non-desorbed gas remains in the adsorber, which affects the following process and changes the result of the next cycle. However, the system gradually converges with an increase in cycles to the so-called "cyclic-steady-state".

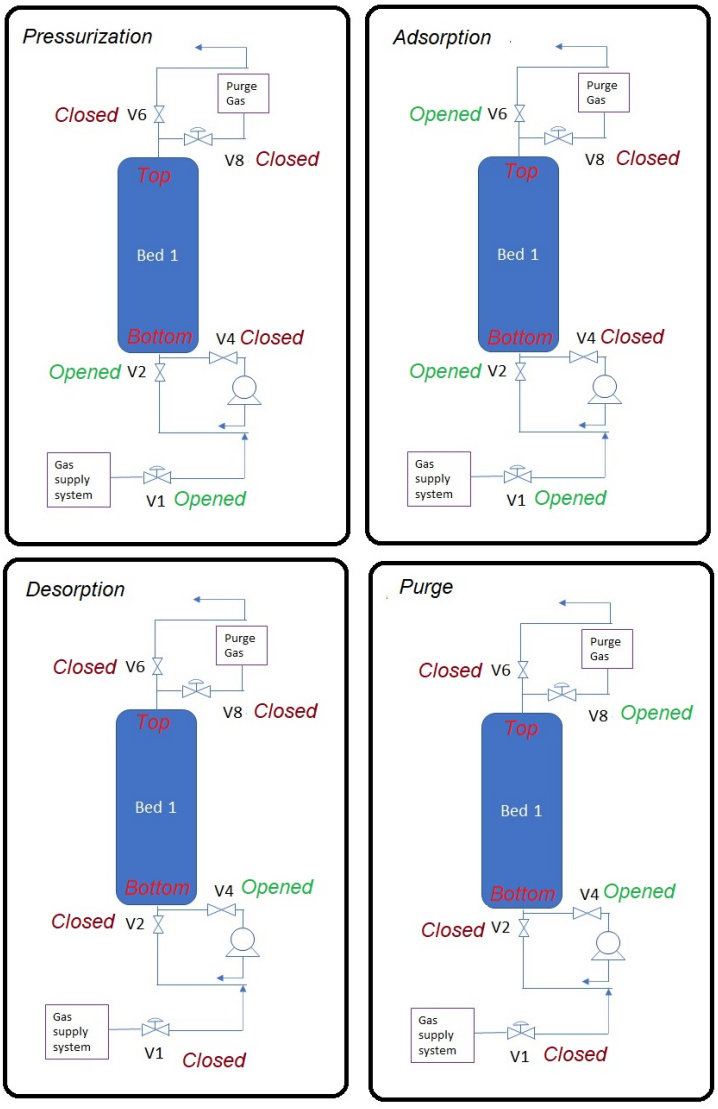


Figure 5: Cycle organization

3.4 Governing transport phenomena

The complexity of the model can vary significantly. A model aims to capture the main physics to reach a particular set of goals. In the optimization task, the model complexity needs to be limited in some instances to reduce the computational time. However, the correct sophistication level must be carefully examined. The primary strategy is to start with a simple model, and complexity should be added if required.

3.4.1 Langmuir equilibrium theory

Many models (isotherms) have been proposed for presenting adsorption equilibria. The EL isotherm is one of the simplest models, yet one that works surprisingly well for many multicomponent systems [20]. Assumptions made by Langmuir of pure component adsorption may also be used in describing multicomponent systems. These assumptions are:

1. The adsorption process is localised, and each cite of adsorbent accumulates only one molecule from the system.
2. No interaction between adsorbate molecules.
3. Surface is homogeneous and constant heat of adsorption over all cites is assumed.
4. No mobility (surface diffusion) on the surface.

The Langmuir equation for multicomponent adsorption [10]:

$$\frac{w_{eq,i}}{w_{max,i}} = \frac{b_i P_i}{1 + \Sigma_{j=1}^{N} b_j P_j} \tag{1}$$

here, $w_{eq,i}$ is an equilibrium concentration of the solute in the solid phase $[kmol/kg]$, $w_{max,i}$ is the maximum solute concentration in the solid phase $[kmol/kg]$, b_i is an affinity coefficient of component i $[1/bar]$, P_i is a partial pressure of component i $[bar]$, N - number of components, b_j is an affinity coefficient of component j $[1/bar]$, P_j is a partial pressure of component j $[bar]$.

Eq.(1) is known in the literature as an EL isotherm. Even though this equation has a sound theoretical background, the assumptions used during the derivation of this equation impose strong limitations on the theory's applicability to practical problems. However,

this isotherm is widely used due to its simplicity. The results are usually coherent with experimental data [7, 12, 13]. Eq.(1) may be expressed in terms of four isotherm parameters: IP1, IP2, IP3,IP4:

$$w_{i,eq} = \frac{(IP1_i - IP2_i T) IP3_i \exp\left(\frac{IP4_i}{T}\right) P_i}{1 + \Sigma_{j=1}^{N} IP3_j \exp\left(\frac{IP4_j}{T}\right) P_j} \tag{2}$$

here, $IP1$ is an isotherm parameter 1 $[kmol/kg]$, $IP2$ is an isotherm parameter 2 $[1/K]$, $IP3$ is an isotherm parameter 3 $[1/bar]$, $IP4$ is an isotherm parameter 4 $[K]$, i is a particular component i, j is a particular component j.

Comparing Eq.(2) with Eq.(1) : $w_{max,i} = (IP1_i - IP2_i \cdot T)$ and $b_i = IP3_i \exp\left(\frac{IP4_i}{T}\right)$.

Coefficient b is the parameter that shows the ratio between adsorption and desorption rates. This parameter is called the affinity coefficient and may be expressed in terms of adsorption heat, temperature, and affinity constant [10]:

$$b_i(T) = b_{\infty,i} \exp \frac{Q_i}{RT} \tag{3}$$

here, $b_{\infty,i}$ is an affinity constant of a component i $[1/bar]$, Q_i is an isosteric heat of adsorption of a component i$[J/mol]$, R is a universal gas constant $R = 8.314$ $[J/molK]$, T is temperature $[K]$.

The adsorption process is exothermic ($Q < 0$), and heat is released during the process. From Eq.(3), one may see that the affinity coefficient decreases with an increase in temperature. Thus, the heat of adsorption is an important parameter in the adsorption system design that can be derived from the equilibrium isotherm model based on Eq.(3).

3.4.2 Mass transfer considerations

The components to be adsorbed need to find their way to the adsorbent particle by convection through the channels, diffusion through the boundary layer around each particle, diffusion transport inside the particle until they reach a vacant site. Finally, they adsorb to the solid surface by van der Waals forces and hydrophobic interactions with heat release. Micropore structure provides the adsorption capacity, while mesopores and macropores play a significant role in the gas passage (see Figure 6). The pores of the solid can be categorized as shown in Table 2.

Table 2: Pores categorization

Micropores	Smaller than $2 \cdot 10^{-9}$ m
Mesopores	$2 \cdot 10^{-9}$ to $50 \cdot 10^{-9}$ m
Macropores	Larger than $50 \cdot 10^{-9}$ m

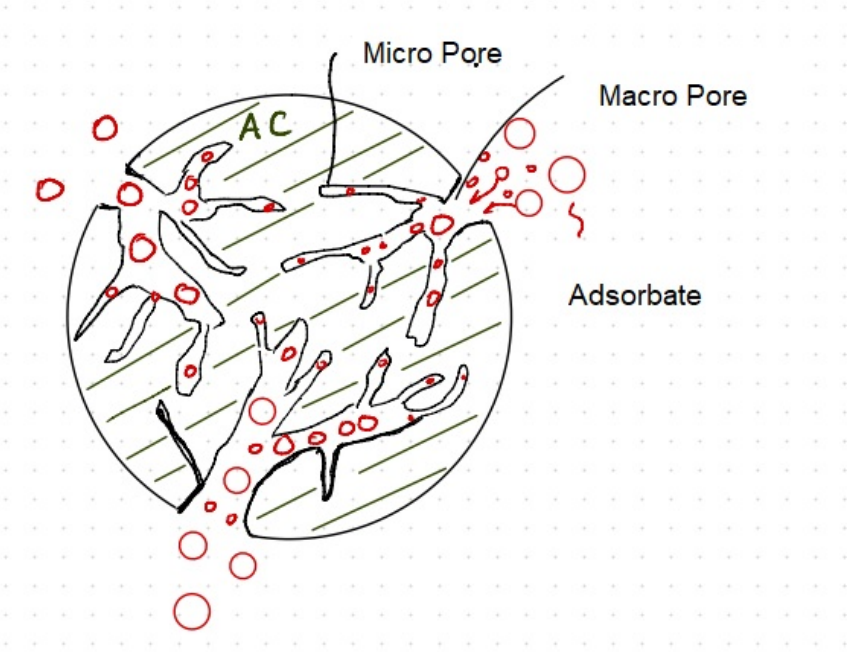


Figure 6: Particle structure of activated carbon

Figure 7 represents the concentration profile of adsorbate along the bed with relatively high mass transfer resistance. The mass transfer process starts immediately upon entering the bed. When the bed reaches equilibrium, the mass transfer tends to zero, and the front moves along the bed like a wave. The length where concentration is changing is called a mass transfer zone. When the mass transfer zone reaches the bed outlet, the breakthrough of effluent occurs. The length where the solute is not present is called unused (fresh bed). The length of the adsorber, where the solid and gas phases are at equilibrium, is called the equilibrium zone (saturated bed). A high mass transfer rate will make the mass transfer zone shorter and the concentration profile along the bed would be steeper. Therefore, higher mass transfer favors the adsorption process, and breakthrough occurs later with a more saturated bed.

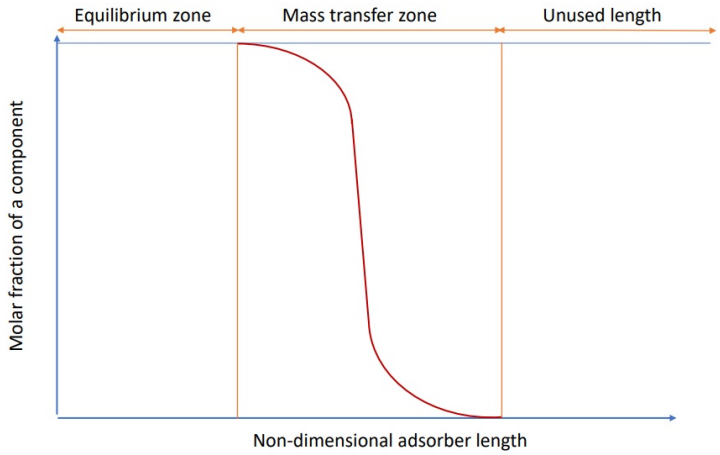


Figure 7: Mass transfer zones

Proper design of mass transfer equations is essential for numerical adsorption modeling. The following mass transfer models are commonly used: local equilibrium, pore diffusion, and LDF.

The local equilibrium model assumes no mass transfer resistance between gas and solid phases. Therefore, the change in equilibrium immediately causes a change in the amount of adsorbed phase:

$$w_i = w_{eq,i} \tag{4}$$

$$\frac{\partial w_i}{\partial t} = \frac{\partial w_{eq,i}}{\partial t} \tag{5}$$

here, w_i is a solid phase concentration of component i $[kmol/kg]$, t is time $[s]$.

The model has limited practical applications because it assumes no mass transfer resistance. Only in the case of ideal adsorption the simulated results would predict well the actual process [15]. At the same time, it is a good start to model with this assumption and then proceed to more complex, for example, pore diffusion and LDF models.

The diffusion process of the adsorbed gas from the external surface to the interior of a particle may be described by the pore diffusion model [15] (see Eq.(6)):

$$\frac{\partial w_i}{\partial t} = \frac{D_{eff}}{r_0^2} \frac{\partial}{\partial r} r^2 \frac{\partial w_i}{\partial r} \tag{6}$$

here, D_{eff} is an effective diffusivity of adsorbate within a particle $[m^2/s]$, r is a radius $[m]$, r_0 is a particle radius $[m]$.

Eq.(6) may be simplified to the following form [15]:

$$\frac{\partial w_i}{\partial t} = 6 w_{eq,i} \Sigma_{n=1}^{\infty} \frac{D_{eff}}{r_0^2} \exp((-n\pi)^2 \frac{D_{eff}}{r_0^2}). \tag{7}$$

The pore diffusion model is also rarely used in numerical simulation because of complexity [15].

The linear driving force is mainly used in practical adsorption simulations on AC [15]. The model determines mass transfer rate through a lumped mass transfer coefficient k:

$$\frac{J_i}{\rho_{bulk}} = \frac{\partial w_i}{\partial t} = k_i \cdot (w_{eq,i} - w_i) \tag{8}$$

here, J_i is an adsorption term $[kmol/kg]$, ρ_{bulk} is a solid bulk density $[kg/m^3]$, k_i is an overall mass transfer coefficient of a component i $[1/s]$.

3.4.3 Mass Balance Equations

The adsorbent bed is divided into small control volumes as shown in Figure 8.

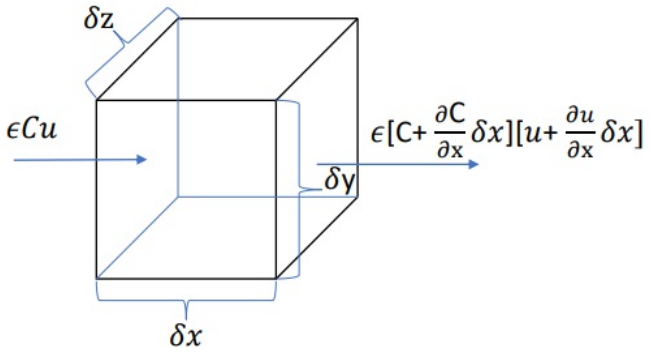


Figure 8: Control volume

The mass balance derivation of a particular component is based on the fact that the molar mass that enters the control volume must be equal to the molar mass that is accumulated and exits the control volume. The used assumptions are:

1. Bed porosity is homogeneous and constant along the bed.
2. Plug flow. Gas flows only in axial direction and there are no derivatives in radial direction.
3. Diffusion in axial direction is negligible.

4. LDF is used for describing solid mass balance.

5. Ideal gas law is used.

Thus, one can write the following equation for a particular component i:

$$-\epsilon u\frac{\partial C_i}{\partial x}\delta x\delta y\delta z-\epsilon C_i\frac{\partial u}{\partial x}\delta x\delta y\delta z=\frac{\partial M_i}{\partial t}\delta x\delta y\delta z \tag{9}$$

here ϵ is the total porosity, C_i is a concentration of component i, x is a spacial coordinate in x direction $[m]$, y is a spacial coordinate in y direction $[m]$, z is a spacial coordinate in z direction $[m]$, M is the total molar mass in the control volume of component i $[kmol/m^3]$.

From the other hand, the term on the right hand side of the Eq.(9) can be expressed as accumulation of molar mass in the control volume plus adsorption term:

$$\frac{\partial M_i}{\partial t}=\epsilon\frac{\partial C_i}{\partial t}+J_i \tag{10}$$

here, J_i is the source term representing adsorption.

Finally, one can get the following equation for a particular component:

$$\epsilon\frac{\partial C_i}{\partial t}+\epsilon u\frac{\partial C_i}{\partial x}+\epsilon C_i\frac{\partial u}{\partial x}+J_i=0. \tag{11}$$

The superficial velocity is defined as the velocity of flow as if the flow of gas was in a hollow pipe:

$$u_{sup}=\frac{Q}{S}=\epsilon u \tag{12}$$

here, Q is a volumetric flow rate $[m^3/s]$, S is a cross sectional area $[m^2]$.

By substituting Eqs.(8) and (12) into Eq.(11), one can get the final equation for describing the total mass balance of a particular component i.

$$\epsilon\frac{\partial C_i}{\partial t}+u_{sup}\frac{\partial C_i}{\partial x}+C_i\frac{\partial u_{sup}}{\partial x}+\rho_{bulk}k_i\cdot(w_{eq,i}-w_i)=0 \tag{13}$$

The same procedure is for deriving the total mass balance for the gas system.

$$\epsilon\frac{\partial C}{\partial t}+u_{sup}\frac{\partial C}{\partial x}+C\frac{\partial u_{sup}}{\partial x}+\rho_{bulk}\sum_{i=1}^{N}k_i\cdot(w_{eq,i}-w_i)=0 \tag{14}$$

Since pressure is assumed to be constant through the bed in the adsorption step, one can get the superficial velocity derivative from Eq.(14), based on the isothermal conditions. The molar concentration according to the ideal gas law:

$$C = \frac{P \cdot 10^2}{RT} \tag{15}$$

here, C is a total concentration of gas $[kmol/m^3]$.

If pressure and temperature do not change, total concentration is not changed along the bed according to the ideal gas law (see Eq.(15)). Hence, $\frac{\partial C}{\partial t} = 0$, and $\frac{\partial C}{\partial x} = 0$. Using Eq.(14) one can get the following formula for velocity derivative:

$$\frac{\partial u_{sup}}{\partial x} = -\frac{1}{C}\rho_{bulk}\sum_{i=1}^{N} k_i \cdot (w_{eq,i} - w_i). \tag{16}$$

In reality, pressure drop occurs in the adsorber, and to specify the boundary conditions for every VPSA step, one should consider implementing the momentum equation. The momentum equation includes pressure drop modeling. The most common models that are used in practice are the Ergun equation (see Eq.(17)), Carman–Kozeny (see Eq.(18)) equation and Darcy law (see Eq.(19)). The simplest packed-bed pressure drop model is the Carman–Kozeny equation. The main assumption in that equation is that pore space behaves like small pipes governed by Poiseuille's flow. Thus, the equation is only valid for laminar flow. The Ergun equation has a squared velocity, making it possible to use in both laminar and turbulent flows. By comparing Eqs. (17) and (18) it is clear that Eq.(17) transforms to Eq.(18) in case of low velocity. The Darcy law also describes laminar flow since pressure drop is linearly dependent on velocity. By comparing Eq.(18) and Eq.(19) one may come to the value of coefficient B (permeability) for the packed bed (see Eq.(20)).

$$\frac{\partial P}{\partial x} = -(1.5 \cdot 10^{-3}\frac{\mu(1-\epsilon_{ext})^2}{(2r_o\phi)^2\epsilon_{ext}^3}u_{sup} + 1.75 \cdot 10^{-3}\frac{P \cdot MW^2(1-\epsilon_{ext})}{RT(2r_o\phi)\epsilon_{ext}^3}u_{sup}^2) \tag{17}$$

here, ϵ_{ext} is an external porosity, μ is a gas dynamic viscosity $[Pas]$, MW - gas molecular weight $[g/mol]$, ϕ adsorbent shape factor, T - temperature, $[K]$.

$$\frac{\partial P}{\partial x} = -1.5 \cdot 10^{-3}\frac{\mu(1-\epsilon_{ext})^2}{(2r_o\phi)^2\epsilon_{ext}^3}u_{sup} \tag{18}$$

$$\frac{Q}{S} = u_{sup} = -\frac{B}{\mu}\frac{\partial P}{\partial x} \tag{19}$$

here, B is a velocity proportionality coefficient.

$$B = \frac{10^5}{150\frac{(1-\epsilon_{ext})^2}{(2r_o\phi)^2\epsilon_{ext}^3}} \tag{20}$$

Combining Eqs. (13, 15, 19) one can get the following form:

$$(\frac{10^2\epsilon}{RT})\frac{\partial(y_iP)}{\partial t} - \frac{10^2}{RT}\frac{B}{\mu}(y_iP\frac{\partial^2P}{\partial x^2} + y_i(\frac{\partial P}{\partial x})^2 + P(\frac{\partial P}{\partial x})(\frac{\partial y_i}{\partial x})) + \rho_{bulk}k_i \cdot (w_{eq,i} - w_i) = 0 \tag{21}$$

here y_i is a molar fraction of a component.

One more equation is needed in order to solve the above equations. The sum of the mole fractions of the components must be equal to 1:

$$\Sigma_{i=1}^{N} y_i = 1 \tag{22}$$

3.4.4 Energy Balance Equations

General assumptions that can be applied:

1. Adiabatic boundary conditions
2. Constant wall temperature
3. More complex boundary conditions

Constant wall temperature assumption will be considered in the master thesis.

Flow assumptions are more or less the same as in the mass balance chapter (constant and homogeneous porosity, no radial flow, axial diffusion is negligible, the LDF is used for solid mass balance, and ideal gas law is used).

Also, assumptions regarding the fluid and solid material properties: thermal conductivity and heat capacity are assumed to be constant and independent of temperature, pressure, and gas composition.

Heat transfer assumptions: the heat of adsorption is assumed to be constant, independent of loading and temperature.

Other general assumptions are negligible thermal conductivity, no viscous dissipation, and negligible influence of pressure work.

To simplify the model, one can assume negligible heat transfer resistance between the gas and solid phases. Thus, one can treat solid and gas phases as a single phase.

The conservation equation in general form:

$$\frac{\partial U_0}{\partial t} + \frac{\partial f(U_0)}{\partial x} = Source \tag{23}$$

here, U_0 are the conservative variables, $f(U_0)$ is the flux of conservative variables.

The conservative variables can be expressed as:

$$U_0 = \epsilon c_{p,g} CT + c_{p,s}\rho_{bulk} T \tag{24}$$

here, $c_{p,g}$ $[J/kmolK]$ and $c_{p,s}$ $[J/kgK]$ are specific heat capacities of gas and solid phases respectively, T is temperature $[K]$.

The flux vector consists of convective and conductive terms. Assuming negligible conduction:

$$f(U_0) = u_{sup} c_{p,g} CT \tag{25}$$

The source term:

$$Source = \frac{h_W}{d}(Tamb - T) + \rho_{bulk} \sum_{i=1}^{N} k_i \cdot (w_{eq,i} - w_i)\Delta H_i \tag{26}$$

here, ΔH_i is an enthalpy change due to the heat of adsorption $[J/kmol]$, hw is a heat transfer coefficient between the adsorbent and environment $[W/m^2K]$, d is a diameter of adsorber $[m]$, $Tamb$ is an ambient temperature (wall temperature) $[K]$.

Thus, taking into account the ideal gas law (Eq.(15)), the following equation is obtained:

$$\begin{gathered}(c_{p,s}\rho_{bulk} + \epsilon c_{p,g}\frac{P \cdot 10^2}{RT})\frac{\partial T}{\partial t} - c_{p,g}\frac{P \cdot 10^2}{RT}\frac{B}{\mu}(\frac{\partial P}{\partial x}\frac{\partial T}{\partial x}) = \\ \Sigma_{i=1}^{N}(\rho_{bulk}\sum_{i=1}^{N} k_i \cdot (w_{eq,i} - w_i)\Delta H_i) + \frac{h_W}{d}(Tamb - T)\end{gathered} \tag{27}$$

3.4.5 Boundary and initial conditions

Table 3: Boundary and initial conditions

Pressurization	Adsorption	Desorption	Purge
$P(x,t=0)=P_{initial}$	$P(x,t=0)=P_{press}$	$P(x,t=0)=P_{ads}$	$P(x,t=0)=P_{\ des}$
$P(x=0,t)=P_1(t)$	$P(x=0,t)=P_{inlet}$	$P(x=0,t)=P_1(t)$	$P(x=0,t)=P_1(t)$
$P(x=L,t)=P_2(t)$	$\frac{\partial^2 P}{\partial x^2}(x=L,t)=0$	$P(x=L,t)=P_2(t)$	$P(x=L,t)=P_2(t)$
$y_i(x=0,t)=y_{i,feed}$	$y_i(x=0,t)=y_{i,feed}$	$\frac{\partial y}{\partial x}(x=L,t)=0$	$y_{N_2}(x=L,t)=1$
$T(x,t=0)=T_{initial}$	$T(x,t=0)=T_{press}$	$T(x,t=0)=T_{ads}$	$T(x,t=0)=T_{des}$
$T(x=0,t)=T_{inlet}$	$T(x=0,t)=T_{inlet}$	$\frac{\partial T}{\partial x}(x=L,t)=0$	$T(x=L,t)=T_{inlet}$

here, $P_1(t)$ is pressure at the bottom (inlet) of the adsorber $[bar]$, $P_2(t)$ is pressure at the top (outlet) of the adsorber $[bar]$, $press, ads, des$ subscripts mean the results of parameter from a particular step.

The pressures $P_1(t)$ and $P_2(t)$ are calculated based on the ideal gas law (see Eq.(15)) using the following equations:

1. Pressurization

$$\frac{\partial P_1}{\partial t} = \frac{P_1}{V_1}((\frac{FRT \cdot 10^{-2}}{P_1}) - u_{out}A) \tag{28}$$

$$\frac{\partial P_2}{\partial t} = \frac{P_2}{V_2}(u_{in}A) \tag{29}$$

2. Desorption

$$\frac{\partial P_1}{\partial t} = \frac{P_1}{V_1}(-q_{pump} + u_{in}A) \tag{30}$$

$$\frac{\partial P_2}{\partial t} = \frac{P_2}{V_2}(u_{out}A) \tag{31}$$

3. Purge

$$\frac{\partial P_1}{\partial t} = \frac{P_1}{V_1}(-q_{pump} + u_{in}A) \tag{32}$$

$$\frac{\partial P_2}{\partial t} = \frac{P_2}{V_2}(q_{purge} - u_{out}A) \tag{33}$$

here, F is the flow rate coming into the adsorber $[kmol/s]$, V_1 is an inlet volume not filled with adsorbent $[m^3]$, V_2 is an outlet volume not filled with adsorbent $[m^3]$, u_{out} is a velocity of gas going to the adsorber $[m/s]$, u_{in} is a velocity of gas going out of the adsorber $[m/s]$, q_{pump} is a volumetric flow rate to the vacuum pump $[m^3/s]$, q_{purge} is a purge volumetruc flow rate of purge gas to the adsorber. It should be noted that velocity has a negative sign during desorption and purge steps.

3.5 Developed models summary

Depending on the developed simplifications, models of varying complexity degrees can be distinguished. However, the difference in the results of these models is not significant for certain conditions. The models used in the work are summarized in the table below.

Table 4: The developed models

Model	Main equations used	Main assumptions	Application
1	13, 16	No pressure drop along the adsorber, isothermal simulation.	Section 6.2: Adsorption step in Cavenati and co-authors experiments, comparison with model 2 and Aspen Adsorption simulations.
2	21	Isothermal simulation.	Section 6.2: Comparison with VPSA Cavenati and co-authors experiments, comparison with model 1 and Aspen Adsorption simulations. Sections 6.3, 6.4,6.5: VOCs simulation.
3	21, 27	-	Section 6.2: comparison with Cavenati et al. [5] experiments

Models are solved with appropriate boundary and initial conditions (see Table 3). The mass transfer coefficients and equilibrium parameters for Cavenati and co-authors experiments in section 6.2 are considered from Vilardi et al. work [17]. The mass transfer coefficients for VOCs simulation are assumed to be equal to 1 s^{-1}. Such high mass transfer coefficients correspond to the local equilibrium model (see Eq.(4)), making it easier at the beginning of the simulation. Equilibrium data for VOCs equilibrium simulation are assumed from [14].

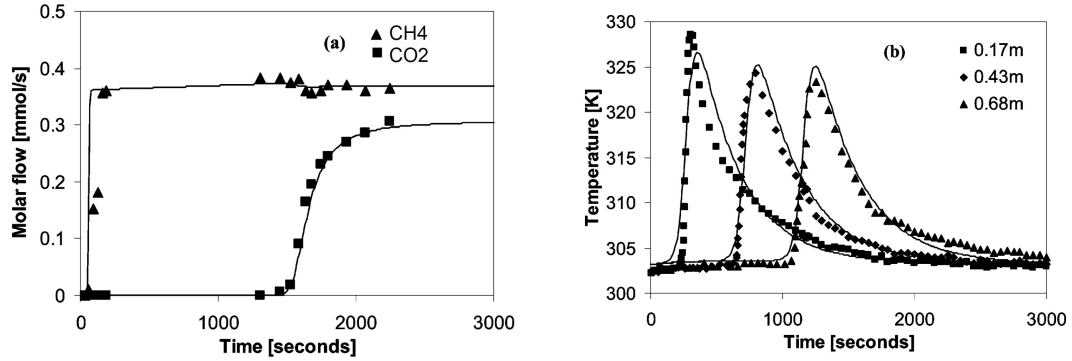
4 Experiments

4.1 Cavenati and co-authors Experiments

The study of Cavenati et al. [5] focused on VPSA for the separation of methane (55 mol %) - carbon dioxide (45 mol %) mixture with a total flow between 1 and 1.5 SLPM (Standard Liter per Minute). A four-step Skarstrom-type cycle was used consisting of co-current pressurization, adsorption, counter-current blowdown, and counter-current purge. The results for binary methane-carbon dioxide adsorption at a constant pressure of 320 kPa are presented. The authors did experiments on the column of 0.83 m in length and 0.021 m in diameter with a bulk density of 715 kg/m^3. The ambient temperature during the experiments was 303 K.

The linear driving force model was used for modeling the mass transfer, and the Langmuir equation was used to describe the equilibrium model. Equilibrium and mass transfer parameters for methane-carbon dioxide mixture adsorption on Carbon Molecular Sieve 3K can be found in Vilardi et al. work [17]. It should be noted that these equilbrium parameters are different from those used in VOCs Equinor experiments simulation because of different AC type.

The results of the binary breakthrough curve for the methane - carbon dioxide mixture during the adsorption step are shown in Figure 9. In the figures, the solid line corresponds to the mathematical model results of Cavenati et al. [5] while the points represent the obtained experimental data.



(a) Molar flow rate of each gas exiting the column (b) Temperature profiles inside the column

Figure 9: Cavenati and co-authors experimental results for adsorption step

The VPSA process results after 46 cycles are presented in Figure 10. These results will also be used in the master thesis to validate the model because it contains only two main components: carbon dioxide and methane, making it easier to compare and check if the

model simulates correctly.

During the first 80 seconds, the pressure rises from 10 kPa to 320 kPa with the constant flow rate of gas supplied. The outlet flow rate of methane and carbon dioxide is 0 during this step because the top of the column is closed. During the next 100 seconds, the top of the column is opened, and the adsorption step with constant pressure and gas inlet flow rate begins. Only methane is obtained as the product at this step. From 180 to 300 seconds, the top of the column is closed, and counter-current blowdown occurs with the vacuum pump. Cavenati and co-authors [5] did not describe in detail the operation of the vacuum pump. The characteristic of the MZ 1C vacuum pump [22] was used in the developed Python model. In the next 50 seconds, there is a counter-current purge to lower the partial pressure of carbon dioxide.

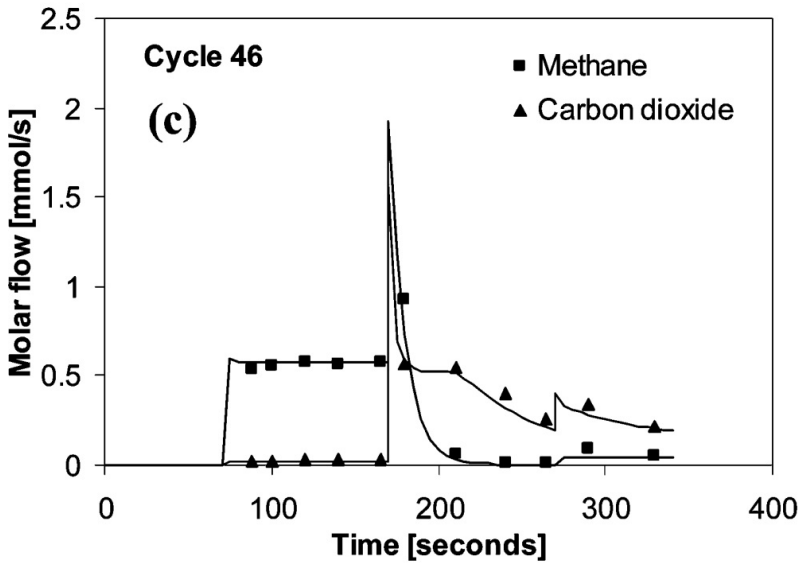


Figure 10: VPSA experiment

4.2 Equinor Experiments

The central part of the master thesis deals with VOCs adsorption simulation. The experiments were performed at Equinor to optimize the adsorption of VOCs on activated carbon adsorption bed [6]. However, a mathematical model should be created to analyze all the possibilities for optimization. The adsorption bed parameters are given in Table 5.

Table 5: Bed parameters

Adsorber height	1	m
Adsorber filled AC height	0.77	m
Adsorber diameter	0.032	m
Filled AC volume	650	ml
Clean adsorbent bulk density	430	kg/m^3

The components used in the experiments are methane, ethane, propane, butane, carbon dioxide and nitrogen. Also, experiments with C5+ components were performed. However, in this study only VOCs up to C4 were considered. The feed gas components were properly mixed and heated to 303 K so that no liquid was accumulated [6]. The temperature in the center of the adsorber was measured by multi-thermocouples and the temperature variations were not higher than 10 degrees for C1-C4 adsorption.

The vacuum pump used during the regeneration cycle was an oil-free membrane type (VacuuBrand MZ1C) [22]. The pump characteristics are also available [22] and it was able to operate as low as 11 mbar without flow through the system. The adsorber was pre-treated for 1 week under vacuum before the experiments started. Thus, a vacuum pump plays an essential part in modeling the desorption process.

As the inlet pressure decreases, the pump's volume flow rate decreases. The volume flow rate at low pressures can be less than indicated in the real pump curve since air leaks can occur. The manufacturer has also included a dotted line that indicates the performance of a different pump subtype that was not used in the experiments.

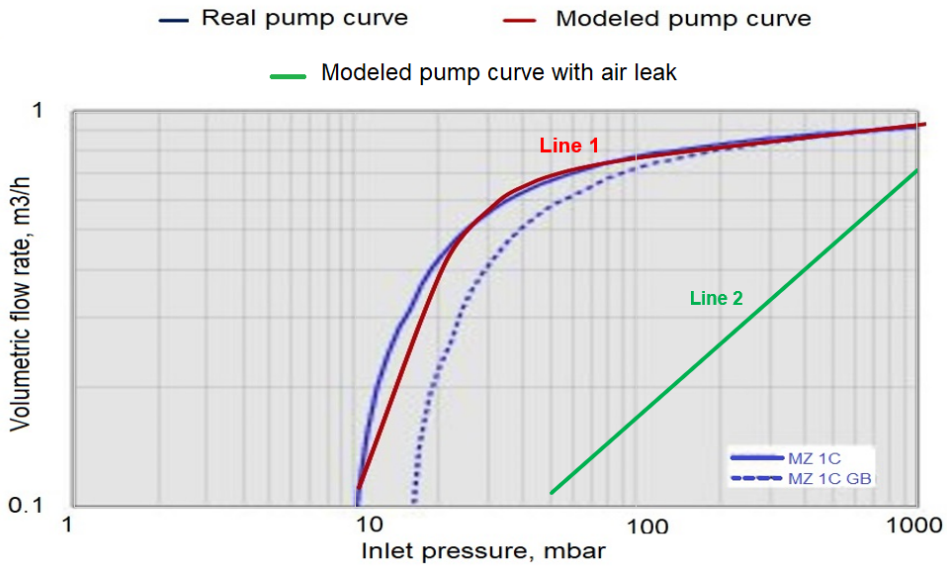


Figure 11: Pump 1 MZ C characteristics curve

The choice of pump characteristic greatly affects the desorption pressure. A simplified pump model that considers the air leak into the system is determined by the formula:

$$q_{pump} = 0.00019 \cdot P^{0.62} \tag{34}$$

The experimental scheme consists of two beds that run continuously, with one bed in the adsorption mode and one bed in the desorption mode (the principal scheme is shown in Figure 4). Upstream the adsorbers, there was a tank of 1-liter volume where constant inlet volumetric flow was supplied. This tank was connected to the adsorber via a 2 m tubing with a diameter of 2 mm. The sequence of the steps is summarized in Table 6. When the pressure reaches at least one bara in the adsorber, the outlet valve opens. Depending on the feed flow rate, it took from 1.5 to 3.5 min to build up the pressure after vacuuming. The pressure in the adsorption mode was between 1.05 and 1.10 bara, depending on the feed rate and time in the cycle. The adsorption step for the considered cases lasts 15 min, after which the counter-current regeneration for 10 min is performed. During the purge, argon was supplied from the top (outlet) of the adsorbers (either through the total period (desorption and purge steps) or only towards the end of the period). In the master thesis, nitrogen is used as a purge gas during 5 min., and this should not create a big difference because nitrogen and argon barely adsorb. The pressure towards the end of the vacuuming period was 60 ± 10 mbar. The flow rate of argon used was 25, 50, and 100 Nml/min, as seen in Table 7.

Table 6: Cycle steps

Step	Time
Pressurization	Until 1 bar is reached
Adsorption	15 min
Desorption	10 min
Purge	5 min

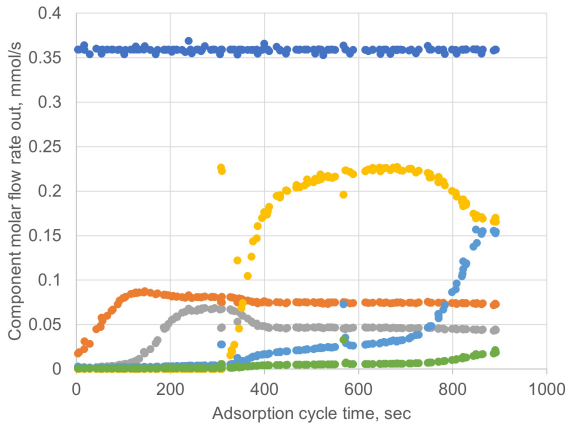
The experiment results for each case are shown in Figure 12. The results are also presented in terms of tables in Appendixes B,C,D.

The main conclusion from these graphs is that adsorption happens differently for every component, and almost 100% adsorption efficiency could be achieved for C4+ components. The adsorption efficiency also highly depends on the inlet flow rate of heavy hydrocarbons. It is important to note that the breakthrough curves are rather steep, which indicates high mass transfer coefficients.

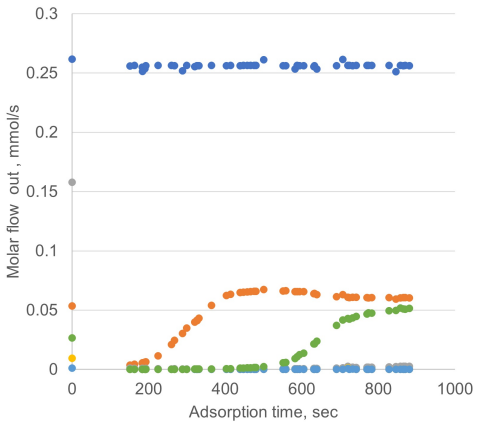
The following three cases used in the experiments were considered in the master thesis:

Table 7: Parameters of the main three cases

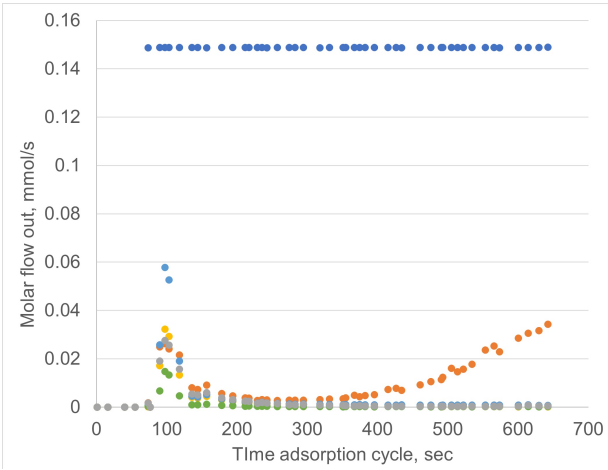
Case number	Composition of feed gas $[mol/mol]$						Flow rate of feed gas $[Nml/min]$	Purge flow rate $[Nml/min]$
	CH_4	C_2H_6	C_3H_8	C_4H_{10}	CO_2	N_2		
1	8.6	16.1	22.4	10	4.7	38.2	1248	50
2	8.3	16.6	20.8	9.6	4.0	40.7	851	100
3	11.7	8.7	15.1	5.8	8.7	50	440	25



(a) Cycle 1



(b) Case 2



(c) Case 3

Figure 12: Equinor experiments

5 Methods

5.1 Python model

The set of partial differential equations (PDE) and ordinary differential equations (ODE) need to be solved with appropriate initial and boundary conditions. The resulting system of PDE and ODE must be solved simultaneously since they are coupled. The spatial and time derivative terms can be approximated. The variables are defined on the grid with finite difference methods.

The common challenge is the propagation of fronts or shocks. The first-order spatial derivative approximations may cause two unwanted problems: numerical diffusion (artificial diffusion) and numerical oscillations. Numerical diffusion can be reduced by decreasing the space step that will be shown in practice in the results of the master thesis.

The first-order upwind difference scheme may be beneficial in the case of sharp front propagation, which is vital in the simulation beginning and in the systems where breakthrough curves are steep. The first-order upwind method is also recommended because of the fast simulation. The scheme is first-order accurate and may give a significant numerical diffusion. However, the method does not produce oscillations (unconditionally stable). The upwind difference scheme for the positive velocity is shown below:

$$\frac{\partial f}{\partial x} = \frac{f_l - f_{l-1}}{\Delta x} \tag{35}$$

here, f is a function, l is a spacing point, Δx is a space step.

The upwind difference scheme for the negative velocity:

$$\frac{\partial f}{\partial x} = \frac{f_{l+1} - f_l}{\Delta x} \tag{36}$$

Thus, the information comes from the direction of the flow.

The second order derivative approximation is shown below:

$$\frac{\partial^2 f}{\partial x^2} = \frac{f_{l+1} - 2f_l + f_{l-1}}{\Delta x^2} \tag{37}$$

The system of PDE and ODE will be solved in Python. Python combines both simplicity and powerful tools. It can be used to create a prototype of almost any program. Python also has extensive open libraries that can help solve the equations described earlier. For

example, GEKKO Python is an object-oriented Python library that is used in the master thesis. It is free for academic and commercial use. It is designed for large-scale engineering optimization problems, including problems with non-linear algebraic and differential equations.

Firstly, GEKKO model creation must be introduced. GEKKO can be easily installed, and the GEKKO model can be created with the following command:

```
!pip install GEKKO
from gekko import GEKKO
m = GEKKO(remote = False)  # create GEKKO model
```

The model formulation is essential for a reliable and fast simulation. First and foremost, it is required to put the equations into the form for the most accurate solution. The model formulation in this project includes the introduction of variables. They are temperature, pressure, velocity, and others that are determined by the set of equations. The variables are introduced in the following way:

```
P = [m.Var(pressure_init) for i in range(seg)]  # Introduction of
#pressure variable. "seg" - number of grid  points. "pressure_init"
#is set for every grid point as an initial condition.
P1 = m.Var(pressure_init) # Boundary left pressure
P2 = m.Var(pressure_init) # Boundary right pressure

y1 =  [m.Var(0) for i in range(seg)] # Molar fraction of component 1
# is 0 at every grid step.
y2 =  [m.Var(0) for i in range(seg)] # Molar fraction of component 2
# is 0 at every grid step.
```

There is always a trade-off between computation accuracy and time. A denser time grid will enhance the accuracy of the simulation; however, the calculation will take longer. Usually, a denser time grid is required in the regions of fast dynamics. The user sets the time grid, and GEKKO solves the set of equations within the specified time grid. The time discretization is defined in GEKKO as follows:

```
nt = int(final_time/time_step) + 1 # number of time grid points

m.time = np.linspace(0,tf,nt) # Crease a GEKKO time grid
```

The set of the equations is discretized to an algebraic form with a finite difference method (see Eqs. (35), (36),(37)) such that the model consists of a system of ordinary differential equations. Equations are expressed as equality constraints. For example, the mass balance equation for component 1 during the pressurization step is described as follows:

```
# Here, P - pressure [bar], P2 - boundary pressure at the top,
#P1 - boundary pressure at the bottom,y1 - molar fraction of component 1
#K - velocity proportionality , et - total porosity,
#y1_feed - molar feed of component 1 to the adsorber
# L_seg - the length of one segment:
#L_seg = Length of adsorber / number of grid points
# R - universal gas constant [J/molK], T - temperature [K],
#rho - density [kg/m3], q1 - adsorption term [kmol/kg].

# First segments Pressure 1
m.Equation(P[0]*y1[0].dt()+y1[0]*P[0].dt() == (1/(K*et))*((y1[0]*P[0]*
(P[1]-2*P[0]+P1)/(L_seg**2))
+ P[0]*((P[0]-P1)/(L_seg))*((y1[0]-y1_feed)/
(L_seg))+ y1[0]*((P[0]-P1)/(L_seg))**2)
- ((1/(et*100))*(R*T)*rho*q1[0].dt()))

# Middle segments component 1.
m.Equations([P[i]*y1[i].dt()+y1[i]*P[i].dt() ==  \
(1/(K*et))*((y1[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
+ P[i]*((P[i]-P[i-1])/(L_seg))*((y1[i]-y1[i-1])/(L_seg))
+ y1[i]*((P[i]-P[i-1])/(L_seg))**2)
- ((1/(et*100))*(R*T)*rho*q1[i].dt())   for i in range(1,seg-1)])

#Last segment component 1
m.Equation(P[seg-1]*y1[seg-1].dt()+y1[seg-1]*P[seg-1].dt() == \
(1/(K*et))*((y1[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
+ P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))*((y1[seg-1]-y1[seg-2])/(L_seg))
+ y1[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
- ((1/(et*100))*(R*T)*rho*q1[seg-1].dt()))
```

From the code above, one may see that it is convenient to set the equation into three main parts: first segment, middle segments, and final segment to include the boundary variables. The time derivative is expressed as follows: $parameter.dt()$. If one needs a second-order time derivative, this can be done by introducing an extra variable equal to

the first-order time derivative.

Before the simulation starts, the user should specify the mode of simulation. They are shown in Table 8.

Table 8: Possible modes of simulation [23]

	Simulation	Estimation	Control
Non-Dynamic	1 Steady-State	2 Steady-State	3 Steady-State
Dynamic Simultaneous	4 Simultaneous	5 Simultaneous	6 Simultaneous
Dynamic Sequential	7 Sequential	8 Sequential	9 Sequential

The most interesting for the master thesis are the following modes of dynamic simulation: 4 Simultaneous (SIM) and 7 Sequential (SQS). Both produce the same solution but in a different way. The sequential mode solves from one-time step to the next until it reaches the solution, while the simultaneous method solves all time steps together. The example of the code to start the solution procedure is shown below:

```
m.options.IMODE = 7
m.solve(disp = False) # disp = False means that the solution report
#will not be shown
```

The complete code for VOCs simulation is presented in Appendix E.

5.2 Aspen Adsorption model

Also, Aspen Adsorption will be used to compare the results. Aspen Adsorption is a comprehensive flowsheet simulator for adsorption modeling. The simple flowsheet with one adsorber for adsorption step is presented in Figure 13.

The user needs to make sure that the system is specified correctly. The feed stream needs the following fixed parameters: flow rate, composition, temperature, and pressure. The product pressure specification is free. Aspen will calculate this pressure by itself. The bed parameters are general partial differential equation handling, material/momentum balance, kinetic model, isotherm, energy balance, reaction, and procedures. Once defined, the flowsheet is checked and initialized.

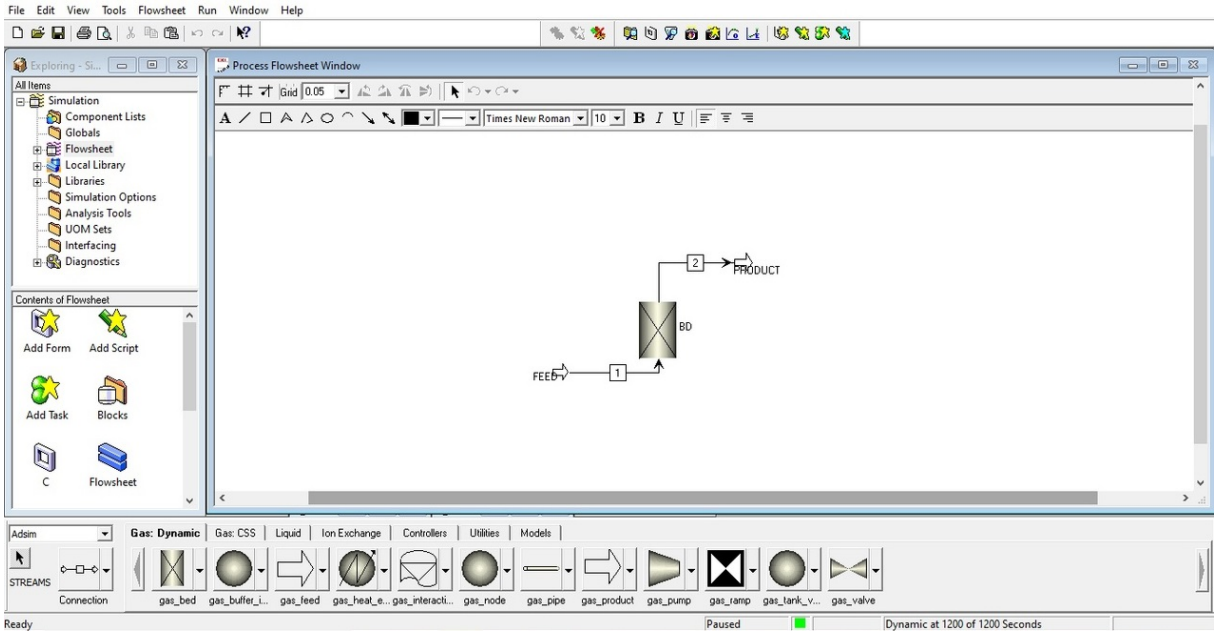


Figure 13: Aspen Adsorption flowsheet

The closest developed model to the conducted VOCs experiments in Aspen Adsorption is shown in Figure 14. In contrast to the previous scheme, the product pressure specification must be fixed. The flow rates of the streams are set with the valves in Aspen Adsorption simulation, and the feed streams' flow rates are calculated based on the valves specifications. The characteristics of the vacuum pump are set via a user-specified submodel.

The cycle organizer switches the valves according to the defined scheme. The cycle options enable to specify the number of cycles, and cyclic-steady-state test, which is a convenient tool in VPSA process. The results are presented in terms of graphs and result values.

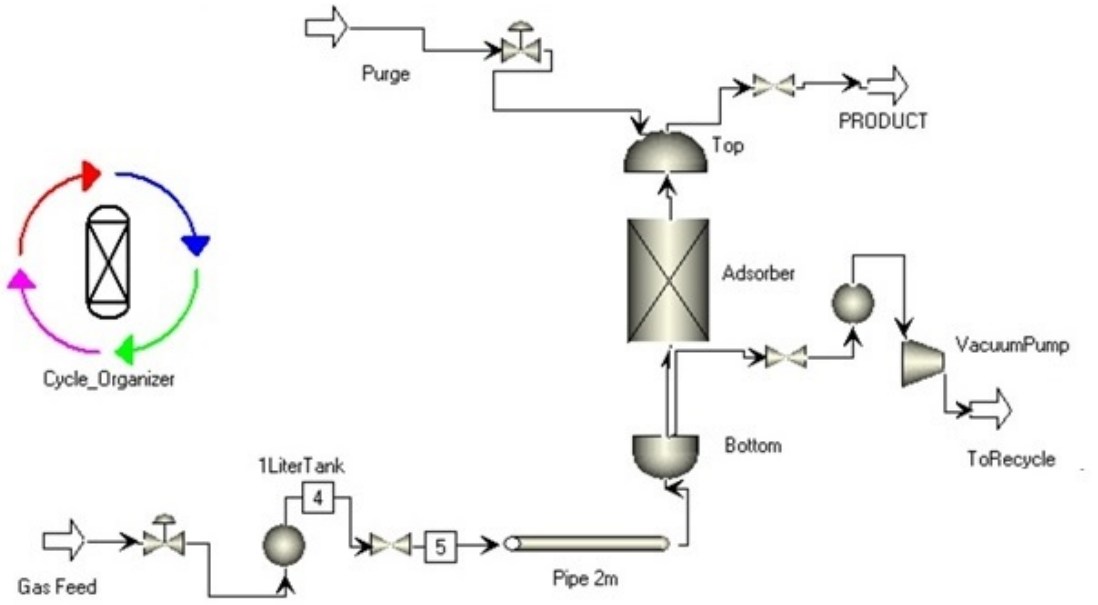


Figure 14: Aspen Adsorption VPSA flowsheet

It is challenging to make the same scheme in Python, including a pipe with a diameter of 2 mm and a length of 2 m since the sonic flow will be obtained at the pipe outlet during pressurization. Therefore, it is worth considering a simplified scheme shown in Figure 15 where a constant amount of gas is fed directly into the adsorber.

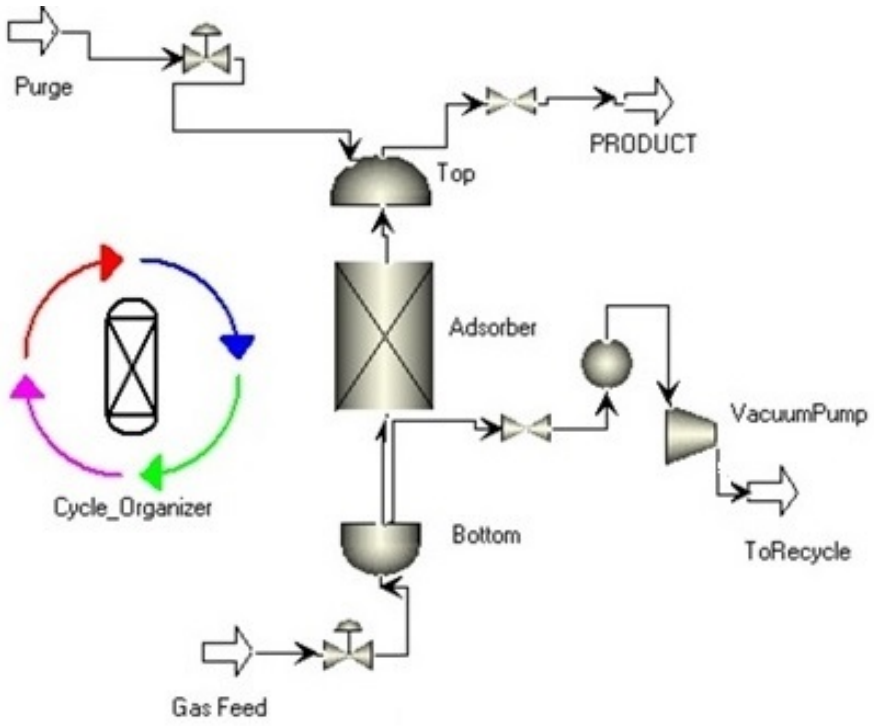


Figure 15: Simplified Aspen Adsorption VPSA flowsheet

6 Results and Discussion

6.1 Multicomponent Equilibrium of Volatile Organic Components on Activated Carbon

In this subsection, the equilibrium model will be covered. Usually, the adsorption equilibrium data of the gas mixture is not available. Pure equilibrium data is the main ingredient of understanding how much can be adsorbed in the process. The data for the prescribed set of components is considered from [14].

The linear form of the Langmuir isotherm:

$$\frac{1}{w_i} = \frac{1}{w_{max,i}} + \frac{1}{b_i}\frac{1}{P_i}\frac{1}{w_{max,i}}. \tag{38}$$

Thus, one can plot the experimental data with $\frac{1}{w_i}$ on y axis and $\frac{1}{P_i}$ on x axis (see Figure 16 for methane). The line in the form $y = kx + a$ can be constructed with certain accuracy. One can notice that a in this case is equal to $\frac{1}{w_{max,i}}$ and k is equal to $\frac{1}{w_{max,i}}\ \frac{1}{b}$. From these expressions $w_{max,i}$ and b in Eq.(1) (Langmuir isotherm) can be estimated.

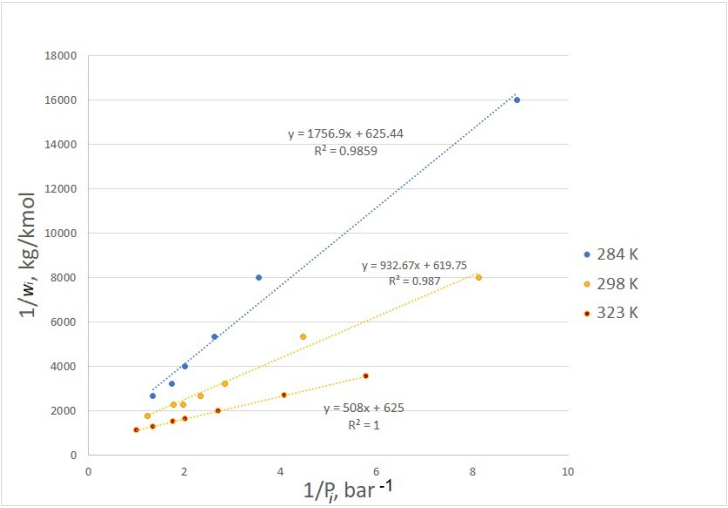


Figure 16: Linear form of the Langmuir isotherm for methane

The heat of adsorption may be derived based on the two lines from Figure 16. The following formula, which is based on Eq.(3), may be used:

$$Q_i = \frac{R}{\frac{1}{T_2} - \frac{1}{T_1}} ln(\frac{b_{2,i}}{b_{1,i}}). \tag{39}$$

Finally, constant $b_{\infty,i}$ can be found.

$$b_{\infty,i} = \frac{b_i}{\exp \frac{Q_i}{RT}} \tag{40}$$

The result of isotherm fitting for methane is shown in Figure 17.

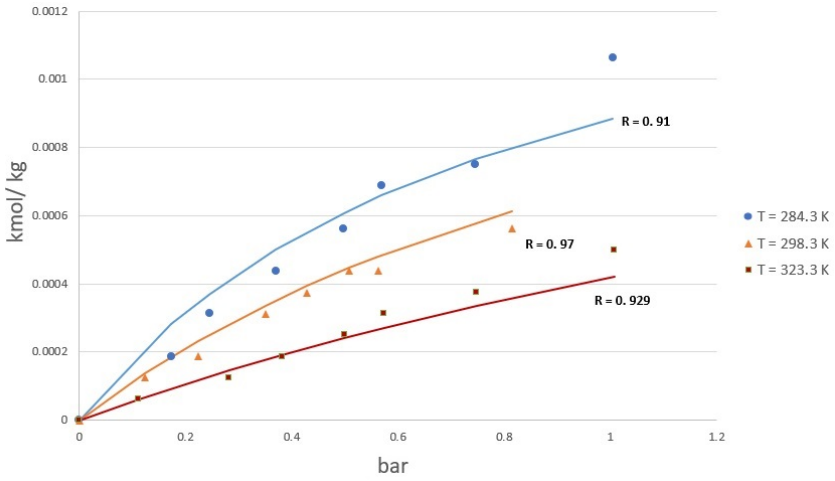


Figure 17: Langmuir isotherm for methane on AC

The figures for isotherm fitting for ethane, propane, butane, nitrogen and carbon dioxide are shown in Appendix A. The following isotherm parameters are obtained:

Table 9: The main parameters of isotherms at temperature 303 K

Methane			Ethane		
w_{max}	0.0016	$kmol/kg$	w_{max}	0.0027	$kmol/kg$
b	0.65	bar^{-1}	b	3.06	bar^{-1}
Propane			Butane		
w_{max}	0.0062	$kmol/kg$	w_{max}	0.0076	$kmol/kg$
b	3.8	bar^{-1}	b	8	bar^{-1}
Carbon Dioxide			Nitrogen		
w_{max}	0.0028	$kmol/kg$	w_{max}	0.0075	$kmol/kg$
b	0.73	bar^{-1}	b	0.02	bar^{-1}

Measuring the mixture adsorption equilibria is a tedious process, and it is common practice to build the isotherm model from available pure component adsorption data. The isotherm parameters from Table 9 can now be used in Extended Langmuir equation for multicomponent equilibrium description (see Eq.(2)).

6.2 Comparison with Cavenati's experiments

The studies involving (Computational Fluid Dynamics) CFD, in particular, need to be proved that results can be trusted. First, one should check the numerical implementation: how numerical aspects such as time step, grid generation, and others affect the accuracy of the solution. The validation means that the results are coherent with the basic theory understanding. One can compare the results with experiment cases received independently. Another option is to compare results obtained from another trustful commercial software such as Aspen Adsorption, widely used in the literature [17, 24]. In the master thesis, both methods are used.

The simulation becomes more complicated with an increase in the number of components. Therefore, it is better first to compare the results with fewer components to prove that simulation works correctly. For example, Cavenati and co-authors [5] investigated only two components' adsorption on activated carbon molecular sieve 3K: methane and carbon dioxide.

Three models were built to simulate the Cavenati and co-authors experiments [5] described earlier:

1. Aspen Adsorption model.
2. Python model without pressure drop calculation across the adsorber (model 1 in Table 4).
3. Python model with pressure drop calculation across the adsorber (model 2 in Table 4).

Figure 18 compares the results of two developed Python models with the Aspen Adsorption (commercial program) results. The first Python model calculates the adsorber's gas flow rate based on the total mass balance assuming negligible pressure drop. The Python model with pressure drop uses Carman-Kozeny equation (see Eqs. (18) and (20)) to calculate velocity dependence on the pressure. It can be seen from Figure 18 that the two developed models in Python give a similar result in comparison with Aspen Adsorption model for the isothermal conditions. That is why it can be argued that the calculation is carried out correctly with more confidence.

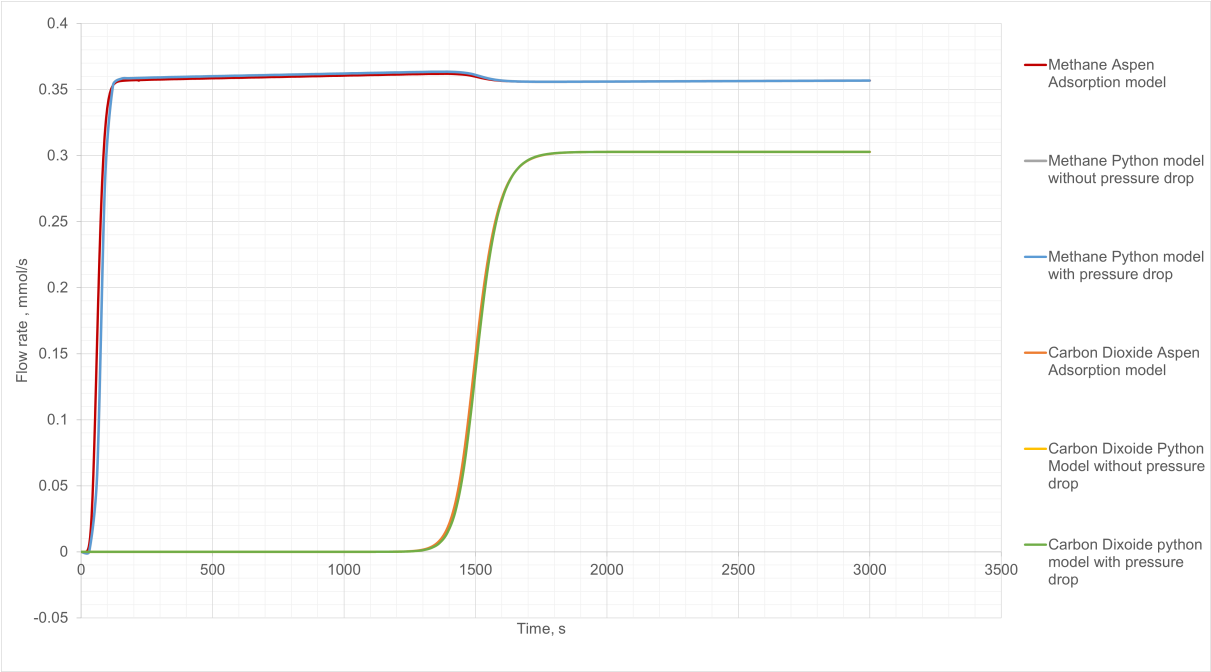


Figure 18: Comparison of Aspen Adsorption model and developed Python models 1 and 2 (see Table 4) for Cavenati and co-authors conditions [5]

The calculation time decreases significantly as the grid size decreases. However, this dramatically affects accuracy and leads to so-called artificial diffusion in both Python models, which can be seen in Figures 19 and 20. A much denser grid is needed for the model with pressure drop calculation to reach the desired accuracy.

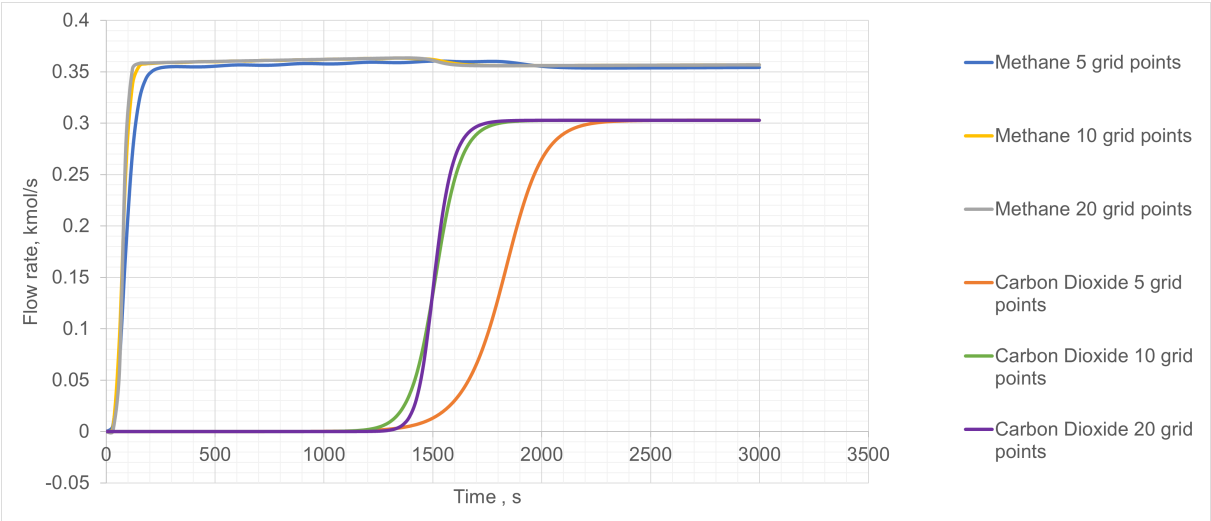


Figure 19: Comparison of results with different grid size for the Python model without pressure drop (model 1 (see Table 4))

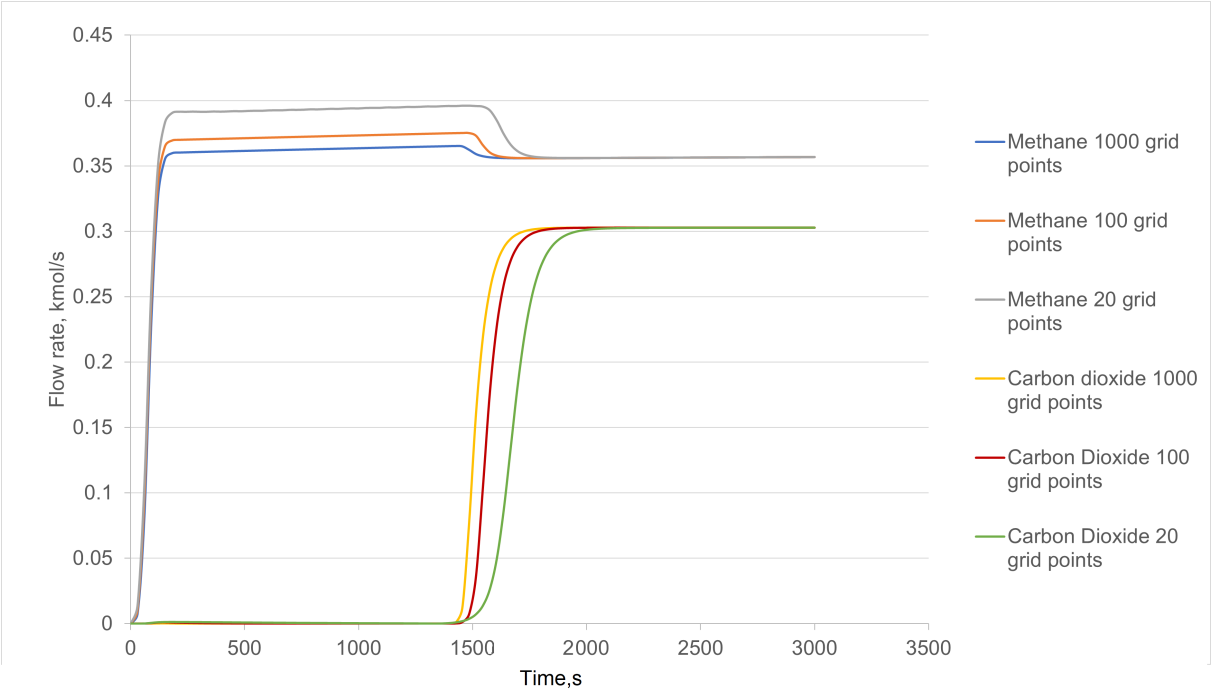


Figure 20: Comparison of results with different grid size for the Python model with pressure drop (model 2 (see Table 4))

The accuracy of the pressure drop calculations can also be seen in Figure 21 which shows the comparison of the adsorber top pressure results from the developed Python model and Aspen Adsorption. The results are approximately the same (note the scale of y-axis in Figure 21).

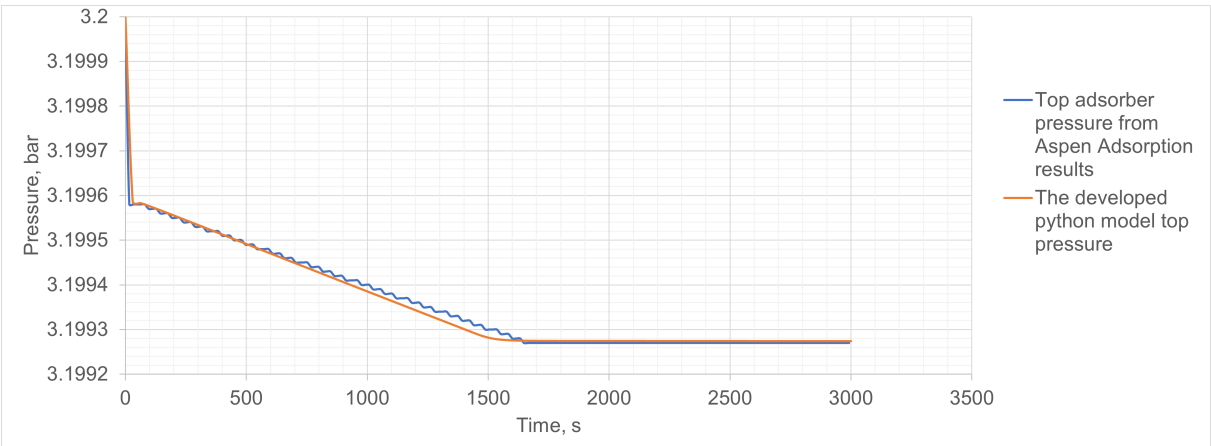


Figure 21: Comparison of results for adsorber bottom pressure (Python model 2 (see Table 4))

An essential part of the simulation is energy balance. Carbon dioxide produces a significant amount of heat during adsorption. Cavenati et al. also reported a substantial amount of heat loss to the environment that needs to be considered. Therefore, approximately

a wall heat transfer coefficient of 4E-5 MW/m^2K was used as a fitting parameter, and the result is shown in Figure 22. The experimental data reported by Cavenati et al. [5] includes temperature profiles at three positions of the adsorption bed (0.17, 0.43 and 0.68 m). These temperature profiles were compared with simulated ones.

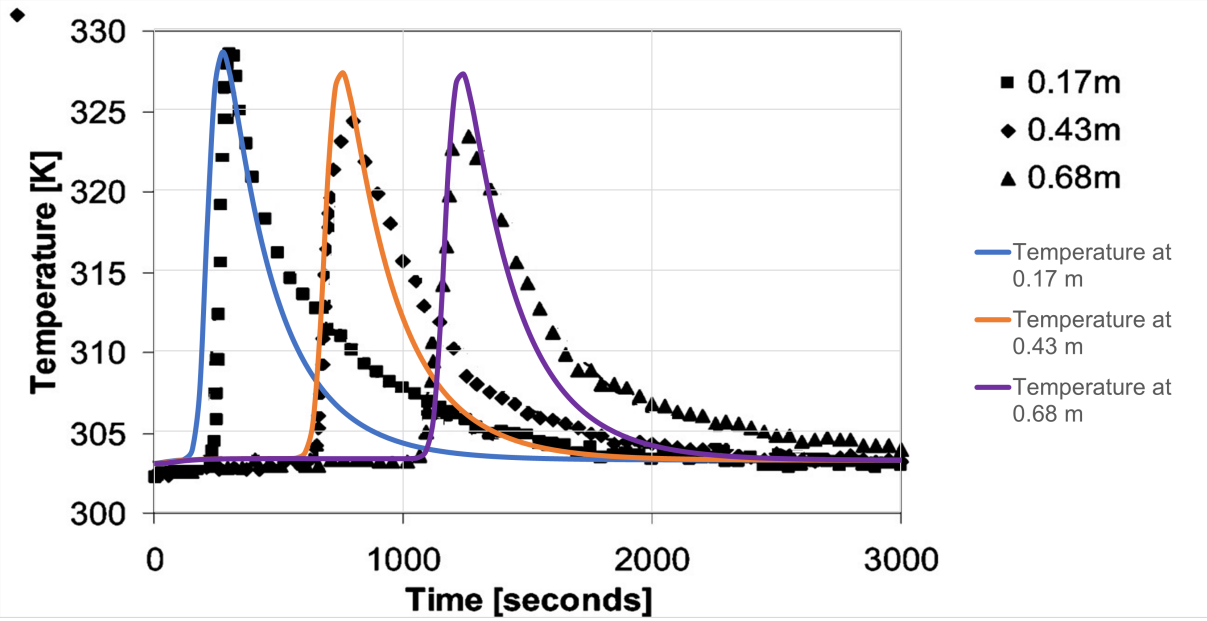


Figure 22: Comparison of temperature results with experiments (experiment results from [5]). Python model 3 (see Table 4)

The results reflect the thermal wave well, but cooling occurs more slowly in experiments. That may be due to the inaccuracy of the component properties (such as heat capacity of gas and adsorbent). Also, neglecting thermal bed conductivity and inaccuracy in heat transfer coefficient may affect.

Temperature variations presented in Figure 22 cannot be ignored and isothermal results represented in Figure 18 would produce errors. Also, the velocity of thermal wave indicates the velocity of carbon dioxide concentration wave.

Figure 23 shows the simulated breakthrough curves and experiment results for non-isothermal conditions. The model describes well the trend of breakthrough for methane and carbon dioxide.

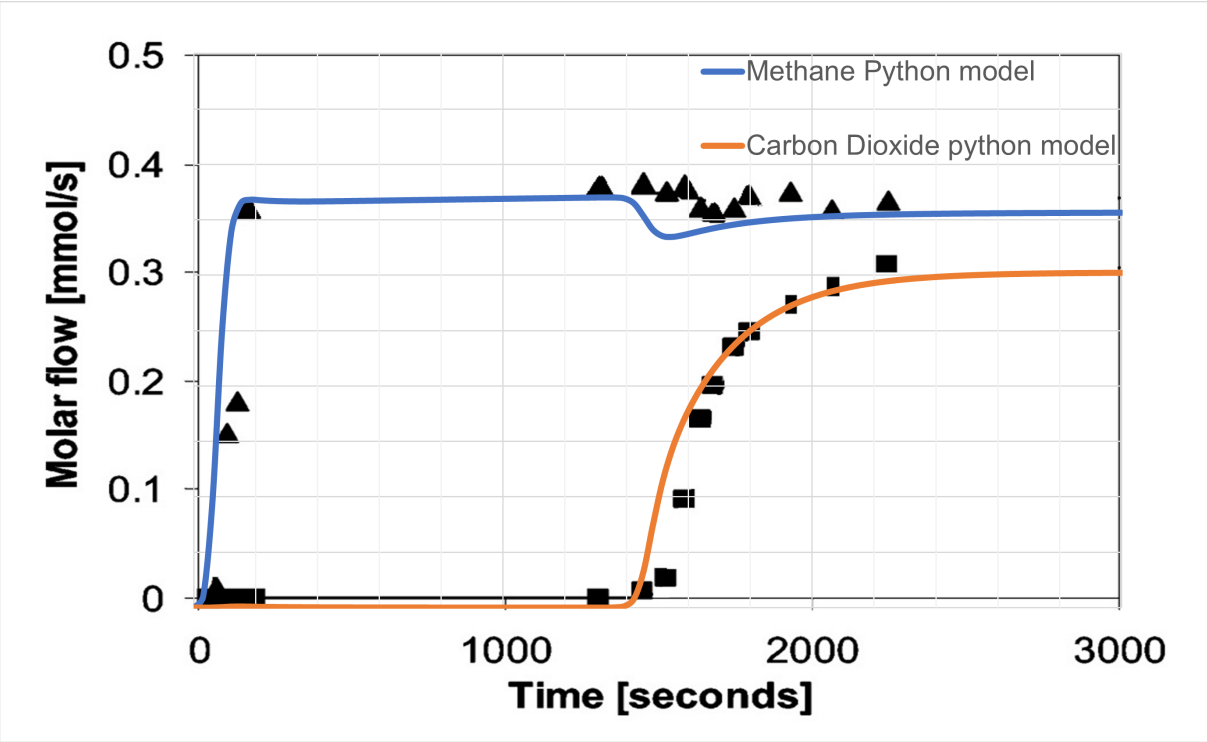


Figure 23: Comparison of breakthrough results with experiments (experiment results from [5]). Python model 3 (see Table 4)

Thus, the developed Python model shows a very good agreement with Cavenati and co-authors [5] experiments on adsorption step. The results obtained from Aspen Adsorption is also similar to the results of Python simulation and experiments.

Next, the simulation after 46 cycles is compared (see Figure 24). The flow rates of the two components, methane, and carbon dioxide, match the experiments well. However, there is a slight discrepancy in methane flow rates during the adsorber regeneration. The final desorption pressure of 10 kPa also coincides with what the authors presented in the article.

Therefore, the model well describes the VPSA Cavenati and co-authors experiments [5]. The geometrical characteristics of the adsorption column are similar to the column used in VOCs Equinor experiments. However, the number of components is 6 instead of 2, complicating the task. The results of experiment comparison with full range VOCs adsorption are discussed below.

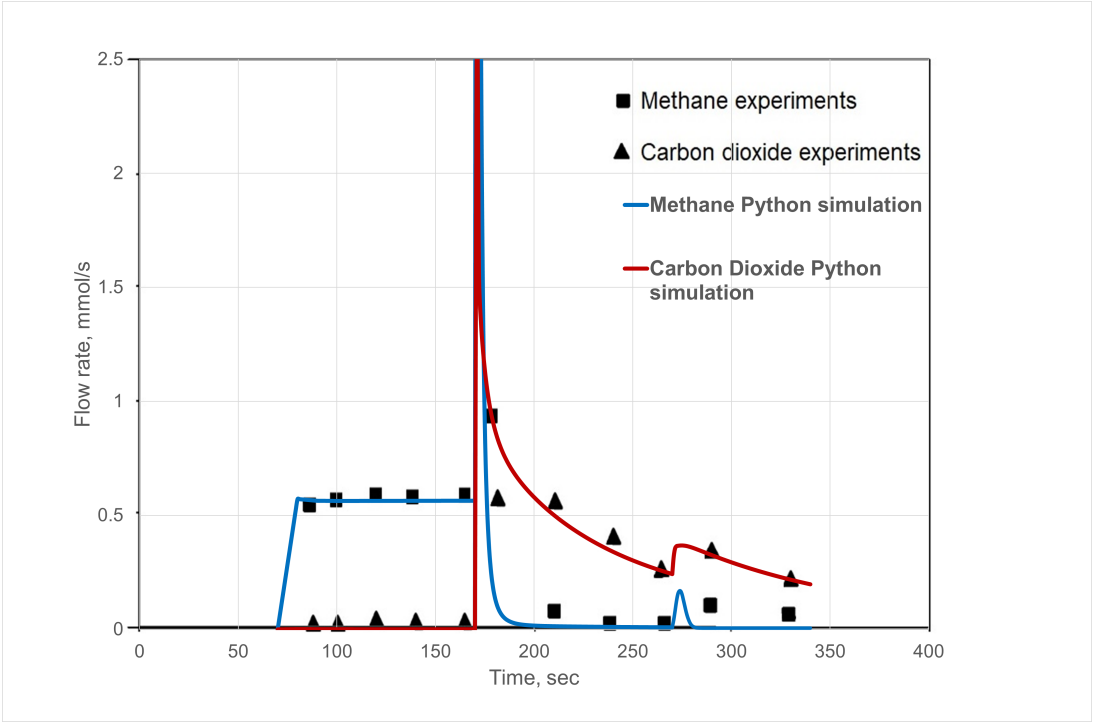


Figure 24: Python simulation VPSA results comparison with experiments from [5] for 46 cycles. Python model 3 (see Table 4)

6.3 Modeling of vacuum pressure swing adsorption for volatile organic components separation

The simulation is carried out for three cases from the experiments summarized in Table 7. Firstly, the solution to case 1 is described. Since temperature variations do not exceed 10 degrees in the experiments, neglecting temperature calculations should not radically change the solution. Thus, the isothermal results (Python model 2 (see Table 4)) are presented below. The main parameters that are fitting to the model are the mass transfer coefficients. Initially, high mass transfer coefficients are assumed (equal to 1 s^{-1}), which means that there is almost no mass transfer resistance. External porosity is assumed to be equal to 0.4, while the total porosity is assumed to be equal to 0.7. The "dead volumes" at the top and at the bottom of the adsorber are assumed to be equal to 6e-5 m^3.

First cycle pressurization (case 1)

Below the results for case 1 are presented (see Table 7). It is necessary to increase the pressure in the adsorber from 0.06 bara to 1 bara after the adsorber regeneration. Gas is supplied with a constant flow rate to the adsorber, while the Aspen Adsorption simulates gas supply with a pipe of 2 m length (see Figure 14). However, from Figure 25, it can be seen that the inclusion of a pipe and an adsorber upstream 1 liter vessel does not

significantly affect the simulation of pressure gain. Still, including this in the Python calculation would greatly complicate the model. Instead, the Python model assumes that constant flow rate directly comes to the adsorber according to the scheme shown in Figure 15.

The Python model shows build up time 133 s., while the Aspen Adsorption predicts pressure buildup 150 s. The pressurization time is within the 1.5 - 3.5 minutes frame indicated in the VOCs Equinor experiments. Still, considering that the flow rate of case 1 is relatively high, one would expect the model to produce a pressure increase in about 1.5 minutes, which may indicate that the model exaggerates the amount of adsorbed gas.

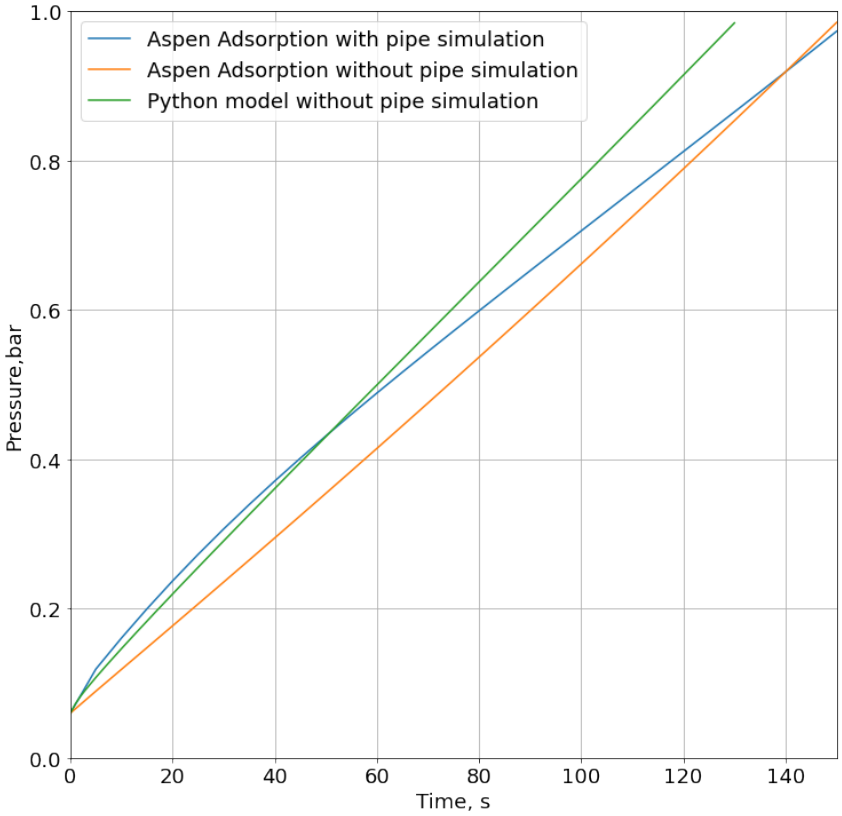


Figure 25: Pressure at the bottom of the adsorber

The pressure drop is minimal across the adsorber and is practically not noticeable in Figure 26, which shows the pressure along the length of the adsorber.

Velocity profiles during pressurization are shown in Figure 27. A constant molar flow rate is supplied to the adsorber, but the volumetric flow rate varies. The velocity of gas shows the behavior of volumetric flow rate since the area in the adsorber is constant. The volumetric flow rate is the largest when the pressure in the adsorber is the smallest (at the beginning of pressurization), as seen from the velocity figure.

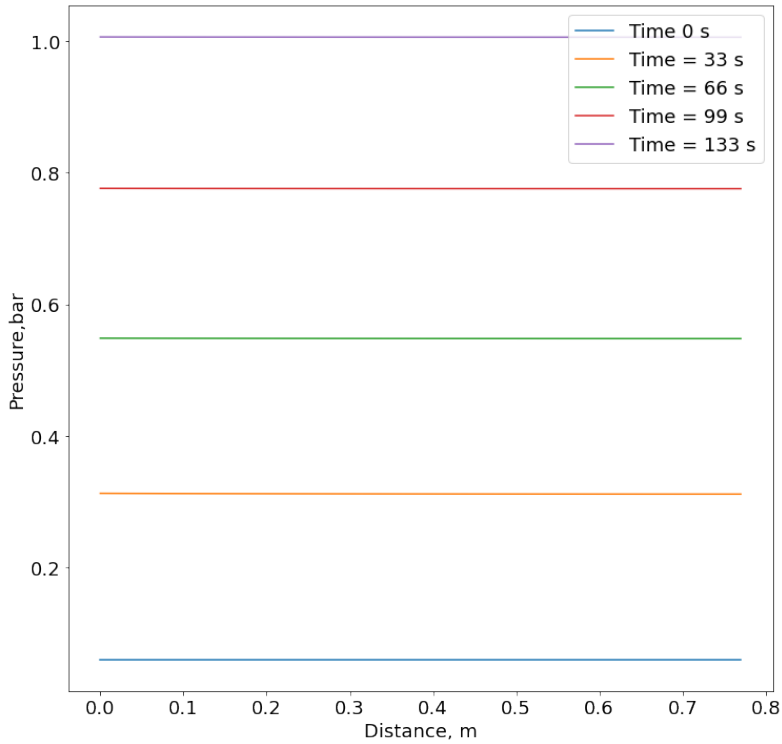


Figure 26: Pressure along the adsorber (Python model calculation)

In addition, the velocity rapidly decreases to a value close to zero along the adsorber. There is a "dead volume" behind the adsorber that is not filled with the adsorbent. So, the velocity drops to a value close to zero but not precisely to zero even when the top of the adsorber is closed.

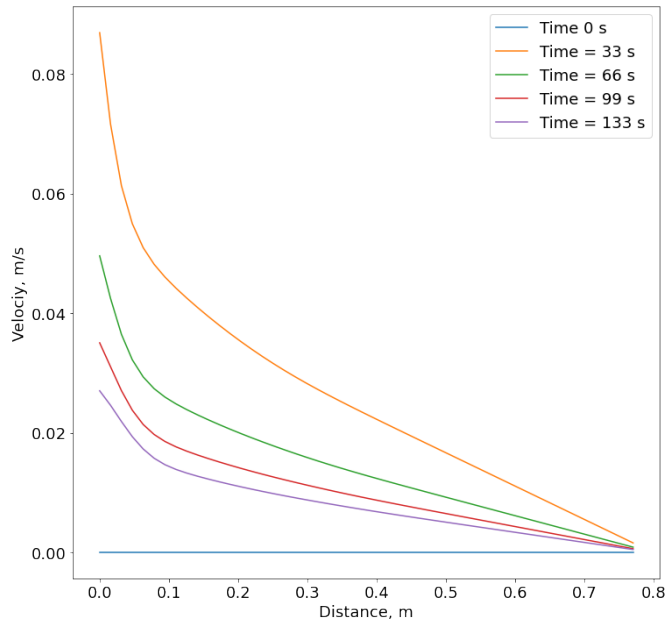
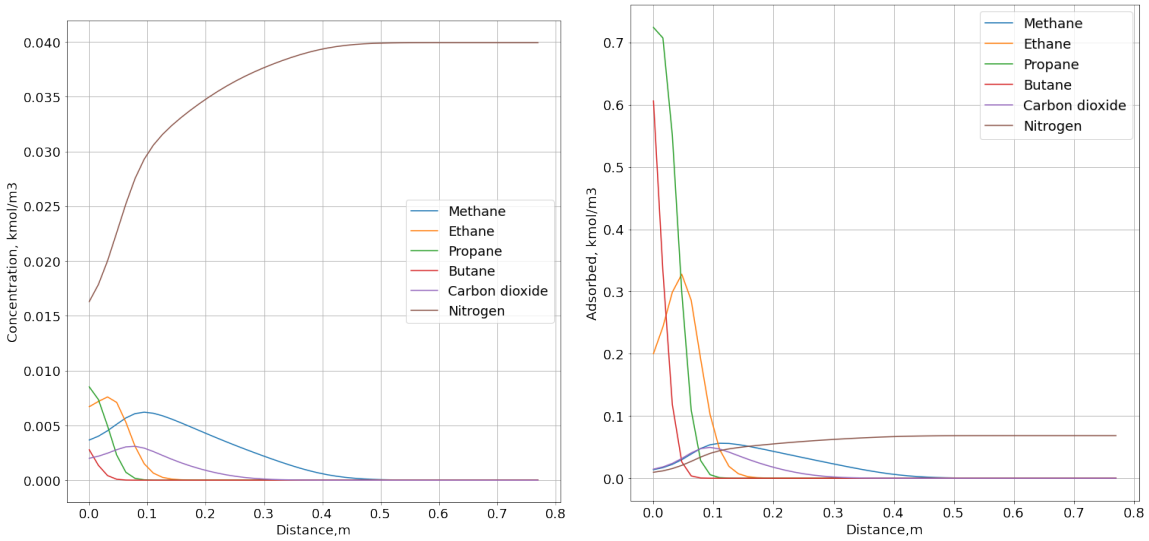


Figure 27: Velocity along the adsorber during pressurization

Penetration profiles after the pressurization step are shown in Figure 28. One can see the competitive behavior of the components. Methane with carbon dioxide advanced the farthest, followed by heavier hydrocarbons in order of increasing molecular weight: ethane, propane, and butane. At the same time, significant adsorption of heavier hydrocarbons is observed at the bottom of the adsorber since they adsorb more and penetrate less.



Gas phase concentration profile along the adsorber

Amount adsorbed

Figure 28: Gas penetration profiles after pressurization step

First cycle adsorption (case 1)

During the next step, the top of the adsorber is opened, and the pressure in the adsorber remains constant. As a result, only methane and carbon dioxide are achieved at the adsorber outlet after 15 min of adsorption, as shown in Figure 29. Furthermore, the flow rate of methane (0.12 mmol/s) after 400 s. is higher than the inlet flow rate of methane (0.075 mmol/s), which shows that some methane desorbs due to the competitive adsorption with other components. Carbon dioxide breakthrough occurs at 400 s., while other VOCs are fully adsorbed. As a result, carbon dioxide reaches the flow rate of almost 0.08 mmol/s, while the inlet flow rate of this component is 0.04 mmol/s. An increase in flow rate also happens because some of the carbon dioxide captured in the beginning is released afterward.

In addition, from the adsorption step results one may see that 100% purity of nitrogen in the product cannot be achieved under the prescribed conditions.

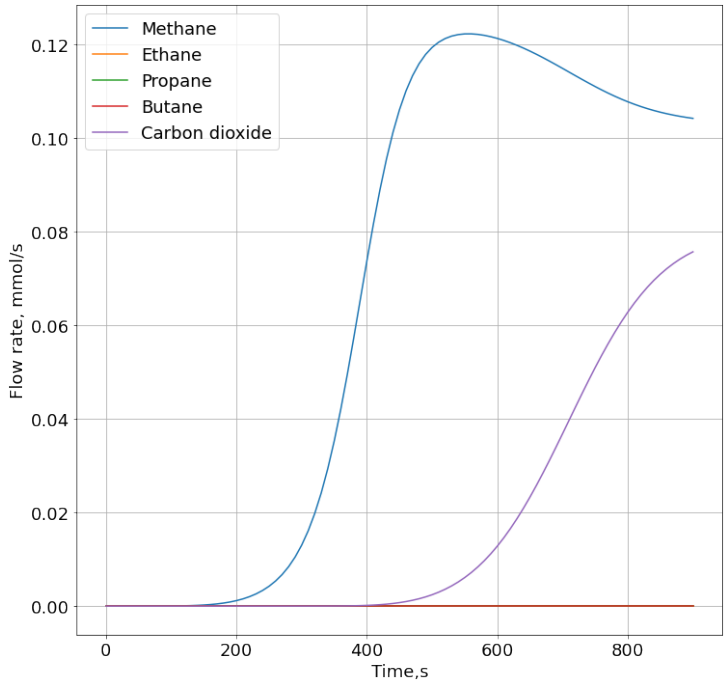
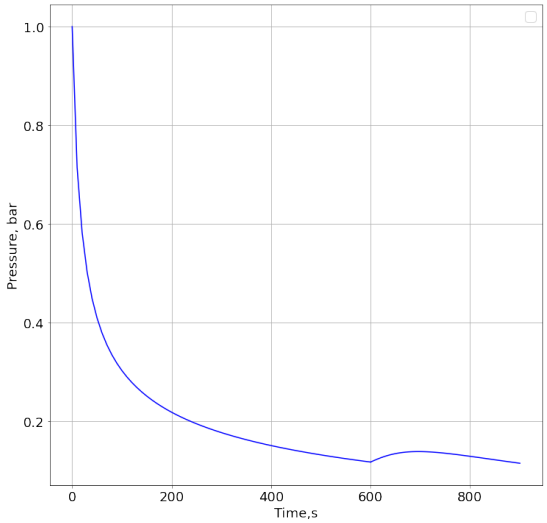


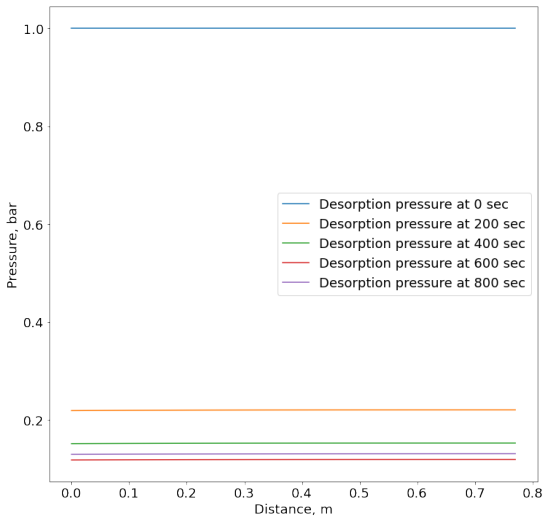
Figure 29: The flow rate of components after 15 min of adsorption

First cycle desorption and purge (case 1)

The pressure in the adsorber is reduced by pumping gas with a vacuum pump. The pressure decreases from 1 bara to 0.1 bara with a trend shown in Figure 30. In this case, a significant pressure reduction to approximately 0.2 bara occurs in the first 200 s., and then the pressure decrease slows down. After 600 s., there is a slight pressure surge as the purge starts. Also, the pressure drop across the adsorber is almost negligible.



Desorption pressure at the top of the adsorber



Desorption pressure along the adsorber

Figure 30: Pressure during desorption step

Velocity calculations are essential for correct boundary pressure determination. As seen in Figure 31, the volumetric flow rate decreases during desorption. However, the top of the adsorber opens when the purge starts at 600 s. After this time step, one can see that velocity increases to match the inlet flow rate of purge gas.

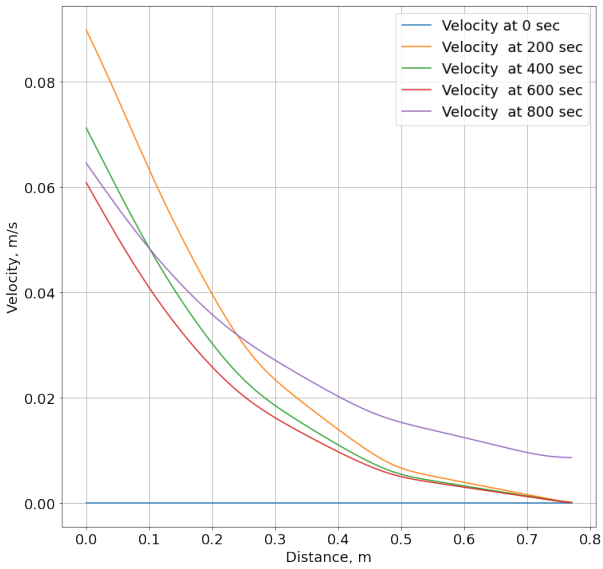


Figure 31: Velocity profiles during desorption

As seen in Figure 32, propane, ethane, and butane are mostly desorbed. Thus, these components form the main composition of the recycling stream. In the first seconds, a relatively large flow of nitrogen is observed. Then it almost instantly drops to zero, which is explained by the low adsorption of this component on activated carbon.

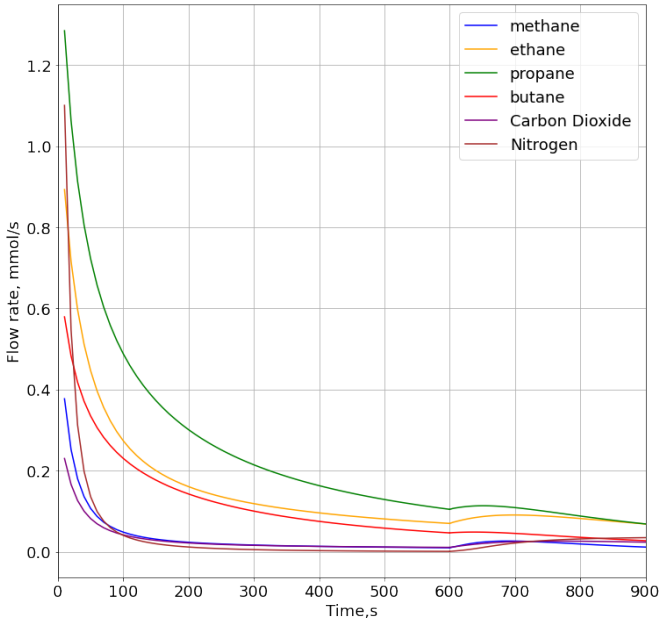


Figure 32: The flow rate of components during desorption and purge (purge starts after 600 s)

Cyclic steady state results case 1

Figure 33 shows the change in pressure during the first 8000 s. for case 1 (see Table 7). The time for pressurization is about 130 s. for this case. When the pressure reaches just above 1 bara, the adsorption process begins while the pressure remains constant. The adsorption cycle lasts 15 minutes, after which the vacuum pump is turned on, and the pressure drops rapidly to 0.1 bara during 300 s. of regeneration. Then pressure rises slightly after desorption due to the desorption of heavy hydrocarbons with the purge. The model developed in Aspen Adsorption shows approximately the same pressure result as the model developed in Python.

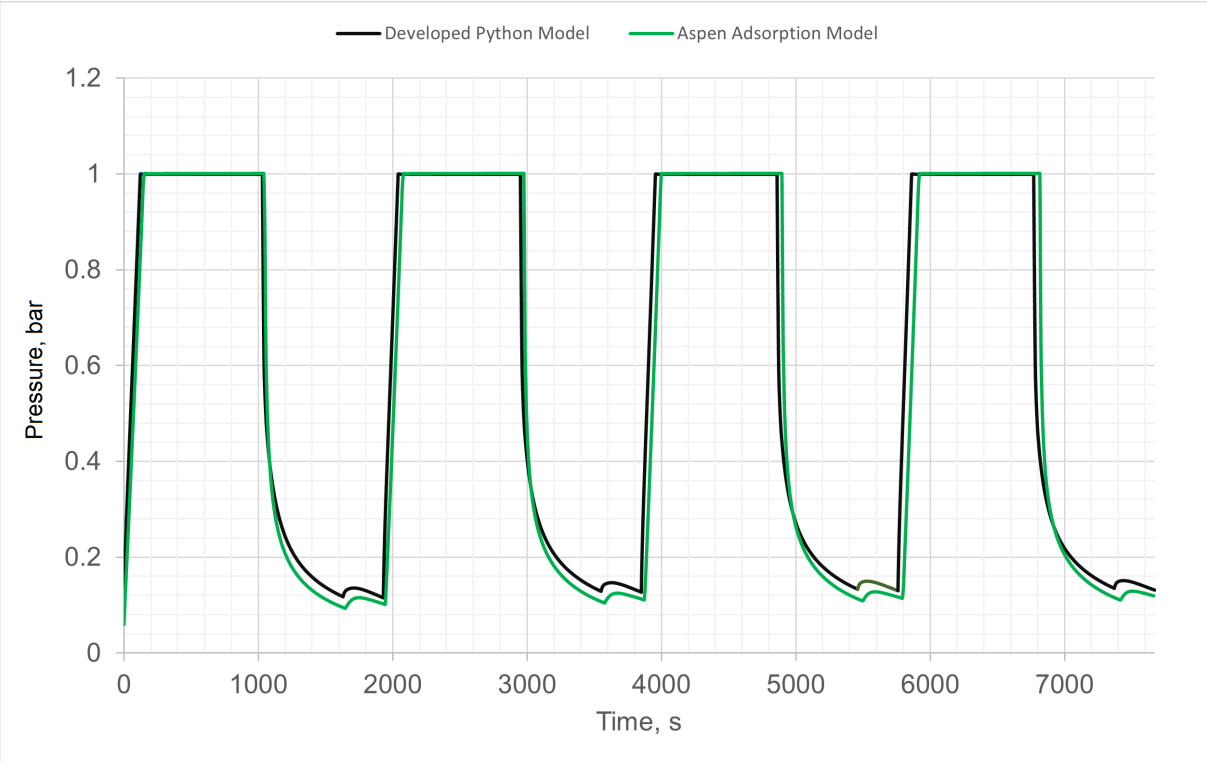
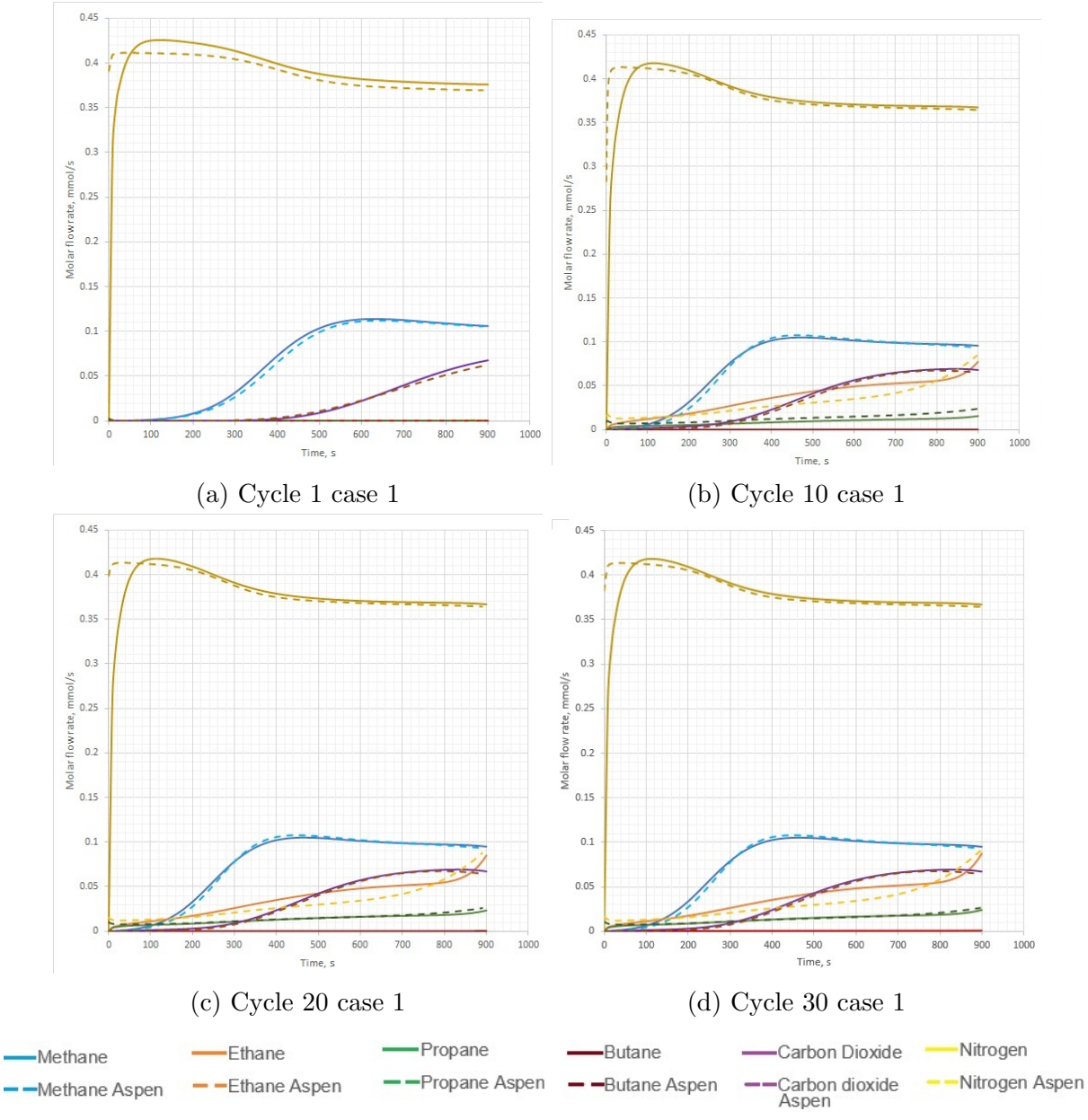


Figure 33: Pressure profile for case 1

The four figures shown in Figure 34 represent the molar flow rate of each gas exiting the adsorber. Cycle 1 means that all four stages were carried out only once: pressurization, adsorption, desorption, and purge. Thus, the number of cycles means how many times the whole process consisting of 4 stages was modeled. Initially, the adsorber is assumed to be in equilibrium with nitrogen at 0.06 bara. However, some other adsorbed components remain adsorbed after the first cycle, affecting the next cycles. One can see from the figures below that almost all heavy hydrocarbons are adsorbed in the first cycle, and only methane and carbon dioxide are achieved at the output. The release of ethane and propane begins in the subsequent cycles, but butane in all processes is completely absorbed.

The model developed in Aspen Adsorption shows approximately the same result as the model developed in Python.



(a) Cycle 1 case 1

(b) Cycle 10 case 1

(c) Cycle 20 case 1

(d) Cycle 30 case 1

Figure 34: Molar flow rates exiting the column

Figure 35 shows the capture efficiency for the first case when the cyclic-steady-state is reached (around 10 cycles). Capture efficiency of 100 % means no flow rate of a component exiting the column. Capture efficiency of 0 % means that the outlet molar flow rate of a component is equal to the inlet molar flow rate. The negative efficiency means that the outlet flow rate is higher than the inlet. The total accumulated parameter shows the integral of the graph for each component. In other words, the parameter shows how many moles are accumulated relative to how many moles of a component were supplied to the adsorber. As one can see from the model the capture of various VOCs happens

differently. The competitive behavior can be noticed. Once an equilibrium zone is formed in AC, components with stronger affinity will push components with less affinity out of the adsorbent. Consequently, the outlet flow of the "weak" components will increase above the inlet flow. It is seen from the figure that heavier components bind to the AC stronger than lightweight components. The time of breakthrough is decreased in order: N_2, CH_4, CO_2, C_2H_6, C_3H_8, C_4H_{10}. According to the model results, highly pure nitrogen is produced during the first 20 s. Overall the model shows positive capture efficiency of ethane, propane, and butane adsorption. Total accumulated efficiency is also high for these components.

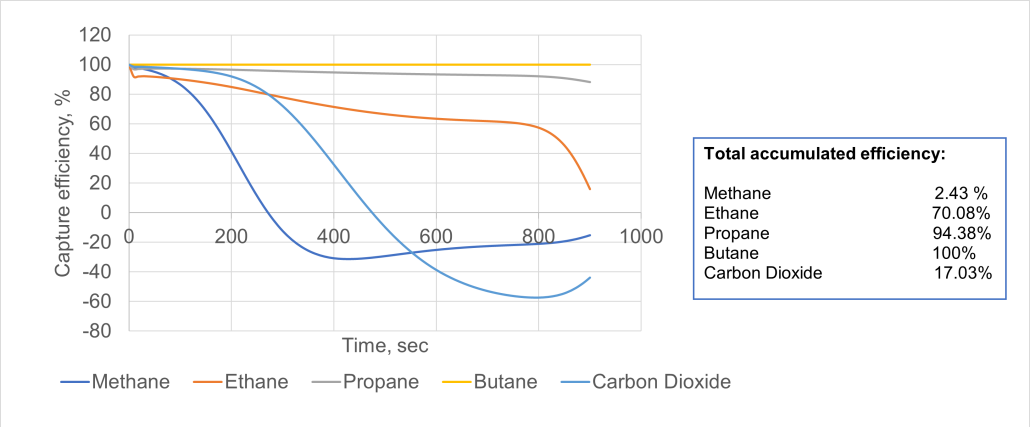


Figure 35: Capture efficiency case 1

Cyclic steady state results case 2

Figure 36 shows the change in pressure during the first 8000 s. for case 2 (see Table 7). The pressurization time is increased compared to the first case because the inlet flow rate is lower. It takes about 160 s. for the second case to pressurize the bed. As well as in the first case, the adsorption process starts when the pressure reaches 1 bara. Also, fewer hydrocarbons are adsorbed because of lower flow rate, and consequently, fewer hydrocarbons desorb, slightly reducing desorption pressure in comparison with the first case. The model developed in Aspen Adsorption (green line in Figure 36) shows approximately the same pressure result as the model developed in Python.

Figure 37 represent the molar flow rate of each component exiting the adsorber during the second case. The exiting gas consists of nitrogen, methane, and carbon dioxide. During the first 100 s. only nitrogen is produced with 0.3 $mmol/s$. Methane breakthrough occurs at around 100 s. reaching 0.07 $mmol/s$ at around 700 s., and carbon dioxide breakthrough occurs at around 400 s. reaching 0.03 $mmol/s$ at 900 s. Heavier hydrocarbons (ethane, propane, and butane) are fully adsorbed.

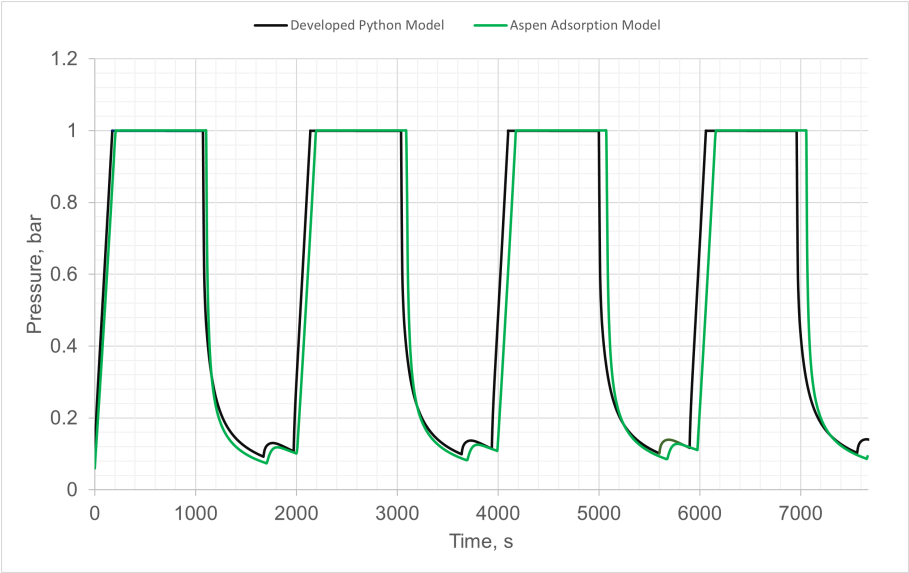
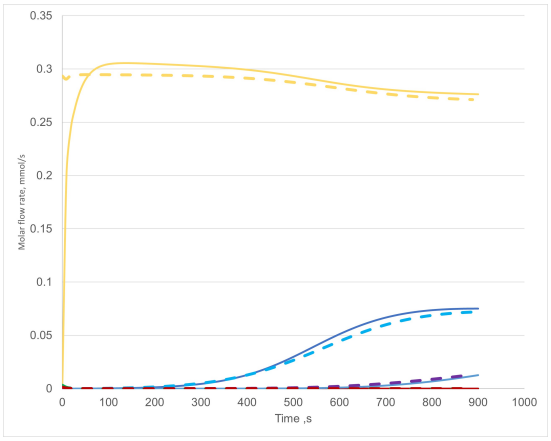
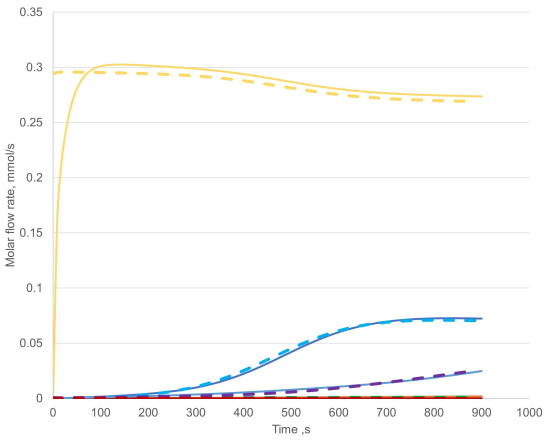


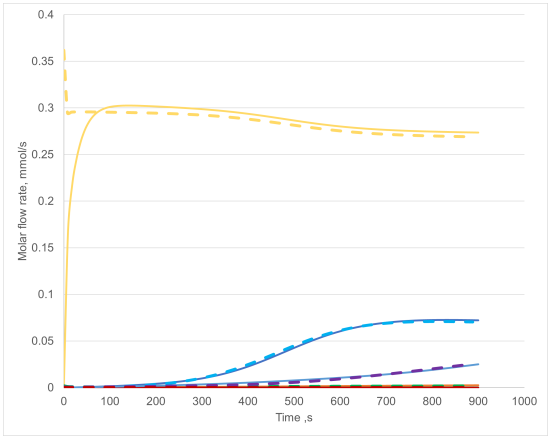
Figure 36: Pressure profile case 2



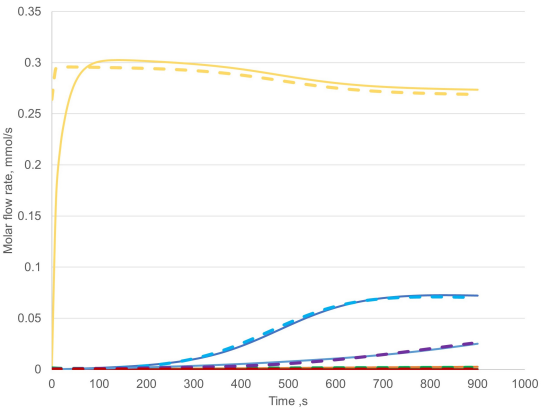
Cycle 1 case 2



Cycle 10 case 2



Cycle 20 case 2



Cycle 30 case 2

Figure 37: Molar flow rates exiting the column for case 2

Figure 38 shows the capture efficiency for the second case when the cyclic-steady-state is reached. As in the first case, one can see that heavier HC bind strongly to the adsorbent. The accumulated efficiency is also significantly higher compared to the first case, reaching almost 100 % for ethane, propane, and butane. That is because of a substantially lower flow rate of the inlet gas.

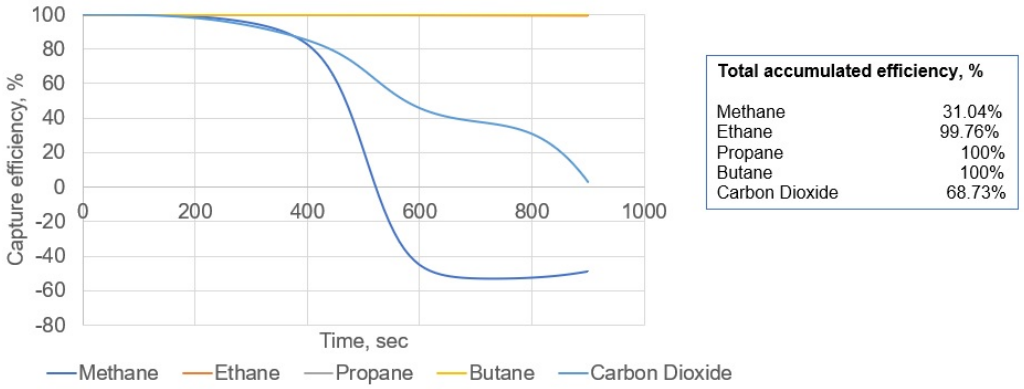


Figure 38: Capture efficiency case 2

Cyclic steady state results case 3

Figure 39 shows the pressure development during the first 8000 s. for case 3 (see Table 7). The pressurization time is increased compared to the previous cases since the inlet flow rate is lower. According to the model, it takes about 300 s. for the third case to pressurize the bed. Also, pressure drop during the desorption step is more significant in this case because fewer components adsorb and, hence, fewer components desorb, causing the reduction in pressure to 0.05 bara. The difference in the results between the Aspen Adsorption model and the developed Python model is more prominent in this case. According to the results from commercial software, it takes longer to pressurize the bed. This error is superimposed on every next cycle, and one can see a more significant difference than in cases 1 and 2. This difference was also noticeable in Figure 25 during the pressurization step of the first case, yet this difference becomes more significant with the inlet flow rate decrease. A possible explanation for this could be that Aspen Adsorption uses adiabatic conditions to describe pressure derivatives in the adsorbent bed dead space. In contrast, the simulation in Python uses ideal gas law with the isothermal assumption for this purpose (see Eqs. 28,29,30,31,32,33). Also, numerical errors can affect boundary pressure. However, the difference between the developed Python model and Aspen Adsorption in pressurization time should not affect the primary solution.

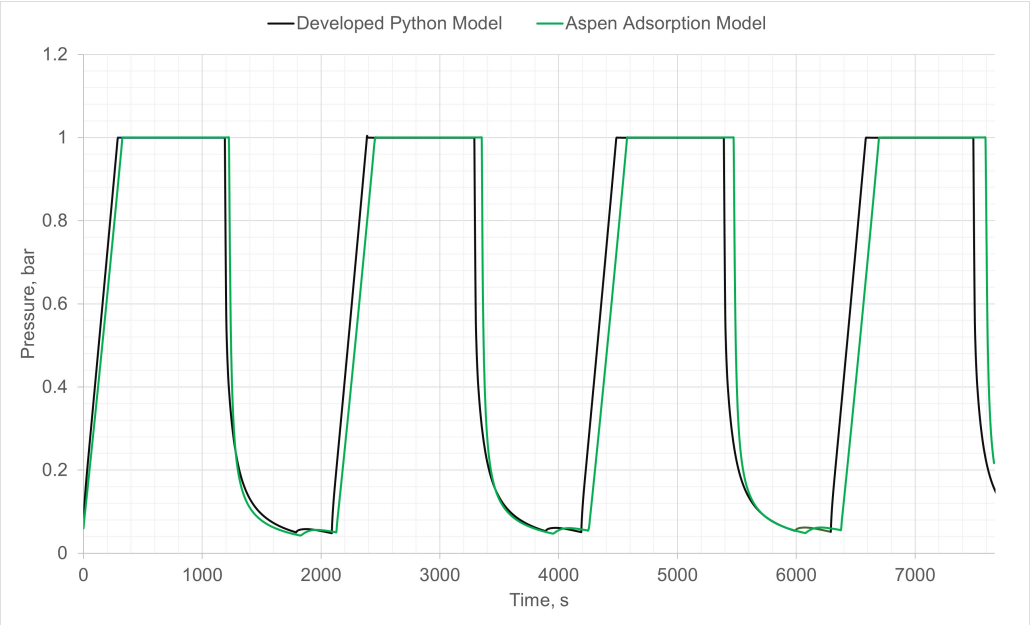
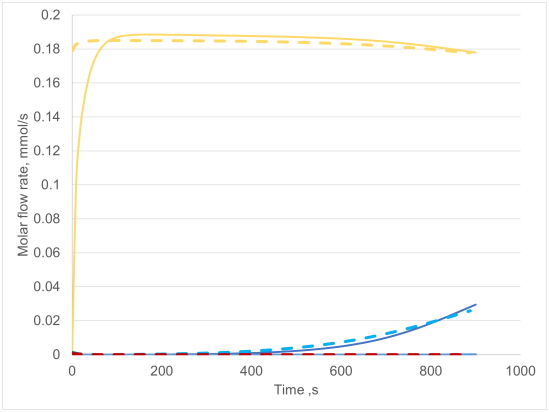
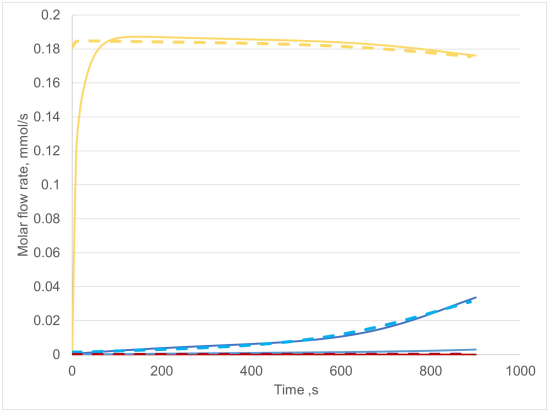


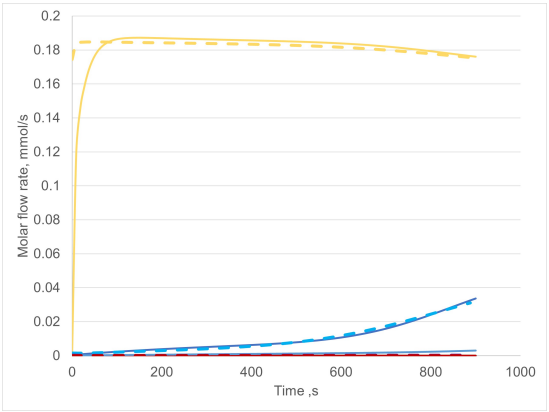
Figure 39: Pressure profile case 3



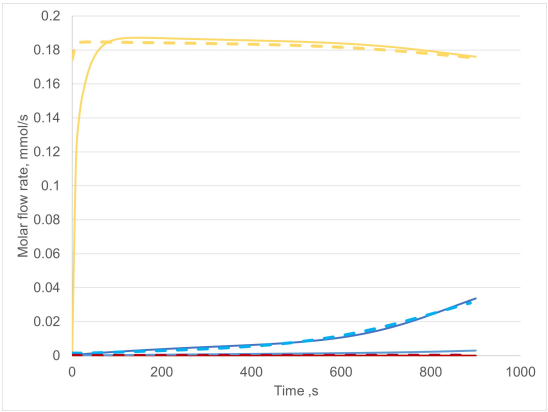
Cycle 1 case 3



Cycle 10 case 3



Cycle 20 case 3



Cycle 30 case 3

Figure 40: Molar flow rates exiting the column for case 3

The four figures shown in Figure 40 represent the molar flow rate of each component exiting the adsorber during the third case. The results of this case show that methane is the main product for the prescribed conditions, reaching almost 0.04 $mmol/s$ at 900 s. The Aspen Adsorption model shows approximately the same result.

Figure 41 shows the capture efficiency for the third case when the cyclic-steady-state is reached. There is no negative capture efficiency, which means that the exiting flow rate is less than the inlet flow rate for every component. Total accumulated efficiency is higher than 75 % for every component.

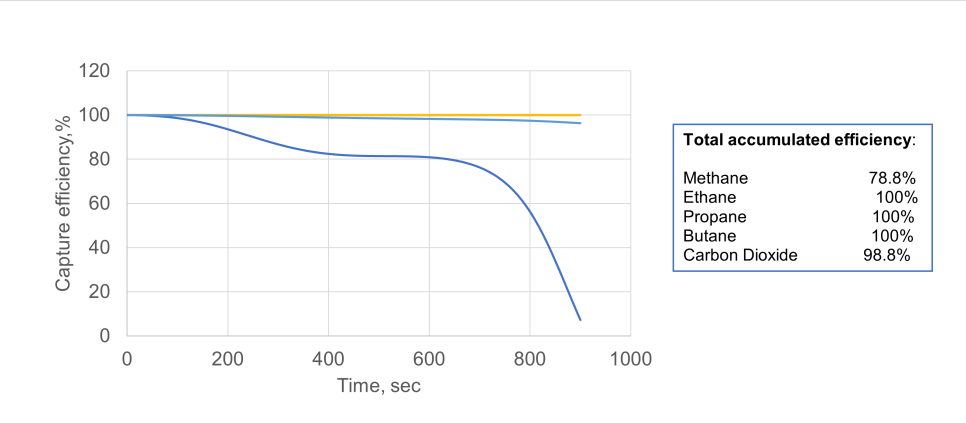


Figure 41: Capture efficiency case 3

6.4 VOCs simulation comparison with experiments

The described model was used to simulate the three cases of experiments performed by Equinor. Experimental data that will be compared is the outlet molar flow rates of the components during the adsorption step.

As can be seen in Figure 42 (flow rate exiting the column for the first case), the model could not well predict the behavior from the experiments for the first case. The model vividly overestimates the adsorption capacity for the VOCs. The molar flow rate of nitrogen is constant in the experiments, while in the model, it is affected by other components, and, thus, the molar flow rate of nitrogen is higher during the cycle. It is also clear from the experiments that methane comes through the adsorber, and the breakthrough of this component happens immediately. The breakthrough of carbon dioxide occurs at 100 s., while the developed model shows the time of 300 s. for this component to start exiting the column. At the end of the cycle, the model shows a higher molar flow rate of methane exiting the column because of the delayed breakthrough. In the experiments, ethane reaches almost 0.25 $mmol/s$ at 700 s. before decreasing to 0.15 $mmol/s$ due to the breakthrough of propane and butane, which affects the adsorption capacity of ethane.

One should compare other cases to see the difference for analyzing the possible improvements of the model for a better prediction of the experiment results.

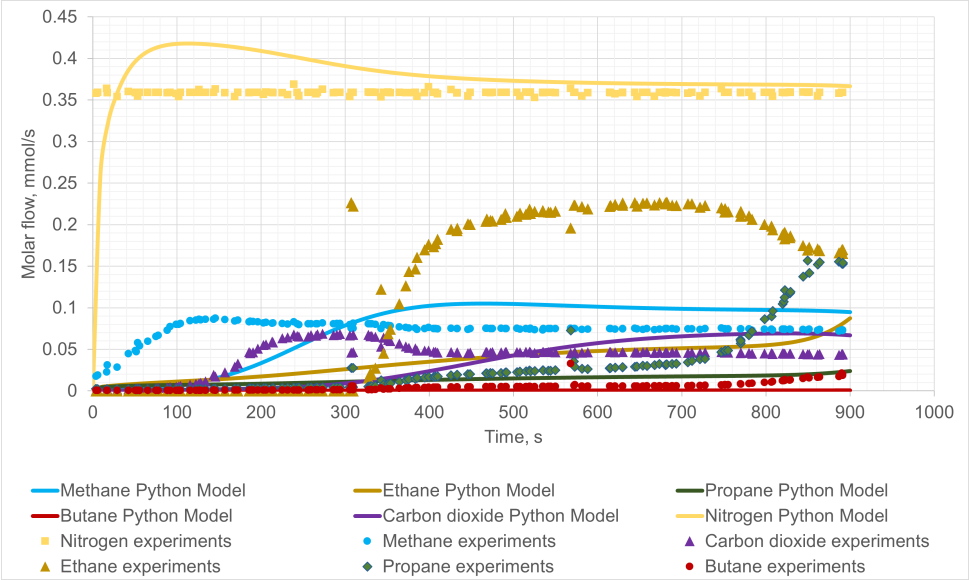


Figure 42: Comparison with experiments case 1

As can be seen in comparison with the next case, the model prediction is better. However, only three components can be compared. First, the nitrogen flow rate is higher than in the experiments. Methane and carbon dioxide breakthroughs coincide; however, the shape of the flow rate increase is different.

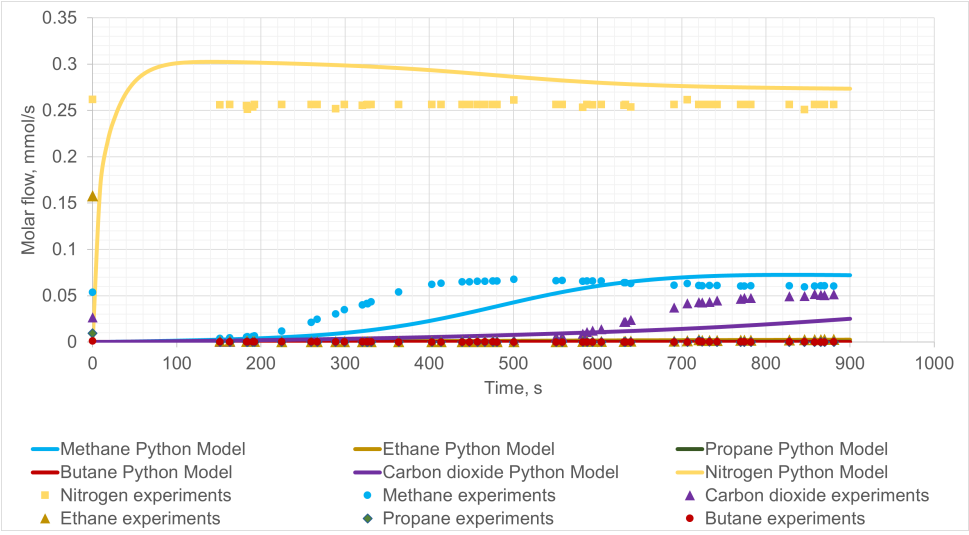


Figure 43: Comparison with experiments case 2

The same situation represents the third case. The nitrogen flow rate is higher than in the experiments, and methane breakthrough happens simultaneously with a different shape.

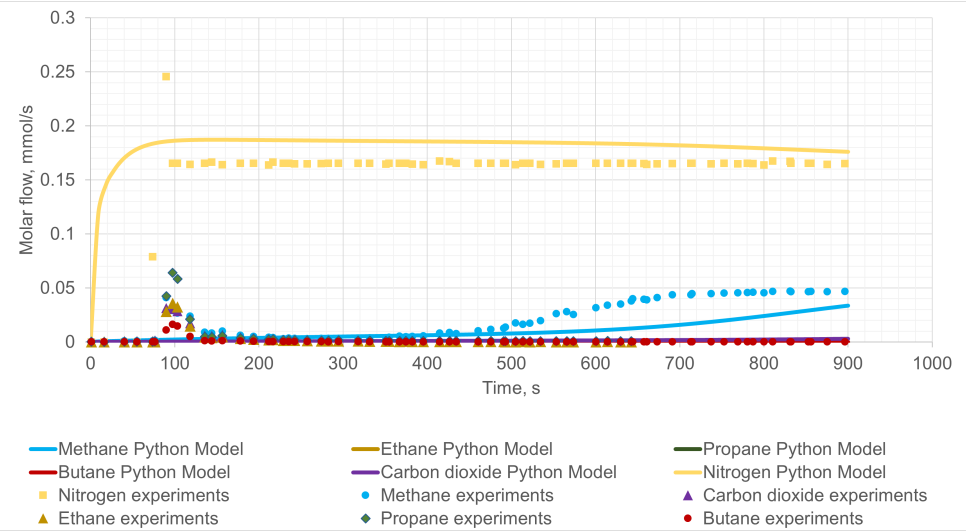


Figure 44: Comparison with experiments case 3

Case 2 and 3 show better results, which could indicate that as AC becomes less saturated with heavier HC, the model describes the system better. The possible explanation for why the results do not coincide is the follows:

1. The model does not correctly predict the equilibrium amount of the gas adsorbed. In other words, the Extended Langmuir equation based on the pure component isotherms cannot reasonably predict the mixture behavior, and more complicated types of isotherms should be checked.

2. The equilibrium data on the adsorbent are not well provided. The data was taken from the available literature.

3. The linear driving force model for the presented case may not provide a proper solution. For example, the quadratic and pore diffusion models can be used instead.

4. Some unclear conditions relevant to the experiments but not included in the simulation can affect the results

6.5 Parametric analysis of VPSA configuration

This chapter aims to analyze the effect of adsorption pressure, vacuum pump characteristics, gas flow rates , and purge flow rates and adsorption cycle time on the adsorption purity and recovery of each component. This chapter also examines how the results correspond to the expected ones and whether they represent the theoretical understanding. For this purpose, the second case (see Table 7) is chosen because it is more similar to the experiment results (see Figure 43).

The key indicators of the VPSA efficiency are the components total accumulated efficiency, purity and recovery. The purity shows the relationship between the flow rate of a component exiting the column to the product flow rate during the adsorption step. In other words the integral of molar fraction of the component in the product stream during adsorption step. Eq.(41) is used to calculate this parameter.

$$Purity_i = \frac{\int_0^{t_{ads}} F_{i,product}\, dt}{\int_0^{t_{ads}} F_{product}\, dt} \cdot 100\% \tag{41}$$

here $Purity_i$ is the purity of component i [%], t_{ads} is the time when adsorption step ends [s], $F_{i,product}$ is a molar flow rate of component i in the product during adsorption step [kmol/s], $F_{product}$ is a total molar flow rate of the product during adsorption step [kmol/s].

The recovery parameter shows how much of the component is achieved in the product and recycling streams in comparison to the inlet flow rate. The equations to calculate the recovery of components are:

$$Recovery_{N2} = \frac{\int_0^{t_{ads}} F_{N2,product}\, dt - \int_0^{t_{PG}} F_{N2,recycling}\, dt}{\int_0^{t_{PG}} F_{N2,feed}\, dt} \cdot 100\% \tag{42}$$

$$Recovery_i = \frac{\int_0^{t_{ads}} F_{i,recycling}\, dt - \int_0^{t_{PG}} F_{i,product}\, dt}{\int_0^{t_{PG}} F_{i,feed}\, dt} \cdot 100\% \tag{43}$$

here $Recovery_i$ is the recovery of component i [%], t_{PG} is the time when purge step ends [s], $F_{i,recycling}$ is a molar flow rate of component i in the recycling stream during regeneration and purge [kmol/s], $F_{i,feed}$ is a feed molar flow rate of a component i in the cycle [kmol/s].

The primary purpose of VOCs separation is to obtain high purity nitrogen at the adsorber outlet. As for environmental pollutants, it is beneficial to send them back for processing. The nitrogen recovery parameter indicates what percentage of the incoming nitrogen stream is recycled and what percentage is sent to the adsorber outlet as the product. For example, -100 % recovery means that all nitrogen has gone to processing, and 100% means that all feed nitrogen has gone to the product. Thus, 0% nitrogen recovery would mean that the nitrogen feed stream is evenly distributed between products and recycled streams. The other is true for methane, ethane, propane, butane, and carbon dioxide since sending them for recycling is beneficial. Therefore, for example, 100% recovery for methane would mean that the entire product was sent for recycling.

The effect of adsorption pressure on the purity and recovery of the components is analyzed by varying the adsorption pressure from 1 to 4 bara. The cycle schedule is not changed, except for the pressurization time because it increases with adsorption pressure. Also, the

pump characteristics only enable correct calculation of the volumetric flow rate through the pump until 1 bara, which may give an unrealistic situation in the region above 1 bara. Moreover, technical constraints due to high pressure should be considered. In addition, compressor power consumption increases as the adsorption pressure rise. Figure 45 shows the recovery and purity of nitrogen as pressure grows. As pressure rises, the purity of nitrogen increases by 13 %, but recovery slightly decreases. Thus, it is favorable to get higher adsorption pressure, but economic evaluation should be considered.

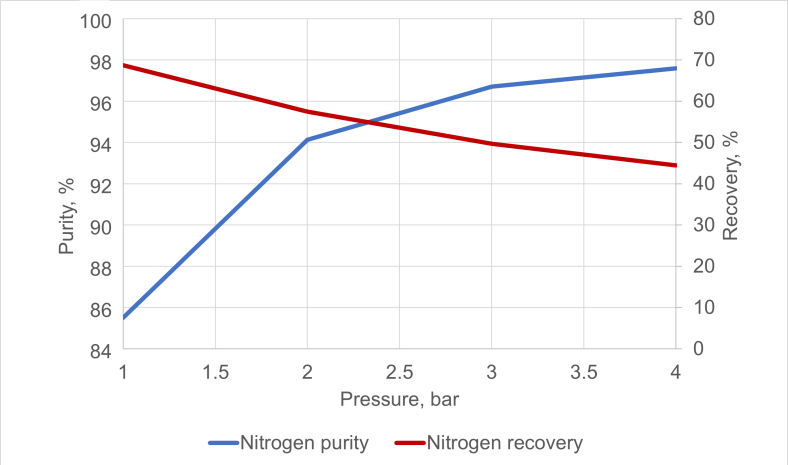


Figure 45: Nitrogen purity and recovery dependence on adsorption pressure

Figure 46 shows that total accumulated efficiency increases with increasing pressure, and it tends to 100 % for every component when the nitrogen purity tends to 100% (see Figure 45) .

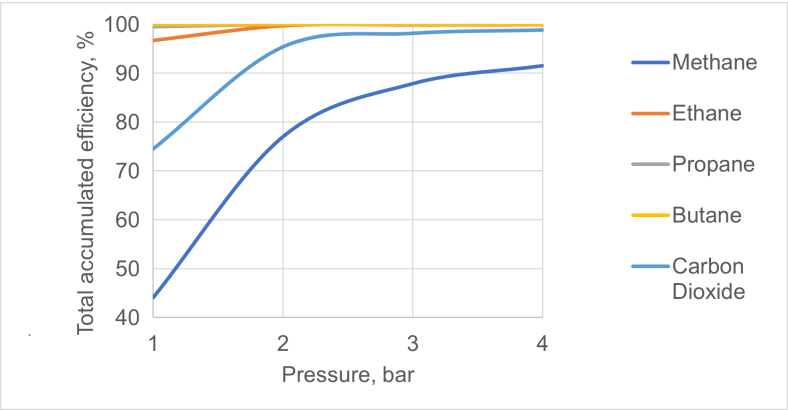


Figure 46: Total accumulated VOCs efficiency dependence on pressure

With the data of case 2, most of the methane entering the adsorption system goes to the product during adsorption step. The pressure rise can improve this situation and increase the methane recovery to 80 % at 3.5 bara. At the same time, the recovery of the remaining components will be close to 100 % as seen in Figure 47.

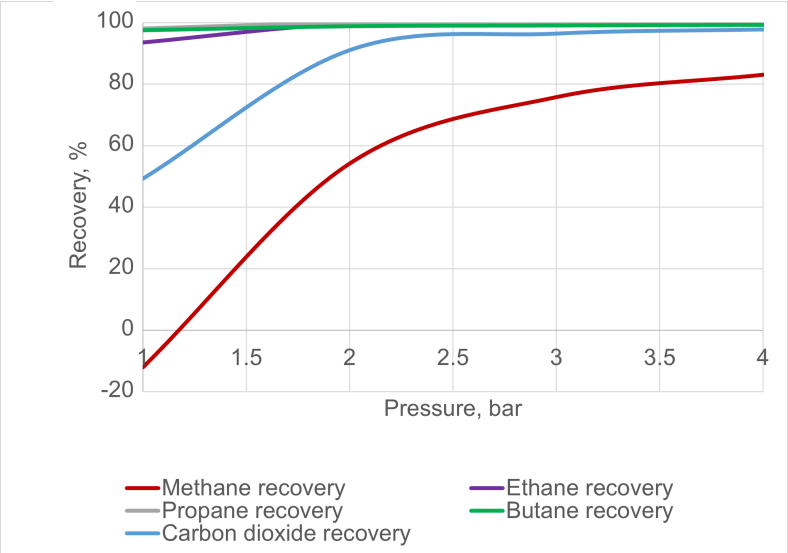


Figure 47: VOCs recovery pressure dependence

The changes in purity and recovery of nitrogen according to the volumetric flow rate of the vacuum pump are shown in Figure 48. As the volumetric flow rate decreases, the final desorption pressure increases. Thus, more adsorbates are left in the adsorbent after the cycle, which is proven by the weight increase of the adsorbent shown in Figure 49. As the desorption pressure increases, the purity of nitrogen decreases, but the recovery increases. Therefore, increasing volumetric flow rate of the pump would be favorable in terms of AC degradation and product purity.

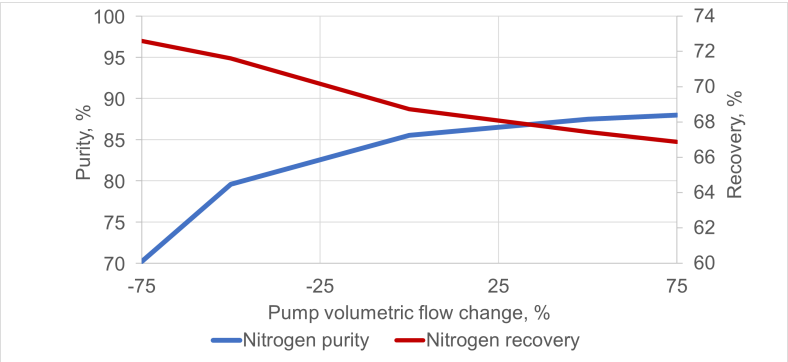


Figure 48: Nitrogen purity and recovery dependence on volumetric pump flow rate

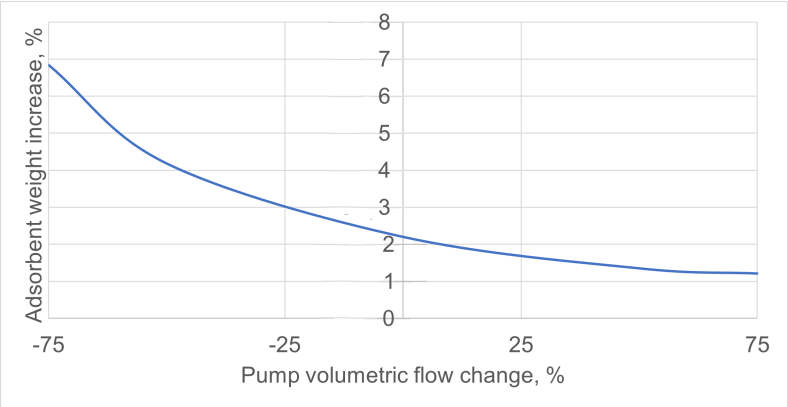


Figure 49: Adsorbent weight increase dependence on volumetric pump flow rate

The effect of gas inlet flow rate on the purity and recovery of the components was analyzed by varying the inlet gas flow rate from 212 Nml/min to 3212 Nml/min. The results for nitrogen purity and recovery parameters are shown in Figure 50. As the flow rate increases, the purity of nitrogen decreases, and recovery gains, meaning that relatively less nitrogen comes to recycling. One can see that the effect of inlet gas flow rate is quite strong on both purity and recovery; however, the recovery rate becomes less steep after 1712 Nml/min. When more gas comes to the adsorber, heavier hydrocarbons tend to breakthrough earlier, which can be seen by the increase in their purity in Figure 50 and that explains the rapid reduction in purity of nitrogen. Also, the adsorbent weight increase decreases with inlet gas rate reduction, as shown in Figure 52. Thus, getting a lower inlet flow rate to the adsorbent is favorable. However, this would require more beds to treat the same amount of gas.

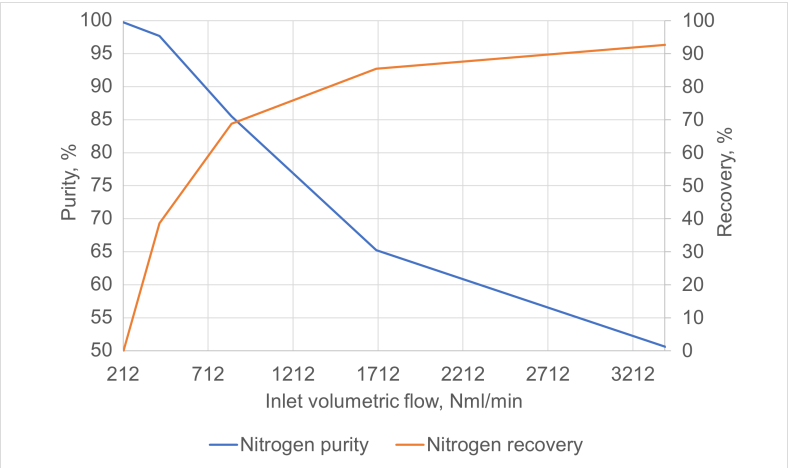


Figure 50: Nitrogen purity and recovery dependence on gas inlet flow rate

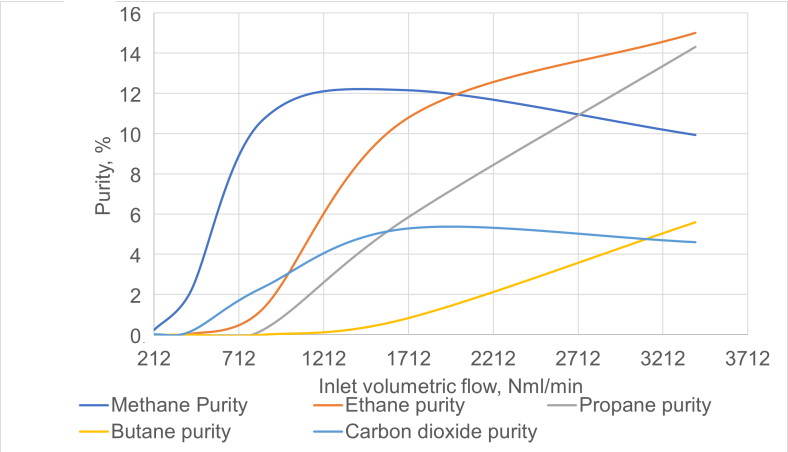


Figure 51: VOCs purity dependence on gas inlet flow rate

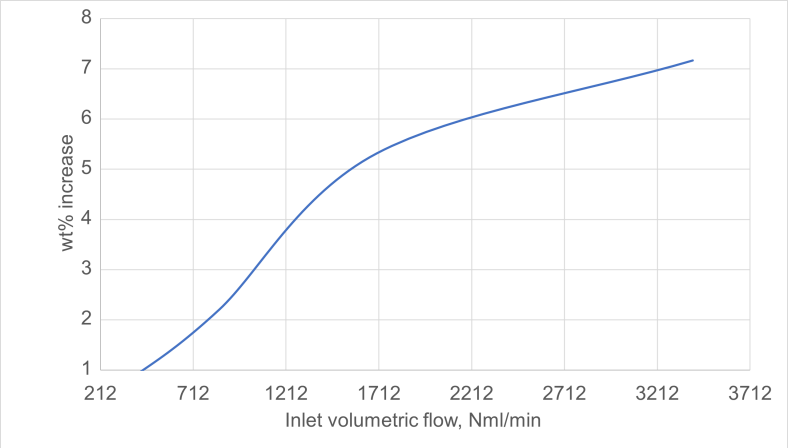


Figure 52: Adsorbent weight increase dependence on gas inlet flow rate

The effect of the purge flow rate on the purity and recovery of the components was analyzed by varying the purge gas flow rate from 25 Nml/min to 175 Nml/min. The purge gas is nitrogen in the model. As expected, nitrogen purity will increase with purge gas flow rate because the adsorbent will be "cleaner" after vacuuming. However, as seen in Figure 53, the drop in nitrogen recovery will be significant because the used nitrogen for purge will directly go to the recycling.

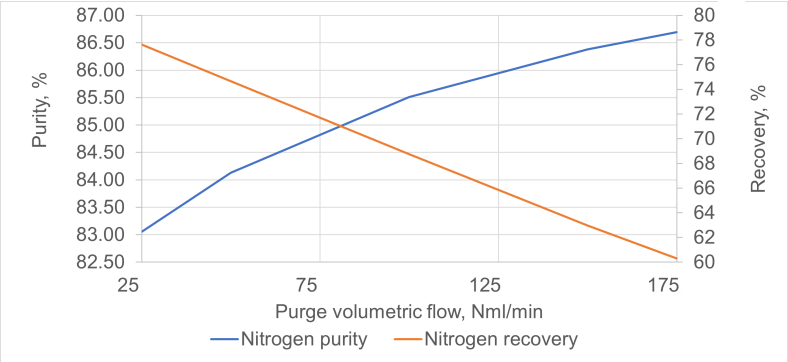


Figure 53: Nitrogen purity and recovery dependence on purge flow rate

The primary purpose of the purge gas is to lower the VOCs partial pressure in the adsorber and cause their desorption. Thus, increasing the purge rate will cause an improvement in the purity of the adsorbent, as shown in Figure 54. However, to get an almost "clean" adsorbent would require quite high purge flow rate.

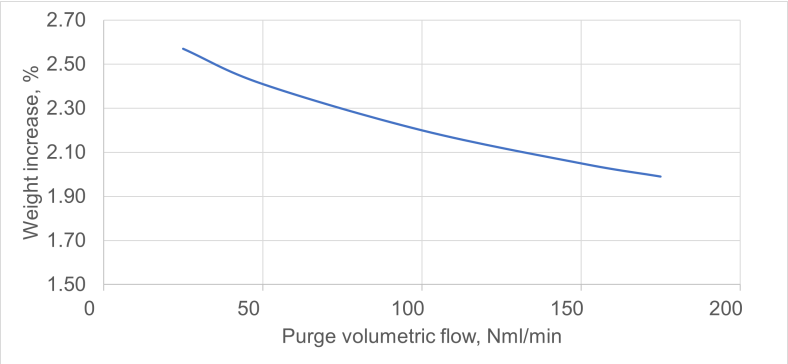


Figure 54: Adsorbent weight increase dependence on purge flow rate

Finally, the effect of the adsorption cycle on the purity and recovery of the components was analyzed by varying adsorption time from 200 s. to 1800 s. As the adsorption time increases, the breakthrough of VOCs occurs, decreasing the nitrogen purity at the adsorbent outlet. Recovery change, as seen in Figure 55, is also quite significant.

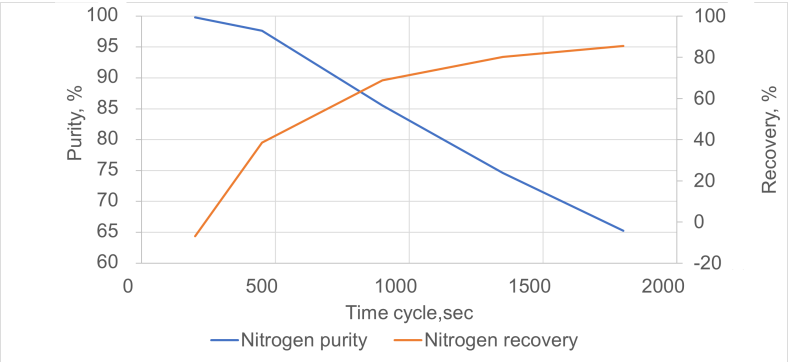


Figure 55: Nitrogen purity and recovery dependence on adsorption time

It is also evident that as the duration of the adsorption stage increases, more VOCs will be adsorbed, which will cause more significant degradation of the adsorbent, as shown in Figure 56.

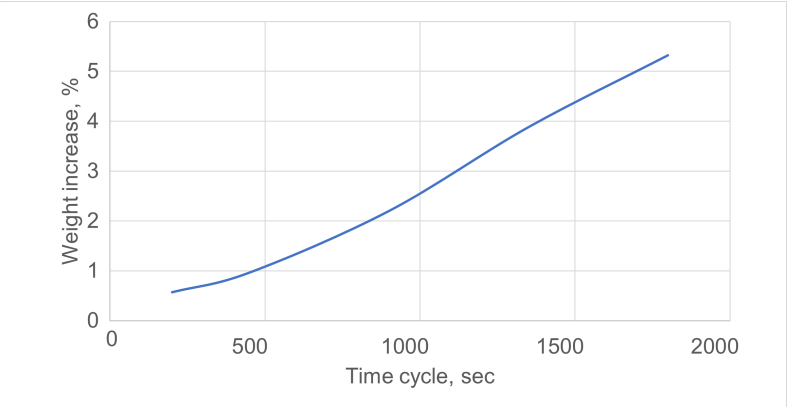


Figure 56: Adsorbent weight increase dependence on adsorption cycle

Thus, recovery and purity are inversely proportional. The same inverse relationship can be seen in other studies on VPSA adsorption [12, 18, 24]. However, it is challenging to find works on multicomponent VOCs adsorption optimization. Figure 57 shows the nitrogen purity dependence on the recovery under the studied parameters. It should be noted that when changing the parameters of the pump or the amount of purge gas supplied, it is impossible to achieve 100 % purity of the nitrogen output. Even with a pure adsorbent under the given conditions of case 2, methane and carbon dioxide will be released, as seen in Figure 43. Only by changing adsorption capacity of adsorbent or the amount of gas to be adsorbed, one can get almost 100 % nitrogen purity. It can be seen from Figure 57 that it is possible to achieve 100 % nitrogen at the outlet by changing the described parameters only if the maximum nitrogen recovery is around 40 %. Therefore, if a minimum required output purity is set, it is possible to find an optimal recovery value and vice versa. However, more work needs to be done to match the experiments to trust the results given in this chapter. Also, one should consider the economic evaluation during a particular optimization process.

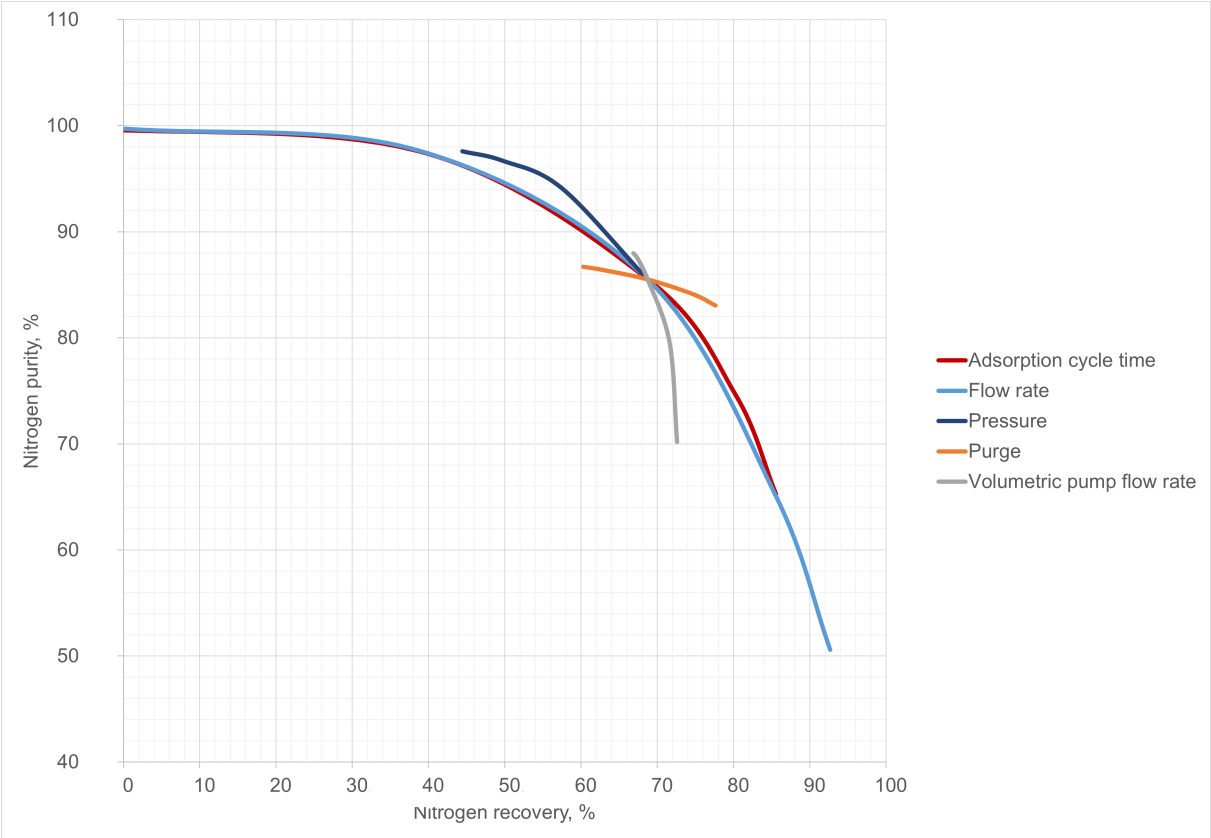


Figure 57: Purity - recovery nitrogen dependence for case 2

7 Conclusions

This project reports a model development of VPSA technology in Python applied to VOCs adsorption. The main conclusions derived from this project are:

1. Literature review shows that some VOCs adsorption was highly investigated, such as adsorption of methane and carbon dioxide. However, the full range of light alkanes adsorption modeling needs to be considered more. Extended Langmuir isotherm model and linear driving force for mass transfer are mostly used for the adsorption modeling. The most popular software to simulate adsorption process was Aspen Adsorption.

2. To simulate the adsorption equilibrium characteristics of methane, ethane, propane, butane, carbon dioxide and nitrogen, an adsorption equilibrium data from Esteves et al. work [14] was used, and the model parameters were determined using EL isotherm model.

3. Two models are developed in the Python (with and without pressure drop calculations) to simulate the adsorption step at a constant pressure. Compared to simulations from the commercial software Aspen Adsorption and experimental results outlined in Cavenati et al. work [5], the developed Python models showed good result. Finally, the Python model also described the trend of VPSA .

4. The VOCs adsorption process is modeled with 30 cycles for three different cases. The cyclic-steady-state is reached at 10th cycle already. The VOCs are methane, ethane, propane, butane, carbon dioxide, and nitrogen. However, the developed mass transfer and equilibrium models could not accurately describe the experiments performed by Equinor. Thus, future work is required for this particular set of components.

5. Parametric analysis is conducted based on the five crucial variables: gas inlet flow rate, purge flow rate, adsorption and desorption (vacuum pump characteristic) pressures, and adsorption time. Almost 100 % nitrogen purity can be achieved, meaning that no VOCs will be emitted. However, this would result in 40% nitrogen recovery, meaning that a considerable amount of nitrogen will be recycled with other VOCs. One should consider how the changed parameters affect the economy before performing an optimization analysis. Section 6.5 showed that the model is consistent with the theoretical understanding. The trends in purity and recovery coincide with those described earlier in the literature for other components. The model allows calculations to optimize the process when the model matches with experiments.

8 Proposals of Future Work

The developed Python model for VPSA simulation shows very good agreement with Aspen Adsorption and Cavenati and co-authors experiments [5]. However, more work is required to get a good prediction of VOCs VPSA adsorption on AC. Therefore, it is evident that the parameters for describing these components' equilibrium and mass transfer are not correct. Case 2 and case 3 showed better results, which could indicate that as AC becomes less saturated with heavier hydrocarbons, the model describes adsorption better. One option to improve the model is to use a more complex scheme for describing multicomponent equilibrium such as IAST (Ideal Adsorption Solution Theory) which requires the solution of an implicit algebraic system of equations. Then, it is possible to find the mass transfer coefficients to describe the experiments obtained with the correct equilibrium model. Temperature changes were neglected in the modeling of VOCs adsorption. However, this should not affect the overall result since the temperature variations are 10 degrees. In case of a more accurate agreement with the experimental results, temperature calculation should improve the simulation.

Moreover, the number of equations and parameters on which the result depends significantly increases with the number of components. Therefore, it would be easier to understand the discrepancy between the experiments and how one of the components affects the other if experiments were conducted with fewer components.

Also, the model can be improved to describe the actual process, but at the same time, complicated. If more accurate results are required, the following adjustments can be made:

1. General possible improvements.
   (a) A more realistic two-dimensional model can be developed.
   (b) The model can be extended, and components C5+ may be included.
2. Mass balance possible improvements. Axial dispersion terms can be added.
3. Energy balance possible improvements. The heat of adsorption dependence on temperature and loading may be introduced.

Bibliography

[1] Hamid Rajabi et al. 'Emissions of volatile organic compounds from crude oil processing – Global emission inventory and environmental release'. In: *Science of the Total Environment* 727 (2020).

[2] *OECD: Organisation for European Economic Co-operation. Air and GHG emissions*. URL: https://data.oecd.org/air/air-and-ghg-emissions.htm.

[3] Wei Wei et al. 'Characteristics of volatile organic compounds (VOCs) emitted from a petroleum refinery in Beijing, China'. In: *Atmospheric environment* 89 (2014).

[4] *Activated Carbon in Vapour Recovery Units*. URL: https://coolsorption.com/wp-content/uploads/2017/05/TSM_feb-mar-2017-Activated-Carbon.pdf.

[5] Simone Cavenati, Carlos A. Grande and Alirio E. Rodrigues. 'Upgrade of methane from Landfill gas by pressure swing adsorption'. In: *Energy and Fuels* 19 (2005), pp. 5108–5117.

[6] Pål Søraker. *Equinor company report : test of C1-C10 adsorption and desorption on activated carbon*. 2021.

[7] Dereje Tamiru Tefera et al. 'Modeling Competitive Adsorption of Mixtures of Volatile Organic Compounds in a Fixed-Bed of Beaded Activated Carbon'. In: *Environmental Science and Technology* 48 (2014), pp. 5108–5117.

[8] Show-Chu Huang, Tsair-Wang Chung and Hung-Ta Wu*. 'Effects of Molecular Properties on Adsorption of Six-Carbon VOCs by Activated Carbon in a Fixed Adsorber'. In: *ACS Omega* 6 (2021), pp. 5825–5835.

[9] Shanshan Wang et al. 'A mini-review on the modeling of volatile organic compound adsorption in activated carbons: Equilibrium, dynamics, and heat effects'. In: *Chinese Journal of Chemical Engineering* 31 (2021), pp. 153–163.

[10] Duong D. Do. *Adsorption Analysis: Equilibria and Kinetics*. Imperial College Press, 1998.

[11] Tom R. C. Van Assche, Gino V. Baron and Joeri F. M. Denayer. 'An explicit multicomponent adsorption isotherm model: accounting for the size-effect for components with Langmuir adsorption behavior'. In: *Adsorption* 24 (2018).

[12] Young Jun Kim, Young Suk Nam and Yong Tae Kang. 'Study on a numerical model and PSA (pressure swing adsorption) process experiment for CH4/CO2 separation from biogas'. In: *Energy* (2015), pp. 732–741.

[13] Jinsheng Xiao et al. 'Thermal effects on breakthrough curves of pressure swing adsorption for hydrogen purification'. In: *International Journal of Hydrogen Energy* 41 (2016), pp. 8236–8245.

[14] Isabel A.A.C. Esteves et al. 'Adsorption of natural gas and biogas components on activated carbon'. In: *Separation and Purification Technology* 62 (2008), pp. 281–296.

[15] Shuangjun Lia et al. 'Mathematical modeling and numerical investigation of carbon capture by adsorption: Literature review and case study'. In: *Applied Energy* 221 (2018), pp. 437–449.

[16] Yonghou Xiao et al. 'Numerical simulation of low-concentration CO2 adsorption on fixed bed using finite element analysis'. In: *Chinese Journal of Chemical Engineering* (2020), pp. 1–10.

[17] Giorgio Vilardi et al. 'Exergy and energy analysis of biogas upgrading by pressure swing adsorption:Dynamic analysis of the process'. In: *Energy Conversion and Management* 226 (2020).

[18] Sol Ahn et al. 'Layered two- and four bed PSA processes for H_2 recovery from coal gas'. In: *Chemical Engineering Science* (2012), pp. 413–423.

[19] P.Cruz et al. 'Cyclic adsorption separation processes: analysis strategy and optimization procedure'. In: *Chemical Engineering Science* (2003), pp. 3143–3158.

[20] P.E. Alan Gabelman. 'Adsorption Basics: Part 1'. In: *American Institute of Chemical Engineers* (2017), pp. 48–53.

[21] Kevin R. Wood, Y. A. Liu and Yueying Yu. *Design, Simulation, and Optimization of Adsorptive and Chromatographic Separations: A Hands-On Approach*. Wiley-VCH Verlag GmbH and Co. KGaA, 2018.

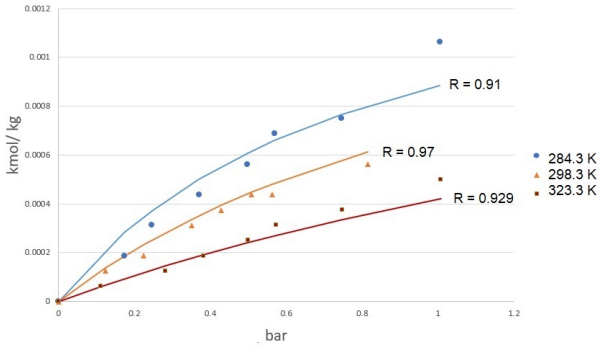
[22] *Vacuum pump VacuuBrand MZ1C*. URL: https://shop.vacuubrand.com/en/chemistry-diaphragm-pump-mz-1c-20724100.html#description.

[23] *GEKKO Optimization Suite documentation*. URL: https://gekko.readthedocs.io/en/latest/.

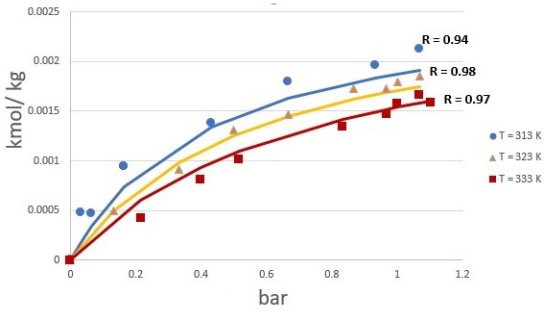
[24] Ines Duran, Fernando Rubiera and Covadonga Pevida. 'Modeling a biogas upgrading PSA unit with a sustainable activated carbon derived from pine sawdust. Sensitivity analysis on the adsorption of CO2 and CH4 mixtures'. In: *Chemical Engineering Journal* 428 (2022).

Appendix

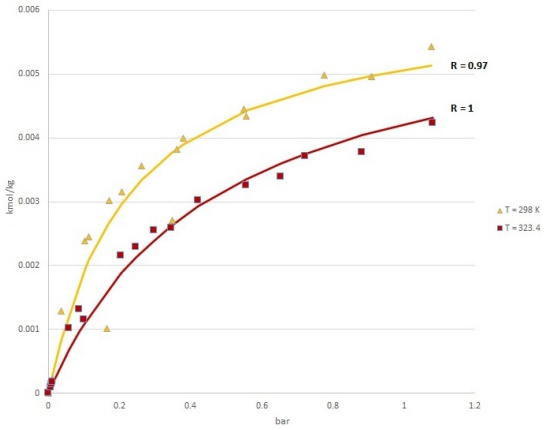
A Langmuir Isotherms of VOC



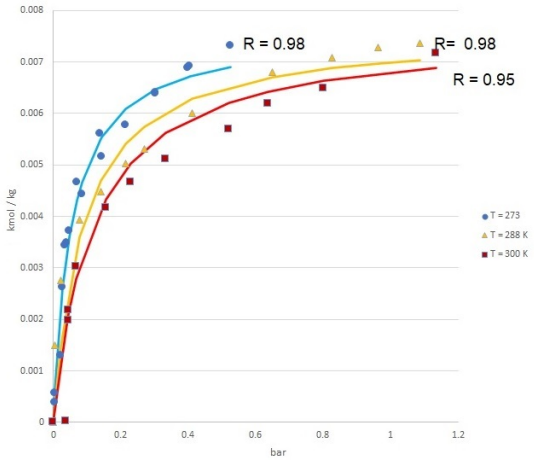
Methane Langmuir isotherm



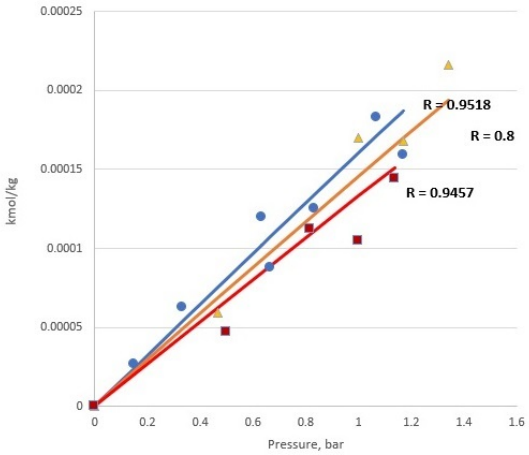
Ethane Langmuir isotherm



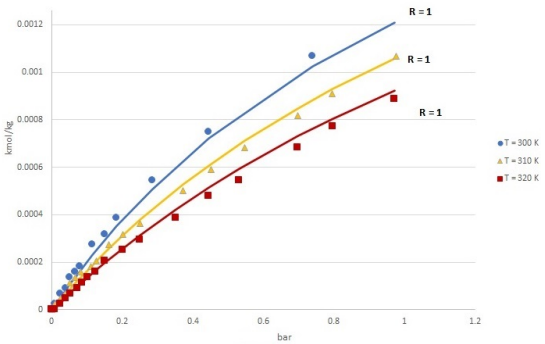
Propane Langmuir isotherm



CH_4 Butane Langmuir isotherm



Nitrogen Langmuir isotherm



CO_2 Langmuir isotherm

Figure 58: Langmuir isotherms of VOC

B Equinor VOCs experiment results for case 1

Time,s	Nitrogen, mmol/s	Methane, mmol/s	CO2 mmol/s	Ethane, mmol/s	Propane, mmol/s	Butane, mmol/s
3.70	0.36	0.02	0.00	0.00	0.00	0.00
50.70	0.36	0.05	0.00	0.00	0.00	0.00
99.00	0.36	0.08	0.01	0.00	0.00	0.00
145.60	0.36	0.09	0.02	0.00	0.00	0.00
196.00	0.36	0.08	0.05	0.00	0.00	0.00
255.90	0.36	0.08	0.07	0.00	0.00	0.00
293.90	0.36	0.08	0.07	0.00	0.00	0.00
345.50	0.36	0.08	0.06	0.05	0.01	0.00
399.20	0.37	0.08	0.05	0.18	0.02	0.00
448.50	0.36	0.08	0.05	0.20	0.02	0.00
504.10	0.36	0.08	0.05	0.21	0.02	0.01
549.70	0.36	0.07	0.05	0.22	0.02	0.01
588.30	0.36	0.07	0.05	0.22	0.03	0.01
653.70	0.36	0.07	0.05	0.23	0.03	0.01
693.50	0.36	0.07	0.05	0.22	0.03	0.01
747.00	0.36	0.08	0.05	0.22	0.05	0.01
798.40	0.36	0.07	0.05	0.20	0.09	0.01
851.60	0.36	0.07	0.04	0.17	0.14	0.02
891.40	0.36	0.07	0.04	0.17	0.15	0.02

C Equinor VOCs experiment results for case 2

Time,s	Nitrogen, mmol/s	Methane, mmol/s	CO2 mmol/s	Ethane, mmol/s	Propane, mmol/s	Butane, mmol/s
0	0.262	0.054	0.026	0.158	0.009	0.001
151.4	0.256	0.004	0.000	0.001	0.000	0.000
192.5	0.256	0.007	0.000	0.001	0.000	0.000
299.3	0.256	0.035	0.000	0.000	0.000	0.000
363.8	0.256	0.054	0.000	0.000	0.000	0.000
403.2	0.256	0.062	0.001	0.000	0.000	0.000
457.3	0.256	0.065	0.001	0.000	0.000	0.000
500.6	0.261	0.068	0.002	0.000	0.000	0.000
550.8	0.256	0.066	0.006	0.001	0.000	0.000
604.6	0.256	0.066	0.014	0.001	0.000	0.000
639.5	0.253	0.063	0.024	0.001	0.000	0.000
742	0.256	0.061	0.045	0.002	0.000	0.000
827.8	0.256	0.061	0.049	0.002	0.000	0.000
880.7	0.256	0.060	0.051	0.003	0.000	0.000

D Equinor VOCs experiment results for case 3

Time	Nitrogen, mmol/s	Methane, mmol/s	CO2 mmol/s	Ethane, mmol/s	Propane, mmol/s	Butane, mmol/s
0.9	0.00	0.00	0.00	0.00	0.00	0.00
55.1	0.00	0.00	0.00	0.00	0.00	0.00
103.3	0.15	0.02	0.03	0.03	0.05	0.01
156.6	0.15	0.01	0.01	0.00	0.01	0.00
216.8	0.15	0.00	0.00	0.00	0.00	0.00
257.4	0.15	0.00	0.00	0.00	0.00	0.00
351	0.15	0.00	0.00	0.00	0.00	0.00
414.7	0.15	0.01	0.00	0.00	0.00	0.00
505.1	0.15	0.02	0.00	0.00	0.00	0.00
565.8	0.15	0.03	0.00	0.00	0.00	0.00
642.5	0.15	0.03	0.00	0.00	0.00	0.00
691.3	0.15	0.04	0.00	0.00	0.00	0.00
752.9	0.15	0.04	0.00	0.00	0.00	0.00
800.2	0.15	0.04	0.00	0.00	0.00	0.00
853.1	0.15	0.04	0.00	0.00	0.00	0.00
896.3	0.15	0.04	0.00	0.00	0.00	0.00

E Main Code VOCs simulation

```
import openpyxl
import numpy as np
from gekko import GEKKO
import matplotlib.pyplot as plt

# Adsorber bed parameters
L                                   = 0.77       #Length of the bed,m
D                                   = 0.0328     #Diameter of the bed,m
ep                                  = 0.4       #Internal porosity
ei                                  = 0.45       #External porosity
et                                  = ep+ei      #Total porosity
V_tank                              = 6e-5       #"One side dead
rho                                 = 430        #Apparent(bulk) density, kg/m3
r0                                  = 2e-3       #Adsorbent pellet radius, m
shape_factor                        = 1

# Process parameters
pressure                            = 1          #Adsorption step
                                                 #pressure, bar
pressure_init                       = 0.06       #Initial pressure,
                                                 #bar
T                                   = 303        #Initial
                                                 #temperature, K


time_adsorption                     = 900        #sec
time_desorption                     = 600        #sec
time_purge                          = 300        #sec

# Case parameters
#Test 1
# Flow_rate                           = 9.28554E-07# Gas inlet flow rate, kmol/s
# Flow_rate_purge                     = 3.72E-08   # kmol/s
# y1_supply                           = 8.6/100    #CH4 feed
# y2_supply                           = 16.1/100   #C2H6 feed
# y3_supply                           = 22.4/100   #C3H8 feed
# y4_supply                           = 10/100     #C4H10 feed
# y5_supply                           = 4.7/100    #CO2 feed
```

```
# y6_supply                          = 38.2/100   #N2 feed
#Test 2
# Flow_rate                          = 6.33173E-07# Gas inlet flow rate, kmol/s
# Flow_rate_purge                    = 7.44034E-08   # kmol/s
# y1_supply                          = 8.3/100    #CH4 feed
# y2_supply                          = 16.6/100   #C2H6 feed
# y3_supply                          = 20.8/100   #C3H8 feed
# y4_supply                          = 9.6/100     #C4H10 feed
# y5_supply                          = 4.0/100    #CO2 feed
# y6_supply                          = 40.7/100   #N2 feed

#Test 3
Flow_rate                          = 3.27375E-07# Gas inlet flow rate, kmol/s
Flow_rate_purge                    = 1.86008E-08   # kmol/s
y1_supply                          = 11.7/100    #CH4 feed
y2_supply                          = 8.7/100   #C2H6 feed
y3_supply                          = 15.1/100   #C3H8 feed
y4_supply                          = 5.8/100     #C4H10 feed
y5_supply                          = 8.7/100    #CO2 feed
y6_supply                          = 50/100   #N2 feed

# Simulation parameters
seg                                = 100         #Number of space
                                                 #grid points
number_cycles                      = 10
# Adsorption isotherm
IP1_CH4                            = 0.0016     #kmol/kg
IP2_CH4                            = 0          #1/bar
IP3_CH4                            = 4.2E-05    #1/bar
IP4_CH4                            = 2922.78    #K

IP1_C2H6                           = 0.0027     #kmol/kg
IP2_C2H6                           = 0.0        #1/bar
IP3_C2H6                           = 2.66E-04   #1/bar
IP4_C2H6                           = 2833.77    # K

IP1_C3H8                           = 0.0062     #kmol/kg
IP2_C3H8                           = 0.0        #1/bar
IP3_C3H8                           = 3.75E-04   #1/bar
```

```
IP4_C3H8                                = 2795.28     #K

IP1_C4H10                               = 0.007       #kmol/kg
IP2_C4H10                               = 0.0         #1/bar
IP3_C4H10                               = 0.0015      #1/bar
IP4_C4H10                               = 2600        #K

IP1_CO2                                 = 0.0028      #kmol/kg
IP2_CO2                                 = 0.0         #(kmol/kg)/bar
IP3_CO2                                 = 0.000748    #1/bar
IP4_CO2                                 = 2084.44     #K

IP1_N2                                  = 0.0075      #kmol/kg
IP2_N2                                  = 0.0         #(kmol/kg)/bar
IP3_N2                                  = 0.00099     #1/bar
IP4_N2                                  = 935.77      #K

# Mass transfer
MTC                                     = [1,1,1,1,1,1]
                                                      #MTC = [CH4,C2H6,C3H8,
                                                      #C4H10,CO2,N2] s^-1
#Constants
R                                       = 8.314        #Universal gas constant, J/(mol K)
exp                                     = 2.7182



### Main simulation
#Permeability of the bed
B = 10**5*shape_factor**2*(2*r0)**2*ei**3/(150*(1-ei)**2)
K = 1/(B/(1e-5))
#Cross-sectional area of the adsorber
Area = np.pi * D**2/4
#Pressurization step, finding the time to pressurize the
#bed
for cycle_count in range(0,number_cycles):
    guess_time = 340 #sec
```

```
pressure_final = 0
while pressure_final < 1:
    m = GEKKO(remote = False)  # Create GEKKO model
                               # for adsorption step
    print(f"--Cycle number {cycle_count+1}--")
    y1_feed = y1_supply
    y2_feed = y2_supply
    y3_feed = y3_supply
    y4_feed = y4_supply
    y5_feed = y5_supply
    y6_feed = y6_supply

    Q1 = IP1_CH4 - IP2_CH4*T # Isotherm max capacity CH4
    Q2 = IP1_C2H6 - IP2_C2H6*T # Isotherm max capacity C2H6
    Q3 = IP1_C3H8 - IP2_C3H8*T # Isotherm max capacity C3H8
    Q4 = IP1_C4H10 - IP2_C4H10*T # Isotherm max capacity C4H10
    Q5 = IP1_CO2 - IP2_CO2*T # Isotherm max capacity CO2
    Q6 = IP1_N2 - IP2_N2*T  # Isotherm max capacity N2


    b1 = IP3_CH4*exp**(IP4_CH4/T) # Isotherm affinity coeff. CH4
    b2 = IP3_C2H6*exp**(IP4_C2H6/T) # Isotherm affinity coeff. C2H6
    b3 = IP3_C3H8*exp**(IP4_C3H8/T) # Isotherm affinity coeff. C3H8
    b4 = IP3_C4H10*exp**(IP4_C4H10/T) # Isotherm affinity coeff. C4H10
    b5 = IP3_CO2*exp**(IP4_CO2/T) # Isotherm affinity coeff. CO2
    b6 = IP3_N2*exp**(IP4_N2/T) # Isotherm affinity coeff. N2

    L_seg = L/seg # Grid step size
    if cycle_count == 0:
          # Introduction of pressure variable.
          #"seg" - number of grid
          # points. "pressure_init"
          #is set for every
          #grid point as an initial condition.
        P = [m.Var(pressure_init) for i in range(seg)]
        # Initial molar fraction
        y1 =  [m.Var(0) for i in range(seg)]

        y2 =  [m.Var(0) for i in range(seg)]
```

```
        y3 =  [m.Var(0) for i in range(seg)]

        y4 =  [m.Var(0) for i in range(seg)]

        y5 =  [m.Var(0) for i in range(seg)]

        y6 =  [m.Var(1) for i in range(seg)]

        v =  [m.Var(0) for i in range(seg)]

        # Initial adsorption capacity for components.
        q1 = [m.Var(0) for i in range(seg)]
        q2 = [m.Var(0) for i in range(seg)]
        q3 = [m.Var(0) for i in range(seg)]
        q4 = [m.Var(0) for i in range(seg)]
        q5 = [m.Var(0) for i in range(seg)]
        q6 = [m.Var((Q6*b6*y6[i].value*P[i].value)
                    /(1+b6*y6[i].value*P[i].value)) for i in range(seg)]
        P1 = m.Var(pressure_init) # Boundary left pressure
        P2 = m.Var(pressure_init) # Boundary right pressure

    else:
        #After purge step
        P = [m.Var(pressure_after_purge[-1][-1]) for i in range(seg)]
        y1 =  [m.Var(methane_after_purge[-1][i]) for i in range(seg)]
        y2 =  [m.Var(ethane_after_purge[-1][i]) for i in range(seg)]
        y3 =  [m.Var(propane_after_purge[-1][i]) for i in range(seg)]
        y4 =  [m.Var(butane_after_purge[-1][i]) for i in range(seg)]
        y5 =  [m.Var(co2_after_purge[-1][i]) for i in range(seg)]
        y6 =  [m.Var(n2_after_purge[-1][i]) for i in range(seg)]

        v =  [m.Var(0) for i in range(seg)]
        q1 = [m.Var(q1_after_purge[-1][i]) for i in range(seg)]
        q2 = [m.Var(q2_after_purge[-1][i]) for i in range(seg)]
        q3 = [m.Var(q3_after_purge[-1][i]) for i in range(seg)]
        q4 = [m.Var(q4_after_purge[-1][i]) for i in range(seg)]
        q5 = [m.Var(q5_after_purge[-1][i]) for i in range(seg)]
        q6 = [m.Var(q6_after_purge[-1][i]) for i in range(seg)]
```

```
    P1 = m.Var(pressure_after_purge[-1][-1])
    P2 = m.Var(pressure_after_purge[-1][-1])

#Time discretization
tf = guess_time
nt = int(tf/1) + 1
m.time = np.linspace(0,tf,nt)

#Left boundary pressure
m.Equation(P1.dt() ==  (P[0]/(V_tank))*(Flow_rate*10**(-2)*R*T
                        /(P[0])-v[0]*Area))
#Right boundary pressure
m.Equation(P2.dt() ==  (P2/(V_tank))*(v[seg-1]*Area))

# First segment component 1
m.Equation(P[0]*y1[0].dt()+y1[0]*P[0].dt() == (1/(K*et))*
           ((y1[0]*P[0]*(P[1]-2*P[0]+P1)/(L_seg**2))
            + P[0]*((P[0]-P1)/(L_seg))*((y1[0]-y1_feed)/(L_seg))
            + y1[0]*((P[0]-P1)/(L_seg))**2)
           - ((1/(et*100))*(R*T)*rho*q1[0].dt()))

#Middle segments component 1
m.Equations([P[i]*y1[i].dt()+y1[i]*P[i].dt() ==  \
            (1/(K*et))*((y1[i]*P[i]*(P[i+1]
            -2*P[i]+P[i-1])/(L_seg**2))
            + P[i]*((P[i]-P[i-1])/(L_seg))*((y1[i]-y1[i-1])/(L_seg))
            + y1[i]*((P[i]-P[i-1])/(L_seg))**2)
             - ((1/(et*100))*
            (R*T)*rho*q1[i].dt()) for i in range(1,seg-1)])
#Last segment component 1
m.Equation(P[seg-1]*y1[seg-1].dt()+y1[seg-1]*P[seg-1].dt() == \
            (1/(K*et))*((y1[seg-1]*P[seg-1]*(P2
            -2*P[seg-1]+P[seg-2])/(L_seg**2))
            + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))*((y1[seg-1]
            -y1[seg-2])/(L_seg))
            + y1[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
           - ((1/(et*100))*(R*T)*rho*q1[seg-1].dt()))
```

```
# First segment component 2
m.Equation(P[0]*y2[0].dt()+y2[0]*P[0].dt() == (1/(K*et))
            *((y2[0]*P[0]*(P[1]
            -2*P[0]+P1)/(L_seg**2))
            + P[0]*((P[0]-P1)/(L_seg))*((y2[0]
            -y2_feed)/(L_seg))
            + y2[0]*((P[0]-P1)/(L_seg))**2)
            - ((1/(et*100))*(R*T)*rho*q2[0].dt()))

# Middle segments component 2
m.Equations([P[i]*y2[i].dt()+y2[i]*P[i].dt() ==  \
            (1/(K*et))*((y2[i]*P[i]*(P[i+1]-2*P[i]
            +P[i-1])/(L_seg**2))
            + P[i]*((P[i]-P[i-1])/(L_seg))*((y2[i]
            -y2[i-1])/(L_seg))
            + y2[i]*((P[i]-P[i-1])/(L_seg))**2)
            - ((1/(et*100))*(R*T)*rho*q2[i].dt())
             for i in range(1,seg-1)])

# Last segment component 2
m.Equation(P[seg-1]*y2[seg-1].dt()+y2[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y2[seg-1]*P[seg-1]*(P2-2*P[seg-1]
        +P[seg-2])/(L_seg**2))
        + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))*((y2[seg-1]
        -y2[seg-2])/(L_seg))
        + y2[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q2[seg-1].dt()))

# First segment component 3
m.Equation(P[0]*y3[0].dt()+y3[0]*P[0].dt() == (1/(K*et))
        *((y3[0]*P[0]*(P[1]
       -2*P[0]+P1)/(L_seg**2))
        + P[0]*((P[0]-P1)/(L_seg))*((y3[0]-y3_feed)/(L_seg))
        + y3[0]*((P[0]-P1)/(L_seg))**2) - ((1/(et*100))*(R*T)
        *rho*q3[0].dt()))

# Middle segments component 3
m.Equations([P[i]*y3[i].dt()+y3[i]*P[i].dt() ==  \
            (1/(K*et))*((y3[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])
```

```
            /(L_seg**2))
            + P[i]*((P[i]-P[i-1])/(L_seg))*((y3[i]-y3[i-1])
            /(L_seg))
            + y3[i]*((P[i]-P[i-1])/(L_seg))**2)
            - ((1/(et*100))*(R*T)*rho*q3[i].dt())
             for i in range(1,seg-1)])

# Last segment component 3
m.Equation(P[seg-1]*y3[seg-1].dt()+y3[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y3[seg-1]*P[seg-1]*(P2
        -2*P[seg-1]+P[seg-2])/(L_seg**2))
        + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))*((y3[seg-1]
        -y3[seg-2])/(L_seg))
        + y3[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q3[seg-1].dt()))


# First segment component 4
m.Equation(P[0]*y4[0].dt()+y4[0]*P[0].dt() == (1/(K*et))*
        ((y4[0]*P[0]*(P[1]
        -2*P[0]+P1)/(L_seg**2))
        + P[0]*((P[0]-P1)/(L_seg))*((y4[0]-y4_feed)/(L_seg))
        + y4[0]*((P[0]-P1)/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q4[0].dt()))

# Middle segments component 4
m.Equations([P[i]*y4[i].dt()+y4[i]*P[i].dt() ==  \
            (1/(K*et))*((y4[i]*P[i]*(P[i+1]
            -2*P[i]+P[i-1])/(L_seg**2))
            + P[i]*((P[i]-P[i-1])/(L_seg))*((y4[i]
            -y4[i-1])/(L_seg))
            + y4[i]*((P[i]-P[i-1])/(L_seg))**2)
             - ((1/(et*100))*(R*T)*rho*q4[i].dt())
            for i in range(1,seg-1)])

# Last segment component 4
m.Equation(P[seg-1]*y4[seg-1].dt()+y4[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y4[seg-1]*P[seg-1]*(P2
        -2*P[seg-1]+P[seg-2])/(L_seg**2))
```

```
        + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))
        *((y4[seg-1]-y4[seg-2])/(L_seg))
        + y4[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q4[seg-1].dt()))

#First segment component 5
m.Equation(P[0]*y5[0].dt()+y5[0]*P[0].dt() ==
          (1/(K*et))*((y5[0]*P[0]*(P[1]
          -2*P[0]+P1)/(L_seg**2))
           + P[0]*((P[0]-P1)/(L_seg))*((y5[0]-y5_feed)/(L_seg))
           + y5[0]*((P[0]-P1)/(L_seg))**2)
          - ((1/(et*100))*(R*T)*rho*q5[0].dt()))

#Middle segments component 5
m.Equations([P[i]*y5[i].dt()+y5[i]*P[i].dt() ==  \
           (1/(K*et))*((y5[i]*P[i]*(P[i+1]
           -2*P[i]+P[i-1])/(L_seg**2))
           + P[i]*((P[i]-P[i-1])/(L_seg))*((y5[i]
           -y5[i-1])/(L_seg)) + y5[i]*((P[i]
           -P[i-1])/(L_seg))**2)
           - ((1/(et*100))*
           (R*T)*rho*q5[i].dt()) for i in range(1,seg-1)])

#Last segment component 5
m.Equation(P[seg-1]*y5[seg-1].dt()+y5[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y5[seg-1]*P[seg-1]*(P2-2*P[seg-1]
        +P[seg-2])/(L_seg**2))
        + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))
        *((y5[seg-1]-y5[seg-2])/(L_seg))
        + y5[seg-1]*((P[seg-1]
        -P[seg-2])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q5[seg-1].dt()))

# First segment component 6
m.Equation(P[0]*y6[0].dt()+y6[0]*P[0].dt() ==
          (1/(K*et))*((y6[0]*P[0]*(P[1]-2*P[0]+P1)/(L_seg**2))
           + P[0]*((P[0]-P1)/(L_seg))*((y6[0]-y6_feed)/(L_seg))
           + y6[0]*((P[0]-P1)/(L_seg))**2)
           - ((1/(et*100))*(R*T)*rho*q6[0].dt()))
```

```
# Middle segments component 6
m.Equations([P[i]*y6[i].dt()+y6[i]*P[i].dt() ==  \
            (1/(K*et))*((y6[i]*P[i]*(P[i+1]
            -2*P[i]+P[i-1])/(L_seg**2))
            + P[i]*((P[i]-P[i-1])/(L_seg))*((y6[i]-y6[i-1])/(L_seg))
            + y6[i]*((P[i]-P[i-1])/(L_seg))**2)
            - ((1/(et*100))*(R*T)*rho*q6[i].dt())
             for i in range(1,seg-1)])

# Last segment component 6
m.Equation(P[seg-1]*y6[seg-1].dt()+y6[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y6[seg-1]*P[seg-1]*(P2-2*P[seg-1]
        +P[seg-2])/(L_seg**2))
        + P[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))*((y6[seg-1]
        -y6[seg-2])/(L_seg))
        + y6[seg-1]*((P[seg-1]-P[seg-2])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q6[seg-1].dt()))

#Velocity calculation
m.Equation(v[0] == -(1/K)*(P[1]-P[0])/(L_seg))
m.Equation([v[i] == -(1/K)*(P[i+1]-P[i])/(L_seg)
           for i in range(1,seg-1)])
m.Equation(v[seg-1] == -(1/K)*(P2-P[seg-1])/(L_seg))
# Sum of all component molar fraction == 1
m.Equation([y1[i] +  y2[i] + y3[i] + y4[i] + y5[i] + y6[i] == 1
            for i in range(0,seg)])
#Solid mass balance
m.Equation([q1[i].dt() == - MTC[0]*(q1[i] -
(Q1*b1*y1[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
+b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
for i in range(0,seg)])
m.Equation([q2[i].dt() == - MTC[1]*(q2[i]
- (Q2*b2*y2[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
+b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
for i in range(0,seg)])
m.Equation([q3[i].dt() == - MTC[2]*(q3[i]
- (Q3*b3*y3[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
+b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
```

```
    for i in range(0,seg)])
    m.Equation([q4[i].dt() == - MTC[3]*(q4[i]
    - (Q4*b4*y4[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
    +b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
    for i in range(0,seg)])
    m.Equation([q5[i].dt() == - MTC[4]*(q5[i]
    - (Q5*b5*y5[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
    +b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
    for i in range(0,seg)])
    m.Equation([q6[i].dt() == - MTC[5]*(q6[i]
    - (Q6*b6*y6[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]
    +b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i])))
    for i in range(0,seg)])

    print("Start Pressurization")
    m.options.IMODE = 7
    m.solve(disp = False)
    print("Finished pressurization")
    pressure_final = P1[-1]
    guess_time = guess_time + 10

# For plotting

methane_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    methane_after_press[i] = np.array(y1[i].value)
methane_after_press= methane_after_press.T

ethane_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    ethane_after_press[i] = np.array(y2[i].value)
ethane_after_press= ethane_after_press.T

propane_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    propane_after_press[i] = np.array(y3[i].value)
propane_after_press= propane_after_press.T

butane_after_press = np.empty((seg,len(m.time)))
```

```
for i in range(seg):
    butane_after_press[i] = np.array(y4[i].value)
butane_after_press= butane_after_press.T

co2_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    co2_after_press[i] = np.array(y5[i].value)
co2_after_press= co2_after_press.T


n2_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    n2_after_press[i] = np.array(y6[i].value)
n2_after_press= n2_after_press.T

q1_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q1_after_press[i] = np.array(q1[i].value)
q1_after_press= q1_after_press.T

q2_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q2_after_press[i] = np.array(q2[i].value)
q2_after_press= q2_after_press.T

q3_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q3_after_press[i] = np.array(q3[i].value)
q3_after_press= q3_after_press.T

q4_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q4_after_press[i] = np.array(q4[i].value)
q4_after_press= q4_after_press.T

q5_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q5_after_press[i] = np.array(q5[i].value)
q5_after_press= q5_after_press.T
```

```
q6_after_press = np.empty((seg,len(m.time)))
for i in range(seg):
    q6_after_press[i] = np.array(q6[i].value)
q6_after_press= q6_after_press.T

#Write the results to Excel file
if cycle_count == number_cycles - 1:

    velocity_out = np.array(v[seg-2])
    pressure_out = np.array(P[seg-2])
    methane_out = np.array(y1[seg-2])
    ethane_out = np.array(y2[seg-2])
    propane_out = np.array(y3[seg-2])
    butane_out = np.array(y4[seg-2])
    co2_out = np.array(y5[seg-2])
    n2_out = np.array(y6[seg-2])

    flow_methane =  [0,0]
    flow_ethane =  [0,0]
    flow_propane =  [0,0]
    flow_butane =  [0,0]
    flow_co2 = [0,0]
    flow_n2 =  [0,0]

    old_indices = np.arange(0,len(flow_methane))
    new_length = 100
    new_indices = np.linspace(0,len(flow_methane)-1,new_length)
    spl = UnivariateSpline(old_indices,flow_methane,k=1,s=0)
    flow_methane  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(flow_ethane))
    new_length = 100
    new_indices = np.linspace(0,len(flow_ethane)-1,new_length)
    spl = UnivariateSpline(old_indices,flow_ethane,k=1,s=0)
    flow_ethane  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(flow_propane))
    new_length = 100
```

```
new_indices = np.linspace(0,len(flow_propane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_propane,k=1,s=0)
flow_propane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_butane))
new_length = 100
new_indices = np.linspace(0,len(flow_butane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_butane,k=1,s=0)
flow_butane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_co2))
new_length = 100
new_indices = np.linspace(0,len(flow_co2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_co2,k=1,s=0)
flow_co2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_n2))
new_length = 100
new_indices = np.linspace(0,len(flow_n2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_n2,k=1,s=0)
flow_n2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(pressure_out))
new_length = 100
new_indices = np.linspace(0,len(pressure_out)-1,new_length)
spl = UnivariateSpline(old_indices,pressure_out,k=1,s=0)
pressure_out  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(m.time))
new_length = 100
new_indices = np.linspace(0,len(m.time)-1,new_length)
spl = UnivariateSpline(old_indices,m.time,k=1,s=0)
time  = np.abs(spl(new_indices))

np_array_rows = (time,pressure_out,flow_methane,flow_ethane,
flow_propane,flow_butane,flow_co2,flow_n2)

with open('results_test.csv','a') as csvfile:
    np.savetxt(csvfile, np_array_rows , delimiter=',',
```

```
        header='Press',fmt='%s', comments='')

#Adsorption step
print("Adsorption step started")
m = GEKKO(remote = False)  # create GEKKO model for ads. step
#Time discretization for the model for adsorption step
tf =  time_adsorption
nt = int(tf/10) + 1
m.time = np.linspace(0,tf,nt)
#Inlet molar fractions
y1_feed = y1_supply
y2_feed = y2_supply
y3_feed = y3_supply
y4_feed = y4_supply
y5_feed = y5_supply
y6_feed = y6_supply
# The length of a segment
L_seg = L/seg
# Variables formulation
P = [m.Var(pressure) for i in range(seg)]
y1 =  [m.Var(methane_after_press[-1][i]) for i in range(seg)]
y2 =  [m.Var(ethane_after_press[-1][i]) for i in range(seg)]
y3 =  [m.Var(propane_after_press[-1][i]) for i in range(seg)]
y4 =  [m.Var(butane_after_press[-1][i]) for i in range(seg)]
y5 =  [m.Var(co2_after_press[-1][i]) for i in range(seg)]
y6 =  [m.Var(n2_after_press[-1][i]) for i in range(seg)]
u_initial = Flow_rate*R*T/(Area*pressure*10**2)
v =  [m.Var(0) for i in range(seg)]
q1 = [m.Var(q1_after_press[-1][i]) for i in range(seg)]
q2 = [m.Var(q2_after_press[-1][i]) for i in range(seg)]
q3 = [m.Var(q3_after_press[-1][i]) for i in range(seg)]
q4 = [m.Var(q4_after_press[-1][i]) for i in range(seg)]
q5 = [m.Var(q5_after_press[-1][i]) for i in range(seg)]
q6 = [m.Var(q6_after_press[-1][i]) for i in range(seg)]

# First segment component 1
m.Equation(P[0]*y1[0].dt()+y1[0]*P[0].dt() ==
          - (u_initial/(et))*(y1[0]*(P[0]-pressure)/(L_seg)
            +P[0]*(y1[0]-y1_feed)/(L_seg))
```

```
            - (P[0]*y1[0]/(et))*((v[0]-u_initial)/(L_seg))
            - (rho/(10**2*et))*R*T*q1[0].dt())

#Middle segments component 1
m.Equation([P[i]*y1[i].dt()+y1[i]*P[i].dt() == -
             (v[i]/(et))*(y1[i]*(P[i]-P[i-1])/(L_seg)
             +P[i]*(y1[i]-y1[i-1])/(L_seg))
            - (P[i]*y1[i]/(et))*((v[i]-v[i-1])/(L_seg))
             - (rho/(10**2*et))*R*T*q1[i].dt()
             for i in range(1,seg-1)])

# Last segment component 1
m.Equation(P[seg-1]*y1[seg-1].dt()+y1[seg-1]*P[seg-1].dt() ==
            - (v[seg-1]/(et))*(y1[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
            + P[seg-1]*(y1[seg-1]-y1[seg-2])/(L_seg))
            - (P[seg-1]*y1[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
            - (rho/(10**2*et))*R*T*q1[seg-1].dt())

#First segment component 2
m.Equation(P[0]*y2[0].dt()+y2[0]*P[0].dt() ==
            - (u_initial/(et))*(y2[0]*(P[0]-pressure)/(L_seg)
            + P[0]*(y2[0]-y2_feed)/(L_seg))
            - (P[0]*y2[0]/(et))*((v[0]-u_initial)/(L_seg))
            - (rho/(10**2*et))*R*T*q2[0].dt())

#Middle segments component 2
m.Equation([P[i]*y2[i].dt()+y2[i]*P[i].dt() ==
             - (v[i]/(et))*(y2[i]*(P[i]-P[i-1])/(L_seg)
             + P[i]*(y2[i]-y2[i-1])/(L_seg))
            - (P[i]*y2[i]/(et))*((v[i]-v[i-1])/(L_seg))
             - (rho/(10**2*et))*R*T*q2[i].dt() for i in range(1,seg-1)])

#Last segment component 2
m.Equation(P[seg-1]*y2[seg-1].dt()+y2[seg-1]*P[seg-1].dt() ==
            - (v[seg-1]/(et))*(y2[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
            + P[seg-1]*(y2[seg-1]-y2[seg-2])/(L_seg))
            - (P[seg-1]**y2[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
            - (rho/(10**2*et))*R*T*q2[seg-1].dt())
```

```
#First segment component 3
m.Equation(P[0]*y3[0].dt()+y3[0]*P[0].dt() ==
          - (u_initial/(et))*(y3[0]*(P[0]-pressure)/(L_seg)
          + P[0]*(y3[0]-y3_feed)/(L_seg))
          - (P[0]*y3[0]/(et))*((v[0]-u_initial)/(L_seg))
          - (rho/(10**2*et))*R*T*q3[0].dt())

#Middle segments component 3
m.Equation([P[i]*y3[i].dt()+y3[i]*P[i].dt() ==
            - (v[i]/(et))*(y3[i]*(P[i]-P[i-1])/(L_seg)
            +P[i]*(y3[i]-y3[i-1])/(L_seg))
           - (P[i]*y3[i]/(et))*((v[i]-v[i-1])/(L_seg))
            - (rho/(10**2*et))*R*T*q3[i].dt() for i in range(1,seg-1)])

#Last segment component 3
m.Equation(P[seg-1]*y3[seg-1].dt()+y3[seg-1]*P[seg-1].dt() ==
           - (v[seg-1]/(et))*(y3[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
           + P[seg-1]*y3[seg-1]*(y3[seg-1]-y3[seg-2])/(L_seg) )
           - (P[seg-1]*y3[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
           - (rho/(10**2*et))*R*T*q3[seg-1].dt())

#First segment component 4
m.Equation(P[0]*y4[0].dt()+y4[0]*P[0].dt() ==
           - (u_initial/(et))*(y4[0]*(P[0]-pressure)/(L_seg)
            +P[0]*(y4[0]-y4_feed)/(L_seg))
           - (P[0]*y4[0]/(et))*((v[0]-u_initial)/(L_seg))
            - (rho/(10**2*et))*R*T*q4[0].dt())

#Middle segments component 4
m.Equation([P[i]*y4[i].dt()+y4[i]*P[i].dt() ==
            - (v[i]/(et))*(y4[i]*(P[i]-P[i-1])/(L_seg)
            +P[i]*(y4[i]-y4[i-1])/(L_seg))
           - (P[i]*y4[i]/(et))*((v[i]-v[i-1])/(L_seg))
            - (rho/(10**2*et))*R*T*q4[i].dt()
            for i in range(1,seg-1)])

#Last segment component 4
m.Equation(P[seg-1]*y4[seg-1].dt()+y4[seg-1]*P[seg-1].dt() ==
           - (v[seg-1]/(et))*(y4[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
```

```
            +P[seg-1]*y4[seg-1]*(y4[seg-1]-y4[seg-2])/(L_seg))
           - (P[seg-1]*y4[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
           - (rho/(10**2*et))*R*T*q4[seg-1].dt())

#First segment component 5
m.Equation(P[0]*y5[0].dt()+y5[0]*P[0].dt() ==
           - (u_initial/(et))*(y5[0]*(P[0]-pressure)/(L_seg)
            +P[0]*(y5[0]-y5_feed)/(L_seg))
           - (P[0]*y5[0]/(et))*((v[0]-u_initial)/(L_seg))
           - (rho/(10**2*et))*R*T*q5[0].dt())
#Middle segments component 5
m.Equation([P[i]*y5[i].dt()+y5[i]*P[i].dt() ==
            - (v[i]/(et))*(y4[i]*(P[i]-P[i-1])/(L_seg)
            +P[i]*(y5[i]-y5[i-1])/(L_seg))
           - (P[i]*y5[i]/(et))*((v[i]-v[i-1])/(L_seg))
            - (rho/(10**2*et))*R*T*q5[i].dt() for i in range(1,seg-1)])
#Last segment component 5
m.Equation(P[seg-1]*y5[seg-1].dt()+y5[seg-1]*P[seg-1].dt() ==
           - (v[seg-1]/(et))*(y5[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
           + P[seg-1]*y5[seg-1]*(y5[seg-1]-y5[seg-2])/(L_seg))
           - (P[seg-1]*y5[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
           - (rho/(10**2*et))*R*T*q5[seg-1].dt())

#First segment component 6
m.Equation(P[0]*y6[0].dt()+y6[0]*P[0].dt() ==
           - (u_initial/(et))*(y6[0]*(P[0]-pressure)/(L_seg)
           + P[0]*(y6[0]-y6_feed)/(L_seg))
           - (P[0]*y6[0]/(et))*((v[0]-u_initial)/(L_seg))
           - (rho/(10**2*et))*R*T*q6[0].dt())
#Middle segments component 6
m.Equation([P[i]*y6[i].dt()+y6[i]*P[i].dt() ==
            - (v[i]/(et))*(y6[i]*(P[i]-P[i-1])/(L_seg)
            + P[i]*(y6[i]-y6[i-1])/(L_seg))
           - (P[i]*y6[i]/(et))*((v[i]-v[i-1])/(L_seg))
            - (rho/(10**2*et))*R*T*q6[i].dt() for i in range(1,seg-1)])
#Last segment component 6
m.Equation(P[seg-1]*y6[seg-1].dt()+y6[seg-1]*P[seg-1].dt() ==
           - (v[seg-1]/(et))*(y6[seg-1]*(P[seg-1]-P[seg-2])/(L_seg)
           + P[seg-1]*y6[seg-1]*(y6[seg-1]-y6[seg-2])/(L_seg))
```

```
            - (P[seg-1]*y6[seg-1]/(et))*((v[seg-1]-v[seg-2])/(L_seg))
            - (rho/(10**2*et))*R*T*q6[seg-1].dt())


#Velocity calculation
m.Equation(v[0] == -(1/K)*(P[0]-pressure)/(L_seg))
m.Equation([v[i] == -(1/K)*(P[i]-P[i-1])/(L_seg)
            for i in range(1,seg-1)])
m.Equation(v[seg-1] == -(1/K)*(P[seg-1]-P[seg-2])/(L_seg))

#Sum of molar fractions == 1
m.Equation([y1[i] +  y2[i] + y3[i] + y4[i] + y5[i] + y6[i] == 1
            for i in range(0,seg)])

#Solid mass balance
m.Equation([q1[i].dt() == -MTC[0]*(q1[i]
- (Q1*b1*y1[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q2[i].dt() == -MTC[1]*(q2[i]
- (Q2*b2*y2[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q3[i].dt() == -MTC[2]*(q3[i]
- (Q3*b3*y3[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q4[i].dt() == -MTC[3]*(q4[i]
- (Q4*b4*y4[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q5[i].dt() == -MTC[4]*(q5[i]
- (Q5*b5*y5[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q6[i].dt() == -MTC[5]*(q6[i]
- (Q6*b6*y6[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])

# Simulation
m.options.IMODE = 7
m.solve(disp = False)
print("Finished Adsorption Step")
```

```
# Write the results
if cycle_count == number_cycles - 1:

    velocity_out = np.array(v[seg-2])
    pressure_out = np.array(P[seg-2])
    methane_out = np.array(y1[seg-2])
    ethane_out = np.array(y2[seg-2])
    propane_out = np.array(y3[seg-2])
    butane_out = np.array(y4[seg-2])
    co2_out = np.array(y5[seg-2])
    n2_out = np.array(y6[seg-2])


    flow_methane =  (velocity_out*Area*methane_out*
                     pressure_out*10**8/(R*T))
    flow_ethane =  (velocity_out*Area*ethane_out*
                    pressure_out*10**8/(R*T))
    flow_propane =  (velocity_out*Area*propane_out*
                     pressure_out*10**8/(R*T))
    flow_butane =  (velocity_out*Area*butane_out*
                    pressure_out*10**8/(R*T))
    flow_co2 = (velocity_out*Area*co2_out*
                pressure_out*10**8/(R*T))
    flow_n2 =  (velocity_out*Area*n2_out*
                pressure_out*10**8/(R*T))

    old_indices = np.arange(0,len(flow_methane))
    new_length = 100
    new_indices = np.linspace(0,len(flow_methane)-1,new_length)
    spl = UnivariateSpline(old_indices,flow_methane,k=1,s=0)
    flow_methane  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(flow_ethane))
    new_length = 100
    new_indices = np.linspace(0,len(flow_ethane)-1,new_length)
    spl = UnivariateSpline(old_indices,flow_ethane,k=1,s=0)
    flow_ethane  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(flow_propane))
```

```
new_length = 100
new_indices = np.linspace(0,len(flow_propane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_propane,k=1,s=0)
flow_propane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_butane))
new_length = 100
new_indices = np.linspace(0,len(flow_butane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_butane,k=1,s=0)
flow_butane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_co2))
new_length = 100
new_indices = np.linspace(0,len(flow_co2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_co2,k=1,s=0)
flow_co2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_n2))
new_length = 100
new_indices = np.linspace(0,len(flow_n2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_n2,k=1,s=0)
flow_n2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(pressure_out))
new_length = 100
new_indices = np.linspace(0,len(pressure_out)-1,new_length)
spl = UnivariateSpline(old_indices,pressure_out,k=1,s=0)
pressure_out  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(m.time))
new_length = 100
new_indices = np.linspace(0,len(m.time)-1,new_length)
spl = UnivariateSpline(old_indices,m.time,k=1,s=0)
time  = np.abs(spl(new_indices))

np_array_rows = (time,pressure_out,flow_methane,flow_ethane,
                 flow_propane,flow_butane,flow_co2,flow_n2)

with open('results_test3.csv','a') as csvfile:
```

```
        np.savetxt(csvfile, np_array_rows , delimiter=',',
                   header='ADS',fmt='%s', comments='')

#Values for the next step
y1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y1_val[i] = np.array(y1[i].value)
y1_val[seg-1] = y1[seg-2]
y1_val = y1_val.T
methane_after_ads = y1_val

y2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y2_val[i] = np.array(y2[i].value)
y2_val[seg-1] = y2[seg-2]
y2_val = y2_val.T
ethane_after_ads = y2_val
y3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y3_val[i] = np.array(y3[i].value)
y3_val[seg-1] = y3[seg-2]
y3_val = y3_val.T
propane_after_ads = y3_val

y4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y4_val[i] = np.array(y4[i].value)
y4_val[seg-1] = y4[seg-2]
y4_val = y4_val.T
butane_after_ads = y4_val
y5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y5_val[i] = np.array(y5[i].value)
y5_val[seg-1] = y5[seg-2]
y5_val = y5_val.T
co2_after_ads = y5_val
y6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y6_val[i] = np.array(y6[i].value)
```

```
y6_val[seg-1] = y6[seg-2]
y6_val = y6_val.T
n2_after_ads = y6_val

q1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q1_val[i] = np.array(q1[i].value)
q1_val[seg-1] = q1[seg-2]
q1_val = q1_val.T
q1_after_ads = q1_val


q2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q2_val[i] = np.array(q2[i].value)
q2_val[seg-1] = q2[seg-2]
q2_val = q2_val.T
q2_after_ads = q2_val


q3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q3_val[i] = np.array(q3[i].value)
q3_val[seg-1] = q3[seg-2]
q3_val = q3_val.T
q3_after_ads = q3_val

q4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q4_val[i] = np.array(q4[i].value)
q4_val[seg-1] = q4[seg-2]
q4_val = q4_val.T
q4_after_ads = q4_val


q5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q5_val[i] = np.array(q5[i].value)
q5_val[seg-1] = q5[seg-2]
```

```
q5_val = q5_val.T
q5_after_ads = q5_val

q6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q6_val[i] = np.array(q6[i].value)
q6_val[seg-1] = q6[seg-2]
q6_val = q6_val.T
q6_after_ads = q6_val

#Segment length
L_seg = 1/seg

m = GEKKO(remote = False)  # create GEKKO model for des.step
#Time discretization
tf = time_desorption
nt = int(tf/10) + 1
m.time = np.linspace(0,tf,nt)

#Variables formulation
P = [m.Var(pressure) for i in range(seg)]
P1 = m.Var(pressure)
P2 = m.Var(pressure)
y1 =  [m.Var(methane_after_ads[-1][i]) for i in range(seg)]
y2 =  [m.Var(ethane_after_ads[-1][i]) for i in range(seg)]
y3 =  [m.Var(propane_after_ads[-1][i]) for i in range(seg)]
y4 =  [m.Var(butane_after_ads[-1][i]) for i in range(seg)]
y5 =  [m.Var(co2_after_ads[-1][i]) for i in range(seg)]
y6 =  [m.Var(n2_after_ads[-1][i]) for i in range(seg)]
v =  [m.Var(0) for i in range(seg)]
q1 = [m.Var(q1_after_ads[-1][i]) for i in range(seg)]
q2 = [m.Var(q2_after_ads[-1][i]) for i in range(seg)]
q3 = [m.Var(q3_after_ads[-1][i]) for i in range(seg)]
q4 = [m.Var(q4_after_ads[-1][i]) for i in range(seg)]
q5 = [m.Var(q5_after_ads[-1][i]) for i in range(seg)]
q6 = [m.Var(q6_after_ads[-1][i]) for i in range(seg)]
Q_pump   =  m.Var(0.8 / 3600)

#Boundary pressures
```

```
m.Equation(P2.dt() ==  (P2/(V_tank))*(v[seg-1]*Area))
m.Equation(P1.dt() == (-((P[0])*Q_pump)/(V_tank)
                       - (P[0]*v[0]*Area)/(V_tank)))

#Pump volumetric flow rate
m.Equation(Q_pump ==  0.0001974563*P[0]**(0.627027))

# First segment component 1
m.Equation(P[0]*y1[0].dt()+y1[0]*P[0].dt() ==
           (1/(K*et))*((y1[0]*P[0]*(P[1]
           - 2*P[0] + P1)/(L_seg**2))
           + P[1]*((P[1]-P[0])/(L_seg))*((y1[1]-y1[0])/(L_seg))
           + y1[1]*((P[1]-P[0])/(L_seg))**2)
           - ((1/(et*100))*(R*T)*rho*q1[0].dt()))

# Middle segments component 1
m.Equations([P[i]*y1[i].dt()+y1[i]*P[i].dt() ==  \
            (1/(K*et))*((y1[i]*P[i]*(P[i+1]
            -2*P[i]+P[i-1])/(L_seg**2))
            + P[i+1]*((P[i+1]-P[i])/(L_seg))*((y1[i+1]-y1[i])/(L_seg))
            + y1[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - (1/(et*100))*(R*T)*rho*q1[i].dt()
             for i in range(1,seg-1)])

# Last segment component 1
m.Equation(P[seg-1]*y1[seg-1].dt()+y1[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y1[seg-1]*P[seg-1]*(-P[seg-1]+P[seg-2])
        /(L_seg**2)))
        - ((1/(et*100))*(R*T)*rho*q1[seg-1].dt()))


# First segment component 2
m.Equation(P[0]*y2[0].dt()+y2[0]*P[0].dt() == (1/(K*et))*((y2[0]*P[0]
                *(P[1]- 2*P[0] + P1)/(L_seg**2)) + P[1]*((P[1]-P[0])
                /(L_seg))*((y2[1]-y2[0])/(L_seg))
                + y2[1]*((P[1]-P[0])/(L_seg))**2)
                - ((1/(et*100))*(R*T)*rho*q2[0].dt()))

# Middle segments component 2
```

```
m.Equations([P[i]*y2[i].dt()+y2[i]*P[i].dt() ==  \
            (1/(K*et))*((y2[i]*P[i]*(P[i+1]-2*P[i]
            +P[i-1])/(L_seg**2)) +  P[i+1]*((P[i+1]
            -P[i])/(L_seg))*((y2[i+1]-y2[i])/(L_seg))
            + y2[i+1]*((P[i+1]-P[i])/(L_seg))**2)
            - ((1/(et*100))*(R*T))*rho*q2[i].dt()
             for i in range(1,seg-1)])

# Last segment component 2
m.Equation(P[seg-1]*y2[seg-1].dt()+y2[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y2[seg-1]*P[seg-1]*(-P[seg-1]
        +P[seg-2])/(L_seg**2)))
        - ((1/(et*100))*(R*T))*rho*q2[seg-1].dt())


# First segment component 3
m.Equation(P[0]*y3[0].dt()+y3[0]*P[0].dt() ==
           (1/(K*et))*((y3[0]*P[0]*(P[1] - 2*P[0] + P1)/(L_seg**2))
           + P[1]*((P[1]-P[0])/(L_seg))*((y3[1]-y3[0])/(L_seg))
+ y3[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q3[0].dt()))

# Middle segments component 3
m.Equations([P[i]*y3[i].dt()+y3[i]*P[i].dt() ==  \
            (1/(K*et))*((y3[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y3[i+1]-y3[i])/(L_seg))
            + y3[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - ((1/(et*100))*(R*T))*rho*q3[i].dt()
             for i in range(1,seg-1)])

# Last segment component 3
m.Equation(P[seg-1]*y3[seg-1].dt()+y3[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y3[seg-1]*P[seg-1]*(-P[seg-1]+P[seg-2])
        /(L_seg**2))) - ((1/(et*100))*(R*T))*rho*q3[seg-1].dt())


# First segment component 4
m.Equation(P[0]*y4[0].dt()+y4[0]*P[0].dt() ==
           (1/(K*et))*((y4[0]*P[0]*(P[1] - 2*P[0] + P1)/(L_seg**2))
           + P[1]*((P[1]-P[0])/(L_seg))*((y4[1]-y4[0])/(L_seg))
```

```
+ y4[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q4[0].dt()))

# Middle segments component 4
m.Equations([P[i]*y4[i].dt()+y4[i]*P[i].dt() ==  \
            (1/(K*et))*((y4[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y4[i+1]-y4[i])/(L_seg))
            + y4[i+1]*((P[i+1]-P[i])/(L_seg))**2) \
            - ((1/(et*100))*(R*T))*rho*q4[i].dt()
             for i in range(1,seg-1)])

# Last segment component 4
m.Equation(P[seg-1]*y4[seg-1].dt()+y4[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y4[seg-1]*P[seg-1]*(-P[seg-1]+P[seg-2])/(L_seg**2)))
        - ((1/(et*100))*(R*T))*rho*q4[seg-1].dt())


# First segment component 5
m.Equation(P[0]*y5[0].dt()+y5[0]*P[0].dt() == (1/(K*et))
                  *((y5[0]*P[0]*(P[1]
                 - 2*P[0] + P1)/(L_seg**2))
                + P[1]*((P[1]-P[0])/(L_seg))*((y5[1]-y5[0])/(L_seg))
+ y5[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q5[0].dt()))

# Middle segments component 5
m.Equations([P[i]*y5[i].dt()+y5[i]*P[i].dt() ==  \
            (1/(K*et))*((y5[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y5[i+1]-y5[i])/(L_seg))
            + y5[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - ((1/(et*100))*(R*T))*rho*q5[i].dt()
             for i in range(1,seg-1)])

# Last segment component 5
m.Equation(P[seg-1]*y5[seg-1].dt()+y5[seg-1]*P[seg-1].dt() == \
            (1/(K*et))*((y5[seg-1]*P[seg-1]*(-P[seg-1]+P[seg-2])/(L_seg**2)))
           - ((1/(et*100))*(R*T))*rho*q5[seg-1].dt())


# First segment component 6
m.Equation(P[0]*y6[0].dt()+y6[0]*P[0].dt() == (1/(K*et))*((y6[0]*P[0]*(P[1]
```

```
            - 2*P[0] + P1)/(L_seg**2))
            + P[1]*((P[1]-P[0])/(L_seg))*((y6[1]-y6[0])/(L_seg))
            + y6[1]*((P[1]-P[0])/(L_seg))**2)
            - ((1/(et*100))*(R*T)*rho*q6[0].dt()))

# Middle segments component 6
m.Equations([P[i]*y6[i].dt()+y6[i]*P[i].dt() ==  \
            (1/(K*et))*((y6[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))
            *((y6[i+1]-y6[i])/(L_seg))
            + y6[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - ((1/(et*100))*(R*T))*rho*q6[i].dt()
             for i in range(1,seg-1)])

# Last segment component 6
m.Equation(P[seg-1]*y6[seg-1].dt()+y6[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y6[seg-1]*P[seg-1]*(-P[seg-1]
        +P[seg-2])/(L_seg**2)))
        - ((1/(et*100))*(R*T))*rho*q6[seg-1].dt())

m.Equation([q1[i].dt() == -MTC[0]*(q1[i]
- (Q1*b1*y1[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q2[i].dt() == -MTC[1]*(q2[i]
- (Q2*b2*y2[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q3[i].dt() == -MTC[2]*(q3[i]
- (Q3*b3*y3[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q4[i].dt() == -MTC[3]*(q4[i]
- (Q4*b4*y4[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q5[i].dt() == -MTC[4]*(q5[i]
- (Q5*b5*y5[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q6[i].dt() == -MTC[5]*(q6[i]
- (Q6*b6*y6[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
```

```
# Velocity calculation, m/s
m.Equation(v[0] == -(1/K)*(P[1]-P[0])/(L_seg))
m.Equation([v[i] == -(1/K)*(P[i+1]-P[i])/(L_seg)
for i in range(1,seg-1)])
m.Equation(v[seg-1] == -(1/K)*(P2 - P[seg-1])/(L_seg))

#Summ of molar fraction == 1
m.Equation([y1[i] +  y2[i] + y3[i] + y4[i] + y5[i] + y6[i] == 1
            for i in range(0,seg)])

print("Desorption step started")
# simulation
m.options.IMODE = 7
m.solve(disp=False)
print("Desorption step finished")

#Write the results
if cycle_count == number_cycles - 1:

    velocity_out = -1*np.array(v[0])
    pressure_out = np.array(P[0])
    methane_out = np.array(y1[0])
    ethane_out = np.array(y2[0])
    propane_out = np.array(y3[0])
    butane_out = np.array(y4[0])
    co2_out = np.array(y5[0])
    n2_out = np.array(y6[0])


    co2_out = np.array(y2[seg-1])

    flow_methane =  (velocity_out*Area*methane_out*
                     pressure_out*10**8/(R*T))
    flow_ethane =  (velocity_out*Area*ethane_out*
                    pressure_out*10**8/(R*T))
    flow_propane =  (velocity_out*Area*propane_out*
                     pressure_out*10**8/(R*T))
    flow_butane =  (velocity_out*Area*butane_out*
                    pressure_out*10**8/(R*T))
```

```
flow_co2 = (velocity_out*Area*co2_out*
            pressure_out*10**8/(R*T))
flow_n2 = (velocity_out*Area*n2_out*
           pressure_out*10**8/(R*T))

old_indices = np.arange(0,len(flow_methane))
new_length = 100
new_indices = np.linspace(0,len(flow_methane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_methane,k=1,s=0)
flow_methane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_ethane))
new_length = 100
new_indices = np.linspace(0,len(flow_ethane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_ethane,k=1,s=0)
flow_ethane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_propane))
new_length = 100
new_indices = np.linspace(0,len(flow_propane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_propane,k=1,s=0)
flow_propane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_butane))
new_length = 100
new_indices = np.linspace(0,len(flow_butane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_butane,k=1,s=0)
flow_butane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_co2))
new_length = 100
new_indices = np.linspace(0,len(flow_co2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_co2,k=1,s=0)
flow_co2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_n2))
new_length = 100
new_indices = np.linspace(0,len(flow_n2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_n2,k=1,s=0)
```

```
    flow_n2  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(pressure_out))
    new_length = 100
    new_indices = np.linspace(0,len(pressure_out)-1,new_length)
    spl = UnivariateSpline(old_indices,pressure_out,k=1,s=0)
    pressure_out  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(m.time))
    new_length = 100
    new_indices = np.linspace(0,len(m.time)-1,new_length)
    spl = UnivariateSpline(old_indices,m.time,k=1,s=0)
    time  = np.abs(spl(new_indices))

    np_array_rows = (time,pressure_out,flow_methane,
                     flow_ethane,flow_propane,flow_butane,flow_co2,flow_n2)

    with open('results_test3.csv','a') as csvfile:
        np.savetxt(csvfile, np_array_rows , delimiter=',',
                   header='DES',fmt='%s', comments='')

y1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y1_val[i] = np.array(y1[i].value)
y1_val[seg-1] = y1[seg-2]
y1_val = y1_val.T
methane_after_des = y1_val
y2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y2_val[i] = np.array(y2[i].value)
y2_val[seg-1] = y2[seg-2]
y2_val = y2_val.T
ethane_after_des = y2_val
y3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y3_val[i] = np.array(y3[i].value)
y3_val[seg-1] = y3[seg-2]
y3_val = y3_val.T
propane_after_des = y3_val
```

```
y4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y4_val[i] = np.array(y4[i].value)
y4_val[seg-1] = y4[seg-2]
y4_val = y4_val.T
butane_after_des = y4_val
y5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y5_val[i] = np.array(y5[i].value)
y5_val[seg-1] = y5[seg-2]
y5_val = y5_val.T
co2_after_des = y5_val
y6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y6_val[i] = np.array(y6[i].value)
y6_val[seg-1] = y6[seg-2]
y6_val = y6_val.T
n2_after_des = y6_val

q1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q1_val[i] = np.array(q1[i].value)
q1_val[seg-1] = q1[seg-2]
q1_val = q1_val.T
q1_after_des = q1_val

q2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q2_val[i] = np.array(q2[i].value)
q2_val[seg-1] = q2[seg-2]
q2_val = q2_val.T
q2_after_des = q2_val


q3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q3_val[i] = np.array(q3[i].value)
q3_val[seg-1] = q3[seg-2]
```

```
q3_val = q3_val.T
q3_after_des = q3_val

q4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q4_val[i] = np.array(q4[i].value)
q4_val[seg-1] = q4[seg-2]
q4_val = q4_val.T
q4_after_des = q4_val

q5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q5_val[i] = np.array(q5[i].value)
q5_val[seg-1] = q5[seg-2]
q5_val = q5_val.T
q5_after_des = q5_val

q6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q6_val[i] = np.array(q6[i].value)
q6_val[seg-1] = q6[seg-2]
q6_val = q6_val.T
q6_after_des = q6_val

vel_val = np.empty((seg,len(m.time)))
for i in range(seg):
    vel_val[i] = np.array(v[i].value)
vel_val = vel_val.T
vel_after_des = vel_val

pressure_val = np.empty((seg,len(m.time)))
for i in range(seg):
    pressure_val[i] = np.array(P[i].value)
pressure_val = pressure_val.T
pressure_after_des = pressure_val

print("Started purge step")
m = GEKKO(remote = False)  # create GEKKO model
#Time discretization
```

```
tf = time_purge
nt = int(tf/5) + 1
m.time = np.linspace(0,tf,nt)

#Molar fraction inlet only nitrogen
y1_feed = 0
y2_feed = 0
y3_feed = 0
y4_feed = 0
y5_feed = 0
y6_feed = 1

#Segment length
L_seg = L/seg

#Variables formulation
P = [m.Var(pressure_after_des[-1][i]) for i in range(seg)]
y1 =  [m.Var(methane_after_des[-1][i]) for i in range(seg)]
y2 =  [m.Var(ethane_after_des[-1][i]) for i in range(seg)]
y3 =  [m.Var(propane_after_des[-1][i]) for i in range(seg)]
y4 =  [m.Var(butane_after_des[-1][i]) for i in range(seg)]
y5 =  [m.Var(co2_after_des[-1][i]) for i in range(seg)]
y6 =  [m.Var(n2_after_des[-1][i]) for i in range(seg)]

v =  [m.Var(vel_after_des[-1][i]) for i in range(seg)]
q1 = [m.Var(q1_after_des[-1][i]) for i in range(seg)]
q2 = [m.Var(q2_after_des[-1][i]) for i in range(seg)]
q3 = [m.Var(q3_after_des[-1][i]) for i in range(seg)]
q4 = [m.Var(q4_after_des[-1][i]) for i in range(seg)]
q5 = [m.Var(q5_after_des[-1][i]) for i in range(seg)]
q6 = [m.Var(q6_after_des[-1][i]) for i in range(seg)]

P1 = m.Var(pressure_after_des[-1][0])
P2 = m.Var(pressure_after_des[-1][-1])

Q_pump   =  m.Var(0.6 / 3600)

m.Equation(Q_pump ==  0.0001974563*P[0]**(0.627027))
```

```
#Boundary pressures

m.Equation(P1.dt() == (-((P[0])*Q_pump)/(V_tank) - (P[0]*v[0]*Area)/(V_tank)))

m.Equation(P2.dt() ==  (P[seg-1]/(V_tank))*(Flow_rate_purge*10**(-2)*R*T/(P[seg-1]


#First segment component 1
m.Equation(P[0]*y1[0].dt()+y1[0]*P[0].dt() == (1/(K*et))*((y1[0]*P[0]*(P[1]
- 2*P[0] + P1)/(L_seg**2))
+ P[1]*((P[1]-P[0])/(L_seg))*((y1[1]-y1[0])/(L_seg))
+ y1[1]*((P[1]-P[0])/(L_seg))**2)
- ((1/(et*100))*(R*T)*rho*q1[0].dt()))

# Middle segments component 1
m.Equations([P[i]*y1[i].dt()+y1[i]*P[i].dt() ==  \
            (1/(K*et))*((y1[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            + P[i+1]*((P[i+1]-P[i])/(L_seg))*((y1[i+1]-y1[i])/(L_seg))
            + y1[i+1]*((P[i+1]-P[i])/(L_seg))**2)
            - (1/(et*100))*(R*T)*rho*q1[i].dt()
             for i in range(1,seg-1)])

# Last segments component 1
m.Equation(P[seg-1]*y1[seg-1].dt()+y1[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y1[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
        + (P2)*((P2-P[seg-1])/(L_seg))*((y1_feed-y1[seg-1])/(L_seg))
        + (y1_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q1[seg-1].dt())

# First segment component 2
m.Equation(P[0]*y2[0].dt()+y2[0]*P[0].dt() == (1/(K*et))*((y2[0]*P[0]*(P[1]
- 2*P[0] + P1)/(L_seg**2)) + P[1]*((P[1]-P[0])/(L_seg))*((y2[1]-y2[0])/(L_seg))
+ y2[1]*((P[1]-P[0])/(L_seg))**2)
- ((1/(et*100))*(R*T)*rho*q2[0].dt()))

# Middle segments component 2
m.Equations([P[i]*y2[i].dt()+y2[i]*P[i].dt() ==  \
            (1/(K*et))*((y2[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y2[i+1]-y2[i])/(L_seg))
```

```
            + y2[i+1]*((P[i+1]-P[i])/(L_seg))**2)
            - ((1/(et*100))*(R*T))*rho*q2[i].dt()
             for i in range(1,seg-1)])

# Last segments component  2
m.Equation(P[seg-1]*y2[seg-1].dt()+y2[seg-1]*P[seg-1].dt() == \
        (1/(K*et))*((y2[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
        + (P2)*((P2-P[seg-1])/(L_seg))*((y2_feed-y2[seg-1])/(L_seg))
        + (y2_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q2[seg-1].dt())

# First segment component 3
m.Equation(P[0]*y3[0].dt()+y3[0]*P[0].dt() ==
        (1/(K*et))*((y3[0]*P[0]*(P[1] - 2*P[0] + P1)/(L_seg**2))
        + P[1]*((P[1]-P[0])/(L_seg))*((y3[1]-y3[0])/(L_seg))
+ y3[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q3[0].dt()))

# Middle segments component 3
m.Equations([P[i]*y3[i].dt()+y3[i]*P[i].dt() ==  \
            (1/(K*et))*((y3[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y3[i+1]-y3[i])/(L_seg))
            + y3[i+1]*((P[i+1]-P[i])/(L_seg))**2)
            - ((1/(et*100))*(R*T))*rho*q3[i].dt()   for i in range(1,seg-1)])

# Last segments component  3
m.Equation(P[seg-1]*y3[seg-1].dt()+y3[seg-1]*P[seg-1].dt() ==
        (1/(K*et))*((y3[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
        + (P2)*((P2-P[seg-1])/(L_seg))*((y3_feed-y3[seg-1])/(L_seg))
        + (y3_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q3[seg-1].dt())


# First segment component 4
m.Equation(P[0]*y4[0].dt()+y4[0]*P[0].dt() ==
        (1/(K*et))*((y4[0]*P[0]*(P[1] - 2*P[0] + P1)/(L_seg**2))
        + P[1]*((P[1]-P[0])/(L_seg))*((y4[1]-y4[0])/(L_seg))
+ y4[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q4[0].dt()))

# Middle segments component  4
```

```
m.Equations([P[i]*y4[i].dt()+y4[i]*P[i].dt() ==  \
            (1/(K*et))*((y4[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y4[i+1]-y4[i])/(L_seg))
            + y4[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - ((1/(et*100))*(R*T))*rho*q4[i].dt()   for i in range(1,seg-1)])

# Last segments component  4
m.Equation(P[seg-1]*y4[seg-1].dt()+y4[seg-1]*P[seg-1].dt() ==
        (1/(K*et))*((y4[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
        + (P2)*((P2-P[seg-1])/(L_seg))*((y4_feed-y4[seg-1])/(L_seg))
        + (y4_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q4[seg-1].dt())

# First segment component 5
m.Equation(P[0]*y5[0].dt()+y5[0]*P[0].dt() ==
        (1/(K*et))*((y5[0]*P[0]*(P[1] - 2*P[0] + P1)/(L_seg**2))
        + P[1]*((P[1]-P[0])/(L_seg))*((y5[1]-y5[0])/(L_seg))
+ y5[1]*((P[1]-P[0])/(L_seg))**2)
        - ((1/(et*100))*(R*T)*rho*q5[0].dt()))

# Middle segments component 5
m.Equations([P[i]*y5[i].dt()+y5[i]*P[i].dt() ==  \
            (1/(K*et))*((y5[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y5[i+1]-y5[i])/(L_seg))
            + y5[i+1]*((P[i+1]-P[i])/(L_seg))**2)
             - ((1/(et*100))*(R*T))*rho*q5[i].dt()
             for i in range(1,seg-1)])

# Last segments component 5
m.Equation(P[seg-1]*y5[seg-1].dt()+y5[seg-1]*P[seg-1].dt() ==
        (1/(K*et))*((y5[seg-1]*P[seg-1]*(P2-2*P[seg-1]
        +P[seg-2])/(L_seg**2))
        + (P2)*((P2-P[seg-1])/(L_seg))*((y5_feed-y5[seg-1])/(L_seg))
        + (y5_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q5[seg-1].dt())


# First segment component 6
m.Equation(P[0]*y6[0].dt()+y6[0]*P[0].dt() == (1/(K*et))*((y6[0]*P[0]*(P[1]
```

```
        - 2*P[0] + P1)/(L_seg**2))
        + P[1]*((P[1]-P[0])/(L_seg))*((y6[1]-y6[0])/(L_seg))
        + y6[1]*((P[1]-P[0])/(L_seg))**2) - ((1/(et*100))*(R*T)*rho*q6[0].dt()))

# Middle segments component 6
m.Equations([P[i]*y6[i].dt()+y6[i]*P[i].dt() ==  \
            (1/(K*et))*((y6[i]*P[i]*(P[i+1]-2*P[i]+P[i-1])/(L_seg**2))
            +  P[i+1]*((P[i+1]-P[i])/(L_seg))*((y6[i+1]-y6[i])/(L_seg))
            + y6[i+1]*((P[i+1]-P[i])/(L_seg))**2)
            - ((1/(et*100))*(R*T))*rho*q6[i].dt()
             for i in range(1,seg-1)])

# Last segments component  6
m.Equation(P[seg-1]*y6[seg-1].dt()+y6[seg-1]*P[seg-1].dt() ==
        (1/(K*et))*((y6[seg-1]*P[seg-1]*(P2-2*P[seg-1]+P[seg-2])/(L_seg**2))
         + (P2)*((P2-P[seg-1])/(L_seg))*((y6_feed-y6[seg-1])/(L_seg))
        + (y6_feed)*((P2-P[seg-1])/(L_seg))**2)
        - ((1/(et*100))*(R*T))*rho*q6[seg-1].dt())

#Velocity calculation
m.Equation(v[0] == -(1/K)*(P[1]-P[0])/(L_seg))
m.Equation([v[i] == -(1/K)*(P[i+1]-P[i])/(L_seg) for i in range(1,seg-1)])
m.Equation(v[seg-1] == -(1/K)*(P2 - P[seg-1])/(L_seg))

#Summ of molar fractions == 1
m.Equation([y1[i] +  y2[i] + y3[i] + y4[i] + y5[i] + y6[i] == 1
            for i in range(0,seg)])

#Solid loading
m.Equation([q1[i].dt() == -MTC[0]*(q1[i]
- (Q1*b1*y1[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q2[i].dt() == -MTC[1]*(q2[i]
- (Q2*b2*y2[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q3[i].dt() == -MTC[2]*(q3[i]
- (Q3*b3*y3[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q4[i].dt() == -MTC[3]*(q4[i]
```

```
- (Q4*b4*y4[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q5[i].dt() == -MTC[4]*(q5[i]
- (Q5*b5*y5[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])
m.Equation([q6[i].dt() == -MTC[5]*(q6[i]
- (Q6*b6*y6[i]*P[i]/(1+b1*y1[i]*P[i]+b2*y2[i]*P[i]+b3*y3[i]*P[i]+
b4*y4[i]*P[i]+b5*y5[i]*P[i]+b6*y6[i]*P[i]))) for i in range(0,seg)])

# simulation
m.options.IMODE = 7
m.solve(disp = False)

print("Finished purge step")

#Write the results to Excel file
if cycle_count == number_cycles - 1:

    velocity_out = -1*np.array(v[0])
    pressure_out = np.array(P[0])
    methane_out = np.array(y1[0])
    ethane_out = np.array(y2[0])
    propane_out = np.array(y3[0])
    butane_out = np.array(y4[0])
    co2_out = np.array(y5[0])
    n2_out = np.array(y6[0])


    co2_out = np.array(y2[seg-1])

    flow_methane =  velocity_out*Area*methane_out*pressure_out*10**8/(R*T)
    flow_ethane =  velocity_out*Area*ethane_out*pressure_out*10**8/(R*T)
    flow_propane =  velocity_out*Area*propane_out*pressure_out*10**8/(R*T)
    flow_butane =  velocity_out*Area*butane_out*pressure_out*10**8/(R*T)
    flow_co2 = velocity_out*Area*co2_out*pressure_out*10**8/(R*T)
    flow_n2 = velocity_out*Area*n2_out*pressure_out*10**8/(R*T)


    old_indices = np.arange(0,len(flow_methane))
```

```
new_length = 100
new_indices = np.linspace(0,len(flow_methane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_methane,k=1,s=0)
flow_methane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_ethane))
new_length = 100
new_indices = np.linspace(0,len(flow_ethane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_ethane,k=1,s=0)
flow_ethane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_propane))
new_length = 100
new_indices = np.linspace(0,len(flow_propane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_propane,k=1,s=0)
flow_propane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_butane))
new_length = 100
new_indices = np.linspace(0,len(flow_butane)-1,new_length)
spl = UnivariateSpline(old_indices,flow_butane,k=1,s=0)
flow_butane  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_co2))
new_length = 100
new_indices = np.linspace(0,len(flow_co2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_co2,k=1,s=0)
flow_co2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(flow_n2))
new_length = 100
new_indices = np.linspace(0,len(flow_n2)-1,new_length)
spl = UnivariateSpline(old_indices,flow_n2,k=1,s=0)
flow_n2  = np.abs(spl(new_indices))

old_indices = np.arange(0,len(pressure_out))
new_length = 100
new_indices = np.linspace(0,len(pressure_out)-1,new_length)
spl = UnivariateSpline(old_indices,pressure_out,k=1,s=0)
```

```
    pressure_out  = np.abs(spl(new_indices))

    old_indices = np.arange(0,len(m.time))
    new_length = 100
    new_indices = np.linspace(0,len(m.time)-1,new_length)
    spl = UnivariateSpline(old_indices,m.time,k=1,s=0)
    time  = np.abs(spl(new_indices))

    np_array_rows = (time,pressure_out,flow_methane,flow_ethane,
                     flow_propane,flow_butane,flow_co2,flow_n2)

    with open('results_test3.csv','a') as csvfile:
        np.savetxt(csvfile, np_array_rows , delimiter=',',
                   header='PURGE',fmt='%s', comments='')

y1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y1_val[i] = np.array(y1[i].value)
y1_val[seg-1] = y1[seg-2]
y1_val = y1_val.T
methane_after_purge = y1_val
y2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y2_val[i] = np.array(y2[i].value)
y2_val[seg-1] = y2[seg-2]
y2_val = y2_val.T
ethane_after_purge = y2_val
y3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y3_val[i] = np.array(y3[i].value)
y3_val[seg-1] = y3[seg-2]
y3_val = y3_val.T
propane_after_purge = y3_val

y4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y4_val[i] = np.array(y4[i].value)
y4_val[seg-1] = y4[seg-2]
y4_val = y4_val.T
```

```
butane_after_purge = y4_val
y5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y5_val[i] = np.array(y5[i].value)
y5_val[seg-1] = y5[seg-2]
y5_val = y5_val.T
co2_after_purge = y5_val
y6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    y6_val[i] = np.array(y6[i].value)
y6_val[seg-1] = y6[seg-2]
y6_val = y6_val.T
n2_after_purge = y6_val



q1_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q1_val[i] = np.array(q1[i].value)
q1_val[seg-1] = q1[seg-2]
q1_val = q1_val.T
q1_after_purge = q1_val


q2_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q2_val[i] = np.array(q2[i].value)
q2_val[seg-1] = q2[seg-2]
q2_val = q2_val.T
q2_after_purge = q2_val


q3_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q3_val[i] = np.array(q3[i].value)
q3_val[seg-1] = q3[seg-2]
q3_val = q3_val.T
q3_after_purge = q3_val
```

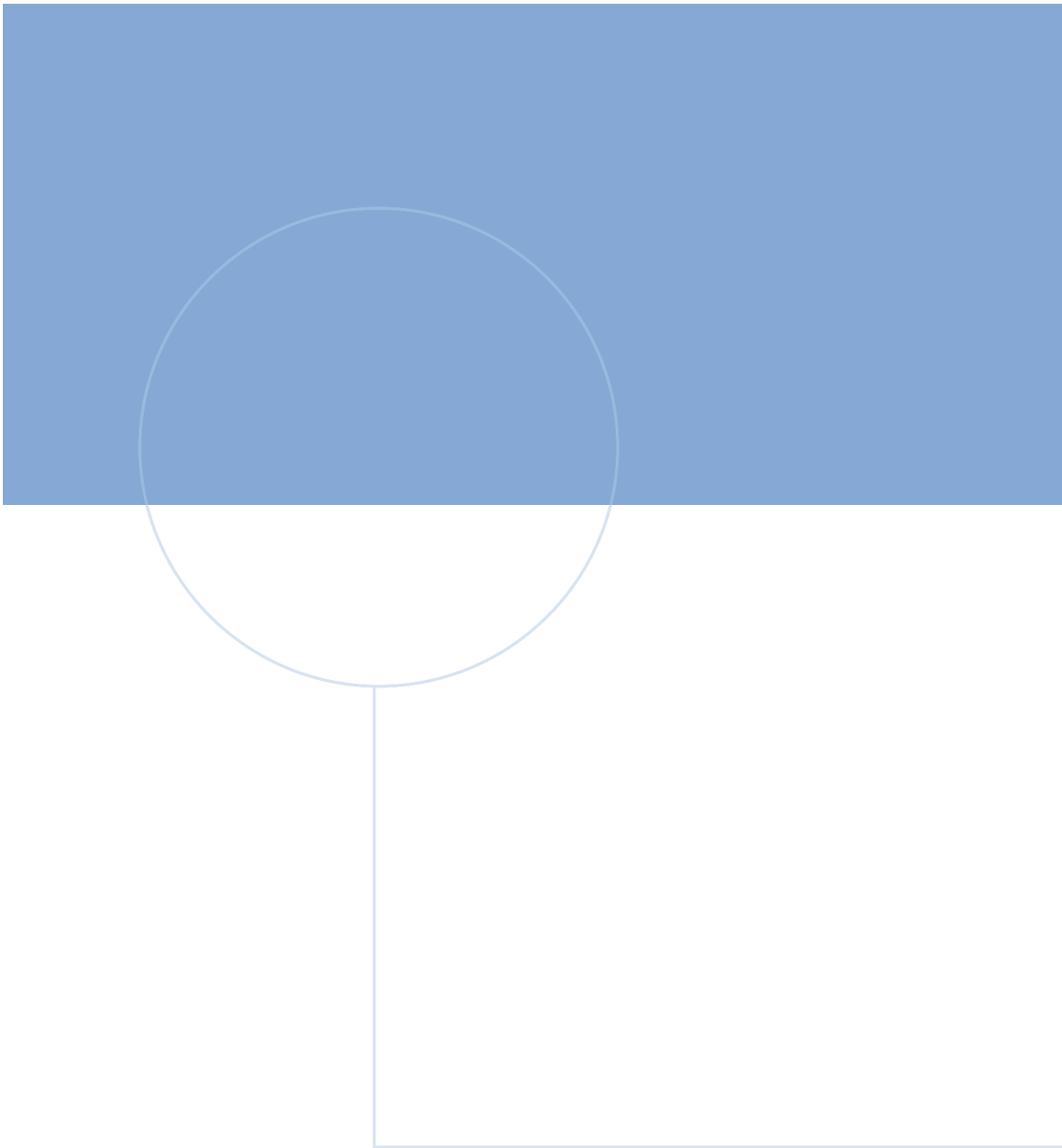
```
q4_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q4_val[i] = np.array(q4[i].value)
q4_val[seg-1] = q4[seg-2]
q4_val = q4_val.T
q4_after_purge = q4_val


q5_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q5_val[i] = np.array(q5[i].value)
q5_val[seg-1] = q5[seg-2]
q5_val = q5_val.T
q5_after_purge = q5_val

q6_val = np.empty((seg,len(m.time)))
for i in range(seg-1):
    q6_val[i] = np.array(q6[i].value)
q6_val[seg-1] = q6[seg-2]
q6_val = q6_val.T
q6_after_purge = q6_val


vel_val = np.empty((seg,len(m.time)))
for i in range(seg):
    vel_val[i] = np.array(v[i].value)
vel_val = vel_val.T
vel_after_purge = vel_val

pressure_val = np.empty((seg,len(m.time)))
for i in range(seg):
    pressure_val[i] = np.array(P[i].value)
pressure_val = pressure_val.T
pressure_after_purge = pressure_val
```



NTNU

Norwegian University of
Science and Technology