Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Yoonsik Oh

Self-Optimizing Control of an Offshore Blue Hydrogen Plant

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Evren M. Turan
June 2023

NTNU
Norwegian University of
Science and Technology

Yoonsik Oh

Self-Optimizing Control of an Offshore Blue Hydrogen Plant

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Evren M. Turan
June 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This thesis is a Master's thesis for the course TKP4900 - "Chemical Process Technology, Master's Thesis". The project was done during the spring of 2023 at the Norwegian University of Science and Technology in the Process System Group at the Department of Chemical Engineering. This work undertakes the challenge of studying the feasibility of an offshore blue hydrogen facility in Norway and the loss of optimality when a control structure developed using "self-optimizing control" is implemented.

I would like to thank my supervisor Johannes Jäschke for providing me with expert process engineering knowledge and especially my co-supervisor in this project, Evren Mert Turan, who has been available for meetings and helped with the smallest of problems to large programming and numerical problems. Without these two, I could never imagine being finished with this project in time. I would also thank SUBPRO for choosing me as their Summer Intern to develop their Offshore Blue Hydrogen model, which motivated me to continue on this project on my Specialization Thesis and Master's Thesis.

I want to thank all of the friends I have made in my five years as a student in Trondheim, and especially the chemistry student organization, Høiskolens Chemikerforening. Without this organization, I would never be the person I am today, and I can never thank you enough. All the experience, the trips to Baalveckan and Berzelii, being a member of the board under corona restrictions, digital and non-digital Vinterblot, and many more memories. I could write another thesis about my student life in Trondheim, but I hope that this preface is enough. I would also like to thank my roommates throughout the years, especially Tobias B. and Stian S. who have been there and supported me at the lowest point of my life. Thank you for keeping me going.

Finally, I am indebted to my parents, my mother, Eunkyong, who moved from South Korea to Norway and built a successful career all by herself, and my father, Seongje, who supported me despite living in South Korea. I really appreciate the encouragement and the support I have received ever since I can remember. I hope you can finally rest, even though I could never repay the debt I owe you two. Their unwavering encouragement and support have been my biggest motivation in completing my Master's degree and I hope that this achievement proves to them that their belief in me was not misplaced. I am privileged to be born with such parents who have committed to providing for me before themselves, regardless of how hard the times are. This Master's Thesis is dedicated to my beloved parents.

Trondheim, May 30, 2023
Yoonsik Oh

Abstract

The development of renewable energy sources is steadily increasing, but further progress is necessary. The discovery of new methods for producing eco-friendly fuels and energy sources is imperative, with hydrogen being a such promising option. However, conventional methods for producing hydrogen result in significant environmental damage due to the large amount of CO_2 produced as a by-product. Blue hydrogen is produced by steam-reforming natural gas with carbon capture and storage and therefore is considered as a low-carbon-emitting method for producing hydrogen. Currently, most blue hydrogen is produced onshore, which necessitates significant transport of gases to and from the natural gas source and the offshore storage site. Moving the production process offshore to a platform or ship would dramatically reduce transport needs, as the hydrogen product would be the sole item requiring transport back to shore. However, modifications to the production process would be necessary due to weight and space constraints when operating offshore. The objective of this project is to develop a model that accurately reflects the process, set up an economic optimization problem, a control structure that can reject disturbances, and study the economic feasibility of the process.

The goal of the optimization problem is to maximize the profit of the process by finding an optimal amount of hydrogen produced and energy required for compressing CO_2, which is dependent on the amount of CO_2 produced in the process. The profit is maximized subject to operational constraints and the physics of the plant model. Important variables in the process have been chosen, which are called input variables, and the optimization results are studied to see which variables can be controlled by manipulating the input variables. By studying how the system behaves when the input variables and disturbance variables change, a control structure has been formed by using the "self-optimizing control" method. Different control structures have been implemented and compared to validate their performance by studying the loss of optimality.

The resulting profit from the nominal case has shown a profit of 1268.2 \$/h, which corresponds to approximately 11 million USD annually, assuming that the facility is operating continuously. Due to this low income, in addition to the neglected costs in the process, such as oxygen costs, PSA operating costs, purifying costs, and hydrogen transport costs, this project is concluded to be infeasible. The demand for hydrogen is also too low for a hydrogen price to make this process feasible. Supplying the facility with self-generated electricity can be a step closer to feasibility, in addition to a carbon tax or higher hydrogen prices. Control structures showed constraint infeasibility in some cases due to the disturbance changes being too high. Some control structures showed good performance with approximately zero loss of optimality, which indicates that control structures from the "self-optimizing control" method show effective disturbance rejection, assuming that the linear assumption holds for all disturbance changes. Further work may include further model development, analysis of different economic scenarios, and studying alternatives to PSA unit for H_2/CO_2 separation, generating electricity, and methods to produce oxygen stream.

Sammendrag

Utviklingen av fornybare energikilder øker stadig, men ytterligere fremgang er nødvendig. Oppdagelsen av nye metoder for å produsere miljøvennlig drivstoff og energikilder er avgjørende, med hydrogen som et lovende alternativ. Konvensjonelle metoder for å produsere hydrogen resulterer imidlertid i betydelig miljøskade på grunn av den store mengden CO_2 som produseres som et biprodukt. Blått hydrogen regnes som en lavkarbon-utslippsmetode for å produsere hydrogen der hydrogenet produseres ved dampreformerende naturgass med karbonfangst og -lagring. Dagen blå hydrogenproduksjon er basert på land, noe som krever en del transport av gasser til og fra naturgasskilden til anlegget tilbake til karbonlagringsstedet i havet. Flytting av produksjonsprosessen offshore til en plattform eller et skip vil dramatisk redusere transportbehovet, ettersom hydrogenproduktet vil være det eneste elementet som krever transport tilbake til land. Samtidig vil modifikasjoner av produksjonsprosessen være nødvendig på grunn av vekt- og plassbegrensninger ved drift offshore. Målet med dette prosjektet er å utvikle en modell som nøyaktig gjenspeiler prosessen, sette opp et økonomisk optimaliseringsproblem, en kontrollstruktur som kan avvise forstyrrelser, og studere den økonomiske gjennomførbarheten av prosessen.

Målet med optimeringsproblemet er å maksimere fortjenesten av prosessen ved å finne en optimal mengde hydrogen produsert med en optimal mengde energi som kreves for å komprimere CO_2, som er avhengig av mengden CO_2 som produseres i prosessen. Fortjenesten maksimeres avhengig av operasjonelle begrensninger og fysikken bak modellen. Viktige variabler i prosessen er valgt, som kalles inputvariabler og optimaliseringsresultatene studeres for å se hvilke variabler som kan bli kontrollert ved å manipulere inputvariablene. Ved å studere hvordan systemet oppfører seg når inputvariablene og forstyrrelsesvariablene endres, er det dannet en kontrollstruktur ved å bruke "self-optimizing control" metoden. Ulike kontrollstrukturer har blitt implementert og sammenlignet for å validere deres ytelse ved å studere tap av optimalitet.

Resultatet fra det nominelle tilfellet har vist et overskudd på 1268.2 $/time, som tilsvarer ca. 11 millioner USD årlig, forutsatt at anlegget er i drift kontinuerlig. På grunn av denne lave inntekten, i tillegg til de neglisjerte kostnadene i prosessen, som oksygenkostnader, PSA-driftskostnader, rensekostnader og hydrogentransportkostnader, konkluderes dette prosjektet med å være ugjennomførbart. Etterspørselen etter hydrogen er også for lav til en hydrogenpris til å gjøre denne prosessen gjennomførbar. Å forsyne anlegget med egenprodusert strøm kan også være et skritt nærmere for at prosessanlegget er gjennomførbart, i tillegg til karbonavgift eller høyere hydrogenpriser. Kontrollstrukturer viste i noen tilfeller umulighet av begrensning på grunn av at forstyrrelsesendringene var for høye. Noen kontrollstrukturer viste god ytelse med tilnærmet null tap av optimalitet, noe som viser at kontrollstrukturer fra "self-optimizing control"-metoden viser effektiv forstyrrelsesavvisning, forutsatt at den lineære antakelsen gjelder for alle forstyrrelsesendringer. Videre arbeid kan omfatte videre modellutvikling, analyse av ulike økonomiske scenarier og å studere alternativer til PSA-enhet for H_2/CO_2-separasjon, generering av elektrisitet og metoder for å produsere oksygenstrøm.

Contents

List of Figures

List of Tables

Abbreviations

ATR	Autothermal reformer
CCS	Carbon capture and storage
CV	Controlled variable
EOR	Enhanced oil recovery
GHR	Gas-heated reformer
ITSR	Isothermal shift reactor
JM	Johnson Matthey
MIQP	Mixed integer quadratic optimization problem
MPC	Model predictive control
PID	Proportional-Integral-Derivative
PR	Pre-reformer
PSA	Pressure swing adsorption
RTO	Real-time optimization
S/C	Steam-to-carbon ratio
SMR	Steam methane reforming
SOC	Self-optimizing control
TPM	Throughput manipulator

List of control symbols

C	Controller
c	Controlled variable
c_s	Control variable setpoint
$c_i(x)$	Constraint function
$\mathcal{D}$	Disturbance noise set
d	Disturbance variables
d'	Scaled disturbance noise
$\mathcal{E}$	Set of equality constraints
e	Error

F	Optimal sensitivity matrix
$\tilde{F}$	Augmented optimal sensitivity matrix
$f(x)$	Objective function depending on x
$f(\bar{u}, x, d)$	Model equations depending on u, x and d
G_i	Generated/consumed amount of component i
G^y	Output gain with respect to inputs
G_d^y	Output gain with respect to disturbances
$g(\bar{u}, x, d)$	Operational constraint depending on u, x and d
H	Optimal combination/selection matrix
$h(y)$	Function of measurements
h	Perturbation size of variable x
k	Perturbation size of variable y
I	Identity matrix
$\mathcal{I}$	Set of inequality constraints
J_u	Reduced gradient of the objective function, J
$J(\bar{u}, x, d)$	Objective function depending on u, x and d
J_{ud}	Hessian matrix of the cost function with respect to inputs and disturbances
J_{uu}	Hessian matrix of the cost function with respect to inputs
L	Loss in optimality
M	Loss Matrix
$m(\bar{u}, x, d)$	Plant model
$\mathcal{N}$	Measurement noise set
n'	Scaled measurement noise
n_c	Number of controlled variables
n_d	Number of disturbance variables
n_f	Number of model equations
n_g	Number of operational constraints
n_u	Number of input variables
n_x	Number of output variables
n^y	Measurement noise

$\mathcal{O}^n$	Sum of remaining taylor expansion term of power n
Q	Any invertible matrix
u	Manipulated/input variable
$\bar{u}$	Near-optimal input
W_d	Diagonal scaling matrix for disturbance noise
W_n	Diagonal scaling matrix for measurement noise
x	States/decision variables
y	Output variable
y_0	Non-disturbed output variable
y_m	Measured output variable
y_s	Setpoint of output variable y
z	Loss variable

List of process model variables

A_i, B_i, C_i	Antoine parameters
c_p	Heat capacity at constant pressure
c_v	Heat capacity at constant volume
E	Total energy
E_k	Kinetic energy
E_p	Potential energy
F	Flow/Feed stream
γ	Specific heat ratio
H	Enthalpy
HHV	Higher heating value
h	Specific enthalpy
$K(T)$	Temperature-dependent equilibrium constant
K_{VLE}	Vapor-liquid equilibrium constant
ψ	Vapor/feed stream ratio
L	Liquid stream

m	Mass stream
n	Mole stream
ν	Stoichiometric coefficient
p	Pressure
p^{sat}	Saturated partial pressure
P_{el}	Electricity cost
P_{H_2}	Hydrogen price
Q	Heat in the process
R	Ideal gas constant
T	Temperature
t	Split ratio
U	Internal energy
V	Volume/Vapor stream
W	Work
W_{flow}	Flow work
W_s^{rev}	Reversible shaft work
ξ	Extent of reaction
x	Liquid stream mole fraction
y	Vapor stream mole fraction
z	Feed stream mole fraction

Convention

Δ	Deviation of a variable from the nominal point
$(\cdot)_{nom}$	Variable at its nominal point
$(\cdot)^{opt}$	Variable at its optimal point
$\|\| \cdot \|\|_n$	n-th norm
$\bar{\sigma}(\cdot)$	Largest singular value
$(\cdot)_j$	Variable for reaction j
$(\cdot)_i$	Variable for stream i

$(\cdot)_{in}$	Inlet variable
$(\cdot)_{out}$	Outlet variable
$(\cdot)_{gen}$	Generated/consumed variable
$+(\cdot)$	Positive perturbation
$-(\cdot)$	Negative perturbation

1 Introduction

1.1 Motivation

The International Energy Agency published a report stating that bringing the global energy-related carbon dioxide emissions to net zero by 2050 might give hope to limit the global temperature rise of 1.5 °C[1]. Following this, the European Union has set a climate-neutral goal by 2050. In addition, since the Paris Agreement in 2015, the participating countries have devoted to reduce greenhouse gas emissions. United Nations' Intergovernmental Panel on Climate Change indicates that if a threshold of 1.5 °C in global average temperature is crossed, irreversible climate changes will impact the world. To limit the temperature below the threshold, greenhouse emissions are required to subside before 2025 at the latest and decline by 43% by 2030[2]. With a steady pace of growth of global energy demand, large efforts within all sectors around the world must be made. A study showed that since 1900, the energy demand has risen by 2.8% with a growth of 2.2% in the 21st century[3]. It is unsurprising that emissions from the oil and gas industry are responsible for a considerable amount in some countries. In 2020, the United States were responsible for yearly emissions of 316 Mt CO_2eq[4] and while the United Kingdom were responsible for 16 Mt CO_2[5], which is equivalent to approximately 5% of the total emission in their respective country[6] [7]. In Norway, the oil and gas sector was responsible for 27% of the country's overall emissions, where the actual amount of emission was 13.3 Mt CO_2eq[8]. A statistical review from BP showed that over 83% of the world's energy consumption derives from fossil fuel sources, and shows hows the global infrastructure today is dependent on energy generated on CO_2 emitting sources[9]. In other words, the problem is that the world is in need of more energy in the future, but today's energy source is heavily dependent on greenhouse gas emitting sources, which all sectors around the world are making efforts to reduce.

The consequences of this problem have led to an increase in media attention and capital investments in renewable energy sources, such as windmills, solar panels, and hydropower. In 2018, wind and solar energy sources produced an approximate annual amount of 1850 TWh with a peak growth of around 130-140 TWh annually. These numbers show that the development rate is not enough to reach the 2050 goal set by many global organizations, and it shows that with the same rate, it would take approximately 180 years to meet only 50% of the global energy demand in 2050[3]. In addition, such renewable energy sources have many obstacles, such as storage, location, investment, and these sources might not always meet the demand as the natural resources such as wind current or access to sunlight can fluctuate. The oil and gas sectors have also implemented several methods to reduce emissions, such as carbon capture and storage (CCS), energy efficiency measures, limiting flaring, and alternative power sources, such as offshore windmills. It is important to note that the oil and gas industry will not shut down immediately, as it would cause a major economic loss and will affect the global market tremendously. Instead, it is wise to phase out the oil and gas slowly, until the global infrastructure is ready for such change. Research on how to reduce emissions and reach future global energy needs is becoming more important and has gained a lot of attention in the latest decades. Among important research and development for solving this problem, hydrogen is one of the promising alternatives.

Hydrogen works excellently as an energy carrier and can be considered a completely carbon-free fuel. Hydrogen addresses three major global energy crisis issues[10]. The first issue is to replace emission-emitting fuels with non-emitting fuel that meets the demand for liquid and gaseous fuels and electricity. Hydrogen addresses this problem well, as the only emission is water in a fuel cell or combustion for gas turbines. The second issue is to increase energy utilization efficiency for fuels and electricity production. Hydrogen fuel cells are shown to have 40-60% energy efficiency, compared to a traditional combustion engine in a vehicle that has approximately 25% efficiency[11], as a major part of the combustion is converted to thermal energy that is not being utilized. The third is to tackle the issue of pollutants and the link between energy utilization and greenhouse gas emission in end-use systems[10].

But unlike other fuels like natural gas, only traces of hydrogen generally exist in a reservoir that can be mined and instead must be produced through chemical processes. Therefore, even though hydrogen itself is emission-free, due to the production process, it is indirectly connected to environmental effects and CO_2 emissions. The most common way to produce hydrogen is through steam-methane reforming and coal gasification, which accounts for almost 96% of all hydrogen production globally. In steam-methane reforming, methane gas is reformed by reacting with high-heated steam under 3-25 bar pressure with catalysts, where hydrogen atoms are separated from methane[12]. Hydrogen produced from steam-methane reforming is better known as “grey hydrogen”. In coal gasification, hydrogen gas or syngas is produced by partial oxidation with enough heat and pressure to break down coal into its basic chemical components. Hydrogen produced from this method is either called “black hydrogen” or “brown hydrogen”.

Low carbon hydrogen, such as “blue hydrogen” or “‘green hydrogen”, can also be produced through steam-methane reforming with carbon capture and storage, or electrolysis of water, respectively. It is worth noting that even though the process of electrolysis does not emit any CO_2, the electricity required for the process is often associated with CO_2 emitting sources. Both methods for producing low-carbon methods are relatively new concepts, where a lot of research in the field is ongoing, and few full-scale productions are implemented worldwide. As of 2021, only two blue hydrogen facilities operate at a commercial scale in North America, where both facilities are based onshore[13]. In an area such as North America, where most natural gas is extracted onshore, it makes sense to operate the plant onshore as well. But in a country like Norway, where natural gas is extracted from the seabed, this implementation comes with additional challenges. Natural gas needs to be transported onshore through a long pipeline, then processed to hydrogen, and the captured CO_2 gas needs to be compressed and transported back to an offshore facility, where the CO_2 is stored. To solve the transportation problem, an offshore blue hydrogen facility is considered and studied with an economic objective, and control structures are investigated to determine its feasibility.

1.2 Project background

This Master’s Thesis is a continuation of the specialization thesis that finalized the steady-state model for the offshore blue hydrogen plant[14] and is a part of the SUBPRO Zero

Blue Hydrogen Project. SUBPRO is a center in Research-based innovation (SFI) within subsea production and processing and has connections with different oil and gas sectors in Norway[15]. This project is a new concept and therefore needs a lot of research before it can be operated at a pilot scale, and then finally at a commercial scale. The results of this thesis will hopefully be used to further study the feasibility of the process and other offshore hydrogen production-related projects.

The present project will look at different control structure designs for an offshore blue hydrogen plant facility, such as a platform or a ship. Natural gas will be extracted from the reservoir in the seabed, and instead of a traditional furnace used in a steam-methane reforming process, the process will use two reactors, a gas-heated reformer, and an autothermal reformer will be used, to reduce the requirement for space, weight, and capital cost[16]. A pressure swing adsorption is used to split the final stream mainly consisting of H_2 and CO_2 into a hydrogen-rich product stream. Hydrogen is being transported back onshore as a product and CO_2 gas along with other impurities in the process is being compressed and transported back onshore in the same reservoir as the extracted natural gas used in the process.

To study the feasibility of this process, an economic optimization problem was formed, where the objective is to maximize profit. The resulting objective value is the hourly profit and is used to study the feasibility of an offshore blue hydrogen facility in Norway. To analyze the possibility of different control structures, control structures from the “self-optimizing control” strategy are implemented and investigated, where important variables in the system are set to a constant setpoint with minimal loss in optimality with disturbance rejection. Suppose such a plant is feasible, with a control structure that can operate despite different disturbances, such as fluctuating hydrogen prices and electricity prices, where low carbon hydrogen can be produced, such a plant can play an important role in the “green change” and in meeting the future global energy demand.

1.3 Scope of the project

In this section, the scope and the focus areas of the project are presented. The present project is a continuation of the specialization thesis where the model and the optimization problem for the plant were finalized and tested its robustness through a sensitivity analysis, where the model was proven to be solved. To summarize, the objectives of this Master’s thesis are:

1. Set up an economic optimization problem where profit is maximized.
2. Develop different control structures from the “self-optimizing control” control strategy.
3. Investigate the performance of the control strategies.
4. Study and evaluate the economic performance of the process.

The most important assumption in the present project is the steady-state assumption as it is generally sufficient when studying the economics of a process. With the steady-state assumption, the model is linearized to assume a linear model. It is also assumed that the only revenue from the process is the hydrogen-rich product stream, and the only cost in the process is the electricity cost for compressing the CO_2 gas for injection into the reservoir. The heating required in the process is assumed to be satisfied by combusting either the natural gas feed stream or the hydrogen product stream as fuel, such that the only external energy required is the electricity imported from onshore for compressing the gas for injection. This fuel switch aspect is handled by the heater unit. The cost for CO_2 injection is considered to be the energy required for compressing the gas to a sufficient pressure, where compression work is assumed to be ideal. The electricity is assumed to be transferred onshore by undersea cables. For the "self-optimizing control" strategy, there will only be considered local methods for comparing loss for different control structures. It is assumed that the active constraint set in the optimization problem remains the same when inputs and disturbances change.

Other costs like transporting hydrogen onshore, and operating the plant and injection unit will not be considered in the economic objective. The cost of separating sea water for heating up steam to saturate the process with steam is also not considered. The oxygen stream required in the autothermal reformer unit will also not be considered and is assumed that a highly effective membrane with minimal costs is used where the small amount of nitrogen from the air separation will not affect the process. The PSA unit is modeled as a stream splitter, and extensive modeling of the adsorption process will not be considered. The model will not take the physics and complexity behind transporting product stream and CO_2.

1.4 Organization

In the present thesis, there are 7 sections and 2 appendices.

Section 1 contains the motivation behind this project, project background, scope, and the present subsection explaining the structure of the present thesis.

Section 2 contains the necessary technical background. An introduction to general optimization and flowsheet optimization is presented. In addition, first principle laws such as conservation laws are presented. Control theory is then presented, followed by hierarchical- and plantwide control. Lastly, self-optimizing control theory is presented, with problem formulation, local methods, how to evaluate the control structures with different loss evaluations, and problems with self-optimizing control, e.g. measurement subset selection and active set changes during operation.

Section 3 describes the process choice behind the present project and a brief description and theory behind the chosen process units, with a flowsheet over the entire process. The chemical reactions taking place in the different units, parameters such as operating temperatures, split ratio, and pressures from studies are also presented.

Section 4 explains the strategy for modeling the process, the new process flowsheet when modeling, and how the model strategy is implemented in JuMP Julia is explained. The economic objective is also presented in this section. Control structure development strategies, with variable selection, degrees of freedom analysis, and measurement subset selection are also shown here.

Section 5 contains the results of the present project. In the results, the economic loss in optimality that comes from implementing the control aspect for different control structures for the model for different disturbance changes, including the resulting gain- and Hessian matrices, other matrices necessary for the local methods, and choices behind the measurement subset selection in “self-optimizing control” is presented.

Section 6 discusses the results of this project by comparing the performance of the different control structures and the feasibility of the offshore blue hydrogen plant.

Section 7 concludes the project with a summary of the control strategies and the feasibility of the project as an overall and the future of hydrogen, with suggestions for improvements for further developments and projects in the future.

Appendix A contains the resulting matrices that were too long to include in the main report, as the tables consisted of over 184 rows. The resulting nominal values, gain matrices, and optimal sensitivity matrix are presented in this appendix.

Appendix B contains the code that was developed and used in this project, both for modeling and calculation purposes. All files contain a brief explanation of the functions in the different Julia code files.

2 Theory

2.1 Introduction to optimization

Optimization is a powerful mathematical tool that can be used for simulating and studying a system. In an optimization problem, an objective function is either minimized or maximized subject to the constraints of the problem. A typical objective function for a chemical process system can be minimizing the energy required or maximizing profit. The constraints are then the equations in the model describing the system, such as mass and energy balances and chemical reactions. The constraints can either be an equality constraint or an inequality constraint. Since the optimization problem needs constraints to be held, the constraints limit the optimal value of the objective function, thus the constraints create the set the objective can be in. A typical optimization problem can be defined as:

$$\max_{x \in \mathbb{R}^n} f(x,p) \tag{2.1.1}$$

$$\text{s.t.} \quad c_i(x,p) = 0, \quad i \in \mathcal{E} \tag{2.1.2}$$

$$c_i(x,p) \leq 0, \quad i \in \mathcal{I}, \tag{2.1.3}$$

where max denotes that the objective function is being maximized, but the problem can also be minimized by switching the sign of the objective function and vice versa. x is the states or the decision variables of the system of n-dimensions, p are parameters in the system that influences the objective and the constraints, $f(x,p)$ is the objective function, $c_i(x,p)$ are the constraints functions and $\mathcal{E}$ and $\mathcal{I}$ are the sets of indices for equality and inequality constraints, respectively. To find the solution to the optimization problem, a correct algorithm needs to be chosen. There is not a universal algorithm that is most suited for all cases, but there is a collection of algorithms that are tailored to be most suitable for different problems. Choosing the correct algorithm for the given problem is important as it will affect the solving time drastically and whether the solution can be found or not[17]. When the correct algorithm is chosen for the problem and a solution is found, the solution will then give the optimal values of the state of the system for either maximizing or minimizing the objective function. Studying the objective value will also be important when performing self-optimizing control, explained in section 2.7.

2.2 Flowsheet optimization

Flowsheet optimization is about optimizing a chemical process by using different techniques, and in chemical process engineering, flowsheet optimization has been an important tool in many aspects. These aspects can for example be the improvement of the process, sensitivity analysis, the feasibility of the plant, simulations, proposed control strategies, and many more. Usually, when optimizing a flowsheet, a mathematical model is involved for studying the process' behavior when certain parameters are changed. The mathematical models usually are based on first principles, such as mass balances, energy balances, and even component mole balances[18]. Some equations can also be based on collected data from other external simulations, such as equilibrium constant, or heat capacity.

When defining a flowsheet optimization problem, the model consists of an objective function and constraints, and the algorithm chosen for the problem will find the optimal solution for the objective function subject to the constraints. Typical objective functions for a chemical process can be to maximize production, profit, product yield, or quality, while it can also minimize energy consumption or waste production. Process safety could also be the main focus. The constraints are the equations in the model and can either be equality or inequality constraints that must be satisfied. In addition to the model equations, the constraint can also be connection constraints, where it connects different submodels together by requiring the output of the previous submodel to equal the input of the next submodel. Overall, flowsheet modeling is used to construct a process diagram, which can be used to simulate and study the behavior of the system under different conditions and operating parameters.

There are mainly two strategies when setting up a flowsheet model; the sequential approach and the simultaneous approach. The sequential approach is when the submodels are connected, but each submodel is solved on its own and passes output information as input for the next submodel in a specified sequence. While the simultaneous approach has the same connections between the submodels through connection equations, the solver will solve all the submodels simultaneously as a larger model.

The advantage of the sequential approach is that the implementation is easier as each submodel is modeled individually, and the appropriate solver can be chosen for each submodel such that it is computationally efficient. The sequential approach is often used when finding steady-state solutions or nominal points of the process and is therefore useful for simulations. This approach may be an easier method for identifying potential errors in the model, such as numerical errors or model mismatches, as each submodel is solved individually. However, as the submodels are connected in a specific sequence, where one submodel passes information to the other, this approach may not capture all of the interactions in the system and can lead to inconsistencies and inaccuracies due to the issues with recycle loops when modeling. In addition, a sequential approach usually only provides the variables in the system and not the gradients as some efficient first and second-order optimization solvers require.

The advantage of the simultaneous approach is that it provides a more detailed and accurate model as it can also solve a system that has recycle-loops or submodels that are exchanging energy with each other, whereas a sequential approach may lead to non-convergence. However, when modeling in a simultaneous approach, compared to a sequential approach, it requires more information about the model, is more computationally heavy, and is often more challenging to implement. In addition, as all of the submodels are solved simultaneously as a larger model, it can be harder to identify potential errors in the model and it is harder to initialize the model as it may require good initial values to find the solution. The main difference between both approaches is the access to gradients and all units converging together when using a simultaneous approach, which is beneficial for optimization.

There are both strengths and weaknesses with both approaches and the selection of the best approach will depend on the modeling goal, system, and the information available. Choosing the appropriate approach will be important to minimize inaccuracies and achieve dependable results.

2.3 Conservation laws

When studying certain systems, such as a chemical process system, quantities or variables in the system are of interest. Such quantities can for example be the mass, the mole of each component, or the energy in the system. To study the quantities, a balance equation can be used to factor in all the input, output, and generation or consumption of the said quantities. This also ensures that physical laws are upheld throughout the process, and every quantity can be studied both the system as a whole and in more detail. The general balance equation can be explained as:

$$\text{Accumulation} = \text{Input} - \text{Output} + \text{Generation} \tag{2.3.1}$$

But as the present project looks at a steady-state process system, accumulation becomes zero as $\frac{d}{dt} = 0$. In other words, the left side of Equation (2.3.1) becomes 0. The general steady-state balance equation can then be described mathematically as:

$$0 = \sum x_{in} - \sum x_{out} + \sum x_{gen}, \tag{2.3.2}$$

where x is the quantity that is being balanced, $\sum x_{in}$ is the sum of all input of the quantity, $\sum x_{out}$ is the sum of all output of the quantity, and $\sum x_{gen}$ is the sum of all terms that explains the quantity being either generated or consumed, for example, a chemical reaction in a mole balance.

2.3.1 Mass balance

The law of conservation of mass states that mass can neither be created nor destroyed. Even though there have been theories that mass can be converted into energy, but it will be assumed that there is no such conversion, and the total mass is conserved in the system. By replacing x in Equation (2.3.2) and with the steady-state assumption and the fact that there is no generation or consumption of mass, the general steady-state can be described as:

$$0 = \sum m_{in} - \sum m_{out} + 0, \tag{2.3.3}$$

where $\sum m_{in}$ is the sum of all mass inlet streams in the system and $\sum m_{out}$ is the sum of all mass outlet streams. Note that the left-hand side of the equation and the x_{gen} term is zero, due to the assumptions made. By rewriting Equation (2.3.3), the final mass balance is:

$$\sum m_{in} = \sum m_{out}. \tag{2.3.4}$$

Equation (2.3.4) can be interpreted as all inlet mass streams in the systems need to be equal to all outlet mass stream out of the system, and it is important that this relation holds for all parts of the process, both the whole process system and in detail as it can ensure that there is no model match. Some deviations are to be expected when looking at the mass balance of a larger system, as there are more calculations that are solved by a computer and can give machine precision and small numerical errors.

2.3.2 Component balance

As a total mass balance equation in the system can be used to study the mass in the system and the inlet and outlet flow as a whole, a component balance can be used to study each component of the system in more detail. This can be done by introducing mole balances for each component in the system. Unlike mass, the amount of mole of a component can be converted into another component, so in a component mole balance, the generation/consumption term plays an important role. By using Equation (2.3.2), with the steady-state assumption, the general steady-state component balance is defined as:

$$0 = \sum n_{i,in} - \sum n_{i,out} + G_i, \tag{2.3.5}$$

where i is the component that is being balanced, $\sum n_{i,in}$ is the sum of all inlet mole streams of component i, $\sum n_{i,out}$ is the sum of all outlet mole streams of component i and G_i is the amount of mole of component i that is being either generated or consumed by for example, a chemical reaction. By rewriting Equation (2.3.5) into:

$$\sum n_{i,out} = \sum n_{i,in} + G_i, \tag{2.3.6}$$

the general steady-state component balance can be interpreted as the sum of all outlet mole streams of component i needing to be equal to the amount of the component that is being introduced in the system by a molar flow addition to generation or consumption of component i. Note that the G_i term does not have a summation notation, and that is because it can be rewritten as another summation. The generated/consumed amount of mole of component i can be described as:

$$G_i = \sum_j \nu_{i,j} \xi_j, \tag{2.3.7}$$

where i is the component that is being balanced, j is the reaction j, $\nu_{i,j}$ is the stoichiometric coefficient of component i in reaction j and ξ_j is the extent of reaction j[19]. As G_i is a sum of all reactions in the system or the control volume that is being studied, it accounts for all the generation and consumption of the component i that might happen in each of the reactions happening. If there is a reaction where the component i is inert, the stoichiometric coefficient simply becomes zero and has no effect on G_i. By combining Equation (2.3.6) and Equation (2.3.7), the final general steady-state component balance becomes:

$$\sum n_{i,in} = \sum n_{i,out} + \sum_j \nu_{i,j} \xi_j \tag{2.3.8}$$

2.3.3 Energy balance

Identical to the law of conservation of mass, the law of conservation of energy states that energy cannot be created nor destroyed, but unlike mass, it can be transferred into different forms. For example, thermal energy can be used to heat up steam for a turbine to generate electricity. However, the total energy of the system remains constant. The general steady-state energy balance is given as:

$$0 = \sum E_{in} - \sum E_{out} + \sum Q + \sum W, \tag{2.3.9}$$

where the left-hand side of the equation is zero due to the steady-state assumption, $\sum E_{in}$ and $\sum E_{out}$ is the sum of all energy inlet and outlet streams, respectively, usually in the form of enthalpy from the mass streams, $\sum Q$ and $\sum W$ is the sum of all heat or all work added or removed in the system from the environment, respectively. The total energy, E, can be defined as the sum of internal energy, U, kinetic energy, E_k and potential energy, E_p:

$$E = U + E_k + E_p. \tag{2.3.10}$$

In a typical process, the last two terms, E_k and E_p can be neglected[20], and the total energy can then be defined as:

$$E = U. \tag{2.3.11}$$

By combining Equation (2.3.9) and Equation (2.3.11), the steady-state energy balance then becomes:

$$0 = \sum U_{in} - \sum U_{out} + \sum Q + \sum W. \tag{2.3.12}$$

The term, $\sum W$, is the sum of different factors such as electrical work, mechanical work, or flow work, but in this process, only flow work will be taken into account, and the rest of the work terms are assumed to have no influence on the system. By using the definition of flow work, $W_{flow} = pV$, the sum of work can be defined as:

$$\begin{aligned} \sum W = \sum W_{flow} &= \sum W_{in,flow} - \sum W_{out,flow} \qquad (2.3.13)\\ &= \sum p_{in} V_{in} - \sum p_{out} V_{out}, \qquad (2.3.14) \end{aligned}$$

where p_{in} and p_{out} are the pressure of the inlet and outlet flows, respectively, and V_{in} and V_{out} are the volume of the inlet and outlet flows, respectively. By combining Equation (2.3.12) and Equation (2.3.14), the steady-state energy balance is defined as:

$$0 = \sum U_{in} - \sum U_{out} + \sum Q + \sum p_{in} V_{in} - \sum p_{out} V_{out}. \tag{2.3.15}$$

Now introducing the definition of enthalpy as:

$$H = U + pV, \tag{2.3.16}$$

which states that enthalpy is the sum of internal energy and flow work. Equation (2.3.15) can be rewritten as:

$$\sum U_{out} + \sum p_{out} V_{out} = \sum U_{in} + \sum p_{in} V_{in} + \sum Q. \tag{2.3.17}$$

By combining Equation (2.3.17) and the definition of enthalpy shown in Equation (2.3.16), the final steady-state energy balance becomes:

$$\sum H_{out} = \sum H_{in} + \sum Q, \tag{2.3.18}$$

where $\sum H_{in}$ and $\sum H_{out}$ are the sum of all inlet and outlet enthalpy flows, respectively. The sum of heat, $\sum Q$, is usually reaction enthalpy gained or lost from a chemical reaction or heat gained or lost from a heat exchanger, such as a heater or a cooler.

2.4 Introduction to control theory

When studying a real-life system, such as a chemical process plant, the system's components and interactions are of interest. To study this, a model through variables (components) and equations (interactions) can simulate and analyze the system's behavior. However, some variables are outside an engineer's control in the system, called disturbances. To achieve a stable system that operates at a desired level, the system needs a new element to counteract the disturbances. By measuring relevant components and interactions in the system, a controller can use variables that can be manipulated to drive the wanted variables in the system to their desired level. A general control system block diagram with feedback is shown in Figure 1 to illustrate how the controller can affect the model,

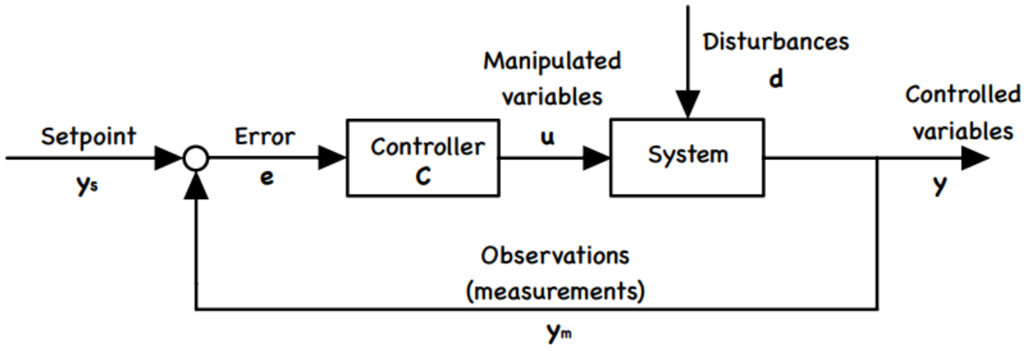

Figure 1: Generalized simple feedback control system.

where y_s is the setpoint which is the desired values of the relevant output variables of the system, y_m is the observed/measured output which is a feedback loop to combine with y_s to calculate the error, e, which is formulated as the difference between the setpoint value and the measured output value. The controller, C, then uses this error to calculate new inputs to the system, which are called manipulated variables, u. With new input and disturbances in the system, the system will then give new resulting output variables or controlled variables. This process is repeated until the error is at an acceptable level; in other words, the resulting output of the system is identical to the setpoint given.

Generally, a control design problem is about formulating a correct controller, C, that gives the system signals u to keep the output close to the setpoint with the disturbances. When formulating such a problem, it needs information about what variables in the system are manipulated, controlled, or disturbances. There are also many different configurations for the controller, such as PID or MPC. However, as mentioned in section 1.3, the scope of this project is not to design a controller, but to select certain controlled variables and keep them at a constant setpoint to achieve near-optimal operation. Such control structure is called "self-optimizing control" and will be explained in detail in section 2.7.

2.5 Hierarchical control

In many control areas, such as chemical processes, the control structures in these areas have complex dynamic behaviors which require advanced process control techniques. One of these techniques is a multi-layered framework developed by Lefkowitz in 1966[21] and is called "Hierarchical control". In hierarchical control theory, the control structures are divided into layers horizontally, where each layer hold responsible for specific aspects of a control structure. The layers higher up have higher priority, influence, and authority than the lower layers. A typical hierarchical control structure proposed by Skogestad in 2000[22] is shown in Figure 2.

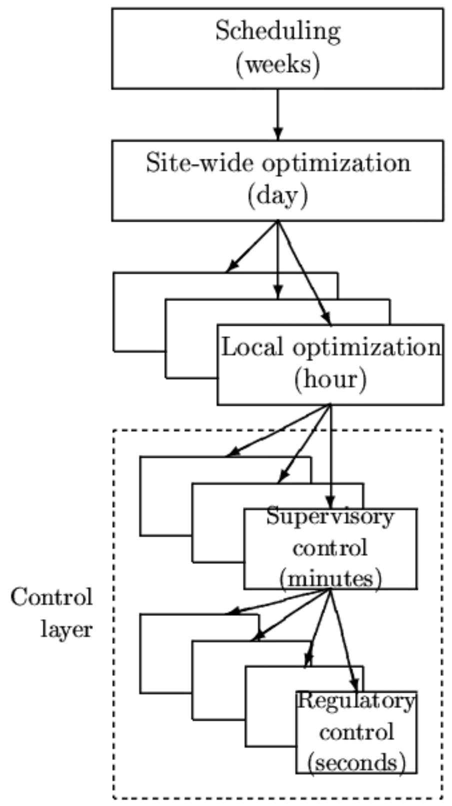

Figure 2: A typical hierarchical structure in chemical process plant[22].

As it can be observed from Figure 2, a typical hierarchical structure can be divided into 5 major layers; scheduling, site-wide optimization, local optimization, supervisory control, and regulatory control, whereas the last two layers can be referred to as the control layer. When considering plantwide control, which will be discussed in more detail in section 2.6, only three lower layers will be considered. The local optimization layer recalculates the optimal setpoints for given disturbances in the process to the supervisory layer by solving a steady-state real-time optimization (RTO) problem. The local optimization layer is also often called the RTO layer. Briefly explained, whenever the supervisory control layer detects that the process is steady-state, the steady-state states are given back to the RTO and it will solve a new optimal setpoint for the next layer. The supervisory layer is the advanced controller such as an MPC and works as the master controller that gives setpoints to the regulatory layer. Controllers in the regulatory layer are the controllers that focus on the stability of the process and have fast dynamics. Examples of controllers in the regulatory controllers are PID controllers.

When applying hierarchical control to a chemical process plant, it is assumed that different layers have different time scales, and these layers are connected by the variables in the process that are of interest, or in other words, the controlled variables (CVs). Since there is a time scale separation between the layers, it can be assumed that the lower layer will immediately react to the CV setpoints given by the upper layer. This allows the optimization and control objectives to be decoupled. Hierarchical control is often preferred over a centralized controller as it is often more robust. If one of the decentralized controllers in a hierarchical control structure shut down, the system does not shut down. It is also more robust since it can handle the disturbances well, due to the time scales between the layers. Another advantage of hierarchical control is that it can combine different control techniques such as optimization, fast local control, and MPC. A further and more detailed explanation of the importance of dividing the control structure into multiple layers and how this can apply to "self-optimizing control" control strategy advantageously is explained in section 2.6 and section 2.7.

2.6 Plantwide control

Plantwide control is referred to as a control strategy where the goal is to achieve a control structure that can operate the whole chemical process plant at a stable and optimum level when the disturbances in the process are in effect and when the objective of the plant changes such as switching from maximizing profits to minimizing energy cost. Plantwide control procedures for operational objectives for achieving short-term stability and long-term economic profitability suggested by Skogestad[23] are:

I. Top-down analysis

Step 1. Define optimal operation in terms of a scalar cost function to be minimized and a set of constraints to be satisfied.

Step 2. Identify steady-state degrees of freedom and determine the steady-state optimal operating point for nominal operation, including active constraints.

Step 3. Identify measurement candidates and select primarily controlled variables for the supervisory control layer based on the principles of self-optimizing control.

Step 4. Select the location of the throughput manipulator (TPM), which will determine where to set the production rate and the structure of the remaining inventory control system.

II. Bottom-down analysis

Step 5. Select secondary controlled variables for the regulatory (stabilizing) control layer and determine the pairing with the manipulated variables.

Step 6. Select the structure of the supervisory control layer (decentralized or multivariable control).

Step 7. Determine the need for an optimization layer (RTO). Validation through nonlinear dynamic simulation.

The top-down analysis focuses on the economics of the plant. When the plant is at steady-state, it can primarily give the economics of the plant, and therefore, a steady-state model is sufficient when performing a top-down analysis. The bottom-down analysis focuses on the robustness and stability of the plant. This project will mainly focus on the top-down analysis where a steady-state model of the process plant is built. The goal of the project is then to select certain control variables and setpoints to achieve near-optimal operation, which will be discussed more in section 2.7.

2.7 Self-optimizing control

A hierarchical control structure model is powerful when handling the complexity of a large multivariable system, such as a chemical process plant. But due to the time-scale separation between the optimization layer and the control layer, the hierarchical control structure comes with a loss when compared to a centralized optimizing controller. This loss comes mainly from the disturbances that influence the plant economics between setpoint updates, and where these changes are not optimally rejected by the lower control layer until the next update. This is due to the slower time-scale setpoint that is updated at low-frequency time intervals in the optimization layer. This loss of optimality depends heavily on the selected controlled variables in the system and is an important decision for the overall profitability of the process plant. One of the first researchers that recognized this problem was Foss (1973), which stated that the most important problem encountered by designers of chemical process control systems is choosing which variables should be measured, which inputs should be manipulated, and what links should be made between these two sets[24].

Furthermore, important research efforts have laid the groundwork for the concept of "self-optimizing control", presented by Skogestad[22]. One of these important research papers is "Studies in the synthesis of control structures for chemical processes" by Morari[25], which introduced the concepts of "feedback optimizing control". He stated that finding a function c of the process variables will automatically lead to the optimal adjustment when held constant, which ultimately leads to the optimal operating conditions. The main idea was to not solve the optimization problem online but through feedback control of optimal variants such as process variables that remain constant whenever the process is operated optimally when the goal is to achieve optimal operation. The "self-optimizing control" concept formulated by Skogestad resembles the work of Morari. The idea is to achieve near-optimal operation and minimize loss from time-scale separation by selecting controlled variables with optimal setpoints that are near-insensitive to measurement noise and disturbances. In other words, self-optimizing control is when an acceptable loss is achieved with constant setpoint values for the controlled variables is achieved without the need to reoptimize when disturbances occur[22].

In summary, by incorporating self-optimizing control principles when designing the control layer, it can reject the disturbances economically, and achieve near-optimal conditions on a fast time scale, without the need to wait for the real-time optimizer to update the setpoints. In addition, the upper optimization layer can still rectify plant-model mismatched or unmodeled disturbances, but on a slower time scale.

2.7.1 Optimal operation problem formulation

Morari stated in his paper about the goal of designing a control structure that the main objective is to translate the economic objective into process control objectives[25]. In other words, process control will always be used for achieving the best performance in terms of economics for given operational conditions and constraints[26], such that the optimal operation problem can be expressed mathematically as an optimization problem.

As disturbances in most continuous processes stay constant long enough to make the short-term economic effects negligible, a quasi-steady-state assumption is made and the problem of optimal operation can be formulated as a static optimization problem where parametric model uncertainties are represented as disturbances:

$$\begin{aligned} \min_{\bar{u}} \quad & J(\bar{u}, x, d) \\ \text{s.t.} \quad & f(\bar{u}, x, d) \\ & g(\bar{u}, x, d), \end{aligned} \tag{2.7.1}$$

where $x \in \mathbb{R}^{n_x}$ are the state variables, $d \in \mathbb{R}^{n_d}$ are the disturbances, $u \in \mathbb{R}^{n_u}$ are the manipulated variables in the system. The manipulated variables can also be called the steady-state degrees of freedom. The variables, x, d, and $\bar{u}$ affect the steady state objective function $J : \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_{\bar{u}}} \rightarrowtail \mathbb{R}$, which is a scalar function and sets the goal of the optimization, such as maximizing revenue or minimizing energy cost. The function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_{\bar{u}}} \rightarrowtail \mathbb{R}^{n_f}$ are the model equations, such as the component and energy balances, and $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_{\bar{u}}} \rightarrowtail \mathbb{R}^{n_g}$ are the operational constraints, such as operating temperatures for the reactors.

When operating the plant, the objective function should be either maximized or minimized depending on the objective goal, while satisfying the operational and plant constraints. Ideally, if the model is completely correct and all the states, x, and the disturbances, d, in the system are known, the objective value can be calculated from a given set of optimal inputs, $\bar{u}$, which will result in optimal operation. In reality, as the plant is nearly and not perfectly at steady-state, and the exact information about the states and the disturbances are not available, this strategy is not implementable. However, this project assumes optimal operation, which means that the model is assumed to be correct and states and disturbances are known. Mathematically, to achieve optimal operation, the necessary conditions need to be satisfied, in other words, the reduced gradient of the objective function with respect to the manipulated variables needs to be zero, $J_u = 0$. In self-optimizing control, the manipulated variable is used to carefully drive the selected controlled variables, c, to their constant setpoints, c_s. Despite measurement noise and disturbances, by monitoring the appropriate variables, a feedback controller can be used to drive the system towards the optimal operating point, in other words, the manipulated variables, $\bar{u}$, will approach u^{opt}.

J_u is an excellent self-optimizing controlled variable, as it is not controlling the objective function directly, but controls the gradient to a constant setpoint of zero. Many problems arise when the objective function is controlled directly, explained in more detail in section 2.7.3. As mentioned previously, there are problems when finding the exact states of the process, and thus is challenging to find the direct evaluation of the gradient. Instead, information about the states of the process is typically available from measurements, and therefore the available plant measurement is assumed to be used for a model:

$$y_0 = m(\bar{u}, x, d), \tag{2.7.2}$$

where the function $m : \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_{\bar{u}}} \rightarrowtail \mathbb{R}^{n_y}$, describes the relationship between the input, states, and disturbances, and $y \in \mathbb{R}^{n_y}$ as the model output. However, in a real process plant, the measured signals, y, are disturbed by measurement noise, $n^y \in \mathbb{R}^{n_y}$, such that:

$$y = y_0 + n^y. \tag{2.7.3}$$

Generally, the controlled variables can be expressed as a function of the measurements, y, where the controller drives the controlled variables to their setpoints given by the real-time optimizer[26]. The controlled variables that are functions of the measurements are given as:

$$c = h(y), \tag{2.7.4}$$

where c is the controlled variables and $h(y)$ is a function of measurements. $h(y)$ can be any function but is often chosen to be linear, such that:

$$c = Hy, \tag{2.7.5}$$

where $H \in \mathbb{R}^{n_c \times n_y}$ is a constant matrix called a selection or combination matrix. When a good set of controlled variables are selected, controlling $c = Hy$ at its setpoint will indirectly result in the corresponding near-optimal inputs, $\bar{u}$, thus the RTO does not require setpoint updates for every operating condition and disturbance changes. Such a control structure is referred to as a self-optimizing control structure. To decide which controlled variables are "good", Skogestad has proposed requirements for good CVs[27]:

1. The CV should be easy to control, that is, the input u should have a significant effect (gain) on c.
2. The optimal value of c should be insensitive to disturbances.
3. The CV should be insensitive to noise.
4. In the case of several CVs, the variables should not be closely correlated.

It is shown from research that controlling the combination of measurements can improve the economic performance of the control strategy, and in addition, lead to lower steady-state losses[27] [28]. A general block diagram for a typical hierarchical control structure with self-optimizing control implemented is shown in Figure 3.

From Figure 3, it can be seen that the constant setpoints are calculated by the real-time optimizer and are passed down to the controller where it adjusts the inputs such that the controlled variables, $c = Hy$ tracks the setpoint closely.

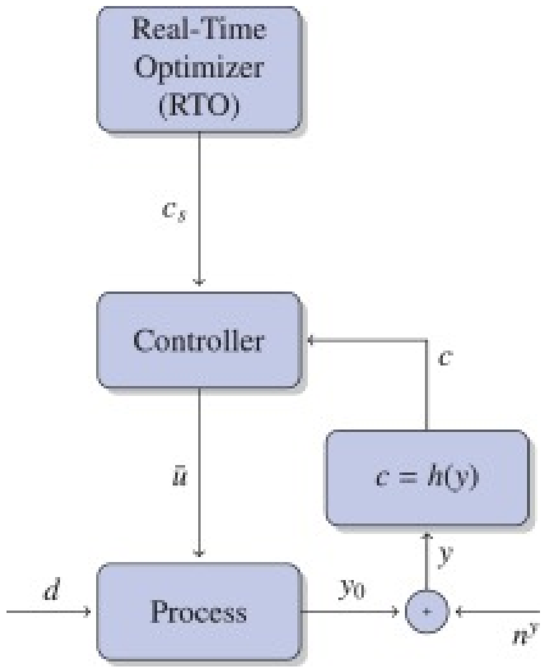

Figure 3: A general hierarchical control structure with self-optimizing control where the figure is taken from a self-optimizing control review article by Jäschke[26].

To evaluate how well the control structure has performed, the economic loss can be used and is defined as the difference between the cost from the control structure and the cost from the true optimal operation. This can be mathematically expressed as:

$$\begin{aligned} L &= J(\bar{u}, x, d) - J(u^{opt}, x, d) \\ &= J(\bar{u}, x, d) - J^{opt}(d), \end{aligned} \tag{2.7.6}$$

where L is the evaluated economic loss, and $J^{opt}(x, d)$ is the true optimal objective value for given disturbances. This economic loss will be used to compare the performance of different control structures, even if the control structure is a self-optimizing control structure.

To summarize, the main issue in self-optimizing control is to select appropriate controlled variables, $c = Hy$, and setpoints. In other words, for a given set of disturbances, d, and measurement noise, n^y, the goal for self-optimizing control is to find an optimal combination matrix, H, and find the correct setpoints which minimize the average or worst-case economic loss for the operating region. Different methods for solving this problem have been reported, and the following section will explain the methods in more detail, such as the local methods, exact local method, and nullspace method.

2.7.2 Local methods

The idea behind local methods is to reduce the number of controlled variables to consider and exclude controlled variables that are poorly chosen when designing the control structure. For a controlled variable to be valid for selection, it needs to perform sufficiently well around the nominal operating point where the process is mostly expected to be. Variables are tested around the nominal operating point to check if it is valid. Such variables will be used for further examination and used for the whole operating range of the process plant. The resulting control structure will be compared against the original nonlinear model for the whole operating region.

Local approximation of the loss

To locally approximate the cost function, the steady-state problem formulation shown in Equation (2.7.1) is used. The model equations, $f(\bar{u}, x, d) = 0$, are used to eliminate the states in the optimization problem, and the new reduced optimization problem then becomes:

$$\begin{aligned} &\min_{\bar{u}} J(\bar{u}, d) = 0 \\ &\text{s.t. } \; g(x, d) \leq 0. \end{aligned} \tag{2.7.7}$$

The set of active constraints, $g(x, d)$ are assumed to not be changing during the operation, and the active constraints will be controlled to their setpoints. Therefore, the set of active constraints is eliminated from the problem formulation and leads to the unconstrained optimization:

$$\min_{u} J(u, d), \tag{2.7.8}$$

where J is the cost function of the unconstrained optimization problem, and u is the remaining steady-state degrees of freedom after controlling the active constraints. u will then be used for controlling the self-optimizing controlled variables. The cost function is then approximated locally by a second-order Taylor expansion around the nominally optimal operating point $(\Delta u^{opt}, \Delta d)$, and is expressed mathematically for given disturbances as:

$$J(\Delta u, \Delta d) = J(\Delta u^{opt}, \Delta d) + J_u^T(\Delta u - \Delta u^{opt}) + \frac{1}{2}(\Delta u - \Delta u^{opt})^T J_{uu}(\Delta u - \Delta u^{opt}) + \mathcal{O}^3, \tag{2.7.9}$$

where J_u and J_{uu} are the Jacobian and the Hessian matrices of the cost function around the nominal operating point, respectively, $\Delta u = u - u_{nom}$ and $\Delta d = d - d_{nom}$ By substituting Equation (2.7.9) into Equation (2.7.6), where $J(\Delta u^{opt}, \Delta d)$ are eliminated from both sides and taking consideration that the Jacobian of the objective value is zero at the optimum, $J_u^T(u - u^{opt})$, the new approximation of loss is formulated as:

$$L \approx \frac{1}{2}(\Delta u - \Delta u^{opt})^T J_{uu}(\Delta u - \Delta u^{opt}). \tag{2.7.10}$$

The loss can then be expressed as:

$$L = ||z||_2^2, \tag{2.7.11}$$

where $|| \cdot ||_2^2$ denotes the two-norm and z is the loss variable and is defined as:

$$z = J_{uu}^{1/2}(\Delta u - \Delta u^{opt}). \tag{2.7.12}$$

Local approximation of the plant and linear measurement combination

To locally approximate the plant, the measurement model shown in Equation (2.7.3) is linearized around the nominal operating point, expressed as:

$$\Delta y = G^y \Delta u + G^y_d \Delta d + n^y, \tag{2.7.13}$$

where $\Delta y = y - y^{opt}$, $\Delta u = u - u_{nom}$ and $\Delta d = d - d_{nom}$. G_y and G^d_y are the gain of the output from inputs and disturbances, respectively, and are defined as the partial derivatives of y with respect to their respective variable around the nominal optimal point. The Jacobian matrices, G_y and G^d_y are defined as:

$$G_y = \frac{\partial y}{\partial u} \tag{2.7.14}$$

$$G^d_y = \frac{\partial y}{\partial d}. \tag{2.7.15}$$

As the linear combinations of measurements, shown in Equation (2.7.5), are linear, Δy can be substituted and the linear combination of measurements can be formulated as:

$$\Delta c = H \Delta y, \tag{2.7.16}$$

where $\Delta c = c - c^{opt}$ By substituting Equation (2.7.13) into Equation (2.7.16), the linear combinations of measurements is formulated as:

$$\Delta c = HG^y \Delta u + HG^y_d \Delta d + Hn^y, \tag{2.7.17}$$

where the matrix, HG^y, needs to fulfill full rank, $\text{rank}(HG^y) = n_u$. This criterion is necessary when obtaining a linearly independent set of CVs that fully specify the system. As long as this criterion is achieved, the elements of the H matrix can take any arbitrary values.

Exact local method

Halvorsen developed a method for evaluating loss for a given control structure, called the exact local method[29]. Exact local method evaluated loss by assuming perfect control despite the disturbances, in other words, $c = c_{nom}$, which leads to $\Delta c = 0$. Substituting $\Delta c = 0$ into Equation (2.7.17), and expressing for Δu gives:

$$\begin{aligned}\Delta u &= (HG^y)^{-1}(\Delta c - HG^y_d \Delta d - Hn^y) \\ &= (HG^y)^{-1}(-HG^y_d \Delta d - Hn^y) \\ &= -(HG^y)^{-1} H (G^y_d \Delta d + n^y).\end{aligned} \tag{2.7.18}$$

Defining the optimal input, Δu^{opt} as

$$\Delta u^{opt} = -J^{-1}_{uu} J_{ud} \Delta d, \tag{2.7.19}$$

which is derived from differentiating Equation (2.7.9) with respect to u, equating the expression to zero since the approximation of the function is around the nominal optimal point, such that $J_u = 0$, then solving for u^{opt}. By substituting Δu^{opt}, shown in Equation (2.7.19), and the input, Δu, derived in Equation (2.7.18) into the loss variable, z, defined in Equation (2.7.12). The loss variable, z, for a given set of disturbances and measurement combination matrix, H, becomes[29]:

$$\begin{aligned}
z &= J_{uu}^{1/2}\Big[\Delta u - \Delta u^{opt}\Big] \\
&= J_{uu}^{1/2}\Big[(-(HG^y)^{-1}H(G_d^y\Delta d + n^y)) - (-J_{uu}^{-1}J_{ud}\Delta d)\Big] \\
&= -J_{uu}^{1/2}((HG^y)^{-1})\Big[(H(G_d^y\Delta d + n^y)) + ((HG^y)J_{uu}^{-1}J_{ud}\Delta d)\Big] \\
&= -J_{uu}^{1/2}((HG^y)^{-1})H\Big[G_d^y\Delta d + n^y + G^yJ_{uu}^{-1}J_{ud}\Delta d\Big] \\
&= -J_{uu}^{1/2}((HG^y)^{-1})H\Big[(G_d^y + G^yJ_{uu}^{-1}J_{ud})\Delta d + n^y\Big] \\
&= -J_{uu}^{1/2}((HG^y)^{-1})H\Big[F\Delta d + n^y\Big],
\end{aligned} \tag{2.7.20}$$

where the optimal sensitivity matrix, F, is defined as:

$$F = G_d^y - G^yJ_{uu}^{-1}J_{ud}, \tag{2.7.21}$$

and represents the matrix of sensitivities of the optimal measurement values with respect to the disturbances[26]:

$$F = \frac{\partial y^{opt}}{\partial d}. \tag{2.7.22}$$

There are many different ways to obtain the optimal sensitivity matrix, F, such as re-optimization, nonlinear programming sensitivity based on the inverse function theorem, or by using the definition shown in Equation (2.7.21), if all Hessian matrices and gain matrices are available. Furthermore, diagonal scaling matrices are defined for the disturbances and measurement noise, W_d and W_n, respectively.

$$\Delta d = W_d d' \tag{2.7.23}$$

$$\Delta n^y = W_n n', \tag{2.7.24}$$

where d' and n' are the scaled disturbances and measurement noise, respectively. It is worth noting that the disturbances are scaled to have a magnitude of 1. The scaling of the variables can be used to evaluate the performance of the chosen set of CVs, as it allows to determine the loss for a given set of disturbances and measurement noise. By using the scaled variables, loss defined in Equation (2.7.11) is formulated as:

$$L = \frac{1}{2}\left|\left|J_{uu}^{1/2}(HG^y)^{-1}H\tilde{F}\begin{bmatrix} d' \\ n' \end{bmatrix}\right|\right|_2^2, \tag{2.7.25}$$

where $\tilde{F}$ is the augmented matrix and is defined as:

$$\tilde{F} \approx \begin{bmatrix} FW_d & W_n \end{bmatrix}. \tag{2.7.26}$$

By introducing the loss matrix, M, as:

$$M = J_{uu}^{1/2}(HG^y)^{-1}HY, \tag{2.7.27}$$

the loss matrix defined in Equation (2.7.25) can be simplified to:

$$L = \frac{1}{2}\left|\left|M\begin{bmatrix}d' \\ n'\end{bmatrix}\right|\right|_2^2. \tag{2.7.28}$$

As the loss matrix, M, has the property of remaining constant when multiplied by any invertible matrix, Q, the steady-state loss will be independent of the scaling from the chosen set of CVs. Jäschke showed this by first considering that the set of CVs is rescaled using the scaling matrix, Q, such that[26]:

$$\begin{aligned}\Delta\tilde{c} &= Q\Delta c \\ &= Q(H\Delta y) \\ &= \tilde{H}\Delta y,\end{aligned} \tag{2.7.29}$$

where $\tilde{H}$ is the rescaled selection matrix, and to further prove that scaling the set of CVs will generally not affect the loss at steady state, it can also be seen that this is true for the loss matrix, M, such that[26]:

$$\begin{aligned}M &= J_{uu}^{1/2}(\tilde{H}G^y)^{-1}\tilde{H}Y \\ &= J_{uu}^{1/2}(QHG^y)^{-1}QHY \\ &= J_{uu}^{1/2}(HG^y)^{-1}Q^{-1}QHY \\ &= J_{uu}^{1/2}(HG^y)^{-1}HY.\end{aligned} \tag{2.7.30}$$

The following paragraphs show how local expressions are derived for the average and the worst-case loss under different assumptions on disturbance and measurement noise distributions, and how these expressions can be used for deriving minimum loss method and the nullspace method suggested by Skogestad[27].

Worst-case loss

Halvorsen stated that the worst-case loss can be derived from the two-norm bounded disturbance and measurement noise under the assumption that the disturbance and measurement noise are independent and uniformly distributed over the set, such that[29]:

$$\mathcal{DN}_2 = \left\{(d', n')\middle| \; \left|\left|\begin{bmatrix}d' & n'\end{bmatrix}^T\right|\right|_2 \leq 1\right\}, \tag{2.7.31}$$

where $\mathcal{DN}_2$ is the two-norm bounded disturbance and measurement noise set that contains all the allowable disturbance and measurement noise values. By using the loss expressed in

Equation (2.7.28), the worst-case loss can be expressed as[29]:

$$\begin{aligned} L_w &= \max_{\left\|\begin{bmatrix} d' \\ n' \end{bmatrix}\right\|_2 \leq 1} \frac{1}{2}\left\| M \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2^2 \\ &= \frac{1}{2}||M||_2^2 \\ &= \frac{1}{2}\bar{\sigma}^2(M), \end{aligned} \tag{2.7.32}$$

where $\bar{\sigma}(\cdot)$ represents the largest singular value.

Average loss

Kariwala stated that the average loss can be derived from the infinity-norm bounded disturbance and measurement noise under the assumption that d' and n' are independent and uniformly distributed over the set, such that[30]:

$$\mathcal{DN}_\infty = \left\{ (d', n') \middle| \left\| \begin{bmatrix} d' & n' \end{bmatrix}^T \right\|_\infty \leq 1 \right\}, \tag{2.7.33}$$

where $\mathcal{DN}_\infty$ is the infinity-norm bounded disturbance and measurement noise set. Identically to the derivation of the worst-case loss, Equation (2.7.28) can be used to derive the average loss, and is formulated as[30]:

$$\begin{aligned} L_{avg} &= \mathop{\mathbb{E}}_{d', n' \in \mathcal{DN}_\infty} \left[\frac{1}{2}\left\| M \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2^2 \right] \\ &= \frac{1}{6}||M||_F^2, \end{aligned} \tag{2.7.34}$$

where $\mathbb{E}[\cdot]$ is the expectation operator.

Kariwala also derived expressions for average loss by normally distributing the disturbance and measurement noise under the assumption that the variables are normally distributed with zero mean and unit variance, such that[30]:

$$\mathcal{DN}_\mathcal{N} = \{ d' \sim \mathcal{N}(0, I), n' \sim \mathcal{N}(0, I) \}, \tag{2.7.35}$$

where $\mathcal{DN}_\mathcal{N}$ is the normally distributed disturbance and measurement noise, with the same method as previous derivations of loss, the average loss becomes:

$$\begin{aligned} L_{avg} &= \mathop{\mathbb{E}}_{d', n' \in \mathcal{DN}_\mathcal{N}} \left[\frac{1}{2} M \begin{bmatrix} d' \\ n' \end{bmatrix} \right] \\ &= \frac{1}{2}||M||_F^2, \end{aligned} \tag{2.7.36}$$

where the worst-case loss results unbounded ($L_w = \infty$), as the disturbance and measurement noise can become large.

Minimum loss method

For a given control structure represented by the optimal combination matrix, H, the worst-case and average loss can be calculated using the methods formulated earlier. For finding a measurement combination, $\Delta c = H\Delta y$, that minimizes the average loss for normally distributed disturbance and measurement noise, the loss can be formulated as[26]:

$$\min_{H} L = \frac{1}{2}\left|\left|J_{uu}^{1/2}(HG^y)^{-1}HY\right|\right|_F^2, \tag{2.7.37}$$

where HG^y is invertible, and $J_{uu}^{1/2}(HG^y)^{-1}HY$ is the loss matrix, M. This formulation can be challenging to solve as it is often non-convex, and nonlinear and the assumption of exact measurements are not valid in practice. To solve this, Alstad presented a reformulation of this non-convex optimization problem, called the minimum loss method, where the non-uniqueness property of the combination matrix, H, was used to derive a formulation where the optimization problem is convex[31]:

$$\begin{aligned} &\min_{H}||HY||_F \\ &\text{s.t. } HG^y = J_{uu}^{1/2}, \end{aligned} \tag{2.7.38}$$

where an invertible matrix, Q, is chosen to make $HG^y = J_{uu}^{1/2}$, as it is proven earlier that multiplying the combination matrix, H, would not change the value of the loss matrix. This results in canceling the nonlinearity in the formulation shown in Equation (2.7.38), which now is a convex problem, and by solving this, the combination matrix, H, can be found. Furthermore, Alstad formulated an analytical solution to this problem[31] and was later simplified by Yelchuru and Skogestad[32], such that:

$$H = (G^y)^T(\tilde{F}\tilde{F}^T)^{-1}, \tag{2.7.39}$$

where this combination gives the locally best measurement combination for a given set of measurements, where the matrix $(\tilde{F}\tilde{F}^T)^{-1}$ is always invertible as long as the measurements are affected by noise. It is worth noting that to evaluate loss from the combination matrix, shown in Equation (2.7.39), the expressions derived for worst-case and average loss must be used.

Nullspace method

Alstad and Skogestad used the minimum loss method under the assumption of no measurement noise to formulate the nullspace method to find the optimal combination matrix, H[33]. In the formulation, it is also shown that as long as the number of measurements is equal or greater than the sum of the number of inputs and disturbances, $n_y \geq n_u + n_d$, it is always possible to find a combination matrix, H, by selecting H as the nullspace of F, such that:

$$HF = 0, \tag{2.7.40}$$

which results in no loss, both for the cases of average and worst-case. In a real plant, this will not give optimal results, as there will always be measurement noise in practice. This method also requires a large number of measurements to fulfill the condition $n_y \geq n_u + n_d$, which often is the case, but a necessary condition nonetheless.

Measurement subset selection

Using all measurements is neither desired nor required, and a control structure that controls only a subset of the measurements leads to a simpler control structure with a usually insignificant increase in loss[28] [30]. Thus, it is important to select an optimal subset of measurements and find the optimal trade-off between implementing more sensors in the system, which leads to better disturbance rejection and less corruption from measurement noise, and increased investments and control complexity that comes with implementing more sensors. Kariwala showed that by plotting the average loss against the number of measurements selected, the trend follows a Pareto frontier[34], and is shown in Figure 4.

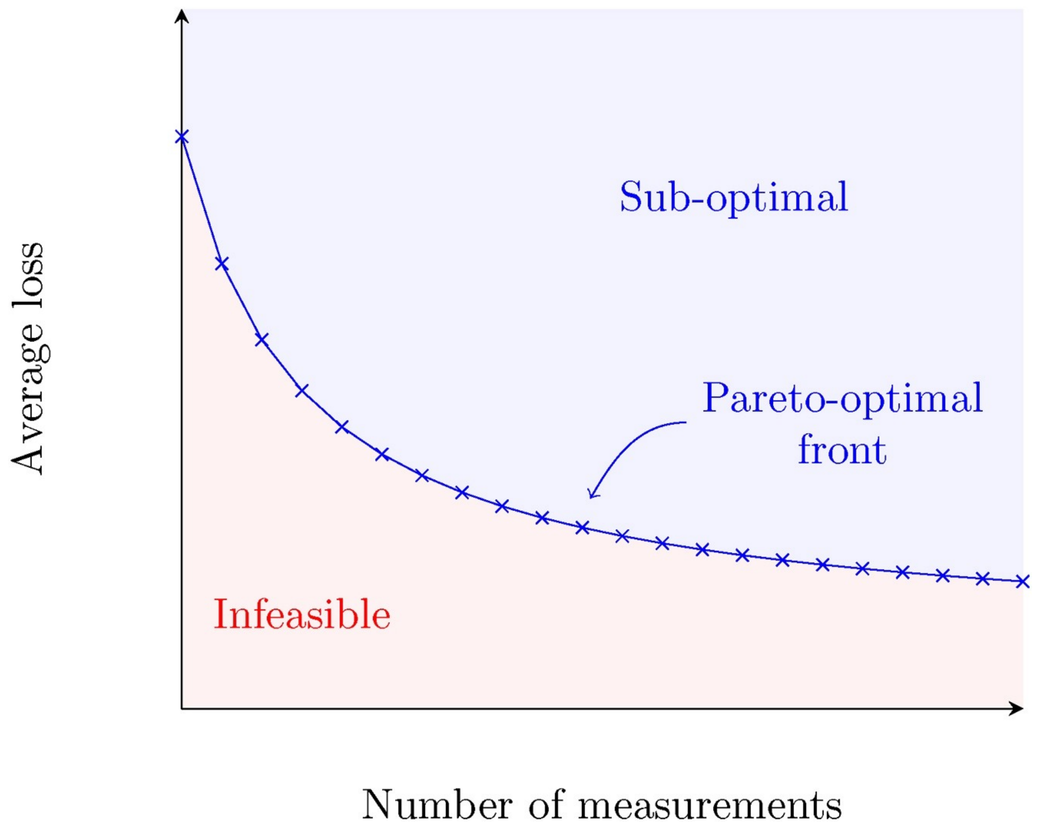

Figure 4: Average loss plotted against the number of measurements, with a Pareto frontier trend representing the optimal set of measurements, where the figure is taken from a self-optimizing control review article by Jäschke[35].

In Figure 4, it can be seen that after a certain threshold, an increasing number of measurements have an insignificant increase in loss and new sensors can be implemented where it affects the investment cost and control structure complexity by an inconsiderable amount.

The problem of selecting the best subset of measurements from the total amount of measurements leads to a combinatorial optimization problem, where the loss is evaluated individually for each CV. This can lead to a large number of possible combinations, and therefore, two approaches have been developed for finding the optimal subset of measurements. The first approach is developed by Cao and Kariwala and solves the problem of finding the optimal subset of measurements by developing a tailor-made branch and bound algorithms[30] [34] [36]. The second approach is developed by Yelchuru and Skogestad, where the problem of selection is formulated as a mixed integer quadratic optimization problem (MIQP) and uses standard MIQP to achieve the optimal subset of measurement set[32]. However, in this project, a branch and bound algorithm developed by Yi Cao will be used to select the best measurement subset selection[37], in addition, good engineering judgment to find the best subset of measurements will also be used. The comprehensive MIQP approach will not be used. This is further elaborated in section 4.3.4.

2.7.3 Active set changes

When defining inequality constraints in the model, it is important to note the difference between active constraints and inactive constraints. When finding an optimal solution, the inequality constraints may or not be at their bound. If it is at their bound, it is an active constraint and it will affect the optimal solution, as the bounds will "limit" the solution space of the solution. When these active constraints either change or are removed, the optimal solution will also change. In general, the set of active constraints may change during operation due to the disturbances, where an illustration of this situation is shown in Figure 5, where for disturbance, d_1, the optimum lies within the constraint and there lies within the feasible region, g. For disturbance, d_2, the optimum is unconstrained and therefore outside of the feasible region. Another important note is the fact that all degrees

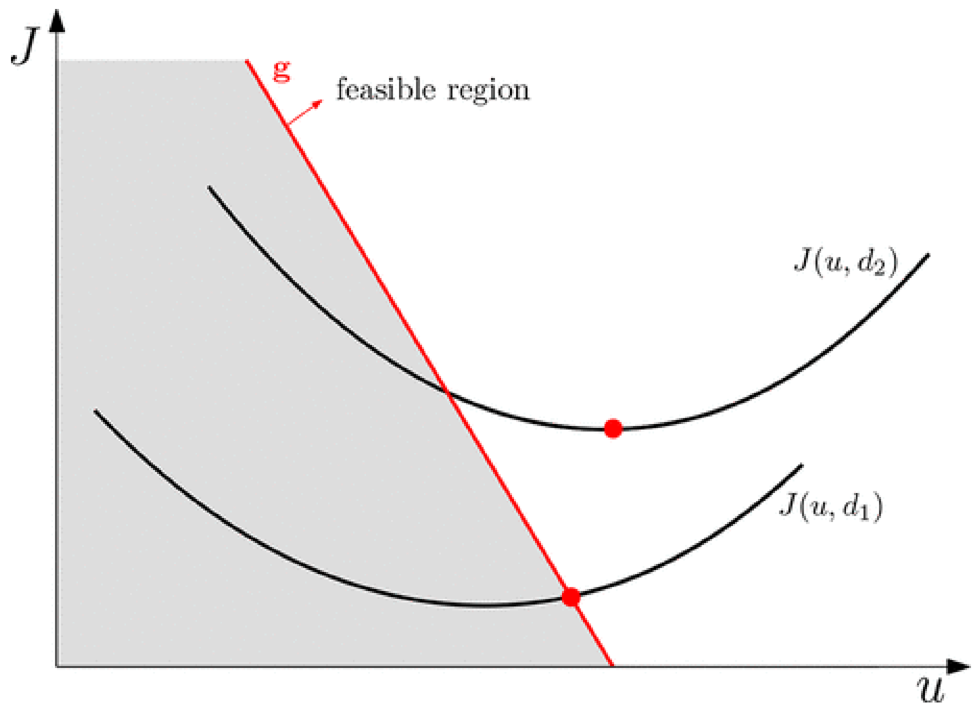

Figure 5: Illustration of how the disturbances can affect the active constraint set[38], where objective function J is a function of the input, u and disturbances, d.

of freedom are used when a control structure is implemented in the system ($\Delta c = H\Delta y$). That means that if an additional constraint becomes active, this needs to be controlled, but since there are no degrees of freedom available, this will lead to constraint violation and infeasibility. On the other hand, if an active constraint becomes inactive, this will result in economic performance loss. There are approaches to handle the problem of active constraint set changes, however, these approaches are not considered in this work.

The first approach is the multi-parametric programming approach developed by Manum and Skogestad[39]. In this approach, a self-optimizing control structure is designed by decomposing the disturbance space into different active set regions, where specific switching laws to change between control structures are implemented. As the number of potential active set regions increases exponentially with the number of constraints, this is well suited for large problems. Another approach is developed by Hu, where the goal is to find a single feasible control structure that minimized the average loss for all disturbances[40]. A cascade approach can be used for the case where it is predicted that the active set will not change frequently, and the number of constraints is less than the number of CVs[41]. The problem of handling active set changes is still not quite solved and is one of the problems in self-optimizing control that is open for further research[26].

3 Process description

Today's blue hydrogen production is based onshore, but when designing and studying an offshore process, there are several factors that take effect, such as limiting constraints for weight and space, and higher establishment costs. A typical blue hydrogen plant uses a large furnace to perform the reforming part of the process, where methane converts to hydrogen and CO_2. When designing an offshore plant, such a furnace would have increased the need for special requirements drastically. Instead of utilizing a furnace, this process will take the basis of a method that the company *Johnson Matthey* (JM) has designed. In this method, the furnace for reforming is replaced with a gas-heated reformer (GHR) and an autothermal reformer (ATR), both of which are columns and do not require as much space as a furnace. The downside of this process is the requirement of pure oxygen in the ATR unit, which can be challenging to obtain in an offshore process. However, this process has reported high hydrogen purity products ($>$99.9 mol%) with low carbon emission with 97% CO_2 capture rate[16], where a pressure swing adsorption unit is the final separation unit.

A more detailed description of the process is presented in section 3.1. A simplified figure of the offshore process is shown in Figure 6, where natural gas is extracted from the gas reservoir under the seabed to the process and there will be mainly two components before the last separation unit, which is hydrogen and CO_2. Hydrogen will be separated and transported back onshore, where it will be sold as a product while CO_2 and a small amount of other components such as H_2O and CO will be injected back into the gas reservoir. This is beneficial both for production and the environment as the carbon extracted from the seabed returns back to where it was, so that the net amount of carbon in the atmosphere remains the same, and at the same time, the pressure in the gas reservoir is held at a desired pressure level. This is beneficial as usually when oil and gas are extracted from a reservoir, not all resources can be extracted due to pressure drop in the reservoir. This process of injection of CO_2 for extracting more oil and gas is also called Enhanced Oil Recovery (EOR).

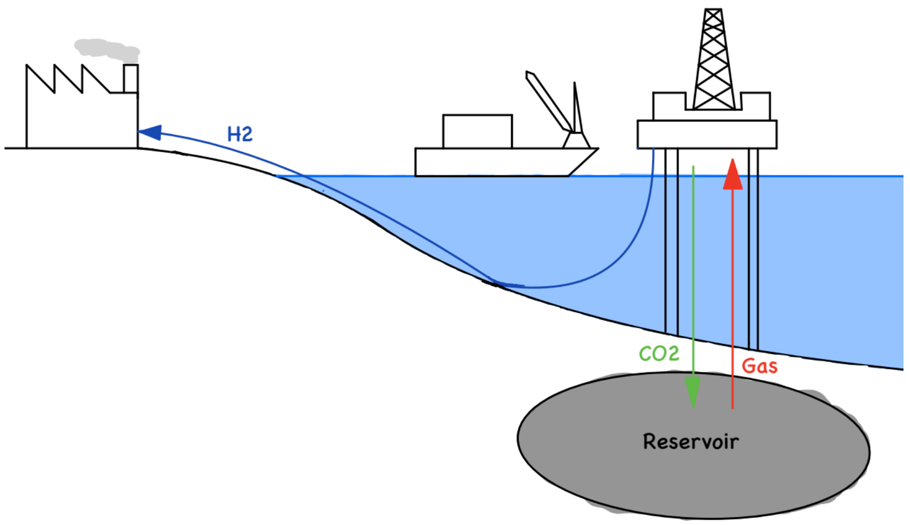

Figure 6: A simplified figure of the offshore blue hydrogen process.

3.1 Flowsheet

The flowsheet in the present thesis was developed in the specialization project[14]. The flowsheet is nearly identical, but unlike in the specialization project, this project has implemented a fuel-switching heater unit and a compressor for compressing the injection gas. The inlet stream and the product stream are therefore split as fuel for the heater unit. Lastly, the process condensate unit is modeled as a flash tank instead of assuming ideal split factors.

A complete flowsheet of the process is shown in Figure 7. The flow rate and the composition of the extracted natural gas are based on the Troll platform in the North Sea. This natural gas is assumed to be purified from sulphuric compounds before entering the process as the initial stream. Then the purified natural gas mixes with H_2O to a certain steam-to-carbon ratio and enters a heater to achieve the operating temperature of the pre-reformer before entering it. After the pre-reformer, the stream is heated up again, but now to the operating temperature of the gas-heated reformer. The outflow of the gas-heated reformer enters the autothermal reformer, where pure oxygen is fed into the unit and gives off large amounts of heat. This highly heated stream is returned to the gas-heated reformer unit where it exchanges heat with the inlet stream of the gas-heated reformer in a cross-counter flow direction. After the GHR and ATR reforming part, it enters the isothermal shift reactor. In all reactors, the operating temperatures have been set to achieve near equilibrium and it is assumed in this project that all reactors are at equilibrium. This is a good assumption as the process is assumed to be at steady state. Finally, it enters the first separator unit, the process condensate, where water is removed after cooling down the stream, and the remaining components, CO_2 and hydrogen are separated at the PSA unit. Hydrogen is transported back onshore to be sold, while CO_2 is being injected back into the reservoir below the seabed after being compressed to the proper pressure level. In addition, to supply the process with its required amount of heat, a small fraction of the natural gas stream and the hydrogen product stream are used for combustion in a heater.

3.1.1 Natural gas feed stream

The initial flow or stream 1 in the process, is the purified natural gas feed stream, which is based on the Troll platform in the North Sea. The flow rate of the initial flow is based on the average amount of natural gas that the Troll platform extracts annually, while the composition is set as the average composition of natural gas from North Sea gas reservoirs[42]. A table of the average composition of natural gas in the North Sea is shown in Table 1.

As mentioned, the initial feed is assumed to be purified, where all sulphuric and nitric compounds are purified and the initial flow mainly consists of CH_4, CO_2, and other heavier hydrocarbons like ethane and propane. The assumption that all heavier hydrocarbons than pentane will be treated as pentane has been made for this project. Sulphuric compounds are treated by adding hydrogen to the natural gas stream to convert it to H_2S and remove it. Although there are only traces of sulphuric compounds, these compounds damage the catalyst in the reactors and can lead to significant economic losses and lower yield. In addition, it is worth noting that a small part of the natural gas stream will be used for combustion to achieve the required amount of heat in the process. This optimal amount of

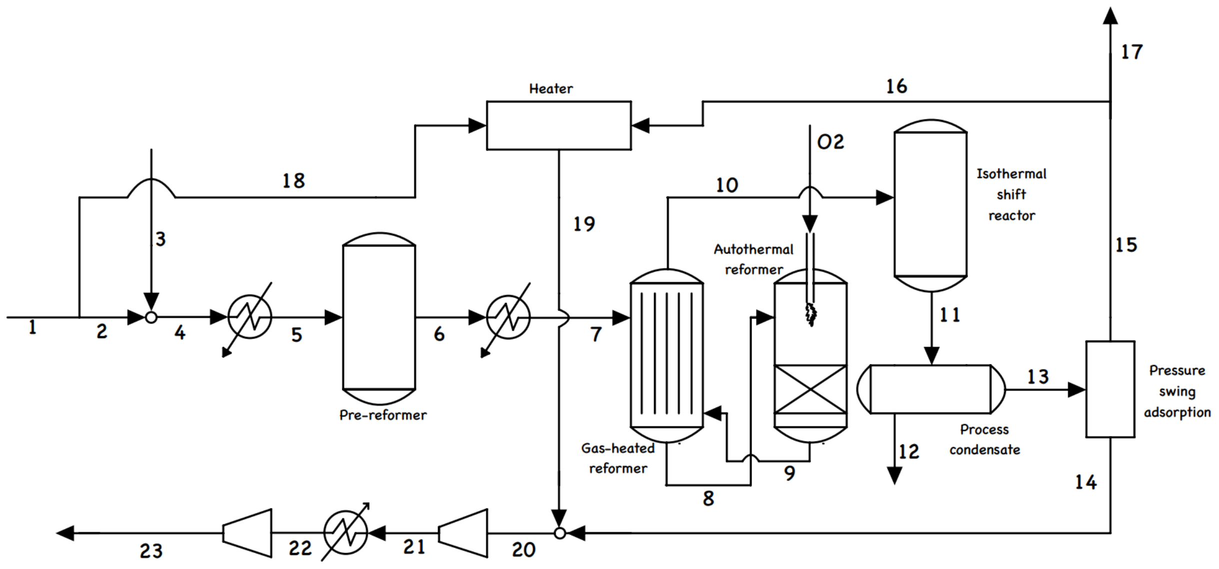

Figure 7: A complete flowsheet of the blue hydrogen process.

Table 1: Average composition of natural gas in the North Sea[42].

Compound	mol%
CH_4	78.27
C_2H_6	6.10
C_3H_8	6.70
n-C_4H_{10}	2.48
i-C_4H_{10}	1.41
C_5H_{12}	3.70
H_2O	≈ 0
H_2	≈ 0
CO	≈ 0
CO_2	1.34
H_2S	≈ 0

natural gas used for combustion will be decided by the optimization solver.

3.1.2 Pre-reformer

A small part of stream 1 will be used as fuel for combustion in the heater (stream 18), which will be explained in section 3.1.7. The purified natural gas, stream 2, mixes with a stream of pure H_2O with a specific steam-to-carbon ratio, where carbon is the sum of all hydrocarbons in stream 2. In this project, a steam-to-carbon ratio has been set to 2.5. After the mixer, stream 4 will enter a heater to heat up stream 4 to the operating temperature of the pre-former unit. This amount of heat will be supplied from the heater, which is explained in more detail in section 3.1.7. The purpose of the pre-reformer unit is to convert all heavier hydrocarbons than methane into CH_4. This is beneficial as it will be easier to operate and select the correct catalysts in the later reforming stage. In the pre-reformer, the following equilibrium takes place:

$$C_nH_m + nH_2O \rightleftharpoons nCO + (n + \frac{m}{2})\, H_2, \quad \Delta H > 0 \tag{3.1.1}$$

$$CH_4 + H_2O \rightleftharpoons CO + 3\, H_2, \quad \Delta H > 0 \tag{3.1.2}$$

where the two equilibrium equations are also called the steam reforming equations[43]. To ensure that the equilibrium is shifted as desired, a desirable operating temperature is chosen. A study by Christensen reported that 693.0 K is a good operating temperature[44] for the pre-reformer unit for achieving equilibrium, and this project will make the same assumption. Due to the conditions, the first equilibrium, shown in Equation (3.1.1), will be shifted heavily towards the right so that all heavier hydrocarbons are converted to CO and H_2. The second equilibrium, shown in Equation (3.1.2), is shifted to the left so that the converted hydrocarbons will convert to CH_4. In summary, in the pre-reformer unit, all hydrocarbons that are heavier than methane in the entering stream will be converted to CO and H_2 and then converted to CH_4 and H_2 again.

3.1.3 Gas-heated reformer and autothermal reformer

After the pre-former step, the next step in the process is the main reforming part, which will happen in the gas-heated reformer and the autothermal reformer part. However, before this step, the outlet of the pre-reformer (stream 6) enters another heater to heat up the stream to 753.0 K. Although a temperature range of 923 K to 1023 K is reported to be a good operating temperature to achieve near equilibrium[43], a temperature of 753.0 K is assumed to be sufficient, as the chemical reactions happening in the reformer units are strongly exothermic. In addition, heat from the autothermal unit will also heat up the gas-heated reformer unit, which will be explained in section 3.1.7. This required amount of heat in the heat exchanger before the gas-heated reformer will also be supplied by the heater, identical to the heat exchanger before the pre-reformer unit.

The heated stream 7, will then enter the gas-heated reformer (GHR), where it will pass through a number of catalyst-filled pipelines and two reactions will take place. The first reaction will convert CH_4 and H_2 into CO and H_2, and the second reaction will convert CO and H_2 into CO_2 and H_2 and is shown in Equation (3.1.2) and Equation (3.1.3), respectively.

$$CO + H_2O \rightleftharpoons CO_2 + H_2, \quad \Delta H < 0, \tag{3.1.3}$$

where Equation (3.1.3) is also called the water-gas shift reaction. After the stream has passed the pipelines in the GHR unit, the outlet stream of GHR will enter the autothermal reformer (ATR) unit where pure oxygen is fed into the reactor. The oxygen combusts with either H_2 or CH_4 to give a high amount of heat. A study by Mukherjee et al. has shown that the combustion of H_2 happens almost 10 times as fast as CH_4, and it will be assumed that oxygen mainly combusts with H_2[45]. This high amount of heat from combusting oxygen will shift Equation (3.1.2) heavily towards the right, as the reaction is endothermic, such that most of the CH_4 is converted to CO and H_2.

The outlet stream of the ATR unit will return back to the GHR unit, only now, instead of entering the pipelines in the GHR, the outlet stream will pass outside of the pipeline, but still in the GHR unit, to exchange heat to the initial inlet stream of the GHR. This is beneficial as the conversion of CH_4 in the pipelines in the GHR unit will increase as it shifts the equilibrium to the right. This exchange of heat is similar to a tube and shell heat exchanger, where counter-flow heat exchange is taking place. Stream 10, which is the outlet stream of the GHR and ATR process, will now mainly consist of CO, CO_2, H_2, and H_2.

3.1.4 Isothermal shift reactor

After the GHR and ATR process, stream 10 will enter the isothermal shift reactor unit (ITSR). The purpose of this unit is to convert the remaining CO into CO_2 and H_2 and is the last reactor unit of the process. Due to the reactor conditions and the catalysts in the reactor, only the water-gas shift reaction, shown in Equation (3.1.3), will take place. It is worth noting that since the temperature is set to be constant in the ISTR, heat is recovered, as the reaction is exothermic. In addition, it is assumed that a cooler takes place in the ISTR unit to cool down the outlet stream from the GHR and ATR process to the operating temperature of the ITSR. A typical operating temperature of ITSR is reported to be in a range of 473.0 K to 623.0 K from a study by Appl[46]. The outlet stream of the ITSR process consists mainly of CO_2, H_2, and H_2O.

3.1.5 Process condensate

After the ITSR process, stream 10 will enter the process condensate unit. The process condensate's role is to remove most of H_2O in the stream and is done by first cooling down stream 11 to 313.0 K. The cooled stream will then enter the process of condensate where a vapor-liquid separation will take place. This unit is also one of the simpler and cheaper options for post-combustion separation[47]. This step is also important for the next step, as liquid H_2O cannot exist in the next separation unit, which is the pressure swing adsorption unit. The outlet stream of the process condensate process will now mainly consist of CO_2 and H_2, and the removed liquid H_2O can be recycled for earlier stages of the process by heating it up to steam and adjoining with stream 3.

3.1.6 Pressure swing adsorption

The last separation unit of the process is the pressure swing adsorption (PSA) unit. The purpose of this unit is to separate H_2 from the rest of the components in the stream, mainly CO_2. It is also assumed that all components in the stream after the PSA unit, except H_2

and CO_2, will be treated as CO_2. The PSA unit consists of two or more tanks, which are filled with particles that CO_2 adsorbs on when operating at high pressure, so that H_2 can pass through. When a tank filled with particles is saturated with CO_2, the pressure lowers, and the CO_2 desorbs from the particles. This process is similar to a batch process, where a tank has high pressure and adsorbs the CO_2, while another tank has low pressure to desorb the CO_2 to achieve a streamlined, continuous process[48].

This process has been reported to achieve a high-purity hydrogen stream with a purity rate of 99.9 mol% with a high CO_2 capture rate of 97 mol%[49], that will be transported back onshore where it can be sold as a product (stream 17). A part of the product stream of H_2 (stream 16) will also be used for combustion for the heater to achieve the required amount of heat that the process requires. An optimization solver will find the optimal amount of burning the product stream and the optimal amount of natural gas to combust while maximizing profit. The purge stream from the PSA unit, consisting mainly of CO_2 (stream 14) and other unreacted components will adjoin the purge stream from the heater (stream 19), where it will be compressed for injection to the gas reservoir to complete the enhanced oil recovery part of the process[50].

3.1.7 Heater

As mentioned in the previous sections, two heaters take place in the process to heat up the pre-reformer unit's and the gas-heated reformer's inlet stream to their respective operating temperature. To achieve this required amount of heat from the two heaters, an additional heater unit that suffices the system with heat has been implemented in the process. The heater combusts a part of the natural gas feed stream (stream 18), and the hydrogen-rich product stream (stream 16) until a desired amount of heat is achieved. Both methods will results in a loss of revenue as using a part of stream 1, less hydrogen in the form of hydrocarbons will enter the system, and combusting hydrocarbons leads to more CO_2 gas that has to be compressed and injected and using a part of the hydrogen-rich product stream leads to less product being sold. The optimization solver will find the optimal amount of natural gas or hydrogen to combust to achieve the required amount of heat in the process while maximizing profit.

The amount of CO_2 generated from combusting hydrocarbons in the natural gas feed stream can be calculated from the combustion reaction shown in Equation (3.1.4).

$$\mathrm{C}_n\mathrm{H}_m + (\mathrm{n} + \frac{\mathrm{m}}{4})\mathrm{O}_2 \longrightarrow \mathrm{nCO}_2 + \frac{\mathrm{m}}{2}\,\mathrm{H}_2\mathrm{O}. \tag{3.1.4}$$

When supplying the combustion with oxygen, unlike the ATR unit which used pure oxygen, it is assumed that air, containing 21% O_2 and 79% N_2, of a satisfying amount is sufficient to achieve complete combustion. To calculate how much N_2 that comes from the air that is being used for combustion to inject, the equation shown in Equation (3.1.5) will be used, where it uses the molar ratio between O_2 and N_2 and the molar ratio between a general hydrocarbon, C_nH_m, and O_2, where n and m is the number of carbons and hydrogen atoms in the hydrocarbon that is being combusted, respectively.

$$F_{\mathrm{N}_2,inj} = \frac{0.21 * \cdot F_{NG,heat}}{2 * 0.79}, \tag{3.1.5}$$

where $F_{\mathrm{N}_2,inj}$ is the amount of N_2 that is generated from combustion that is being injected and $F_{NG,heat}$ is the amount of natural gas feed stream that is being combusted for generating the required amount of heat in the process. As there are mostly CH_4 in the natural gas, n and m are assumed to be 1 and 4, respectively, for all combustion of natural gas.

3.1.8 Compression

To inject the CO_2 gas from the PSA unit and combustion, the gas needs to be compressed to a pressure level higher than the reservoir. A study by Zhang et al. showed that injecting gas into a reservoir at the Sleipner gas field needed to compress to a pressure level of 106 bar[51]. The Sleipner gas field is approximately 2300 meters below sea level[52], while the Troll gas field, which this project is based on, is approximately 1300-1600 meters below sea level[53]. A deeper-positioned gas field requires the gas to be injected to be compressed to a higher pressure. It is therefore assumed that for this project, a pressure level of 100 bars will be sufficient for injecting the CO_2 to the reservoir to complete the enhanced oil recovery.

Firstly, it is assumed that the CO_2 gas from the PSA unit and combustion has 1 bar as initial pressure. To compress this gas to 100 bar there will be two compressors that perform an isentropic adiabatic compression with a cooler in between the compressors. The reason behind the choice of a two-way compression with a cooler in between is because when compressing gas from 1 bar to 100 bar, the required compression work is much higher compared to compressing it once, cooling down the stream and then compressing it again. Even though increasing the number of compressors will reduce the total compression work, when operating offshore, fewer units are an important factor to consider as it can be more expensive and complicated when operating with many units. Note that the first compression and the second compression compress the gas with the same compression factor of 10.

4 Model description

Several assumptions were made to model the process, in addition to the assumptions that were made in the previous sections. When modeling the offshore blue hydrogen plant, no dynamics were implemented. In other words, the model is assumed to be at steady-state. As mentioned in section 2.6, a steady-state model is sufficient when performing a top-down analysis to analyze the economics of the plant. The optimization problem is an economic objective function that maximizes profit and constraints formulated as nonlinear algebraic equations and is explained in more detail in section 4.2 and section 4.1, respectively. To estimate the flow rate of the initial natural gas stream in the process plant model, stream data from *Norsk Petroleum* were used, where the platform *Troll* in the North Sea was chosen as it extracted most natural gas annually in the last decade[54]. It was assumed from the stream data that the platform produced an amount of 35 million Sm^3 annually, which is calculated to an hourly rate of approximately $4000\,Sm^3$ of natural gas, assuming that the platform is operating without interruption. The density of the natural gas is assumed to be $0.829\,kg/Sm^3$. Finally, by using the natural gas density to convert the natural gas stream data to kg, and using the natural gas composition found in Table 1, the initial natural gas stream used in the model was calculated to be approximate $145.4\,kmol\,h^{-1}$.

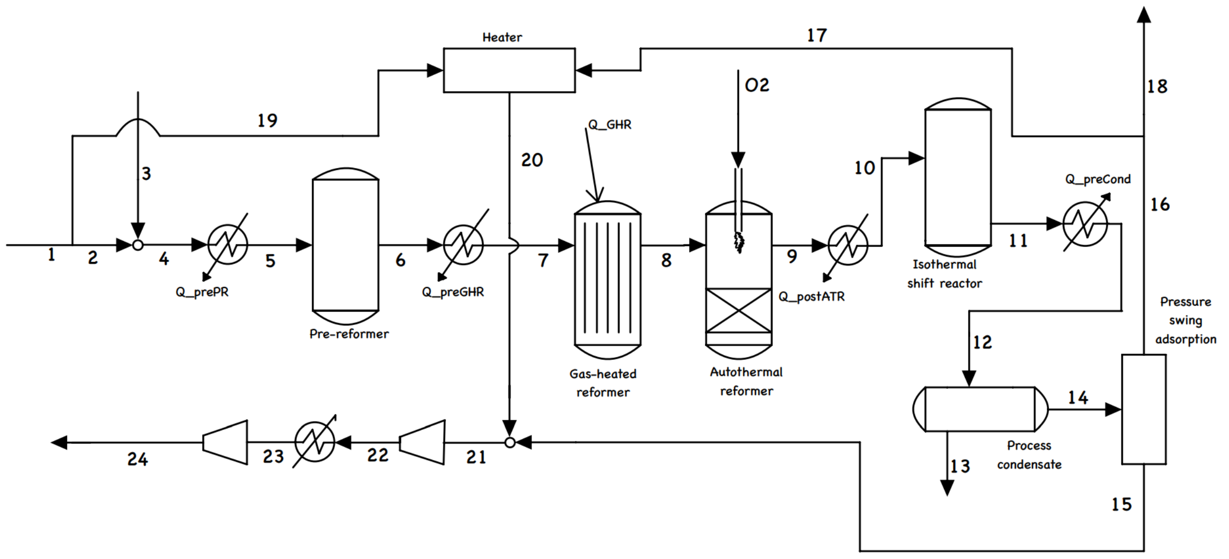

Figure 8: Flowsheet of the process plant model.

The flowsheet that the model is based on is a modified version of the initial flowsheet that was shown in Figure 7. The modification comes from implementing the heat exchange between the catalyst-filled pipelines in the GHR unit and the heated outlet from the ATR unit. Even though the GHR unit is adiabatic, a new dummy variable, Q_{GHR}, has been introduced, which represents the amount of heat that the pipelines in the GHR unit receive from the outlet of the ATR unit. Instead of the ATR outlet entering the GHR unit, it enters a cooler which cools down the ATR outlet stream to the same temperature as stream 10 shown in Figure 7. The heat released from the cooler after the ATR unit is then $Q_{postATR}$, and has the same amount of energy as Q_{GHR} but with the opposite sign. A further explanation of the GHR and ATR part of the model is explained in section 4.2.2. Otherwise, the new flowsheet is identical to the one shown in Figure 7, but only with new stream numbers as the new implementation created an additional stream. The new flowsheet that the model is based on is shown in Figure 8. All code that has been used for building the model, submodel, calculation functions, printing functions, simulation functions, and testing functions is shown in appendix B with descriptions.

4.1 JuMP Julia model structure

There were created 12 submodels in JuMP Julia, which represent different control volumes of the process and were assembled into a larger model at the end. To connect the submodels, connectivity constraints were implemented. In other words, constraints requiring the output stream of one submodel to be equal to the input stream of the next submodel were implemented. The larger assembled JuMP Julia model was then solved in a simultaneous approach. To study the process in an easier way, a function that prints out mole, composition, temperature, and pressure in a stream table was created. In addition, the function also prints out other relevant variables in the process and a mass stream table to ensure that mass is conserved at all points of the process.

To solve the model, an interior point method, Ipopt, interfaced by JuMP in Julia was used[55] [56]. It is worth noting that the submodel that models the heater unit in the process does not have its own Julia model, but is implemented directly into the larger model file where other submodels were assembled. In each submodel, component mole balance defined in Equation (2.3.6), is implemented. The extent of reaction term simply becomes zero if there are no reactions taking place, such as mixers, heat exchangers, and splitters. The variables are first defined in each submodel and then assigned their initial value for optimizing, where the initial value is the calculated steady-state nominal value[14]. Energy balance defined in Equation (2.3.17) is also implemented for each submodel, where the Q term becomes zero if there is no heat gained or lost, such as mixer, pre-reformer, process condensate, or PSA unit.

In addition, a function that prints out the active constraints has been made, where it takes in the variables of the optimized model and checks if certain specific freed-up constraints are active or not. To calculate the enthalpy for the energy balance, a function that calculates a vector of enthalpy for each component in the stream for a given temperature has been created. To calculate and estimate the enthalpy for each component, the equation shown in Equation (4.1.1) has been used to estimate the heat capacity, where the constants in the equation are from Smith et al[57]. The estimated heat capacity is then integrated from $T_0 = 298\,\mathrm{K}$ to the temperature of the stream i, T_i, to achieve a term for enthalpy for a specific temperature, T_i. The final term for enthalpy, H_i, is shown in Equation (4.1.1)

$$\frac{C_p}{R} = A + B \cdot T + C \cdot T^2, \tag{4.1.1}$$

$$\int_{T_0=298K}^{T_i} \frac{C_p}{R} = A + B \cdot T + C \cdot T^2 \tag{4.1.2}$$

$$H_i = R \cdot \left[A \cdot T + \frac{B}{2} \cdot T^2 + \frac{C}{3} \cdot T^3 \right]_{T_0=298K}^{T_i}. \tag{4.1.3}$$

A summary of the description of each submodel with each input and output to clarify which control volume each submodel represents is shown in Table 2, where n_i is the respective mole stream for stream i. In the end, there were 192 variables with 184 constraints: 182 equality constraints and 2 inequality constraints. The objective function for the optimization problem is an economic objective function and is explained more in detail in section 4.2.

Table 2: Summary of the description of all models.

Submodel	Inlet	Outlet	Control Volume
1Mix.jl	n_1, n_2	n_3	Mixer for streams 1 and 2 and calculate the required amount of steam
2PrePR.jl	n_3	n_4	Heater before pre reformer
3PR.jl	n_4	n_5	Pre reformer
4PreGHR.jl	n_5	n_6	Heater before GHR
5GHR.jl	n_6	n_7	Gas heated reformer
6ATR.jl	n_7	n_8	Autothermal reformer
7PostATR.jl	n_8	n_9	Cooler after ATR
8ITSR.jl	n_9	n_{10}	Isothermal shift reactor
9PreCondensate.jl	n_{10}	n_{11}	Cooler before process condensate
10Condensate.jl	n_{11}	n_{12}, n_{13}	Process condensate
11PSA.jl	n_{12}	n_{14}, n_{15}	Pressure swing adsorption
-	n_{12}	n_{12}	Heater

4.2 Objective function

The optimization objective function in this project will be to maximize revenue from hydrogen while minimizing production costs such as electrical work from compressing gas for injection. The primary and only source of income will be the hydrogen product, and the price will be estimated from an external source. The main costs in the model are the cost of injecting flue gas and the removed CO_2 gas back into the oil reservoir, which is also estimated, and the required heating achieved by combusting a combination of the natural gas feed stream and the hydrogen product stream. The equation for the objective function, which is being maximized, is shown in Equation (4.2.1).

$$\max J = F_{\mathrm{H}_2,prod} \cdot P_{\mathrm{H}_2} - W_{s,tot}^{rev} \cdot P_{el}, \tag{4.2.1}$$

where $F_{\mathrm{H}_2,prod}$ is the hydrogen product stream that is being sold, P_{H_2} is the selling price of hydrogen, $W_{s,tot}^{rev}$ is the total compression work for compressing the injection gas to a sufficient enough high pressure for injection, formulated in section 4.2.7, and P_{el} is the electricity cost. A table of these prices is shown in Table 3,

Table 3: Estimated prices of hydrogen and electricity.

Variable	Price	Unit
P_{H_2}	3.347	\$/kgH$_2$
P_{el}	0.14	\$/Kwh

where the price of hydrogen, P_{H_2}, was estimated from a study that did a comparative assessment for different blue hydrogen plants[58] and the price of injecting the purge stream, P_{el}, was estimated from Statistics Norway[59], which reported the average electricity cost for a non-electric intensive industry. It is worth noting that these two prices will affect the optimization and the objective function value, and is considered as a disturbance for the process, which will be used for the self-optimizing control later, explained in section 4.3.2.

4.2.1 Pre-reformer model

The model equations for the pre-reformer unit are derived from the component balances and energy balances, shown in Equation (2.3.8) and Equation (2.3.18), respectively. And it is worth noting that as mentioned earlier, it is assumed that all hydrocarbons heavier than pentane will be considered as pentane. The equilibrium reaction equations taking place in the pre-reformer are then:

$$\mathrm{CH_4 + H_2O \rightleftharpoons CO + 3\,H_2} \tag{4.2.2}$$
$$\mathrm{C_2H_6 + 2\,H_2O \rightleftharpoons 2\,CO + 5\,H_2} \tag{4.2.3}$$
$$\mathrm{C_3H_8 + 3\,H_2O \rightleftharpoons 3\,CO + 7\,H_2} \tag{4.2.4}$$
$$\mathrm{i\text{-}C_4H_{10} + 4\,H_2O \rightleftharpoons 4\,CO + 9\,H_2} \tag{4.2.5}$$
$$\mathrm{n\text{-}C_4H_{10} + 4\,H_2O \rightleftharpoons 4\,CO + 9\,H_2} \tag{4.2.6}$$
$$\mathrm{C_5H_{12} + 5\,H_2O \rightleftharpoons 5\,CO + 11\,H_2} \tag{4.2.7}$$
$$\mathrm{CO + H_2O \rightleftharpoons CO_2 + H_2.} \tag{4.2.8}$$

Due to the reactor conditions such as temperature and catalysts in the pre-reformer, it is assumed that all equilibrium reactions that are heavier than methane is shifted heavily towards the right, such that the extent of the reaction for these reactions can be calculated from the inlet mole stream since the outlet mole stream will become 0. For the equilibrium reaction for ethane, the component balance for ethane can be formulated as:

$$n_{\mathrm{C_2H_6}} = n_{0,\mathrm{C_2H_6}} - \xi_{1,pr}, \tag{4.2.9}$$

where $n_{0,\mathrm{C_2H_6}}$ and $n_{\mathrm{C_2H_6}}$ is the inlet and outlet mole stream of ethane, respectively, and $\xi_{1,pr}$ is the extent of reaction for Equation (4.2.3). As it is assumed that the equilibrium reaction is shifted heavily towards the right, the outlet mole stream of ethane becomes zero. The component balance for ethane then becomes:

$$0 = n_{0,\mathrm{C_2H_6}} - \xi_{1,pr} \tag{4.2.10}$$
$$\xi_{1,pr} = n_{0,\mathrm{C_2H_6}}. \tag{4.2.11}$$

By using the same logic for the rest of the hydrocarbons heavier than methane, the extent of reactions in the pre-reformer becomes:

$$\xi_{2,pr} = n_{0,\mathrm{C_3H_8}} \tag{4.2.12}$$
$$\xi_{3,pr} = n_{0,\mathrm{i\text{-}C_4H_{10}}} \tag{4.2.13}$$
$$\xi_{4,pr} = n_{0,\mathrm{n\text{-}C_4H_{10}}} \tag{4.2.14}$$
$$\xi_{5,pr} = n_{0,\mathrm{C_5H_{12}}}. \tag{4.2.15}$$

The remaining extent of reactions of the pre-reformer equilibrium reaction equations can then be calculated from the component balance of CH_4 and CO_2, respectively.

$$n_{\mathrm{CH_4}} = n_{0,\mathrm{CH_4}} - \xi_{6,pr} \tag{4.2.16}$$
$$n_{\mathrm{CO_2}} = n_{0,\mathrm{CO_2}} + \xi_{7,pr}, \tag{4.2.17}$$

where Equation (4.2.16) and Equation (4.2.17) is the component balance for CH_4 and CO_2 in the pre-reformer, respectively. By rearranging the component balances, the remaining extent of reactions of Equation (4.2.2) and Equation (4.2.8), can then be formulated as:

$$\xi_{6,pr} = n_{0,\mathrm{CH_4}} - n_{\mathrm{CH_4}} \tag{4.2.18}$$
$$\xi_{7,pr} = n_{\mathrm{CO_2}} - n_{0,\mathrm{CO_2}}. \tag{4.2.19}$$

When defining the extent of reactions in the model for the solver, the equations are defined as explicit expressions in the model and not a variable that is being optimized directly. The remaining component balances for H_2O, H_2 and CO is derived from Equation (2.3.8), and is formulated as:

$$n_{H_2O} = n_{0,H_2O} - 2\xi_{1,pr} - 3\xi_{2,pr} - 4\xi_{3,pr} - 4\xi_{4,pr} - 5\xi_{5,pr} - \xi_{6,pr} - \xi_{7,pr} \tag{4.2.20}$$
$$n_{H_2} = n_{0,H_2} + 5\xi_{1,pr} + 7\xi_{2,pr} + 9\xi_{3,pr} + 9\xi_{4,pr} + 11\xi_{5,pr} + 3\xi_{6,pr} + \xi_{7,pr} \tag{4.2.21}$$
$$n_{CO} = n_{0,CO} + 2\xi_{1,pr} + 3\xi_{2,pr} + 4\xi_{3,pr} + 4\xi_{4,pr} + 5\xi_{5,pr} + \xi_{6,pr} - \xi_{7,pr}. \tag{4.2.22}$$

The steam-methane reforming and water-gas shift reactions, shown in Equation (4.2.2) and Equation (4.2.8), respectively, are not assumed to be shifted towards the right, but in equilibrium. The ratio between the reactant and product moles is given from an equilibrium constant and is dependent on temperature. A general equation for the equilibrium constant is shown in Equation (4.2.23).

$$K(T) = \prod_i x_i^{\nu_i}. \tag{4.2.23}$$

where $K(T)$ is the temperature dependent equilibrium constant for the equation, x_i is the mole fraction component i, and the definition of mole fraction is shown in Equation (4.2.24). And lastly, ν_i is the stoichiometric coefficient of component i.

$$x_i = \frac{n_i}{n_{tot}}. \tag{4.2.24}$$

By using Equation (4.2.23) for the reactions shown in Equation (4.2.2) and Equation (4.2.8), the equations for equilibrium constants can be formulated as:

$$K_{SMR}(T) = \frac{x_{CO} \cdot x_{H_2}^3}{x_{CH_4} \cdot x_{H_2O}} \tag{4.2.25}$$
$$K_{WGSR}(T) = \frac{x_{CO_2} \cdot x_{H_2}}{x_{CO} \cdot x_{H_2O}}, \tag{4.2.26}$$

where $K_{SMR}(T)$ and $K_{WGSR}(T)$ are the temperature-dependent equilibrium constants for the steam-methane reforming and water-gas shift reactions, respectively. To estimate the function for the actual equilibrium constants, a GHR reactor was simulated for a temperature range of $700-1400$K in the flow simulation software, *Aspen HYSYS*. A linear regression was then performed to estimate the functions for equilibrium constants. The resulting regression plot is shown in Figure 9, and the estimated equilibrium constant equations for the steam-methane reforming and water-gas shift reactions are shown in Equation (4.2.27) and Equation (4.2.28), respectively.

$$K_{SMR}(T) = \exp\left(-1.52 \cdot 10^{-5}T^2 + 4.80 \cdot 10^{-2}T - 29.19\right) \tag{4.2.27}$$
$$K_{WGSR}(T) = \exp\left(-2.95 \cdot 10^{-6}T^2 - 8.47 \cdot 10^{-3}T - 4.99\right). \tag{4.2.28}$$

Lastly, the energy balance of the pre-reformer unit was derived from Equation (2.3.18), where enthalpy is defined as the mole stream of the component multiplied by the specific

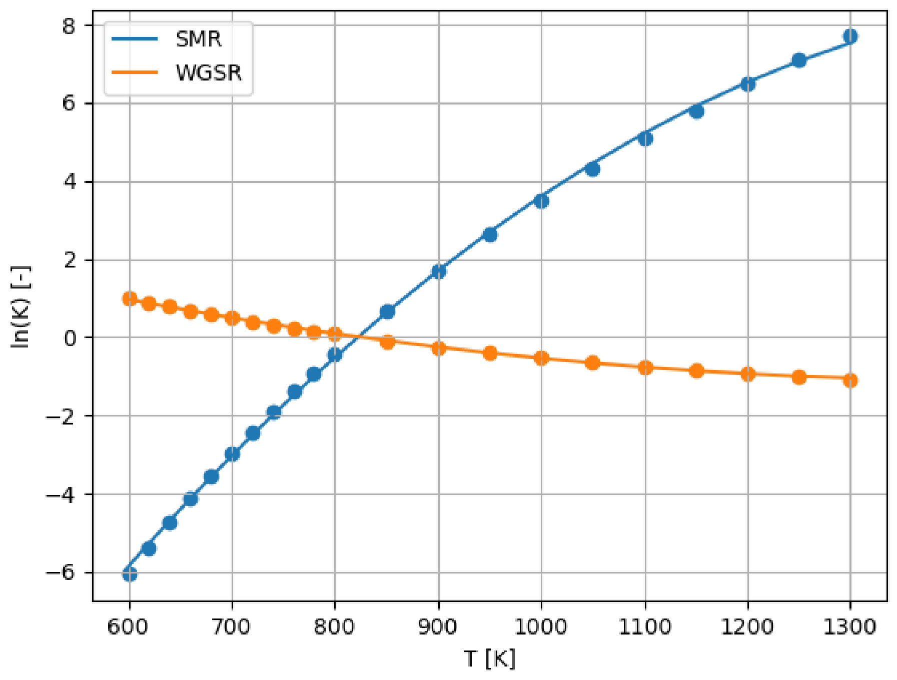

Figure 9: The linear regression for estimating the equations Equation (4.2.27) and Equation (4.2.28) for the equilibrium constants K_{SMR} and K_{WGSR}, respectively[14].

enthalpy of the component for a given temperature. The formulated energy balance is shown in Equation (4.2.29).

$$0 = \sum_i n_{i,in} \cdot h_i(T_{in}) - \sum_i n_{i,out} \cdot h_i(T_{out}) + Q_{PR}, \tag{4.2.29}$$

where Q_{PR} is the heat gained or required in the pre-reformer unit. In summary, with the component balances, energy balance, and the equilibrium constants, the equations of the pre-reformer models are:

$$0 = n_{0,\mathrm{H_2O}} - n_{\mathrm{H_2O}} - 2\xi_{1,pr} - 3\xi_{2,pr} - 4\xi_{3,pr} - 4\xi_{4,pr} - 5\xi_{5,pr} - \xi_{6,pr} - \xi_{7,pr} \tag{4.2.30}$$

$$0 = n_{0,\mathrm{H_2}} - n_{\mathrm{H_2}} + 5\xi_{1,pr} + 7\xi_{2,pr} + 9\xi_{3,pr} + 9\xi_{4,pr} + 11\xi_{5,pr} + 3\xi_{6,pr} + \xi_{7,pr} \tag{4.2.31}$$

$$0 = n_{0,\mathrm{CO}} - n_{\mathrm{CO}} + 2\xi_{1,pr} + 3\xi_{2,pr} + 4\xi_{3,pr} + 4\xi_{4,pr} + 5\xi_{5,pr} + \xi_{6,pr} - \xi_{7,pr} \tag{4.2.32}$$

$$0 = K_{SMR}(T_{PR})(x_{\mathrm{CH_4}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO}} \cdot x_{\mathrm{H_2}}^3) \tag{4.2.33}$$

$$0 = K_{WGSR}(T_{PR})(x_{\mathrm{CO}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO_2}} \cdot x_{\mathrm{H_2}}) \tag{4.2.34}$$

$$0 = n_{0,\mathrm{C_2H_6}} - n_{\mathrm{C_2H_6}} - \xi_{1,pr} \tag{4.2.35}$$

$$0 = n_{0,\mathrm{C_3H_8}} - n_{\mathrm{C_3H_8}} - \xi_{2,pr} \tag{4.2.36}$$

$$0 = n_{0,\mathrm{i\text{-}C_4H_{10}}} - n_{\mathrm{i\text{-}C_4H_{10}}} - \xi_{3,pr} \tag{4.2.37}$$

$$0 = n_{0,\text{n-C}_4\text{H}_{10}} - n_{\text{n-C}_4\text{H}_{10}} - \xi_{4,pr} \tag{4.2.38}$$
$$0 = n_{0,\text{C}_5\text{H}_{12}} - n_{\text{C}_5\text{H}_{12}} - \xi_{5,pr} \tag{4.2.39}$$
$$0 = \sum_i n_{i,in} \cdot h_i(T_{in}) - \sum_i n_{i,out} \cdot h_i(T_{out}) + Q_{PR}. \tag{4.2.40}$$

4.2.2 GHR and ATR model

As mentioned in section 4, to model the shell-and-tube-like connectivity between the GHR and ATR, a new slack variable, Q_{GHR}, has been introduced. This is the required amount of heat that the catalyst-filled pipelines in the GHR unit receive from the high-heated outlet stream from the ATR unit. Instead of the original cross-connection between the two reactors, a cooler has been introduced, and the duty of this cooler is given as $Q_{postATR}$. $Q_{postATR}$ should be same as Q_{GHR}, only with opposite signs. The new constraint in the model that ensures this connectivity is formulated as:

$$Q_{GHR} + Q_{postATR} = 0. \tag{4.2.41}$$

To ensure that the exiting hot stream has a higher temperature than the entering cold stream in the heat exchanging process, 2 additional inequality constraints have been implemented in the model, with an approach temperature of 25 °C. The constraints are shown in Equation (4.2.42) and Equation (4.2.43), and are the only inequality constraints in the model.

$$T_{ATR,out} - T_{GHR,out} \geq 25 \tag{4.2.42}$$
$$T_{postATR,out} - T_{GHR,in} \geq 25. \tag{4.2.43}$$

The constraints can be explained by first introducing the original heat transfer heat connectivity between the GHR and the ATR unit. A simplified figure of the original connectivity is shown in Figure 10, and the modified version where an additional cooler is added is shown in Figure 11.

From Figure 10, it can be seen that the inlet cold stream is GHR_in, the outlet cold stream is GHR_out, inlet hot stream is GHR_in2 and the outlet hot stream is GHR_out2. For the model with the new connectivity, it can be seen from Figure 11, that the inlet and outlet cold stream remains the same, but the inlet hot stream is now ATR_out and the outlet hot stream is postATR_out, which is the stream after the cooler after ATR gave heat to the GHR unit. To visualize how the cold and hot streams change in temperature over a length x, a figure has been made with the corresponding variable names from Figure 11, shown in Figure 12

It can be seen from the Figure 12 that the heat exchange at the outer bounds of x is between ATR_out and GHR_out, and postATR_out and GHR_in. To ensure that ATR_out and postATR_out have always a higher temperature than GHR_out and GHR_in, respectively, an arbitrary value of 25 °C has been set as their difference, shown in Equation (4.2.42) and Equation (4.2.43).

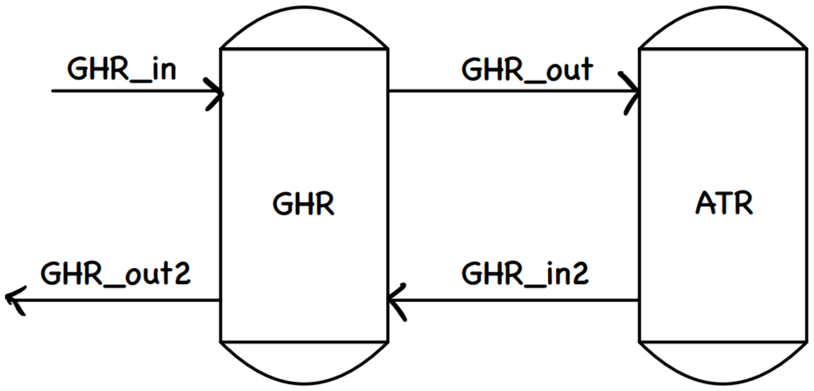

Figure 10: Simplified figure of the connectivity between GHR and ATR.

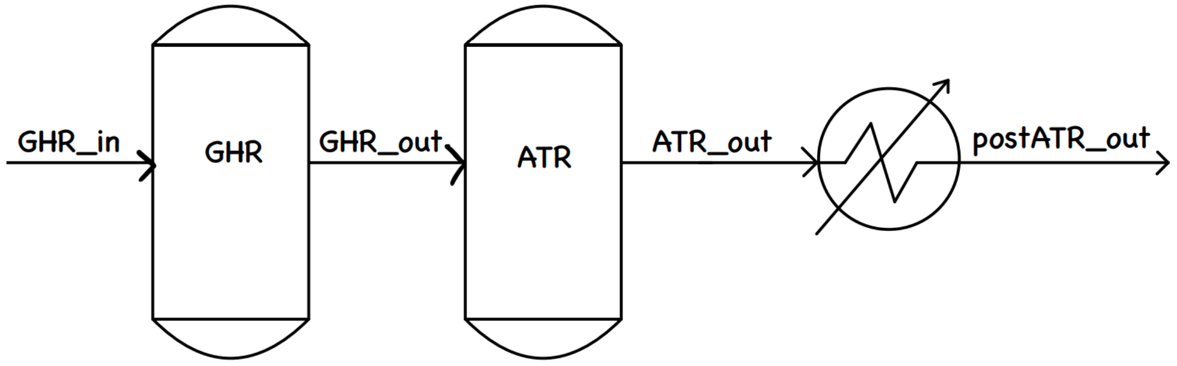

Figure 11: The modified version of GHR and ATR model.

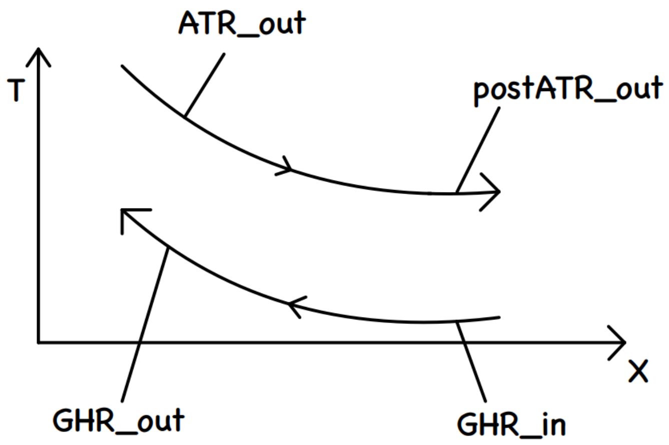

Figure 12: Heat exchanger temperature plot in the GHR.

The component balances and energy balances for the GHR unit are identical to the balances made for the pre-reformer unit, explained in section 4.2.1. But instead of 7 equilibrium reactions, there are only two, the steam-methane reforming and the water-gas shift reactions, shown in Equation (3.1.2) and Equation (3.1.3), respectively. The same method for calculating the extent of reactions for the pre-reformer unit is used for the GHR unit and is formulated as:

$$\xi_{1,ghr} = n_{0,\mathrm{CH_4}} - n_{\mathrm{CH_4}} \tag{4.2.44}$$
$$\xi_{2,ghr} = n_{\mathrm{CO_2}} - n_{0,\mathrm{CO_2}}. \tag{4.2.45}$$

The remaining component balances for H_2O, H_2 and CO is then:

$$n_{\mathrm{H_2O}} = n_{0,\mathrm{H_2O}} - \xi_{1,ghr} - \xi_{2,ghr} \tag{4.2.46}$$
$$n_{\mathrm{H_2}} = n_{0,\mathrm{H_2}} + 3\xi_{1,ghr} + \xi_{2,ghr} \tag{4.2.47}$$
$$n_{\mathrm{CO}} = n_{0,\mathrm{CO}} + \xi_{1,ghr} - \xi_{2,ghr}. \tag{4.2.48}$$

With the same method as the pre-reformer unit for formulating the equilibrium constant and energy balance, the equations for the GHR model can be summarized as:

$$0 = K_{SMR}(T_{GHR})(x_{\mathrm{CH_4}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO}} \cdot x_{\mathrm{H_2}}^3) \tag{4.2.49}$$
$$0 = K_{WGSR}(T_{GHR})(x_{\mathrm{CO}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO_2}} \cdot x_{\mathrm{H_2}}) \tag{4.2.50}$$
$$0 = n_{\mathrm{H_2O}} - n_{0,\mathrm{H_2O}} + \xi_{1,ghr} + \xi_{2,ghr} \tag{4.2.51}$$
$$0 = n_{\mathrm{H_2}} - n_{0,\mathrm{H_2}} - 3\xi_{1,ghr} - \xi_{2,ghr} \tag{4.2.52}$$
$$0 = n_{\mathrm{CO}} - n_{0,\mathrm{CO}} - \xi_{1,ghr} + \xi_{2,ghr} \tag{4.2.53}$$
$$0 = \sum_i n_{i,in} \cdot h_i(T_{in}) - \sum_i n_{i,out} \cdot h_i(T_{out}) + Q_{GHR}. \tag{4.2.54}$$

The component balance and energy balance equations for the ATR unit are identical to the equations, but in addition, there is a combustion reaction of pure O_2 and H_2, shown in Equation (4.2.55):

$$2\,\mathrm{H_2} + \mathrm{O_2} \longrightarrow 2\,\mathrm{H_2O}. \tag{4.2.55}$$

Since all oxygen is being combusted, the extent of reaction for the reaction shown in Equation (4.2.55) can be calculated from the component balance of oxygen:

$$n_{\mathrm{O_2}} = n_{0,\mathrm{O_2}} + \xi_{atr,1} \tag{4.2.56}$$
$$\xi_{atr,1} = n_{\mathrm{O_2}} - n_{0,\mathrm{O_2}}. \tag{4.2.57}$$

The equations for the ATR model can then be summarized as:

$$0 = K_{SMR}(T_{ATR})(x_{\mathrm{CH_4}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO}} \cdot x_{\mathrm{H_2}}^3) \tag{4.2.58}$$
$$0 = K_{WGSR}(T_{ATR})(x_{\mathrm{CO}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO_2}} \cdot x_{\mathrm{H_2}}) \tag{4.2.59}$$
$$0 = n_{\mathrm{H_2O}} - n_{0,\mathrm{H_2O}} - 2\xi_{1,atr} + \xi_{2,atr} + \xi_{3,atr} \tag{4.2.60}$$
$$0 = n_{\mathrm{H_2}} - n_{0,\mathrm{H_2}} + 2\xi_{1,atr} - 3\xi_{2,atr} - \xi_{3,atr} \tag{4.2.61}$$
$$0 = n_{\mathrm{CO}} - n_{0,\mathrm{CO}} - \xi_{2,atr} + \xi_{3,atr} \tag{4.2.62}$$
$$0 = \sum_i n_{i,in} \cdot h_i(T_{in}) - \sum_i n_{i,out} \cdot h_i(T_{out}), \tag{4.2.63}$$

where $\xi_{2,atr}$ and $\xi_{3,atr}$ are the extent of reactions of the steam-methane reforming and water-gas shift reactions, respectively. It is worth noting that the outlet enthalpy summation in the energy balance, shown in Equation (4.2.63), also has a term for enthalpy change given by the oxygen feed.

4.2.3 Isothermal shift reactor model

In the isothermal shift reactor, due to the reactor conditions and the catalysts, there is only 1 equilibrium reaction happening in the reactor, which is the water-gas shift reaction. The reasoning behind this is to remove as much of CO as possible in the process into H_2 and CO_2 by reacting CO with H_2O. The extent of reaction in the reactor can be calculated from the component balance of H_2O_2, and with the same method as the previous reactors, the extent of reaction is formulated as:

$$\xi_{itsr} = n_{0,\mathrm{H_2O}} - n_{\mathrm{H_2O}}. \tag{4.2.64}$$

As the water-gas shift reaction is exothermic, heat is recovered in this reactor. In addition, an internal cooler that cools down the inlet stream to the operating temperature of the reactor is also implemented, such that even more heat is recovered. With the identical component balance and energy balance equations as the previous reactors, the equations for the isothermal shift reactor model can be summarized as:

$$0 = n_{0,\mathrm{CH_4}} - n_{\mathrm{CH_4}} \tag{4.2.65}$$
$$0 = K_{WGSR}(T_{ITSR})(x_{\mathrm{CO}} \cdot x_{\mathrm{H_2O}}) - (x_{\mathrm{CO_2}} \cdot x_{\mathrm{H_2}}) \tag{4.2.66}$$
$$0 = n_{\mathrm{H_2}} - n_{0,\mathrm{H_2}} - \xi_{1,itsr} \tag{4.2.67}$$
$$0 = n_{\mathrm{CO}} - n_{0,\mathrm{CO}} + \xi_{1,itsr} \tag{4.2.68}$$
$$0 = n_{\mathrm{CO_2}} - n_{0,\mathrm{CO_2}} - \xi_{1,itsr} \tag{4.2.69}$$
$$0 = \sum_i n_{i,in} \cdot h_i(T_{in}) - \sum_i n_{i,out} \cdot h_i(T_{out}) + Q_{ITSR}. \tag{4.2.70}$$

4.2.4 Process condensate model

In the process condensate unit, there are no reactions happening. Instead, a flash tank model has been used to calculate the split ratio of the components based on vapor-liquid separation principles. Firstly, it is assumed that the feed stream, F, is separated into a vapor and liquid stream, V and L, respectively, where a phase equilibrium assumption has been made. The component balance for the process condensate model can then be formulated as:

$$Fz_i = Vy_i + Lx_i, \tag{4.2.71}$$

where z_i, y_i and x_i are mole fractions of the respective phases. Furthermore, an assumption that the vapor-liquid equilibrium can be described by a K-value:

$$y_i = K_{VLE,i}x_i, \tag{4.2.72}$$

where $K_{VLE,i}$ can be calculated from Raoult's law. It is therefore assumed that Raoult's law is applicable in the process. Then $K_{VLE,i}$ can be assumed to be dependent on only temperature and total pressure:

$$K_{VLE,i} = p_i^{sat}(T)/p, \tag{4.2.73}$$

where p is the total pressure and p_i^{sat} is the saturated partial pressure of component i. The saturated partial pressure is then assumed by the Antoine equation:

$$\log_{10}(p_i^{sat}) = A_i - \frac{B_i}{T + C_i}, \tag{4.2.74}$$

where A_i, B_i and C_i are Antoine parameters for component i. As there were difficulties in finding a source that contained all of the parameters for the relevant components since most of the components are in the gas phase for the temperature that the process condensate is operating in, the only calculated saturated partial pressure is H_2O. The rest of the components are assumed to have a sufficiently high $K_{VLE,i}$ value (1e6), which assumes that the components are in a gas phase in the operating temperature. The parameters data for H_2O is from Gubkov et al.(1964)[60].

Since the flash tank is isothermic and the temperature and pressure are known, the flash equations can be rearranged into the Rachford-Rice equations by firstly substituting Equation (4.2.72) into Equation (4.2.71), and then rearranging for x_i:

$$x_i = \frac{F z_i}{L + V K_{VLE,i}} = \frac{z_i}{1 + \frac{V}{F}(K_{VLE,i} - 1)}. \tag{4.2.75}$$

Since the sum of all vapor and liquid fractions become 0, $\sum_i y_i - x_i = 0$, this condition can then be used for Equation (4.2.75) for formulating the Rachford-Rice equations:

$$f(\psi) = \sum_i \frac{z_i(K_{VLE,i} - 1)}{1 + \psi(K_{VLE,i} - 1)} = 0, \tag{4.2.76}$$

where $\psi = V/F$. It is worth noting that $f(\psi)$ is monotonic in ψ, where $0 \leq \psi \leq 1$. The equations for the process condensate model can then be summarized as:

$$0 = z_i - x_i(1 + \frac{V}{F}(K_{VLE,i} - 1)) \tag{4.2.77}$$

$$0 = y_i - K_{VLE,i} x_i \tag{4.2.78}$$

$$0 = \sum_i \frac{z_i(K_{VLE,i} - 1)}{1 + \psi(K_{VLE,i} - 1)} \tag{4.2.79}$$

$$0 = F - L - V, \tag{4.2.80}$$

where Equation (4.2.80) is the total mass balance. Equation (4.2.77) and Equation (4.2.78) apply for every component i in the unit, and therefore are 12 equations in total.

4.2.5 Pressure swing adsorption model

To model the pressure swing adsorption model, a split ratio for the different components has been assumed from a study that did case studies for different configurations for PSA[48]. Hydrogen purity for the 2-tank case from the study was used for this process, and it is assumed that other components have been split near ideally. The split ratio, t, used for the PSA unit in this process is:

$$t = [0.001, 0.001, 0.995, 0.001, 0.001], \tag{4.2.81}$$

where the order of the components are CH_4, H_2O, H_2, CO and CO_2. In other words, the product stream recovers 97.0% of the hydrogen and 0.1% of the other components in the inlet stream, which results in a 99.96% hydrogen purity rate. It is assumed that the PSA unit is modeled as a stream splitter, and the equations in the PSA model can be summarized as:

$$0 = t_i n_{0,i} - n_{prod,i} \tag{4.2.82}$$
$$0 = (1 - t_i) n_{0,i} - n_{purge,i}, \tag{4.2.83}$$

where the split ratio is multiplied by its respective component mole stream, $n_{prod,i}$ is the product stream containing majorly hydrogen, and $n_{purge,i}$ is the purge stream containing majorly CO_2. The part of the product stream (stream 17) is then used for combustion for the heater unit with a part of the natural gas feed stream (stream 19), as shown in Figure 8.

4.2.6 Fuel switch heater model

For the heater model or the fuel switch model, a constraint that will account for the required heat in the model was implemented, which will be supplied by the heat generated from combusting part of the hydrogen product stream (stream 17) and part of the natural gas feed (stream 19) and is formulated as:

$$Q_{heat} = F_{\mathrm{H_2},heat} \cdot \mathrm{HHV}_{\mathrm{H_2}} + F_{NG,heat} \cdot \mathrm{HHV}_{NG}, \tag{4.2.84}$$

where Q_{heat} is the required amount of heating required in the process, $F_{\mathrm{H_2},heat}$ is the amount of product hydrogen stream used for combustion, or stream 17, $\mathrm{HHV}_{\mathrm{H_2}}$ is the high heated value of H_2, $F_{NG,heat}$ is the amount of natural gas feed stream used for combustion, or stream 19, and HHV_{NG} is the high heated value of natural gas.

The second constraint will account for the hydrogen stream being sold as a product, being the difference between the PSA outlet product stream with the hydrogen stream that is being combusted for the required heating in the process. This constraint is expressed as

$$F_{\mathrm{H_2},prod} = F_{16} - F_{\mathrm{H_2},heat}, \tag{4.2.85}$$

where $F_{\mathrm{H_2},prod}$ is the amount of hydrogen that is being sold, or stream 18. Stream 18 is also one of the variables in the economic objective function, explained in section 4.2. F_{16} is the outlet product stream from the PSA unit shown in Figure 8 and $F_{\mathrm{H_2},heat}$ is the amount of hydrogen used for combustion and is the same variable shown in Equation (4.2.84).

Furthermore, a similar constraint as the one shown in Equation (4.2.85), a constraint to account for natural gas feed stream used for combustion can be expressed as:

$$F_2 = F_{NG,init} - F_{NG,heat}, \tag{4.2.86}$$

where F_2 is the first flow that enters the process shown in Figure 7, $F_{NG,init}$ is the initial purified natural gas flow that is extracted from the gas well and $F_{NG,heat}$ is the same variable that is shown in Equation (4.2.84).

The last newly added variable, $F_{\mathrm{CO_2},inj}$, is the CO_2 stream in addition to other impurities separated from the product stream in the PSA, which is being injected back into the gas well. This stream is also referred to as stream 21 in the model flowsheet shown in Figure 8. The injection stream, $F_{\mathrm{CO_2},inj}$, is the sum of the purge stream from PSA unit (stream 15) and the amount of CO_2 generated from combustion and N_2 generated from the use of air for combustion (stream 20). The constraint that accounts for this sum is expressed as:

$$F_{\mathrm{CO_2},inj} = F_{15} + F_{20}, \tag{4.2.87}$$

where F_{15} is the purge stream from the PSA unit and F_{20} is the sum of CO_2 generated from the combustion and inert N_2 that comes with the air used for combustion. The constraint explaining F_{20} is shown as:

$$F_{20} = F_{NG,heat} + \frac{2 \cdot F_{NG,heat}}{0.79}, \tag{4.2.88}$$

where $F_{NG,heat}$ is the amount of CO_2 generated from combusting natural gas due to the combustion of hydrocarbons of n-carbons has a 1:1 molar ratio. This ratio is also shown in Equation (3.1.4). Lastly, the last term comes from the average molar composition of standard air used for combustion is approximately 21% O_2 and 79% N_2 and the molar ratio of hydrocarbons of n-carbons and O_2, which is shown in Equation (3.1.5).

4.2.7 Compression work model

The compression work model calculates the amount of work that it requires to compress stream 21, which is the injection stream, from 1 bar to 100 bar. To model this, a two-stage compressor is used, with a cooler in between. The compressors have the same compression factors, in other words, the first compressor performs an isentropic (adiabatic and reversible) compression from 1 bar to 10 bar. Due to the adiabatic conditions and constant volume, the temperature will increase. To deal with this, a cooler is implemented to cool down the compressed stream to the initial temperature before the first compression. Then the last compressor compresses the gas from 10 bar to 100 bar. The equation for calculating the reversible work for an isentropic compression is given as[61]:

$$W_s^{rev} = n \cdot c_p \cdot T_1 \left(\left(\frac{P_2}{P_1} \right)^{\frac{\gamma-1}{\gamma}} - 1 \right), \tag{4.2.89}$$

where W_s^{rev} is the reversible work, n is the total mole stream, T_1 is the temperature of the inlet stream, P_1 and P_2 is the pressure inlet and post-compression pressure of the stream, respectively, c_p is the heat capacity of the stream. However, since the flow from the heater is considered as a total mole stream, and with the ideal gas assumption, the heat capacity is estimated to be $5R/2$ and not calculated as a mean heat capacity, which is the sum of the heat capacity of each component multiplied by its respective molar fraction. γ is the specific heat ratio and is defined as the ratio between c_p/c_v. The ideal gas law is assumed in this process, and thus $c_v = c_p + R$. The mathematical expression for the specific heat ratio is shown in Equation (4.2.90).

$$\begin{aligned}\gamma &= \frac{c_p}{c_p + R} \\ \gamma &= \frac{5R/2}{5R/2 - R} \\ \gamma &= \frac{5}{3}\end{aligned} \tag{4.2.90}$$

For calculating the temperature after compression, the following expression is given[61]:

$$T_2 = T_1 \left(\frac{P_1}{P_2}\right)^{\frac{\gamma-1}{\gamma}}. \tag{4.2.91}$$

The heat recovered in the cooler part can be used in a heat exchanger network to minimize the required amount of heat, but this implementation will not be considered in this project and can be a potential future improvement for the process to reduce the costs of operating the plant. Another cooler after the last compression will not be implemented as it will not be significant for the objective and optimization problem. The total compression work is assumed to be the major cost of injecting the CO_2 gas into the oil reservoir. In reality, the compressor needs higher work to actually perform the compression than the reversible compression work due to the effectiveness of the compressor and how well it can convert electrical work into compression work. In this project, it is assumed that an ideal compressor converts all electrical work into compression work, where the work required in the process is assumed to be reversible.

4.3 Selection of variables

4.3.1 Controlled variables

Generally, when selecting control variables for self-optimizing control, local or global methods can be used, where both approaches approximate the nonlinear loss function quadratically around the nominal operating point. In this project, good engineering decisions will be used to decide the good variables in the system that can be considered as input variables. These variables will then be given their respective lower and upper bounds instead of an equality constraint to free up the variable, and then studying the optimization results to see if there are any control variables that are active. An active variable needs to be controlled, and a degree of freedom is lost since the control variable can not be used for control purposes. In addition, it is assumed that the active set does not change from disturbance changes.

Control variables should be measurements that are available for control purposes. Most measurements in the system are flow data, such as mole flow, composition, or temperature in the flow. In theory, these measurements, such as mole streams, may be selected for control purposes, but it can lead to bottleneck or snowballing, which causes the process to break or lead to economic losses. In addition, these streams are not easily manipulated. Instead, the heating and cooling required in the process, reactor temperatures, steam-to-carbon (S/C) ratio, and oxygen stream will be used as control variables, which amounts to a total of 8 control variables. These measurements are easy to manipulate and measure and have a big impact on economic evaluation.

Table 4: Selected control variables with their respective upper and lower bounds and their nominal optimal value.

Variable	Nominal optimal value	Unit	l_b	u_b
S/C-ratio	5.0	[-]	0.0	5.0
n_{O_2}	79.3	$[\mathrm{kmol\,h^{-1}}]$	0.0	100.0
T_{prePR}	644.6	[K]	643.0	743.0
T_{PR}	609.2	[K]	609.2	709.2
T_{ATR}	1291.8	[K]	1273.0	1373.0
$T_{postATR}$	634.2	[K]	550.0	650.0
T_{ITSR}	473.0	[K]	473.0	573.0
T_{Cond}	293.0	[K]	293.0	333.0

The variables to be tested for active constraints are shown in Table 4, where l_b and u_b are their respective variable's upper and lower bounds, respectively. The temperature for the pre-GHR heater unit is dependent on the temperature of the GHR unit, which again is dependent on the post-ATR cooler, and will therefore not be a degree of freedom. The resulting optimization problem shows that S/C-ratio, and temperature for PR, ITSR, and process condensator are at their bounds, and therefore need to use a degree of freedom to control them. In addition, the inequality constraint for heat exchange between the GHR and ATR units, shown in Equation (4.2.43), is active, and therefore, $T_{postATR}$ needs to control as well. In conclusion, the control variables that will be used for self-optimizing control are n_{O_2}, T_{prePR}, and T_{ATR}.

4.3.2 Disturbances

Disturbance variables in the process are the variables that can not be controlled as they are fluctuating unpredictable variables. In reality, there are numerous disturbances that can impact the process and the economy both by a substantial or negligible amount. In this project, only the most important disturbances are chosen with good engineering decisions. The chosen disturbance variables are the natural gas flow amount, price of hydrogen, and electricity, as these disturbances can have a huge impact on the economy of the process. These variables with their respective nominal values and their chosen deviation are shown in Table 5.

Table 5: Disturbances with their respective nominal value and deviations.

Variable	Nominal value	Unit	Deviation
F_{NG}	145.4	[kmol/h]	±5
$P_{\mathrm{H_2}}$	3.347	[$/kmol]	$\pm10\%$
P_{el}	0.14	[$/kWh]	$\pm10\%$

4.3.3 Degrees of freedom analysis

The nominal optimization results showed that there were 192 variables, 184 equality constraints, and 2 inequality constraints. In the case where both inequality constraints are inactive, this gives a total number of degrees of freedom of 8. As presented in section 4.3.1, 1 inequality constraint was active, and 4 variables were at their respective bounds. Therefore, the remaining degrees of freedom remaining for self-optimizing control is 3.

4.3.4 Measurement subset selection

As mentioned in section 2.7.2, the MIQP method will not be used, while the Tailor-made Branch and Bound algorithm was used for selecting the best measurement subset. A Branch and Bound algorithm developed by Yi Cao[37] was used for selecting the best measurement subset, which uses the exact local method to evaluate the performance for different subsets to ultimately choose the subset. The same subset will be used for the nullspace method. In addition to the measurement subset selected by the Branch and Bound algorithm, good engineering decisions will be used to choose a subset for control purposes. The resulting measurement subset selection was shown in section 5.

5 Results

5.1 Nominal case

As the upper or lower bounds for certain controlled variables were active, as shown in section 4.3.1, these variables use one degree of freedom and cannot be used for self-optimizing control. In addition, the heat exchange inequality constraint shown in Equation (4.2.43), also showed to be active, which leads to the variable, $T_{postATR}$, to be used to control the constraint. The remaining variables that were not active (n_{O_2}, T_{prePR} and T_{ATR}), are used for self-optimizing control, with the assumption that the set of active constraints does not change during operation when disturbances change. The resulting optimization results where the inputs and disturbances were not perturbed will be considered as the nominal case, and the resulting output, inputs, and disturbances will be referred to as y_{nom}, u_{nom}, and d_{nom}, respectively. The nominal values will be crucial when calculating various vectors and matrices necessary for self-optimizing control. The output or measurement values are shown in appendix A.1, and the inputs and the disturbances remain the same as shown in section 4.3.1 and section 4.3.2.

5.2 Local method

To perform local methods for self-optimizing control, different gain vectors that represents the sensitivity of the output when input or disturbance variables are pertubated were calculated. In addition, the Hessian matrix of the sensitivity of the output with respect to input change and input and disturbance change is also calculated. The gain matrices used for performing self-optimizing control are G^y and G^y_d, for input and disturbance change, respectively, where the resulting matrices are shown in section 5.2.1. Lastly, the Hessian matrices J_{uu} and J_{ud} denote the Hessian matrix of the output sensitivity with respect to input change and input and disturbance change, respectively, and the resulting Hessian matrices are shown in section 5.2.2. It is important that the assumption of linearity in the model needs to hold for all operations.

To check if the resulting gain and Hessian matrices were calculated correctly with respect to model mismatch and numerical errors, the gain matrices and Hessian matrices were tested by simulating the linearization equations shown in Equation (2.7.13) and Equation (2.7.19), respectively. By changing one variable at a time and keeping the other variables fixed, the left-hand side of both equations could be checked with their respective right-hand side for each variable. When the matrices were checked to be correctly calculated, the equation shown in Equation (2.7.21) for calculating the optimal sensitivity matrix, F, was used. Lastly, the scaled diagonal matrices for measurements and disturbances were calculated, denoted as W_n and W_d, respectively, which contain the magnitude of the perturbation.

The only remaining matrix for local methods, such as the nullspace method or the exact local method, is the optimal combination matrix, H. Note that the values in G^y, G^y_d, J_{uu}, J_{ud}, W_n and W_d will be the same for both local methods, and therefore presented with all values before the results of the local methods are presented. The only thing varying with the method was the optimal combination matrix, H. However, based on the measurement subset selection for the different methods, the optimal combination matrices will differ from each method. In other words, even though all values in the calculated matrices are presented, the measurement subset selection chooses which variable and its value to use for self-optimizing control.

5.2.1 Gain matrices

To calculate the gain matrices, G^y and G^y_d, each element in the matrices were calculated using finite difference, and is shown as:

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h)-f(x)}{h}, \tag{5.2.1}$$

where h is the size of the perturbation and should be as small as possible. In the case of calculating the gain matrices, a value of $h = 1e-5$ was used. A smaller perturbation should, in theory, give more accurate solutions, the perturbation size should however be carefully chosen as it can lead to constraint violations. It is also worth noting that the perturbation size will be relative to its respective nominal value. The dimensions of the gain matrices when all measurements are used will be $n_y \times n_u$, where $n_y = 184$ and $n_u = 3$. The gain matrices are calculated as:

$$G^y = \begin{bmatrix} \frac{\partial y_1}{\partial u_1} & \frac{\partial y_1}{\partial u_2} & \frac{\partial y_1}{\partial u_3} \\ \frac{\partial y_2}{\partial u_1} & \frac{\partial y_2}{\partial u_2} & \frac{\partial y_2}{\partial u_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_{184}}{\partial u_1} & \frac{\partial y_{184}}{\partial u_2} & \frac{\partial y_{184}}{\partial u_3} \end{bmatrix} \tag{5.2.2}$$

$$G^y_d = \begin{bmatrix} \frac{\partial y_1}{\partial d_1} & \frac{\partial y_1}{\partial d_2} & \frac{\partial y_1}{\partial d_3} \\ \frac{\partial y_2}{\partial d_1} & \frac{\partial y_2}{\partial d_2} & \frac{\partial y_2}{\partial d_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_{184}}{\partial d_1} & \frac{\partial y_{184}}{\partial d_2} & \frac{\partial y_{184}}{\partial d_3} \end{bmatrix}. \tag{5.2.3}$$

The resulting matrices for G^y and G^y_d are shown in appendix A.1.1 and appendix A.1.2, respectively. Note that the gain matrices that are shown in appendix A.1.1 and appendix A.1.2, contains all of the measurements, but later when the matrices are used for different algorithms, certain measurements that give the best performance will be chosen and thus will reduce the size of the gain matrices.

5.2.2 Hessian matrices

For calculating the Hessian matrices, J_{uu} and J_{ud}, each element were calculated with multi-variable finite difference methods, shown as:

$$\begin{aligned}\frac{\partial^2 f(x,y)}{\partial x^2} &\approx \frac{f(x+h,y)-2f(x,y)+f(x-h,y)}{h^2}\\ \frac{\partial^2 f(x,y)}{\partial xy} &\approx \left[\frac{\begin{array}{r} f(x+h,y+k)-f(x+h,y)-f(x,y+k)+2f(x,y)\\ -f(x-h,y)-f(x,y-k)+f(x-h,y-k)\end{array}}{2hk}\right].\end{aligned} \tag{5.2.4}$$

where h and k is the relative perturbation size of variable x and y, respectively. The dimensions for J_{uu} is $n_u \times n_u$ and for J_{ud}, the dimensions are $n_u \times n_d$. The Hessian matrices for this case are then defined as:

$$J_{uu} \approx \begin{bmatrix} \frac{\partial^2 J}{\partial u_1^2} & \frac{\partial^2 J}{\partial u_1 \partial u_2} & \frac{\partial^2 J}{\partial u_1 \partial u_3} \\ \frac{\partial^2 J}{\partial u_2 \partial u_1} & \frac{\partial^2 J}{\partial u_2^2} & \frac{\partial^2 J}{\partial u_2 \partial u_3} \\ \frac{\partial^2 J}{\partial u_3 \partial u_1} & \frac{\partial^2 J}{\partial u_3 \partial u_2} & \frac{\partial^2 J}{\partial u_3^2} \end{bmatrix} \tag{5.2.5}$$

$$J_{ud} \approx \begin{bmatrix} \frac{\partial^2 J}{\partial u_1 \partial d_1} & \frac{\partial^2 J}{\partial u_1 \partial d_2} & \frac{\partial^2 J}{\partial u_1 \partial d_3} \\ \frac{\partial^2 J}{\partial u_2 \partial d_1} & \frac{\partial^2 J}{\partial u_2 \partial d_2} & \frac{\partial^2 J}{\partial u_2 \partial d_3} \\ \frac{\partial^2 J}{\partial u_3 \partial d_1} & \frac{\partial^2 J}{\partial u_3 \partial d_2} & \frac{\partial^2 J}{\partial u_3 \partial d_3} \end{bmatrix}. \tag{5.2.6}$$

where each element is the change in the objective value with respect to inputs and disturbances. It is worth noting that when calculating each element, absolute perturbation size was used to perpetuate the variables in finite difference and not relative perturbation size that is being used in calculating gain matrices. In this case, the absolute perturbation size of $h = k = 1 \cdot 10^{-2}$, is relatively high compared to the relative perturbation used in calculating gain matrices, but this size was necessary due to numerical issues. The resulting Hessian matrices are shown as:

$$J_{uu} \approx \begin{bmatrix} -1051.96 & -0.0022 & 10.7973 \\ -0.0022 & -224.72 & 10.7657 \\ 10.7973 & 10.7657 & -10.7609 \end{bmatrix} \tag{5.2.7}$$

$$J_{ud} \approx \begin{bmatrix} -9.0226 \cdot 10^{-5} & -3.0687 & 0.0030 \\ -1.0272 \cdot 10^{-7} & -0.0749 & 2.7102 \cdot 10^{-5} \\ 5.7413 \cdot 10^{-5} & 0.0019 & -3.0804 \cdot 10^{-6} \end{bmatrix}. \tag{5.2.8}$$

5.2.3 Optimal sensitivity matrix

The optimal sensitivity matrix, F, is defined as shown in Equation (2.7.21) or Equation (2.7.22). However, as it was tested that the necessary gain matrices and Hessian matrices were calculated correctly, the equation shown in Equation (2.7.21) was used to calculate the optimal sensitivity matrix:

$$F = G_d^y - G^y J_{uu}^{-1} J_{ud}. \tag{5.2.9}$$

The dimensions of the matrix are $n_y \times n_d$, and the resulting matrix is shown in appendix A.1.3. The optimal sensitivity matrix will be used in the nullspace method, where only certain rows of the matrix will be used, decided by the selected measurement subset.

5.2.4 Diagonal scaling matrices

To calculate the diagonal scaling matrices, different measurement noises were chosen by good engineering judgment, as plant measurements were not available in this project. There were 4 different types of measurements; temperature, flow, component fraction, and heat flow. For temperature, an absolute measurement noise of 1 K was assumed. For flow measurements, a relative measurement noise size of 2% of the nominal value of the variable was assumed. For component fractions, an absolute measurement noise of 0.01 was chosen, and lastly, a relative measurement noise of 5% for heat flow measurements was assumed. After the measurement noises were applied for the appropriate measurements, a diagonal scaling matrix was made from the measurement vector to finally calculate the diagonal scaling matrix for measurement noise, W_n. For calculating the W_d diagonal matrix, which represents the magnitude of the fluctuations of the disturbance variables, a relative disturbance size of 10% of the nominal value of d_2 and d_3 were assumed, which are the electricity price and the hydrogen price, respectively, and an absolute disturbance size of 5 were assumed for d_1, which is the initial inlet natural gas stream. The same disturbance sizes are implemented when perturbing the disturbance variables when studying the loss of optimality when control elements take place in the system.

5.2.5 Exact local method

To select the optimal combination matrix, H, measurement subsets were selected using a Bidirection branch and bound solver developed by Yi Cao[37]. This method performs a custom branch and bound algorithm to evaluate the best measurement subset, evaluated by worst-case loss. This algorithm gave a subset of 3 variables which are chosen as optimal combination matrices. These matrices are then implemented to the optimization problem as:

$$\Delta c = H \Delta y^{opt} = 0, \tag{5.2.10}$$

where Δy^{opt} is the difference between the optimal and nominal output. Δc is set as zero to calculate the worst-case loss and the loss from controlling the measurement variables will be used as a measurement of performance when ranking which measurement variable is best to use to control. The selected measurement variables for the subset were: $n_{6,\mathrm{H_2}}$, $n_{15,\mathrm{CH_4}}$ and T_5, which corresponds to the hydrogen outlet mole stream of the pre-reformer, the methane outlet purge mole stream out of the pressure swing adsorption unit, and the temperature of the heater before the pre-reformer, respectively.

5.2.6 Nullspace method

Measurement subset selection for the nullspace method uses the same measurement subset that was found by the algorithm by Yi Cao[37], which was used for finding the worst-case loss with Exact Local Method. The optimal sensitivity matrix that was calculated earlier then contains only the selected measurements from the measurement subset. The optimal combination matrix, H is then found by calculating the nullspace matrix of the F matrix such that:

$$HF = 0. \tag{5.2.11}$$

Under the assumption that there is no measurement noise, the average loss and worst-case loss from the optimal combination matrix found from the nullspace method is zero. As such, even though the optimal measurement matrix found from the nullspace method minimizes loss in optimality due to disturbances, it does not guarantee optimal performance for all cases. One important note is that the assumption of linearity needs to be held for all disturbance changes such that the linear model holds and the matrices calculated from the linear model are considered to be valid, in addition to the assumption that the set of active constraints is not changed. This is also why other methods are considered, as in nonlinear systems or systems where the measurement noise is high and can significantly impact the system, the nullspace method is not the best method.

5.2.7 Measurement subset from good engineering judgement

In addition to the optimal combination matrix calculated from the measurement subset selection algorithm for the exact local method and the nullspace method, another measurement subset was selected through good engineering judgment to create another optimal combination matrix. When selecting such measurements, good variables for the control variables to control needs to be considered. For the first input variable, $n_{\mathrm{O_2}}$, measuring the $\mathrm{CO_2}$ amount that enters the ITSR unit can be used to control the equilibrium in the ATR such that maximum CO is converted. For the second input, T_6, measuring the $\mathrm{CH_4}$ stream out of the pre-reformer can be used for control, as keeping an optimal temperature will ensure that in addition to all heavier hydrocarbons converted to methane, but also keeping the amount of methane high such that the conversions in the following units, GHR and ATR, can operate optimally. Lastly, T_9, which is the last input variable and represents the outlet temperature of the ATR unit. To control this variable, measuring T_8 or the outlet temperature of GHR or inlet temperature of the ATR can be measured to be controlled. In summary, the measurement subset selected from good engineering decisions is T_8, $n_{10,\mathrm{CO_2}}$, and $n_{6,\mathrm{CH_4}}$.

5.2.8 Loss

This subsection shows the loss for different control structures when different disturbance variables changes. The disturbance variables are the inlet natural gas flow, electricity price, and hydrogen price, denoted as d_1, d_2, and d_3, respectively. The signs in front of the disturbance variables denote if the change is either $+10\%$ of its nominal value or -10% of its nominal value for d_2 and d_3, while the signs denote an absolute value of ±5 for d_1. Furthermore, the optimal combination matrices, H, found from the local methods, exact local method, and nullspace method are denoted as H_{exact}, and H_{null}, respectively. The optimal combination matrix decided from good engineering judgment is denoted as H_{gej}. In addition, the nominal performance where inputs are fixed at their nominal values and disturbances are ignored, is reported to compare the results with the implemented advanced control element to improve the economy of the process. This is denoted by $u = u_{nom}$ in the reported loss in the tables below. Firstly, only one disturbance variable is perturbed to study how each disturbance change impacts the system by itself and the resulting loss is shown in Table 6. Then two disturbance variables are pertubated for all combinations possible, to see which disturbance variable impacts the system more than the other variable, where the results are reported in Table 7. Lastly, combinations of all disturbance variables pertubated are reported in Table 8, to see overall how the economic loss is impacted when all disturbance variables are in effect. A best and worst case of disturbances can also be seen from this result. The units of the resulting loss from different cases with different optimal combination matrices, H, are profit in USD per hour, or $\$/hour$. In the next section, the performance of different control structures will be discussed.

When the optimization problem resulted in constraint infeasibility, all variable bounds were removed including inequality constraints and the self-optimizing constraints can lead to an infeasible problem. To study the system, inequality constraints that could lead to infeasibility were removed. After studying the infeasible cases, all cases were infeasible due to the constraints regarding the GHR and ATR unit being violated. This indicates that when the disturbance is changed, the assumption of the linear model did not hold. In other words, as the model is assumed locally linear around the nominal points, a such change in disturbances drives the system out of its linear assumption, which results in constraint violations. In the cases where the optimal problem resulted in a constraint violation, the resulting loss is denoted by a red color.

Table 6: Reported loss from different control structures, where 1 disturbance variable is changed. The values have $\$/hour$ as their unit, and the numbers in red denote constraint violation.

Case	Perturbation	$u = u_{nom}$	H_{exact}	H_{null}	H_{gej}
1	$+d_1$	-19.9865	-22.0054	-21.5284	-19.9862
2	$+d_2$	-0.0066	-2.0778	577.6286	-0.000
3	$+d_3$	-0.0035	-2.1761	-2.1704	-0.000
4	$-d_1$	-16.4390	-1.1232	-1.1239	22.2995
5	$-d_2$	-0.0010	-1.6012	-1.9688	-0.000
6	$-d_3$	-0.0011	-1.8701	-1.8721	-0.000

Table 7: Reported loss from different control structures, where 2 disturbance variables are changed. The values have $\$/hour$ as their unit, and the numbers in red denote constraint violation.

Case	Perturbation	$u = u_{nom}$	H_{exact}	H_{null}	H_{gej}
7	$+d_1, +d_2$	-22.4891	-24.5610	-24.5316	-22.4889
8	$+d_1, +d_3$	-19.4825	-21.6450	-20.3731	-19.4822
9	$+d_2, +d_3$	-0.0013	-2.2283	-2.2283	-0.000
10	$-d_1, -d_2$	-16.4813	-1.0685	-1.0685	22.5375
11	$-d_1, -d_3$	-14.7528	-1.0664	-1.067	19.8346
12	$-d_2, -d_3$	-0.001	-1.7983	-0.000	-0.000
13	$+d_1, -d_2$	-78.0521	159.483	161.415	161.4149
14	$-d_1, +d_2$	-16.3967	-1.1794	565.6105	22.0594
15	$+d_1, -d_3$	-20.4903	-22.3607	-8.6928	-20.4903
16	$-d_1, +d_3$	-18.1252	-1.1802	-1.1809	24.7687
17	$+d_2, -d_3$	576.4591	-1.9281	577.4468	-0.000
18	$-d_2, +d_3$	598.9526	-1.1802	-1.1809	127.3042

Table 8: Reported loss from different control structures, where 3 disturbance variables are changed. The values have $\$/hour$ as their unit, and the numbers in red denote constraint violation.

Case	Perturbation	$u = u_{nom}$	H_{exact}	H_{null}	H_{gej}
19	$+d_1, +d_2, +d_3$	-21.9849	-24.2071	-24.2134	-21.9849
20	$-d_1, -d_2, -d_3$	-14.7951	-0.8525	-1.0115	-320.7931
21	$+d_1, -d_2, +d_3$	299.7693	-19.0741	-19.0963	-16.9796
22	$+d_1, +d_2, -d_3$	-22.9932	-24.9147	-24.8835	-22.9929
23	$+d_1, -d_2, -d_3$	-18.5576	-19.7988	-19.7578	-17.9876
24	$-d_1, +d_2, +d_3$	-17.9878	-1.2363	-1.2363	24.5294
25	$-d_1, -d_2, +d_3$	-18.1675	-1.1255	-1.1255	25.0076
26	$-d_1, +d_2, -d_3$	-14.7105	-1.1224	-1.1224	19.5906

6 Discussion

Overall, no control structure ensured an optimal performance for all disturbance changes, and all approaches lead to constraint violations. All constraints that were violated were the constraints that involved the GHR and ATR models which is the main reforming part of the process. An improvement to fix the constraint violations regarding the GHR and ATR modeling is to limit the reaction rate in the GHR. Simulations show that in the cases where constraints are violated, CH_4 is converted to a high amount, such that the energy balance constraints are violated. However, some control structures performed better than others and can be discussed by comparing the resulting loss of optimality for different cases. Cases 1 to 6 are the cases where 1 disturbance variable was changed at a time and the rest set to its nominal value and is shown in Table 6. Case 7 to 18 shows the cases where two disturbances were changed at the same time, where the last one is set to its nominal value. The resulting loss is shown in Table 7. Lastly, the cases where all disturbance variables are changed are cases 19 to 26 and are shown in Table 8. The largest loss when inputs are controlled to their nominal setpoint was reported to be -22.9932. It is hard to conclude whether this is an acceptable loss as it is around a 1-2% loss compared to the nominal objective value.

When observing the nominal performance, all cases except case 4, found an optimal solution when 1 disturbance is changed. It can also be observed that for all other cases where d_1 is perturbed with a negative change resulted in infeasibility. Some control structures, however, managed to find an optimal solution in the cases where d_1 is negatively perturbed. In general, all reporting loss should be a negative value as it is comparing the objective value when there are no degrees of freedom as all inputs are controlled with the optimal objective value when the inputs are free and can be adjusted to find the optimal solution when the disturbances are changed and impacts the system. Some cases reported a positive loss, but these cases were concluded as infeasible.

6.1 Exact local method

When implementing the control structure from the optimal combination matrix (H_{exact}) found from the exact local method, the loss in optimality was compared to the nominal case to discuss its performance. In terms of infeasibility, the exact local method proved to be better than only controlling the inputs to their nominal value. For cases 14, 16, 17, 19, and 23, the nominal case showed infeasibility, while the results from the exact local method showed to found an optimal solution. In addition, in all cases where the exact local method showed infeasibility, the nominal case also showed infeasibility. When comparing these two methods in terms of infeasibility, the control structure from the exact local method showed to be better.

However, when comparing results where both the nominal case and the exact local method case found a solution, all cases showed that the exact local method case performed worse in terms of loss. So it indicates that, when only comparing the nominal case and the exact local method case, the nominal case performed better if the solution is found. This may indicate that the assumption of linearity breaks the same range as the nominal case, but as it is performing worse, a bad selection of measurement subset may be the reason. It is worth noting that the measurement subset was chosen by the Bidirection branch and bound algorithm[37] that uses the calculated gain and Hessian matrices of the system, and the approximated linear model. As this is just an approximation of the true nonlinear model, the branch and bound algorithm is not guaranteed to find a reasonable solution. When comparing the exact local method with the other methods, H_{exact} showed a better performance in terms of finding a solution and not resulting in infeasibility. For instance, the exact local method was the only method that found an optimal solution for cases 14 and 16, where other methods could not. However, in the cases where the exact local method and another approach had a feasible operation, the exact local method performed worse in terms of loss in optimality and can indicate a bad selection of measurement subset. In summary, the exact local methods were better than the other methods to find a solution in terms of infeasibility, while in the cases where other methods also found solutions, the exact local methods showed a worse performance.

6.2 Nullspace method

When calculating the optimal combination matrix in the nullspace method, the same measurement subset as the exact local method was used. It is therefore expected that the performance of the nullspace method is similar to the exact local method. In terms of infeasibility, it is hard to conclude whether it performed better than the nominal case or not. In cases 23 and 21, the nominal case showed infeasibility, while the nullspace method found an optimal solution. However, the nullspace method showed infeasibility in a relatively high number of cases and showed that it cannot handle the disturbance change well. In the cases where only 1 disturbance variable is perturbed, the nullspace method performed worse out of all methods. In other cases where both the exact local method and the nullspace method found a solution, the nullspace method performed generally better. Notable cases are 12 and 15, where the nullspace method performed exceedingly well. It indicates that the nullspace method handles negative change in hydrogen price relatively well when it first finds an optimal solution. Apart from cases 12 and 15, it performed worse than the nominal case and may indicate a bad selection of measurement subset. There is no set pattern of the performance, in some cases, it performed excellent and poorly in others, however, it was in general outperformed by H_{gej}. In addition, the nullspace method resulted in the highest number of cases with constraint violations, and due to this, it is concluded that the nullspace method performed worse out of all approaches.

As the optimal combination matrix is found from the optimal sensitivity matrix with the same measurement subset as the exact local method, there are suggestions for improving the method. As the optimal sensitivity matrix is calculated from the gain and Hessian matrices, if the assumption of linearity breaks, the optimal sensitivity matrix is not valid, which can lead to constraint violations. A better measurement subset selection may result in better performance. And in the cases where the nullspace method performed worse, this may indicate that the measurement noise in the system is high and can significantly impact the system, and cannot be assumed to be negligible.

6.3 Good engineering judgment

In general, out of all methods, the optimal combination matrix chosen by good engineering judgment performed the best overall. When first observing the cases where the nominal case found a solution, H_{gej} resulted in a lower loss for all instances. Many cases were also observed to be -0.000, which is good. By implementing control elements into the system, there is no loss in performance when a disturbance is changed and the performance is identical to optimal performance. Compared to the other control structures, H_{gej} gave the best results, except for case 15, where it was discussed that the nullspace method rejected a negative change in hydrogen relatively well.

This result indicated that the measurement subset selection indeed was a problem. However, the selection of measurements in H_{gej} is not guaranteed to be the best measurement subset as it is chosen from good engineering judgment. It is worth noting that selecting the best measurement subset mathematically does not guarantee the best subset either due to assumptions, e.g. the nullspace method is "optimal", but performs worst. These results are not universal, as they are related to how good the linear approximation is. One suggestion to deal with constraint changes is to develop a more flexible control structure with selectors.

6.4 Feasiblity of offshore blue hydrogen

The nominal case resulted in a gross profit of 1268.2 \$/h, which corresponds to approximately 11 million USD per annum, assuming that this platform runs 24 hours a day and 365 days a year. This number is also the resulting profit per hour when the disturbance variables are at their nominal value. Assuming the number of workers needed on a such platform, the profit does not even cover the cost of the salary of the operators and the workers on the platform, and many factors indicate that did project is not feasible. However, improvements in different aspects of this project for a step closer to decarbonization with hydrogen as its key will be discussed.

6.4.1 Reducing costs

For reducing the investment costs of constructing a platform, this offshore hydrogen plant can be integrated from an already existing platform, where the same extraction equipment can be used for the feed to the hydrogen plant. Another suggestion is to implement a heat exchanger network or heat integration to reduce the need for fuel to the required amount of heat, as there are coolers in the process that are not taken advantage of.

6.4.2 Additional costs

In this project, the natural gas sweetening cost is not considered and is an additional important cost that should be considered in the future when calculating the economic feasibility of this project. Calculation of pure oxygen is also not a part of this project's scope and is assumed to be accessible for free. In reality, this is an important cost to measure, as good separation methods are expensive to operate with, whereas lesser separation methods such as air membrane separation lead to nitrogen being introduced to the system. However, such membrane separation may be advantageous as a way to import hydrogen back onshore, which will be discussed.

6.4.3 Hydrogen transportation

As hydrogen becomes a liquid at $-253.0\,^{\circ}\mathrm{C}$, transporting hydrogen by liquifying it can lead to tremendous costs, and be unsafe and unreliable as well. Another method for transporting hydrogen is by first converting it to ammonia, NH_3, which is seen as a hydrogen carrier, by ammonia synthesis where N_2 and H_2 react together, and is why an air membrane with a sufficient separation grade can be used[62].

6.4.4 Methods for producing oxygen

Air separation is also a cheap method of obtaining oxygen with relatively good purity. The last method to be considered as a way to obtain oxygen is by electrolysis, where water is separated to H_2 and O_2 by adding an electric current. This process requires a high amount of electricity, which is not always easily accessible at all times, which will be discussed later. However, if cheap electricity is obtainable in this process, an electrolysis unit for obtaining oxygen is ideal, as the oxygen is guaranteed to be pure, which makes the process predictable in terms of reactions. Water is also easily accessible by boiling seawater. Lastly, hydrogen is a byproduct in an electrolysis process, which can be sold together with the product stream of the process.

6.4.5 Process choice

The process choices in the project were chosen with respect to investment cost, weight, space, and infrastructure. For instance, instead of operating a large SMR furnace, a GHR, and an ATR unit are used, as it takes up less space, give higher conversions, are more energy effective, and lower investment costs. However, the PSA unit which is used for separating CO_2 from the hydrogen in the last stream, is not necessarily the most compact solution for separation, nor the cheapest option with respect to both investment and operating cost. The reasoning behind the choice of the PSA unit as a separate unit instead of more compact solutions like using hydrogen-selective membranes, which are a more energy-efficient and cost-effective alternative, is due to the resulting pressure of the product and purge streams.

PSA results in low pressure CO_2 stream and a high-pressure H_2, while a hydrogen-selective membrane, such as a palladium-alloy membrane, results in a high-pressure CO_2 stream and a low-pressure hydrogen stream.[63] Both gases need to be compressed to a certain level of pressure, where CO_2 needs to be compressed such that it can be transported to the reservoir under the seabed, while hydrogen needs to be compressed for transport, either by liquefying it or reacting to NH_3. It also requires less energy to cool down a gas that has higher pressure, and reacting hydrogen N_2 to NH_3 requires relatively high pressure. Generally, hydrogen has a lower density than CO_2, and thus requires less energy to compress to the same pressure for hydrogen than CO_2, and this is why a PSA unit is preferred in this project over a hydrogen-selective membrane.

The PSA unit is also well studied, while the hydrogen-selective membrane is a relatively new technology that needs to be studied more. The detailed costs behind the two methods are dependable on how much hydrogen and CO_2 there are in the product and purge streams, and if there is a larger amount of CO_2 than hydrogen, a hydrogen-selective membrane may be preferable, and can be calculated by formulating it as an optimization problem where the required compression work is minimized.

6.4.6 Electricity in an offshore facility

Electricity for compression in the objective function formulated in the optimization problem is an important factor in improving the economy of the process. In this project, it is assumed that electricity is transferred through subsea cables from shore, and the price is the same as for all other industries with non-intensive energy requirements. Electricity prices fluctuate and are considered a disturbance variable. To handle this problem, alternatives for generating electricity for the facility are considered. In a decarbonization project, renewable energy needs to be considered, such as wind power, solar panels, or wave power. However, most renewable energy is non-dispatchable, which means that they do not have an on/off switch, are not controllable, and cannot guarantee to match the electricity demand at all times. And this decentralized nature, unreliability, and unsafe reasons are why such renewable energy sources need to be combined with dispatchable energy source technologies or energy storage[62].

For instance, one dispatchable technology is gas turbines, where gas such as steam or CO_2, is heated through fuel. This should be feasible to implement in this process, as there is already a furnace with fuel switching between hydrogen and natural gas implemented. Renewable source like nuclear power is also an option in the future but needs to be studied more as their own project before being implemented into a such facility as this project.

In summary, to supply the process with electricity that is not transferred from onshore, non-dispatchable energy sources such as wind power, solar power, or wave power can be combined with dispatchable energy such as a gas turbine[62]. Note that the CO_2 from burning fuel for operating the gas turbine needs to be captured and stored. However, there are some challenges that arise when these two methods are combined together. The first one is to limit the frequent start and stops of gas turbines, which is decided by the amount of energy that is generated from non-dispatchable sources. This limitation helps the gas turbine's lifetime. When the gas turbine has not operated for a specific time, it requires good control strategies that allow the gas turbine power to quickly operate to meet its demands, due to the delay of heating up the gas to its specific temperature[62].

7 Conclusion

Blue hydrogen production, as described in this project, is economically infeasible. Even with all the assumptions such as neglected natural gas sweetening/purifying costs, oxygen costs, PSA operating costs, and hydrogen transportation costs, the process does not have sufficient revenue at the current hydrogen prices. In addition, the linear model did not hold for all disturbance changes and the assumption of linearization was no longer valid due to constraint violations. All constraint violations were involved with the GHR and the ATR model. This was due to the disturbance changes being too high and the model not being able to reject the disturbances while operating optimally. To improve the model, a reaction limit on the GHR unit can be implemented, as the main reforming part should happen in the ATR unit and too high conversion in the GHR causes the energy balance constraints to be violated. However, in some cases where the linear model did hold, self-optimizing control showed a reduction in loss compared to the nominal case and leads strongly to the conclusion that in an improved model with a better selection of measurements, self-optimizing control is achievable, where control is achieved without the need for re-optimizing with disturbance changes with minimal loss in profits.

The measurement subset that was chosen from good engineering judgment showed the best operation with disturbance change. However, this result is not necessarily the optimal measurement subset. Other measurement subset selection methods that aim to perform better for larger operating windows are the Polynomial zero loss method or Controlled variable adaption[26]. The other approaches, the exact local method, and the nullspace method showed room for improvement, as all methods lead to constraint violations for certain disturbance changes, but in the cases where the method found a solution, it showed a relatively small loss in optimality when disturbances are rejected.

It is assumed that the only generated revenue in the process is the hydrogen that is being sold, which means that the only way to increase profits is by either increasing the hydrogen price or reducing the other costs. When developing this model further, even more costs need to be implemented in the objective function that the optimization problem is trying to minimize. Instances of such costs are mentioned earlier in this section.

There is also a large number of research with a focus on developing technologies that are advantageous for an offshore blue hydrogen facility but faces challenges in terms of weight, HSE risks, or infrastructure requirement. Further work on the offshore sector for blue hydrogen production can lead to more support from policymakers and oil and gas companies investing and developing roadmaps for reaching climate targets[62]. Future studies can involve heat exchanger networks, a combination of dispatchable and non-dispatchable energy sources to supply the process with its own electricity that can be controlled easier and is more predictable. Alternatives for the PSA unit, which is a relatively expensive unit to operate, such as absorption-/adsorption-based separation, which still is under development, and can be used for replacing PSA if the methods are proven to be effective and cheaper.

Methods to produce oxygen to supply the ATR unit is another aspect that needs to be studied, whether separating with an air membrane such that some nitrogen is added to the system through the air, electrolysis, or even a combination of both. If hydrogen is transported by synthesizing it to ammonia, as there is enough nitrogen in the system, electrolysis can be used, but that also increases the need for electricity in the system which may switch the dispatchable energy sources on. In the future, for a such project to be feasible, increased funding and investment from the government and industry are needed.

Due to the high profitability of the oil and gas industry, a such plant is not convincing enough for investments yet. When oil and gas demand is reduced, and other energy sources without emissions are in demand, such hydrogen plants may be in more demand and more industries will invest in them, in addition to other circumstances in the industry such as a carbon tax.

References

[1] International Energy Agency. Net zero by 2050, 2021. URL https://www.iea.org/reports/net-zero-by-2050.

[2] United Nations Climate Change. What is the paris agreement?, 2023. URL https://unfccc.int/process-and-meetings/the-paris-agreement.

[3] Rob West and Bassam Fattouh. The energy transition and oil companies' hard choices, 2019. URL https://www.oxfordenergy.org/wpcms/wp-content/uploads/2019/07/The-Energy-Transition-and-Oil-Companies-Hard-Choices-Energy-Insight-51.pdf.

[4] statista. Greenhouse gas emissions produced by oil and gas facilities in the united states from 2011 to 2020, 2021. URL https://www.statista.com/statistics/615975/ghg-emissions-emitted-by-oil-and-gas-facilities-in-the-us/#:~:text=Oil%20and%.

[5] statista. Carbon dioxide (co2) emissions from crude petroleum and natural gas extraction in the united kingdom from 1990 to 2020, 2022. URL https://www.statista.com/statistics/485941/co2-emissions-of-crude-petroleum-and-natural-gas-uk/.

[6] statista. Annual greenhouse gas emissions in the united states from 1990 to 2020, 2022. URL https://www.statista.com/statistics/517376/us-greenhouse-gas-emissions/.

[7] statista. Carbon dioxide emissions in the united kingdom (uk) from 1990 to 2021, 2023. URL https://www.statista.com/statistics/449503/co2-emissions-united-kingdom-uk/.

[8] Statistics Norway. Emissions from norwegian economic activity, 2022. URL https://www.ssb.no/en/natur-og-miljo/miljoregnskap/statistikk/utslipp-fra-norsk-okonomisk-aktivitet.

[9] Hannah Ritchie, Max Roser, and Pablo Rosado. Energy, 2022. URL https://ourworldindata.org/energy-mix.

[10] Chunshan Song. *Hydrogen and Syngas Production and Purification Technologies*, pages 1 – 13. Wiley, 11 2009. ISBN 9780470561256. doi: 10.1002/9780470561256.ch1.

[11] Plugpower. Fuel cell benefits: 5 facts you should know, 2023. URL https://www.plugpower.com/fuel-cell-power/fuel-cell-benefits/.

[12] U.S. Energy Information Administration. How is hydrogen produced?, 2022. URL https://www.eia.gov/energyexplained/hydrogen/production-of-hydrogen.php.

[13] Robert Howarth and Mark Jacobson. How green is blue hydrogen. *Energy Science and Engineering*, 9, 08 2021. doi: 10.1002/ese3.956.

[14] Yoonsik Oh, Johannes Jäschke, and Evren M. Turan. Tkp4580 - chemical engineering, specialization project: Modeling of offshore blue hydrogen production, 2022. URL `https://folk.ntnu.no/jaschke/Masters/ProjectTheses/2022/YoonsikOh/TKP4580_Specialization_Project.pdf`.

[15] SUBPRO. About subpro, 2022. URL `https://www.ntnu.edu/web/subpro/aboutsubpro`.

[16] Robert Hardy and Neha Vijh. Accelerating the Path to Net Zero with Blue Hydrogen: A Route to Achieving Best-In-Class Environmental Performance and Economics. *Abu Dhabi International Petroleum Exhibition and Conference*, Day 3 Wed, November 02, 2022, 10 2022. doi: 10.2118/210888-MS. URL `https://doi.org/10.2118/210888-MS`.

[17] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006. page 2.

[18] Seongkyu Yoon, Shaun Galbraith, Bumjoon Cha, and Huolong Liu. Chapter 5 - flowsheet modeling of a continuous direct compression process. In Ravendra Singh and Zhihong Yuan, editors, *Process Systems Engineering for Pharmaceutical Manufacturing*, volume 41 of *Computer Aided Chemical Engineering*, pages 121–139. Elsevier, 2018. doi: https://doi.org/10.1016/B978-0-444-63963-9.00005-1. URL `https://www.sciencedirect.com/science/article/pii/B9780444639639000051`.

[19] Sigurd Skogestad. *Prosessteknikk, masse- og energibalanser*. Fagbokforlaget, 3rd edition, 2014. ISBN 9788251924573. pages 72-74.

[20] Sigurd Skogestad. *Prosessteknikk, masse- og energibalanser*. Fagbokforlaget, 3rd edition, 2014. ISBN 9788251924573. pages 88-94.

[21] I. Lefkowitz. Multilevel Approach Applied to Control System Design. *Journal of Basic Engineering*, 88(2):392–398, 06 1966. ISSN 0021-9223. doi: 10.1115/1.3645868. URL `https://doi.org/10.1115/1.3645868`.

[22] Sigurd Skogestad. Plantwide control: the search for the self-optimizing control structure. *Journal of Process Control*, 10(5):487–507, 2000. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(00)00023-8. URL `https://www.sciencedirect.com/science/article/pii/S0959152400000238`.

[23] Sigurd Skogestad. Control structure design for complete chemical plants. *Computers & Chemical Engineering*, 28(1):219–234, 2004. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2003.08.002. URL `https://www.sciencedirect.com/science/article/pii/S0098135403001984`. Escape 12.

[24] Alan S. Foss. Critique of chemical process control theory. *Aiche Journal*, 19:209–214, 1973.

[25] Manfred Morari, Yaman Arkun, and George Stephanopoulos. Studies in the synthesis of control structures for chemical processes: Part i: Formulation of the problem. process decomposition and the classification of the control tasks. analysis of the optimizing control structures. *AIChE Journal*, 26(2):220 – 232, 1980. doi: 10.1002/aic.690260205. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-0018999009&doi=10.1002%2faic.690260205&partnerID=40&md5=669a1eb474a9178bdf52d27b449f85c6`.

[26] Johannes Jäschke, Yi Cao, and Vinay Kariwala. Self-optimizing control – a survey. *Annual Reviews in Control*, 43:199–223, 2017. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2017.03.001. URL `https://www.sciencedirect.com/science/article/pii/S1367578816301055`.

[27] Sigurd Skogestad and Ian Postlethwaite. *Multivariable feedback control: Analysis and Design*. John Wiley, 2005.

[28] Wuhua Hu, Lia Maisarah Umar, Vinay Kariwala, and Gaoxi Xiao. Local self-optimizing control with input and output constraints. *IFAC Proceedings Volumes*, 44(1):9850–9855, 2011. ISSN 1474-6670. doi: https://doi.org/10.3182/20110828-6-IT-1002.01765. URL `https://www.sciencedirect.com/science/article/pii/S1474667016451943`. 18th IFAC World Congress.

[29] Ivar J. Halvorsen, Sigurd Skogestad, John C. Morud, and Vidar Alstad. Optimal selection of controlled variables. *Industrial and Engineering Chemistry Research*, 42(14): 3273 – 3284, 2003. doi: 10.1021/ie020833t. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-0037662130&doi=10.1021%2fie020833t&partnerID=40&md5=17f40794e6b81c9dfa653cc5c17839a7`.

[30] Vinay Kariwala, Yi Cao, and Janardhanan Sivaramakrishnan. Local self-optimizing control with average loss minimization. *Industrial & Engineering Chemistry Research*, 47:1150–1158, 01 2008. doi: 10.1021/ie070897+.

[31] Vidar Alstad, Sigurd Skogestad, and Eduardo S. Hori. Optimal measurement combinations as controlled variables. *Journal of Process Control*, 19(1):138–148, 2009. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2008.01.002. URL `https://www.sciencedirect.com/science/article/pii/S0959152408000073`.

[32] Ramprasad Yelchuru and Sigurd Skogestad. Convex formulations for optimal selection of controlled variables and measurements using mixed integer quadratic programming. *Journal of Process Control*, 22(6):995–1007, 2012. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2012.04.013. URL `https://www.sciencedirect.com/science/article/pii/S0959152412000984`.

[33] Vidar Alstad and Sigurd Skogestad. Null space method for selecting optimal measurement combinations as controlled variables. *Industrial & engineering chemistry research*, 46(3):846–853, 2007. ISSN 0888-5885.

[34] Vinay Kariwala and Yi Cao. Bidirectional branch and bound for controlled variable selection. part ii: Exact local method for self-optimizing control. *Computers & Chemical Engineering*, 33(8):1402–1412, 2009. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2009.01.014. URL https://www.sciencedirect.com/science/article/pii/S0098135409000350.

[35] Adriaen Verheyleweghen and Johannes Jäschke. Self-optimizing control of an lng liquefaction plant. *Journal of Process Control*, 74:63–75, 2019. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2018.01.007. URL https://www.sciencedirect.com/science/article/pii/S0959152418300192. Efficient energy management.

[36] Vinay Kariwala and Yi Cao. Bidirectional branch and bound for controlled variable selection part iii: Local average loss minimization. *IEEE Transactions on Industrial Informatics*, 6(1):54 – 61, 2010. doi: 10.1109/TII.2009.2037494. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-76849088939&doi=10.1109%2fTII.2009.2037494&partnerID=40&md5=20f90821b132581fd171378a72abfd0b.

[37] Yi Cao. Bidirectional branch and bound solvers for worst case loss minimization, 2009. URL https://se.mathworks.com/matlabcentral/fileexchange/22632-bidirectional-branch-and-bound-solvers-for-worst-case-loss-minimization.

[38] Dinesh Krishnamoorthy and Sigurd Skogestad. Online process optimization with active constraint set changes using simple control structures. *Industrial & Engineering Chemistry Research*, 58:13555–13567, 05 2019. doi: 10.1021/acs.iecr.9b00308.

[39] Henrik Manum and Sigurd Skogestad. Self-optimizing control with active set changes. *Journal of Process Control*, 22:873–883, 06 2012. doi: 10.1016/j.jprocont.2012.02.015.

[40] Wuhua Hu, Lia Maisarah Umar, Gaoxi Xiao, and Vinay Kariwala. Local self-optimizing control of constrained processes. *Journal of Process Control*, 22(2):488–493, 2012. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2011.11.003. URL https://www.sciencedirect.com/science/article/pii/S0959152411002228.

[41] Yi Cao. Constrained self-optimizing control via differentiation1. *IFAC Proceedings Volumes*, 37(1):63–70, 2004. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)38710-4. URL https://www.sciencedirect.com/science/article/pii/S1474667017387104. 7th International Symposium on Advanced Control of Chemical Processes (ADCHEM 2003), Hong-Kong, 11-14 January 2004.

[42] Tuong-Van Nguyen, Brian Elmegaard, Leonardo Pierobon, Fredrik Haglind, and Peter Breuhaus. Modelling and analysis of offshore energy systems on north sea oil and gas platforms. In *53rd International Conference of Scandinavian Simulation Society*, 10 2012.

[43] Y. Shirasaki and I. Yasuda. 12 - membrane reactor for hydrogen production from natural gas at the tokyo gas company: a case study. *Woodhead Publishing Series in Energy*, 2: 487–507, 2013. doi: https://doi.org/10.1533/9780857097347.2.487. URL https://www.sciencedirect.com/science/article/pii/B9780857094155500123.

[44] Thomas S. Christensen. Adiabatic prereforming of hydrocarbons — an important step in syngas production. *Applied Catalysis A: General*, 138(2):285–309, 1996. ISSN 0926-860X. doi: https://doi.org/10.1016/0926-860X(95)00302-9. URL https://www.sciencedirect.com/science/article/pii/0926860X95003029.

[45] Rupam Mukherjee and Shilpa Singh. Evaluating hydrogen rich fuel gas firing, 2021. URL digitalrefining.com/article/1002591/evaluating-hydrogen-rich-fuel-gas-firing#.Y40CjXbMJPY.

[46] Dr. Max Appl. *Process Steps of Ammonia Production*, chapter 4, pages 65–176. John Wiley & Sons, Ltd, 1999. ISBN 9783527613885. doi: https://doi.org/10.1002/9783527613885.ch04.

[47] Laura Pellegrini, Giorgia Guido, and Stefania Moioli. Design of the co2 removal section for psa tail gas treatment in a hydrogen production plant. *Frontiers in Energy Research*, 8, 05 2020. doi: 10.3389/fenrg.2020.00077.

[48] Rafael M. Siqueira, Geovane R. Freitas, Hugo R. Peixoto, Jailton F. do Nascimento, Ana Paula S. Musse, Antonio E.B. Torres, Diana C.S. Azevedo, and Moises Bastos-Neto. Carbon dioxide capture by pressure swing adsorption. *Energy Procedia*, 114: 2182–2192, 2017. ISSN 1876-6102. doi: https://doi.org/10.1016/j.egypro.2017.03.1355. URL https://www.sciencedirect.com/science/article/pii/S1876610217315382. 13th International Conference on Greenhouse Gas Control Technologies, GHGT-13, 14-18 November 2016, Lausanne, Switzerland.

[49] Johnson Matthey. Lch process for the production of blue hydrogen, 2022. URL https://matthey.com/documents/161599/474986/26367+JM+LCH+Process+-+Production+of+Low+Carbon+Hydrogen+TP+%28screen%29+13.pdf/71bb11a0-47ce-e609-412b-0678b1b4e5da?t=1654695668629.

[50] Eva K.Hallanda, Van Phama, Fridtjof Riisa, and Ann-Helen Hansena. Co2 for eor combined with storage in the norwegian north sea, 2019. 14th International Conference on Greenhouse Gas Control Technologies, GHGT-14.

[51] Erik Lindeberg. Modelling pressure and temperature profile in a co2 injection well. *Energy Procedia*, 4:3935–3941, 2011. ISSN 1876-6102. doi: https://doi.org/10.1016/j.egypro.2011.02.332. URL https://www.sciencedirect.com/science/article/pii/S1876610211006114. 10th International Conference on Greenhouse Gas Control Technologies.

[52] Kai Zhang, Hon Chung Lau, and Zhangxin Chen. Extension of co2 storage life in the sleipner ccs project by reservoir pressure management. *Journal of Natural Gas Science and Engineering*, 108:104814, 2022. ISSN 1875-5100. doi: https://doi.org/10.1016/j.jn

gse.2022.104814. URL `https://www.sciencedirect.com/science/article/pii/S1875510022004000`.

[53] Norsk Olje Museum. Oil and gas fields in norway - industrial heritage plan, 2016. URL `https://www.norskolje.museum.no/wp-content/uploads/2016/02/3467_321daaaf2b0644d897762f3bb73224cc.pdf`.

[54] Norsk Petroleum. Troll, 2021. URL `https://www.norskpetroleum.no/fakta/felt/troll/`.

[55] JuMP. Nonlinear modeling, 2022. URL `https://jump.dev/JuMP.jl/stable/manual/nlp/`.

[56] COIN-OR. Ipopt: Documentation, 2022. URL `https://coin-or.github.io/Ipopt/`.

[57] J. M. Smith and H. C. Van Ness. *Introduction to chemical engineering thermodynamics*. McGraw-Hill Education,, New York, NY :, eighth edition. edition, 2018.

[58] A.O. Oni, K. Anaya, T. Giwa, G. Di Lullo, and A. Kumar. Comparative assessment of blue hydrogen from steam methane reforming, autothermal reforming, and natural gas decomposition technologies for natural gas-producing regions. *Energy Conversion and Management*, 254:115245, 2022. ISSN 0196-8904. doi: https://doi.org/10.1016/j.enconman.2022.115245. URL `https://www.sciencedirect.com/science/article/pii/S0196890422000413`.

[59] Statistics Norway. Electricity prices, 2023. URL `https://www.ssb.no/en/energi-og-industri/energi/statistikk/elektrisitetspriser`.

[60] A.N. Gubkov, N.A. Fermor, and N.I. Smirnov. *Vapor Pressure of Mono-Poly Systems*. Zh. Prikl. Khim., 1964. URL `https://webbook.nist.gov/cgi/cbook.cgi?ID=C7732185&Mask=4&Type=ANTOINE&Plot=on`.

[61] Sigurd Skogestad. *Prosessteknikk, masse- og energibalanser - Chapter 7.5.2*. Fagbokforlaget, 3rd edition, 2014. ISBN 9788251924573. pages 135-137.

[62] Mari Voldsund, Adriana Reyes-Lúa, Chao Fu, Mario Ditaranto, Petter Nekså, Marit J. Mazzetti, Olaf Brekke, Arne Ulrik Bindingsbø, David Grainger, and Jostein Pettersen. Low carbon power generation for offshore oil and gas production. *Energy Conversion and Management: X*, 17:100347, 2023. ISSN 2590-1745. doi: https://doi.org/10.1016/j.ecmx.2023.100347. URL `https://www.sciencedirect.com/science/article/pii/S259017452300003X`.

[63] Julian Straus, Vidar Torarin Skjervold, Rahul Anantharaman, and David Berstad. Novel approach for low co 2 intensity hydrogen production from natural gas. *Sustainable Energy & Fuels*, 6(21):4948–4961, 2022.

[64] Yoonsik Oh. Github repository for the code used in the master's thesis, 2023. URL `https://github.com/yoonsiko/MastersThesis`.

A Results

A.1 Nominal output

Table 9: Nominal case outputs, where * denoted the actual name of the variable

Variable	Variable*	Value	Unit
mix_in_mol[1]	n_{2,CH_4}	113.761	$kmol\,h^{-1}$
mix_in_mol[2]	n_{2,H_2O}	0.000	$kmol\,h^{-1}$
mix_in_mol[3]	n_{2,H_2}	0.000	$kmol\,h^{-1}$
mix_in_mol[4]	$n_{2,CO}$	0.000	$kmol\,h^{-1}$
mix_in_mol[5]	n_{2,CO_2}	1.948	$kmol\,h^{-1}$
mix_in_mol[6]	n_{2,C_2H_6}	8.869	$kmol\,h^{-1}$
mix_in_mol[7]	n_{2,C_3H_8}	9.742	$kmol\,h^{-1}$
mix_in_mol[8]	$n_{2,i\text{-}C_4H_{10}}$	4.606	$kmol\,h^{-1}$
mix_in_mol[9]	$n_{2,n\text{-}C_4H_{10}}$	2.050	$kmol\,h^{-1}$
mix_in_mol[10]	$n_{2,C_5H_{12}}$	5.336	$kmol\,h^{-1}$
H2Ostream	n_3	716.822	$kmol\,h^{-1}$
mix_out_mol[1]	n_{4,CH_4}	113.761	$kmol\,h^{-1}$
mix_out_mol[2]	n_{4,H_2O}	716.822	$kmol\,h^{-1}$
mix_out_mol[3]	n_{4,H_2}	0.000	$kmol\,h^{-1}$
mix_out_mol[4]	$n_{4,CO}$	0.000	$kmol\,h^{-1}$
mix_out_mol[5]	n_{4,CO_2}	1.948	$kmol\,h^{-1}$
mix_out_mol[6]	n_{4,C_2H_6}	8.869	$kmol\,h^{-1}$
mix_out_mol[7]	n_{4,C_3H_8}	9.742	$kmol\,h^{-1}$
mix_out_mol[8]	$n_{4,i\text{-}C_4H_{10}}$	3.606	$kmol\,h^{-1}$
mix_out_mol[9]	$n_{4,n\text{-}C_4H_{10}}$	2.050	$kmol\,h^{-1}$
mix_out_mol[10]	$n_{4,C_5H_{12}}$	5.336	$kmol\,h^{-1}$
prePR_in_mol[1]	n_{4,CH_4}	113.761	$kmol\,h^{-1}$
prePR_in_mol[2]	n_{4,H_2O}	716.822	$kmol\,h^{-1}$
prePR_in_mol[3]	n_{4,H_2}	0.000	$kmol\,h^{-1}$
prePR_in_mol[4]	$n_{4,CO}$	0.000	$kmol\,h^{-1}$
prePR_in_mol[5]	n_{4,CO_2}	1.948	$kmol\,h^{-1}$
prePR_in_mol[6]	n_{4,C_2H_6}	8.869	$kmol\,h^{-1}$
prePR_in_mol[7]	n_{4,C_3H_8}	9.742	$kmol\,h^{-1}$
prePR_in_mol[8]	$n_{4,i\text{-}C_4H_{10}}$	3.606	$kmol\,h^{-1}$
prePR_in_mol[9]	$n_{4,n\text{-}C_4H_{10}}$	2.050	$kmol\,h^{-1}$
prePR_in_mol[10]	$n_{4,C_5H_{12}}$	5.336	$kmol\,h^{-1}$
prePR_out_mol[1]	n_{5,CH_4}	113.761	$kmol\,h^{-1}$
prePR_out_mol[2]	n_{5,H_2O}	716.822	$kmol\,h^{-1}$
prePR_out_mol[3]	n_{5,H_2}	0.000	$kmol\,h^{-1}$
prePR_out_mol[4]	$n_{5,CO}$	0.000	$kmol\,h^{-1}$

Continued on Next Page

Variable	Variable*	Value	Unit
prePR_out_mol[5]	$n_{5,\mathrm{CO_2}}$	1.948	$\mathrm{kmol\,h^{-1}}$
prePR_out_mol[6]	$n_{5,\mathrm{C_2H_6}}$	8.869	$\mathrm{kmol\,h^{-1}}$
prePR_out_mol[7]	$n_{5,\mathrm{C_3H_8}}$	9.742	$\mathrm{kmol\,h^{-1}}$
prePR_out_mol[8]	$n_{5,\mathrm{i\text{-}C_4H_{10}}}$	3.606	$\mathrm{kmol\,h^{-1}}$
prePR_out_mol[9]	$n_{5,\mathrm{n\text{-}C_4H_{10}}}$	2.050	$\mathrm{kmol\,h^{-1}}$
prePR_out_mol[10]	$n_{5,\mathrm{C_5H_{12}}}$	5.336	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[1]	$n_{5,\mathrm{CH_4}}$	113.761	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[2]	$n_{5,\mathrm{H_2O}}$	716.822	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[3]	$n_{5,\mathrm{H_2}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[4]	$n_{5,\mathrm{CO}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[5]	$n_{5,\mathrm{CO_2}}$	1.948	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[6]	$n_{5,\mathrm{C_2H_6}}$	8.869	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[7]	$n_{5,\mathrm{C_3H_8}}$	9.742	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[8]	$n_{5,\mathrm{i\text{-}C_4H_{10}}}$	3.606	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[9]	$n_{5,\mathrm{n\text{-}C_4H_{10}}}$	2.050	$\mathrm{kmol\,h^{-1}}$
pr_in_mol[10]	$n_{5,\mathrm{C_5H_{12}}}$	5.336	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[1]	$n_{6,\mathrm{CH_4}}$	179.522	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[2]	$n_{6,\mathrm{H_2O}}$	656.033	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[3]	$n_{6,\mathrm{H_2}}$	0.55.141	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[4]	$n_{6,\mathrm{CO}}$	0.228	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[5]	$n_{6,\mathrm{CO_2}}$	32.229	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[6]	$n_{6,\mathrm{C_2H_6}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[7]	$n_{6,\mathrm{C_3H_8}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[8]	$n_{6,\mathrm{i\text{-}C_4H_{10}}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[9]	$n_{6,\mathrm{n\text{-}C_4H_{10}}}$	0.000	$\mathrm{kmol\,h^{-1}}$
pr_out_mol[10]	$n_{6,\mathrm{C_5H_{12}}}$	0.000	$\mathrm{kmol\,h^{-1}}$
preGHR_in_mol[1]	$n_{6,\mathrm{CH_4}}$	179.522	$\mathrm{kmol\,h^{-1}}$
preGHR_in_mol[2]	$n_{6,\mathrm{H_2O}}$	656.033	$\mathrm{kmol\,h^{-1}}$
preGHR_in_mol[3]	$n_{6,\mathrm{H_2}}$	55.141	$\mathrm{kmol\,h^{-1}}$
preGHR_in_mol[4]	$n_{6,\mathrm{CO}}$	0.228	$\mathrm{kmol\,h^{-1}}$
preGHR_in_mol[5]	$n_{6,\mathrm{CO_2}}$	32.229	$\mathrm{kmol\,h^{-1}}$
preGHR_out_mol[1]	$n_{7,\mathrm{CH_4}}$	179.522	$\mathrm{kmol\,h^{-1}}$
preGHR_out_mol[2]	$n_{7,\mathrm{H_2O}}$	656.033	$\mathrm{kmol\,h^{-1}}$
preGHR_out_mol[3]	$n_{7,\mathrm{H_2}}$	55.141	$\mathrm{kmol\,h^{-1}}$
preGHR_out_mol[4]	$n_{7,\mathrm{CO}}$	0.228	$\mathrm{kmol\,h^{-1}}$
preGHR_out_mol[5]	$n_{7,\mathrm{CO_2}}$	32.229	$\mathrm{kmol\,h^{-1}}$
ghr_in_mol[1]	$n_{7,\mathrm{CH_4}}$	179.522	$\mathrm{kmol\,h^{-1}}$
ghr_in_mol[2]	$n_{7,\mathrm{H_2O}}$	656.033	$\mathrm{kmol\,h^{-1}}$
ghr_in_mol[3]	$n_{7,\mathrm{H_2}}$	55.141	$\mathrm{kmol\,h^{-1}}$
ghr_in_mol[4]	$n_{7,\mathrm{CO}}$	0.228	$\mathrm{kmol\,h^{-1}}$
ghr_in_mol[5]	$n_{7,\mathrm{CO_2}}$	32.229	$\mathrm{kmol\,h^{-1}}$
ghr_out_mol[1]	$n_{8,\mathrm{CH_4}}$	94.626	$\mathrm{kmol\,h^{-1}}$

Continued on Next Page

Variable	Variable*	Value	Unit
ghr_out_mol[2]	$n_{8,\mathrm{H_2O}}$	518.618	$\mathrm{kmol\,h^{-1}}$
ghr_out_mol[3]	$n_{8,\mathrm{H_2}}$	362.347	$\mathrm{kmol\,h^{-1}}$
ghr_out_mol[4]	$n_{8,\mathrm{CO}}$	32.605	$\mathrm{kmol\,h^{-1}}$
ghr_out_mol[5]	$n_{8,\mathrm{CO_2}}$	84.6258	$\mathrm{kmol\,h^{-1}}$
atr_in_mol[1]	$n_{8,\mathrm{CH_4}}$	94.626	$\mathrm{kmol\,h^{-1}}$
atr_in_mol[2]	$n_{8,\mathrm{H_2O}}$	518.618	$\mathrm{kmol\,h^{-1}}$
atr_in_mol[3]	$n_{8,\mathrm{H_2}}$	362.347	$\mathrm{kmol\,h^{-1}}$
atr_in_mol[4]	$n_{8,\mathrm{CO}}$	32.605	$\mathrm{kmol\,h^{-1}}$
atr_in_mol[5]	$n_{8,\mathrm{CO_2}}$	84.6258	$\mathrm{kmol\,h^{-1}}$
atr_out_mol[1]	$n_{9,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
atr_out_mol[2]	$n_{9,\mathrm{H_2O}}$	582.934	$\mathrm{kmol\,h^{-1}}$
atr_out_mol[3]	$n_{9,\mathrm{H_2}}$	486.682	$\mathrm{kmol\,h^{-1}}$
atr_out_mol[4]	$n_{9,\mathrm{CO}}$	126.977	$\mathrm{kmol\,h^{-1}}$
atr_out_mol[5]	$n_{9,\mathrm{CO_2}}$	84.7013	$\mathrm{kmol\,h^{-1}}$
postATR_in_mol[1]	$n_{9,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
postATR_in_mol[2]	$n_{9,\mathrm{H_2O}}$	582.934	$\mathrm{kmol\,h^{-1}}$
postATR_in_mol[3]	$n_{9,\mathrm{H_2}}$	486.682	$\mathrm{kmol\,h^{-1}}$
postATR_in_mol[4]	$n_{9,\mathrm{CO}}$	126.977	$\mathrm{kmol\,h^{-1}}$
postATR_in_mol[5]	$n_{9,\mathrm{CO_2}}$	84.7013	$\mathrm{kmol\,h^{-1}}$
postATR_out_mol[1]	$n_{10,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
postATR_out_mol[2]	$n_{10,\mathrm{H_2O}}$	582.934	$\mathrm{kmol\,h^{-1}}$
postATR_out_mol[3]	$n_{10,\mathrm{H_2}}$	486.682	$\mathrm{kmol\,h^{-1}}$
postATR_out_mol[4]	$n_{10,\mathrm{CO}}$	126.977	$\mathrm{kmol\,h^{-1}}$
postATR_out_mol[5]	$n_{10,\mathrm{CO_2}}$	84.7013	$\mathrm{kmol\,h^{-1}}$
itsr_in_mol[1]	$n_{10,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
itsr_in_mol[2]	$n_{10,\mathrm{H_2O}}$	582.934	$\mathrm{kmol\,h^{-1}}$
itsr_in_mol[3]	$n_{10,\mathrm{H_2}}$	486.682	$\mathrm{kmol\,h^{-1}}$
itsr_in_mol[4]	$n_{10,\mathrm{CO}}$	126.977	$\mathrm{kmol\,h^{-1}}$
itsr_in_mol[5]	$n_{10,\mathrm{CO_2}}$	84.7013	$\mathrm{kmol\,h^{-1}}$
itsr_out_mol[1]	$n_{11,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
itsr_out_mol[2]	$n_{11,\mathrm{H_2O}}$	464.639	$\mathrm{kmol\,h^{-1}}$
itsr_out_mol[3]	$n_{11,\mathrm{H_2}}$	604.976	$\mathrm{kmol\,h^{-1}}$
itsr_out_mol[4]	$n_{11,\mathrm{CO}}$	8.683	$\mathrm{kmol\,h^{-1}}$
itsr_out_mol[5]	$n_{11,\mathrm{CO_2}}$	202.995	$\mathrm{kmol\,h^{-1}}$
preCond_in_mol[1]	$n_{11,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
preCond_in_mol[2]	$n_{11,\mathrm{H_2O}}$	464.639	$\mathrm{kmol\,h^{-1}}$
preCond_in_mol[3]	$n_{11,\mathrm{H_2}}$	604.976	$\mathrm{kmol\,h^{-1}}$
preCond_in_mol[4]	$n_{11,\mathrm{CO}}$	8.683	$\mathrm{kmol\,h^{-1}}$
preCond_in_mol[5]	$n_{11,\mathrm{CO_2}}$	202.995	$\mathrm{kmol\,h^{-1}}$
preCond_out_mol[1]	$n_{12,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
preCond_out_mol[2]	$n_{12,\mathrm{H_2O}}$	464.639	$\mathrm{kmol\,h^{-1}}$
preCond_out_mol[3]	$n_{12,\mathrm{H_2}}$	604.976	$\mathrm{kmol\,h^{-1}}$

Continued on Next Page

Variable	Variable*	Value	Unit
preCond_out_mol[4]	$n_{12,\mathrm{CO}}$	8.683	$\mathrm{kmol\,h^{-1}}$
preCond_out_mol[5]	$n_{12,\mathrm{CO_2}}$	202.995	$\mathrm{kmol\,h^{-1}}$
cond_in_mol[1]	$n_{12,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
cond_in_mol[2]	$n_{12,\mathrm{H_2O}}$	464.639	$\mathrm{kmol\,h^{-1}}$
cond_in_mol[3]	$n_{12,\mathrm{H_2}}$	604.976	$\mathrm{kmol\,h^{-1}}$
cond_in_mol[4]	$n_{12,\mathrm{CO}}$	8.683	$\mathrm{kmol\,h^{-1}}$
cond_in_mol[5]	$n_{12,\mathrm{CO_2}}$	202.995	$\mathrm{kmol\,h^{-1}}$
cond_L	n_{13}	463.346	$\mathrm{kmol\,h^{-1}}$
cond_liq_frac[1]	$x_{13,\mathrm{CH_4}}$	0.000	-
cond_liq_frac[2]	$x_{13,\mathrm{H_2O}}$	0.999	-
cond_liq_frac[3]	$x_{13,\mathrm{H_2}}$	0.000	-
cond_liq_frac[4]	$x_{13,\mathrm{CO}}$	0.000	-
cond_liq_frac[5]	$x_{13,\mathrm{CO_2}}$	0.000	-
cond_V	n_{14}	818.248	$\mathrm{kmol\,h^{-1}}$
cond_vap_frac[1]	$x_{14,\mathrm{CH_4}}$	0.000	-
cond_vap_frac[2]	$x_{14,\mathrm{H_2O}}$	0.002	-
cond_vap_frac[3]	$x_{14,\mathrm{H_2}}$	0.739	-
cond_vap_frac[4]	$x_{14,\mathrm{CO}}$	0.011	-
cond_vap_frac[5]	$x_{14,\mathrm{CO_2}}$	0.248	-
psa_in_mol[1]	$n_{14,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
psa_in_mol[2]	$n_{14,\mathrm{H_2O}}$	1.294	$\mathrm{kmol\,h^{-1}}$
psa_in_mol[3]	$n_{14,\mathrm{H_2}}$	604.976	$\mathrm{kmol\,h^{-1}}$
psa_in_mol[4]	$n_{14,\mathrm{CO}}$	8.683	$\mathrm{kmol\,h^{-1}}$
psa_in_mol[5]	$n_{14,\mathrm{CO_2}}$	202.995	$\mathrm{kmol\,h^{-1}}$
psa_outPurge_mol[1]	$n_{15,\mathrm{CH_4}}$	0.301	$\mathrm{kmol\,h^{-1}}$
psa_outPurge_mol[2]	$n_{15,\mathrm{H_2O}}$	1.293	$\mathrm{kmol\,h^{-1}}$
psa_outPurge_mol[3]	$n_{15,\mathrm{H_2}}$	18.149	$\mathrm{kmol\,h^{-1}}$
psa_outPurge_mol[4]	$n_{15,\mathrm{CO}}$	8.674	$\mathrm{kmol\,h^{-1}}$
psa_outPurge_mol[5]	$n_{15,\mathrm{CO_2}}$	202.792	$\mathrm{kmol\,h^{-1}}$
psa_outProduct_mol[1]	$n_{16,\mathrm{CH_4}}$	0.000	$\mathrm{kmol\,h^{-1}}$
psa_outProduct_mol[2]	$n_{16,\mathrm{H_2O}}$	0.001	$\mathrm{kmol\,h^{-1}}$
psa_outProduct_mol[3]	$n_{16,\mathrm{H_2}}$	586.826	$\mathrm{kmol\,h^{-1}}$
psa_outProduct_mol[4]	$n_{16,\mathrm{CO}}$	0.009	$\mathrm{kmol\,h^{-1}}$
psa_outProduct_mol[5]	$n_{16,\mathrm{CO_2}}$	0.203	$\mathrm{kmol\,h^{-1}}$
mix_in_T	T_2	311.000	K
H2O_T	T_3	423.000	K
mix_out_T	T_4	397.246	K
prePR_in_T	T_4	397.246	K
prePR_out_T	T_5	644.595	K
preGHR_in_T	T_6	609.2	K
preGHR_out_T	T_7	609.2	K
ghr_in_T	T_7	609.2	K

Continued on Next Page

Variable	Variable*	Value	Unit
ghr_out_T	T_8	951.482	K
atr_in_T	T_8	951.482	K
postATR_in_T	T_9	1291.820	K
postATR_out_T	T_{10}	634.2	K
preCond_out_T	T_{12}	293.0	K
cond_in_T	T_{12}	293.0	K
cond_L_T	T_{13}	293.0	K
cond_V_T	T_{14}	293.0	K
psa_in_T	T_{14}	293.0	K
psa_outPurge_T	T_{15}	293.0	K
psa_outProduct_T	T_{16}	293.0	K
prePR_Q	Q_{prePR}	8.520e6	$\mathrm{MJ\,h^{-1}}$
preGHR_Q	Q_{preGHR}	0.006	$\mathrm{MJ\,h^{-1}}$
ghr_Q	Q_{ghr}	3.074e7	$\mathrm{MJ\,h^{-1}}$
postATR_Q	$Q_{postATR}$	-3.074e7	$\mathrm{MJ\,h^{-1}}$
itsr_Q	Q_{itsr}	-1.151e7	$\mathrm{MJ\,h^{-1}}$
preCond_Q	$Q_{preCond}$	-7.658e6	$\mathrm{MJ\,h^{-1}}$
F_H2	n_{18}	556.999	$\mathrm{kmol\,h^{-1}}$
F_H2_heat	n_{17}	29.827	$\mathrm{kmol\,h^{-1}}$
F_NG	n_1	145.400	$\mathrm{kmol\,h^{-1}}$
F_NG_heat	n_{19}	0.000	$\mathrm{kmol\,h^{-1}}$
F_fluegas	n_{20}	0.000	$\mathrm{kmol\,h^{-1}}$
F_inj	n_{21}	231.208	$\mathrm{kmol\,h^{-1}}$

A.1.1 Gain matrix of the output with respect to input change

Table 10: Gain matrix Gy, which represents the output sensitivity with respect to input change

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
mix_in_mol[1]	$n_{2,\mathrm{CH_4}}$	1.345E-05	1.653E-06	-1.471E-02
mix_in_mol[2]	$n_{2,\mathrm{H_2O}}$	1.373E-41	1.722E-42	-9.632E-31
mix_in_mol[3]	$n_{2,\mathrm{H_2}}$	-1.144E-39	1.516E-40	2.628E-29
mix_in_mol[4]	$n_{2,\mathrm{CO}}$	2.513E-39	1.283E-39	-5.549E-39
mix_in_mol[5]	$n_{2,\mathrm{CO_2}}$	2.303E-07	2.830E-08	-2.520E-04
mix_in_mol[6]	$n_{2,\mathrm{C_2H_6}}$	1.048E-06	1.289E-07	-1.147E-03
mix_in_mol[7]	$n_{2,\mathrm{C_3H_8}}$	1.151E-06	1.415E-07	-1.260E-03
mix_in_mol[8]	$n_{2,\mathrm{i\text{-}C_4H_{10}}}$	4.262E-07	5.239E-08	-4.664E-04
mix_in_mol[9]	$n_{2,\mathrm{n\text{-}C_4H_{10}}}$	2.423E-07	2.978E-08	-2.652E-04
mix_in_mol[10]	$n_{2,\mathrm{C_5H_{12}}}$	6.307E-07	7.752E-08	-6.902E-04
H2Ostream	n_3	-1.179E+02	-1.714E+00	-9.236E-02

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
mix_out_mol[1]	$n_{4,\mathrm{CH_4}}$	1.345E-05	1.653E-06	-1.471E-02
mix_out_mol[2]	$n_{4,\mathrm{H_2O}}$	-1.179E+02	-1.714E+00	-9.236E-02
mix_out_mol[3]	$n_{4,\mathrm{H_2}}$	-3.409E-40	4.471E-40	-1.314E-29
mix_out_mol[4]	$n_{4,\mathrm{CO}}$	2.864E-39	-5.035E-40	-2.315E-29
mix_out_mol[5]	$n_{4,\mathrm{CO_2}}$	2.303E-07	2.830E-08	-2.520E-04
mix_out_mol[6]	$n_{4,\mathrm{C_2H_6}}$	1.048E-06	1.289E-07	-1.147E-03
mix_out_mol[7]	$n_{4,\mathrm{C_3H_8}}$	1.151E-06	1.415E-07	-1.260E-03
mix_out_mol[8]	$n_{4,\mathrm{i\text{-}C_4H_{10}}}$	4.262E-07	5.239E-08	-4.664E-04
mix_out_mol[9]	$n_{4,\mathrm{n\text{-}C_4H_{10}}}$	2.423E-07	2.978E-08	-2.652E-04
mix_out_mol[10]	$n_{4,\mathrm{C_5H_{12}}}$	6.307E-07	7.752E-08	-6.902E-04
prePR_in_mol[1]	$n_{4,\mathrm{CH_4}}$	1.345E-05	1.653E-06	-1.471E-02
prePR_in_mol[2]	$n_{4,\mathrm{H_2O}}$	-1.179E+02	-1.714E+00	-9.236E-02
prePR_in_mol[3]	$n_{4,\mathrm{H_2}}$	-1.721E-39	-1.387E-39	2.178E-39
prePR_in_mol[4]	$n_{4,\mathrm{CO}}$	-6.411E-39	-9.190E-40	-4.629E-29
prePR_in_mol[5]	$n_{4,\mathrm{CO_2}}$	2.303E-07	2.830E-08	-2.520E-04
prePR_in_mol[6]	$n_{4,\mathrm{C_2H_6}}$	1.048E-06	1.289E-07	-1.147E-03
prePR_in_mol[7]	$n_{4,\mathrm{C_3H_8}}$	1.151E-06	1.415E-07	-1.260E-03
prePR_in_mol[8]	$n_{4,\mathrm{i\text{-}C_4H_{10}}}$	4.262E-07	5.239E-08	-4.664E-04
prePR_in_mol[9]	$n_{4,\mathrm{n\text{-}C_4H_{10}}}$	2.423E-07	2.978E-08	-2.652E-04
prePR_in_mol[10]	$n_{4,\mathrm{C_5H_{12}}}$	6.307E-07	7.752E-08	-6.902E-04
prePR_out_mol[1]	$n_{5,\mathrm{CH_4}}$	1.345E-05	1.653E-06	-1.471E-02
prePR_out_mol[2]	$n_{5,\mathrm{H_2O}}$	-1.179E+02	-1.714E+00	-9.236E-02
prePR_out_mol[3]	$n_{5,\mathrm{H_2}}$	-2.580E-39	-4.000E-40	6.570E-30
prePR_out_mol[4]	$n_{5,\mathrm{CO}}$	-3.462E-39	7.104E-40	-4.827E-29
prePR_out_mol[5]	$n_{5,\mathrm{CO_2}}$	2.303E-07	2.830E-08	-2.520E-04
prePR_out_mol[6]	$n_{5,\mathrm{C_2H_6}}$	1.048E-06	1.289E-07	-1.147E-03
prePR_out_mol[7]	$n_{5,\mathrm{C_3H_8}}$	1.151E-06	1.415E-07	-1.260E-03
prePR_out_mol[8]	$n_{5,\mathrm{i\text{-}C_4H_{10}}}$	4.262E-07	5.239E-08	-4.664E-04
prePR_out_mol[9]	$n_{5,\mathrm{n\text{-}C_4H_{10}}}$	2.423E-07	2.978E-08	-2.652E-04
prePR_out_mol[10]	$n_{5,\mathrm{C_5H_{12}}}$	6.307E-07	7.752E-08	-6.902E-04
pr_in_mol[1]	$n_{5,\mathrm{CH_4}}$	1.345E-05	1.653E-06	-1.471E-02
pr_in_mol[2]	$n_{5,\mathrm{H_2O}}$	-1.179E+02	-1.714E+00	-9.236E-02
pr_in_mol[3]	$n_{5,\mathrm{H_2}}$	-1.701E-39	-1.736E-40	-2.628E-29
pr_in_mol[4]	$n_{5,\mathrm{CO}}$	-3.800E-39	-2.007E-40	-1.448E-28
pr_in_mol[5]	$n_{5,\mathrm{CO_2}}$	2.303E-07	2.830E-08	-2.520E-04
pr_in_mol[6]	$n_{5,\mathrm{C_2H_6}}$	1.048E-06	1.289E-07	-1.147E-03
pr_in_mol[7]	$n_{5,\mathrm{C_3H_8}}$	1.151E-06	1.415E-07	-1.260E-03
pr_in_mol[8]	$n_{5,\mathrm{i\text{-}C_4H_{10}}}$	4.262E-07	5.239E-08	-4.664E-04
pr_in_mol[9]	$n_{5,\mathrm{n\text{-}C_4H_{10}}}$	2.423E-07	2.978E-08	-2.652E-04
pr_in_mol[10]	$n_{5,\mathrm{C_5H_{12}}}$	6.307E-07	7.752E-08	-6.902E-04
pr_out_mol[1]	$n_{6,\mathrm{CH_4}}$	1.543E+00	-4.347E-02	-2.323E-02
pr_out_mol[2]	$n_{6,\mathrm{H_2O}}$	-1.148E+02	-1.798E+00	-8.451E-02

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
pr_out_mol[3]	$n_{6,\mathrm{H_2}}$	-6.179E+00	1.712E-01	-7.113E-03
pr_out_mol[4]	$n_{6,\mathrm{CO}}$	8.454E-03	2.707E-03	-2.950E-05
pr_out_mol[5]	$n_{6,\mathrm{CO_2}}$	-1.551E+00	4.077E-02	-4.164E-03
pr_out_mol[6]	$n_{6,\mathrm{C_2H_6}}$	5.120E-13	-3.256E-14	4.636E-14
pr_out_mol[7]	$n_{6,\mathrm{C_3H_8}}$	8.221E-13	1.049E-13	5.304E-14
pr_out_mol[8]	$n_{6,\mathrm{i\text{-}C_4H_{10}}}$	-2.359E-13	-1.335E-14	1.057E-15
pr_out_mol[9]	$n_{6,\mathrm{n\text{-}C_4H_{10}}}$	-2.645E-14	-6.822E-15	-4.443E-15
pr_out_mol[10]	$n_{6,\mathrm{C_5H_{12}}}$	1.089E-12	5.318E-14	4.204E-14
preGHR_in_mol[1]	$n_{6,\mathrm{CH_4}}$	1.543E+00	-4.347E-02	-2.323E-02
preGHR_in_mol[2]	$n_{6,\mathrm{H_2O}}$	-1.148E+02	-1.798E+00	-8.451E-02
preGHR_in_mol[3]	$n_{6,\mathrm{H_2}}$	-6.179E+00	1.712E-01	-7.113E-03
preGHR_in_mol[4]	$n_{6,\mathrm{CO}}$	8.454E-03	2.707E-03	-2.950E-05
preGHR_in_mol[5]	$n_{6,\mathrm{CO_2}}$	-1.551E+00	4.077E-02	-4.164E-03
preGHR_out_mol[1]	$n_{7,\mathrm{CH_4}}$	1.543E+00	-4.347E-02	-2.323E-02
preGHR_out_mol[2]	$n_{7,\mathrm{H_2O}}$	-1.148E+02	-1.798E+00	-8.451E-02
preGHR_out_mol[3]	$n_{7,\mathrm{H_2}}$	-6.179E+00	1.712E-01	-7.113E-03
preGHR_out_mol[4]	$n_{7,\mathrm{CO}}$	8.454E-03	2.707E-03	-2.950E-05
preGHR_out_mol[5]	$n_{7,\mathrm{CO_2}}$	-1.551E+00	4.077E-02	-4.164E-03
ghr_in_mol[1]	$n_{7,\mathrm{CH_4}}$	1.543E+00	-4.347E-02	-2.323E-02
ghr_in_mol[2]	$n_{7,\mathrm{H_2O}}$	-1.148E+02	-1.798E+00	-8.451E-02
ghr_in_mol[3]	$n_{7,\mathrm{H_2}}$	-6.179E+00	1.712E-01	-7.113E-03
ghr_in_mol[4]	$n_{7,\mathrm{CO}}$	8.454E-03	2.707E-03	-2.950E-05
ghr_in_mol[5]	$n_{7,\mathrm{CO_2}}$	-1.551E+00	4.077E-02	-4.164E-03
ghr_out_mol[1]	$n_{8,\mathrm{CH_4}}$	9.903E+00	1.218E-01	-1.713E-01
ghr_out_mol[2]	$n_{8,\mathrm{H_2O}}$	-9.891E+01	-1.465E+00	-2.619E-01
ghr_out_mol[3]	$n_{8,\mathrm{H_2}}$	-3.880E+01	-4.924E-01	4.665E-01
ghr_out_mol[4]	$n_{8,\mathrm{CO}}$	-8.126E-01	5.326E-03	1.188E-01
ghr_out_mol[5]	$n_{8,\mathrm{CO_2}}$	-9.090E+00	-1.271E-01	2.515E-02
atr_in_mol[1]	$n_{8,\mathrm{CH_4}}$	9.903E+00	1.218E-01	-1.713E-01
atr_in_mol[2]	$n_{8,\mathrm{H_2O}}$	-9.891E+01	-1.465E+00	-2.619E-01
atr_in_mol[3]	$n_{8,\mathrm{H_2}}$	-3.880E+01	-4.924E-01	4.665E-01
atr_in_mol[4]	$n_{8,\mathrm{CO}}$	-8.126E-01	5.326E-03	1.188E-01
atr_in_mol[5]	$n_{8,\mathrm{CO_2}}$	-9.090E+00	-1.271E-01	2.515E-02
atr_out_mol[1]	$n_{9,\mathrm{CH_4}}$	1.106E-01	1.675E-03	-5.176E-03
atr_out_mol[2]	$n_{9,\mathrm{H_2O}}$	-1.075E+02	-1.587E+00	4.223E-02
atr_out_mol[3]	$n_{9,\mathrm{H_2}}$	-1.064E+01	-1.304E-01	-1.699E-01
atr_out_mol[4]	$n_{9,\mathrm{CO}}$	8.193E+00	1.237E-01	9.061E-02
atr_out_mol[5]	$n_{9,\mathrm{CO_2}}$	-8.304E+00	-1.254E-01	-1.129E-01
postATR_in_mol[1]	$n_{9,\mathrm{CH_4}}$	1.106E-01	1.675E-03	-5.176E-03
postATR_in_mol[2]	$n_{9,\mathrm{H_2O}}$	-1.075E+02	-1.587E+00	4.223E-02
postATR_in_mol[3]	$n_{9,\mathrm{H_2}}$	-1.064E+01	-1.304E-01	-1.699E-01
postATR_in_mol[4]	$n_{9,\mathrm{CO}}$	8.193E+00	1.237E-01	9.061E-02

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
postATR_in_mol[5]	n_{9,CO_2}	-8.304E+00	-1.254E-01	-1.129E-01
postATR_out_mol[1]	n_{10,CH_4}	1.106E-01	1.675E-03	-5.176E-03
postATR_out_mol[2]	n_{10,H_2O}	-1.075E+02	-1.587E+00	4.223E-02
postATR_out_mol[3]	n_{10,H_2}	-1.064E+01	-1.304E-01	-1.699E-01
postATR_out_mol[4]	$n_{10,CO}$	8.193E+00	1.237E-01	9.061E-02
postATR_out_mol[5]	n_{10,CO_2}	-8.304E+00	-1.254E-01	-1.129E-01
itsr_in_mol[1]	n_{10,CH_4}	1.106E-01	1.675E-03	-5.176E-03
itsr_in_mol[2]	n_{10,H_2O}	-1.075E+02	-1.587E+00	4.223E-02
itsr_in_mol[3]	n_{10,H_2}	-1.064E+01	-1.304E-01	-1.699E-01
itsr_in_mol[4]	$n_{10,CO}$	8.193E+00	1.237E-01	9.061E-02
itsr_in_mol[5]	n_{10,CO_2}	-8.304E+00	-1.254E-01	-1.129E-01
itsr_out_mol[1]	n_{11,CH_4}	1.106E-01	1.675E-03	-5.176E-03
itsr_out_mol[2]	n_{11,H_2O}	-1.137E+02	-1.681E+00	-4.948E-02
itsr_out_mol[3]	n_{11,H_2}	-4.415E+00	-3.625E-02	-7.824E-02
itsr_out_mol[4]	$n_{11,CO}$	1.973E+00	2.956E-02	-1.102E-03
itsr_out_mol[5]	n_{11,CO_2}	-2.083E+00	-3.123E-02	-2.114E-02
preCond_in_mol[1]	n_{11,CH_4}	1.106E-01	1.675E-03	-5.176E-03
preCond_in_mol[2]	n_{11,H_2O}	-1.137E+02	-1.681E+00	-4.948E-02
preCond_in_mol[3]	n_{11,H_2}	-4.415E+00	-3.625E-02	-7.824E-02
preCond_in_mol[4]	$n_{11,CO}$	1.973E+00	2.956E-02	-1.102E-03
preCond_in_mol[5]	n_{11,CO_2}	-2.083E+00	-3.123E-02	-2.114E-02
preCond_out_mol[1]	n_{12,CH_4}	1.106E-01	1.675E-03	-5.176E-03
preCond_out_mol[2]	n_{12,H_2O}	-1.137E+02	-1.681E+00	-4.948E-02
preCond_out_mol[3]	n_{12,H_2}	-4.415E+00	-3.625E-02	-7.824E-02
preCond_out_mol[4]	$n_{12,CO}$	1.973E+00	2.956E-02	-1.102E-03
preCond_out_mol[5]	n_{12,CO_2}	-2.083E+00	-3.123E-02	-2.114E-02
cond_in_mol[1]	n_{12,CH_4}	1.106E-01	1.675E-03	-5.176E-03
cond_in_mol[2]	n_{12,H_2O}	-1.137E+02	-1.681E+00	-4.948E-02
cond_in_mol[3]	n_{12,H_2}	-4.415E+00	-3.625E-02	-7.824E-02
cond_in_mol[4]	$n_{12,CO}$	1.973E+00	2.956E-02	-1.102E-03
cond_in_mol[5]	n_{12,CO_2}	-2.083E+00	-3.123E-02	-2.114E-02
cond_L	n_{13}	-1.137E+02	-1.681E+00	-4.931E-02
cond_liq_frac[1]	x_{13,CH_4}	1.371E-10	2.064E-12	-6.278E-12
cond_liq_frac[2]	x_{13,H_2O}	2.800E-13	1.722E-14	8.594E-15
cond_liq_frac[3]	x_{13,H_2}	-1.400E-09	-1.150E-11	4.145E-15
cond_liq_frac[4]	$x_{13,CO}$	2.468E-09	3.659E-11	2.503E-14
cond_liq_frac[5]	x_{13,CO_2}	-1.205E-09	-2.716E-11	6.249E-12
cond_V	n_{14}	-4.422E+00	-3.630E-02	-1.058E-01
cond_vap_frac[1]	x_{14,CH_4}	1.371E-04	2.064E-06	-6.278E-06
cond_vap_frac[2]	x_{14,H_2O}	4.649E-15	6.728E-17	-2.518E-16
cond_vap_frac[3]	x_{14,H_2}	-1.400E-03	-1.150E-05	4.145E-09
cond_vap_frac[4]	$x_{14,CO}$	2.468E-03	3.659E-05	2.503E-08

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
cond_vap_frac[5]	x_{14,CO_2}	-1.205E-03	-2.716E-05	6.249E-06
psa_in_mol[1]	n_{14,CH_4}	1.106E-01	1.675E-03	-5.176E-03
psa_in_mol[2]	n_{14,H_2O}	-6.993E-03	-5.740E-05	-1.673E-04
psa_in_mol[3]	n_{14,H_2}	-4.415E+00	-3.625E-02	-7.824E-02
psa_in_mol[4]	$n_{14,CO}$	1.973E+00	2.956E-02	-1.102E-03
psa_in_mol[5]	n_{14,CO_2}	-2.083E+00	-3.123E-02	-2.114E-02
psa_outPurge_mol[1]	n_{15,CH_4}	1.106E-04	1.675E-06	-5.176E-06
psa_outPurge_mol[2]	n_{15,H_2O}	-6.993E-06	-5.740E-08	-1.673E-07
psa_outPurge_mol[3]	n_{15,H_2}	-4.283E+00	-3.516E-02	-7.589E-02
psa_outPurge_mol[4]	$n_{15,CO}$	1.973E-03	2.956E-05	-1.102E-06
psa_outPurge_mol[5]	n_{15,CO_2}	-2.083E-03	-3.123E-05	-2.114E-05
psa_outProduct_mol[1]	n_{16,CH_4}	1.105E-01	1.674E-03	-5.170E-03
psa_outProduct_mol[2]	n_{16,H_2O}	-6.986E-03	-5.735E-05	-1.672E-04
psa_outProduct_mol[3]	n_{16,H_2}	-1.324E-01	-1.087E-03	-2.347E-03
psa_outProduct_mol[4]	$n_{16,CO}$	1.971E+00	2.953E-02	-1.101E-03
psa_outProduct_mol[5]	n_{16,CO_2}	-2.081E+00	-3.120E-02	-2.112E-02
mix_in_T	T_2	0.000E+00	0.000E+00	0.000E+00
H2O_T	T_3	-3.217E+00	-4.676E-02	9.732E-06
mix_out_T	T_4	0.000E+00	0.000E+00	0.000E+00
prePR_in_T	T_4	-3.217E+00	-4.676E-02	9.732E-06
prePR_out_T	T_5	0.000E+00	1.000E+00	0.000E+00
preGHR_in_T	T_6	3.428E+00	7.334E-01	-1.037E-05
preGHR_out_T	T_7	3.428E+00	7.334E-01	-1.269E-05
ghr_in_T	T_7	3.428E+00	7.334E-01	-1.269E-05
ghr_out_T	T_8	-2.942E-01	4.405E-02	3.468E-01
atr_in_T	T_8	-2.942E-01	4.405E-02	3.468E-01
postATR_in_T	T_9	0.000E+00	0.000E+00	1.000E+00
postATR_out_T	T_{10}	3.428E+00	7.335E-01	-1.982E-05
preCond_out_T	T_{12}	0.000E+00	0.000E+00	-4.400E-12
cond_in_T	T_{12}	0.000E+00	0.000E+00	-4.400E-12
cond_L_T	T_{13}	0.000E+00	0.000E+00	-4.400E-12
cond_V_T	T_{14}	0.000E+00	0.000E+00	-4.400E-12
psa_in_T	T_{14}	0.000E+00	0.000E+00	-4.400E-12
psa_outPurge_T	T_{15}	0.000E+00	0.000E+00	-4.400E-12
psa_outProduct_T	T_{16}	0.000E+00	0.000E+00	-4.400E-12
prePR_Q	Q_{prePR}	-9.219E+05	2.350E+04	-1.099E+03
preGHR_Q	Q_{preGHR}	7.883E-01	9.685E-02	-8.458E-02
ghr_Q	Q_{ghr}	-3.314E+06	-7.815E+04	4.568E+04
postATR_Q	$Q_{postATR}$	3.314E+06	7.815E+04	-4.568E+04
itsr_Q	Q_{itsr}	2.959E+05	-2.558E+04	-2.561E+03
preCond_Q	$Q_{preCond}$	7.138E+05	1.041E+04	9.203E+02
F_H2	n_{18}	-1.055E+00	-1.174E-01	5.667E-03

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial u_1}$	$\frac{\partial y}{\partial u_2}$	$\frac{\partial y}{\partial u_3}$
F_H2_heat	n_{17}	-3.227E+00	8.228E-02	-8.156E-02
F_NG	n_1	1.718E-05	2.112E-06	-1.881E-02
F_NG_heat	n_{19}	-1.718E-05	-2.112E-06	1.881E-02
F_fluegas	n_{20}	-6.069E-05	-7.460E-06	6.642E-02
F_inj	n_{21}	-1.394E-01	-1.149E-03	3.651E-02

A.1.2 Gain matrix of the output with respect to disturbance change

Table 11: Gain matrix Gyd, which represents the output sensitivity with respect to disturbance change

Variable	Variable*	$\frac{\partial y}{\partial d_1}$	$\frac{\partial y}{\partial d_2}$	$\frac{\partial y}{\partial d_3}$
mix_in_mol[1]	$n_{2,\mathrm{CH_4}}$	4.705E-05	2.639E-07	2.123E-09
mix_in_mol[2]	$n_{2,\mathrm{H_2O}}$	1.607E-40	5.263E-38	-3.310E-39
mix_in_mol[3]	$n_{2,\mathrm{H_2}}$	1.971E-38	-6.441E-26	2.888E-37
mix_in_mol[4]	$n_{2,\mathrm{CO}}$	-7.057E-38	-2.115E-25	9.141E-38
mix_in_mol[5]	$n_{2,\mathrm{CO_2}}$	8.059E-07	4.441E-09	3.980E-11
mix_in_mol[6]	$n_{2,\mathrm{C_2H_6}}$	3.669E-06	2.030E-08	1.592E-10
mix_in_mol[7]	$n_{2,\mathrm{C_3H_8}}$	4.029E-06	2.284E-08	2.654E-10
mix_in_mol[8]	$n_{2,\mathrm{i\text{-}C_4H_{10}}}$	1.491E-06	8.565E-09	7.961E-11
mix_in_mol[9]	$n_{2,\mathrm{n\text{-}C_4H_{10}}}$	8.480E-07	4.441E-09	5.307E-11
mix_in_mol[10]	$n_{2,\mathrm{C_5H_{12}}}$	2.207E-06	1.269E-08	1.327E-10
H2Ostream	n_3	3.466E-03	1.713E-05	5.095E-08
mix_out_mol[1]	$n_{4,\mathrm{CH_4}}$	4.705E-05	2.639E-07	2.123E-09
mix_out_mol[2]	$n_{4,\mathrm{H_2O}}$	3.466E-03	1.713E-05	5.095E-08
mix_out_mol[3]	$n_{4,\mathrm{H_2}}$	5.459E-39	-3.221E-26	-5.086E-40
mix_out_mol[4]	$n_{4,\mathrm{CO}}$	-1.247E-38	-9.678E-26	-8.315E-38
mix_out_mol[5]	$n_{4,\mathrm{CO_2}}$	8.059E-07	4.441E-09	3.980E-11
mix_out_mol[6]	$n_{4,\mathrm{C_2H_6}}$	3.669E-06	2.030E-08	1.592E-10
mix_out_mol[7]	$n_{4,\mathrm{C_3H_8}}$	4.029E-06	2.284E-08	2.654E-10
mix_out_mol[8]	$n_{4,\mathrm{i\text{-}C_4H_{10}}}$	1.491E-06	8.565E-09	7.961E-11
mix_out_mol[9]	$n_{4,\mathrm{n\text{-}C_4H_{10}}}$	8.480E-07	4.441E-09	5.307E-11
mix_out_mol[10]	$n_{4,\mathrm{C_5H_{12}}}$	2.207E-06	1.269E-08	1.327E-10
prePR_in_mol[1]	$n_{4,\mathrm{CH_4}}$	4.705E-05	2.639E-07	2.123E-09
prePR_in_mol[2]	$n_{4,\mathrm{H_2O}}$	3.466E-03	1.713E-05	5.095E-08
prePR_in_mol[3]	$n_{4,\mathrm{H_2}}$	-3.429E-38	-5.305E-26	-3.009E-37
prePR_in_mol[4]	$n_{4,\mathrm{CO}}$	4.138E-38	2.670E-26	-1.103E-37
prePR_in_mol[5]	$n_{4,\mathrm{CO_2}}$	8.059E-07	4.441E-09	3.980E-11
prePR_in_mol[6]	$n_{4,\mathrm{C_2H_6}}$	3.669E-06	2.030E-08	1.592E-10
prePR_in_mol[7]	$n_{4,\mathrm{C_3H_8}}$	4.029E-06	2.284E-08	2.654E-10
prePR_in_mol[8]	$n_{4,\mathrm{i\text{-}C_4H_{10}}}$	1.491E-06	8.565E-09	7.961E-11

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial d_1}$	$\frac{\partial y}{\partial d_2}$	$\frac{\partial y}{\partial d_3}$
prePR_in_mol[9]	$n_{4,\text{n-C}_4\text{H}_{10}}$	8.480E-07	4.441E-09	5.307E-11
prePR_in_mol[10]	$n_{4,\text{C}_5\text{H}_{12}}$	2.207E-06	1.269E-08	1.327E-10
prePR_out_mol[1]	n_{5,CH_4}	4.705E-05	2.639E-07	2.123E-09
prePR_out_mol[2]	$n_{5,\text{H}_2\text{O}}$	3.466E-03	1.713E-05	5.095E-08
prePR_out_mol[3]	n_{5,H_2}	-7.932E-39	9.473E-28	-1.208E-37
prePR_out_mol[4]	$n_{5,\text{CO}}$	1.573E-38	-2.227E-25	5.298E-38
prePR_out_mol[5]	n_{5,CO_2}	8.059E-07	4.441E-09	3.980E-11
prePR_out_mol[6]	$n_{5,\text{C}_2\text{H}_6}$	3.669E-06	2.030E-08	1.592E-10
prePR_out_mol[7]	$n_{5,\text{C}_3\text{H}_8}$	4.029E-06	2.284E-08	2.654E-10
prePR_out_mol[8]	$n_{5,\text{i-C}_4\text{H}_{10}}$	1.491E-06	8.565E-09	7.961E-11
prePR_out_mol[9]	$n_{5,\text{n-C}_4\text{H}_{10}}$	8.480E-07	4.441E-09	5.307E-11
prePR_out_mol[10]	$n_{5,\text{C}_5\text{H}_{12}}$	2.207E-06	1.269E-08	1.327E-10
pr_in_mol[1]	n_{5,CH_4}	4.705E-05	2.639E-07	2.123E-09
pr_in_mol[2]	$n_{5,\text{H}_2\text{O}}$	3.466E-03	1.713E-05	5.095E-08
pr_in_mol[3]	n_{5,H_2}	-1.223E-38	6.063E-26	-6.113E-39
pr_in_mol[4]	$n_{5,\text{CO}}$	5.656E-38	-8.352E-26	2.206E-38
pr_in_mol[5]	n_{5,CO_2}	8.059E-07	4.441E-09	3.980E-11
pr_in_mol[6]	$n_{5,\text{C}_2\text{H}_6}$	3.669E-06	2.030E-08	1.592E-10
pr_in_mol[7]	$n_{5,\text{C}_3\text{H}_8}$	4.029E-06	2.284E-08	2.654E-10
pr_in_mol[8]	$n_{5,\text{i-C}_4\text{H}_{10}}$	1.491E-06	8.565E-09	7.961E-11
pr_in_mol[9]	$n_{5,\text{n-C}_4\text{H}_{10}}$	8.480E-07	4.441E-09	5.307E-11
pr_in_mol[10]	$n_{5,\text{C}_5\text{H}_{12}}$	2.207E-06	1.269E-08	1.327E-10
pr_out_mol[1]	n_{6,CH_4}	3.278E-05	2.436E-07	3.397E-09
pr_out_mol[2]	$n_{6,\text{H}_2\text{O}}$	3.358E-03	1.673E-05	5.095E-08
pr_out_mol[3]	n_{6,H_2}	1.889E-04	8.527E-07	2.548E-09
pr_out_mol[4]	$n_{6,\text{CO}}$	-1.330E-07	-1.427E-09	-5.805E-12
pr_out_mol[5]	n_{6,CO_2}	5.503E-05	2.436E-07	4.246E-10
pr_out_mol[6]	$n_{6,\text{C}_2\text{H}_6}$	-1.581E-13	2.706E-11	2.016E-11
pr_out_mol[7]	$n_{6,\text{C}_3\text{H}_8}$	-2.202E-13	-4.907E-10	1.880E-11
pr_out_mol[8]	$n_{6,\text{i-C}_4\text{H}_{10}}$	-1.207E-13	8.759E-12	3.956E-12
pr_out_mol[9]	$n_{6,\text{n-C}_4\text{H}_{10}}$	-7.325E-14	-3.300E-11	-4.220E-12
pr_out_mol[10]	$n_{6,\text{C}_5\text{H}_{12}}$	3.465E-13	1.911E-10	3.059E-12
preGHR_in_mol[1]	n_{6,CH_4}	3.278E-05	2.436E-07	3.397E-09
preGHR_in_mol[2]	$n_{6,\text{H}_2\text{O}}$	3.358E-03	1.673E-05	5.095E-08
preGHR_in_mol[3]	n_{6,H_2}	1.889E-04	8.527E-07	2.548E-09
preGHR_in_mol[4]	$n_{6,\text{CO}}$	-1.330E-07	-1.427E-09	-5.805E-12
preGHR_in_mol[5]	n_{6,CO_2}	5.503E-05	2.436E-07	4.246E-10
preGHR_out_mol[1]	n_{7,CH_4}	3.278E-05	2.436E-07	3.397E-09
preGHR_out_mol[2]	$n_{7,\text{H}_2\text{O}}$	3.358E-03	1.673E-05	5.095E-08
preGHR_out_mol[3]	n_{7,H_2}	1.889E-04	8.527E-07	2.548E-09
preGHR_out_mol[4]	$n_{7,\text{CO}}$	-1.330E-07	-1.427E-09	-5.805E-12
preGHR_out_mol[5]	n_{7,CO_2}	5.503E-05	2.436E-07	4.246E-10

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial d_1}$	$\frac{\partial y}{\partial d_2}$	$\frac{\partial y}{\partial d_3}$
ghr_in_mol[1]	$n_{7,\mathrm{CH_4}}$	3.278E-05	2.436E-07	3.397E-09
ghr_in_mol[2]	$n_{7,\mathrm{H_2O}}$	3.358E-03	1.673E-05	5.095E-08
ghr_in_mol[3]	$n_{7,\mathrm{H_2}}$	1.889E-04	8.527E-07	2.548E-09
ghr_in_mol[4]	$n_{7,\mathrm{CO}}$	-1.330E-07	-1.427E-09	-5.805E-12
ghr_in_mol[5]	$n_{7,\mathrm{CO_2}}$	5.503E-05	2.436E-07	4.246E-10
ghr_out_mol[1]	$n_{8,\mathrm{CH_4}}$	-2.361E-04	-1.167E-06	-4.246E-09
ghr_out_mol[2]	$n_{8,\mathrm{H_2O}}$	2.862E-03	1.413E-05	4.076E-08
ghr_out_mol[3]	$n_{8,\mathrm{H_2}}$	1.222E-03	6.212E-06	2.887E-08
ghr_out_mol[4]	$n_{8,\mathrm{CO}}$	4.231E-05	2.436E-07	3.184E-09
ghr_out_mol[5]	$n_{8,\mathrm{CO_2}}$	2.815E-04	1.391E-06	4.670E-09
atr_in_mol[1]	$n_{8,\mathrm{CH_4}}$	-2.361E-04	-1.167E-06	-4.246E-09
atr_in_mol[2]	$n_{8,\mathrm{H_2O}}$	2.862E-03	1.413E-05	4.076E-08
atr_in_mol[3]	$n_{8,\mathrm{H_2}}$	1.222E-03	6.212E-06	2.887E-08
atr_in_mol[4]	$n_{8,\mathrm{CO}}$	4.231E-05	2.436E-07	3.184E-09
atr_in_mol[5]	$n_{8,\mathrm{CO_2}}$	2.815E-04	1.391E-06	4.670E-09
atr_out_mol[1]	$n_{9,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
atr_out_mol[2]	$n_{9,\mathrm{H_2O}}$	3.121E-03	1.543E-05	4.416E-08
atr_out_mol[3]	$n_{9,\mathrm{H_2}}$	4.970E-04	2.680E-06	1.868E-08
atr_out_mol[4]	$n_{9,\mathrm{CO}}$	-1.658E-04	-7.714E-07	4.246E-10
atr_out_mol[5]	$n_{9,\mathrm{CO_2}}$	2.563E-04	1.269E-06	3.821E-09
postATR_in_mol[1]	$n_{9,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
postATR_in_mol[2]	$n_{9,\mathrm{H_2O}}$	3.121E-03	1.543E-05	4.416E-08
postATR_in_mol[3]	$n_{9,\mathrm{H_2}}$	4.970E-04	2.680E-06	1.868E-08
postATR_in_mol[4]	$n_{9,\mathrm{CO}}$	-1.658E-04	-7.714E-07	4.246E-10
postATR_in_mol[5]	$n_{9,\mathrm{CO_2}}$	2.563E-04	1.269E-06	3.821E-09
postATR_out_mol[1]	$n_{10,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
postATR_out_mol[2]	$n_{10,\mathrm{H_2O}}$	3.121E-03	1.543E-05	4.416E-08
postATR_out_mol[3]	$n_{10,\mathrm{H_2}}$	4.970E-04	2.680E-06	1.868E-08
postATR_out_mol[4]	$n_{10,\mathrm{CO}}$	-1.658E-04	-7.714E-07	4.246E-10
postATR_out_mol[5]	$n_{10,\mathrm{CO_2}}$	2.563E-04	1.269E-06	3.821E-09
itsr_in_mol[1]	$n_{10,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
itsr_in_mol[2]	$n_{10,\mathrm{H_2O}}$	3.121E-03	1.543E-05	4.416E-08
itsr_in_mol[3]	$n_{10,\mathrm{H_2}}$	4.970E-04	2.680E-06	1.868E-08
itsr_in_mol[4]	$n_{10,\mathrm{CO}}$	-1.658E-04	-7.714E-07	4.246E-10
itsr_in_mol[5]	$n_{10,\mathrm{CO_2}}$	2.563E-04	1.269E-06	3.821E-09
itsr_out_mol[1]	$n_{11,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
itsr_out_mol[2]	$n_{11,\mathrm{H_2O}}$	3.238E-03	1.596E-05	3.397E-08
itsr_out_mol[3]	$n_{11,\mathrm{H_2}}$	3.803E-04	2.030E-06	2.717E-08
itsr_out_mol[4]	$n_{11,\mathrm{CO}}$	-4.907E-05	-2.309E-07	-9.022E-09
itsr_out_mol[5]	$n_{11,\mathrm{CO_2}}$	1.396E-04	7.511E-07	1.359E-08
preCond_in_mol[1]	$n_{11,\mathrm{CH_4}}$	-2.820E-06	-1.431E-08	-5.307E-11
preCond_in_mol[2]	$n_{11,\mathrm{H_2O}}$	3.238E-03	1.596E-05	3.397E-08

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial d_1}$	$\frac{\partial y}{\partial d_2}$	$\frac{\partial y}{\partial d_3}$
preCond_in_mol[3]	n_{11,H_2}	3.803E-04	2.030E-06	2.717E-08
preCond_in_mol[4]	$n_{11,CO}$	-4.907E-05	-2.309E-07	-9.022E-09
preCond_in_mol[5]	n_{11,CO_2}	1.396E-04	7.511E-07	1.359E-08
preCond_out_mol[1]	n_{12,CH_4}	-2.820E-06	-1.431E-08	-5.307E-11
preCond_out_mol[2]	n_{12,H_2O}	3.238E-03	1.596E-05	3.397E-08
preCond_out_mol[3]	n_{12,H_2}	3.803E-04	2.030E-06	2.717E-08
preCond_out_mol[4]	$n_{12,CO}$	-4.907E-05	-2.309E-07	-9.022E-09
preCond_out_mol[5]	n_{12,CO_2}	1.396E-04	7.511E-07	1.359E-08
cond_in_mol[1]	n_{12,CH_4}	-2.820E-06	-1.431E-08	-5.307E-11
cond_in_mol[2]	n_{12,H_2O}	3.238E-03	1.596E-05	3.397E-08
cond_in_mol[3]	n_{12,H_2}	3.803E-04	2.030E-06	2.717E-08
cond_in_mol[4]	$n_{12,CO}$	-4.907E-05	-2.309E-07	-9.022E-09
cond_in_mol[5]	n_{12,CO_2}	1.396E-04	7.511E-07	1.359E-08
cond_L	n_{13}	3.237E-03	1.584E-05	2.548E-08
cond_liq_frac[1]	x_{13,CH_4}	-3.657E-15	-1.865E-17	-7.878E-20
cond_liq_frac[2]	x_{13,H_2O}	1.527E-13	2.379E-10	6.634E-12
cond_liq_frac[3]	x_{13,H_2}	4.124E-14	3.025E-16	9.490E-18
cond_liq_frac[4]	$x_{13,CO}$	-6.605E-14	-3.155E-16	-1.142E-17
cond_liq_frac[5]	x_{13,CO_2}	2.847E-14	1.134E-16	6.327E-18
cond_V	n_{14}	4.687E-04	2.517E-06	3.057E-08
cond_vap_frac[1]	x_{14,CH_4}	-3.657E-09	-1.863E-11	-7.936E-14
cond_vap_frac[2]	x_{14,H_2O}	-5.100E-14	-2.416E-11	1.296E-14
cond_vap_frac[3]	x_{14,H_2}	4.124E-08	3.172E-10	9.951E-12
cond_vap_frac[4]	$x_{14,CO}$	-6.605E-08	-3.147E-10	-1.140E-11
cond_vap_frac[5]	x_{14,CO_2}	2.847E-08	1.190E-10	7.463E-12
psa_in_mol[1]	n_{14,CH_4}	-2.820E-06	-1.431E-08	-5.307E-11
psa_in_mol[2]	n_{14,H_2O}	7.411E-07	-1.586E-08	5.307E-11
psa_in_mol[3]	n_{14,H_2}	3.803E-04	2.193E-06	2.717E-08
psa_in_mol[4]	$n_{14,CO}$	-4.907E-05	-2.322E-07	-9.076E-09
psa_in_mol[5]	n_{14,CO_2}	1.396E-04	7.308E-07	1.274E-08
psa_outPurge_mol[1]	n_{15,CH_4}	-2.820E-09	-1.433E-11	-5.345E-14
psa_outPurge_mol[2]	n_{15,H_2O}	7.411E-10	-1.580E-11	5.831E-14
psa_outPurge_mol[3]	n_{15,H_2}	3.689E-04	2.111E-06	2.378E-08
psa_outPurge_mol[4]	$n_{15,CO}$	-4.907E-08	-2.317E-10	-9.122E-12
psa_outPurge_mol[5]	n_{15,CO_2}	1.396E-07	7.335E-10	1.327E-11
psa_outProduct_mol[1]	n_{16,CH_4}	-2.817E-06	-1.427E-08	-5.307E-11
psa_outProduct_mol[2]	n_{16,H_2O}	7.404E-07	-1.586E-08	5.307E-11
psa_outProduct_mol[3]	n_{16,H_2}	1.141E-05	6.598E-08	8.492E-10
psa_outProduct_mol[4]	$n_{16,CO}$	-4.902E-05	-2.322E-07	-9.076E-09
psa_outProduct_mol[5]	n_{16,CO_2}	1.394E-04	7.308E-07	1.274E-08
mix_in_T	T_2	0.000E+00	0.000E+00	0.000E+00
H2O_T	T_3	8.647E-05	1.218E-07	-3.057E-08

Continued on Next Page

Variable	Variable*	$\frac{\partial y}{\partial d_1}$	$\frac{\partial y}{\partial d_2}$	$\frac{\partial y}{\partial d_3}$
mix_out_T	T_4	8.647E-05	1.218E-07	-3.057E-08
prePR_in_T	T_4	0.000E+00	0.000E+00	0.000E+00
prePR_out_T	T_5	-9.213E-05	-5.684E-07	0.000E+00
preGHR_in_T	T_6	-1.127E-04	-9.745E-07	-5.435E-08
preGHR_out_T	T_7	-1.127E-04	-9.745E-07	-5.435E-08
ghr_in_T	T_7	2.760E-05	2.436E-07	6.793E-09
ghr_out_T	T_8	2.760E-05	2.436E-07	6.793E-09
atr_in_T	T_8	0.000E+00	0.000E+00	0.000E+00
postATR_in_T	T_9	-1.761E-04	-1.218E-06	-7.812E-08
postATR_out_T	T_{10}	-5.864E-10	-2.842E-07	0.000E+00
preCond_out_T	T_{12}	-5.864E-10	-2.842E-07	0.000E+00
cond_in_T	T_{12}	-5.864E-10	-2.842E-07	0.000E+00
cond_L_T	T_{13}	-5.864E-10	-2.842E-07	0.000E+00
cond_V_T	T_{14}	-5.864E-10	-2.842E-07	0.000E+00
psa_in_T	T_{14}	-5.864E-10	-2.842E-07	0.000E+00
psa_outPurge_T	T_{15}	-5.864E-10	-2.842E-07	0.000E+00
psa_outProduct_T	T_{16}	2.831E+01	1.397E-01	1.280E-03
prePR_Q	Q_{prePR}	-7.515E-01	-1.857E-03	-1.750E-03
preGHR_Q	Q_{preGHR}	1.054E+02	5.322E-01	3.784E-03
ghr_Q	Q_{ghr}	-1.054E+02	-5.322E-01	-3.784E-03
postATR_Q	$Q_{postATR}$	-9.104E+00	-2.794E-02	-4.619E-03
itsr_Q	Q_{itsr}	-2.234E+01	-1.324E-01	6.205E-03
preCond_Q	$Q_{preCond}$	3.466E-03	1.713E-05	5.095E-08
F_H2	n_{18}	4.132E+00	2.436E-07	1.359E-08
F_H2_heat	n_{17}	-4.132E+00	1.860E-06	1.040E-08
F_NG	n_1	6.014E-05	3.451E-07	3.397E-09
F_NG_heat	n_{19}	9.999E-01	-3.337E-07	-2.930E-09
F_fluegas	n_{20}	3.531E+00	-1.178E-06	-1.035E-08
F_inj	n_{21}	3.532E+00	-6.496E-07	-5.944E-09

A.1.3 Optimal sensitivity matrix with respect to disturbance change

Table 12: Optimal sensitivity matrix, F, which represents the optimal output change when the distubances change

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
mix_in_mol[1]	$n_{2,\mathrm{CH_4}}$	4.697E-05	4.840E-05	-4.025E-08
mix_in_mol[2]	$n_{2,\mathrm{H_2O}}$	-5.368E-36	3.154E-33	-2.780E-36
mix_in_mol[3]	$n_{2,\mathrm{H_2}}$	1.465E-34	-6.441E-26	7.604E-35
mix_in_mol[4]	$n_{2,\mathrm{CO}}$	-7.057E-38	-2.115E-25	9.141E-38
mix_in_mol[5]	$n_{2,\mathrm{CO_2}}$	8.045E-07	8.289E-07	-6.859E-10

Continued on Next Page

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
mix_in_mol[6]	n_{2,C_2H_6}	3.662E-06	3.774E-06	-3.144E-09
mix_in_mol[7]	n_{2,C_3H_8}	4.022E-06	4.145E-06	-3.363E-09
mix_in_mol[8]	$n_{2,i\text{-}C_4H_{10}}$	1.489E-06	1.534E-06	-1.263E-09
mix_in_mol[9]	$n_{2,n\text{-}C_4H_{10}}$	8.465E-07	8.720E-07	-7.105E-10
mix_in_mol[10]	$n_{2,C_5H_{12}}$	2.203E-06	2.271E-06	-1.855E-09
H2Ostream	n_3	3.468E-03	3.491E-01	-3.426E-04
mix_out_mol[1]	n_{4,CH_4}	4.697E-05	4.840E-05	-4.025E-08
mix_out_mol[2]	n_{4,H_2O}	3.468E-03	3.491E-01	-3.426E-04
mix_out_mol[3]	n_{4,H_2}	-7.323E-35	-3.221E-26	-3.787E-35
mix_out_mol[4]	$n_{4,CO}$	-1.290E-34	-9.678E-26	-6.680E-35
mix_out_mol[5]	n_{4,CO_2}	8.045E-07	8.289E-07	-6.859E-10
mix_out_mol[6]	n_{4,C_2H_6}	3.662E-06	3.774E-06	-3.144E-09
mix_out_mol[7]	n_{4,C_3H_8}	4.022E-06	4.145E-06	-3.363E-09
mix_out_mol[8]	$n_{4,i\text{-}C_4H_{10}}$	1.489E-06	1.534E-06	-1.263E-09
mix_out_mol[9]	$n_{4,n\text{-}C_4H_{10}}$	8.465E-07	8.720E-07	-7.105E-10
mix_out_mol[10]	$n_{4,C_5H_{12}}$	2.203E-06	2.271E-06	-1.855E-09
prePR_in_mol[1]	n_{4,CH_4}	4.697E-05	4.840E-05	-4.025E-08
prePR_in_mol[2]	n_{4,H_2O}	3.468E-03	3.491E-01	-3.426E-04
prePR_in_mol[3]	n_{4,H_2}	-3.429E-38	-5.305E-26	-3.009E-37
prePR_in_mol[4]	$n_{4,CO}$	-2.580E-34	2.670E-26	-1.335E-34
prePR_in_mol[5]	n_{4,CO_2}	8.045E-07	8.289E-07	-6.859E-10
prePR_in_mol[6]	n_{4,C_2H_6}	3.662E-06	3.774E-06	-3.144E-09
prePR_in_mol[7]	n_{4,C_3H_8}	4.022E-06	4.145E-06	-3.363E-09
prePR_in_mol[8]	$n_{4,i\text{-}C_4H_{10}}$	1.489E-06	1.534E-06	-1.263E-09
prePR_in_mol[9]	$n_{4,n\text{-}C_4H_{10}}$	8.465E-07	8.720E-07	-7.105E-10
prePR_in_mol[10]	$n_{4,C_5H_{12}}$	2.203E-06	2.271E-06	-1.855E-09
prePR_out_mol[1]	n_{5,CH_4}	4.697E-05	4.840E-05	-4.025E-08
prePR_out_mol[2]	n_{5,H_2O}	3.468E-03	3.491E-01	-3.426E-04
prePR_out_mol[3]	n_{5,H_2}	3.661E-35	9.472E-28	1.882E-35
prePR_out_mol[4]	$n_{5,CO}$	-2.690E-34	-2.227E-25	-1.391E-34
prePR_out_mol[5]	n_{5,CO_2}	8.045E-07	8.289E-07	-6.859E-10
prePR_out_mol[6]	n_{5,C_2H_6}	3.662E-06	3.774E-06	-3.144E-09
prePR_out_mol[7]	n_{5,C_3H_8}	4.022E-06	4.145E-06	-3.363E-09
prePR_out_mol[8]	$n_{5,i\text{-}C_4H_{10}}$	1.489E-06	1.534E-06	-1.263E-09
prePR_out_mol[9]	$n_{5,n\text{-}C_4H_{10}}$	8.465E-07	8.720E-07	-7.105E-10
prePR_out_mol[10]	$n_{5,C_5H_{12}}$	2.203E-06	2.271E-06	-1.855E-09
pr_in_mol[1]	n_{5,CH_4}	4.697E-05	4.840E-05	-4.025E-08
pr_in_mol[2]	n_{5,H_2O}	3.468E-03	3.491E-01	-3.426E-04
pr_in_mol[3]	n_{5,H_2}	-1.465E-34	6.063E-26	-7.575E-35
pr_in_mol[4]	$n_{5,CO}$	-8.071E-34	-8.352E-26	-4.174E-34
pr_in_mol[5]	n_{5,CO_2}	8.045E-07	8.289E-07	-6.859E-10
pr_in_mol[6]	n_{5,C_2H_6}	3.662E-06	3.774E-06	-3.144E-09

Continued on Next Page

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
pr_in_mol[7]	n_{5,C_3H_8}	4.022E-06	4.145E-06	-3.363E-09
pr_in_mol[8]	$n_{5,\text{i-}C_4H_{10}}$	1.489E-06	1.534E-06	-1.263E-09
pr_in_mol[9]	$n_{5,\text{n-}C_4H_{10}}$	8.465E-07	8.720E-07	-7.105E-10
pr_in_mol[10]	$n_{5,C_5H_{12}}$	2.203E-06	2.271E-06	-1.855E-09
pr_out_mol[1]	n_{6,CH_4}	3.260E-05	-4.455E-03	4.399E-06
pr_out_mol[2]	n_{6,H_2O}	3.360E-03	3.400E-01	-3.336E-04
pr_out_mol[3]	n_{6,H_2}	1.891E-04	1.817E-02	-1.789E-05
pr_out_mol[4]	$n_{6,CO}$	-1.326E-07	-2.618E-05	2.512E-08
pr_out_mol[5]	n_{6,CO_2}	5.506E-05	4.571E-03	-4.499E-06
pr_out_mol[6]	n_{6,C_2H_6}	-1.581E-13	2.706E-11	2.016E-11
pr_out_mol[7]	n_{6,C_3H_8}	-2.202E-13	-4.907E-10	1.880E-11
pr_out_mol[8]	$n_{6,\text{i-}C_4H_{10}}$	-1.207E-13	8.760E-12	3.956E-12
pr_out_mol[9]	$n_{6,\text{n-}C_4H_{10}}$	-7.325E-14	-3.300E-11	-4.220E-12
pr_out_mol[10]	$n_{6,C_5H_{12}}$	3.465E-13	1.911E-10	3.059E-12
preGHR_in_mol[1]	n_{6,CH_4}	3.260E-05	-4.455E-03	4.399E-06
preGHR_in_mol[2]	n_{6,H_2O}	3.360E-03	3.400E-01	-3.336E-04
preGHR_in_mol[3]	n_{6,H_2}	1.891E-04	1.817E-02	-1.789E-05
preGHR_in_mol[4]	$n_{6,CO}$	-1.326E-07	-2.618E-05	2.512E-08
preGHR_in_mol[5]	n_{6,CO_2}	5.506E-05	4.571E-03	-4.499E-06
preGHR_out_mol[1]	n_{7,CH_4}	3.260E-05	-4.455E-03	4.399E-06
preGHR_out_mol[2]	n_{7,H_2O}	3.360E-03	3.400E-01	-3.336E-04
preGHR_out_mol[3]	n_{7,H_2}	1.891E-04	1.817E-02	-1.789E-05
preGHR_out_mol[4]	$n_{7,CO}$	-1.326E-07	-2.618E-05	2.512E-08
preGHR_out_mol[5]	n_{7,CO_2}	5.506E-05	4.571E-03	-4.499E-06
ghr_in_mol[1]	n_{7,CH_4}	3.260E-05	-4.455E-03	4.399E-06
ghr_in_mol[2]	n_{7,H_2O}	3.360E-03	3.400E-01	-3.336E-04
ghr_in_mol[3]	n_{7,H_2}	1.891E-04	1.817E-02	-1.789E-05
ghr_in_mol[4]	$n_{7,CO}$	-1.326E-07	-2.618E-05	2.512E-08
ghr_in_mol[5]	n_{7,CO_2}	5.506E-05	4.571E-03	-4.499E-06
ghr_out_mol[1]	n_{8,CH_4}	-2.373E-04	-2.872E-02	2.825E-05
ghr_out_mol[2]	n_{8,H_2O}	2.863E-03	2.935E-01	-2.879E-04
ghr_out_mol[3]	n_{8,H_2}	1.226E-03	1.132E-01	-1.113E-04
ghr_out_mol[4]	$n_{8,CO}$	4.300E-05	2.007E-03	-2.010E-06
ghr_out_mol[5]	n_{8,CO_2}	2.818E-04	2.680E-02	-2.632E-05
atr_in_mol[1]	n_{8,CH_4}	-2.373E-04	-2.872E-02	2.825E-05
atr_in_mol[2]	n_{8,H_2O}	2.863E-03	2.935E-01	-2.879E-04
atr_in_mol[3]	n_{8,H_2}	1.226E-03	1.132E-01	-1.113E-04
atr_in_mol[4]	$n_{8,CO}$	4.300E-05	2.007E-03	-2.010E-06
atr_in_mol[5]	n_{8,CO_2}	2.818E-04	2.680E-02	-2.632E-05
atr_out_mol[1]	n_{9,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
atr_out_mol[2]	n_{9,H_2O}	3.124E-03	3.178E-01	-3.120E-04
atr_out_mol[3]	n_{9,H_2}	4.964E-04	3.200E-02	-3.135E-05

Continued on Next Page

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
atr_out_mol[4]	$n_{9,CO}$	-1.655E-04	-2.453E-02	2.405E-05
atr_out_mol[5]	n_{9,CO_2}	2.559E-04	2.493E-02	-2.443E-05
postATR_in_mol[1]	n_{9,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
postATR_in_mol[2]	n_{9,H_2O}	3.124E-03	3.178E-01	-3.120E-04
postATR_in_mol[3]	n_{9,H_2}	4.964E-04	3.200E-02	-3.135E-05
postATR_in_mol[4]	$n_{9,CO}$	-1.655E-04	-2.453E-02	2.405E-05
postATR_in_mol[5]	n_{9,CO_2}	2.559E-04	2.493E-02	-2.443E-05
postATR_out_mol[1]	n_{10,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
postATR_out_mol[2]	n_{10,H_2O}	3.124E-03	3.178E-01	-3.120E-04
postATR_out_mol[3]	n_{10,H_2}	4.964E-04	3.200E-02	-3.135E-05
postATR_out_mol[4]	$n_{10,CO}$	-1.655E-04	-2.453E-02	2.405E-05
postATR_out_mol[5]	n_{10,CO_2}	2.559E-04	2.493E-02	-2.443E-05
itsr_in_mol[1]	n_{10,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
itsr_in_mol[2]	n_{10,H_2O}	3.124E-03	3.178E-01	-3.120E-04
itsr_in_mol[3]	n_{10,H_2}	4.964E-04	3.200E-02	-3.135E-05
itsr_in_mol[4]	$n_{10,CO}$	-1.655E-04	-2.453E-02	2.405E-05
itsr_in_mol[5]	n_{10,CO_2}	2.559E-04	2.493E-02	-2.443E-05
itsr_out_mol[1]	n_{11,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
itsr_out_mol[2]	n_{11,H_2O}	3.240E-03	3.365E-01	-3.303E-04
itsr_out_mol[3]	n_{11,H_2}	3.799E-04	1.330E-02	-1.301E-05
itsr_out_mol[4]	$n_{11,CO}$	-4.913E-05	-5.832E-03	5.717E-06
itsr_out_mol[5]	n_{11,CO_2}	1.395E-04	6.233E-03	-6.097E-06
preCond_in_mol[1]	n_{11,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
preCond_in_mol[2]	n_{11,H_2O}	3.240E-03	3.365E-01	-3.303E-04
preCond_in_mol[3]	n_{11,H_2}	3.799E-04	1.330E-02	-1.301E-05
preCond_in_mol[4]	$n_{11,CO}$	-4.913E-05	-5.832E-03	5.717E-06
preCond_in_mol[5]	n_{11,CO_2}	1.395E-04	6.233E-03	-6.097E-06
preCond_out_mol[1]	n_{12,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
preCond_out_mol[2]	n_{12,H_2O}	3.240E-03	3.365E-01	-3.303E-04
preCond_out_mol[3]	n_{12,H_2}	3.799E-04	1.330E-02	-1.301E-05
preCond_out_mol[4]	$n_{12,CO}$	-4.913E-05	-5.832E-03	5.717E-06
preCond_out_mol[5]	n_{12,CO_2}	1.395E-04	6.233E-03	-6.097E-06
cond_in_mol[1]	n_{12,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
cond_in_mol[2]	n_{12,H_2O}	3.240E-03	3.365E-01	-3.303E-04
cond_in_mol[3]	n_{12,H_2}	3.799E-04	1.330E-02	-1.301E-05
cond_in_mol[4]	$n_{12,CO}$	-4.913E-05	-5.832E-03	5.717E-06
cond_in_mol[5]	n_{12,CO_2}	1.395E-04	6.233E-03	-6.097E-06
cond_L	n_{13}	3.239E-03	3.365E-01	-3.303E-04
cond_liq_frac[1]	x_{13,CH_4}	-3.695E-15	-3.851E-13	3.800E-16
cond_liq_frac[2]	x_{13,H_2O}	1.527E-13	2.379E-10	6.634E-12
cond_liq_frac[3]	x_{13,H_2}	4.127E-14	4.137E-12	-4.054E-15
cond_liq_frac[4]	$x_{13,CO}$	-6.611E-14	-7.302E-12	7.156E-15

Continued on Next Page

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
cond_liq_frac[5]	x_{13,CO_2}	2.853E-14	3.550E-12	-3.478E-15
cond_V	n_{14}	4.682E-04	1.341E-02	-1.311E-05
cond_vap_frac[1]	x_{14,CH_4}	-3.695E-09	-3.851E-07	3.800E-10
cond_vap_frac[2]	x_{14,H_2O}	-5.100E-14	-2.416E-11	1.296E-14
cond_vap_frac[3]	x_{14,H_2}	4.127E-08	4.137E-06	-4.053E-09
cond_vap_frac[4]	$x_{14,CO}$	-6.611E-08	-7.302E-06	7.156E-09
cond_vap_frac[5]	x_{14,CO_2}	2.853E-08	3.550E-06	-3.477E-09
psa_in_mol[1]	n_{14,CH_4}	-2.851E-06	-3.102E-04	3.061E-07
psa_in_mol[2]	n_{14,H_2O}	7.404E-07	2.119E-05	-2.072E-08
psa_in_mol[3]	n_{14,H_2}	3.799E-04	1.330E-02	-1.301E-05
psa_in_mol[4]	$n_{14,CO}$	-4.913E-05	-5.832E-03	5.716E-06
psa_in_mol[5]	n_{14,CO_2}	1.395E-04	6.233E-03	-6.098E-06
psa_outPurge_mol[1]	n_{15,CH_4}	-2.851E-09	-3.102E-07	3.061E-10
psa_outPurge_mol[2]	n_{15,H_2O}	7.404E-10	2.119E-08	-2.072E-11
psa_outPurge_mol[3]	n_{15,H_2}	3.685E-04	1.290E-02	-1.262E-05
psa_outPurge_mol[4]	$n_{15,CO}$	-4.913E-08	-5.832E-06	5.716E-09
psa_outPurge_mol[5]	n_{15,CO_2}	1.395E-07	6.233E-06	-6.097E-09
psa_outProduct_mol[1]	n_{16,CH_4}	-2.849E-06	-3.099E-04	3.058E-07
psa_outProduct_mol[2]	n_{16,H_2O}	7.396E-07	2.117E-05	-2.070E-08
psa_outProduct_mol[3]	n_{16,H_2}	1.140E-05	3.991E-04	-3.903E-07
psa_outProduct_mol[4]	$n_{16,CO}$	-4.908E-05	-5.827E-03	5.711E-06
psa_outProduct_mol[5]	n_{16,CO_2}	1.394E-04	6.226E-03	-6.092E-06
mix_in_T	T_2	0.000E+00	0.000E+00	0.000E+00
H2O_T	T_3	8.655E-05	9.515E-03	-9.371E-06
mix_out_T	T_4	0.000E+00	0.000E+00	0.000E+00
prePR_in_T	T_4	8.655E-05	9.515E-03	-9.371E-06
prePR_out_T	T_5	2.665E-07	-4.904E-04	2.587E-07
preGHR_in_T	T_6	-9.204E-05	-1.047E-02	1.013E-05
preGHR_out_T	T_7	-1.126E-04	-1.047E-02	1.008E-05
ghr_in_T	T_7	-1.126E-04	-1.047E-02	1.008E-05
ghr_out_T	T_8	2.955E-05	-2.890E-04	1.648E-07
atr_in_T	T_8	2.955E-05	-2.890E-04	1.648E-07
postATR_in_T	T_9	5.573E-06	-3.274E-03	2.882E-06
postATR_out_T	T_{10}	-1.760E-04	-1.048E-02	1.005E-05
preCond_out_T	T_{12}	-5.864E-10	-2.842E-07	-1.268E-17
cond_in_T	T_{12}	-5.864E-10	-2.842E-07	-1.268E-17
cond_L_T	T_{13}	-5.864E-10	-2.842E-07	-1.268E-17
cond_V_T	T_{14}	-5.864E-10	-2.842E-07	-1.268E-17
psa_in_T	T_{14}	-5.864E-10	-2.842E-07	-1.268E-17
psa_outPurge_T	T_{15}	-5.864E-10	-2.842E-07	-1.268E-17
psa_outProduct_T	T_{16}	-5.864E-10	-2.842E-07	-1.268E-17
prePR_Q	Q_{prePR}	2.833E+01	2.712E+03	-2.669E+00

Continued on Next Page

Variable	Variable*	$\frac{\partial y^{opt}}{\partial d_1}$	$\frac{\partial y^{opt}}{\partial d_2}$	$\frac{\partial y^{opt}}{\partial d_3}$
preGHR_Q	Q_{preGHR}	-7.515E-01	-3.953E-03	-1.748E-03
ghr_Q	Q_{ghr}	1.057E+02	9.667E+03	-9.494E+00
postATR_Q	$Q_{postATR}$	-1.057E+02	-9.667E+03	9.494E+00
itsr_Q	Q_{itsr}	-9.133E+00	-8.522E+02	8.395E-01
preCond_Q	$Q_{preCond}$	-2.236E+01	-2.115E+03	2.082E+00
F_H2	n_{18}	4.132E+00	3.153E-03	-3.061E-06
F_H2_heat	n_{17}	-4.132E+00	9.751E-03	-9.562E-06
F_NG	n_1	6.004E-05	6.187E-05	-5.076E-08
F_NG_heat	n_{19}	9.999E-01	-6.186E-05	5.123E-08
F_fluegas	n_{20}	3.531E+00	-2.185E-04	1.809E-07
F_inj	n_{21}	3.532E+00	2.918E-04	-3.054E-07

B Julia Code

This section contains all code that has been developed and used in this project for multiple purposes, such as building the model, testing data, simulations and more. Under each subsection, the function of the code is explained. All the files containing julia code used in this thesis are available in the author's GitHub repository[64].

B.1 0Model.jl

The file 0Model.jl builds all the submodels to a larger model and connects the submodels with connectivity constraints. Parameters from 0par.jl file is imported and used for building the model. After building the model, the objective function is formed and optimized. At the end, the dataframe.jl file is used for printing out the variables of the optimization results.

```
using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")
```

```julia
const MOI = MathOptInterface
C = 12.01;
H = 1.008;
O = 16;
Base.@kwdef mutable struct _par
    init::init_par = init_par();
    mix::mix_par=mix_par();
    prePR::prePR_par=prePR_par();
    pr::pr_par = pr_par();
    preGHR::preGHR_par = preGHR_par();
    ghr::ghr_par = ghr_par();
    atr::atr_par = atr_par();
    postATR::postATR_par = postATR_par();
    itsr::itsr_par = itsr_par();
    preCond::preCond_par = preCond_par();
    cond::cond_par = cond_par();
    psa::psa_par = psa_par();
    hconst = heavy_const;
    smr_const = smr_const;
    wgsr_const = wgsr_const;
    HHV_H2::Float64 = 141.7*1000;
    HHV_NG::Vector = [55.5,0.0,141.7,0.0,0.0,51.9,50.4,49.1,49.1,48.6]*1000; #
        CH4, H2O, H2, CO, CO2, C2H6, C3H8, n-C4H10, i-C4H10, C5H12
    molarMass::Vector = [C+H*4, H*2+O, H*2, C+O, C+O*2, C*2+H*6, C*3+H*8, C*4+H*
        10, C*4+H*10, C*5+H*12];
    P_H2::Float64 = 3.347; #[\$/kmol]
    P_inj::Float64 = 9.650; # [\$/ton CO2]
    R::Float64 = 8.314;
    elCost::Float64 = 0.14; # [\$/Kwh]
end

################# Initializing the model ################
par = _par();
optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-6, "constr_viol_tol" => 1e-8, "print_level" => 0)
m = Model(optimizer);

###### Assembling the submodels to a larger model #######
MIX_model(m, par);
prePR_model(m, par);
PR_model(m, par);
preGHR_model(m, par);
GHR_model(m, par);
ATR_model(m, par);
postATR_model(m, par);
ITSR_model(m, par);
preCond_model(m, par);
Cond_model(m, par);
PSA_model(m, par);
#all_variables(m)

#### Introducing new variables for the economic objective function #######
@variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the obj
```

```julia
     function
@variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that is
    being used to heat up the process
@variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that is
    being used to heat up the process
@variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is being
     used in the process
@variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
    combusting natural gas which needs to be injected
@variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that is
    being injected

##################### Connection constraints #######################
for i = 1:10
    @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
    @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
end

for i = 1:5 # After all heavier carbons are removed
    @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
    @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
    @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
    @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
    @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
    @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
    @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
    @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] == 0)
end
################# Same for the temperature ################################
@NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
@NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
@NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
@NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
@NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
@NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
@NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
@NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
@NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
@NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


############## Initial values ##########################################
for i = 1:10
    @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0);
end
@NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
@NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

############# To ensure the GHR and ATR heat exchange ##################
@NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

##To ensure that the inlet hot stream is hotter than outlet cold stream##
@NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
@NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);
```

```
######## New constraints for the economic objective function #############
@NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
@NLconstraint(m, m[:F_NG] - par.init.init_stream + m[:F_NG_heat] == 0);
@NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
@NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][i])
    for i = 1:5) == 0);
@NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016 -
    sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[i] for
    i = 1:10) == 0);


compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
T2 = compT(m, m[:psa_outPurge_T],1,10);
compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
compWsum = @NLexpression(m, compW1 + compW2);
@NLobjective(m, Max, m[:F_H2]*par.P_H2 - compWsum*par.elCost/1000); # 44*m[:
    F_inj]/1000*par.P_inj
#@NLobjective(m, Max, m[:psa_outProduct_mol][3])
optimize!(m)
#@show m

##################### Printing output ################################
streamdf, otherdf, massdf, compositiondf = printTable(m);
println("Stream table"); show(streamdf, allrows=true);
println("\n\nOther variables"); show(otherdf, allrows=true);
println("\n\nMass table"); show(massdf, allrows=true);
println("\n\nCompostion table"); show(compositiondf, allrows=true);
println("");
printActive(m);
```

B.2 0par.jl

The file 0par.jl contains all the parameters from each submodel that is being built into a larger parameter set used in 0Model.jl. Most parameters are stream, temperature and composition data.

```
using JuMP, Ipopt, MathOptInterface
# Parameters for initial flow values

Base.@kwdef mutable struct init_par
    init_stream::Float64 = 145.4;
    init_comp::Vector = [0.7824, 0.0, 0.0, 0.0, 0.0134, 0.0610, 0.0670, 0.0248,
        0.0141, 0.0367]
end

Base.@kwdef mutable struct mix_par
    in_T::Float64 = 311.00;
    out_T::Float64 = 381.41;
    H2O_T::Float64 = 423.00;
    carbon_ratio::Float64 = 2.5;
    ini_mix_in::Vector = [113.76022309722791, 0.0, 0.0, 0.0, 1.9483473792214394,
         8.869342547202073,
     9.741736896107197, 3.605896642141171, 2.0501267199270368, 5.379765151581587
         ];
end

Base.@kwdef mutable struct prePR_par
    out_T::Float64 = 693.00;
end

Base.@kwdef mutable struct pr_par
    out_T::Float64 = 630.52;
end

Base.@kwdef mutable struct preGHR_par
    out_T::Float64 = 753.0;
end

Base.@kwdef mutable struct ghr_par
    out_T::Float64 = 973.0;
end

Base.@kwdef mutable struct atr_par
    out_T::Float64 = 1323.0;
    nO2_H::Float64 = 22.7;
end

Base.@kwdef mutable struct postATR_par
    out_T::Float64 = 512.79;
end

Base.@kwdef mutable struct itsr_par
    out_T::Float64 = 523.0;
end

Base.@kwdef mutable struct preCond_par
    out_T::Float64 = 313.0;
end

Base.@kwdef mutable struct cond_par
    L_T::Float64 = 313.0;
    V_T::Float64 = 313.0;
    ant_par::Vector = [6.20963, 2354.731, 7.559];
    cond_P::Float64 = 15.0;
end

Base.@kwdef mutable struct psa_par
    outPurge_T::Float64 = 313.0;
    outProduct_T::Float64 = 313.0;
    splitratio::Vector = [0.001, 0.001, 0.97, 0.001, 0.001]; # Almost ideal
end
```

B.3 1MIX.jl

1Mix.jl contains the function MIX_model() which builds the submodel for the first submodel, which calculates the amount of steam required in the process depending on the natural gas feed and the steam-to-carbon ratio. After mixing the two water streams, the mixed temperature is calculated by the enthalpy balance, where the function build_enthalpy() imported from the enthalpy.jl file is used.

```julia
function MIX_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= mix_in_mol[1:10]); # Stream 1
  @variable(model, 0 <= H2Ostream, start = 358.5177276354675); # Stream 2
  @variable(model, 0 <= mix_out_mol[1:10]); # Stream 3
   # CH4 H2O H2 CO CO2 C2 C3 i-C4 n-C4 C5
  ini_mix_in = [113.76022309722791, 0.0, 0.0, 0.0, 1.9483473792214394, 8.8
      69342547202073, 9.741736896107197, 3.605896642141171, 2.0501267199270368,
       5.379765151581587];
  ini_mix_out = [113.76022309722791, 358.5177276354675, 0.0, 0.0, 1.9
      483473792214394, 8.869342547202073, 9.741736896107197, 3.605896642141171,
       2.0501267199270368, 5.379765151581587];

  for i=1:10
      set_start_value(mix_in_mol[i] , ini_mix_in[i]);
      set_start_value(mix_out_mol[i] , ini_mix_out[i]);
  end

  @variable(model, 273 <= mix_in_T, start = 311);
  @variable(model, 273 <= mix_out_T, start = 381.41);
  @variable(model, 273 <= H2O_T, start = 423.00);
  @variable(model, 0 <= SC_ratio <= 5.0, start = 2.5);

  # Expressions
  mix_H_out = build_enthalpy(model, mix_out_T, par)
  mix_H_in = build_enthalpy(model, mix_in_T, par)
  mix_H_inH2O = build_enthalpy(model, H2O_T, par)
  mix_n_Carbon = @NLexpression(model, sum(mix_out_mol[i] for i=6:10)+mix_out_mol
      [1])

  # Mass balance
  @NLconstraint(model, H2Ostream - SC_ratio*mix_n_Carbon == 0);
  @NLconstraint(model, mix_out_mol[1] - mix_in_mol[1] == 0)
  @NLconstraint(model, mix_out_mol[2] - mix_in_mol[2] - H2Ostream == 0)
  @NLconstraint(model, mix_out_mol[3] - mix_in_mol[3] == 0)
  @NLconstraint(model, mix_out_mol[4] - mix_in_mol[4] == 0)
  @NLconstraint(model, mix_out_mol[5] - mix_in_mol[5] == 0)
  @NLconstraint(model, mix_out_mol[6] - mix_in_mol[6] == 0)
  @NLconstraint(model, mix_out_mol[7] - mix_in_mol[7] == 0)
  @NLconstraint(model, mix_out_mol[8] - mix_in_mol[8] == 0)
  @NLconstraint(model, mix_out_mol[9] - mix_in_mol[9] == 0)
  @NLconstraint(model, mix_out_mol[10] - mix_in_mol[10] == 0)
  # Energy balance
  @NLconstraint(model, sum(mix_H_out[i]*mix_out_mol[i] - mix_H_in[i]*mix_in_mol[
      i] for i=1:10) - mix_H_inH2O[2]*H2Ostream == 0);
end
```

B.4 2prePR.jl

2prePR.jl file contains the prePR_model function that builds the submodel for the heater before the pre-reformer unit. The energy balance calculates the heat required in this process to reach its desired temperature.

```
function prePR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= prePR_in_mol[1:10]);
  @variable(model, 0 <= prePR_out_mol[1:10]);

  # CH4 H2O H2 CO CO2 C2 C3 i-C4 n-C4 C5
  ini_prePR_in = [113.76022309722791, 358.5177276354675, 0.0, 0.0, 1.9
      483473792214394, 8.869342547202073, 9.741736896107197, 3.605896642141171,
       2.0501267199270368, 5.379765151581587];   # Stream 3
  ini_prePR_out = [113.76022309722791, 358.5177276354675, 0.0, 0.0, 1.9
      483473792214394, 8.869342547202073, 9.741736896107197, 3.605896642141171,
       2.0501267199270368, 5.379765151581587]; # Stream 4

  for i=1:10
      set_start_value(prePR_in_mol[i] , ini_prePR_in[i])
      set_start_value(prePR_out_mol[i] , ini_prePR_out[i])
  end

  @variable(model, 273 <= prePR_in_T, start = 381.41);
  @variable(model, 273 <= prePR_out_T, start = 693.00);
  @variable(model, 0 <= prePR_Q, start = 6890.45);

  # Expressions
  prePR_H_out = build_enthalpy(model, prePR_out_T, par)
  prePR_H_in = build_enthalpy(model, prePR_in_T, par)

  # Constraints
  # Mass balance
  @NLconstraint(model, prePR_out_mol[1] - prePR_in_mol[1] == 0)
  @NLconstraint(model, prePR_out_mol[2] - prePR_in_mol[2] == 0)
  @NLconstraint(model, prePR_out_mol[3] - prePR_in_mol[3] == 0)
  @NLconstraint(model, prePR_out_mol[4] - prePR_in_mol[4] == 0)
  @NLconstraint(model, prePR_out_mol[5] - prePR_in_mol[5] == 0)
  @NLconstraint(model, prePR_out_mol[6] - prePR_in_mol[6] == 0)
  @NLconstraint(model, prePR_out_mol[7] - prePR_in_mol[7] == 0)
  @NLconstraint(model, prePR_out_mol[8] - prePR_in_mol[8] == 0)
  @NLconstraint(model, prePR_out_mol[9] - prePR_in_mol[9] == 0)
  @NLconstraint(model, prePR_out_mol[10] - prePR_in_mol[10] == 0)
  # Energy balance
  @NLconstraint(model, sum(prePR_H_out[i]*prePR_out_mol[i] - prePR_H_in[i]*
      prePR_in_mol[i] for i=1:10) - prePR_Q==0);

  # Energy balance  - equipment specification
  #@NLconstraint(model, prePR_out_T - par.prePR.out_T == 0);

  return model;
end
```

B.5 3PR.jl

3PR.jl file contains the function PR_model() and builds the submodel for the pre-reformer. In this function, mole balances are balanced with the extent of reactions to calculate the amount of methane that is reformed from heavier hydrocarbons and the equilibrium. As this is an adiabatic reaction, the outlet temperature is also calculated from the energy balance.

```
function PR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= pr_in_mol[1:10]); # Stream 4
  @variable(model, 0 <= pr_out_mol[1:10]); # Stream 5

  ini_pr_in = [113.76022309722791, 358.5177276354675, 0.0, 0.0, 1.9
      483473792214394, 8.869342547202073, 9.741736896107197, 3.605896642141171,
       2.0501267199270368, 5.379765151581587];  # CH4 H2O H2 CO CO2 C2 C3 i-C4
      n-C4 C5
  ini_pr_out = [178.58124131020057, 295.5198912988238, 59.489482856563995, 0.3
      3375721522392365, 33.28038693993136, 0, 0, 0, 0, 0];

  for i=1:10
    set_start_value(pr_in_mol[i] , ini_pr_in[i]);
    set_start_value(pr_out_mol[i] , ini_pr_out[i]);
  end

  @variable(model, 643 <= pr_in_T <= 743, start = 693.00);
  @variable(model, 609.2 <= pr_out_T <= 709.2, start = 659.2);

  # Expressions
  pr_ksi1 = @NLexpression(model, pr_in_mol[6]); # C2H6
  pr_ksi2 = @NLexpression(model, pr_in_mol[7]); # C3H8
  pr_ksi3 = @NLexpression(model, pr_in_mol[8]); # n-C4H10
  pr_ksi4 = @NLexpression(model, pr_in_mol[9]); # i-C4H10
  pr_ksi5 = @NLexpression(model, pr_in_mol[10]); # C5+
  pr_ksi6 = @NLexpression(model, pr_in_mol[1] - pr_out_mol[1]);
  pr_ksi7 = @NLexpression(model, pr_out_mol[5] - pr_in_mol[5]);

  #pr_Ksmr_model = @NLexpression(model, exp(-22790 / pr_out_T + 8.156 * log(
      pr_out_T) - 4.421 / 10^3 * pr_out_T
  #- 4.330 * 10^3 / (pr_out_T^2) - 26.030));

  #pr_Kwgsr_model = @NLexpression(model, exp(5693.5/pr_out_T + 1.077*log(
      pr_out_T) + 5.44e-4*pr_out_T - 1.125e-7*pr_out_T^2 - 49170/(pr_out_T^2)
      -13.148));
  pr_K_smr_model = K_smr(model, pr_out_T, par);
  pr_K_wgsr_model = K_wgsr(model, pr_out_T, par);

  pr_ntot = @NLexpression(model, sum(pr_out_mol[i] for i=1:10));

  pr_H_out = build_enthalpy(model, pr_out_T, par);
  pr_H_in = build_enthalpy(model, pr_in_T, par);

  # Constraints
  # Mass balance
  @NLconstraint(model, pr_out_mol[2] - pr_in_mol[2] + 2 * pr_ksi1 + 3 * pr_ksi2
      + 4 * pr_ksi3 + 4 * pr_ksi4 + 5 * pr_ksi5 + pr_ksi6 + pr_ksi7 == 0);
  @NLconstraint(model, pr_out_mol[3] - pr_in_mol[3] - 5 * pr_ksi1 - 7 * pr_ksi2
      - 9 * pr_ksi3 - 9 * pr_ksi4 - 11 * pr_ksi5 - 3 * pr_ksi6 - pr_ksi7 == 0);
  @NLconstraint(model, pr_out_mol[4] - pr_in_mol[4] - 2 * pr_ksi1 - 3 * pr_ksi2
      - 4 * pr_ksi3 - 4 * pr_ksi4 - 5 * pr_ksi5 - pr_ksi6 + pr_ksi7 == 0);
  @NLconstraint(model, pr_K_smr_model*((pr_out_mol[1]/pr_ntot) * (pr_out_mol[2]/
      pr_ntot )) - (((pr_out_mol[4]/pr_ntot) * (pr_out_mol[3]/pr_ntot)^3)) == 0
      );
  @NLconstraint(model, pr_K_wgsr_model*((pr_out_mol[4]/pr_ntot) * (pr_out_mol[2]
      /pr_ntot)) - (((pr_out_mol[5]/pr_ntot) * (pr_out_mol[3]/pr_ntot))) == 0);
  @NLconstraint(model, pr_out_mol[6] - pr_in_mol[6] + pr_ksi1 == 0)
  @NLconstraint(model, pr_out_mol[7] - pr_in_mol[7] + pr_ksi2 == 0)
  @NLconstraint(model, pr_out_mol[8] - pr_in_mol[8] + pr_ksi3 == 0)
  @NLconstraint(model, pr_out_mol[9] - pr_in_mol[9] + pr_ksi4 == 0)
  @NLconstraint(model, pr_out_mol[10] - pr_in_mol[10] + pr_ksi5 == 0)

  # Add NLconstraint for the last 5 carbons as 0

  # Energy balance
  @NLconstraint(model, sum(pr_H_out[i]*pr_out_mol[i] - pr_H_in[i]*pr_in_mol[i]
      for i=1:10) ==0);
  return model;
end
```

B.6 4preGHR.jl

4preGHR.jl file contains the function preGHR_model() which builds the submodel for the heater before the GHR unit and calculates the required amount of heat required for reaching the inlet GHR temperature.

```
function preGHR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= preGHR_in_mol[1:5]); # Stream 5
  @variable(model, 0 <= preGHR_out_mol[1:5]); # Stream 6

  ini_preGHR_in = [178.58124131020057, 295.5198912988238, 59.489482856563995, 0
      .33375721522392365, 33.28038693993136];  # CH4 H2O H2 CO CO2
  ini_preGHR_out = [178.58124131020057, 295.5198912988238, 59.489482856563995, 0
      .33375721522392365, 33.28038693993136];

  for i=1:5
    set_start_value(preGHR_in_mol[i] , ini_preGHR_in[i]);
    set_start_value(preGHR_out_mol[i] , ini_preGHR_out[i]);
  end

  @variable(model, 273 <= preGHR_in_T, start = 630.5201);
  @variable(model, 273 <= preGHR_out_T, start = 753.00);

  preGHR_H_out = build_enthalpy(model, preGHR_out_T, par)
  preGHR_H_in = build_enthalpy(model, preGHR_in_T, par)
  @variable(model, 0 <= preGHR_Q, start = 8918.36225526873);

  # Constraints
  # Mass balance
  @NLconstraint(model, preGHR_out_mol[1] - preGHR_in_mol[1] == 0)
  @NLconstraint(model, preGHR_out_mol[2] - preGHR_in_mol[2] == 0)
  @NLconstraint(model, preGHR_out_mol[3] - preGHR_in_mol[3] == 0)
  @NLconstraint(model, preGHR_out_mol[4] - preGHR_in_mol[4] == 0)
  @NLconstraint(model, preGHR_out_mol[5] - preGHR_in_mol[5] == 0)

  # Energy balance
  @NLconstraint(model, sum(preGHR_H_out[i]*preGHR_out_mol[i] - preGHR_H_in[i]*
      preGHR_in_mol[i] for i=1:5) - preGHR_Q==0);

  # Energy balance - equipment specification
  #@NLconstraint(model, preGHR_out_T - par.preGHR.out_T == 0);

  return model;
end
```

B.7 5GHR.jl

5GHR.jl file contains the function GHR_model() which builds the submodel for the GHR unit. In the GHR unit, mole and energy balances are formed to calculate the exiting temperature from the energy balance and mole streams from the equilibrium constant. The equilibrium constant is calculated from functions in the equilibrium.jl file.

```
function GHR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= ghr_in_mol[1:5]); # Stream 6
  @variable(model, 0 <= ghr_out_mol[1:5]); # Stream 7
  # CH4 H2O H2 CO CO2
  ini_ghr_in = [178.58124131020057, 295.5198912988238, 59.489482856563995, 0.3
      3375721522392365, 33.28038693993136];
  ini_ghr_out = [100.543406840790453, 122.15875765109533, 544.9262854431126, 139
      .04829250631568, 50.60368611824971];

  for i=1:5
      set_start_value(ghr_in_mol[i] , ini_ghr_in[i]);
      set_start_value(ghr_out_mol[i] , ini_ghr_out[i]);
  end

  @variable(model, 559.2 <= ghr_in_T <= 659.2, start = 609.2);
  @variable(model, 914.8 <= ghr_out_T <= 1014.8, start = 964.8);

  @variable(model, 0 <= ghr_Q, start = 34472000);

  ghr_K_smr_model = K_smr(model, ghr_out_T, par);
  ghr_K_wgsr_model = K_wgsr(model, ghr_out_T, par);

  ghr_ksi_smr = @NLexpression(model, ghr_in_mol[1] - ghr_out_mol[1]);

  ghr_ksi_wgsr = @NLexpression(model, ghr_out_mol[5] - ghr_in_mol[5]);

  ghr_ntot = @NLexpression(model, sum(ghr_out_mol[i] for i=1:5));

  ghr_H_out = build_enthalpy(model, ghr_out_T, par);
  ghr_H_in = build_enthalpy(model, ghr_in_T, par);

  # Constraints
  # Mass balance
  @NLconstraint(model, ghr_K_smr_model*((ghr_out_mol[1]/ghr_ntot) * (ghr_out_mol
      [2]/ghr_ntot )) -
   (((ghr_out_mol[4]/ghr_ntot) * (ghr_out_mol[3]/ghr_ntot)^3)) == 0);
  @NLconstraint(model, ghr_K_wgsr_model*((ghr_out_mol[4]/ghr_ntot) * (
      ghr_out_mol[2]/ghr_ntot)) -
   (((ghr_out_mol[5]/ghr_ntot) * (ghr_out_mol[3]/ghr_ntot))) == 0);
  @NLconstraint(model, ghr_out_mol[2] - ghr_in_mol[2] + ghr_ksi_smr +
      ghr_ksi_wgsr == 0);
  @NLconstraint(model, ghr_out_mol[3] - ghr_in_mol[3] - 3 * ghr_ksi_smr -
      ghr_ksi_wgsr == 0);
  @NLconstraint(model, ghr_out_mol[4] - ghr_in_mol[4] - ghr_ksi_smr +
      ghr_ksi_wgsr == 0);

  # Energy balance
  @NLconstraint(model, sum(ghr_H_out[i]*ghr_out_mol[i] - ghr_H_in[i]*ghr_in_mol[
      i] for i=1:5) - ghr_Q==0);

  # Energy balance - equipment specification
  #@NLconstraint(model, ghr_out_T - par.ghr.out_T == 0);
  return model;
end
```

B.8 6ATR.jl

6ATR.jl file contains the function ATR_model() which builds the submodel for the ATR unit. In the ATR unit, mole and energy balances are formed to calculate the exiting temperature from the energy balance and mole streams from the equilibrium constant. The equilibrium constant is calculated from functions in the equilibrium.jl file.

```
function ATR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= atr_in_mol[1:5]); # Stream 7
  @variable(model, 0 <= atr_out_mol[1:5]); # Stream 8

  ini_atr_in = [22.543406840790453, 122.15875765109533, 544.9262854431126, 139.0
      4829250631568, 50.60368611824971];  # CH4 H2O H2 CO CO2
  ini_atr_out = [0.00032515179570229987, 217.23336631977338, 494.9378401524241,
      184.6822048577565, 27.51285545580364];

  for i=1:5
      set_start_value(atr_in_mol[i] , ini_atr_in[i])
      set_start_value(atr_out_mol[i] , ini_atr_out[i])
  end

  @variable(model, 912.18 <= atr_in_T <= 1012.18, start = 962.18);
  @variable(model, 1273 <= atr_out_T <= 1373, start = 1323);
  @variable(model, 0 <= nO2 <= 100, start = 47.26342984761337);

  # Expressions
  atr_K_smr_model = K_smr(model, atr_out_T, par);
  atr_K_wgsr_model = K_wgsr(model, atr_out_T, par);

  atr_ksi_smr = @NLexpression(model, atr_in_mol[1] - atr_out_mol[1]);
  atr_ksi_wgsr = @NLexpression(model, atr_out_mol[5] - atr_in_mol[5]);

  atr_ntot = @NLexpression(model, sum(atr_out_mol[i] for i=1:5));

  atr_H_out = build_enthalpy(model, atr_out_T, par);
  atr_H_in = build_enthalpy(model, atr_in_T, par);


  # Constraints
  # Mass balance
  @NLconstraint(model, atr_K_smr_model*((atr_out_mol[1]/atr_ntot) * (atr_out_mol
      [2]/atr_ntot )) - (((atr_out_mol[4]/atr_ntot) * (atr_out_mol[3]/atr_ntot)
      ^3)) == 0);
  @NLconstraint(model, atr_K_wgsr_model*((atr_out_mol[4]/atr_ntot) * (
      atr_out_mol[2]/atr_ntot)) - (((atr_out_mol[5]/atr_ntot) * (atr_out_mol[3]
      /atr_ntot))) == 0);
  @NLconstraint(model, atr_out_mol[2] - atr_in_mol[2] + atr_ksi_smr +
      atr_ksi_wgsr - 2*nO2 == 0);
  @NLconstraint(model, atr_out_mol[3] - atr_in_mol[3] - 3 * atr_ksi_smr -
      atr_ksi_wgsr + 2*nO2 == 0);
  @NLconstraint(model, atr_out_mol[4] - atr_in_mol[4] - atr_ksi_smr +
      atr_ksi_wgsr == 0);

  # Energy balance
  @NLconstraint(model, sum(atr_H_out[i]*atr_out_mol[i] - atr_H_in[i]*atr_in_mol[
      i] for i=1:5) - nO2*par.atr.nO2_H == 0);

  # Energy balance - equipment specification
  #@NLconstraint(model, atr_out_T - par.atr.out_T == 0);
  return model;
end
```

B.9 7PostATR.jl

7PostATR.jl file contains the function postATR_model() which builds the submodel for the cooler after the ATR unit. The amount of heat recovered in this cooler is set to be the same as the amount of heat that the GHR unit requires with a constraint in the 0Model.jl file where all submodels are assembled. The outlet temperature will depend on the mole composition, the amount of heat required in the GHR, and the outlet temperature of the ATR.

```
function postATR_model(model, par)
   # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= postATR_in_mol[1:5]); # Stream 8
  @variable(model, 0 <= postATR_out_mol[1:5]); # Stream 9

  ini_postATR_in = [0.00032515179570229987, 217.23336631977338, 494.9
      378401524241, 184.6822048577565, 27.51285545580364];  # CH4 H2O H2 CO CO2
  ini_postATR_out = [0.00032515179570229987, 217.23336631977338, 494.9
      378401524241, 184.6822048577565, 27.51285545580364];

  for i=1:5
      set_start_value(postATR_in_mol[i] , ini_postATR_in[i])
      set_start_value(postATR_out_mol[i] , ini_postATR_out[i])
  end

  @variable(model, 273 <= postATR_in_T, start = 1323.00);
  @variable(model, 273 <= postATR_out_T, start = 600.00);

  @variable(model, 0 >= postATR_Q, start = -34472000)

  # Expressions
  postATR_H_out = build_enthalpy(model, postATR_out_T, par)
  postATR_H_in = build_enthalpy(model, postATR_in_T, par)

  # Constraints
  # Mass balance
  @NLconstraint(model, postATR_out_mol[1] - postATR_in_mol[1] == 0)
  @NLconstraint(model, postATR_out_mol[2] - postATR_in_mol[2] == 0)
  @NLconstraint(model, postATR_out_mol[3] - postATR_in_mol[3] == 0)
  @NLconstraint(model, postATR_out_mol[4] - postATR_in_mol[4] == 0)
  @NLconstraint(model, postATR_out_mol[5] - postATR_in_mol[5] == 0)

  # Energy balance
  @NLconstraint(model, sum(postATR_H_out[i]*postATR_out_mol[i] - postATR_H_in[i]
      *postATR_in_mol[i] for i=1:5) - postATR_Q==0)
  return model;
end
```

B.10 8ITSR.jl

8ITSR.jl contains the function ITSR_model() and builds the submodel for the ITSR unit. Note that there is only 1 reaction happening, which is the water-gas shift reaction and not the steam methane reforming equations like the other earlier reactors in the process. In reality, an ITSR is operating adiabatic, but an internal cooler is built into the model such that the recovered amount of heat in this process is both from cooling the inlet stream to its operating temperature and the heat recovered from the reaction heat.

```
function ITSR_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= itsr_in_mol[1:5]); # Stream 9
  @variable(model, 0 <= itsr_out_mol[1:5]); # Stream 10

  ini_itsr_in = [0.00032515179570229987, 217.23336631977338, 494.9378401524241,
      184.6822048577565, 27.51285545580364];  # CH4 H2O H2 CO CO2
  ini_itsr_out = [0.00032515179570229987, 55.89429423256371, 656.2769122396338,
      23.343132770546816, 188.85192754301332];

  for i=1:5
      set_start_value(itsr_in_mol[i] , ini_itsr_in[i])
      set_start_value(itsr_out_mol[i] , ini_itsr_out[i])
  end

  @variable(model, 550 <= itsr_in_T <= 650, start = 600.00);
  @variable(model, 473 <= itsr_out_T <= 573, start = 523.00);

  @variable(model, 0 >= itsr_Q, start = -5913000)

  # Expressions
  #itsr_Kwgsr_model = @NLexpression(model, exp(5693.5/itsr_out_T + 1.077*log(
      itsr_out_T) + 5.44e-4*itsr_out_T - 1.125e-7*itsr_out_T^2 - 49170/(
      itsr_out_T^2)-13.148))
  itsr_ksi_wgsr = @NLexpression(model, itsr_in_mol[2] - itsr_out_mol[2])

  itsr_K_wgsr_model = K_wgsr(model, itsr_out_T, par);
  itsr_ntot = @NLexpression(model, sum(itsr_out_mol[i] for i=1:5))

  itsr_H_out = build_enthalpy(model, itsr_out_T, par)
  itsr_H_in = build_enthalpy(model, itsr_in_T, par)

  # Constraints
  # Mass balance
  @NLconstraint(model, itsr_out_mol[1] - itsr_in_mol[1] == 0)
  @NLconstraint(model, itsr_K_wgsr_model*((itsr_out_mol[4]/itsr_ntot) * (
      itsr_out_mol[2]/itsr_ntot)) - (((itsr_out_mol[5]/itsr_ntot) * (
      itsr_out_mol[3]/itsr_ntot))) == 0)
  @NLconstraint(model, itsr_out_mol[3] - itsr_in_mol[3] - itsr_ksi_wgsr == 0)
  @NLconstraint(model, itsr_out_mol[4] - itsr_in_mol[4] + itsr_ksi_wgsr == 0)
  @NLconstraint(model, itsr_out_mol[5] - itsr_in_mol[5] - itsr_ksi_wgsr == 0)

  # Energy balance
  @NLconstraint(model, sum(itsr_H_out[i]*itsr_out_mol[i] - itsr_H_in[i]*
      itsr_in_mol[i] for i=1:5) - itsr_Q==0)

  # Energy balance - equipment specification
  #@NLconstraint(model, itsr_out_T - par.itsr.out_T == 0)
  return model;
end
```

B.11 9PreCondensate.jl

9PreCondensate.jl contains the function preCond_model() and builds the submodel for the cooler before the process condensate unit. This submodel is a simple heat exchanger that calculates the amount of heat recovered when cooling down the outlet stream from ITSR to reach the operating temperature of the process condensate unit.

```
function preCond_model(model, par)
  # Variables
  # CH4, H2O, H2, CO, CO2
  @variable(model, 0 <= preCond_in_mol[1:5]); # Stream 10
  @variable(model, 0 <= preCond_out_mol[1:5]); # Stream 11

  ini_preCond_in = [0.00032515179570229987, 55.89429423256371, 656.2769122396338
      , 23.343132770546816, 188.85192754301332];  # CH4 H2O H2 CO CO2
  ini_preCond_out = [0.00032515179570229987, 55.89429423256371, 656.2
      769122396338, 23.343132770546816, 188.85192754301332];

  for i=1:5
    set_start_value(preCond_in_mol[i] , ini_preCond_in[i]);
    set_start_value(preCond_out_mol[i] , ini_preCond_out[i]);
  end

  @variable(model, 273 <= preCond_in_T, start = 523.00);
  @variable(model, 293 <= preCond_out_T <= 333, start = 313.00);
  preCond_H_out = build_enthalpy(model, preCond_out_T, par)
  preCond_H_in = build_enthalpy(model, preCond_in_T, par)
  @variable(model, 0 >= preCond_Q, start = -6345000);

  # Constraints
  # Mass balance
  @NLconstraint(model, preCond_out_mol[1] - preCond_in_mol[1] == 0)
  @NLconstraint(model, preCond_out_mol[2] - preCond_in_mol[2] == 0)
  @NLconstraint(model, preCond_out_mol[3] - preCond_in_mol[3] == 0)
  @NLconstraint(model, preCond_out_mol[4] - preCond_in_mol[4] == 0)
  @NLconstraint(model, preCond_out_mol[5] - preCond_in_mol[5] == 0)

  # Energy balance
  @NLconstraint(model, sum(preCond_H_out[i]*preCond_out_mol[i] - preCond_H_in[i]
      *preCond_in_mol[i] for i=1:5) - preCond_Q==0);

  # Energy balance - equipment specification
  #@NLconstraint(model, preCond_out_T - par.preCond.out_T == 0);

  return model;
end
```

B.12 10Condensate.jl

10Condensate.jl file contains the function Cond_model() which builds the submodel for the process condensate model which is modeled as a flash tank using flash equations, Raoults law, and Rachford-Rice equations. As the flash tank only splits streams based on phases, no energy balances are formed in this submodel.

```
function Cond_model(model, par)
  @variable(model, 0 <= cond_in_mol[1:5]); # Stream 11
  #@variable(model, 0 <= cond_outPurge_mol[1:5]); # Stream 13 (H2O stream)
  #@variable(model, 0 <= cond_outProduct_mol[1:5]); # Stream 12 (PSA)
  @variable(model, 0 <= cond_liq_frac[1:5]<=1);  # Outlet liquid phase fractions
  @variable(model, 0 <= cond_vap_frac[1:5]<=1);
  @variable(model, 0 <= cond_L, start = 141);
  @variable(model, 0 <= cond_V, start = 923);

  ini_cond_in = [0.00032515179570229987, 55.89429423256371, 656.2769122396338,
      23.343132770546816, 188.85192754301332];  # CH4 H2O H2 CO CO2
  ini_cond_liq_frac = [0.01,0.999,0.01,0.01,0.01];
  ini_cond_vap_frac = [0.99,0.001,0.99,0.99,0.99];

  for i=1:5
    set_start_value(cond_in_mol[i] , ini_cond_in[i]);
    set_start_value(cond_liq_frac[i], ini_cond_liq_frac[i]);
    set_start_value(cond_vap_frac[i], ini_cond_vap_frac[i]);
  end

  @variable(model, 273 <= cond_in_T, start = 313.00);
  @variable(model, 273 <= cond_L_T, start = 313.00);
  @variable(model, 273 <= cond_V_T, start = 313.00);

  psat_H2O = @NLexpression(model, (10^(par.cond.ant_par[1] - par.cond.ant_par[2]
      /(cond_in_T + par.cond.ant_par[3])))/par.cond.cond_P);
  cond_K = @expression(model, [1e6, psat_H2O, 1e6, 1e6, 1e6]);
  cond_F = @expression(model, sum(cond_in_mol[i] for i = 1:5));
  cond_z = @NLexpression(model, [i=1:5], cond_in_mol[i]/cond_F);

  # Constraints
  # Flash equations and Raoults law
  for i=1:5
    @NLconstraint(model, cond_z[i] - cond_liq_frac[i]*(1+(cond_V/cond_F*(cond_K[
        i]-1))) == 0);
    @NLconstraint(model, cond_vap_frac[i] - cond_K[i]*cond_liq_frac[i] == 0);
  end

  # Rachford-Rice equation
  @NLconstraint(model, sum(cond_z[i]*(cond_K[i]-1)/(1+(cond_V/cond_F*(cond_K[i]-
      1)))  for i = 1:5) == 0); # insert sum rice equation here

  # Total mass balance
  @NLconstraint(model, cond_L + cond_V - cond_F == 0);

  # Energy balance - Equipment specification
  @NLconstraint(model, cond_V_T - cond_in_T == 0);
  @NLconstraint(model, cond_L_T - cond_in_T == 0);

  return model;
end
```

B.13 11PSA.jl

11PSA.jl file contains the function PSA_model() which is the last submodel that builds the model for the entire plant. The function uses split ratios imported from the parameter set from 0par.jl file and calculates the product stream and purge stream. No energy balances are formed in this submodel, as PSA is modeled only as a stream splitter.

```
function PSA_model(model, par)
  @variable(model, 0 <= psa_in_mol[1:5]); # Stream 12
  @variable(model, 0 <= psa_outPurge_mol[1:5]); # Stream 15 (Purge)
  @variable(model, 0 <= psa_outProduct_mol[1:5]); # Stream 14 (H2)

  ini_psa_in = [0.00032515179570229987, 0.05589429423256371, 656.2769122396338,
      23.343132770546816, 188.85192754301332];  # CH4 H2O H2 CO CO2
  ini_psa_outPurge = [0.0003219002777452769, 0.05533535129023807, 0.6
      562769122396344, 23.10970144284135, 186.9634082675832];
  ini_psa_outProduct = [3.2515179570229986e-06, 0.0005589429423256371, 655.6
      206353273942, 0.23343132770546818, 1.8885192754301332];
  for i=1:5
    set_start_value(psa_in_mol[i] , ini_psa_in[i]);
    set_start_value(psa_outPurge_mol[i], ini_psa_outPurge[i]);
    set_start_value(psa_outProduct_mol[i], ini_psa_outProduct[i]);
  end

  @variable(model, 273 <= psa_in_T, start = 313.00);
  @variable(model, 273 <= psa_outPurge_T, start = 313.00);
  @variable(model, 273 <= psa_outProduct_T, start = 313.00)

  # Constraints
  # Mass balance
  for i = 1:5
    @NLconstraint(model, par.psa.splitratio[i]*psa_in_mol[i] -
        psa_outProduct_mol[i] == 0);
    @NLconstraint(model, (1-par.psa.splitratio[i])*psa_in_mol[i] -
        psa_outPurge_mol[i] == 0);
  end

  @NLconstraint(model, psa_outPurge_T - psa_in_T == 0);
  @NLconstraint(model, psa_outProduct_T - psa_in_T == 0);
  return model;
end
```

B.14 active.jl

The function printActive uses the optimized model as input and checks if specific variables in the model chosen as input variables are at their respective bounds or not. If a variable is at its bound, it is classified as active and needs to be controlled which means that it cannot be used for control. In the loss.jl file, which contains the function that calculates the loss of optimality, the variables that are active are set to their respective nominal setpoint as equality constraints.

```
function printActive(model)
    # CV:     S/C_ratio, n_O2,    prePR_T,    PR_T,    ATR_T,    postATR_T,    ITSR_T
        ,  Cond_T
    l_bound = [0.0,       0.0,     643.0,      609.2,   1273,     550,          473,
            293];
    u_bound = [5.0,       100.0,   743.0,      709.2,   1373,     650,          573,
            333];

    variables = [value(model[:SC_ratio]),
                 value(model[:nO2]),
                 value(model[:prePR_out_T]),
                 value(model[:pr_out_T]),
                 value(model[:atr_out_T]),
                 value(model[:postATR_out_T]),
                 value(model[:itsr_out_T]),
                 value(model[:preCond_out_T])]
    variable_name = ["SC_ratio", "n_O2", "T_prePR", "T_PR", "T_ATR", "T_postATR"
        , "T_ITSR", "T_Cond"]
    for i = 1:8
        if variables[i] ≈ l_bound[i]
            println("The variable ", variable_name[i], " is at its lower bound
                of ",l_bound[i]);
        elseif variables[i] ≈ u_bound[i]
            println("The variable ", variable_name[i], " is at its upper bound
                of ",u_bound[i]);
        else
            println("The variable ",variable_name[i], " is not active, with a
                value of ", variables[i], ", where the bounds are: ",l_bound[i]
                ," ≤ ",variable_name[i], " ≤ ",u_bound[i]);
        end
    end
end
```

B.15 compWork.jl

compWork.jl contains the functions Wrev() and compT(). The Wrev() function uses the model, mole stream, initial pressure, pressure, initial temperature, and parameters to calculate the work required to compress a gas, while compT() calculates the final temperature after compression. The compressor is assumed to be isentropic.

```
# Function for calculating the reversible adiabatic isentropic process
    compression work

function Wrev(model, n, P1, P2, T1, par)
    #ntot = @NLexpression(model, sum(n[i] for i = 1:5));
    #cp = @NLexpression(model, sum(n[i]*par.cp_list[i]/ntot for i = 1:5));
    gamma = @NLexpression(model, 5/3.0);
    return @NLexpression(model, n*5/2*par.R*T1*((P2/P1)^((gamma-1)/gamma)-1));
end

function compT(model, T1, P1, P2)
    #ntot = @NLexpression(model, sum(n[i] for i = 1:5));
    #cp = @NLexpression(model, sum(n[i]*par.cp_list[i]/ntot for i = 1:5));
    gamma = @NLexpression(model, 5/3.0);
    return @NLexpression(model, T1*(P1/P2)^((gamma - 1)/gamma))
end
```

B.16 dataframe.jl

The printTable() function in the dataframe.jl file uses the model as inputs and prints out all the variables in the terminal in a readable table. The tables are sorted in mole stream, composition, and mass data, in addition to other variables like heat required/recovered in the heat exchangers.

```
#using Pkg
#Pkg.add("DataFrames")
#Pkg.add("PrettyTables")

function printTable(model)
    # Stream table (mole)
    CH₄ = [value(model[:mix_in_mol][1]), 0.0, value(model[:prePR_in_mol][1]),
        value(model[:pr_in_mol][1]),
     value(model[:preGHR_in_mol][1]), value(model[:ghr_in_mol][1]), value(model[
         :atr_in_mol][1]),
     value(model[:postATR_in_mol][1]), value(model[:itsr_in_mol][1]), value(
         model[:preCond_in_mol][1]),
     value(model[:cond_in_mol][1]), value(model[:cond_liq_frac][1])*value(model[
         :cond_L]), value(model[:psa_in_mol][1]),
     value(model[:psa_outProduct_mol][1]), value(model[:psa_outPurge_mol][1])];

    H₂O = [value(model[:mix_in_mol][2]), value(model[:H2Ostream]), value(model[:
        prePR_in_mol][2]),
    value(model[:pr_in_mol][2]), value(model[:preGHR_in_mol][2]), value(model[:
        ghr_in_mol][2]),
    value(model[:atr_in_mol][2]),value(model[:postATR_in_mol][2]), value(model[:
        itsr_in_mol][2]),
    value(model[:preCond_in_mol][2]), value(model[:cond_in_mol][2]), value(model
        [:cond_liq_frac][2])*value(model[:cond_L]),
    value(model[:psa_in_mol][2]), value(model[:psa_outProduct_mol][2]), value(
        model[:psa_outPurge_mol][2])];

    H₂ = [value(model[:mix_in_mol][3]), 0.0, value(model[:prePR_in_mol][3]),
        value(model[:pr_in_mol][3]),
    value(model[:preGHR_in_mol][3]), value(model[:ghr_in_mol][3]), value(model[:
        atr_in_mol][3]),
    value(model[:postATR_in_mol][3]), value(model[:itsr_in_mol][3]), value(model
        [:preCond_in_mol][3]),
    value(model[:cond_in_mol][3]), value(model[:cond_liq_frac][3])*value(model[:
        cond_L]), value(model[:psa_in_mol][3]),
    value(model[:psa_outProduct_mol][3]), value(model[:psa_outPurge_mol][3])];

    CO = [value(model[:mix_in_mol][4]), 0.0, value(model[:prePR_in_mol][4]),
        value(model[:pr_in_mol][4]),
    value(model[:preGHR_in_mol][4]), value(model[:ghr_in_mol][4]), value(model[:
        atr_in_mol][4]),
    value(model[:postATR_in_mol][4]), value(model[:itsr_in_mol][4]), value(model
        [:preCond_in_mol][4]),
     value(model[:cond_in_mol][4]), value(model[:cond_liq_frac][4])*value(model[
         :cond_L]), value(model[:psa_in_mol][4]),
    value(model[:psa_outProduct_mol][4]), value(model[:psa_outPurge_mol][4])];

    CO₂ = [value(model[:mix_in_mol][5]), 0.0, value(model[:prePR_in_mol][5]),
        value(model[:pr_in_mol][5]),
    value(model[:preGHR_in_mol][5]), value(model[:ghr_in_mol][5]), value(model[:
        atr_in_mol][5]),
    value(model[:postATR_in_mol][5]), value(model[:itsr_in_mol][5]), value(model
        [:preCond_in_mol][5]),
     value(model[:cond_in_mol][5]), value(model[:cond_liq_frac][5])*value(model[
         :cond_L]), value(model[:psa_in_mol][5]),
    value(model[:psa_outProduct_mol][5]), value(model[:psa_outPurge_mol][5])];

    $C_2H_6$ = [value(model[:mix_in_mol][6]), 0.0, value(model[:prePR_in_mol][6]),
        value(model[:pr_in_mol][6]),
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];

    $C_3H_8$ = [value(model[:mix_in_mol][7]), 0.0, value(model[:prePR_in_mol][7]),
        value(model[:pr_in_mol][7]),
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];

    $nC_4H_{10}$ = [value(model[:mix_in_mol][8]), 0.0, value(model[:prePR_in_mol][8]),
         value(model[:pr_in_mol][8]),
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];

    $iC_4H_{10}$ = [value(model[:mix_in_mol][9]), 0.0, value(model[:prePR_in_mol][9]),
         value(model[:pr_in_mol][9]),
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];

    $C_{5+}$ = [value(model[:mix_in_mol][10]), 0.0, value(model[:prePR_in_mol][10]),
        value(model[:pr_in_mol][10]),
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];

    T = [value(model[:mix_in_T]), value(model[:H2O_T]), value(model[:prePR_in_T]
        ), value(model[:pr_in_T]),
    value(model[:preGHR_in_T]), value(model[:ghr_in_T]), value(model[:atr_in_T])
        , value(model[:postATR_in_T]),
    value(model[:itsr_in_T]), value(model[:preCond_in_T]), value(model[:
        cond_in_T]), value(model[:cond_L_T]),
    value(model[:psa_in_T]), value(model[:psa_outProduct_T]), value(model[:
        psa_outPurge_T])];

    streamdf = DataFrame(T = T, $CH_4$ = $CH_4$, $H_2O$ = $H_2O$, $H_2$ = $H_2$, CO = CO, $CO_2$ = CO
        2,
     $C_2H_6$ = $C_2H_6$, $C_3H_8$ = $C_3H_8$, $nC_4H_{10}$ = $nC_4H_{10}$, $iC_4H_{10}$ = $iC_4H_{10}$, $C_{5+}$ = $C_{5+}$);


    # Other variable table
    variable = ["prePR_Q","preGHR_Q","ghr_Q", "postATR_Q", "itsr_Q","preCond_Q",
    "nO2", "S/C_ratio",  "F_H2", "F_H2_heat", "F_NG_Heat", "F_NG", "F_fluegas",
        "F_inj", "Profit [\$/h]"]#,"additional_Q"]
    values = [value(model[:prePR_Q]),value(model[:preGHR_Q]),value(model[:ghr_Q]
        ), value(model[:postATR_Q]),
    value(model[:itsr_Q]),value(model[:preCond_Q]),value(model[:nO2]),value(
        model[:SC_ratio]) , value(model[:F_H2]),
    value(model[:F_H2_heat]),value(model[:F_NG_heat]),value(model[:F_NG]),value(
        model[:F_fluegas]),
    value(model[:F_inj]),objective_value(model)]#,value(model[:additional_Q])];
    otherdf = DataFrame(Variable = variable, Value = values);
```

```
# Mass table
C = 12.01;
H = 1.008;
O = 16;
# CH4, H2O, H2, CO, CO2, C2H6, C3H8, n-C4H10, i-C4H10, C5+
Mm = [C+H*4, H*2+O, H*2, C+O, C+O*2, C*2+H*6, C*3+H*8, C*4+H*10, C*4+H*10, C
    *5+H*12];
mass = zeros(15);
for i=1:10
    mass[1] += value(model[:mix_in_mol][i]) * Mm[i];
    mass[3] += value(model[:prePR_in_mol][i]) * Mm[i];
    mass[4] += value(model[:pr_in_mol][i]) * Mm[i];
end

mass[2] = value(model[:H2Ostream])*Mm[2];

for j=1:5
    mass[5] += value(model[:preGHR_in_mol][j])*Mm[j]
    mass[6] += value(model[:ghr_in_mol][j])*Mm[j]
    mass[7] += value(model[:atr_in_mol][j])*Mm[j]
    mass[8] += value(model[:postATR_in_mol][j])*Mm[j]
    mass[9] += value(model[:itsr_in_mol][j])*Mm[j]
    mass[10] += value(model[:preCond_in_mol][j])*Mm[j]
    mass[11] += value(model[:cond_in_mol][j])*Mm[j]
    mass[12] += value(model[:psa_in_mol][j])*Mm[j]
    mass[13] += value(model[:cond_liq_frac][j])*value(model[:cond_L])*Mm[j]
    mass[14] += value(model[:psa_outProduct_mol][j])*Mm[j]
    mass[15] += value(model[:psa_outPurge_mol][j])*Mm[j]
end
oxygenstream = zeros(15);
oxygenstream[7] = value(model[:nO2])*32;
massdf = DataFrame(Mass= mass, O₂ = oxygenstream);


# Composition table
xCH4 = zeros(15);
xH2O = zeros(15);
xH2 = zeros(15);
xCO = zeros(15);
xCO2 = zeros(15);
xC2H6 = zeros(15);
xC3H8 = zeros(15);
xnC4H10 = zeros(15);
xiC4H10 = zeros(15);
xC5 = zeros(15);

totalmolestream = zeros(15);
for i=1:15
    totalmolestream[i] = CH₄[i] + H₂O[i] + H₂[i] + CO[i] + CO₂[i] + C₂H₆[i]
        + C₃H₈[i] + nC₄H₁₀[i] + iC₄H₁₀[i] + C₅₊[i]
end

for i=1:15
    xCH4[i] = CH₄[i]/totalmolestream[i]
```

```
        xH2O[i] = H₂O[i]/totalmolestream[i]
        xH2[i] = H₂[i]/totalmolestream[i]
        xCO[i] = CO[i]/totalmolestream[i]
        xCO2[i] = CO₂[i]/totalmolestream[i]
        xC2H6[i] = C₂H₆[i]/totalmolestream[i]
        xC3H8[i] = C₃H₈[i]/totalmolestream[i]
        xnC4H10[i] = nC₄H₁₀[i]/totalmolestream[i]
        xiC4H10[i] = iC₄H₁₀[i]/totalmolestream[i]
        xC5[i] = C₅₊[i]/totalmolestream[i]
    end

    compositiondf = DataFrame(xCH₄ = xCH4, xH₂O = xH2O, xH₂ = xH2, xCO = xCO,
        xCO₂ = xCO2,
     xC₂H₆ = xC2H6, xC₃H₈ = xC3H8, xnC₄H₁₀ = xnC4H10, xiC₄H₁₀ = xiC4H10, xC₅₊ =
         xC5);

    return streamdf, otherdf, massdf,compositiondf
end
```

B.17 diagM.jl

The function diagWn() and diagWd() uses the nominal values calculated from nominal() function in nominal_case.jl file, to calculate the diagonal scaling matrices for disturbances- and measurement noises, respectively. The last function diag_to_xlsx() stores the resulting diagonal matrices into excel sheets which can be imported for later calculations, e.g. Bound and Branch algorithm for choosing the best measurement subset.

```
include("nominal_case.jl")
using LinearAlgebra, XLSX

nominal_states, nom_f = nominal();
#@show(nominal_states[1:142], allrows = true)

function diagWn(nom)
    ##### Index list #####
    # [1:126] - flow
    # [127:131] - compfrac
    # [132] - flow
    # [133:137] - compfrac
    # [138:152] - flow
    # [153:171] - temp
    # [172:177] - Q
    # [178:184] - flow
    flow_pert = 0.02; # relative% [kmol/h]
    temp_pert = 1; # absolute [K]
    Q_pert = 0.05; # relative% [kmol/h]
    comp_pert = 0.01; # absolute [-]
    for i in eachindex(nom)
        if 1<=i<=126
```

```
            nom[i] = nom[i]*flow_pert;
        elseif 127<=i<=131
            nom[i] = comp_pert;
        elseif i == 132
            nom[i] = nom[i]*flow_pert;
        elseif 133<=i<=137
            nom[i] = comp_pert;
        elseif 138<=i<=152
            nom[i] = nom[i]*flow_pert;
        elseif 153<=i<=171
            nom[i] = temp_pert;
        elseif 172<=i<=177
            nom[i] = nom[i]*Q_pert;
        else
            nom[i] = nom[i]*flow_pert;
        end
    end
    df = DataFrame(diagm(nom),:auto);
    return df
end

function diagWd()
    d= [145.4, 0.14, 3.347]
    for i in eachindex(d)
        d[i] = d[i]*0.1
    end
    return DataFrame(diagm(d),:auto)
end


function diag_to_xlsx()
    df1 = diagWn(nominal_states)
    XLSX.writetable("data/Wn.xlsx", df1);
    df2 = diagWd()
    XLSX.writetable("data/Wd.xlsx", df2)
end
#diag_to_xlsx()
```

B.18 enthalpy.jl

build_enthalpy() function calculates the specific enthalpy depending on the temperature by using coefficients for each component from the literature and summing them up.

```
# Order of components: CH4, H2O, H2, CO, CO2, C2H6, C3H8, n-C4H10, i-C4H10, C5+
# Order of coefficients: A, B, C, H298, Cp298/R <- remember to multiply with
    8.314
heavy_const = [1.702 9.081 -2.164 -74.520 4.217;
                3.470 1.450 0.000 -241.818 4.038;
                3.249 0.422 0.000 0.000 3.468;
                3.376 0.557 0.000 -110.525 3.507;
                5.457 0.557 0.000 -393.509 4.467;
                1.131 19.225 -5.561 -83.820 6.369;
                1.213 28.785 -8.824 -104.680 9.011;
                1.935 36.915 -11.402 -125.790 11.298;
                1.935 36.915 -11.402 -125.790 11.298;
                2.464 45.351 -14.111 -146.760 14.731]


function build_enthalpy(model, T, par)
  return @NLexpression(model, [i = 1:10], (par.hconst[i,4] + (((par.hconst[i,1]*
      T + par.hconst[i,2]/2*T^2*10^(-3) + par.hconst[i,3]/3*T^3*10^(-6))
  - (par.hconst[i,1]*298 + par.hconst[i,2]/2*298^2*10^(-3) + par.hconst[i,3]/3*
      298^3*10^(-6)))*8.314/1000))*1000)
end
```

B.19 equilibrium.jl

The equilibrium.jl file contains the functions K_smr() and K_wgsr(), which calculates the equilibrium constant for steam methane reforming equation and the water gas shift reaction for a given temperature, respectively. The coefficients used for both functions are imported from literature.

```
# Row order: CH4, H2O, H2, CO, CO2

smr_const = [-1.51735037e-05, 4.79654639e-02, -2.91867555e+01];
wgsr_const = [2.95405564e-06, -8.48603361e-03, 4.99373583e+00];

function K_smr(model, T, par)
    T -= 273.15; # Convert kelvin to celsius
    return @NLexpression(model, exp(par.smr_const[1]*T^2 + par.smr_const[2]*T +
        par.smr_const[3]));
end

function K_wgsr(model, T, par)
    T -= 273.15; # Convert kelvin to celsius
    return @NLexpression(model, exp(par.wgsr_const[1]*T^2 + par.wgsr_const[2]*T
        + par.wgsr_const[3]));
end
```

B.20 finite_diff.jl

finite_diff.jl file contains two functions, where the first function, finite_diff_1() calculates the double partial derivate of the same variable, e.g. $f(x+h, y+k)$, where x and y are the same variable that is being perturbated, depending on the option it was given and the step size of the perturbation. The different options decide which variable to perturbate, either input or disturbance, and either positive or negative change. The second function finite_diff_2(), is almost identical to the first function, only that it has two different options, one for each variable as it calculates the double partial derivate with respect to two different variables, e.g. $f(x+h, y+k)$.

```julia
using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")
include("nominal_case.jl")

# Options for finite diff (eps = 1e-5)
# 1: u1+h, n_O2
# 2: u2+h, T_prePR
# 3: u3+h, T_ATR
# 4: d1+k, NG flow
# 5: d2+k, P_el
# 6: d3+k, P_H2
# -1: u1-h, n_O2
# -2: u2-h, T_prePR
# -3: u3-h, T_ATR
# -4: d1-k, NG flow
# -5: d2-k, P_el
# -6: d3-k, P_H2

eps = 1e-5

function finite_diff_1(option, eps)
    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-10, "constr_viol_tol" => 1e-10,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);
    h = 0;
    k = 0;
    d1 = par.init.init_stream;
    d2 = par.elCost;
    d3 = par.P_H2;
    if option == 1 # u1 + h
        h = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805+eps== 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 2 # u2 + h
        h = eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283+eps== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 3 # u3 + h
        h = eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818+eps== 0);
    elseif option == 4 # d1 + h
        k = eps*par.init.init_stream;
        d1 = par.init.init_stream+eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 5 # d2 + h
        k = eps*par.elCost;
        d2 = par.elCost+eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 6 # d3 + h
        k = eps*par.P_H2;
        d3 = par.P_H2+eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -1 # u1 - h
        h = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805-eps== 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -2 # u2 - h
        h = eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283-eps== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
```

```julia
    elseif option == -3 # u3 - h
        h = eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818-eps== 0);
    elseif option == -4 # d1 - h
        k = eps*par.init.init_stream;
        d1 = par.init.init_stream-eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -5 # d2 - h
        k = eps*par.elCost;
        d2 = par.elCost-eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -6 # d3 - h
        k = eps*par.P_H2;
        d3 = par.P_H2-eps;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    else
        print("Option not valid")
    end

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints ######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);

    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);

    optimize!(m)
    if termination_status(m) == LOCALLY_SOLVED || termination_status(m) ==
        OPTIMAL || termination_status(m) == ALMOST_LOCALLY_SOLVED
        return objective_value(m), h, k
    else
        return Inf, h, k
    end
```

```
end

function finite_diff_2(option_x, option_y, eps;m=0.)
    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-10, "constr_viol_tol" => 1e-10
             ,"print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);
    h = 0;
    k = 0;
    d1 = par.init.init_stream;
    d2 = par.elCost;
    d3 = par.P_H2;

    if option_x == 1 # u1 + h
        h = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805+eps== 0);
    elseif option_x == 2 # u2 + h
        h = eps*644.5953165006283;
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283+eps== 0);
    elseif option_x == 3 # u3 + h
        h = eps*1291.817465833818;
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818+eps== 0);
    elseif option_x == 4 # d1 + k
        h = eps*par.init.init_stream;
        d1 = par.init.init_stream+eps;
    elseif option_x == 5 # d2 + k
        h = eps*par.elCost;
        d2 = par.elCost+eps;
    elseif option_x == 6 # d3 + k
        h = eps*par.P_H2;
        d3 = par.P_H2+eps;
    elseif option_x == -1 # u1 - h
        h = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805-eps== 0);
    elseif option_x == -2 # u2 - h
        h = eps*644.5953165006283;
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283-eps== 0);
    elseif option_x == -3 # u3 - h
        h = eps*1291.817465833818;
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818-eps== 0);
    elseif option_x == -4 # d1 - k
        h = eps*par.init.init_stream;
        d1 = par.init.init_stream-eps;
    elseif option_x == -5 # d2 - k
        h = eps*par.elCost;
        d2 = par.elCost-eps;
    elseif option_x == -6 # d3 - k
        h = eps*par.P_H2;
        d3 = par.P_H2-eps;
    else
        print("Option not valid")
    end

    if option_y == 1 # u1 + h
        k = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805+eps== 0);
    elseif option_y == 2 # u2 + h
        k = eps*644.5953165006283;
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283+eps== 0);
    elseif option_y == 3 # u3 + h
        k = eps*1291.817465833818;
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818+eps== 0);
    elseif option_y == 4 # d1 + k
        k = eps*par.init.init_stream;
        d1 = par.init.init_stream+eps;
    elseif option_y == 5 # d2 + k
        k = eps*par.elCost;
        d2 = par.elCost+eps;
    elseif option_y == 6 # d3 + k
        k = eps*par.P_H2;
        d3 = par.P_H2+eps;
    elseif option_y == -1 # u1 - h
        k = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805-eps== 0);
    elseif option_y == -2 # u2 - h
        k = eps*644.5953165006283;
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283-eps== 0);
    elseif option_y == -3 # u3 - h
        k = eps*1291.817465833818;
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818-eps== 0);
    elseif option_y == -4 # d1 - k
        k = eps*par.init.init_stream;
        d1 = par.init.init_stream-eps;
    elseif option_y == -5 # d2 - k
        k = eps*par.elCost;
        d2 = par.elCost-eps;
    elseif option_y == -6 # d3 - k
        k = eps*par.P_H2;
        d3 = par.P_H2-eps;
    else
        print("Option not valid")
    end
    if abs(option_y) != 1 && abs(option_x) !=1
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
    end
    if abs(option_y) != 2 && abs(option_x) !=2
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
    end
    if abs(option_y) != 3 && abs(option_x) !=3
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    end

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints ######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################ Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############# Initial values ########################################
    for i = 1:10
```

```
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);

    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);

    optimize!(m)

    if termination_status(m) == LOCALLY_SOLVED || termination_status(m) ==
        OPTIMAL || termination_status(m) == ALMOST_LOCALLY_SOLVED
        return objective_value(m), h, k
    else
        return Inf, h, k
    end
    #return f, h, k
end
```

B.21 Fmatrix.jl

In the Fmatrix.jl file, there are two functions. The first function Fmatrix() imports all the functions for calculating the gain matrices and Hessian matrices from other files to calculate the optimal sensitivity matrix, F. The second function print_F() prints the resulting matrix in a readable way in the output terminal.

```
include("gain_d.jl")
include("gain_u.jl")
include("J_uu.jl")
include("J_ud.jl")
include("nominal_case.jl")
using LinearAlgebra, XLSX

eps = 1e-5
nominal_values, nominal_J = nominal();


function Fmatrix(nominal)
    G_y_matrix = matrix_Gy(nominal, eps);
    G_y_d_matrix = matrix_Gyd(nominal, eps);
    J_uu_matrix = J_uu(1e-2);
    J_ud_matrix = J_ud(1e-2);
    F = G_y_d_matrix' - G_y_matrix'*inv(J_uu_matrix)*J_ud_matrix;
    return F
end

F = Fmatrix(nominal_values);

function print_F(matrix)
    return DataFrame(Variable = variable_name,
                     Nominal = nominal_values,
                     ∂y∂d_1 = matrix[:,1],
                     ∂y∂d_2 = matrix[:,2],
                     ∂y∂d_3 = matrix[:,3])
end

#show(print_F(Fmatrix(nominal_values)), allrows=true)
#show(nullspace(Fmatrix(nominal_value)))
XLSX.writetable("data/F.xlsx", print_F(Fmatrix(nominal_values)));
```

B.22 gain_d.jl

The function G_yd() calculates the output gain with respect to disturbance change for each disturbance perturbation, which is assembled to a matrix in the function matrix_Gyd and exported to an Excel sheet. printG_yd() prints out the G_yd matrix into a readable table in the output terminal.

```
variable_name = [
    "mix_in_mol[1]", "mix_in_mol[2]", "mix_in_mol[3]", "mix_in_mol[4]", "
        mix_in_mol[5]",
    "mix_in_mol[6]", "mix_in_mol[7]", "mix_in_mol[8]", "mix_in_mol[9]", "
        mix_in_mol[10]",
    "mix_out_mol[1]", "mix_out_mol[2]", "mix_out_mol[3]", "mix_out_mol[4]", "
        mix_out_mol[5]",
    "mix_out_mol[6]", "mix_out_mol[7]", "mix_out_mol[8]", "mix_out_mol[9]", "
        mix_out_mol[10]",
    "prePR_in_mol[1]", "prePR_in_mol[2]", "prePR_in_mol[3]", "prePR_in_mol[4]",
        "prePR_in_mol[5]",
    "prePR_in_mol[6]", "prePR_in_mol[7]", "prePR_in_mol[8]", "prePR_in_mol[9]",
        "prePR_in_mol[10]",
    "prePR_out_mol[1]", "prePR_out_mol[2]", "prePR_out_mol[3]", "prePR_out_mol
        [4]", "prePR_out_mol[5]",
    "prePR_out_mol[6]", "prePR_out_mol[7]", "prePR_out_mol[8]", "prePR_out_mol
        [9]", "prePR_out_mol[10]",
    "pr_in_mol[1]", "pr_in_mol[2]", "pr_in_mol[3]", "pr_in_mol[4]", "pr_in_mol
        [5]",
    "pr_in_mol[6]", "pr_in_mol[7]", "pr_in_mol[8]", "pr_in_mol[9]", "pr_in_mol
        [10]",
    "pr_out_mol[1]", "pr_out_mol[2]", "pr_out_mol[3]", "pr_out_mol[4]", "
        pr_out_mol[5]",
    "pr_out_mol[6]", "pr_out_mol[7]", "pr_out_mol[8]", "pr_out_mol[9]", "
        pr_out_mol[10]",
    "preGHR_in_mol[1]", "preGHR_in_mol[2]", "preGHR_in_mol[3]", "preGHR_in_mol
        [4]", "preGHR_in_mol[5]",
    "preGHR_out_mol[1]", "preGHR_out_mol[2]", "preGHR_out_mol[3]", "
        preGHR_out_mol[4]", "preGHR_out_mol[5]",
    "ghr_in_mol[1]", "ghr_in_mol[2]", "ghr_in_mol[3]", "ghr_in_mol[4]", "
        ghr_in_mol[5]",
    "ghr_out_mol[1]", "ghr_out_mol[2]", "ghr_out_mol[3]", "ghr_out_mol[4]", "
        ghr_out_mol[5]",
    "atr_in_mol[1]", "atr_in_mol[2]", "atr_in_mol[3]", "atr_in_mol[4]", "
        atr_in_mol[5]",
    "atr_out_mol[1]", "atr_out_mol[2]", "atr_out_mol[3]", "atr_out_mol[4]", "
        atr_out_mol[5]",
    "postATR_in_mol[1]", "postATR_in_mol[2]", "postATR_in_mol[3]", "
        postATR_in_mol[4]", "postATR_in_mol[5]",
    "postATR_out_mol[1]", "postATR_out_mol[2]", "postATR_out_mol[3]", "
        postATR_out_mol[4]", "postATR_out_mol[5]",
    "itsr_in_mol[1]", "itsr_in_mol[2]", "itsr_in_mol[3]", "itsr_in_mol[4]", "
        itsr_in_mol[5]",
    "itsr_out_mol[1]", "itsr_out_mol[2]", "itsr_out_mol[3]", "itsr_out_mol[4]",
        "itsr_out_mol[5]",
    "preCond_in_mol[1]", "preCond_in_mol[2]", "preCond_in_mol[3]", "
        preCond_in_mol[4]", "preCond_in_mol[5]",
    "preCond_out_mol[1]", "preCond_out_mol[2]", "preCond_out_mol[3]", "
        preCond_out_mol[4]", "preCond_out_mol[5]",
    "cond_in_mol[1]", "cond_in_mol[2]", "cond_in_mol[3]", "cond_in_mol[4]", "
        cond_in_mol[5]",
    "cond_L", "cond_liq_frac[1]", "cond_liq_frac[2]", "cond_liq_frac[3]", "
        cond_liq_frac[4]", "cond_liq_frac[5]",
    "cond_V", "cond_vap_frac[1]", "cond_vap_frac[2]", "cond_vap_frac[3]", "
        cond_vap_frac[4]", "cond_vap_frac[5]",
    "psa_in_mol[1]", "psa_in_mol[2]", "psa_in_mol[3]", "psa_in_mol[4]", "
        psa_in_mol[5]",
    "psa_outProduct_mol[1]", "psa_outProduct_mol[2]", "psa_outProduct_mol[3]", "
        psa_outProduct_mol[4]", "psa_outProduct_mol[5]",
    "psa_outPurge_mol[1]", "psa_outPurge_mol[2]", "psa_outPurge_mol[3]", "
        psa_outPurge_mol[4]", "psa_outPurge_mol[5]",
    "mix_in_T", "mix_out_T", "H2O_T", "prePR_in_T", "prePR_out_T", "preGHR_in_T"
        , "preGHR_out_T",
    "ghr_in_T", "ghr_out_T", "atr_in_T", "postATR_in_T", "postATR_out_T",
    "preCond_out_T", "cond_in_T", "cond_L_T", "cond_V_T", "psa_in_T", "
        psa_outProduct_T", "psa_outPurge_T",
    "prePR_Q", "preGHR_Q", "ghr_Q", "postATR_Q", "itsr_Q", "preCond_Q",
    "H2Ostream", "F_H2", "F_H2_heat", "F_NG", "F_NG_heat", "F_fluegas", "F_inj"
]
# Variables that are removed = S/C, nO2, T_prePR, T_PR, T_ATR, T_postATR, T_ITSR
    , T_cond

using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables, XLSX
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")
include("nominal_case.jl")

eps = 1e-5
nominal_values, nominal_J = nominal();

# 1: d1+k, NG_flow
# 2: d2+k, P_el
# 3: d3+k, P_H2
# -1: d1-k, NG_flow
# -2: d-k, P_el
# -3: d3-k, P_H2
function G_yd(nominal, option, eps)

    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-6, "constr_viol_tol" => 1e-8,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);
```

```julia
    d1 = par.init.init_stream;
    d2 = par.elCost;
    d3 = par.P_H2;
    if option == 1
        delta_d = eps*par.init.init_stream;
        d1 = par.init.init_stream*(1+eps);
    elseif option == 2
        delta_d = eps*par.elCost;
        d2 = par.elCost*(1+eps);
    elseif option == 3
        delta_d = eps*par.P_H2;
        d3 = par.P_H2*(1+eps);
    elseif option == -1
        delta_d = -eps*par.init.init_stream;
        d1 = par.init.init_stream*(1-eps);
    elseif option == -2
        delta_d = -eps*par.elCost;
        d2 = par.elCost*(1-eps);
    elseif option == -3
        delta_d = -eps*par.P_H2;
        d3 = par.P_H2*(1+eps);
    else
        print("Option not valid")
    end
    @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
    @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
    @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    ##################### Connection constraints #######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);

    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);

    optimize!(m)

    disturbed = [
        value(m[:mix_in_mol][1]), value(m[:mix_in_mol][2]), value(m[:mix_in_mol]
            [3]), value(m[:mix_in_mol][4]), value(m[:mix_in_mol][5]),
        value(m[:mix_in_mol][6]), value(m[:mix_in_mol][7]), value(m[:mix_in_mol]
            [8]), value(m[:mix_in_mol][9]), value(m[:mix_in_mol][10]),
        value(m[:mix_out_mol][1]), value(m[:mix_out_mol][2]), value(m[:
            mix_out_mol][3]), value(m[:mix_out_mol][4]), value(m[:mix_out_mol][
            5]),
        value(m[:mix_out_mol][6]), value(m[:mix_out_mol][7]), value(m[:
            mix_out_mol][8]), value(m[:mix_out_mol][9]), value(m[:mix_out_mol][
            10]),
        value(m[:prePR_in_mol][1]), value(m[:prePR_in_mol][2]), value(m[:
            prePR_in_mol][3]), value(m[:prePR_in_mol][4]), value(m[:
            prePR_in_mol][5]),
        value(m[:prePR_in_mol][6]), value(m[:prePR_in_mol][7]), value(m[:
            prePR_in_mol][8]), value(m[:prePR_in_mol][9]), value(m[:
            prePR_in_mol][10]),
        value(m[:prePR_out_mol][1]), value(m[:prePR_out_mol][2]), value(m[:
            prePR_out_mol][3]), value(m[:prePR_out_mol][4]), value(m[:
            prePR_out_mol][5]),
        value(m[:prePR_out_mol][6]), value(m[:prePR_out_mol][7]), value(m[:
            prePR_out_mol][8]), value(m[:prePR_out_mol][9]), value(m[:
            prePR_out_mol][10]),
        value(m[:pr_in_mol][1]), value(m[:pr_in_mol][2]), value(m[:pr_in_mol][3]
            ), value(m[:pr_in_mol][4]), value(m[:pr_in_mol][5]),
        value(m[:pr_in_mol][6]), value(m[:pr_in_mol][7]), value(m[:pr_in_mol][8]
            ), value(m[:pr_in_mol][9]), value(m[:pr_in_mol][10]),
        value(m[:pr_out_mol][1]), value(m[:pr_out_mol][2]), value(m[:pr_out_mol]
            [3]), value(m[:pr_out_mol][4]), value(m[:pr_out_mol][5]),
        value(m[:pr_out_mol][6]), value(m[:pr_out_mol][7]), value(m[:pr_out_mol]
            [8]), value(m[:pr_out_mol][9]), value(m[:pr_out_mol][10]),
        value(m[:preGHR_in_mol][1]), value(m[:preGHR_in_mol][2]), value(m[:
            preGHR_in_mol][3]), value(m[:preGHR_in_mol][4]), value(m[:
            preGHR_in_mol][5]),
        value(m[:preGHR_out_mol][1]), value(m[:preGHR_out_mol][2]), value(m[:
            preGHR_out_mol][3]), value(m[:preGHR_out_mol][4]), value(m[:
            preGHR_out_mol][5]),
        value(m[:ghr_in_mol][1]), value(m[:ghr_in_mol][2]), value(m[:ghr_in_mol]
            [3]), value(m[:ghr_in_mol][4]), value(m[:ghr_in_mol][5]),
        value(m[:ghr_out_mol][1]), value(m[:ghr_out_mol][2]), value(m[:
            ghr_out_mol][3]), value(m[:ghr_out_mol][4]), value(m[:ghr_out_mol][
            5]),
        value(m[:atr_in_mol][1]), value(m[:atr_in_mol][2]), value(m[:atr_in_mol]
            [3]), value(m[:atr_in_mol][4]), value(m[:atr_in_mol][5]),
        value(m[:atr_out_mol][1]), value(m[:atr_out_mol][2]), value(m[:
            atr_out_mol][3]), value(m[:atr_out_mol][4]), value(m[:atr_out_mol][
            5]),
        value(m[:postATR_in_mol][1]), value(m[:postATR_in_mol][2]), value(m[:
            postATR_in_mol][3]), value(m[:postATR_in_mol][4]), value(m[:
            postATR_in_mol][5]),
        value(m[:postATR_out_mol][1]), value(m[:postATR_out_mol][2]), value(m[:
            postATR_out_mol][3]), value(m[:postATR_out_mol][4]), value(m[:
            postATR_out_mol][5]),
        value(m[:itsr_in_mol][1]), value(m[:itsr_in_mol][2]), value(m[:
            itsr_in_mol][3]), value(m[:itsr_in_mol][4]), value(m[:itsr_in_mol][
            5]),
        value(m[:itsr_out_mol][1]), value(m[:itsr_out_mol][2]), value(m[:
            itsr_out_mol][3]), value(m[:itsr_out_mol][4]), value(m[:
```

```
            itsr_out_mol][5]),
        value(m[:preCond_in_mol][1]), value(m[:preCond_in_mol][2]), value(m[:
            preCond_in_mol][3]), value(m[:preCond_in_mol][4]), value(m[:
            preCond_in_mol][5]),
        value(m[:preCond_out_mol][1]), value(m[:preCond_out_mol][2]), value(m[:
            preCond_out_mol][3]), value(m[:preCond_out_mol][4]), value(m[:
            preCond_out_mol][5]),
        value(m[:cond_in_mol][1]), value(m[:cond_in_mol][2]), value(m[:
            cond_in_mol][3]), value(m[:cond_in_mol][4]), value(m[:cond_in_mol][
            5]),
        value(m[:cond_L]), value(m[:cond_liq_frac][1]), value(m[:cond_liq_frac][
            2]), value(m[:cond_liq_frac][3]), value(m[:cond_liq_frac][4]),
            value(m[:cond_liq_frac][5]),
        value(m[:cond_V]), value(m[:cond_vap_frac][1]), value(m[:cond_vap_frac][
            2]), value(m[:cond_vap_frac][3]), value(m[:cond_vap_frac][4]),
            value(m[:cond_vap_frac][5]),
        value(m[:psa_in_mol][1]), value(m[:psa_in_mol][2]), value(m[:psa_in_mol]
            [3]), value(m[:psa_in_mol][4]), value(m[:psa_in_mol][5]),
        value(m[:psa_outProduct_mol][1]), value(m[:psa_outProduct_mol][2]),
            value(m[:psa_outProduct_mol][3]), value(m[:psa_outProduct_mol][4]),
             value(m[:psa_outProduct_mol][5]),
        value(m[:psa_outPurge_mol][1]), value(m[:psa_outPurge_mol][2]), value(m[
            :psa_outPurge_mol][3]), value(m[:psa_outPurge_mol][4]), value(m[:
            psa_outPurge_mol][5]),
        value(m[:mix_in_T]), value(m[:mix_out_T]), value(m[:H2O_T]), value(m[:
            prePR_in_T]), value(m[:prePR_out_T]), value(m[:preGHR_in_T]), value
            (m[:preGHR_out_T]),
        value(m[:ghr_in_T]), value(m[:ghr_out_T]), value(m[:atr_in_T]), value(m[
            :postATR_in_T]), value(m[:postATR_out_T]),
        value(m[:preCond_out_T]), value(m[:cond_in_T]), value(m[:cond_L_T]),
            value(m[:cond_V_T]), value(m[:psa_in_T]), value(m[:psa_outProduct_T
            ]), value(m[:psa_outPurge_T]),
        value(m[:prePR_Q]), value(m[:preGHR_Q]), value(m[:ghr_Q]), value(m[:
            postATR_Q]), value(m[:itsr_Q]), value(m[:preCond_Q]),
        value(m[:H2Ostream]), value(m[:F_H2]), value(m[:F_H2_heat]), value(m[:
            F_NG]), value(m[:F_NG_heat]), value(m[:F_fluegas]), value(m[:F_inj]
            )
    ]

    for i in eachindex(disturbed)
        disturbed[i] = (disturbed[i]-nominal[i])/delta_d;
    end
    return disturbed
end

dydd1_plus = G_yd(nominal_values, 1, eps)
dydd2_plus = G_yd(nominal_values, 2, eps)
dydd3_plus = G_yd(nominal_values, 3, eps)

function printG_yd()
    return DataFrame(Variable = variable_name,
                     Nominal = nominal_values,
                     ∂y∂d_1_plus = dydd1_plus,
                     ∂y∂d_2_plus = dydd2_plus,
                      ∂y∂d_3_plus = dydd3_plus)
end
function matrix_Gyd(nominal_values, eps)
    matrix = zeros(3,184);
    for i in 1:3
        if i == 1
            vector = G_yd(nominal_values, 1, eps)
        elseif i == 2
            vector = G_yd(nominal_values, 2, eps)
        else
            vector = G_yd(nominal_values, 3, eps)
        end
        matrix[i,:] = vector'
    end
    return matrix
end

#XLSX.writetable("data/G_yd.xlsx", printG_yd());

#G_yd_table = printG_yd()
#println("G_yd"); show(G_yd_table, allrows=true);
```

B.23 gain_u.jl

The function G_y() calculates the output gain with respect to disturbance change for each disturbance perturbation, which is assembled to a matrix in the function matrix_Gy and exported to an Excel sheet. printG_y() prints out the G_y matrix into a readable table in the output terminal.

```
variable_name = [
    "mix_in_mol[1]", "mix_in_mol[2]", "mix_in_mol[3]", "mix_in_mol[4]", "
        mix_in_mol[5]",
    "mix_in_mol[6]", "mix_in_mol[7]", "mix_in_mol[8]", "mix_in_mol[9]", "
        mix_in_mol[10]",
    "mix_out_mol[1]", "mix_out_mol[2]", "mix_out_mol[3]", "mix_out_mol[4]", "
        mix_out_mol[5]",
    "mix_out_mol[6]", "mix_out_mol[7]", "mix_out_mol[8]", "mix_out_mol[9]", "
        mix_out_mol[10]",
    "prePR_in_mol[1]", "prePR_in_mol[2]", "prePR_in_mol[3]", "prePR_in_mol[4]",
        "prePR_in_mol[5]",
    "prePR_in_mol[6]", "prePR_in_mol[7]", "prePR_in_mol[8]", "prePR_in_mol[9]",
        "prePR_in_mol[10]",
    "prePR_out_mol[1]", "prePR_out_mol[2]", "prePR_out_mol[3]", "prePR_out_mol
        [4]", "prePR_out_mol[5]",
    "prePR_out_mol[6]", "prePR_out_mol[7]", "prePR_out_mol[8]", "prePR_out_mol
        [9]", "prePR_out_mol[10]",
    "pr_in_mol[1]", "pr_in_mol[2]", "pr_in_mol[3]", "pr_in_mol[4]", "pr_in_mol
        [5]",
    "pr_in_mol[6]", "pr_in_mol[7]", "pr_in_mol[8]", "pr_in_mol[9]", "pr_in_mol
        [10]",
    "pr_out_mol[1]", "pr_out_mol[2]", "pr_out_mol[3]", "pr_out_mol[4]", "
        pr_out_mol[5]",
    "pr_out_mol[6]", "pr_out_mol[7]", "pr_out_mol[8]", "pr_out_mol[9]", "
        pr_out_mol[10]",
    "preGHR_in_mol[1]", "preGHR_in_mol[2]", "preGHR_in_mol[3]", "preGHR_in_mol
        [4]", "preGHR_in_mol[5]",
    "preGHR_out_mol[1]", "preGHR_out_mol[2]", "preGHR_out_mol[3]", "
        preGHR_out_mol[4]", "preGHR_out_mol[5]",
    "ghr_in_mol[1]", "ghr_in_mol[2]", "ghr_in_mol[3]", "ghr_in_mol[4]", "
        ghr_in_mol[5]",
    "ghr_out_mol[1]", "ghr_out_mol[2]", "ghr_out_mol[3]", "ghr_out_mol[4]", "
        ghr_out_mol[5]",
    "atr_in_mol[1]", "atr_in_mol[2]", "atr_in_mol[3]", "atr_in_mol[4]", "
        atr_in_mol[5]",
    "atr_out_mol[1]", "atr_out_mol[2]", "atr_out_mol[3]", "atr_out_mol[4]", "
        atr_out_mol[5]",
    "postATR_in_mol[1]", "postATR_in_mol[2]", "postATR_in_mol[3]", "
        postATR_in_mol[4]", "postATR_in_mol[5]",
    "postATR_out_mol[1]", "postATR_out_mol[2]", "postATR_out_mol[3]", "
        postATR_out_mol[4]", "postATR_out_mol[5]",
    "itsr_in_mol[1]", "itsr_in_mol[2]", "itsr_in_mol[3]", "itsr_in_mol[4]", "
        itsr_in_mol[5]",
    "itsr_out_mol[1]", "itsr_out_mol[2]", "itsr_out_mol[3]", "itsr_out_mol[4]",
        "itsr_out_mol[5]",
    "preCond_in_mol[1]", "preCond_in_mol[2]", "preCond_in_mol[3]", "
        preCond_in_mol[4]", "preCond_in_mol[5]",
    "preCond_out_mol[1]", "preCond_out_mol[2]", "preCond_out_mol[3]", "
        preCond_out_mol[4]", "preCond_out_mol[5]",
    "cond_in_mol[1]", "cond_in_mol[2]", "cond_in_mol[3]", "cond_in_mol[4]", "
        cond_in_mol[5]",
    "cond_L", "cond_liq_frac[1]", "cond_liq_frac[2]", "cond_liq_frac[3]", "
        cond_liq_frac[4]", "cond_liq_frac[5]",
    "cond_V", "cond_vap_frac[1]", "cond_vap_frac[2]", "cond_vap_frac[3]", "
        cond_vap_frac[4]", "cond_vap_frac[5]",
    "psa_in_mol[1]", "psa_in_mol[2]", "psa_in_mol[3]", "psa_in_mol[4]", "
        psa_in_mol[5]",
    "psa_outProduct_mol[1]", "psa_outProduct_mol[2]", "psa_outProduct_mol[3]", "
        psa_outProduct_mol[4]", "psa_outProduct_mol[5]",
    "psa_outPurge_mol[1]", "psa_outPurge_mol[2]", "psa_outPurge_mol[3]", "
        psa_outPurge_mol[4]", "psa_outPurge_mol[5]",
    "mix_in_T", "mix_out_T", "H2O_T", "prePR_in_T", "prePR_out_T", "preGHR_in_T"
        , "preGHR_out_T",
    "ghr_in_T", "ghr_out_T", "atr_in_T", "postATR_in_T", "postATR_out_T",
    "preCond_out_T", "cond_in_T", "cond_L_T", "cond_V_T", "psa_in_T", "
        psa_outProduct_T", "psa_outPurge_T",
    "prePR_Q", "preGHR_Q", "ghr_Q", "postATR_Q", "itsr_Q", "preCond_Q",
    "H2Ostream", "F_H2", "F_H2_heat", "F_NG", "F_NG_heat", "F_fluegas", "F_inj"
]
# Variables that are removed = S/C, nO2, T_prePR, T_PR, T_ATR, T_postATR, T_ITSR
    , T_cond

using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables, XLSX
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")
include("nominal_case.jl")

eps = 1e-5
nominal_values, nominal_J = nominal();

# 1: u1+h, n_O2
# 2: u2+h, T_prePR
# 3: u3+h, T_ATR
# -1: u1-h, n_O2
# -2: u2-h, T_prePR
# -3: u3-h, T_ATR
function G_y(nominal, option, eps)

    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-6, "constr_viol_tol" => 1e-8,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
```

```
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    if option == 1
        delta_u = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805*(1+eps) == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 2
        delta_u = eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283*(1+eps) == 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 3
        delta_u = eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818*(1+eps) == 0);
    elseif option == -1
        delta_u = -eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805*(1-eps) == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -2
        delta_u = -eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283*(1-eps) == 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -3
        delta_u = -eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818*(1-eps) == 0);
    else
        print("Option not valid")
    end
    #################### Connection constraints ######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - par.init.init_stream + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);

    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*par.P_H2 - compWsum*par.elCost/1000);

    optimize!(m)

    input_change = [
        value(m[:mix_in_mol][1]), value(m[:mix_in_mol][2]), value(m[:mix_in_mol]
            [3]), value(m[:mix_in_mol][4]), value(m[:mix_in_mol][5]),
        value(m[:mix_in_mol][6]), value(m[:mix_in_mol][7]), value(m[:mix_in_mol]
            [8]), value(m[:mix_in_mol][9]), value(m[:mix_in_mol][10]),
        value(m[:mix_out_mol][1]), value(m[:mix_out_mol][2]), value(m[:
            mix_out_mol][3]), value(m[:mix_out_mol][4]), value(m[:mix_out_mol][
            5]),
        value(m[:mix_out_mol][6]), value(m[:mix_out_mol][7]), value(m[:
            mix_out_mol][8]), value(m[:mix_out_mol][9]), value(m[:mix_out_mol][
            10]),
        value(m[:prePR_in_mol][1]), value(m[:prePR_in_mol][2]), value(m[:
            prePR_in_mol][3]), value(m[:prePR_in_mol][4]), value(m[:
            prePR_in_mol][5]),
        value(m[:prePR_in_mol][6]), value(m[:prePR_in_mol][7]), value(m[:
            prePR_in_mol][8]), value(m[:prePR_in_mol][9]), value(m[:
            prePR_in_mol][10]),
        value(m[:prePR_out_mol][1]), value(m[:prePR_out_mol][2]), value(m[:
            prePR_out_mol][3]), value(m[:prePR_out_mol][4]), value(m[:
            prePR_out_mol][5]),
        value(m[:prePR_out_mol][6]), value(m[:prePR_out_mol][7]), value(m[:
            prePR_out_mol][8]), value(m[:prePR_out_mol][9]), value(m[:
            prePR_out_mol][10]),
        value(m[:pr_in_mol][1]), value(m[:pr_in_mol][2]), value(m[:pr_in_mol][3]
            ), value(m[:pr_in_mol][4]), value(m[:pr_in_mol][5]),
        value(m[:pr_in_mol][6]), value(m[:pr_in_mol][7]), value(m[:pr_in_mol][8]
            ), value(m[:pr_in_mol][9]), value(m[:pr_in_mol][10]),
        value(m[:pr_out_mol][1]), value(m[:pr_out_mol][2]), value(m[:pr_out_mol]
            [3]), value(m[:pr_out_mol][4]), value(m[:pr_out_mol][5]),
        value(m[:pr_out_mol][6]), value(m[:pr_out_mol][7]), value(m[:pr_out_mol]
            [8]), value(m[:pr_out_mol][9]), value(m[:pr_out_mol][10]),
        value(m[:preGHR_in_mol][1]), value(m[:preGHR_in_mol][2]), value(m[:
            preGHR_in_mol][3]), value(m[:preGHR_in_mol][4]), value(m[:
            preGHR_in_mol][5]),
        value(m[:preGHR_out_mol][1]), value(m[:preGHR_out_mol][2]), value(m[:
            preGHR_out_mol][3]), value(m[:preGHR_out_mol][4]), value(m[:
            preGHR_out_mol][5]),
        value(m[:ghr_in_mol][1]), value(m[:ghr_in_mol][2]), value(m[:ghr_in_mol]
            [3]), value(m[:ghr_in_mol][4]), value(m[:ghr_in_mol][5]),
        value(m[:ghr_out_mol][1]), value(m[:ghr_out_mol][2]), value(m[:
            ghr_out_mol][3]), value(m[:ghr_out_mol][4]), value(m[:ghr_out_mol][
            5]),
        value(m[:atr_in_mol][1]), value(m[:atr_in_mol][2]), value(m[:atr_in_mol]
            [3]), value(m[:atr_in_mol][4]), value(m[:atr_in_mol][5]),
        value(m[:atr_out_mol][1]), value(m[:atr_out_mol][2]), value(m[:
            atr_out_mol][3]), value(m[:atr_out_mol][4]), value(m[:atr_out_mol][
            5]),
        value(m[:postATR_in_mol][1]), value(m[:postATR_in_mol][2]), value(m[:
            postATR_in_mol][3]), value(m[:postATR_in_mol][4]), value(m[:
            postATR_in_mol][5]),
        value(m[:postATR_out_mol][1]), value(m[:postATR_out_mol][2]), value(m[:
            postATR_out_mol][3]), value(m[:postATR_out_mol][4]), value(m[:
            postATR_out_mol][5]),
        value(m[:itsr_in_mol][1]), value(m[:itsr_in_mol][2]), value(m[:
            itsr_in_mol][3]), value(m[:itsr_in_mol][4]), value(m[:itsr_in_mol][
            5]),
        value(m[:itsr_out_mol][1]), value(m[:itsr_out_mol][2]), value(m[:
            itsr_out_mol][3]), value(m[:itsr_out_mol][4]), value(m[:
            itsr_out_mol][5]),
        value(m[:preCond_in_mol][1]), value(m[:preCond_in_mol][2]), value(m[:
```

```julia
            preCond_in_mol][3]), value(m[:preCond_in_mol][4]), value(m[:
            preCond_in_mol][5]),
        value(m[:preCond_out_mol][1]), value(m[:preCond_out_mol][2]), value(m[:
            preCond_out_mol][3]), value(m[:preCond_out_mol][4]), value(m[:
            preCond_out_mol][5]),
        value(m[:cond_in_mol][1]), value(m[:cond_in_mol][2]), value(m[:
            cond_in_mol][3]), value(m[:cond_in_mol][4]), value(m[:cond_in_mol][
            5]),
        value(m[:cond_L]), value(m[:cond_liq_frac][1]), value(m[:cond_liq_frac][
            2]), value(m[:cond_liq_frac][3]), value(m[:cond_liq_frac][4]),
            value(m[:cond_liq_frac][5]),
        value(m[:cond_V]), value(m[:cond_vap_frac][1]), value(m[:cond_vap_frac][
            2]), value(m[:cond_vap_frac][3]), value(m[:cond_vap_frac][4]),
            value(m[:cond_vap_frac][5]),
        value(m[:psa_in_mol][1]), value(m[:psa_in_mol][2]), value(m[:psa_in_mol]
            [3]), value(m[:psa_in_mol][4]), value(m[:psa_in_mol][5]),
        value(m[:psa_outProduct_mol][1]), value(m[:psa_outProduct_mol][2]),
            value(m[:psa_outProduct_mol][3]), value(m[:psa_outProduct_mol][4]),
             value(m[:psa_outProduct_mol][5]),
        value(m[:psa_outPurge_mol][1]), value(m[:psa_outPurge_mol][2]), value(m[
            :psa_outPurge_mol][3]), value(m[:psa_outPurge_mol][4]), value(m[:
            psa_outPurge_mol][5]),
        value(m[:mix_in_T]), value(m[:mix_out_T]), value(m[:H2O_T]), value(m[:
            prePR_in_T]), value(m[:prePR_out_T]), value(m[:preGHR_in_T]), value
            (m[:preGHR_out_T]),
        value(m[:ghr_in_T]), value(m[:ghr_out_T]), value(m[:atr_in_T]), value(m[
            :postATR_in_T]), value(m[:postATR_out_T]),
        value(m[:preCond_out_T]), value(m[:cond_in_T]), value(m[:cond_L_T]),
            value(m[:cond_V_T]), value(m[:psa_in_T]), value(m[:psa_outProduct_T
            ]), value(m[:psa_outPurge_T]),
        value(m[:prePR_Q]), value(m[:preGHR_Q]), value(m[:ghr_Q]), value(m[:
            postATR_Q]), value(m[:itsr_Q]), value(m[:preCond_Q]),
        value(m[:H2Ostream]), value(m[:F_H2]), value(m[:F_H2_heat]), value(m[:
            F_NG]), value(m[:F_NG_heat]), value(m[:F_fluegas]), value(m[:F_inj]
            )
    ]
    nominal= nominal
    for i in eachindex(input_change)
        input_change[i] = (input_change[i]-nominal[i])/delta_u;
    end
    return input_change
end

dydu1_plus = G_y(nominal_values, 1, eps)
dydu2_plus = G_y(nominal_values, 2, eps)
dydu3_plus = G_y(nominal_values, 3, eps)
dydu1_minus = G_y(nominal_values, -1, eps)
dydu2_minus = G_y(nominal_values, -2, eps)
dydu3_minus = G_y(nominal_values, -3, eps)

function printG_y()
    return DataFrame(Variable = variable_name,
                     Nominal = nominal_values,
                     ∂y∂u_1 = dydu1_plus,
                     ∂y∂u_2 = dydu2_plus,
                     ∂y∂u_3 = dydu3_plus)
end

function matrix_Gy(nominal_values, eps)
    matrix = zeros(3,184);
    for i in 1:3
        if i == 1
            vector = G_y(nominal_values, 1, eps)
        elseif i == 2
            vector = G_y(nominal_values, 2, eps)
        else
            vector = G_y(nominal_values, 3, eps)
        end
        matrix[i,:] = vector'
    end
    return matrix
end

#XLSX.writetable("data/G_y.xlsx", printG_y());

#G_y_table = printG_y()
#println("G_y"); show(G_y_table, allrows=true);
```

B.24 J_ud.jl

The function J_ud() uses the functions finite_diff_1() and finite_diff_2() from the finite_diff() file to calculate each perturbation to approximate each double partial derivate element in the Hessian matrix, J_ud(), which is the Hessian matrix of the cost function with respect to inputs and disturbances. The function J_ud_xlsx() exports the resulting Hessian matrix to an Excel sheet.

```
include("finite_diff.jl")
include("nominal_case.jl")

# Options for finite diff (eps = 1e-5)
# 1: u1+h, n_O2
# 2: u2+h, T_prePR
# 3: u3+h, T_ATR
# 4: d1+k, NG flow
# 5: d2+k, P_el
# 6: d3+k, P_H2
# -1: u1-h, n_O2
# -2: u2-h, T_prePR
# -3: u3-h, T_ATR
# -4: d1-k, NG flow
# -5: d2-k, P_el
# -6: d3-k, P_H2
```

```
nominal_states, nom_f = nominal();
eps = 1e-5
function J_ud(eps)
    J_ud = zeros(3,3);
    for i=1:3
        for j=4:6
            f1, h, k = finite_diff_2(i,j,eps); # f(x+h, y+k)
            f2, h, k = finite_diff_1(i,eps) # f(x+h, y)
            f3, h, k = finite_diff_1(-i,eps) # f(x-h, y)
            f4, h, k = finite_diff_1(j,eps) # f(x, y+k)
            f5, h, k = finite_diff_1(-j,eps) # f(x, y-k)
            f6, h, k = finite_diff_2(-i, -j, eps); # f(x-h, y-k)
            f_xy = (f1 - f2 - f3 - f4 - f5 + f6 + 2*nom_f)/(2*h*k)
            J_ud[i,j-3] = f_xy
        end
    end
    return J_ud
end

function J_ud_xlsx()
    J_ud_matrix = J_ud(1e-2)';
    df = DataFrame(col1 = J_ud_matrix[1,:],
                col2 = J_ud_matrix[2,:],
                col3 = J_ud_matrix[3,:]);
    XLSX.writetable("data/Jud.xlsx", df);
end
```

B.25 J_uu.jl

The function J_uu() uses the functions finite_diff_1() and finite_diff_2() from the finite_diff() file to calculate each perturbation to approximate each double partial derivate element in the Hessian matrix, J_uu(), which is the Hessian matrix of the cost function with respect to inputs and disturbances. The function J_uu_xlsx() exports the resulting Hessian matrix to an Excel sheet.

```
include("finite_diff.jl")
include("nominal_case.jl")

# Options for finite diff (eps = 1e-5)
# 1: u1+h, n_O2
# 2: u2+h, T_prePR
# 3: u3+h, T_ATR
# 4: d1+k, NG flow
# 5: d2+k, P_el
# 6: d3+k, P_H2
# -1: u1-h, n_O2
# -2: u2-h, T_prePR
# -3: u3-h, T_ATR
# -4: d1-k, NG flow
# -5: d2-k, P_el
# -6: d3-k, P_H2

nominal_states, nom_f = nominal();
eps = 1e-2
function J_uu(eps)
    J_uu = zeros(3,3);
    for i=1:3
        for j=1:3
            if i==j
                f1, h, k = finite_diff_1(i,eps) # f(x+h, y)
                f2, h, k = finite_diff_1(-i,eps) # f(x-h, y)
                f_xx = (f1 - 2*nom_f + f2)/(eps^2) # f(x+h,y) - 2f(x,y) + f(x-h,
                    y) / h^2
                J_uu[i,j] = f_xx
            else
                f1, h, k = finite_diff_2(i,j,eps); # f(x+h, y+k)
                f2, h, k = finite_diff_1(i,eps) # f(x+h, y)
                f3, h, k = finite_diff_1(-i,eps) # f(x-h, y)
                f4, h, k = finite_diff_1(j,eps) # f(x, y+k)
                f5, h, k = finite_diff_1(-j,eps) # f(x, y-k)
                f6, h, k = finite_diff_2(-i, -j, eps); # f(x-h, y-k)
                f_xy = (f1 - f2 - f3 - f4 - f5 + f6 + 2*nom_f)/(2*eps*eps)
                J_uu[i,j] = f_xy
            end
        end
    end
    return J_uu
end

function J_uu_xlsx()
J_uu_matrix = J_uu(1e-2)';
df = DataFrame(col1 = J_uu_matrix[1,:],
                col2 = J_uu_matrix[2,:],
                col3 = J_uu_matrix[3,:]);
XLSX.writetable("data/Juu.xlsx", df);
end
```

B.26 loss.jl

The Loss() function calculates the loss in optimality by calculating the difference between the objective value when all inputs are used for control in a control strategy and the optimal cases where the inputs are free when different disturbance changes affect the system. The function nullspacePrint() calculates the nullspace matrix of the optimal sensitivity matrix chosen from a measurement subset.

```
using LinearAlgebra
include("nominal_case.jl")
variable_name = [
    "mix_in_mol[1]", "mix_in_mol[2]", "mix_in_mol[3]", "mix_in_mol[4]", "
        mix_in_mol[5]",
    "mix_in_mol[6]", "mix_in_mol[7]", "mix_in_mol[8]", "mix_in_mol[9]", "
        mix_in_mol[10]",
    "mix_out_mol[1]", "mix_out_mol[2]", "mix_out_mol[3]", "mix_out_mol[4]", "
        mix_out_mol[5]",
    "mix_out_mol[6]", "mix_out_mol[7]", "mix_out_mol[8]", "mix_out_mol[9]", "
        mix_out_mol[10]",
    "prePR_in_mol[1]", "prePR_in_mol[2]", "prePR_in_mol[3]", "prePR_in_mol[4]",
        "prePR_in_mol[5]",
    "prePR_in_mol[6]", "prePR_in_mol[7]", "prePR_in_mol[8]", "prePR_in_mol[9]",
        "prePR_in_mol[10]",
    "prePR_out_mol[1]", "prePR_out_mol[2]", "prePR_out_mol[3]", "prePR_out_mol
        [4]", "prePR_out_mol[5]",
    "prePR_out_mol[6]", "prePR_out_mol[7]", "prePR_out_mol[8]", "prePR_out_mol
        [9]", "prePR_out_mol[10]",
    "pr_in_mol[1]", "pr_in_mol[2]", "pr_in_mol[3]", "pr_in_mol[4]", "pr_in_mol
        [5]",
    "pr_in_mol[6]", "pr_in_mol[7]", "pr_in_mol[8]", "pr_in_mol[9]", "pr_in_mol
        [10]",
    "pr_out_mol[1]", "pr_out_mol[2]", "pr_out_mol[3]", "pr_out_mol[4]", "
        pr_out_mol[5]",
    "pr_out_mol[6]", "pr_out_mol[7]", "pr_out_mol[8]", "pr_out_mol[9]", "
        pr_out_mol[10]",
    "preGHR_in_mol[1]", "preGHR_in_mol[2]", "preGHR_in_mol[3]", "preGHR_in_mol
        [4]", "preGHR_in_mol[5]",
    "preGHR_out_mol[1]", "preGHR_out_mol[2]", "preGHR_out_mol[3]", "
        preGHR_out_mol[4]", "preGHR_out_mol[5]",
    "ghr_in_mol[1]", "ghr_in_mol[2]", "ghr_in_mol[3]", "ghr_in_mol[4]", "
        ghr_in_mol[5]",
    "ghr_out_mol[1]", "ghr_out_mol[2]", "ghr_out_mol[3]", "ghr_out_mol[4]", "
        ghr_out_mol[5]",
    "atr_in_mol[1]", "atr_in_mol[2]", "atr_in_mol[3]", "atr_in_mol[4]", "
        atr_in_mol[5]",
    "atr_out_mol[1]", "atr_out_mol[2]", "atr_out_mol[3]", "atr_out_mol[4]", "
        atr_out_mol[5]",
    "postATR_in_mol[1]", "postATR_in_mol[2]", "postATR_in_mol[3]", "
        postATR_in_mol[4]", "postATR_in_mol[5]",
    "postATR_out_mol[1]", "postATR_out_mol[2]", "postATR_out_mol[3]", "
        postATR_out_mol[4]", "postATR_out_mol[5]",
    "itsr_in_mol[1]", "itsr_in_mol[2]", "itsr_in_mol[3]", "itsr_in_mol[4]", "
        itsr_in_mol[5]",
    "itsr_out_mol[1]", "itsr_out_mol[2]", "itsr_out_mol[3]", "itsr_out_mol[4]",
        "itsr_out_mol[5]",
    "preCond_in_mol[1]", "preCond_in_mol[2]", "preCond_in_mol[3]", "
        preCond_in_mol[4]", "preCond_in_mol[5]",
    "preCond_out_mol[1]", "preCond_out_mol[2]", "preCond_out_mol[3]", "
        preCond_out_mol[4]", "preCond_out_mol[5]",
    "cond_in_mol[1]", "cond_in_mol[2]", "cond_in_mol[3]", "cond_in_mol[4]", "
        cond_in_mol[5]",
    "cond_L", "cond_liq_frac[1]", "cond_liq_frac[2]", "cond_liq_frac[3]", "
        cond_liq_frac[4]", "cond_liq_frac[5]",
    "cond_V", "cond_vap_frac[1]", "cond_vap_frac[2]", "cond_vap_frac[3]", "
        cond_vap_frac[4]", "cond_vap_frac[5]",
    "psa_in_mol[1]", "psa_in_mol[2]", "psa_in_mol[3]", "psa_in_mol[4]", "
        psa_in_mol[5]",
    "psa_outProduct_mol[1]", "psa_outProduct_mol[2]", "psa_outProduct_mol[3]", "
        psa_outProduct_mol[4]", "psa_outProduct_mol[5]",
    "psa_outPurge_mol[1]", "psa_outPurge_mol[2]", "psa_outPurge_mol[3]", "
        psa_outPurge_mol[4]", "psa_outPurge_mol[5]",
    "mix_in_T", "mix_out_T", "H2O_T", "prePR_in_T", "prePR_out_T", "preGHR_in_T"
        , "preGHR_out_T",
    "ghr_in_T", "ghr_out_T", "atr_in_T", "postATR_in_T", "postATR_out_T",
    "preCond_out_T", "cond_in_T", "cond_L_T", "cond_V_T", "psa_in_T", "
        psa_outProduct_T", "psa_outPurge_T",
    "prePR_Q", "preGHR_Q", "ghr_Q", "postATR_Q", "itsr_Q", "preCond_Q",
    "H2Ostream", "F_H2", "F_H2_heat", "F_NG", "F_NG_heat", "F_fluegas", "F_inj"
]
optimal_J = optJ_func();
y_nom, J_nom = nominal();
function Loss()
    __par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-9, "constr_viol_tol" => 1e-5,
             "print_level" => 0);
    m = Model(optimizer);

    eps = 0.1

    par = deepcopy(__par)
    #par.init.init_stream  = par.init.init_stream +5; # +d1
    #3par.elCost = par.elCost*(1+eps); # +d2
    #par.P_H2 = par.P_H2*(1+eps); # +d3
    #par.init.init_stream  = par.init.init_stream -5; # -d1
    #par.elCost = par.elCost*(1-eps); # -d2
    #par.P_H2 = par.P_H2*(1-eps); # -d3


    d1 = par.init.init_stream;
    d2 = par.elCost;
    d3 = par.P_H2;

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    # Constraints for setting the active constraints to their boundary value
    @constraint(m, m[:SC_ratio] - 5.0 == 0);
    @constraint(m, m[:pr_out_T] - 609.2 == 0);
    @constraint(m, m[:itsr_out_T] - 473.0  == 0);
    @constraint(m, m[:preCond_out_T] - 293.0 == 0);
    @constraint(m, m[:postATR_out_T] - 634.2 == 0);

    # u = unom case, comment out the three constraints when implementing
        advanced control
    @constraint(m, m[:nO2] - 79.29706225438805 == 0);
    @constraint(m, m[:pr_in_T] - 644.5953165006283 == 0);
    @constraint(m, m[:atr_out_T] - 1291.817465833818 == 0);

    # Disturb your disturbances here
    # Choose your H matrix
    # H1 Exact Local Method
    #@constraint(m, m[:pr_out_mol][3] - y_nom[53] == 0);
    #@constraint(m, m[:psa_outPurge_mol][1] - y_nom[148] == 0);
    #@constraint(m, m[:prePR_out_T] - y_nom[157] == 0);

    # H2 Nullspace method
    H2 = [-0.010396358480702392 -0.9990843821558919 -0.041500759782440846;
    -0.9999454720656192 0.010428244690954458 -0.0005519131554076154;
     0.0009841878917459415 0.041492758944720855 -0.9991383199183929];
    #@NLconstraint(m, H2[1,1]*(m[:pr_out_mol][3] - y_nom[53]) + H2[1,2]*(m[:
        psa_outPurge_mol][1] - y_nom[148]) + H2[1,3]*(m[:prePR_out_T] - y_nom
        [157]) == 0);
    #@NLconstraint(m, H2[2,1]*(m[:pr_out_mol][3] - y_nom[53]) + H2[2,2]*(m[:
        psa_outPurge_mol][1] - y_nom[148]) + H2[2,3]*(m[:prePR_out_T] - y_nom
        [157]) == 0);
    #@NLconstraint(m, H2[3,1]*(m[:pr_out_mol][3] - y_nom[53]) + H2[3,2]*(m[:
        psa_outPurge_mol][1] - y_nom[148]) + H2[3,3]*(m[:prePR_out_T] - y_nom
        [157]) == 0);

    # H3 Good Engineering Decision
    #@constraint(m, m[:ghr_out_T] - y_nom[161] == 0);
    #@constraint(m, m[:itsr_out_mol][5] - y_nom[110] == 0);
    #@constraint(m, m[:pr_out_mol][1] - y_nom[51] == 0);


    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints ######################
    for i = 1:10
        @constraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @constraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @constraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @constraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @constraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @constraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @constraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @constraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @constraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @constraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] == 0)
    end
    ################# Same for the temperature
        ##################################
    @constraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @constraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @constraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @constraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @constraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @constraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @constraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @constraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @constraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @constraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);



    ############## Initial values ##########################################
    for i = 1:10
        @constraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0);
    end
    @constraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @constraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @constraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @constraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    #@NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @constraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @constraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @constraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @constraint(m, m[:F_inj] - m[:F_fluegas] - sum(m[:psa_outPurge_mol][i] for i
         = 1:5) == 0);
    @constraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016 -
         sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[i]
         for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
```

```julia
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);

    # Checking where in the system the constraints are violated.
    #=
    for x in all_variables(m)
        if has_upper_bound(x)
            JuMP.delete_upper_bound(x)
        end
        if has_lower_bound(x)
           JuMP.delete_lower_bound(x)
        end
    end
    =#

    # Optimize
    optimize!(m)
    # Calculate loss
    L = objective_value(m) - optimal_J;
    streamdf, otherdf, massdf, compositiondf = printTable(m);
    println("Stream table"); show(streamdf, allrows=true);
    println("\n\nOther variables"); show(otherdf, allrows=true);
    println("\n\nMass table"); show(massdf, allrows=true);
    #println("\n\nCompostion table"); show(compositiondf, allrows=true);
    println("");
    println("The new objective value is: ", objective_value(m));
    println("While the optimal objective value is: ", optimal_J);
    if termination_status(m) == LOCALLY_SOLVED || termination_status(m) ==
        OPTIMAL || termination_status(m) == ALMOST_LOCALLY_SOLVED
        println(termination_status(m))
        return round(L; digits = 4)
    else
        println(termination_status(m))
        return round(L; digits = 4)
    end
end

@show(Loss())
function nullspacePrint(data)
    df = DataFrame(XLSX.readtable(data, "Sheet1"));
    A = Matrix(df);
    A = A[:, end-2:end];
    A = A[[53,68, 148, 157], :];
    F = convert(Matrix{Float64}, A);
    return nullspace(F, rtol=1);
end
#@show(nullspacePrint("data/F.xlsx"));
```

B.27 nominal_case.jl

The nominal() function calculates and returns the nominal state values and the nominal objective function, such that it can be used for other calculations, such as gain matrices and the Hessian matrices. The optJ_func() calculates and returns the optimal state values and the optimal objective function, for calculating e.g. loss in optimality.

```julia
using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")

const MOI = MathOptInterface
C = 12.01;
H = 1.008;
O = 16;
Base.@kwdef mutable struct _par
    init::init_par = init_par();
    mix::mix_par=mix_par();
    prePR::prePR_par=prePR_par();
    pr::pr_par = pr_par();
    preGHR::preGHR_par = preGHR_par();
    ghr::ghr_par = ghr_par();
    atr::atr_par = atr_par();
    postATR::postATR_par = postATR_par();
    itsr::itsr_par = itsr_par();
    preCond::preCond_par = preCond_par();
    cond::cond_par = cond_par();
    psa::psa_par = psa_par();
    hconst = heavy_const;
    smr_const = smr_const;
    wgsr_const = wgsr_const;
    HHV_H2::Float64 = 141.7*1000;
    HHV_NG::Vector = [55.5,0.0,141.7,0.0,0.0,51.9,50.4,49.1,49.1,48.6]*1000; #
        CH4, H2O, H2, CO, CO2, C2H6, C3H8, n-C4H10, i-C4H10, C5H12
    molarMass::Vector = [C+H*4, H*2+O, H*2, C+O, C+O*2, C*2+H*6, C*3+H*8, C*4+H*
        10, C*4+H*10, C*5+H*12];
    P_H2::Float64 = 3.347; #[\$/kmol]
    P_inj::Float64 = 9.650; # [\$/ton CO2]
    R::Float64 = 8.314;
    elCost::Float64 = 0.14; # [\$/Kwh]
end

eps = 0.1

# return_list should be either d1, d2 or d3,
function nominal()
    __par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-9, "constr_viol_tol" => 1e-5,
             "print_level" => 0)
    m = Model(optimizer);
    d1 = par.init.init_stream;
    d2 = par.elCost;
    d3 = par.P_H2;
    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints ######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ##################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);
    optimize!(m)
    nominal_J = objective_value(m)
    nominal_values = [
        value(m[:mix_in_mol][1]), value(m[:mix_in_mol][2]), value(m[:mix_in_mol]
            [3]), value(m[:mix_in_mol][4]), value(m[:mix_in_mol][5]),
        value(m[:mix_in_mol][6]), value(m[:mix_in_mol][7]), value(m[:mix_in_mol]
            [8]), value(m[:mix_in_mol][9]), value(m[:mix_in_mol][10]),
        value(m[:mix_out_mol][1]), value(m[:mix_out_mol][2]), value(m[:
            mix_out_mol][3]), value(m[:mix_out_mol][4]), value(m[:mix_out_mol][
            5]),
        value(m[:mix_out_mol][6]), value(m[:mix_out_mol][7]), value(m[:
            mix_out_mol][8]), value(m[:mix_out_mol][9]), value(m[:mix_out_mol][
            10]),
        value(m[:prePR_in_mol][1]), value(m[:prePR_in_mol][2]), value(m[:
            prePR_in_mol][3]), value(m[:prePR_in_mol][4]), value(m[:
            prePR_in_mol][5]),
        value(m[:prePR_in_mol][6]), value(m[:prePR_in_mol][7]), value(m[:
            prePR_in_mol][8]), value(m[:prePR_in_mol][9]), value(m[:
            prePR_in_mol][10]),
        value(m[:prePR_out_mol][1]), value(m[:prePR_out_mol][2]), value(m[:
            prePR_out_mol][3]), value(m[:prePR_out_mol][4]), value(m[:
            prePR_out_mol][5]),
        value(m[:prePR_out_mol][6]), value(m[:prePR_out_mol][7]), value(m[:
            prePR_out_mol][8]), value(m[:prePR_out_mol][9]), value(m[:
            prePR_out_mol][10]),
        value(m[:pr_in_mol][1]), value(m[:pr_in_mol][2]), value(m[:pr_in_mol][3]
            ), value(m[:pr_in_mol][4]), value(m[:pr_in_mol][5]),
        value(m[:pr_in_mol][6]), value(m[:pr_in_mol][7]), value(m[:pr_in_mol][8]
            ), value(m[:pr_in_mol][9]), value(m[:pr_in_mol][10]),
        value(m[:pr_out_mol][1]), value(m[:pr_out_mol][2]), value(m[:pr_out_mol]
            [3]), value(m[:pr_out_mol][4]), value(m[:pr_out_mol][5]),
        value(m[:pr_out_mol][6]), value(m[:pr_out_mol][7]), value(m[:pr_out_mol]
            [8]), value(m[:pr_out_mol][9]), value(m[:pr_out_mol][10]),
        value(m[:preGHR_in_mol][1]), value(m[:preGHR_in_mol][2]), value(m[:
            preGHR_in_mol][3]), value(m[:preGHR_in_mol][4]), value(m[:
            preGHR_in_mol][5]),
        value(m[:preGHR_out_mol][1]), value(m[:preGHR_out_mol][2]), value(m[:
            preGHR_out_mol][3]), value(m[:preGHR_out_mol][4]), value(m[:
            preGHR_out_mol][5]),
        value(m[:ghr_in_mol][1]), value(m[:ghr_in_mol][2]), value(m[:ghr_in_mol]
            [3]), value(m[:ghr_in_mol][4]), value(m[:ghr_in_mol][5]),
        value(m[:ghr_out_mol][1]), value(m[:ghr_out_mol][2]), value(m[:
            ghr_out_mol][3]), value(m[:ghr_out_mol][4]), value(m[:ghr_out_mol][
            5]),
        value(m[:atr_in_mol][1]), value(m[:atr_in_mol][2]), value(m[:atr_in_mol]
            [3]), value(m[:atr_in_mol][4]), value(m[:atr_in_mol][5]),
        value(m[:atr_out_mol][1]), value(m[:atr_out_mol][2]), value(m[:
            atr_out_mol][3]), value(m[:atr_out_mol][4]), value(m[:atr_out_mol][
            5]),
        value(m[:postATR_in_mol][1]), value(m[:postATR_in_mol][2]), value(m[:
            postATR_in_mol][3]), value(m[:postATR_in_mol][4]), value(m[:
            postATR_in_mol][5]),
        value(m[:postATR_out_mol][1]), value(m[:postATR_out_mol][2]), value(m[:
            postATR_out_mol][3]), value(m[:postATR_out_mol][4]), value(m[:
            postATR_out_mol][5]),
        value(m[:itsr_in_mol][1]), value(m[:itsr_in_mol][2]), value(m[:
            itsr_in_mol][3]), value(m[:itsr_in_mol][4]), value(m[:itsr_in_mol][
            5]),
        value(m[:itsr_out_mol][1]), value(m[:itsr_out_mol][2]), value(m[:
            itsr_out_mol][3]), value(m[:itsr_out_mol][4]), value(m[:
            itsr_out_mol][5]),
        value(m[:preCond_in_mol][1]), value(m[:preCond_in_mol][2]), value(m[:
            preCond_in_mol][3]), value(m[:preCond_in_mol][4]), value(m[:
            preCond_in_mol][5]),
        value(m[:preCond_out_mol][1]), value(m[:preCond_out_mol][2]), value(m[:
            preCond_out_mol][3]), value(m[:preCond_out_mol][4]), value(m[:
            preCond_out_mol][5]),
        value(m[:cond_in_mol][1]), value(m[:cond_in_mol][2]), value(m[:
            cond_in_mol][3]), value(m[:cond_in_mol][4]), value(m[:cond_in_mol][
            5]),
        value(m[:cond_L]), value(m[:cond_liq_frac][1]), value(m[:cond_liq_frac][
            2]), value(m[:cond_liq_frac][3]), value(m[:cond_liq_frac][4]),
            value(m[:cond_liq_frac][5]),
        value(m[:cond_V]), value(m[:cond_vap_frac][1]), value(m[:cond_vap_frac][
            2]), value(m[:cond_vap_frac][3]), value(m[:cond_vap_frac][4]),
            value(m[:cond_vap_frac][5]),
        value(m[:psa_in_mol][1]), value(m[:psa_in_mol][2]), value(m[:psa_in_mol]
            [3]), value(m[:psa_in_mol][4]), value(m[:psa_in_mol][5]),
        value(m[:psa_outProduct_mol][1]), value(m[:psa_outProduct_mol][2]),
            value(m[:psa_outProduct_mol][3]), value(m[:psa_outProduct_mol][4]),
             value(m[:psa_outProduct_mol][5]),
        value(m[:psa_outPurge_mol][1]), value(m[:psa_outPurge_mol][2]), value(m[
            :psa_outPurge_mol][3]), value(m[:psa_outPurge_mol][4]), value(m[:
            psa_outPurge_mol][5]),
        value(m[:mix_in_T]), value(m[:mix_out_T]), value(m[:H2O_T]), value(m[:
            prePR_in_T]), value(m[:prePR_out_T]), value(m[:preGHR_in_T]), value
            (m[:preGHR_out_T]),
        value(m[:ghr_in_T]), value(m[:ghr_out_T]), value(m[:atr_in_T]), value(m[
            :postATR_in_T]), value(m[:postATR_out_T]),
        value(m[:preCond_out_T]), value(m[:cond_in_T]), value(m[:cond_L_T]),
            value(m[:cond_V_T]), value(m[:psa_in_T]), value(m[:psa_outProduct_T
            ]), value(m[:psa_outPurge_T]),
        value(m[:prePR_Q]), value(m[:preGHR_Q]), value(m[:ghr_Q]), value(m[:
            postATR_Q]), value(m[:itsr_Q]), value(m[:preCond_Q]),
        value(m[:H2Ostream]), value(m[:F_H2]), value(m[:F_H2_heat]), value(m[:
            F_NG]), value(m[:F_NG_heat]), value(m[:F_fluegas]), value(m[:F_inj]
            )
]
    return nominal_values, nominal_J
end

function optJ_func()
    __par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-9, "constr_viol_tol" => 1e-5,
             "print_level" => 0)
    m = Model(optimizer);

    eps = 0.1
```

```
par = deepcopy(__par)
#par.init.init_stream  = par.init.init_stream +5; # +d1
par.elCost = par.elCost*(1+eps); # +d2
#par.P_H2 = par.P_H2*(1+eps); # +d3
par.init.init_stream  = par.init.init_stream -5; # -d1
#par.elCost = par.elCost*(1-eps); # -d2
par.P_H2 = par.P_H2*(1-eps); # -d3


d1 = par.init.init_stream;
d2 = par.elCost;
d3 = par.P_H2;
###### Assembling the submodels to a larger model #######
MIX_model(m, par);
prePR_model(m, par);
PR_model(m, par);
preGHR_model(m, par);
GHR_model(m, par);
ATR_model(m, par);
postATR_model(m, par);
ITSR_model(m, par);
preCond_model(m, par);
Cond_model(m, par);
PSA_model(m, par);

@variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
     obj function
@variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
    is being used to heat up the process
@variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
    is being used to heat up the process
@variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
    being used in the process
@variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
    combusting natural gas which needs to be injected
@variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
    is being injected

##################### Connection constraints #######################
for i = 1:10
    @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
    @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
end

for i = 1:5 # After all heavier carbons are removed
    @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
    @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
    @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
    @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
    @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
    @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
    @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
    @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
        0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ##################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - d1 + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*d3 - compWsum*d2/1000);
    optimize!(m)
    opt_J = objective_value(m)
    return opt_J
end

function warm_nominal()
    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-9, "constr_viol_tol" => 1e-9,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    ##################### Connection constraints #######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end

    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
```

```julia
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ##################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - par.init.init_stream + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*par.P_H2 - compWsum*par.elCost/1000); # 44*m[:
        F_inj]/1000*par.P_inj
    #@NLobjective(m, Max, m[:psa_outProduct_mol][3])
    optimize!(m)
    variable_primal = Dict(x => value(x) for x in all_variables(m))
    return variable_primal
end
```

B.28 test.jl

The test.jl file contains the functions test_delta_y() and test_matrix() for calculating both the left-hand side and right-hand side of the linearization equation to see if the calculated gain matrices and Hessian matrices are correct.

```
variable_name = [
    "mix_in_mol[1]", "mix_in_mol[2]", "mix_in_mol[3]", "mix_in_mol[4]", "
        mix_in_mol[5]",
    "mix_in_mol[6]", "mix_in_mol[7]", "mix_in_mol[8]", "mix_in_mol[9]", "
        mix_in_mol[10]",
    "mix_out_mol[1]", "mix_out_mol[2]", "mix_out_mol[3]", "mix_out_mol[4]", "
        mix_out_mol[5]",
    "mix_out_mol[6]", "mix_out_mol[7]", "mix_out_mol[8]", "mix_out_mol[9]", "
        mix_out_mol[10]",
    "prePR_in_mol[1]", "prePR_in_mol[2]", "prePR_in_mol[3]", "prePR_in_mol[4]",
        "prePR_in_mol[5]",
    "prePR_in_mol[6]", "prePR_in_mol[7]", "prePR_in_mol[8]", "prePR_in_mol[9]",
        "prePR_in_mol[10]",
    "prePR_out_mol[1]", "prePR_out_mol[2]", "prePR_out_mol[3]", "prePR_out_mol
        [4]", "prePR_out_mol[5]",
    "prePR_out_mol[6]", "prePR_out_mol[7]", "prePR_out_mol[8]", "prePR_out_mol
        [9]", "prePR_out_mol[10]",
    "pr_in_mol[1]", "pr_in_mol[2]", "pr_in_mol[3]", "pr_in_mol[4]", "pr_in_mol
        [5]",
    "pr_in_mol[6]", "pr_in_mol[7]", "pr_in_mol[8]", "pr_in_mol[9]", "pr_in_mol
        [10]",
    "pr_out_mol[1]", "pr_out_mol[2]", "pr_out_mol[3]", "pr_out_mol[4]", "
        pr_out_mol[5]",
    "pr_out_mol[6]", "pr_out_mol[7]", "pr_out_mol[8]", "pr_out_mol[9]", "
        pr_out_mol[10]",
    "preGHR_in_mol[1]", "preGHR_in_mol[2]", "preGHR_in_mol[3]", "preGHR_in_mol
        [4]", "preGHR_in_mol[5]",
    "preGHR_out_mol[1]", "preGHR_out_mol[2]", "preGHR_out_mol[3]", "
        preGHR_out_mol[4]", "preGHR_out_mol[5]",
    "ghr_in_mol[1]", "ghr_in_mol[2]", "ghr_in_mol[3]", "ghr_in_mol[4]", "
        ghr_in_mol[5]",
    "ghr_out_mol[1]", "ghr_out_mol[2]", "ghr_out_mol[3]", "ghr_out_mol[4]", "
        ghr_out_mol[5]",
    "atr_in_mol[1]", "atr_in_mol[2]", "atr_in_mol[3]", "atr_in_mol[4]", "
        atr_in_mol[5]",
    "atr_out_mol[1]", "atr_out_mol[2]", "atr_out_mol[3]", "atr_out_mol[4]", "
        atr_out_mol[5]",
    "postATR_in_mol[1]", "postATR_in_mol[2]", "postATR_in_mol[3]", "
        postATR_in_mol[4]", "postATR_in_mol[5]",
    "postATR_out_mol[1]", "postATR_out_mol[2]", "postATR_out_mol[3]", "
        postATR_out_mol[4]", "postATR_out_mol[5]",
    "itsr_in_mol[1]", "itsr_in_mol[2]", "itsr_in_mol[3]", "itsr_in_mol[4]", "
        itsr_in_mol[5]",
    "itsr_out_mol[1]", "itsr_out_mol[2]", "itsr_out_mol[3]", "itsr_out_mol[4]",
        "itsr_out_mol[5]",
    "preCond_in_mol[1]", "preCond_in_mol[2]", "preCond_in_mol[3]", "
        preCond_in_mol[4]", "preCond_in_mol[5]",
    "preCond_out_mol[1]", "preCond_out_mol[2]", "preCond_out_mol[3]", "
        preCond_out_mol[4]", "preCond_out_mol[5]",
    "cond_in_mol[1]", "cond_in_mol[2]", "cond_in_mol[3]", "cond_in_mol[4]", "
        cond_in_mol[5]",
    "cond_L", "cond_liq_frac[1]", "cond_liq_frac[2]", "cond_liq_frac[3]", "
        cond_liq_frac[4]", "cond_liq_frac[5]",
    "cond_V", "cond_vap_frac[1]", "cond_vap_frac[2]", "cond_vap_frac[3]", "
```

```julia
        cond_vap_frac[4]", "cond_vap_frac[5]",
    "psa_in_mol[1]", "psa_in_mol[2]", "psa_in_mol[3]", "psa_in_mol[4]", "
        psa_in_mol[5]",
    "psa_outProduct_mol[1]", "psa_outProduct_mol[2]", "psa_outProduct_mol[3]", "
        psa_outProduct_mol[4]", "psa_outProduct_mol[5]",
    "psa_outPurge_mol[1]", "psa_outPurge_mol[2]", "psa_outPurge_mol[3]", "
        psa_outPurge_mol[4]", "psa_outPurge_mol[5]",
    "mix_in_T", "mix_out_T", "H2O_T", "prePR_in_T", "prePR_out_T", "preGHR_in_T"
        , "preGHR_out_T",
    "ghr_in_T", "ghr_out_T", "atr_in_T", "postATR_in_T", "postATR_out_T",
    "preCond_out_T", "cond_in_T", "cond_L_T", "cond_V_T", "psa_in_T", "
        psa_outProduct_T", "psa_outPurge_T",
    "prePR_Q", "preGHR_Q", "ghr_Q", "postATR_Q", "itsr_Q", "preCond_Q",
    "H2Ostream", "F_H2", "F_H2_heat", "F_NG", "F_NG_heat", "F_fluegas", "F_inj"
]
using JuMP, Ipopt, MathOptInterface, DataFrames, PrettyTables
include("enthalpy.jl")
include("0par.jl")
include("1MIX.jl")
include("2PrePR.jl")
include("3PR.jl")
include("4PreGHR.jl")
include("5GHR.jl")
include("6ATR.jl")
include("7PostATR.jl")
include("8ITSR.jl")
include("9PreCondensate.jl")
include("10Condensate.jl")
include("11PSA.jl")
include("dataframe.jl")
include("equilibrium.jl")
include("compWork.jl")
include("active.jl")
include("nominal_case.jl")
include("gain_d.jl")
include("gain_u.jl")
include("J_uu.jl")
include("J_ud.jl")
eps = 1e-5
nominal_values, nominal_J = nominal();

function test_delta_y(option)
    delta_u = 0;
    delta_d = 0;

    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-6, "constr_viol_tol" => 1e-8,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    if option == 1 # u1 + h
        delta_u = eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805*(1+eps) == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 2 # u2 + h
        delta_u = eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283*(1+eps) == 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 3 # u3 + h
        delta_u = eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818*(1+eps) == 0);
    elseif option == 4 # d1 + k
        delta_d = eps*par.init.init_stream;
        par.init.init_stream = par.init.init_stream*(1+eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 5 # d2 + k
        delta_d = eps*par.elCost;
        par.elCost = par.elCost*(1+eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == 6 # d3 + k
        delta_d = eps*par.P_H2;
        par.P_H2 = par.P_H2*(1+eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -1 # u1 - h
        delta_u = -eps*79.29706225438805;
        @NLconstraint(m, m[:nO2]-79.29706225438805*(1-eps) == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -2 # u2 - h
        delta_u = -eps*644.5953165006283;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283*(1-eps) == 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -3 # u3 - h
```

```
        delta_u = -eps*1291.817465833818;
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818*(1-eps) == 0);
    elseif option == -4 # d1 - k
        delta_d = -eps*par.init.init_stream;
        par.init.init_stream = par.init.init_stream*(1-eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -5 # d2 - k
        delta_d = -eps*par.elCost;
        par.elCost = par.elCost*(1-eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    elseif option == -6 # d3 - k
        delta_d = -eps*par.P_H2;
        par.P_H2 = par.P_H2*(1-eps);
        @NLconstraint(m, m[:nO2]-79.29706225438805 == 0);
        @NLconstraint(m, m[:pr_in_T]-644.5953165006283== 0);
        @NLconstraint(m, m[:atr_out_T]-1291.817465833818 == 0);
    else
        print("Option not valid")
    end

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints #######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################# Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############## Initial values ##########################################
    for i = 1:10
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ##################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - par.init.init_stream + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*par.P_H2 - compWsum*par.elCost/1000); # 44*m[:
        F_inj]/1000*par.P_inj
    #@NLobjective(m, Max, m[:psa_outProduct_mol][3])
    optimize!(m)
    nominal_J = objective_value(m)

    output_change = [
        value(m[:mix_in_mol][1]), value(m[:mix_in_mol][2]), value(m[:mix_in_mol]
            [3]), value(m[:mix_in_mol][4]), value(m[:mix_in_mol][5]),
        value(m[:mix_in_mol][6]), value(m[:mix_in_mol][7]), value(m[:mix_in_mol]
            [8]), value(m[:mix_in_mol][9]), value(m[:mix_in_mol][10]),
        value(m[:mix_out_mol][1]), value(m[:mix_out_mol][2]), value(m[:
            mix_out_mol][3]), value(m[:mix_out_mol][4]), value(m[:mix_out_mol][
            5]),
        value(m[:mix_out_mol][6]), value(m[:mix_out_mol][7]), value(m[:
            mix_out_mol][8]), value(m[:mix_out_mol][9]), value(m[:mix_out_mol][
            10]),
        value(m[:prePR_in_mol][1]), value(m[:prePR_in_mol][2]), value(m[:
            prePR_in_mol][3]), value(m[:prePR_in_mol][4]), value(m[:
            prePR_in_mol][5]),
        value(m[:prePR_in_mol][6]), value(m[:prePR_in_mol][7]), value(m[:
            prePR_in_mol][8]), value(m[:prePR_in_mol][9]), value(m[:
            prePR_in_mol][10]),
        value(m[:prePR_out_mol][1]), value(m[:prePR_out_mol][2]), value(m[:
            prePR_out_mol][3]), value(m[:prePR_out_mol][4]), value(m[:
            prePR_out_mol][5]),
        value(m[:prePR_out_mol][6]), value(m[:prePR_out_mol][7]), value(m[:
            prePR_out_mol][8]), value(m[:prePR_out_mol][9]), value(m[:
            prePR_out_mol][10]),
        value(m[:pr_in_mol][1]), value(m[:pr_in_mol][2]), value(m[:pr_in_mol][3]
            ), value(m[:pr_in_mol][4]), value(m[:pr_in_mol][5]),
        value(m[:pr_in_mol][6]), value(m[:pr_in_mol][7]), value(m[:pr_in_mol][8]
            ), value(m[:pr_in_mol][9]), value(m[:pr_in_mol][10]),
        value(m[:pr_out_mol][1]), value(m[:pr_out_mol][2]), value(m[:pr_out_mol]
            [3]), value(m[:pr_out_mol][4]), value(m[:pr_out_mol][5]),
        value(m[:pr_out_mol][6]), value(m[:pr_out_mol][7]), value(m[:pr_out_mol]
            [8]), value(m[:pr_out_mol][9]), value(m[:pr_out_mol][10]),
        value(m[:preGHR_in_mol][1]), value(m[:preGHR_in_mol][2]), value(m[:
            preGHR_in_mol][3]), value(m[:preGHR_in_mol][4]), value(m[:
            preGHR_in_mol][5]),
        value(m[:preGHR_out_mol][1]), value(m[:preGHR_out_mol][2]), value(m[:
            preGHR_out_mol][3]), value(m[:preGHR_out_mol][4]), value(m[:
            preGHR_out_mol][5]),
        value(m[:ghr_in_mol][1]), value(m[:ghr_in_mol][2]), value(m[:ghr_in_mol]
            [3]), value(m[:ghr_in_mol][4]), value(m[:ghr_in_mol][5]),
        value(m[:ghr_out_mol][1]), value(m[:ghr_out_mol][2]), value(m[:
            ghr_out_mol][3]), value(m[:ghr_out_mol][4]), value(m[:ghr_out_mol][
            5]),
        value(m[:atr_in_mol][1]), value(m[:atr_in_mol][2]), value(m[:atr_in_mol]
            [3]), value(m[:atr_in_mol][4]), value(m[:atr_in_mol][5]),
        value(m[:atr_out_mol][1]), value(m[:atr_out_mol][2]), value(m[:
            atr_out_mol][3]), value(m[:atr_out_mol][4]), value(m[:atr_out_mol][
            5]),
        value(m[:postATR_in_mol][1]), value(m[:postATR_in_mol][2]), value(m[:
            postATR_in_mol][3]), value(m[:postATR_in_mol][4]), value(m[:
            postATR_in_mol][5]),
        value(m[:postATR_out_mol][1]), value(m[:postATR_out_mol][2]), value(m[:
            postATR_out_mol][3]), value(m[:postATR_out_mol][4]), value(m[:
            postATR_out_mol][5]),
        value(m[:itsr_in_mol][1]), value(m[:itsr_in_mol][2]), value(m[:
            itsr_in_mol][3]), value(m[:itsr_in_mol][4]), value(m[:itsr_in_mol][
            5]),
        value(m[:itsr_out_mol][1]), value(m[:itsr_out_mol][2]), value(m[:
            itsr_out_mol][3]), value(m[:itsr_out_mol][4]), value(m[:
            itsr_out_mol][5]),
        value(m[:preCond_in_mol][1]), value(m[:preCond_in_mol][2]), value(m[:
            preCond_in_mol][3]), value(m[:preCond_in_mol][4]), value(m[:
            preCond_in_mol][5]),
        value(m[:preCond_out_mol][1]), value(m[:preCond_out_mol][2]), value(m[:
            preCond_out_mol][3]), value(m[:preCond_out_mol][4]), value(m[:
            preCond_out_mol][5]),
        value(m[:cond_in_mol][1]), value(m[:cond_in_mol][2]), value(m[:
            cond_in_mol][3]), value(m[:cond_in_mol][4]), value(m[:cond_in_mol][
            5]),
        value(m[:cond_L]), value(m[:cond_liq_frac][1]), value(m[:cond_liq_frac][
            2]), value(m[:cond_liq_frac][3]), value(m[:cond_liq_frac][4]),
            value(m[:cond_liq_frac][5]),
        value(m[:cond_V]), value(m[:cond_vap_frac][1]), value(m[:cond_vap_frac][
            2]), value(m[:cond_vap_frac][3]), value(m[:cond_vap_frac][4]),
            value(m[:cond_vap_frac][5]),
        value(m[:psa_in_mol][1]), value(m[:psa_in_mol][2]), value(m[:psa_in_mol]
            [3]), value(m[:psa_in_mol][4]), value(m[:psa_in_mol][5]),
        value(m[:psa_outProduct_mol][1]), value(m[:psa_outProduct_mol][2]),
            value(m[:psa_outProduct_mol][3]), value(m[:psa_outProduct_mol][4]),
             value(m[:psa_outProduct_mol][5]),
        value(m[:psa_outPurge_mol][1]), value(m[:psa_outPurge_mol][2]), value(m[
            :psa_outPurge_mol][3]), value(m[:psa_outPurge_mol][4]), value(m[:
            psa_outPurge_mol][5]),
        value(m[:mix_in_T]), value(m[:mix_out_T]), value(m[:H2O_T]), value(m[:
            prePR_in_T]), value(m[:prePR_out_T]), value(m[:preGHR_in_T]), value
            (m[:preGHR_out_T]),
        value(m[:ghr_in_T]), value(m[:ghr_out_T]), value(m[:atr_in_T]), value(m[
            :postATR_in_T]), value(m[:postATR_out_T]),
        value(m[:preCond_out_T]), value(m[:cond_in_T]), value(m[:cond_L_T]),
            value(m[:cond_V_T]), value(m[:psa_in_T]), value(m[:psa_outProduct_T
            ]), value(m[:psa_outPurge_T]),
        value(m[:prePR_Q]), value(m[:preGHR_Q]), value(m[:ghr_Q]), value(m[:
            postATR_Q]), value(m[:itsr_Q]), value(m[:preCond_Q]),
        value(m[:H2Ostream]), value(m[:F_H2]), value(m[:F_H2_heat]), value(m[:
            F_NG]), value(m[:F_NG_heat]), value(m[:F_fluegas]), value(m[:F_inj]
            )
    ]

    ###### Calculating left hand side ######
    lhs = zeros(length(output_change))
    for i in eachindex(output_change)
        lhs[i] = output_change[i] - nominal_values[i]
    end

    ##### Calculating right hand side #####
    if -3 <= option <= 3
        G_y_matrix = G_y(nominal_values, option, eps);
        rhs = G_y_matrix*delta_u;
    else
        G_yd_matrix = G_yd(nominal_values, option+3, eps);
        rhs = G_yd_matrix*delta_d;
    end

    ##### Calculating the difference between left hand side and right hand side
        #####
    diff = zeros(length(lhs));
    for i in eachindex(lhs)
        diff[i] = lhs[i] - rhs[i];
    end

    ##### Printing the results of the linearization #####
    return DataFrame(Variable = variable_name,
                     delta_y = lhs,
                     linearized = rhs,
                     Difference = diff)
end

#show(test_delta_y(-6), allrows=true)

function test_matrix(option)
    delta_d = 0;
    input_list = [79.29706225438805,644.5953165006283,1291.817465833818]
    par = _par();
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
             "tol" => 1e-10, "constr_viol_tol" => 1e-10,
             "print_level" => 0)
    m = Model(optimizer);

    ###### Assembling the submodels to a larger model #######
    MIX_model(m, par);
    prePR_model(m, par);
    PR_model(m, par);
    preGHR_model(m, par);
    GHR_model(m, par);
    ATR_model(m, par);
    postATR_model(m, par);
    ITSR_model(m, par);
    preCond_model(m, par);
    Cond_model(m, par);
    PSA_model(m, par);

    if option == 1 # d1 + k
        delta_d = [eps*par.init.init_stream; 0; 0];
        par.init.init_stream = par.init.init_stream*(1+eps);
    elseif option == 2 # d2 + k
        delta_d = [0;eps*par.elCost;0];
        par.elCost = par.elCost*(1+eps);
    elseif option == 3 # d3 + k
        delta_d = [0;0;eps*par.P_H2];
        par.P_H2 = par.P_H2*(1+eps);
    elseif option == -1 # d1 - k
        delta_d = [-eps*par.init.init_stream; 0; 0];
        par.init.init_stream = par.init.init_stream*(1-eps);
    elseif option == -2 # d2 - k
        delta_d = [0;-eps*par.elCost;0];
        par.elCost = par.elCost*(1-eps);
    elseif option == -3 # d3 - k
        delta_d = [0;0;-eps*par.P_H2];
        par.P_H2 = par.P_H2*(1-eps);
    else
        print("Option not valid")
    end

    @variable(m, 0 <= F_H2, start = 500); # H2 product that is being sold in the
         obj function
    @variable(m, 0 <= F_H2_heat, start = 1); # H2 from the product stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG_heat, start = 1); # NG from the initial stream that
        is being used to heat up the process
    @variable(m, 0 <= F_NG, start = 150); # NG from the initial stream that is
        being used in the process
    @variable(m, 0 <= F_fluegas, start = 1); # CO2 gas that generates from
        combusting natural gas which needs to be injected
    @variable(m, 0 <= F_inj, start = 200); # CO2 and other component gases that
        is being injected

    #################### Connection constraints ######################
    for i = 1:10
        @NLconstraint(m, m[:mix_out_mol][i] - m[:prePR_in_mol][i] == 0)
        @NLconstraint(m, m[:prePR_out_mol][i] - m[:pr_in_mol][i] == 0)
    end

    for i = 1:5 # After all heavier carbons are removed
        @NLconstraint(m, m[:pr_out_mol][i] - m[:preGHR_in_mol][i] == 0)
        @NLconstraint(m, m[:preGHR_out_mol][i] - m[:ghr_in_mol][i] == 0)
        @NLconstraint(m, m[:ghr_out_mol][i] - m[:atr_in_mol][i] == 0)
        @NLconstraint(m, m[:atr_out_mol][i] - m[:postATR_in_mol][i] == 0)
        @NLconstraint(m, m[:postATR_out_mol][i] - m[:itsr_in_mol][i] == 0)
        @NLconstraint(m, m[:itsr_out_mol][i] - m[:preCond_in_mol][i] == 0)
        @NLconstraint(m, m[:preCond_out_mol][i] - m[:cond_in_mol][i] == 0)
        @NLconstraint(m, m[:cond_vap_frac][i]*m[:cond_V] - m[:psa_in_mol][i] ==
            0)
    end
    ################ Same for the temperature
        ##################################
    @NLconstraint(m, m[:mix_out_T] - m[:prePR_in_T] == 0);
    @NLconstraint(m, m[:prePR_out_T] - m[:pr_in_T] == 0);
    @NLconstraint(m, m[:pr_out_T] - m[:preGHR_in_T] == 0);
    @NLconstraint(m, m[:preGHR_out_T] - m[:ghr_in_T] == 0);
    @NLconstraint(m, m[:ghr_out_T] - m[:atr_in_T] == 0);
    @NLconstraint(m, m[:atr_out_T] - m[:postATR_in_T] == 0);
    @NLconstraint(m, m[:postATR_out_T] - m[:itsr_in_T] == 0);
    @NLconstraint(m, m[:itsr_out_T] - m[:preCond_in_T] == 0);
    @NLconstraint(m, m[:preCond_out_T] - m[:cond_in_T] == 0);
    @NLconstraint(m, m[:cond_V_T] - m[:psa_in_T] == 0);


    ############# Initial values ########################################
    for i = 1:10
```

```
        @NLconstraint(m, par.init.init_comp[i]*m[:F_NG] - m[:mix_in_mol][i] == 0
            );
    end
    @NLconstraint(m, par.mix.in_T - m[:mix_in_T] == 0);
    @NLconstraint(m, par.mix.H2O_T - m[:H2O_T] == 0);

    ############# To ensure the GHR and ATR heat exchange ###################
    @NLconstraint(m, m[:ghr_Q] + m[:postATR_Q] == 0); #- additional_Q == 0);

    ##To ensure that the inlet hot stream is hotter than outlet cold stream##
    @NLconstraint(m, m[:atr_out_T] - m[:ghr_out_T] >= 25);
    @NLconstraint(m, m[:postATR_out_T] - m[:ghr_in_T] >= 25);

    ######## New constraints for the economic objective function #############
    @NLconstraint(m, m[:F_H2] - m[:psa_outProduct_mol][3] + m[:F_H2_heat] == 0);
    @NLconstraint(m, m[:F_NG] - par.init.init_stream + m[:F_NG_heat] == 0);
    @NLconstraint(m, m[:F_fluegas] - m[:F_NG_heat] - 2*m[:F_NG_heat]/0.79 == 0);
    @NLconstraint(m, m[:F_inj] - m[:F_fluegas] - sum(value(m[:psa_outPurge_mol][
        i]) for i = 1:5) == 0);
    @NLconstraint(m, m[:prePR_Q] + m[:preGHR_Q] - m[:F_H2_heat]*par.HHV_H2*2.016
         - sum(m[:F_NG_heat]*par.HHV_NG[i]*par.init.init_comp[i]*par.molarMass[
        i] for i = 1:10) == 0);


    compW1 = Wrev(m, m[:F_inj], 1, 10, m[:psa_outPurge_T], par);
    T2 = compT(m, m[:psa_outPurge_T],1,10);
    compW2 = Wrev(m, m[:F_inj],10,100,m[:psa_outPurge_T], par);
    compWsum = @NLexpression(m, compW1 + compW2);
    @NLobjective(m, Max, m[:F_H2]*par.P_H2 - compWsum*par.elCost/1000); # 44*m[:
        F_inj]/1000*par.P_inj
    #@NLobjective(m, Max, m[:psa_outProduct_mol][3])
    optimize!(m);

    J_uu_matrix = J_uu(5e-2);
    J_ud_matrix = J_ud(5e-2);
    rhs = inv(J_uu_matrix)*J_ud_matrix*(delta_d);
    lhs = [value(m[:nO2])-79.29706225438805,
        value(m[:pr_in_T])-644.5953165006283,
        value(m[:atr_out_T])-1291.817465833818]
    input_name = ["nO2", "pr_in_T", "atr_out_T"];
    diff = zeros(length(lhs));

    for i in eachindex(lhs)
        diff[i] = lhs[i] - rhs[i];
    end
    return DataFrame(Variable = input_name,
                    lhs = lhs,
                    rhs = rhs,
                    diff = diff);
end

#show(test_matrix(1), allrows=true)
```

NTNU
Norwegian University of
Science and Technology