Department of Chemical Engineering

University of California, Santa Barbara

Theoretical Green Dimethyl Carbonate Plant Design and Techno-Economic Analysis

Authors:	**Position:**
Isaiah Huma	Process Development Engineer
Wesley Johanson	Process Development Engineer
Thejas Ravish	Process Development Engineer

June 5th, 2024

Group 10

Executive Summary

This report presents a detailed techno-economic analysis and design for a proposed 100 kiloton per year (kta) plant producing polymer-grade Dimethyl Carbonate (DMC) for optical applications. The analysis was conducted by Chemical Investment Advisors (CIA) on behalf of the University of California Retirement Savings Program (UCRSP). The motivation for this project aligns with UCRSP's decision to divest from fossil fuel companies and invest in green chemistry initiatives that contribute to carbon dioxide (CO) reduction. The project is characterized by a Net Present Value (NPV) of -$88.9 million, indicating a significant loss under current economic conditions, and an Internal Rate of Return (IRR) of -56%, reflecting the unprofitability of the plant without substantial market price adjustments. The Total Capital Investment (TCI) required for the plant is estimated at $110.8 million. The plant is designed to achieve a production rate of 100 kta of polymer-grade DMC. The energy consumption for producing DMC is calculated at 34.6 MJ/kg DMC, and the process generates CO emissions of 0.38 kg CO per kg of DMC produced. Significant safety concerns include handling hazardous chemicals such as Ethylene Oxide (EO), Methanol (MeOH), and Aniline, which pose risks due to their flammability, explosiveness, and toxicity. As such, we can not recommend building of this plant under current market rates for chemicals.

Table of Contents

1 Introduction

1.1 Motivation and Background

The University of California's decision to divest the UC Retirement Savings Program (UCRSP) from companies owning fossil fuel reserves as of June 2022 means that there is significant amounts of available cash to invest. This, combined with current initiatives to reduce CO_2 emissions to reduce the impacts from global warming make green chemistry a strong contender for potential investments. Dimethyl carbonate (DMC) is particularly of note, as it has a wide variety of usages and, depending on the reaction route chosen, will consume Carbon dioxide as a feedstock. As such UCRSP hired Chemical Investment Advisors (CIA) to complete an FEL-1 Technoeconomic analysis up to and including the heat integration on the feasibility of a 100 kiloton per year (kta) plant for the production of polymer grade DMC for use in optical applications. Contracts are available to sell DMC at $1,100/MT (metric ton), while ethylene oxide (EO), the most expensive reactant, is available at $1,250/MT. However, the reaction chosen produces ethylene glycol (EG) as a side product equimolar to DMC, which can be sold for $500/MT, which makes the process potentially profitable when factoring in the cost of methanol (MeOH), $600/MT, and the incentive of $45/MT for the consumption of CO_2. Using these values for products, the profitability of the plant was measured and optimized to maximize net present value (NPV) and investor rate of return (IRR) for the UCRSP over the course of a 15 year plant life. In order to ensure the process remains sustainable, the cost of capturing and sequestering all carbon produced in the plant was accounted for as well.

1.2 Market Analysis

Numerous methods for the production of DMC exist, with the current most popular methods being through partial carbonylation [1] using methanol, oxygen, and carbon monoxide (CO) as feed stock, or through a methyl nitrite process[2], using methyl nitrile and CO as feedstock to produce DMC and nitric oxide as a byproduct. While both these options have lower feedstock costs, with oxygen being under $100/MT, and CO under $300/MT, the first method requires corrosion resistant equipment in the plant due to water formation, and tends to have low reaction rates[3], while the second produces an equimolar amount of NO, which is very toxic and has to properly captured and stored due to health concerns. The method utilized for this plant, detailed in Section 1.3, does not produce water, negating the need for corrosion resistance, and only produces 2-methoxyethanol (ME), a toxic chemical, in a separate side reaction that can be minimized with efficient reactor design, reducing the risk. Additionally, this process is expected to be more energy efficient and produce less CO_2[3] than both alternatives. While this does not guarantee it to be a better process, it means that the process is worth investigating. It is also worth noting that the DMC market is expected to grow to 1.8 billion USD by 2032, up from 1.25 billion USD as of 2024, with a compounded annual growth rate (CAGR) of 7.00%.[4] This is primarily driven by demand for a non-toxic methylating agent for the pharmaceutical industry, as well as the need for polycarbonate, which DMC is the precursor of polycarbonate is used in numerous industries, with its market expected to increase to 30.1 billion USD by 2030, up from 18.9 billion as of 2021.[5] Because demand for for DMC is expected to match demand for polycarbonate due to DMC having a key role in the production of polycarbonate, demand for DMC is likely to match these predictions, and as such it should be a reliable revenue stream for the plant over the course of its life. However, it should be noted that that annual production of DMC only about 100,000 tons, with an estimated capacity of up to 200,000 tons if desired,

meaning that competition for the market will be strong.[6]

1.3 Reaction Chemistry

The long term study performed by CIA found that the conversion of EO, CO_2, and MeOH into DMC and EG follows the reaction pathway outlined below, with the potential for an undesired side reaction into ME and CO_2.

$$\mathrm{EO} + \mathrm{CO_2} \xrightarrow{\Delta \mathrm{H_{rxn}} = -60.8\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{EC} \tag{1}$$

$$\mathrm{EC} + 2\,\mathrm{MeOH} \xrightleftharpoons{\Delta \mathrm{H_{rxn}} = \text{-}53.5\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{DMC} + \mathrm{EG} \tag{2}$$

$$\mathrm{EC} + \mathrm{MeOH} \xrightarrow{\Delta \mathrm{H_{rxn}} = \text{-}55.3\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{ME} + \mathrm{CO_2} \tag{3}$$

Testing found that the ideal catalyst for this reaction is a solid mixture of potassium iodide (KI) and potassium Carbonate (K_2CO_3) in a 1:1 mass ratio, with the optimal ratio found to be 1 gram of catalyst for every 50 mL of MeOH-EO solution entering the reactor. The catalyst was tested under both isothermal and isobaric conditions, and Reaction 3 was found to have the highest rate constant under all conditions (see Appendix B for rate data), making it important to control reaction rates to maximize conversion of EO to DMC, and to avoid production of excess CO_2. Kinetic data was provided for both isobaric and isothermal models. Isobaric data was collected at 150 bar with temperature varying from 80 °C to 140 °C, while isothermal data was collected at 140 °C with pressure varied from 50 to 150 bar. Additionally, under isothermal conditions the rate data will also vary as a function of mass density of CO_2, due to it being a supercritical fluid at pressure above 74 bar in the system, and primarily a gas below that pressure. Finally, all rate data was collected with CO_2 to EO reactor feed mole ratio of of 13:1, and MeOH to EO mole ratio of 15:1 at reactor inlet, due to Reaction 2 being an equilibrium reaction.

2 Conceptual Design

2.1 Process Overview

The process was optimized around assumptions that the necessary feedstocks of CO_2, MeOH, and EO were all available as 100% pure feeds. 53 kta of EO is provided at 10 bar and ambient temperature, 75 kta of MeOH is available at ambient pressure and temperature, and 51 kta of CO_2 is provided as a supercritical fluid at 50 bar and ambient temperature. All streams are pressurized to the operating pressure of 74 bar, and are subsequently mixed with the recycle streams before they are then heated up to the operating temperature of 140°C before entering the isothermal CSTR. The CSTR operates at 140°C and 74 bar, with a volume of 210, requiring a heat duty of 3.6 MW to maintain constant temperature. Additionally, 3.7 tonnes of catalyst are present in the reactor, which turn the solution in the reactor into a slurry. The reactor ef-

fluent is sent through a separator to return catalyst back to the reactor. The catalyst separation was not costed directly due to lack of data from the CIA lab, but was instead calculated using a minimum work of mixing calculation as indicated in Appendix C.3. The rest of the effluent is then depressurized to 1 bar and cooled to 0°C before entering flash drum V-100, where 98.3% of the CO_2 is separated from the reactor effluent as a vapor. It can then be sent to MIX-103.

The liquid effluent is then fed into distillation column T-100, with a V/F of 1.1, and 75 stages. T-100 separates DMC, MeOH, and the remainder CO_2 as distillate, and EG, EC, and ME in the bottoms. By utilizing a partial condenser at the top, a vent stream can be utilized to remove 68.8% of the remaining CO_2, along with some MeOH and trace DMC. This stream is then mixed with CO_2 from V-100 and sent to compressor K-101, where it is compressed back to 74 bar before entering mixer MIX-104.

Meanwhile, the distillate from T-100, containing primarily MeOH and DMC with trace CO_2, is sent to an extractive distillation column, T-101, where aniline is added as an entrainer in order to break the azeotrope that forms between methanol and DMC. 1,540 kta of aniline is added such that it is equimolar to the distillate stream from T-100, of which 0.5 kta is fresh feed and the remainder is recycled continuously. T-101 is 42 trays, and, with stage numbering being top down with the total condenser being stage 0 and the reboiler being stage 43, aniline is fed in at stage 2 while the DMC and MeOH stream is fed at stage 40. Aniline was chosen as the entrainer due to it's being more effective than phenol, which is normally used.

The distillate from T-101, which consists primarily of methanol, alongside the remainder CO_2 that was not removed during the flash and vent, is then pumped back to 74 bar before entering MIX-104 with the rest of the seperated CO_2, after which the combined stream is sent back to be recycled into the reactor.

The bottoms from T-100 is then fed into T-103, an 18 stage distillation column with a V/F of 0.4 which removes 2.1 kta of ME as distillate, while the EG and EC are removed in the bottoms. The ME is removed in the distillate as toxic liquid waste, as based on correlation and recommendation from *Predesign for Pollution Prevention and Control*.[7]

The bottoms stream, consisting of EC and EG to 10 bar in pump P-103, where it is then fed into distillation column T-104 at 10 bar. This produces 64 kta of 99.0 wt% EG as distillate, which is undergoes adiabatic pressure drop in valve V-102 and fed to MIX-107. The bottoms, consisting of EG and EC, is then depressurized by valve V-101 to 1 bar and fed to distillation column T-105, which has a V/F of 1.5 and 22 stages. This produces 0.999 mole% purity EC in the bottoms that can be recycled after being compressed to 74 bar. The distillate, consisting of an azeotropic mixture of EG and some EC is then sent to CSTR-101 where the 99% of the EC is hydrolyzed into EG so that it can be sold as a valuable byproduct. The reaction data did not include a kinetic model, and as such the reactor was not sized at this level. Data for this reaction was sourced from [8]. The reactor requires a feed of 2.0 kta of water in order to ensure 25% water by volume in the reactor. The reactor effluent, consisting primarily of EG and water, alongside trace EC and CO_2, must then be separated, providing 8.6 kta of EG at 99.0 wt% purity alongside 0.5 kta of CO_2 which is compressed back to 74 bar and recycled back into the plant. The EG is mixed with the EG distillate from T-104, which can then be sold as a valuable byproduct once cooled to 25°C. This separation system has not been designed due to not having our own reaction data to model performance on, and as such will require further analysis from our lab to design to ensure accuracy. The cost of carbon capture for the plant was also accounted for with a carbon tax, which accounts for carbon produces both in the plant and carbon used to produce steam and electricity for the plant as well.

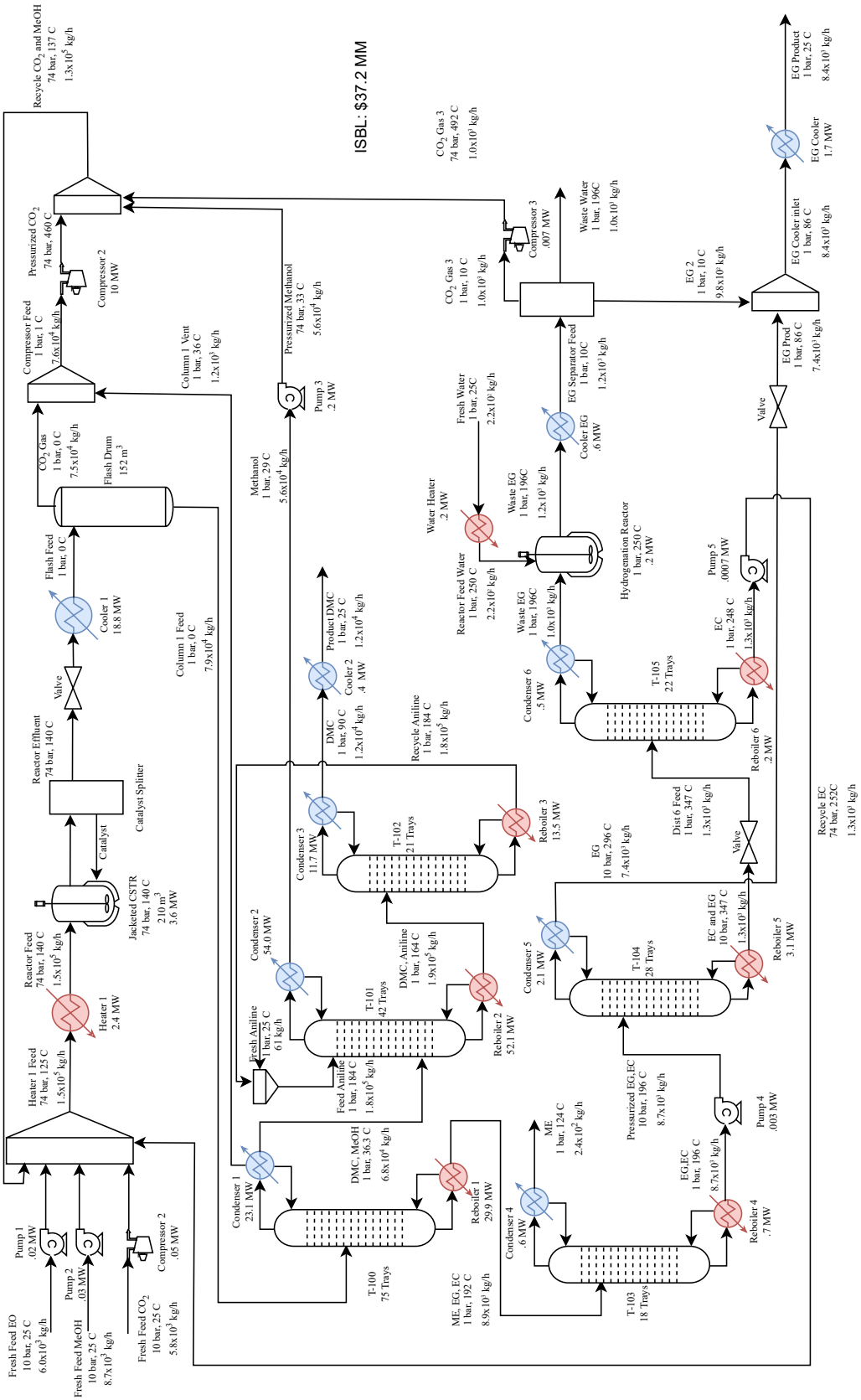


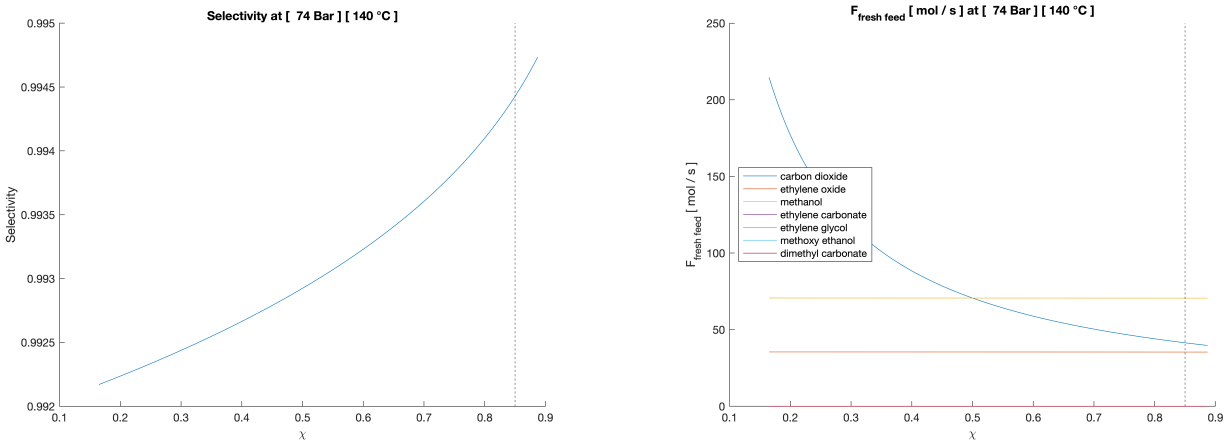
Figure 1: Process Flow Diagram

Table 1: Process stream labels and specifications

Name	Temperature (°C)	Pressure (bar)	Molar Compositions	Mass Flow Rate (kg/hr)
Fresh Feed EO	25	1.0	1.0 EO	140
Fresh Feed CO_2	25	5.0	1.0 CO	130
Fresh Feed MeOH	25	1.0	1.0 MeOH	270
Recycle EC	250	74	0.99 EC, $1.0 * 10^{-3}$ EG	3.0
Recycle MeOH and CO_2	140	74	$2.7 * 10-3$ DMC, 0.48 CO_2 0.52 MeOH, $1.8 * 10^{-4}$ Aniline	$3.5 * 10^3$
Conversion Reactor Feed	140	74	$7.5 * 10^{-4}$ EC, $3.4 * 10^{-2}$ EO $2.3 * 10^{-3}$ DMC, 0.44 CO_2, 0.52 MeOH, $1.6 * 10^{-4}$ Aniline	$4.0 * 10^3$
Separation System Feed	0	1.0	$3.5 * 10^{-2}$ EG, $1.1 * 10^{-3}$ EC, $3.8 * 10^{-2}$ DMC, 0.44 CO_2, 0.48 MeOH, $8.4 * 10^{-4}$ ME, $1.7 * 10^{-4}$ Aniline	$3.8 * 10^3$
CO_2 Gas	0	1.0	$4.5 * 10^{-3}$ DMC, 0.96 CO_2, $3.3 * 10^{-2}$ MeOH	$1.7 * 10^3$
Column 1 Feed	0	1.0	$6.4 * 10-2$ EG, $2.1 * 10^{-3}$ EC, $6.5 * 10^{-2}$ DMC, $1.4 * 10^{-2}$ CO_2, 0.85 MeOH, $1.5 * 10^{-3}$ ME, $3.1 * 10^{-4}$ Aniline	$2.1 * 10^3$
CO_2 Gas 2	36	1.0	$2.9 * 10^{-2}$ DMC, 0.69 CO_2, 0.28 MeOH	28
DMC, MeOH	36	1.0	$7.1 * 10^{-2}$ DMC, $4.7 * 10^{-3}$ CO_2, 0.92 MeOH	$1.9 * 10^3$
ME, EG, EC	190	1.0	0.94 EG, $3.0 * 10^{-2}$ EC, $2.2 * 10^{-2}$ ME, $4.6 * 10^{-3}$ Aniline	140
Fresh Aniline	25	1.0	1.0 Aniline	0.65
Recycle Aniline	180	1.0	1.0 Aniline	$1.9 * 10^3$
Methanol	29	1.0	$4.5 * 10^{-4}$ DMC, $5.0 * 10^{-3}$ CO_2, 0.99 MeOH, $3.7 * 10^{-4}$ Aniline	$1.8 * 10^3$
DMC, Aniline	160	1.0	$6.6 * 10^{-2}$ DMC, $1.2 * 10^{-4}$ MeOH, 0.93 Aniline,	$2.0 * 10^3$
DMC	90	1.0	0.99 DMC, $1.0 * 10^{-3}$ MeOH	130
ME	120	1.0	0.99 ME, $7.6 * 10^{-5}$ Aniline	3.1
Dist Feed 5	200	10	0.96 EG, $3.1 * 10^{-2}$ EC, $8.1 * 10^{-5}$ ME, $4.7 * 10^{-3}$ Aniline	140
EG Prod	86	1.0	0.99 EG, $9.4 * 10^{-5}$ ME, $5.4 * 10^{-3}$ Aniline	120
Dist Feed 6	286	1.0	0.78 EG, 0.22 EC	20
"Waste" EG	200	1.0	0.92 EG, $7.6 * 10^{-2}$	16
Water	25	1.0	1.0 Water	12
EG Sep	10	1.0	0.56 EG, $4.3 * 10^{-2}$ CO_2 0.40 Water	28
CO_2 Gas 3	490	74	1.0 CO_2	1.2
Waste Water	10	1.0	1.0	11
Product EG	25	1.0	0.99 EG, $1.0 * 10^{-4}$ ME, $7.1 * 10^{-3}$ Aniline	130

2.2 Decision Variables and Design

The reactor was modelled under both isothermal and isobaric conditions using the provided reaction rate data, varying pressure from 74 to 150 bar under the isothermal case, and from 80 to 140°C for the isobaric case. While the reaction rate data was provided for pressures as low as 50 bar, it was decided that pressures below the critical pressure of CO_2 would be ignored to ensure that no phase separation of CO_2 occurred. This ensured that the desired mole ratio of 13:1 for CO_2:EO would be maintained, as under phase separation the ratio would decrease significantly, as CO_2 would more readily enter the vapor phase than EO. Because the rate data was provided for conditions under great excess of CO_2 to EO, operating outside those conditions could result in the provided reaction kinetics being inaccurate. In order to



(a) S vs. X

(b) $F_{f,EO}$ vs X

Figure 2: 2a provided an ideal range of conversion, defined as percent of EO reacted in each pass of the reactor as seen in 2b. These charts are for isothermal reactor conditions at 140°C, 74 bar, and 210 m^3.

Table 2: Equipment specifications and energy streams

Equipment Name	Description	Energy Flow (MW)	Size	Material[9]	CapEx ($MM)	OpEx ($MM/yr)
CSTR-100	CSTR Reactor	3.6	210 m^3	Stainless Steel	0.3	$2.2*10^{-4}$
E-100	Heater 1	2.4	990 m^2	Stainless Steel	0.2	0.2
E-101	Cooler 1	18	$2.8*10^4$ m^2	Stainless Steel	1.9	
E-104	Heater Water	$1.8*10^{-1}$	21 m^2	Stainless Steel	$1.7*10^{-2}$	$1.6*10^{-2}$
E-105	Cooler EG	$5.7*10^{-1}$	140 m^2	Stainless Steel	$6.0*10^{-2}$	
E-106	EG Cooler	1.7	570 m^2	Stainless Steel	$1.5*10^{-1}$	
E-107	Cooler DMC	$3.8*10^{-1}$	130 m^2	Stainless Steel	$5.7*10^{-2}$	
K-100	Feed Compressor	$5.0*10^{-2}$	N/A	Stainless Steel	0.3	$3.5*10^{-2}$
K-101	Compressor 2	10	N/A	Stainless Steel	25	7.0
K-102	Compressor 3	$7.2*10^{-3}$	N/A	Stainless Steel	$6.7*10^{-2}$	$5.1*10^{-3}$
(T-100) E-108	Condenser 1	23	$3.1*10^2$ m^2	Stainless Steel	2.1	
(T-100) E-109	Reboiler 1	30	$6.0*10^2$ m^2	Stainless Steel	$7.1*10^{-1}$	2.7
(T-100) E-110	Condenser 2	54	$9.2*10^2$ m^2	Stainless Steel	$9.4*10^{-1}$	
(T-101) E-111	Reboiler 2	52	$1.4*10^3$ m^2	Stainless Steel	1.2	4.7
(T-102) E-112	Condenser 3	12	$3.0*10^2$ m^2	Stainless Steel	$4.5*10^{-1}$	
(T-102) E-113	Reboiler 3	14	$1.6*10^3$ m^2	Stainless Steel	1.3	1.2
(T-103) E-114	Condenser 4	$6.5*10^{-1}$	$2.7*10^2$ m^2	Stainless Steel	$4.2*10^{-1}$	
(T-103) E-115	Reboiler 4	$6.7*10^{-1}$	$5.0*10^{-1}$ m^2	Stainless Steel	$1.4*10^{-1}$	$6.1*10^{-2}$
(T-104) E-116	Condenser 5	2.1	51 m^2	Stainless Steel	$1.4*10^{-1}$	
(T-104) E-117	Reboiler 5	3.1	31 m^2	Stainless Steel	$1.0*10^{-1}$	$2.9*10^{-1}$
(T-105) E-118	Condenser 6	$5.3*10^{-1}$	$3.8*10^2$ m^2	Stainless Steel	$5.3*10^{-1}$	
(T-105) E-119	Reboiler 6	3.1	2.5 m^2	Stainless Steel	$2.0*10^{-2}$	$1.9*10^{-2}$
T-100	Distillation Column 1	N/A	D= 1.5 m, H= 73 m	Stainless Steel	$2.5*10^{-1}$	
T-101	Distillation Column 2	N/A	D= 1.5 m, H= 41 m	Stainless Steel	$1.5*10^{-1}$	
T-102	Distillation Column 3	N/A	D= 1.5 m, H= 20 m	Stainless Steel	$8.4*10^{-2}$	
T-103	Distillation Column 4	N/A	D= 1.5 m, H= 17 m	Stainless Steel	$7.4*10^{-2}$	
T-104	Distillation Column 5	N/A	D= 1.5 m, H= 27 m	Stainless Steel	$1.2*10^{-1}$	
T-105	Distillation Column 6	N/A	D= 1.5 m, H= 21 m	Stainless Steel	$8.7*10^{-2}$	
V-100	Flash Drum	N/A	152 m^3	Stainless Steel	$2.1*10^{-1}$	
X-100	Separator	$2.4*10^{-3}$	N/A	Stainless Steel	1.2*10−3	$2.1*10^{-4}$

determine optimal reactor conditions, separation was modelled using the minimum work of separation correlations (See Appendix C.3), which indicated that isothermal operation at the lowest possible pressure would maximize profit. This can be attributed to the compressor work required for the system being very large as a result of the large excess of CO_2 and MeOH required for operation not being offset by the improved reaction kinetics at higher pressure. The opposite is true for temperature, as the work of heating the streams to 140°C is offset by the better reaction kinetics present at high temperatures, as seen in Figure 3. Looking at this figure, a conversion of 85% was found to be optimal, leading to an optimized reactor volume of 210 m^3 at based on the Matlab code for level 3. The separation system was designed in Aspen Hysys, where the fractional recovery and composition in each exit stream was defined, and the number of stages were manually varied by hand until each column could be converged with reasonable

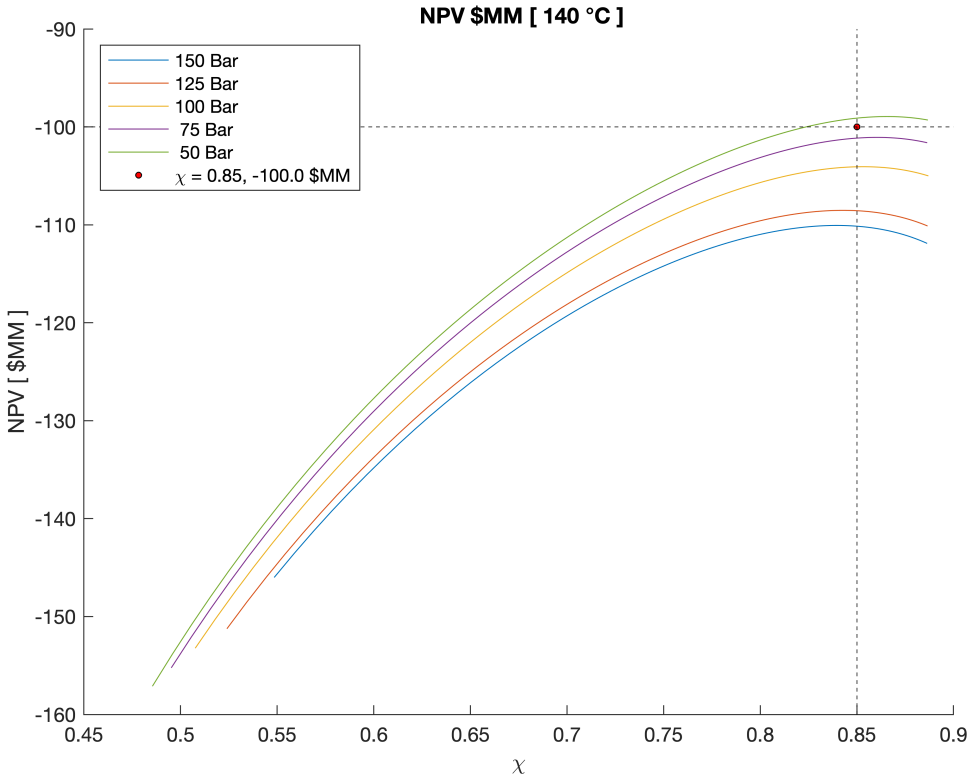


Figure 3: Plot of NPV vs. X under isothermal reaction with varied operating pressures.

duties on the reboiler and condenser for each one. The separation system was complicated by the presence of two azeotropes, one between DMC and MeOH and another between EG and EC. The DMC and MeOH azeotrope was bypassed by using a heavy entrainer, as specified earlier. In order to separate past the azeotrope of EG and EC, distillation was performed at two different pressures. By first separating at 10 bar, more EG can be removed from the system as the azeotrope is shifted to a mole fraction of EG than at atmospheric. After this, the bottoms, consisting of EG and EC, can then be separated at atmospheric pressure, and as the composition is below the mole fraction of EG required for the azeotrope, distillation will result in pure EC and a mixture of EG and EC out the bottom.

2.3 Comparison With Aspen Model

When modelling the system in Hysys, the NRTL model was utilized primarily due to its ability to effectively model the azeotrope present between EG and EC, as well as DMC and MeOH. Additionally, it was able to effectively model the impact of aniline as an entrainer on the system, reinforcing the decision to use NRTL. Doing this, different conversions and flowrates were calculated when compared with Matlab, likely due to the assumptions made in Matlab level 3 such as ideal gas law. As such, to find the optimal operating volume for the reactor, flowrates and conversions from Hysys were used to maximize accuracy. Once the flowsheet was converged and all columns were fully designed in Hysys, the volume of the reactor was varied and the NPV of the plant was calculated as a function of that volume at 4 points in total (See Appendix D), and based off the volumes tested, 210 m^3 remained as the optimal reactor volume.

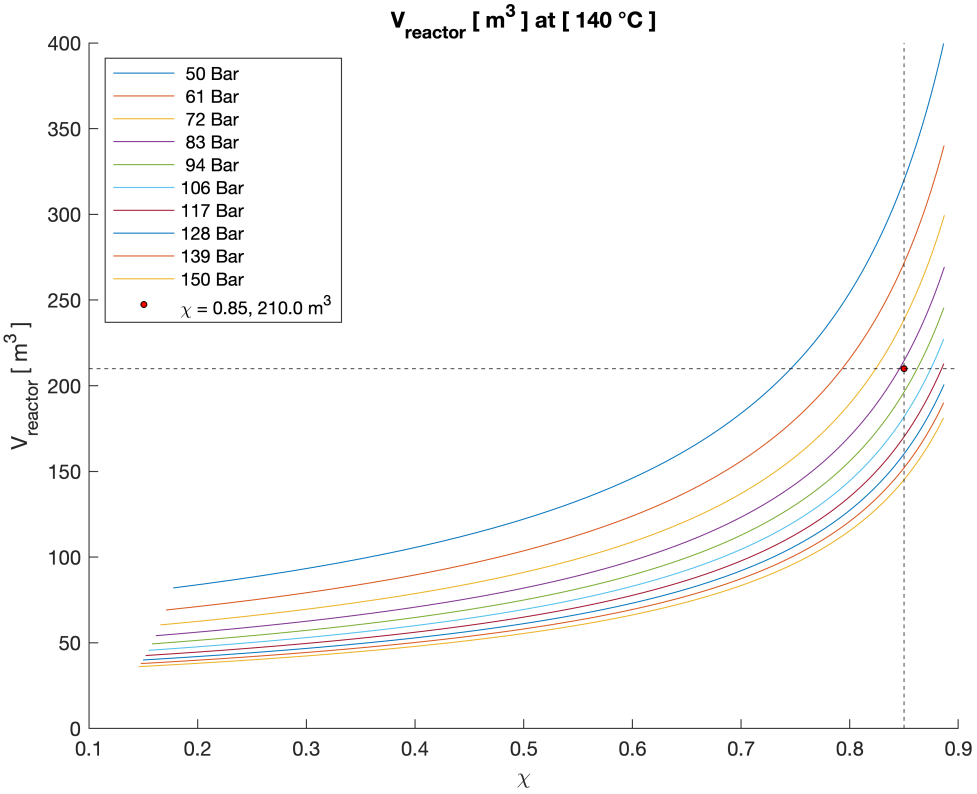
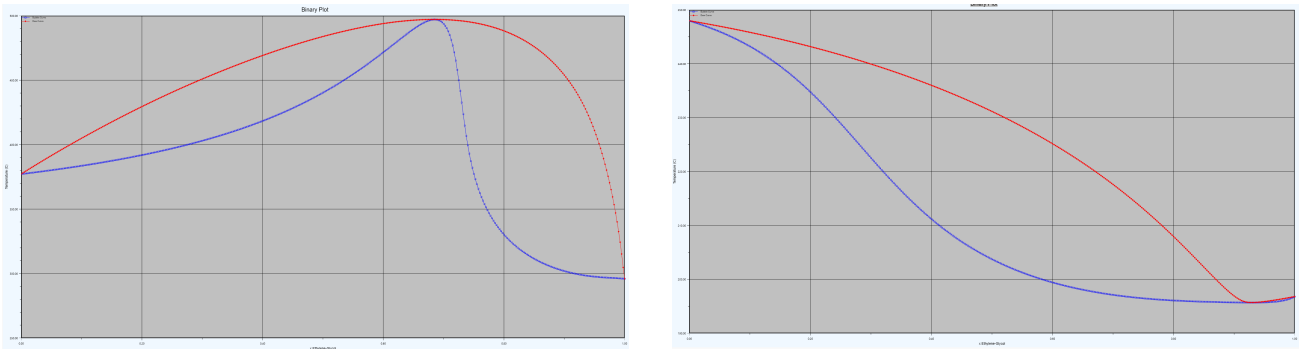


Figure 4: Plot of reactor volume vs conversion at varied pressures under isothermal operation.



(a) Txy diagram of EG and EC at 10 bar

(b) Txy diagram of EG and EC at 1 bar

Figure 5: Txy diagrams of EG and EC at 10 bar and 1 bar respectively, used to find the azeotropic composition and distill to desired purity of EG and EC.

3 Economic Analysis

Economic calculations were made using a 10 year straight line depreciation schedule, with assumptions of constant values for products, feed, and fuels. Additionally, estimates were used for outside battery limit costs and total capital investment based on the inside battery limit costs of the separation system, reactor, heaters, coolers, and heat exchangers (See Appendix D).

Based on the NPV calculations in Excel and in Matlab, the NPV of the plant at year 15 is $-88.9 MM, with an IRR of -56%. As seen in Figure 6, the cash flow for the plant remains negative, and as such it will not break even under current conditions, despite the carbon dioxide feedstock providing a profit stream due to government subsidies of $60/MT for consumption

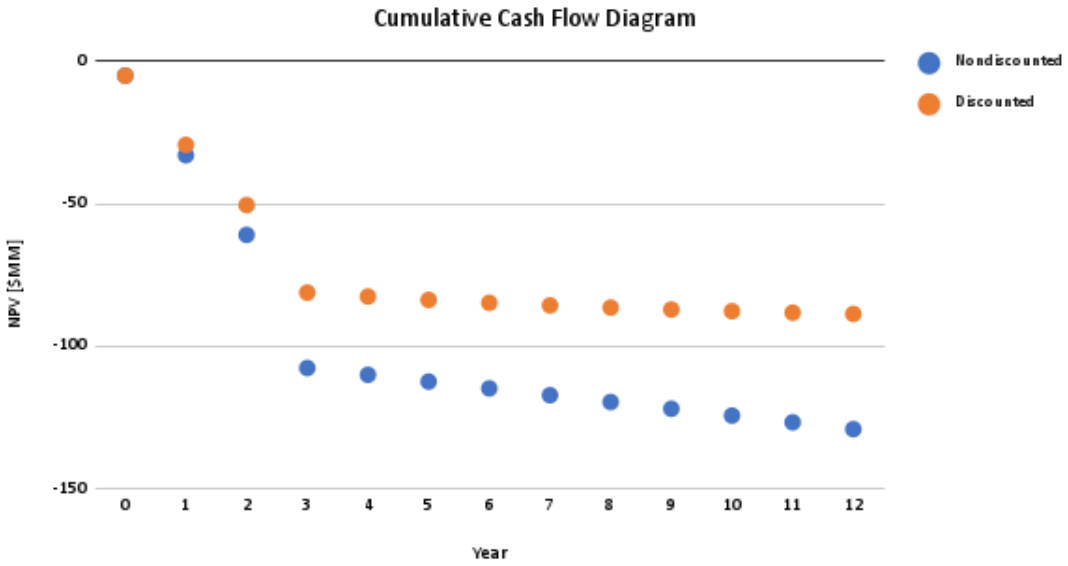


Figure 6: Discounted and non-discounted cash flow with costs accrued at end of year with 15% interest rate.

of CO_2 in industrial plants.

3.1 Inside Battery Loop Costs and Yearly Cost of Operation

The ISBL costs were calculated based on estimates for separator costs using the work of mixing for fluids (Appendix C.3) for removing all catalyst exiting the reactor, and Douglas' Cost Correlations (Appendix C.3) for the cost of sieve trays in a distillation column, with the column itself modeled as a pressure vessel. The reactors, flash drums, and PSAS vessels were also modeled as pressure vessels, and correlations for heat exchangers, heaters, and compressors were also obtained from Douglas' Cost Correlations.[10] Additionally, the final separation system after the hydrogenation reactor was also modelled with a minimum work of mixing calculation. As seen in Figure 7, the capital investment for the plant is dominated by the compressor costs, specifically compressor K-101 which is used to pressurize the recycled CO_2 back into the plant, due to it compressing 641 kta of CO_2 from 1 bar to 74 bar. Meanwhile, due to the high cost of EO, the yearly cost of operation for the plant is dominated by the feedstock cost, which makes up 71% of the operation cost, as seen in Figure 8.

Table 3: Economic Data

Substance	Price (value/cost)
Polymer–Grade DMC	$1,100/MT
Ethylene Glycol	$500/MT
Methanol	$6.00/MT
Ethylene Oxide	$1,250/MT
Carbon Dioxide Feedstock	$45.0/MT
Carbon Dioxide Subsidy	$60.0/MT
Natural Gas Fuel	$3.00/GJ
CO_2 (low P, high nitrogen content)	$125/MT Total Outsource Charge
Process Steam	Table 6.5[11]
Waste Water & Other Waste Streams	Ulrich and Vasudevan[7]

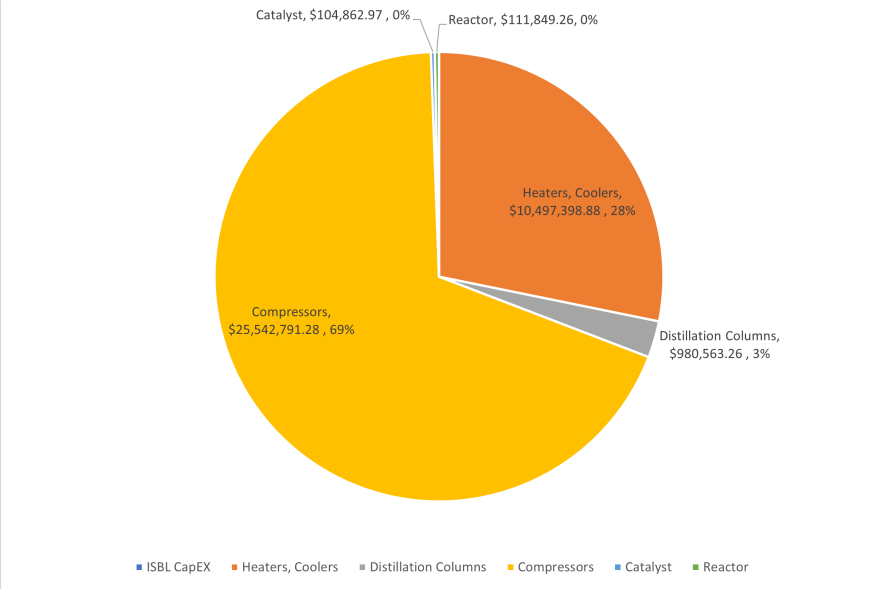


Figure 7: Total CAPEX for plant

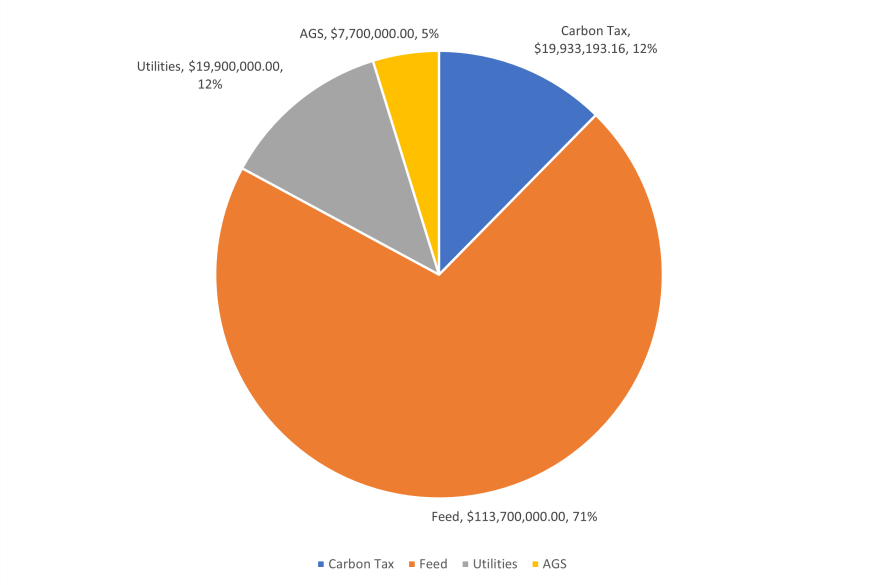


Figure 8: Yearly expenditures for plant

3.2 Total Capital Investment

The total capital investment for the plant was calculated using assumptions (See Appendix D) to convert the ISBL expenditures into the complete upfront cost of the plant, leading to a total capital investment of $105.8 MM, using the values seen below in Table 4.

3.3 Sensitivity Analysis

A sensitivity analysis on the plant was performed (See Appendix D.4) for details on value changed), and found that under all normal conditions the plant remains unprofitable, with the least negative NPV occurring with an EG value of $920/MT, which has occurred before[12], but is not the regular price and can not be reliable achieved. In order to maintain a positive NPV, DMC needs to sell for a minimum of $1,500/MT, which is well outside the normal fluctuations

Table 4: Total Fixed Capital Cost (TFCC) and Total Capital Investment (TCI).

Cost Type	$MM
ISBL	37.2
OSBL	14.9
Contingency Fee (CF)	13.0
Indirect Costs (IC)	19.5
TFCC	84.6
Working Capital (WC)	12.7
Start-up Costs (SC)	8.5
Land	5.0
TCI	110.8

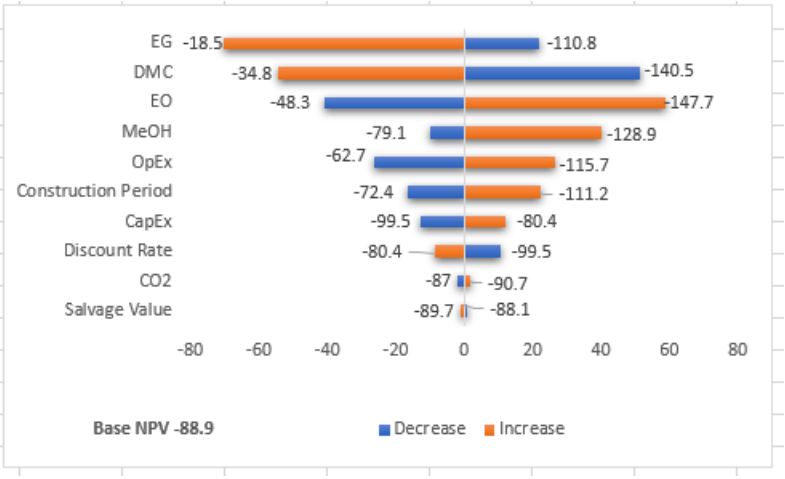


Figure 9: Tornado plot on the effect on NPV with changing parameters for plant based on past changes in values.

to be expected from DMC[13], with typical prices for DMC being in the range of $1,010-$1,070 per MT, meaning that even the contract price of $1,100/MT is above the values that can be expected long term.

4 Safety and Environmental Impact

Based on the provided HAZOP table (Appendix E.2) and chemical safety data summary (Appendix E.1), several critical safety concerns have been identified, primarily associated with the handling and processing of highly hazardous chemicals characterized by their flammability, explosiveness, and toxicity. EO poses the largest risk due to its extreme flammability and wide explosive limits, making it highly susceptible to explosive reactions under increased temperature or pressure. Furthermore, EO is moderately toxic to the skin, eyes, and when inhaled, thus presenting a significant risk to personnel and the environment. MeOH is another highly flammable substance with severe toxicity risks, including blindness and death if ingested. Increased temperatures can exacerbate these risks by producing toxic fumes, while pressure build-up could create hazardous conditions. Aniline is highly toxic, causing severe eye damage and systemic health effects, and is environmentally damaging. Deviations in temperature and pressure for DMC and ME can lead to hazardous conditions, including explosions and toxic emissions. EG presents toxicity risks to the kidneys and liver, while Carbon Dioxide CO poses asphyxiation risks by displacing oxygen in confined spaces.

To mitigate these risks, key safeguards must be implemented, including pressure relief de-

vices, temperature control systems, and flow control systems. Safety interlocks and alarms provide early warnings and automatic shutdowns to prevent hazardous conditions, while emergency cooling systems control runaway reactions or overheating. Regular maintenance and operator training are crucial for ensuring safe plant operation. Robust monitoring, control systems, and emergency protocols are essential to mitigate these risks, ensuring the safety of the plant and its personnel. Overall, the primary safety concerns revolve around managing deviations in pressure, temperature, and flow rates to prevent explosions, equipment rupture, toxic chemical release, and inadequate process performance.

5 Process Alternatives/Next Steps/Key Experiments Needed

A process alternative was considered in which the hydrogenation reactor was foregone and the waste EG from T-105 was instead directly combusted for fuel credit rather than purified to be sold as a product. However, it was quickly scrapped as the extra 8.6 kta of EG that can be sold results in an extra $4.3MM/year, while the extra cost in both capital expense and operating expense is minimal, as seen in Table 2 by looking at the combined costs of E-104, E-105, and X-100. As such, adding in the final reactor to maximize profit from the valuable byproduct of EG is worth the minimal extra cost of installing and operating the equipment necessary to do so.

As the plant NPV is highly negative, and the cash flow each year is negative, as seen in Figure 6, we do not recommend following through with plant design, but there are still important points to consider. If a better entrainer is found for separating the azeotrope of MeOH and DMC, or a more efficient separation process entirely, it may be possible to reduce operating costs such that a positive cash flow is created, allowing for a positive NPV given the same economic assumptions. If this is possible, lab data from CIA will be necessary on the solubility of the catalysts potassium iodide and potassium carbonate in the reactor fluid, as well as information on the ability to separate these catalysts from the process stream if they will be exiting the reactor. Based on information from other papers, the catalyst will fully dissolve in the liquid portion of the reactor[14], and as such data from the lab on separation of the catalyst will be needed if further development is deemed worthwhile given a more efficient DMC and MeOH separation.

6 Conclusions

The feasibility study for the 100 kta DMC production plant indicates several economic and safety challenges. The projected NPV remains negative unless DMC prices significantly exceed current market rates. The total capital investment is substantial, and the plant's profitability is highly sensitive to the prices of DMC and its byproducts. Additionally, the handling of hazardous chemicals necessitates rigorous safety measures to ensure the well-being of personnel and the environment.

While the production process achieves notable energy efficiency and relatively low CO emissions, the economic viability under current market conditions is questionable. Therefore, strategic adjustments in production costs, product pricing, and market conditions are critical for the project's success.

In summary, the project aligns well with UCRSP's green investment goals but our team firmly recommends against further development due to the highly negative NPV of $88.9MMoverthe15yearlif

7 References

[1] *Process for preparing dimethyl carbonate* (n.d.). https://patents.google.com/patent/US5523452A/en.

[2] "Dimethyl carbonate synthesis and other oxidative reactions using alkyl nitrites" (1997). In: *Springer Link* 1, pp. 77–88. DOI: 03.1997. URL: https://link.springer.com/article/10.1023/a:1019020812365?utm_source=getftr&utm_medium=getftr&utm_campaign=getftr_pilot.

[3] "Sustainable Dimethyl Carbonate Production from Ethylene Oxide and Methanol" (2020). In: *Chemical Engineering Technology* 43.12, pp. 2484–2492. DOI: 11.2020. URL: https://onlinelibrary.wiley.com/doi/epdf/10.1002/ceat.202000150.

[4] *Dimethyl Carbonate Market Analysis* (n.d.). https://www.chemanalyst.com/industry-report/dimethyl-carbonate-market-1829.

[5] *Polycarbonate Market Research Report* (n.d.). https://www.marketresearchfuture.com/reports/polycarbonate-market-1080.

[6] "Evaluating the Viability of Dimethyl Carbonate as an Alternative Fuel for the Transportation Sector" (2017). In: *National Center for Sustainable Transportation*. DOI: 06.2017. URL: https://rosap.ntl.bts.gov/view/dot/32485/dot_32485_DS1.pdf.

[7] G.D. Ulrich, P.T. Vasudevan (2007). "Predesign for Pollution Prevention and Control". In: *CEP*.

[8] "Hydrothermal conversion of ethylene carbonate to ethylene glycol" (2016). In: *International Journal of Hydrogen Energy* 41.21, pp. 9118–9122. DOI: 06.2016. URL: https://www.sciencedirect.com/science/article/pii/S0360319915027615.

[9] Perry, R.H. (2008). *Materials of Construction Perrys Handbook*. URL: https://ucsb.instructure.com/courses/16230/files/2039012?module_item_id=1025092.

[10] Douglas, J. (1988). *Conceptual Design of Chemical Processes*. McGraw-Hill Higher Education.

[11] M.F. Doherty, M.F. Malone (2001). *Conceptual Design of Distillation Systems*. McGraw-Hill Higher Education.

[12] *Ethylene Glycol Price Index* (n.d.). https://businessanalytiq.com/procurementanalyt index/ethylene-glycol-price-index/.

[13] *DiMethyl Carbonate Prices Trend and Forecast* (n.d.). https://www.chemanalyst.com/industry-report/dimethyl-carbonate-market-1829.

[14] "Solubility of Potassium Carbonate and Potassium Hydrocarbonate in Methanol" (2002). In: *J. Chem. Eng. Data* 47.5, pp. 1175–1176. URL: https://pubs.acs.org/doi/pdf/10.1021/je020012v.

[15] *Safety Data Sheet Ethylene Oxide* (n.d.). URL: https://www.airgas.com/msds/001081.pdf.

[16] *Safety Data Sheet Carbon Dioxide* (n.d.). URL: https://www.airgas.com/msds/001013.pdf.

[17] *Safety Data Sheet Methanol* (n.d.). URL: https://www.airgas.com/msds/001065.pdf.

[18] *Safety Data Sheet Ethylene Carbonate* (n.d.). URL: https://www.fishersci.com/store/msds?partNumber=AC118410051&countryCode=US&language=en.

[19] *Safety Data Sheet Dimethyl Carbonate* (n.d.). URL: https://www.fishersci.com/store/msds?partNumber=AC116120025&countryCode=US&language=en.

[20] *Safety Data Sheet Ethylene Glycol* (n.d.). URL: https://www.fishersci.com/msdsproxy%3FproductName%3DE1774%26productDescription%3DETHYLENE%2BGLYCOL%2BLABORATORY%2B4L%26catNo%3DE177-4%2B%26vendorId%3DVN00033897%26storeId%3D10652.

[21] *Safety Data Sheet 2-Methoxyethanol* (n.d.). URL: https://www.fishersci.com/store/msds?partNumber=AC396891000&productDescription=2-METHOXYETHANOL&vendorId=VN00032119&countryCode=US&language=en.

[22] *Safety Data Sheet Aniline* (n.d.). URL: https://www.fishersci.com/content/dam/fishersci/en_US/documents/programs/education/regulatory-documents/sds/chemicals/chemicals-a/S25179.pdf.

[22]v.

A Appendix

A.1 Level 1-3 Decisions and Mole Balances (Douglas Hierarchy)

Recall the systems of reactions,

$$\mathrm{EO} + \mathrm{CO_2} \xrightarrow{\Delta \mathrm{H_{rxn}} = \text{-}60.8\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{EC} \tag{A.1}$$

$$\mathrm{EC} + 2\,\mathrm{MeOH} \xrightleftharpoons{\Delta \mathrm{H_{rxn}} = -53.5\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{DMC} + \mathrm{EG} \tag{A.2}$$

$$\mathrm{EC} + \mathrm{MeOH} \xrightarrow{\Delta \mathrm{H_{rxn}} = \text{-}55.3\ \frac{\mathrm{kJ}}{\mathrm{mol}}} \mathrm{ME} + \mathrm{CO_2} \tag{A.3}$$

where EO = ethylene oxide, EC = ethylene Carbon ate, DMC = dimethyl Carbon ate, EG = ethylene glycol, and ME = 2-methoxyethanol.

Using the above reactions, it can be determined the are C - R = 7 - 3 = 4 independent mole balances, where C is the number of components and R is the number of reactions.

As such, the Level 2 mole balances using Douglas' Hierarchy of Design [10] and the systematic approach outlined by Doherty[11] can be written as follows:

Ethylene oxide mole balance (instantaneous reaction, therefore no EO leaves the plant):

$$\mathrm{F_{f,EO}} - \xi_1 = 0 \tag{A.4}$$

Carbon dioxide mole balance (assuming no CO_2 leaves the plant):

$$\mathrm{F_{f,CO_2}} + \xi_3 - \xi_1 = 0 \tag{A.5}$$

Ethylene Carbon ate mole balance (assuming no EC leaves the plant):

$$\xi_1 - \xi_2 - \xi_3 = 0 \tag{A.6}$$

Methanol mole balance (assuming no MeOH leaves the plant):

$$\mathrm{F_{f,MeOH}} - 2\xi_2 - \xi_3 = 0 \tag{A.7}$$

Dimethyl Carbon ate balance (same equation as ethylene glycol balance):

$$-\mathrm{P_{DMC}} + \xi_2 = 0 \tag{A.8}$$

2-Methoxyethanol balance:

$$-\mathrm{P_{ME}} + \xi_3 = 0 \tag{A.9}$$

where F_i and P_i are the flow rate of species i in $\frac{mol}{s}$ entering and exiting the plant, respectively and ξ_i is the extent of reaction in $\frac{mol}{s}$.

It is pivotal to understand the relationship between the desired product and the undesired product, such that selectivity is defined as

$$S = \frac{P_{DMC}}{F_{f,EO}} \tag{A.10}$$

Subbing equations (A.4, (A.9), and (A.10) into equation (A.5 and rearranging for P_{ME} gives,

$$P_{ME} = \frac{P_{DMC}}{S} - F_{f,\,CO_2} \tag{A.11}$$

Plugging in equations (A.4), (A.8), and (A.9) into equation (A.6), we see that

$$F_{f,EO} + P_{EG} - P_{ME} = 0 \tag{A.12}$$

Subbing in using equations (A.10), (A.11), and (A.8) and rearranging for F_{f,CO_2}

$$F_{f,CO_2} = P_{DMC} \tag{A.13}$$

Back substituting equation (A.13) into equation (A.11)

$$P_{ME} = P_{DMC}\left(\frac{1}{S} - 1\right) \tag{A.14}$$

Plugging in equations (A.8) and (A.9) into (A.5)

$$F_{MeOH} - 2P_{DMC} - P_{ME} = 0 \tag{A.15}$$

Additionally, we sub in equation (A.14) into (A.16) and rearrange and simplify for $F_{f,MeOH}$ to get

$$F_{MeOH} = P_{DMC}\left(1 + \frac{1}{S}\right) \tag{A.16}$$

Using a degree of freedom analysis DOF = V - E = 7 - 5, where V is the number of variables and E is the number of equations, there are 2 design specifications required. As defined in the problem statement, the production of dimethyl Carbon ate is fixed, such that

$$P_{DMC} = 100 \tag{A.17}$$

(Note: P_{DMC} [=] kta and must be converted to $\frac{mol}{s}$ to use these equations)

Now that the global plant balance has been accounted for, a closer examination of the internal streams should be evaluated. Proceeding with Douglas' Hierarchy[10], deliberate Level 3 Balance derivations for the recycle and reactor streams are required. To simplify the mole

balances, a theoretical conversion reactor is considered for the instantaneous reaction (A.1), where the mixing point occurs before this reactor, such that:

Recycle balance on EO at the mixing point:

$$F_{f,EO} = G_{EO} \tag{A.18}$$

Recycle balance on EC at the mixing point:

$$R_{EC} = G_{EC} \tag{A.19}$$

Recycle balance on MeOH at the mixing point:

$$F_{f,MeOH} + R_{MeOH} = G_{MeOH} \tag{A.20}$$

Recycle balance on CO_2 at the mixing point:

$$F_{f,CO_2} + R_{CO_2} = G_{CO_2} \tag{A.21}$$

where R_i is the recycle flow rate of species i in $\frac{mol}{s}$ leaving the reactor effluent and entering at the mixing point before the conversion reactor, G_i is the flow rate of species i in $\frac{mol}{s}$ entering the conversion reactor, and F_i is the flow rate of species i in $\frac{mol}{s}$ entering the reactor.

MeOH balance at the conversion reactor:

$$G_{MeOH} = F_{MeOH} = MR_{MeOH} * G_{EO} \tag{A.22}$$

Subbing (A.18) into (A.22), then using the definition of $F_{f,EO}$ from (A.10)

$$G_{MeOH} = F_{MeOH} * \frac{P_{DMC}}{S} \tag{A.23}$$

Subbing (A.23) into (A.20) and rearranging for R_{MeOH}

$$R_{MeOH} = P_{DMC}\left(\frac{MR_{MeOH}}{S} - 1 - \frac{1}{S}\right) \tag{A.24}$$

EO balance at the conversion reactor:

$$G_{EO} = \tau * r_{EO} \tag{A.25}$$

CO_2 balance at the conversion reactor:

$$G_{CO_2} - \tau * r_{EO} = F_{CO_2} \tag{A.26}$$

Plugging in A.25 into the above and known constraint $F_{CO_2} = MR_{CO_2} * G_{EO}$ and solving for G_{CO_2}

$$G_{CO_2} = G_{EO} * (MR_{CO_2} + 1) \tag{A.27}$$

Subbing in (A.18) and the definition of F_{MeOH}

$$R_{CO_2} = P_{DMC}\left(\frac{MR_{CO_2}+1}{S} - 1\right) \tag{A.28}$$

EC balance at the conversion reactor:

$$G_{EC} - F_{EC} + \tau * r_{EO} = 0 \tag{A.29}$$

Plugging in (A.25), rearranging for F_{EC}, and back subbing into (A.19)

$$F_{EC} = R_{EC} + \frac{P_{DMC}}{S} \tag{A.30}$$

We define the conversion of the reactor as

$$X = \frac{F_{EC} - R_{EC}}{F_{EC}} \tag{A.31}$$

rearranging this definition for R_{EC}, we get

$$R_{EC} = F_{EC} * (1 - X) \tag{A.32}$$

Plugging in (A.32) into (A.30) and solving for F_{EC}

$$F_{EC} = \frac{P_{DMC}}{S\,X} \tag{A.33}$$

Back substituting into (A.32)

$$R_{EC} = \frac{P_{DMC}}{S} * \frac{1-X}{X} \tag{A.34}$$

B Reaction Models, Rate Constants, and Calculated Design Variables

The final step to determine all the flow rates of the plant is to determine the flow rates of all species exiting the reactor system. The rates of reactions, rate constants, and reactor design equations are needed to generate a system equations. Within reactor pressures of 50 to 150 bar and reactor temperatures of 80 to 140°C, reaction (A.1) is instantaneous. The rate expressions for reactions (A.2) and (A.3) are given by

$$r_{2,f} = k_{2,f} C_{EC}^{0.8} \tag{B.1}$$

$$r_{2,r} = k_{2,r} C_{DMC} C_{EG} \tag{B.2}$$

$$r_3 = k_3 C_{EC} \tag{B.3}$$

where the rate expressions for $r_{2,f}$ and r_3 have lumped the concentration dependence of methanol into an effective rate constant since methanol is present in large excess, such that the molar ratio of MeOH to EO = 15. Additionally, the molar ratio of CO_2 to EO = 12 to favor the forward reaction rate of (A.2). Note: CO_2 is not 13 because 1 mol is consumed in the conversion reactor.

The reactor can be run at either isobaric or isothermal conditions, in which the respective rate constants vary depending on the temperature, T, or the mass density of CO_2, ρ. For the isobaric model, the Arrhenius dependence of the rate constants is

$$k_{2,f} = 6.59 * 10^2 \exp\left(\frac{-37,200}{RT}\right) \tag{B.4}$$

$$k_{2,r} = 1.19 * 10^4 \exp\left(\frac{-53,700}{RT}\right) \tag{B.5}$$

$$k_3 = 1.89 * 10^6 \exp\left(\frac{-82,400}{RT}\right) \tag{B.6}$$

where the T is in Kelvin, the activation energy is in $\frac{J}{mol}$, and the concentrations of the reacting species are in $\frac{mol}{L}$. The units of $k_{2,f}$ are $\frac{mol^{0.2}}{L^{0.2}s}$; the units of $k_{2,r}$ are $\frac{mol}{L\,s}$; the units of k_3 are $\frac{1}{s}$. Therefore, the units of $r_{2,f}, r_{2,r}$, andr_3 are $\frac{mol}{L\,s}$. Furthermore, a fixed catalyst concentration of 1 g of catalyst for every 50 mL of EO to MeOH solution is required inside the reactor.

On the other hand, the isothermal reaction rate constants are given by

$$k_{2,f} = 0.013 \qquad 50 < \rho < 246.82 \ \frac{g}{L} \tag{B.7}$$

$$k_{2,f} = 0.02486 - 4.943 * 10^{-5}\rho \qquad \rho > 246.82 \ \frac{g}{L} \tag{B.8}$$

$$k_{2,r} = 0.01486\rho^{-0.873} \qquad \rho > 50 \ \frac{g}{L} \tag{B.9}$$

$$k_3 = 3.014 * 10^{-4} \exp(-5.99 * 10^{-3}\rho) \qquad \rho > 50 \ \frac{g}{L} \tag{B.10}$$

The design equation for a CSTR for i species is

$$C_{i,o} - C_i + \tau r_i = 0 \tag{B.11}$$

where C_i is the molar concentration of species i in $\frac{mol}{L}$ and τ is the ratio of the volume over the total volumetric flow rate, such that $\tau = \frac{V}{q}$, otherwise known as the resonance time.

Writing Equation (B.11) for each species (except EO, see Appendix A.1) gives the follow-

ing:

$$C_{EC,o} - C_{EC} + \tau r_{EC} = 0 \tag{B.12}$$

$$C_{MeOH,o} - C_{MeOH} + \tau r_{MeOH} = 0 \tag{B.13}$$

$$C_{CO_2,o} - C_{CO_2} + \tau r_{CO_2} = 0 \tag{B.14}$$

$$C_{DMC,o} - C_{DMC} + \tau r_{DMC} = 0 \tag{B.15}$$

$$C_{EG,o} - C_{EG} + \tau r_{EG} = 0 \tag{B.16}$$

$$C_{ME,o} - C_{ME} + \tau r_{ME} = 0 \tag{B.17}$$

where $C_{DMC,o}$, $C_{EG,o}$, and $C_{ME,o}$ are all 0 as there is no feed into the reactor.

Given reactions (A.2) and (A.3) and reaction rates (B.1) to (B.3), we can derive the following expressions

$$r_{EC} = k_{2,r} C_{DMC} C_{EG} - k_{2,f} C_{EC}^{0.8} - k_3 C_{EC} \tag{B.18}$$

$$r_{MeOH} = 2\, k_{2,r} C_{DMC} C_{EG} - 2\, k_{2,f} C_{EC}^{0.8} - k_3 C_{EC} \tag{B.19}$$

$$r_{CO_2} = k_3 C_{EC} \tag{B.20}$$

$$r_{DMC} = k_{2,f} C_{EC}^{0.8} - k_{2,r} C_{DMC} C_{EG} \tag{B.21}$$

$$r_{EG} = k_{2,f} C_{EC}^{0.8} - k_{2,r} C_{DMC} C_{EG} \tag{B.22}$$

$$r_{ME} = k_3 C_{EC} \tag{B.23}$$

Given the molar mass and densities of the inlet species, the volumetric flow rates at the inlet, $q_{i,o}$, are calculated and used with τ to calculate the basis reactor volume and, in turn, $C_{i,o}$. Therefore, to solve the system of equations for the outlet concentrations, we utilize the above system of equations and a nonlinear solver by setting the basis initial conditions for $C_{i,o}$ and varying τ and scaling with respect to the DMC production rate.

C Equipment Design Summary

C.1 Heater Design

The heaters and coolers are designed to be adiabatic, shell and tube heat exchangers, so the capital cost is determined by

$$\text{Installed Cost, \$} = \left(\frac{1800}{280}\right)(101.3)\text{A}^{0.65}(2.29 + \text{F}_\text{c}) \tag{C.1}$$

where $\text{A} = \frac{\text{Q}}{\text{U}\Delta\text{T}_\text{LM}}$ [=] ft^2 and $\text{F}_\text{c} = (\text{F}_\text{d} + \text{F}_\text{p})\text{F}_\text{m}$. For Heater 1, $\text{F}_c = 5.6$ to account for the higher pressure. All other heaters and coolers have an $\text{F}_c = 2.3$.[9]

C.2 Reactor Design and Installation Cost

The reactor is designed to be isothermal, therefore the amount of energy removed from the system is equal to the amount of heat added to the heater. The volume of the reactor was utilized to calculate the installed cost through the following correlation:

$$\text{Installed Cost, \$} = \left(\frac{1800}{280}\right)101.9\text{D}^{1.066}\text{H}^{0.82}(2.18 + \text{F}_\text{c}) \tag{C.2}$$

where D= 5.6 [=] ft [9], H= 8.4 [=] ft, $\text{F}_c = \text{F}_\text{m} * \text{F}_\text{p} = 5.6$ [9]

C.3 Separation Costs

Similar to the reactor, the installation costs for the flash drums and distillation columns are correlated by the diameter and the height of the unit modeled by Equation (C.2). The diameter was found directly using Aspen Hysys for the flash drums and distillation columns (measured in feet). With regards to the height, the flash drums also utilized Aspen Hysys, but the distillation columns used the tray spacing along with number of trays from Aspen Hysys to determine the column height. Additionally, the material specifications were selected to calculate by $\text{F}_c = \text{F}_\text{m,CS} + \text{F}_\text{p,CS} = 4.4$. [9]

Lastly, due to the complicated separation system from the DMC-MeOH and EC-EG azeotropes, the separation system requires a rigorous calculation to fully define across all reactor volumes and conversions. Therefore, besides the specified conditions in the report, the entire separation system is designed as a black box, therefore the minimum work, W_min, is used such that

$$\text{W}_\text{min} = \sum_{k=1}^{N}(\text{F}_k)\text{RT}\sum_{i=1}^{c}\left(\text{x}_i^k \ln\frac{\text{x}_i^k}{\text{z}_i}\right) + ... \tag{C.3}$$

where F_k is the flow rate of exiting stream k, x_i^k is the molar composition species i of exiting stream k, and z_i is the molar composition of the stream entering the separation system, for N streams. Equation (C.3) assumes constant temperature and pressure throughout the separation system. The separation system operating expenses (OPEX) and capital expenses (CAPEX) are respectively calculated by

$$\text{OPEX} = \epsilon\lambda\text{W}_{min} \tag{C.4}$$

$$\text{CAPEX} = \text{c}(\text{W}_{real}) \tag{C.5}$$

where energy cost ϵ [=] $\frac{\$}{J}$, efficiency factor $\lambda = 50$, and capital cost correction factor $\text{c} = 1$ [=] $\frac{\$}{\text{W}}$.

D Economic Assumptions, Formulas, Spreadsheets

In addition to the estimations for the separation costs, the economic assumptions in Tables (3) and (D.1) were used to calculate the TCI of the plant.

Table D.1: Assumptions for Total Fixed Capital Cost (TFCC). All values are in $MM

Cost	Assumptions
OSBL	0.4 (ISBL)
Contingency Fee (CF)	0.25 (ISBL + OSBL)
Indirect Costs (IC)	0.3 (ISBL + OSBL + Contingency Fee)
TFCC	**OSBL + CF + IC**
Working Capital (WC)	0.15 (TFCC)
Start-up Costs (SC)	0.1 (TFCC)
Land	0
TCI	**TFCC + (WC + SC + Land)**

After evaluating the capital costs, the following tables are used to evaluate the operating costs

Table D.2: Assumptions used to calculate VCOP in $\frac{\$\text{MM}}{\text{yr}}$

Revenue and Production Costs	Assumptions
Main Product Revenue (MPR)	$(\text{F}_{\text{Ethylene}})(\text{Price}_{\text{Ethylene}})$
Byproduct Revenue (BR)	$(\text{F}_{\text{Hydrogen}})(\text{Price}_{\text{Hydrogen}})$
Raw Materials Cost (COM)	$(\text{F}_{\text{Ethane}})(\text{Price}_{\text{Ethane}})$
Utilities Cost (UC)	$(\text{F}_{\text{Steam}})(\text{Price}_{\text{Steam}}) + (\text{F}_{\text{LPG}} + \text{F}_{\text{Methane}})(\text{Price}_{\text{Fuel}})$
CO_2 Sustainability Charge (CO_2 SC)	$[(\text{F}_{\text{Methane}} + \text{F}_{\text{Propane}} + \text{F}_{\text{Butane}})(\text{Price}_{\text{Fuel}})]\ \rho_{CO_2}$
VCOP	BR - COM + UC + (CO_2 SC)

where $\text{F}_i[=]\frac{\text{MT}}{\text{yr}}$ $\text{Price}_i[=]\frac{\$}{\text{MT}}$, $\text{F}_{\text{Fuel}}[=]\frac{\text{mol}}{\text{yr}}$, $\text{Price}_{\text{Fuel}}[=]\frac{\$}{\text{GJ}}$, $\rho_{CO_2}[=]\frac{\text{g}_{CO_2}}{\text{mol}}$

Table D.3: Assumptions used to calculate FCOP in $\frac{\$\text{MM}}{\text{yr}}$

Fixed Production Costs	Assumptions
Interest Rate (IR)	0.15 (FCI)
Administrative Costs (AGS)	0.05 (Total Revenue)
FCOP	Interest + AGS

The plant was designed with a 15 year project life, and a 3 year construction schedule, with fixed capital evenly distributed across the three years. Working capital and start up costs were priced at year 3, and a 10 year straight-line depreciation estimate was used to calculate depreciation. A 27% tax on taxable income, defined as gross profit minus depreciation, was used to calculate cash flow, and at the end of the project, the salvage value was assumed to be 5% of the FCI. The following two pages are the cash flow spreadsheets using the values from these assumptions and during a recession, respectively.

REVENUES AND PRODUCTION COSTS

	$MM/yr
Main product revenue	115.2
Byproduct revenue	39.5
Raw materials cost	113.7
Utilities cost	19.90
Consumables cost	0.0
CO2 sustainability charge	19.9
VCOP	114.0
Salary and overheads	0.0
Maintenance	0.0
Interest	0.0
AGS	7.7
FCOP	7.7

CAPITAL COSTS

	$MM
ISBL Capital Cost	37.2
OSBL Capital Cost	14.9
Indirect Cost	19.5
Contingency	13.0
Total Fixed Capital Cost	84.6
Working Capital	12.7
Start-up Costs	8.5
Land	5.0
Total Capital Investment	110.8

CONSTRUCTION SCHEDULE

Year	% FC	% WC	% SU	% FCOP	% VCOP
0					
1	33%				
2	33%				
3	33%	100%	100%		
4				100%	100%
5				100%	100%
6+				100%	100%

ECONOMIC ASSUMPTIONS

On Stream	8400 hr/yr	Discount Rate	15%	Depreciation method	Straight-Line
	350 day/yr	Tax Rate	27%	Depreciation period	10 yrs
Project Life	10 yr	Salvage Value	5%		

CASH FLOW ANALYSIS

All figures in $MM unless indicated

Project Year	Cap. Ex.	Revenue	COM	Gr. Profit	Deprcn.	Taxable Inc	Taxes Paid	Cash Flow	PV of CF	NPV
0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	-5.0	-5.0	-5.0
1	27.9	0.0	0.0	0.0	0.0	0.0	0.0	-27.9	-24.3	-29.3
2	27.9	0.0	0.0	0.0	0.0	0.0	0.0	-27.9	-21.1	-50.4
3	49.1	0.0	0.0	0.0	8.9	-8.9	-2.4	-46.7	-30.7	-81.1
4	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-1.4	-82.5
5	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-1.2	-83.6
6	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-1.0	-84.7
7	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.9	-85.6
8	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.8	-86.3
9	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.7	-87.0
10	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.6	-87.6
11	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.5	-88.1
12	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.4	-88.6
13	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.4	-89.0
14	0.0	115.2	121.8	-6.5	8.9	-15.4	-4.2	-2.4	-0.3	-89.3
15	-4.2	115.2	121.8	-6.5	8.9	-15.4	-4.2	1.9	0.3	-87.8

ECONOMIC ANALYSIS

NPV	15 years	-88.9	$MM	IRR	-56%
NPV at yr	12	-88.6	$MM		

NOTES

1. All cash flows are assumed to occur at the end of the project year.
2.
3.

Figure D.1: Year-over-year cashflow spreadsheet

Table D.4: Sensitivity Analysis Fluctuations

Adjusted Variable	Decreased Amount	NPV ($MM)	Increased Amount	NPV ($MM)
MeOH	$419/tonne	-79.1	$800/tonne	-128.9
EO	$960/tonne	-48.3	$1,670/tonne	-147.7
CO_2	$30/tonne	-87.0	$60/tonne	-90.7
DMC	$900/tonne	-140.5	$1,300/tonne	-34.8
EG	$420/tonne	-110.8	$920/tonne	-18.5
CapEx	$30MM	-76.0	$44MM	-101.1
OpEx	104.1$/yr	-62.7	124.1$/yr	-115.7
Construction Period	5 yr	-72.4	2 yr	-111.2
Discount Rate	10%	-99.5	20%	-80.4
Salvage Value	0%	-89.7	10%	-88.1

E Safety

E.1 Safety Data Sheet

Table E.1: Safety precautions for plant chemicals

Species	Flammability	Explosive Limits	Toxicology	Corrosiveness
Ethylene Oxide[15]	Extremely flammable gas (Category 1) Auto-ignition temperature: 429°C Flash point: -29°C	Upper explosion limit: 100% Lower explosion limit: 3%	TLV: 1 ppm 8 hours	Moderate eye irritant Skin irritant Toxic if inhaled May form explosive mixtures with air
Carbon Dioxide[16]	Not flammable Auto-ignition temperature: N/a Flash point: N/a	Upper explosion limit: N/a Lower explosion limit: N/a	TLV: 30,000 ppm 15 minutes	Contains gas under pressure; may explode if heated May displace oxygen and cause rapid suffocation May increase respiration and heart rate
Methanol[17]	Flammable liquids (Category 2) Auto-ignition temperature: 455°C Flash point: 9.7°C	Upper explosion limit: 44% Lower explosion limit: 6%	TLV: 250 ppm 15 minutes	Highly flammable liquid and vapor May displace oxygen and cause rapid suffocation Corrosive to the respiratory tract
Ethylene Carbonate[18]	Auto-ignition temperature: 465°C Flash point: 150°C	Upper explosion limit: 16.1% Lower explosion limit: 3.6%	TLV: N/a	Acute oral toxicity Serious eye damage specific target organ toxicity
Dimethyl Carbonate[19]	Flammable liquids (Category 2) Auto-ignition temperature: 458°C Flash point: 18°C	Upper explosion limits: 12.9 vol% Lower explosion limit: 4.2 vol%	TLV: N/a	Highly flammable liquid and vapor Moderate skin irritant Moderate eye irritant
Ethylene Glycol[20]	Extremely flammable gas (Category 1) Auto-ignition temperature: 413°C Flash point: 111°C	Upper explosion limit: 15.3 vol% Lower explosion limit: 3.2 vol%	TLV: 50 ppm (STEL)	Acute oral toxicity Specific target organ toxicity: CNS, Kidney, Liver
2-Methoxyethanol[21]	Flammable liquids (Category 3) Auto-ignition temperature: 285°C Flash point: 38°C	Upper explosion limits: 20 vol% Lower explosion limits: 1.8 vol%	TLV: 0.1 ppm (TWA)	Acute oral toxicity Acute dermal toxicity Acute inhalation toxicity Reproductive toxicity Specific target organ toxicity: Immune system, Thymus
Aniline[22]	Flammable liquids (Category 3) Auto-ignition temperature: 540°C Flash point: 76°C	Upper explosion limits: 11% Lower explosion limits: 1.3%	TLV: 5 ppm	Acute toxicity Acute dermal toxicity Acute oral toxicity Serious eye damage Environmentally damaging

E.2 Preliminary HAZOP

Table E.2: HAZOP Summary for Main Process Units

Process Unit	Deviation	Consequences	Safeguards	Relevant Chemicals and Safety Data
Pumps	Increased Flow Rate	Overloading downstream units, potential for cavitation, increased wear and tear	Flow control valves, relief valves	
	Decreased Flow Rate	Insufficient feed to downstream units, risk of pump damage due to low flow	Low-flow alarms, pump protection circuits	
Compressors	Increased Flow Rate	Overpressure in downstream equipment, increased energy consumption	Pressure relief valves, flow control systems	
	Decreased Flow Rate	Insufficient pressure buildup, reduced efficiency	Monitoring and control systems	
Heat Exchangers	Increased Temperature	Ethylene Oxide: Risk of explosion, moderate eye irritation, and skin irritant. Methanol: Increased risk of toxic fumes, may cause narcosis and blindness. Aniline: Risk of acute toxicity and severe eye damage.	Temperature control systems, safety interlocks, temperature alarms, emergency cooling systems	Ethylene Oxide, Methanol, Aniline
	Decreased Temperature	Inadequate heating/cooling, process inefficiencies	Temperature alarms, backup heating/cooling systems	
	Increased Pressure	Ethylene Oxide: Highly explosive, severe pressure risks. Dimethyl Carbonate: Risk of container rupture, mild to moderate skin irritant.	Pressure relief devices, robust monitoring systems, pressure relief valves	Ethylene Oxide, Dimethyl Carbonate
	Decreased Pressure	Potential for vacuum conditions, equipment failure	Vacuum breakers, pressure control systems	
Reactors	Increased Temperature	Ethylene Oxide: Risk of runaway reactions and thermal decomposition. Ethylene Glycol: Can cause kidney and liver toxicity, requires temperature monitoring.	Temperature control systems, emergency cooling, temperature monitoring and control	Ethylene Oxide, Ethylene Glycol
	Decreased Temperature	Reduced reaction rates, incomplete reactions	Temperature monitoring and control	
	Increased Pressure	2-Methoxyethanol: Acute toxicity, risk of explosion. Aniline: Risk of container rupture, environmental damage.	Pressure relief valves, burst discs, pressure monitoring and control systems	2-Methoxyethanol, Aniline
	Decreased Pressure	Incomplete reactions, vacuum conditions	Pressure monitoring and control systems	
Distillation Columns	Increased Flow Rate	Flooding of trays/packing, reduced separation efficiency	Flow control, monitoring systems	
	Decreased Flow Rate	Reduced throughput, poor separation	Flow control, monitoring systems	
	Increased Temperature	Methanol: Increased risk of toxic emissions, narcosis. Ethylene Carbonate: High toxicity, risk of serious eye damage.	Temperature control systems, reflux control, temperature alarms and control systems	Methanol, Ethylene Carbonate
	Decreased Temperature	Inadequate separation, higher energy consumption	Temperature control systems	
	Increased Pressure	Ethylene Oxide: Severe pressure risks, explosion hazards	Pressure relief systems, monitoring	Ethylene Oxide
	Decreased Pressure	Reduced boiling points, inefficient separation	Pressure control systems	

F Aspen HYSYS Simulation vs. MATLAB Conceptual Design

Table E.3: MATLAB process stream labels and specifications

Name	Temperature (°C)	Pressure (bar)	Molar Compositions	Mass Flow Rate (kg/hr)
Fresh Feed EO	25	1.0	1.0 EO	140
Fresh Feed CO_2	25	5.0	1.0 CO	140
Fresh Feed MeOH	25	1.0	1.0 MeOH	280
Recycle EC	250	74	1.0 EC	3.0
Recycle MeOH	140	74	1.0 MeOH	$1.8 * 10^3$
Recycle CO2	140	74	1.0 CO_2	$1.7 * 10^3$
Conversion Reactor Feed	140	74	$7.5 * 10^{-4}$ EC, $3.4 * 10^{-2}$ EO 0.44 CO_2, 0.52 MeOH	$4.0 * 10^3$
Separation System Feed	0	1.0	$3.5 * 10^{-2}$ EG, $1.1 * 10^{-3}$ EC, $3.8 * 10^{-2}$ DMC, 0.44 CO_2, 0.48 MeOH, $8.4 * 10^{-4}$ ME	$4.0 * 10^3$
Fresh Aniline	25	1.0	1.0 Aniline	0.65
Recycle Aniline	25	1.0	1.0 Aniline	$1.9 * 10^3$
DMC	25	1.0	1.0 DMC	130
ME	25	1.0	1.0 ME	3.1
EG Prod	25	1.0	1.0 EG	120
Water	25	1.0	1.0 Water	12
CO_2 Gas 3	140	74	1.0 CO_2	1.2
Waste Water	25	1.0	1.0	11
Product EG	25	1.0	1.0 EG	130

G Additional MATLAB Generated Figures

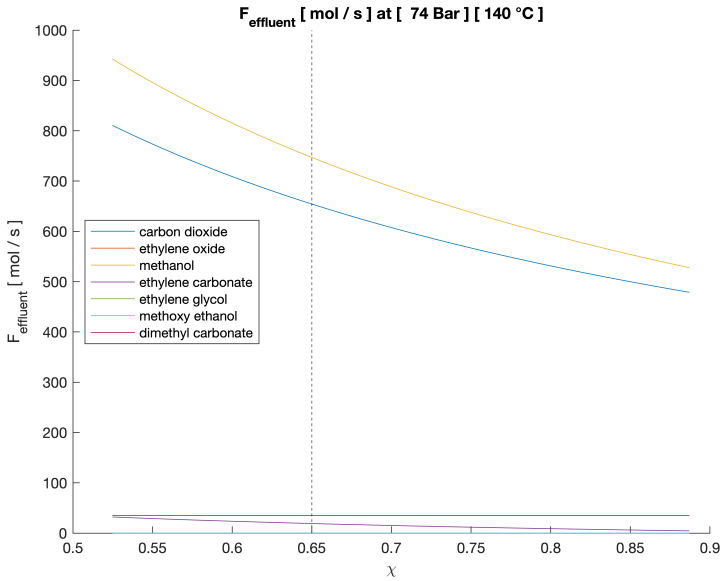


Figure G.1: Production rate of all products leaving the reactor

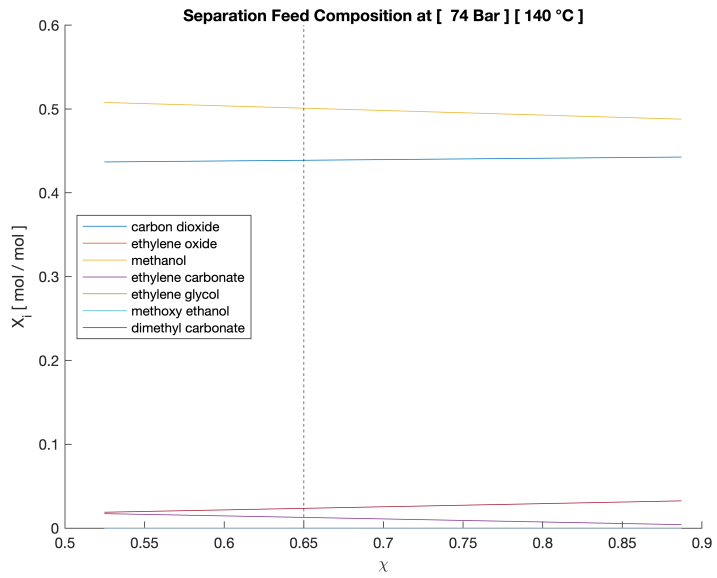


Figure G.2: Composition of all components entering the separation system

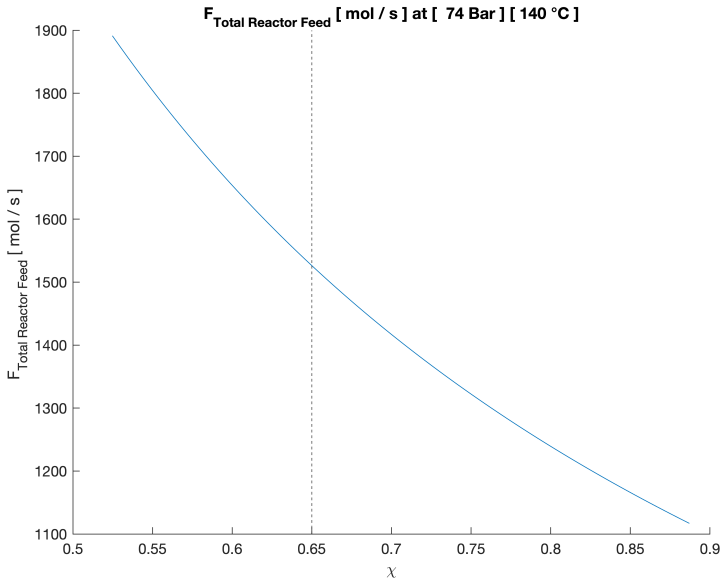


Figure G.3: Total molar flow rate into the separation system

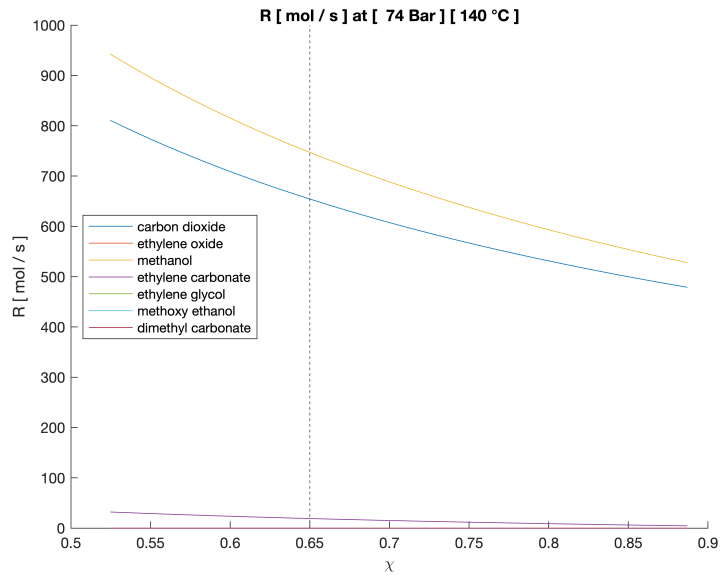


Figure G.4: Molar flow rates of the recycle stream

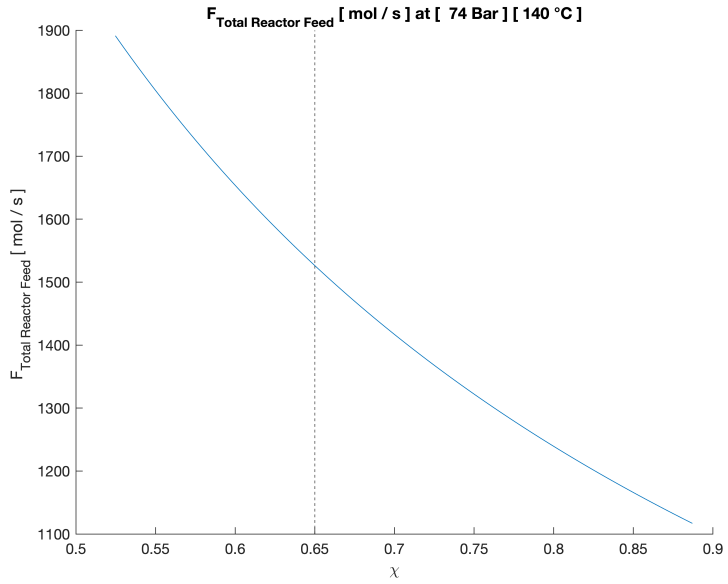


Figure G.5: Total molar flow rate of the reactor feed

H Commented Matlab Code

```
clc; clear; close all;

% SCRIPT_____________________________________________________________________
level3()

% FUNCTIONS__________________________________________________________________

function void = level3()
    void = NaN;
    P = 74; % bar
%     plt_fxns = plot_fxns();
%     level3_isothermal_aspen_compare(P);
    level3_isothermal(NaN);
    level3_isothermal(P);
    level3_isobaric();
    level3_isothermal(NaN);

end



function puppy = level3_isothermal_aspen_compare(P_specify)


    console = get_console();
    const = get_constants();
    user = get_user_inputs();
    F_fxns = flowrate_fxns();
    rxtr_fxns = reactor_fxns();
    plt_fxns = plot_fxns();
    econ_fxns = economic_fxns();
    T = user.level3.isothermal_temp.C;
    opt = 'isothermal';
    console.section("Starting Level 3 " + opt + " calculations")

    if ~isnan(P_specify)
        % i = 1;
        P = P_specify;
        % for P = user.level3.press_range
        console.subsection(sprintf("P = %3.2f", P), 1);
        row = 1;
        isoTherm_plt = plt_fxns.get_plot_struct(T, P, opt);

        for tau = user.level3.tau_range.P_specify.isothermal
            console.subsection(sprintf("tau = %3.2f", tau), 2)
            [F_fresh, F_rxtr, F_out, R, V_rxtr] = level3_flowrates(tau, T, P, opt);
            conversion = rxtr_fxns.get_conversion(F_rxtr, F_out);

            if isnan(conversion)
                disp("ERROR : COMPLEX CONC. BREAKING TO NEXT TEMP")
                break;
            end

            npv_opt = 'matlab';
            % npv_opt = 'matlab';
            npv = econ_fxns.get_work_min_npv(F_out, T, P, V_rxtr, conversion,↙
npv_opt);

```

```
58              % Store row data
59              plot_row.F_fresh = F_fresh;
60              plot_row.F_rxtr = F_rxtr;
61              plot_row.F_out = F_out;
62              plot_row.R = R;
63              plot_row.conversion = conversion;
64              plot_row.row_number = row;
65              plot_row.tau = tau;
66              plot_row.V_rxtr = V_rxtr ;
67              plot_row.npv = npv;
68              isoTherm_plt = plt_fxns.set_plot_row(isoTherm_plt, plot_row);
69              % increment
70              row = row + 1;
71          end
72
73          % all_pressure_data(i) = isoTherm_plt;
74          % i = i + 1;
75          % plt_fxns.plot_selectivity(isoTherm_plt);
76          % plt_fxns.plot_effluent_composition(isoTherm_plt);
77          % plt_fxns.plot_total_separation_feed(isoTherm_plt);
78          % plt_fxns.plot_total_reactor_feed(isoTherm_plt);
79          % plt_fxns.plot_recycle_flowrates(isoTherm_plt);
80          % plt_fxns.plot_fresh_feed_conversion(isoTherm_plt);
81          % plt_fxns.plot_molar_flowrates_conversion(isoTherm_plt);
82          % plt_fxns.plot_npv(isoTherm_plt);
83          plt_fxns.plot_npv_with_aspen_data(isoTherm_plt);
84
85      end
86      console.section("Level 3 " + opt + " calculations are complete")
87  end
88
89
90  function void = level3_isothermal(P_specify)
91      void = NaN;
92
93      console = get_console();
94      const = get_constants();
95      user = get_user_inputs();
96      F_fxns = flowrate_fxns();
97      rxtr_fxns = reactor_fxns();
98      plt_fxns = plot_fxns();
99      econ_fxns = economic_fxns();
100     T = user.level3.isothermal_temp.C;
101     opt = 'isothermal';
102     console.section("Starting Level 3 " + opt + " calculations")
103
104     if ~isnan(P_specify)
105         i = 1;
106         P = P_specify;
107         % for P = user.level3.press_range
108         console.subsection(sprintf("P = %3.2f", P), 1);
109         row = 1;
110         isoTherm_plt = plt_fxns.get_plot_struct(T, P, opt);
111
112         for tau = user.level3.tau_range.P_specify.isothermal
113             console.subsection(sprintf("tau = %3.2f", tau), 2)
114             [F_fresh, F_rxtr, F_out, R, V_rxtr] = level3_flowrates(tau, T, P, opt);
115             conversion = rxtr_fxns.get_conversion(F_rxtr, F_out);
```

```
116
117             if isnan(conversion)
118                 disp("ERROR : COMPLEX CONC. BREAKING TO NEXT TEMP")
119                 break;
120             end
121
122             % npv_opt = 'aspen';
123             npv_opt = 'matlab';
124             npv = econ_fxns.get_work_min_npv(F_out, T, P, V_rxtr, conversion,↙
npv_opt);
125
126             % Store row data
127             plot_row.F_fresh = F_fresh;
128             plot_row.F_rxtr = F_rxtr;
129             plot_row.F_out = F_out;
130             plot_row.R = R;
131             plot_row.conversion = conversion;
132             plot_row.row_number = row;
133             plot_row.tau = tau;
134             plot_row.V_rxtr = V_rxtr ;
135             plot_row.npv = npv;
136             isoTherm_plt = plt_fxns.set_plot_row(isoTherm_plt, plot_row);
137             % increment
138             row = row + 1;
139         end
140
141         all_pressure_data(i) = isoTherm_plt;
142         i = i + 1;
143         % end
144         plt_fxns.plot_selectivity(isoTherm_plt);
145         plt_fxns.plot_effluent_composition(isoTherm_plt);
146         plt_fxns.plot_total_separation_feed(isoTherm_plt);
147         plt_fxns.plot_total_reactor_feed(isoTherm_plt);
148         plt_fxns.plot_recycle_flowrates(isoTherm_plt);
149         plt_fxns.plot_fresh_feed_conversion(isoTherm_plt);
150         plt_fxns.plot_molar_flowrates_conversion(isoTherm_plt);
151         plt_fxns.plot_npv(isoTherm_plt);
152         plt_fxns.plot_npv_with_aspen_data(isoTherm_plt);
153     else
154         i = 1;
155         for P = user.level3.press_range
156             console.subsection(sprintf("P = %3.2f", P), 1);
157             row = 1;
158             isoTherm_plt = plt_fxns.get_plot_struct(T, P, opt);
159
160
161             for tau = user.level3.tau_range.isothermal
162                 console.subsection(sprintf("tau = %3.2f", tau), 2)
163                 [F_fresh, F_rxtr, F_out, R, V_rxtr] = level3_flowrates(tau, T, P,↙
opt);
164                 conversion = rxtr_fxns.get_conversion(F_rxtr, F_out);
165
166                 if isnan(conversion)
167                     disp("ERROR : COMPLEX CONC. BREAKING TO NEXT TEMP")
168                     break;
169                 end
170                 npv_opt = 'matlab';
171                 npv = econ_fxns.get_work_min_npv(F_out, T, P, V_rxtr, conversion,↙
```

```
npv_opt);
172
173                 % Store row data
174                 plot_row.F_fresh = F_fresh;
175                 plot_row.F_rxtr = F_rxtr;
176                 plot_row.F_out = F_out;
177                 plot_row.R = R;
178                 plot_row.conversion = conversion;
179                 plot_row.row_number = row;
180                 plot_row.tau = tau;
181                 plot_row.V_rxtr = V_rxtr ;
182                 plot_row.npv = npv;
183                 isoTherm_plt = plt_fxns.set_plot_row(isoTherm_plt, plot_row);
184                 % increment
185                 row = row + 1;
186             end
187
188             all_pressure_data(i) = isoTherm_plt;
189             i = i + 1;
190         end
191         plt_fxns.plot_npv_all_pressures(all_pressure_data);
192         plt_fxns.plot_reactor_volume_conversion_allP(all_pressure_data);
193
194     end
195     console.section("Level 3 " + opt + " calculations are complete");
196 end
197
198 function [F_fresh, F_rxtr, F_out, R, V_rxtr] = level3_flowrates(tau, temp, P, opt)
199     F_fresh = NaN; F_rxtr = NaN; F_out = NaN; R = NaN;
200     user = get_user_inputs();
201     flow_fxns = flowrate_fxns();
202     rxtr_fxns = reactor_fxns();
203
204     F_basis = flow_fxns.get_basis_feed_flowrates();
205     [F_fresh, F_rxtr, F_out, R, V_rxtr] = rxtr_fxns.get_reactor_flows(F_basis, temp,↙
P, opt, tau);
206
207
208 end
209
210
211
212 function void = level3_isobaric()
213     void = NaN;
214
215     console = get_console();
216     const = get_constants();
217     user = get_user_inputs();
218     F_fxns = flowrate_fxns();
219     rxtr_fxns = reactor_fxns();
220     plt_fxns = plot_fxns();
221     econ_fxns = economic_fxns();
222
223     P = user.level3.isobaric_press.bar;
224     opt = 'isobaric';
225     console.section("Starting Level 3 " + opt + " calculations")
226
227     i = 1;
```

```
228     for T = user.level3.temp_range
229         console.subsection(sprintf("T = %3.2f", T), 1);
230         row = 1;
231         isoBar_plt = plt_fxns.get_plot_struct(T, P, opt);
232 
233 
234         for tau =  user.level3.tau_range.isobaric
235             console.subsection(sprintf("tau = %3.2f", tau), 2)
236             [F_fresh, F_rxtr, F_out, R, V_rxtr] = level3_flowrates(tau, T, P, opt);
237             conversion = rxtr_fxns.get_conversion(F_rxtr, F_out);
238 
239             if isnan(conversion)
240                 disp("ERROR : COMPLEX CONC. BREAKING TO NEXT TEMP")
241                 break;
242             end
243 
244             npv_opt = 'matlab';
245             npv = econ_fxns.get_work_min_npv(F_out, T, P, V_rxtr, conversion,↙
npv_opt);
246 
247             % Store row data
248             plot_row.F_fresh = F_fresh;
249             plot_row.F_rxtr = F_rxtr;
250             plot_row.F_out = F_out;
251             plot_row.R = R;
252             plot_row.conversion = conversion;
253             plot_row.row_number = row;
254             plot_row.tau = tau;
255             plot_row.V_rxtr = V_rxtr;
256             plot_row.npv = npv;
257             isoBar_plt = plt_fxns.set_plot_row(isoBar_plt, plot_row);
258             % increment
259             row = row + 1;
260         end
261 
262         all_temp_data(i) = isoBar_plt;
263         i = i + 1;
264     end
265 
266     plt_fxns.plot_reactor_volume_conversion_allT(all_temp_data);
267 
268     console.section("Level 3 " + opt + " calculations are complete")
269 end
270 
271 
272 
```

```
 1
 2 function fxns = separation_fxns()
 3     fxns.get_work_min = @get_work_min;
 4
 5 end
 6
 7 function w_min = get_work_min(F, T)
 8     % Assumes that all separation products streams are pure
 9     % w_min has units of watts (J / s)
10     % input T is in Celcius
11     const = get_constants();
12     R = const.thermo.R;
13     T = const.units.temperature.c_to_k(T);
14
15     w_min = 0;
16     fieldNames = fieldnames(F);
17     for i = 1:length(fieldNames)
18         species = fieldNames{i};
19         if strcmp(species, 'ethylene_oxide'), continue, end
20         x = 1; % assumes that each separated flowstream is pure
21         z = F.(species).x;
22         F_i = F.(species).mol;
23         w_min = w_min + (F_i * R * T * x * log(x/z));
24
25     end
26 end
27
28
29
30 function [sep_top1, sep_btm1] = flash_v100(sep)
31
32     sep.heat = 0;
33
34     K.ethane = 3.760 * 10^9;
35     K.ethylene = 7.266 * 10^8;
36     K.hydrogen = 3.193 * 10^6;
37     K.methane = 8.488 * 10^7;
38     K.propane = 5.252 * 10^11;
39     K.butane = 3.978 * 10^14;
40     K.water = 1.561 * 10^-2;
41
42     [sep_top1, sep_btm1]= rachford_rice(sep, K);
43
44 end
45
46 function phi = underwood(z, r, s, alpha, y, x, q)
47     % y are the distillate compositions
48     % alpha has the relative volatilities
49     % r is reflux ratio
50     % s is boilup ratio
51     r_min_factor = 1.2;
52
53     % Doherty & Malone eq 4.21
54     eqn_421_top = @(phi, r) -r - 1 + (alpha.a * y.a / (alpha.a - phi)) + (alpha.b *↙
y.b / (alpha.b - phi));
55     eqn_421_bot = @(phi, s) s + (alpha.a * x.a / (alpha.a - phi)) + (alpha.b * x.b /↙
(alpha.b - phi));
56
```

```
 57     init_phi = alpha.b + (alpha.a / 2);
 58         % alpha_a > phi > alpha_b
 59     phi_1_top =  fzero( @(phi) eqn_421_top(phi, r), init_phi);
 60
 61     init_phi = alpha.b / 2;
 62         % alpha_b > phi > 0
 63     phi_2_top = fzero( @(phi) eqn_421_top(phi,r), init_phi);
 64
 65     % Doherty and Malone eq 4.25
 66     term1 = alpha.a * z.a / (alpha.a - phi_2_top);
 67     term2 = alpha.b * z.b / (alpha.b - phi_2_top);
 68     term3 = alpha.a * z.a / (alpha.a - phi_1_top);
 69     term4 = alpha.b * z.b / (alpha.b - phi_1_top);
 70     trays_above_feed = log((term1 + term2) / (term3 + term4)) / log(phi_1_top /↙
phi_2_top);
 71
 72
 73     init_phi = alpha.a * 2 ;
 74     init_phi = alpha.b * (0.5 * (alpha.a - alpha.b));
 75         % alpha_a > phi > alpha_b
 76     phi_2_bar =  fzero( @(phi) eqn_421_bot(phi, s), init_phi);
 77
 78     init_phi = alpha.a * 1.5;
 79         % inf > phi > alpha_a
 80     phi_1_bar = fzero( @(phi) eqn_421_bot(phi, s), init_phi);
 81
 82     term1 = alpha.a * z.a / (alpha.a - phi_1_bar);
 83     term2 = alpha.b * z.b / (alpha.b - phi_1_bar);
 84     term3 = alpha.a * z.a / (alpha.a - phi_2_bar);
 85     term4 = alpha.b * z.b / (alpha.b - phi_2_bar);
 86     trays_below_feed = log((term1 + term2) / (term3 + term4)) / log(phi_1_bar /↙
phi_2_bar);
 87
 88     % Doherty and Malone eq4.29
 89     find_theta = @(theta) q - 1 + (alpha.a * z.a / (alpha.a - theta)) + ...
 90                                   (alpha.b * z.b / (alpha.b - theta));
 91     init_theta = phi_1_top + (phi_2_bar - phi_1_top)*0.5; % ???
 92     theta = fzero( @(theta) find_theta(theta), init_theta);
 93
 94     r_min = -1 + (alpha.a * y.a / (alpha.a - theta)) + (alpha.b * y.b / (alpha.b -↙
theta));
 95     r = r_min_factor * r_min; % WHAT MULTIPLE OF R_MIN SHOULD WE USE?
 96     phi = 0;
 97 end
 98
 99
100 function W = compressor_work_TJ(sep, P_f)
101     R = 8.314;      % [ J / mol K]
102     n = total_molar_flowrate(sep.F);
103     T = sep.T;
104     P_0 = sep.P;
105
106     W = - n * R * T * log10(P_f / P_0); % ?? I think that it's base 10
107
108
109     % ?? THIS ALWAYS RETURNS 0 OR NULL, NOT IMPLEMENTED YET
110 end
111
```

```
112
113
114 function T_f = adiabatic_temp(T_0, P_0, P_f)
115
116     T_f = T_0 * ( P_0 / P_f);
117 end
118
119
120
121 function [sep_top, sep_btm]= rachford_rice(sep, K)
122     global KT_PER_G MOLMASS_BUTANE MOLMASS_ETHANE MOLMASS_ETHYLENE MOLMASS_HYDROGEN↙
MOLMASS_METHANE MOLMASS_PROPANE MOLMASS_WATER
123
124     sep.x = all_mol_fractions(sep.F);
125
126     f_phi = @(phi, sep, K) ((sep.x.methane * (K.methane - 1)) / (1 + phi*(K.methane -↙
1))) + ...
127         ((sep.x.ethane * (K.ethane - 1)) / (1 + phi*(K.ethane - 1))) + ...
128         ((sep.x.ethylene * (K.ethylene - 1)) / (1 + phi*(K.ethylene - 1))) + ...
129         ((sep.x.hydrogen * (K.hydrogen - 1)) / (1 + phi*(K.hydrogen - 1))) + ...
130         ((sep.x.propane * (K.propane - 1)) / (1 + phi*(K.propane - 1))) + ...
131         ((sep.x.butane * (K.butane - 1)) / (1 + phi*(K.butane - 1))) + ...
132         ((sep.x.water * (K.water - 1)) / (1 + phi*(K.water - 1)));
133
134     init_cond = 0.5;
135
136     phi = fzero(@(phi) f_phi(phi, sep, K), init_cond);
137
138     if phi > 1
139         phi = 1;
140     elseif phi < 0
141         phi = 0;
142     end
143
144
145     % Liquid compositions
146     x.hydrogen = sep.x.hydrogen / (1 + phi*(K.hydrogen - 1));
147     x.methane = sep.x.methane / (1 + phi*(K.methane - 1));
148     x.ethane = sep.x.ethane / (1 + phi*(K.ethane - 1));
149     x.ethylene = sep.x.ethylene / (1 + phi*(K.ethylene - 1));
150     x.propane = sep.x.propane / (1 + phi*(K.propane - 1));
151     x.butane = sep.x.butane / (1 + phi*(K.butane - 1));
152     x.water = sep.x.water / (1 + phi*(K.water - 1));
153
154     % Vapor compositions
155     y.hydrogen = K.hydrogen * x.hydrogen;
156     y.methane = K.methane * x.methane;
157     y.ethane = K.ethane * x.ethane;
158     y.ethylene = K.ethylene * x.ethylene;
159     y.propane = K.propane * x.propane;
160     y.butane = K.butane * x.butane;
161     y.water = K.water * x.water;
162
163     % Splitting
164     F_tot = total_molar_flowrate(sep.F);
165     V = phi * F_tot;
166     L = (1 - phi) * F_tot;
167
```

```
168     % Tops
169     sep_top = sep;
170     sep_top.y = y;
171     sep_top.x = NaN;
172
173
174     % kta             = (mol/yr) * (mol / mol) * (g / mol)   * (kt / g)
175     sep_top.F.hydrogen = V * y.hydrogen * (MOLMASS_HYDROGEN) * KT_PER_G;
176     sep_top.F.methane = V * y.methane * (MOLMASS_METHANE) * KT_PER_G;
177     sep_top.F.ethane = V * y.ethane * (MOLMASS_ETHANE) * KT_PER_G;
178     sep_top.F.ethylene = V * y.ethylene * (MOLMASS_ETHYLENE) * KT_PER_G;
179     sep_top.F.propane = V * y.propane * (MOLMASS_PROPANE) * KT_PER_G;
180     sep_top.F.butane = V * y.butane * (MOLMASS_BUTANE) * KT_PER_G;
181     sep_top.F.water = V * y.water * (MOLMASS_WATER) * KT_PER_G;
182
183     % Bottoms
184     sep_btm = sep;
185     sep_btm.x = x;
186     sep_btm.y = NaN;
187
188     % kta     = (mol/yr) * (mol / mol) * (g / mol)   * (kt / g)
189     sep_btm.F.hydrogen = L * x.hydrogen * (MOLMASS_HYDROGEN) * KT_PER_G;
190     sep_btm.F.methane = L * x.methane * (MOLMASS_METHANE) * KT_PER_G;
191     sep_btm.F.ethane = L * x.ethane * (MOLMASS_ETHANE) * KT_PER_G;
192     sep_btm.F.ethylene = L * x.ethylene * (MOLMASS_ETHYLENE) * KT_PER_G;
193     sep_btm.F.propane = L * x.propane * (MOLMASS_PROPANE) * KT_PER_G;
194     sep_btm.F.butane = L * x.butane * (MOLMASS_BUTANE) * KT_PER_G;
195     sep_btm.F.water = L * x.water * (MOLMASS_WATER) * KT_PER_G;
196
197
198 end
```

```
clc; clear; close all ;

% test_level2_feedstream()
% test_units()
% test_constants()

% test_scripts()
% disp("running main script")
% run_dmc

function void = find_V_reactor_bug()
    void = NaN;
    const = get_constants();



end

function void = test_scripts()
    void = NaN;

    disp("running all scripts")
    flowrate_fxns
    get_console
    get_constants
    get_user_inputs
    reactor_fxns
    % separation_fxns
    % heat_exchange_fxns
    economic_fxns

end

function void = test_units()
    const = get_constants();
    console = get_console();

    fprintf("TESTING UNIT CONVERSIONS%s\n",  console.divider);
    fprintf("\tmass %s\n",  console.divider);
    const.units.mass

    fprintf("\tenergy %s\n",  console.divider);
    const.units.energy

    fprintf("\ttemperature %s\n",  console.divider);
    const.units.temperature

    fprintf("\tvalue %s\n",  console.divider);
    const.units.value

    fprintf("\tpressure %s\n",  console.divider);
    const.units.pressure

    fprintf("\ttime %s\n",  console.divider);
    const.units.time

    fprintf("\tvolume %s\n",  console.divider);
    const.units.volume
```

```
 59
 60      fprintf("\theat %s\n",  console.divider);
 61      const.units.heat
 62
 63  end
 64
 65
 66  function void = test_constants()
 67
 68      disp("const")
 69      const = get_constants();
 70
 71      test_units(const.units);
 72      % test_molar_mass(const.molar_mass);
 73      % test_stoich(const.stoich);
 74      % test_thermo(const.thermo);
 75      % test_econ(const.econ);
 76
 77  end
 78
 79  function void = test_level2_feedstream()
 80
 81      const = get_constants();
 82
 83      user = get_user_inputs();
 84
 85      F_fxns = flowrate_fxns();
 86
 87      console = get_console();
 88
 89      F = user.level2.feed_stream;
 90
 91      fprintf("LEVEL 2 FEEDSTREAM FLOWRATES%s\n",  console.divider);
 92      fprintf("Carbon Dioxide \n\t%4.3f kta\t %4.3f mol /s\n",F.carbon_dioxide.kta, F.↙
carbon_dioxide.mol)
 93      fprintf("Ethylene Oxide \n\t%4.3f kta\t %4.3f mol /s\n", F.ethylene_oxide.kta, F.↙
ethylene_oxide.mol);
 94      fprintf("Methanol \n\t%4.3f kta\t %4.3f mol /s\n", F.methanol.kta, F.methanol.↙
mol);
 95      fprintf("Ethylene Carbonate \n\t%4.3f kta\t %4.3f mol /s\n", F.↙
ethylene_carbonate.kta, F.ethylene_carbonate.mol);
 96      fprintf("Ethylene Glycol \n\t%4.3f kta\t %4.3f mol /s\n", F.ethylene_glycol.kta,↙
F.ethylene_glycol.mol);
 97      fprintf("Methoxy Ethanol \n\t%4.3f kta\t %4.3f mol /s\n", F.methoxy_ethanol.kta,↙
F.methoxy_ethanol.mol);
 98
 99  end
100
101
102
```

```
% Reactor Simulation Functions

function fxns = reactor_fxns()
    % fxns.get_reaction_rate = @get_reaction_rate;
    fxns.get_reactor_flows = @get_reactor_flows;
    fxns.get_conversion = @get_conversion;
end

function [F_fresh, F_real_feed, F_real_eff, R, V_rxtr] = get_reactor_flows(F_real_feed_basis, T, P, opt, tau)
    F_fresh = NaN; F_real_feed = NaN; F_real_eff = NaN; R = NaN;
    % basis calculations for the real reactor
    q_tot.basis = get_total_volumetric_flowrate(F_real_feed_basis, T, P, opt);
    V_rxtr.basis = q_tot.basis * tau;
    C_out = get_reactor_effluent_concentrations(F_real_feed_basis, T, P, opt, tau);

    if any(imag(C_out) ~= 0)
        % disp('ERROR : Complex valued concentrations');
        return
    elseif any(real(C_out) < 0)
        % disp('ERROR : Negative valued concentrations');
        return
    else
        % disp('Valid solution!!!!!!!!!!')
        C_out = get_concentration_struct(C_out);
    end

    F_real_eff_basis = conc_to_flowrate(C_out, q_tot.basis);
        % assumption : liquid flow has no change in vol in effluent

    % Plant Scale Calculations
    [F_fresh, F_real_feed, F_real_eff, R] = get_plant_flowrates(F_real_feed_basis, F_real_eff_basis);
    scale_factor = get_scale_factor(F_real_eff_basis);
    V_rxtr.plant = V_rxtr.basis * scale_factor ;

    V_rxtr = V_rxtr.plant;

end

function chi = get_conversion(F_real_feed, F_real_eff)
    if ~isstruct(F_real_feed) || ~isstruct(F_real_eff), chi = NaN;, return, end;
    if ~isreal(F_real_feed.ethylene_carbonate.mol) || ~isreal(F_real_eff.ethylene_carbonate.mol), chi = NaN;, end
    chi = ( F_real_feed.ethylene_carbonate.mol - F_real_eff.ethylene_carbonate.mol) / ...
            F_real_feed.ethylene_carbonate.mol;
end


function [F_fresh, F_real_feed, F_real_effluent, R] = get_plant_flowrates(F_real_feed_basis, F_real_eff_basis)
    scale_factor = get_scale_factor(F_real_eff_basis);

    F_real_effluent = get_scaled_flowrate(F_real_eff_basis, scale_factor);
    F_real_feed = get_scaled_flowrate(F_real_feed_basis, scale_factor);

    F_virt_feed = get_virtual_reactor_feed(F_real_feed);
```

```
 54      [F_fresh, R] = get_recycle_and_fresh_flowrates(F_virt_feed, F_real_effluent);
 55
 56  end
 57
 58  function F_scaled  = get_scaled_flowrate(F_basis, scale_factor)
 59      % Conserves mass by scaling the mass flowrates
 60
 61      flow_fxns = flowrate_fxns();
 62      F_scaled = flow_fxns.get_blank_flowstream();
 63      fieldNames = fieldnames(F_basis);
 64      for i = 1:length(fieldNames)
 65          if strcmp(fieldNames{i},'units') , continue, end ;
 66          F_scaled.(fieldNames{i}).kta = scale_factor * F_basis.(fieldNames{i}).kta;
 67      end
 68      F_scaled = flow_fxns.set_F_kta(F_scaled);
 69  end
 70
 71  function F_virt_feed = get_virtual_reactor_feed(F_virtual_effluent)
 72      user = get_user_inputs();
 73      flow_fxns = flowrate_fxns();
 74      F_virt_feed = flow_fxns.get_blank_flowstream();
 75
 76      % Assume that C02 is in excess
 77      % Get the flow into the virtual reactor
 78      F_virt_feed.ethylene_oxide.mol = F_virtual_effluent.ethylene_carbonate.mol;
 79          % Assume that EO -> EC Complete conversion in virtual reactor
 80      F_virt_feed.methanol.mol = F_virt_feed.ethylene_oxide.mol * user.level3.↙
molar_ratio_methanol_EO;
 81      F_virt_feed.carbon_dioxide.mol = F_virt_feed.ethylene_oxide.mol * user.level3.↙
molar_ratio_carbon_dioxide_EO;
 82      F_virt_feed = flow_fxns.set_F_mol(F_virt_feed);
 83  end
 84
 85  function [F_fresh, R] = get_recycle_and_fresh_flowrates(F_virt_feed, F_real_effluent)
 86      flow_fxns = flowrate_fxns();
 87
 88      % Initialize
 89      R = flow_fxns.get_blank_flowstream();
 90      F_fresh = flow_fxns.get_blank_flowstream();
 91
 92      % Recycle flow
 93      R.ethylene_carbonate.mol = F_real_effluent.ethylene_carbonate.mol;
 94      R.methanol.mol = F_real_effluent.methanol.mol;
 95      R.carbon_dioxide.mol = F_real_effluent.carbon_dioxide.mol;
 96      R = flow_fxns.set_F_mol(R);
 97
 98      % Fresh feed flow
 99      F_fresh.ethylene_oxide.mol = F_virt_feed.ethylene_oxide.mol - R.↙
ethylene_carbonate.mol;
100      F_fresh.methanol.mol = F_virt_feed.methanol.mol - R.methanol.mol;
101      F_fresh.carbon_dioxide.mol = F_virt_feed.carbon_dioxide.mol - R.carbon_dioxide.↙
mol;
102      F_fresh = flow_fxns.set_F_mol(F_fresh);
103
104      % F_fresh.ethylene_oxide.mol = F_fresh.ethylene_carbonate.mol;
105      % F_fresh.ethylene_carbonate.mol = 0;
106          % EC should be turned back into EO because we need the feed into the virtual↙
reactor
```

```
107         % ?? Look into more detail of the EO / EC and recycle stream because the
108         % VR really complicates things
109
110
111 end
112
113
114 function scale_factor = get_scale_factor(F_out)
115     scale_factor = (get_user_inputs().dmc_production_rate) / F_out.↙
dimethyl_carbonate.kta;
116
117 end
118
119 function F = conc_to_flowrate(C, q_tot)
120     % ?? Assumption : densities don't change after reaction (which will change
121     % the T and P, thus densities should change) modify q_tot(T, P) to get a more
122     % accurate result
123     flow_fx = flowrate_fxns();
124     F = flow_fx.get_blank_flowstream();
125
126     fieldNames = fieldnames(C);
127     for i = 1:length(fieldNames)
128         if strcmp(fieldNames{i},'units') , continue, end ;
129
130         % mol / s         = ( L / s )         * (mol / L)
131         F.(fieldNames{i}).mol = q_tot * C.(fieldNames{i});
132     end
133
134     F = flow_fx.set_F_mol(F);
135 end
136
137 function C_vector = get_concentration_vector(F, T, P, opt, tau)
138     C = get_concentrations(F, T, P, opt, tau);
139     C_vector(1) = C.ethylene_carbonate;
140     C_vector(2) = C.ethylene_glycol;
141     C_vector(3) = C.methanol;
142     C_vector(4) = C.carbon_dioxide;
143     C_vector(5) = C.dimethyl_carbonate;
144     C_vector(6) = C.methoxy_ethanol;
145 end
146
147 function k = get_all_rate_constants(T, P, opt)
148     k.k2f = get_rate_constant('2f', T, P, opt);
149     k.k2r = get_rate_constant('2r', T, P, opt);
150     k.k3 = get_rate_constant('3', T, P, opt);
151 end
152
153 function C = get_reactor_effluent_concentrations(F, T, P, opt, tau)
154     tau = tau / 10;
155     user = get_user_inputs();
156     Ci0 = get_concentrations(F, T, P, opt, tau);
157     Ci_init = get_concentration_vector(F, T, P, opt, tau);
158     params.Ci0 = Ci0;
159     params.tau = tau;
160     params.T = T;
161     params.P = P;
162     params.opt = opt;
163
```

```
164     eqns = @(C) sys_of_eqns(C, params);
165     C = fsolve(eqns, Ci_init, user.level3.fsolveOpt);
166 end
167 
168 function C_struct = get_concentration_struct(C_vector)
169     C_struct.ethylene_carbonate = C_vector(1);
170     C_struct.ethylene_glycol = C_vector(2);
171     C_struct.methanol = C_vector(3);
172     C_struct.carbon_dioxide = C_vector(4);
173     C_struct.dimethyl_carbonate = C_vector(5);
174     C_struct.methoxy_ethanol = C_vector(6);
175 end
176 
177 function eqn = sys_of_eqns(C, params)
178     T = params.T;
179     P = params.P;
180     opt = params.opt;
181     Ci0 = params.Ci0;
182     tau = params.tau;
183     C_struct = get_concentration_struct(C);
184     r = get_all_reaction_rates(C_struct, T, P, opt);
185 
186     r.ec =  r.r2r - r.r2f - r.r3;
187     r.meoh = (2 * r.r2r) - (2 * r.r2f) - r.r3;
188     r.co2 = r.r3;
189     r.dmc = r.r2f - r.r2r;
190     r.eg = r.r2f - r.r2r;
191     r.me = r.r3;
192 
193     eqn(1) = Ci0.ethylene_carbonate - C(1) + (tau * r.ec);
194     eqn(2) = Ci0.methanol - C(3) + (tau * r.meoh);
195     eqn(3) = Ci0.carbon_dioxide - C(4) + (tau * r.co2);
196     eqn(4) = (-C(5)) + (tau * r.dmc);
197     eqn(5) = (-C(2)) + (tau * r.eg);
198     eqn(6) = (-C(6)) + (tau * r.me);
199 end
200 
201 function r = get_all_reaction_rates(C, T, P, opt)
202     r.r2f = get_reaction_rate(C, '2f', T, P, opt);
203     r.r2r = get_reaction_rate(C, '2r', T, P, opt);
204     r.r3 = get_reaction_rate(C, '3', T, P, opt);
205 end
206 
207 function r = get_reaction_rate(C, reaction, T, P, opt)
208     % input:
209     %   opt = 'isothermal' or 'isobaric'
210 
211     k = get_rate_constant(reaction, T, P, opt);
212     switch reaction
213         case '2f'
214             r = k * (C.ethylene_carbonate)^0.8;
215         case '2r'
216             r = k * C.dimethyl_carbonate * C.ethylene_carbonate;
217         case '3'
218             r = k * C.ethylene_carbonate;
219         otherwise
220             r = NaN;
221             disp("ERROR: get_reaction_rate(): invalid reaction option")
```

```
222     end
223 end
224
225 function C = get_concentrations(F, T, P, opt, tau)
226     V = get_reactor_volume(F, T, P, opt, tau);
227     fieldNames = fieldnames(F);
228     for i = 1:length(fieldNames)
229         C.(fieldNames{i}) = F.(fieldNames{i}).mol * tau / V;
230     end
231 end
232
233 function V = get_reactor_volume(F, T, P, opt, tau)
234     q_tot = get_total_volumetric_flowrate(F, T, P, opt);
235     V = q_tot * tau;
236 end
237
238 function rho = get_all_molar_densities(T, P, opt)
239     rho = get_all_densities(T, P, opt);
240     const = get_constants();
241
242     fieldNames = fieldnames(rho);
243     for i = 1:length(fieldNames)
244         if strcmp(fieldNames{i},'units')
245             rho.(fieldNames{i}) = 'mol / L';
246             continue
247         end
248
249         % mol / L              = (kg / m^3)              * (g / kg)
250         rho.(fieldNames{i}) = rho.(fieldNames{i}) * const.units.mass.g_per_kg * ...
251         ...% (mol / g)                                 * (m^3 / L)
252         (1 / const.molar_mass.(fieldNames{i})) * const.units.volume.m3_per_l;
253     end
254 end
255
256 function rho = get_all_densities(T, P, opt)
257     % out is in kg / m^3
258
259     rho = get_constants().densities;
260     rho.methanol = get_methanol_density(T, P);
261     rho.carbon_dioxide = get_supercritical_c02_density(T, P, opt);
262 end
263
264 function q_total = get_total_volumetric_flowrate(F, T, P, opt)
265     q = get_volumetric_flowrates(F, T, P, opt);
266     % output [L / s]
267     q_total.value = 0;
268     fieldNames = fieldnames(q);
269     for i = 1:length(fieldNames)
270         if strcmp(fieldNames{i},'units')
271             q_total.units = q.units;
272             continue
273         end
274         q_total.value = q_total.value +  q.(fieldNames{i});
275     end
276     q_total = q_total.value;
277 end
278
279 function q = get_volumetric_flowrates(F, T, P, opt)
```

```
280     const = get_constants();
281     rho = get_all_densities(T, P, opt);
282
283     % % Carbon dioxide
284     % % (L / s)        = (mol / s)            * (g / mol)
285     % q.carbon_dioxide = F.carbon_dioxide.mol * const.molar_mass.carbon_dioxide * ...
286     %     ...% (kg / g)        * (m^3 / kg)             * (L / m^3)
287     %     const.units.mass.kg_per_g * (1/rho.carbon_dioxide) * const.units.volume.↙
l_per_m3;
288     % % Methanol
289     % q.methanol = F.methanol.mol * const.molar_mass.methanol * ...
290     %     const.units.mass.kg_per_g * (1 / rho.methanol) * const.units.volume.↙
l_per_m3;
291     % % Ethylene Carbonate
292     % q.ethylene_carbonate = F.ethylene_carbonate.mol * const.molar_mass.↙
ethylene_carbonate *...
293     %      const.units.mass.kg_per_g * (1 / rho.ethylene_carbonate) * const.units.↙
volume.l_per_m3;
294
295
296     fieldNames = fieldnames(F);
297     for i = 1:length(fieldNames)
298         if strcmp(fieldNames{i},'units')
299             % rho.(fieldNames{i}) = 'mol / L';
300             continue
301         end
302         % (L / s)        = (mol / s)            * (g / mol)
303         q.(fieldNames{i}) = F.(fieldNames{i}).mol * const.molar_mass.(fieldNames{i})↙
* ...
304             ...% (kg / g)            * (m^3 / kg)             * (L / m^3)
305             const.units.mass.kg_per_g * (1/rho.(fieldNames{i})) * const.units.volume.↙
l_per_m3;
306     end
307     q.units = 'L / s' ;
308
309 end
310
311 function k = get_rate_constant(reaction, T, P, opt)
312     if strcmp(opt, 'isothermal')
313         k = get_isothermal_rate_constant(reaction, T, P);
314     elseif strcmp(opt, 'isobaric')
315         k = get_isobaric_rate_constant(reaction, T);
316     else
317         k = NaN;
318         disp("ERROR: get_rate_constant(): invalid opt");
319     end
320
321 end
322
323 function k = get_isobaric_rate_constant(reaction, T)
324     % input:
325     %   T [ C ]
326     % output:
327     %   k [mol / L s ]
328
329     const = get_constants();
330     thermo = const.thermo;
331     T = const.units.temperature.c_to_k(T); % [ K ]
```

```
332
333     switch reaction
334         case '2f'
335             k = 6.69 * 10^2 * exp(-37200 / (thermo.R * T));
336         case '2r'
337             k = 1.19 * 10^4 * exp(-53700 / (thermo.R * T));
338         case '3'
339             k = 1.89 * 10^6 * exp(-82400 / (thermo.R * T));
340         otherwise
341             k = NaN;
342             disp("ERROR: get_isobaric_rate_constant(): invalid reaction option");
343     end
344 end
345
346 function k = get_isothermal_rate_constant(reaction, T, P)
347     % input:
348     %   P [ bar ]
349     % These functions are from the research paper
350     rho = get_supercritical_c02_density(T, P, 'isothermal');
351     switch reaction
352         case '2f'
353             if rho > 246.82 % [g / L]
354                 k = (2.486 * 10^(-2)) - (4.943 * (10^(-5)) * rho);
355             else
356                 k = (1.362 * 10^(-2)) - (1.569 * (10^(-6)) * rho);
357             end
358         case '2r'
359             k = 0.01486 * rho^(-0.873);
360         case '3'
361             k = 3.014 * (10^(-4)) * exp(-5.99 * (10^(-3)) * rho);
362         otherwise
363             k = NaN;
364             disp("ERROR: get_isothermal_rate_constant(): invalid reaction option")
365     end
366 end
367
368 function rho = get_supercritical_c02_density(T, P, opt)
369     % Input: condition = T or P. Depending on option
370     %   P [=] bar
371     %   T [=] celcius
372     % Assumptions:
373     %   Isobaric model is at 150 bar
374     %   Isothermal model is at 140 C
375     % Ranges of input
376     %   P = [50 bar, 150 bar]
377     %   T = [80 C, 140 C]
378     % Output:
379     %   rho [=] kg / m^3
380     withinTempRange = @(T) T >= 80 && T <= 140;
381     withinPressureRange = @(P) P >= 50 && P <= 150;
382
383     rho.units = 'kg / m^3';
384     switch opt
385         case 'isothermal'
386             if withinPressureRange(P)
387                 rho = 1.6746 * P - 12.592;
388                     % NIST / Excel Regression
389             else
```

```
390                 rho = NaN;
391                 disp("get_supercritical_c02_density : ERROR : P out of range")
392             end
393         case 'isobaric'
394             if withinTempRange(T)
395                 if P < 125 % [ Bar ]
396                     rho = 356.08 * exp(-0.006 * T);
397                         % NIST Data at 100 bar
398                 else
399                     rho = 838.87 * exp(-0.009 * T);
400                         % NIST Data at 150 bar
401                 end
402             else
403                 rho = NaN;
404                 disp("get_supercritical_c02_density : ERROR : T out of range")
405             end
406         otherwise
407             disp("SUPERCRITICAL C02 DENSITY FUNCTION ERROR: invalid opt")
408             rho = NaN;
409     end
410 end
411
412 function rho = get_methanol_density(T, P)
413     % Input:
414     %   P [=] bar
415     %   T [=] celcius
416     % Assumptions:
417     %   Isobaric model is at 140 C
418     %   Isothermal model is at 150 bar
419     % Ranges of input
420     %   P = [50 bar, 150 bar]
421     %   T = [80 C, 140 C]
422     % Output:
423     %   rho [=] kg / m^3
424     withinTempRange = @(T) T >= 80 && T <= 140;
425     % withinPressureRange = @(P) P >= 50 && P <= 150;
426     rho.units = "kg / m^3";
427     if withinTempRange(T)%  && withinPressureRange(P)
428         if P < 125; % [C]
429             rho = -1.0866 * T + 833.31;
430                 % NIST Data at 100 bar
431         else
432             rho = -1.0354 * T + 834.79;
433                 % NIST Data at 150 bar
434         end
435     else
436         rho = NaN;
437         disp("get_methanol_density : ERROR : T or P out of range")
438     end
439
440 end
441
```

```
 1
 2
 3
 4 function fxns = plot_fxns()
 5
 6     fxns.get_empty_vector = @get_empty_vector;
 7     fxns.get_plot_struct = @get_plot_structure;
 8     fxns.set_plot_row = @set_plot_row;
 9     fxns.plot_reactor_volume_conversion = @plot_reactor_volume_conversion;
10     fxns.plot_reactor_volume_conversion_allT = @plot_reactor_volume_conversion_allT;
11     fxns.plot_reactor_volume_conversion_allP = @plot_reactor_volume_conversion_allP;
12     fxns.delete_old_plots = @delete_old_plots;
13     fxns.plot_molar_flowrates_conversion = @plot_molar_flowrates_conversion;
14     fxns.plot_fresh_feed_conversion = @plot_fresh_feed_conversion;
15     fxns.plot_recycle_flowrates = @plot_recycle_flowrates;
16     fxns.plot_effluent_composition = @plot_effluent_composition;
17     fxns.plot_total_reactor_feed = @plot_total_reactor_feed;
18     fxns.plot_total_separation_feed = @plot_total_separation_feed;
19     fxns.plot_selectivity = @plot_selectivity;
20     fxns.plot_npv = @plot_npv;
21     fxns.plot_npv_with_aspen_data = @plot_npv_with_aspen_data;
22     fxns.plot_npv_all_pressures = @plot_npv_all_pressures;
23
24 end
25
26 function void = plot_npv_with_aspen_data(plot_struct)
27     void = NaN;
28     user = get_user_inputs();
29     const = get_constants();
30     flow_fxns = flowrate_fxns();
31
32     aspen_data = economic_fxns().get_aspen_datapoints();
33
34     figure
35     hold on
36     % F_total = flow_fxns.get_total_flowrate(plot_struct.data.F_out, 'mol');
37     % fieldNames = fieldnames(plot_struct.data.F_rxtr);
38     % for i = 1:length(fieldNames)
39         x = plot_struct.data.conversion(:);
40         % y = plot_struct.data.F_out.(fieldNames{i}).x(:);
41         y = plot_struct.data.npv(:) / 15;
42
43         % figure
44         plot(x, y);
45         title(sprintf('Nominal Present Value (with HYSYS data) [ %3.0f Bar ] [ %3.0f↙
°C ]',plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
46         xlabel('\chi', 'Interpreter', 'tex');
47         ylabel('NPV [ $MM ]', 'Interpreter', 'tex');
48         % Create the legend entry for this plot
49         % legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
50     % end
51     % The design variable point
52     % legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.↙
isothermal.x_point, ...
53                                 % (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
54     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
```

```
 55     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
 56     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
 57     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
 58
 59     % Add Legend
 60     % legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
 61     hold off
 62
 63     fileName =  "isothermal_npv_"+ string(plot_struct.P) + "Bar";
 64     save_plot(fileName);
 65 end
 66
 67
 68
 69 function void = plot_npv(plot_struct)
 70     void = NaN;
 71     user = get_user_inputs();
 72     const = get_constants();
 73     flow_fxns = flowrate_fxns();
 74     figure
 75     hold on
 76     % F_total = flow_fxns.get_total_flowrate(plot_struct.data.F_out, 'mol');
 77     % fieldNames = fieldnames(plot_struct.data.F_rxtr);
 78     % for i = 1:length(fieldNames)
 79         x = plot_struct.data.conversion(:);
 80         % y = plot_struct.data.F_out.(fieldNames{i}).x(:);
 81         y = plot_struct.data.npv(:) / 15;
 82
 83         % figure
 84         plot(x, y);
 85         title(sprintf('Nominal Present Value at [ %3.0f Bar ] [ %3.0f °C ]',↙
plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
 86         xlabel('\chi', 'Interpreter', 'tex');
 87         ylabel('NPV [ $MM ]', 'Interpreter', 'tex');
 88         % Create the legend entry for this plot
 89         % legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
 90     % end
 91     % The design variable point
 92     % legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.↙
isothermal.x_point, ...
 93                                 % (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
 94     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
 95     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
 96     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
 97     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
 98
 99     % Add Legend
100     % legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
101     hold off
102
```

```
103     fileName =  "isothermal_npv_"+ string(plot_struct.P) + "Bar";
104     save_plot(fileName);
105 end
106
107
108 function void = plot_selectivity(plot_struct)
109     void = NaN;
110     user = get_user_inputs();
111     const = get_constants();
112     flow_fxns = flowrate_fxns();
113     figure
114     hold on
115     % F_total = flow_fxns.get_total_flowrate(plot_struct.data.F_out, 'mol');
116     S = plot_struct.data.F_out.dimethyl_carbonate.mol(:) ./ plot_struct.data.F_fresh.↙
ethylene_oxide.mol(:);
117         x = plot_struct.data.conversion(:);
118         % y = plot_struct.data.F_out.(fieldNames{i}).x(:);
119         y = S(:);
120
121         % figure
122         plot(x, y);
123         title(sprintf('Selectivity at [ %3.0f Bar ] [ %3.0f °C ]',plot_struct.P,↙
plot_struct.T), 'Interpreter', 'tex');
124         xlabel('\chi', 'Interpreter', 'tex');
125         ylabel('Selectivity', 'Interpreter', 'tex');
126         % Create the legend entry for this plot
127         % legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
128     % end
129     % The design variable point
130     % legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.↙
isothermal.x_point, ...
131                                 % (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
132     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
133     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
134     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
135     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
136
137     % Add Legend
138     % legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
139     hold off
140
141     fileName =  "isothermal_selectivity" + string(plot_struct.P) + "Bar";
142     save_plot(fileName);
143
144
145 end
146
147 function void = plot_total_separation_feed(plot_struct)
148     void = NaN;
149     user = get_user_inputs();
150     const = get_constants();
151     flow_fxns = flowrate_fxns();
152     figure
```

```
153     hold on
154     F_total = flow_fxns.get_total_flowrate(plot_struct.data.F_out, 'mol');
155     % fieldNames = fieldnames(plot_struct.data.F_rxtr);
156     % for i = 1:length(fieldNames)
157         x = plot_struct.data.conversion(:);
158         % y = plot_struct.data.F_out.(fieldNames{i}).x(:);
159         y = F_total(:);
160 
161         % figure
162         plot(x, y);
163         title(sprintf('F_{Total Separator Feed} [ mol / s ] at [ %3.0f Bar ] [ %3.0f↙
°C ]',plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
164         xlabel('\chi', 'Interpreter', 'tex');
165         ylabel('F_{Total Separator Feed} [ mol / s ] ', 'Interpreter', 'tex');
166         % Create the legend entry for this plot
167         % legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
168     % end
169     % The design variable point
170     % legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.↙
isothermal.x_point, ...
171                                 % (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
172     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
173     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
174     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
175     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
176 
177     % Add Legend
178     % legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
179     hold off
180 
181     fileName =  "isothermal_F_separator_total" + string(plot_struct.P) + "Bar";
182     save_plot(fileName);
183 end
184 
185 function void = plot_total_reactor_feed(plot_struct)
186     void = NaN;
187     user = get_user_inputs();
188     const = get_constants();
189     flow_fxns = flowrate_fxns();
190     figure
191     hold on
192     F_total = flow_fxns.get_total_flowrate(plot_struct.data.F_rxtr, 'mol');
193     % fieldNames = fieldnames(plot_struct.data.F_rxtr);
194     % for i = 1:length(fieldNames)
195         x = plot_struct.data.conversion(:);
196         % y = plot_struct.data.F_out.(fieldNames{i}).x(:);
197         y = F_total(:);
198 
199         % figure
200         plot(x, y);
201         title(sprintf('F_{Total Reactor Feed} [ mol / s ] at [ %3.0f Bar ] [ %3.0f °C↙
]',plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
202         xlabel('\chi', 'Interpreter', 'tex');
```

```
203         ylabel('F_{Total Reactor Feed} [ mol / s ] ', 'Interpreter', 'tex');
204         % Create the legend entry for this plot
205         % legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
206     % end
207     % The design variable point
208     % legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.↙
isothermal.x_point, ...
209                                 % (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
210     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
211     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
212     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
213     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
214
215     % Add Legend
216     % legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
217     hold off
218
219     fileName =  "isothermal_F_rxtr_total" + string(plot_struct.P) + "Bar";
220     save_plot(fileName);
221 end
222
223
224 function void = plot_effluent_composition(plot_struct)
225     void = NaN;
226     user = get_user_inputs();
227     const = get_constants();
228     flow_fxns = flowrate_fxns();
229     figure
230     hold on
231     fieldNames = fieldnames(plot_struct.data.F_out);
232     % plot_struct.data.F_out = flow_fxns.set_mol_fractions(plot_struct.data.F_out);
233     for i = 1:length(fieldNames)
234         x = plot_struct.data.conversion(:);
235         y = plot_struct.data.F_out.(fieldNames{i}).x(:);
236
237         % figure
238         plot(x, y);
239         title(sprintf('X_i [ mol / mol ] at [ %3.0f Bar ] [ %3.0f °C ]',plot_struct.↙
P, plot_struct.T), 'Interpreter', 'tex');
240         xlabel('\chi', 'Interpreter', 'tex');
241         ylabel('X_i [ mol / mol ]', 'Interpreter', 'tex')
242         % Create the legend entry for this plot
243         legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
244     end
245     % The design variable point
246     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.isothermal.↙
x_point, ...
247                                 (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
248     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
249     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
```

```
250     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
251     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
252
253     % Add Legend
254     legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
255     hold off
256
257     fileName =  "isothermal_effluent_composition_" + string(plot_struct.P) + "Bar";
258     save_plot(fileName);
259 end
260
261 function void = plot_recycle_flowrates(plot_struct)
262     void = NaN;
263     user = get_user_inputs();
264     const = get_constants();
265     figure
266     hold on
267     fieldNames = fieldnames(plot_struct.data.F_out);
268     for i = 1:length(fieldNames)
269         x = plot_struct.data.conversion(:);
270         y = plot_struct.data.R.(fieldNames{i}).mol(:);
271
272         % figure
273         plot(x, y);
274         title(sprintf('R [ mol / s ] at [ %3.0f Bar ] [ %3.0f °C ]',plot_struct.P,↙
plot_struct.T), 'Interpreter', 'tex');
275         xlabel('\chi', 'Interpreter', 'tex');
276         ylabel('R [ mol / s ]', 'Interpreter', 'tex')
277         % Create the legend entry for this plot
278         legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
279     end
280     % The design variable point
281     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.isothermal.↙
x_point, ...
282                                 (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
283     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
284     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
285     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
286     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
287
288     % Add Legend
289     legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
290     hold off
291
292     fileName =  "isothermal_recycle_" + string(plot_struct.P) + "Bar";
293     save_plot(fileName);
294 end
295
296 function void = plot_fresh_feed_conversion(plot_struct)
297     void = NaN;
298     user = get_user_inputs();
```

```
299     const = get_constants();
300     figure
301     hold on
302     fieldNames = fieldnames(plot_struct.data.F_out);
303     for i = 1:length(fieldNames)
304         x = plot_struct.data.conversion(:);
305         y = plot_struct.data.F_fresh.(fieldNames{i}).mol(:);
306 
307         % figure
308         plot(x, y);
309         title(sprintf('F_{fresh feed} [ mol / s ] at [ %3.0f Bar ] [ %3.0f °C ]',↙
plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
310         xlabel('\chi', 'Interpreter', 'tex');
311         ylabel('F_{fresh feed} [ mol / s ]', 'Interpreter', 'tex')
312         % Create the legend entry for this plot
313         legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
314     end
315     % The design variable point
316     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.isothermal.↙
x_point, ...
317                                    (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
318     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
319     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
320     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
321     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
322 
323     % Add Legend
324     legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
325     hold off
326 
327     fileName =  "isothermal_F_fresh_feed_" + string(plot_struct.P) + "Bar";
328     save_plot(fileName);
329 end
330 
331 function void = plot_molar_flowrates_conversion(plot_struct)
332     void = NaN;
333     user = get_user_inputs();
334     const = get_constants();
335     figure
336     hold on
337     fieldNames = fieldnames(plot_struct.data.F_out);
338     for i = 1:length(fieldNames)
339         x = plot_struct.data.conversion(:);
340         y = plot_struct.data.F_out.(fieldNames{i}).mol(:);
341 
342         % figure
343         plot(x, y);
344         title(sprintf('F_{effluent} [ mol / s ] at [ %3.0f Bar ] [ %3.0f °C ]',↙
plot_struct.P, plot_struct.T), 'Interpreter', 'tex');
345         xlabel('\chi', 'Interpreter', 'tex');
346         ylabel('F_{effluent} [ mol / s ]', 'Interpreter', 'tex')
347         % Create the legend entry for this plot
348         legendEntries{i} = sprintf('%s', strrep(fieldNames{i}, '_', " "));
```

```
349     end
350     % The design variable point
351     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.isothermal.↙
x_point, ...
352                                    (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
353     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
354     % yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
355     % plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
356     %             'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
357
358     % Add Legend
359     legend(legendEntries, 'Interpreter', 'tex', 'location', 'west');
360     hold off
361
362     fileName =  "isothermal_F_effluent_" + string(plot_struct.P) + "Bar";
363     save_plot(fileName);
364 end
365
366 function void = save_plot(fileName)
367     user = get_user_inputs();
368     if ispc
369         dir = [pwd  '\plots\' ];
370     elseif ismac
371         dir = [pwd  '/plots/' ];
372     else
373         dir = [pwd  '/plots/' ];
374     end
375
376     if ~exist(dir, 'dir')
377         mkdir(dir);
378     end
379
380     delete_png_if_exists(fileName);
381     print(fullfile(dir, fileName), '-dpng', user.plot.image_dpi) ;  % Save as PNG↙
with 300 DPI
382 end
383
384 function void = delete_png_if_exists(fileName)
385     % Check platform and set directory
386     if ispc
387         dirName = [pwd  '\plots\' ];
388     elseif ismac
389         dirName = [pwd  '/plots/' ];
390     else
391         dirName = [pwd  '/plots/' ];
392     end
393
394     % Construct the full file path
395     filePath = fullfile(dirName, fileName);
396
397     % Check if the file exists and delete if it does
398     if exist(filePath, 'file') == 2 % 2 indicates it is a file
399         delete(filePath);
```

```
400     end
401 end
402
403 function void = delete_old_plots()
404     % disp("test ")
405     if ispc
406         dirName = [pwd  '\plots\' ];
407     elseif ismac
408         dirName = [pwd  '/plots/' ];
409     else
410         dirName = [pwd  '/plots/' ];
411     end
412
413     pngFiles = dir(fullfile(dirName, '*.png'));
414
415     for k = 1:length(pngFiles)
416         filePath = fullfile(pngFiles(k).folder, pngFiles(k).name);
417         delete(filePath);
418     end
419
420 end
421
422
423 function void = plot_npv_all_pressures(all_pressure_data)
424
425     user = get_user_inputs();
426     const = get_constants();
427     figure
428     hold on
429     for i = 1:length(all_pressure_data)
430         plot_struct = all_pressure_data(i);
431         x = plot_struct.data.conversion(:);
432         % y = plot_struct.data.V_rxtr(:) .* const.units.volume.m3_per_l;
433         y = plot_struct.data.npv(:) / 10^1;
434
435         % figure
436         plot(x, y);
437         title(sprintf(' NPV $MM [ %3.0f °C ]', plot_struct.T), 'Interpreter', 'tex');
438         xlabel('\chi', 'Interpreter', 'tex');
439         ylabel(' NPV [ $MM ] ', 'Interpreter', 'tex')
440         % Create the legend entry for this plot
441         legendEntries{length(all_pressure_data) - i + 1} = sprintf('%3.0f Bar',↙
plot_struct.P);
442     end
443     % The design variable point
444     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f $MM' , user.plot.isothermal.↙
x_point, ...
445                                 (user.plot.isothermal.y_point ));
446     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
447     yline(user.plot.isothermal.y_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
448     plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point, ...
449                 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
450
451     % Add Legend
452     legend(legendEntries, 'Interpreter', 'tex', 'location', 'northwest');
```

```
453     hold off
454
455     fileName =  "isothermal_NPV_all_pressures";
456     save_plot(fileName);
457
458
459 end
460
461 function void = plot_reactor_volume_conversion_allP(all_pressure_data)
462     user = get_user_inputs();
463     const = get_constants();
464     figure
465     hold on
466     for i = 1:length(all_pressure_data)
467         plot_struct = all_pressure_data(i);
468         x = plot_struct.data.conversion(:);
469         y = plot_struct.data.V_rxtr(:) .* const.units.volume.m3_per_l;
470
471         % figure
472         plot(x, y);
473         title(sprintf('V_{reactor} [ m^3 ] at [ %3.0f °C ]', plot_struct.T),↙
'Interpreter', 'tex');
474         xlabel('\chi', 'Interpreter', 'tex');
475         ylabel('V_{reactor} [ m^3 ]', 'Interpreter', 'tex')
476         % Create the legend entry for this plot
477         legendEntries{i} = sprintf('%3.0f Bar', plot_struct.P);
478     end
479     % The design variable point
480     legendEntries{i + 1} = sprintf('\\chi = %0.2f, %3.1f m^3' , user.plot.isothermal.↙
x_point, ...
481                                    (user.plot.isothermal.y_point * const.units.volume.↙
m3_per_l));
482     xline(user.plot.isothermal.x_point, '--k', 'LineWidth', 0.5, 'HandleVisibility',↙
'off');
483     yline(user.plot.isothermal.y_point * const.units.volume.m3_per_l, '--k',↙
'LineWidth', 0.5, 'HandleVisibility', 'off');
484     plot(user.plot.isothermal.x_point, user.plot.isothermal.y_point * const.units.↙
volume.m3_per_l, ...
485                 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize',↙
3);
486
487     % Add Legend
488     legend(legendEntries, 'Interpreter', 'tex', 'location', 'northwest');
489     hold off
490
491     fileName =  "isothermal_V_reactor";
492     save_plot(fileName);
493 end
494
495
496 function void = plot_reactor_volume_conversion_allT(all_temp_data)
497     user = get_user_inputs();
498     const = get_constants();
499     figure
500     hold on
501     for i = 1:length(all_temp_data)
502         plot_struct = all_temp_data(i);
503         x = plot_struct.data.conversion(:) ;
```

```
504         y = plot_struct.data.V_rxtr(:) * const.units.volume.m3_per_l;
505
506         % figure
507         plot(x, y);
508         title(sprintf('V_{reactor} [ m^3 ] [ %3.0f Bar ]', plot_struct.P),↙
'Interpreter', 'tex');
509         xlabel('\chi', 'Interpreter', 'tex');
510         ylabel('V_{reactor} [ m^3 ]', 'Interpreter', 'tex')
511         % Create the legend entry for this plot
512         legendEntries{i} = sprintf('%3.0f°C', plot_struct.T);
513     end
514     legend(legendEntries, 'Interpreter', 'tex', 'location', 'northwest');
515     hold off
516
517     % if ispc
518     %     dir = [pwd  '\plots\' ];
519     % elseif ismac
520     %     dir = [pwd  '/plots/' ];
521     % else
522     %     dir = [pwd  '/plots/' ];
523     % end
524
525     % if ~exist(dir, 'dir')
526     %     mkdir(dir);
527     % end
528
529     % print(fullfile(dir, "isobaric_V_reactor"), '-dpng', user.plot.image_dpi) ;  %↙
Save as PNG with 300 DPI
530
531     fileName =   "isobaric_V_reactor";
532     save_plot(fileName);
533 end
534
535
536
537 function void = plot_reactor_volume_conversion(plot_struct)
538
539     x = plot_struct.data.conversion(:);
540     y = plot_struct.data.V_rxtr(:);
541
542     hold on
543     figure
544     plot(x, y);
545     title(sprintf('V_{rxtr} [L] vs \\chi %3.0f [C] %3.0f [Bar]', plot_struct.T,↙
plot_struct.P), 'Interpreter', 'tex');
546
547     xlabel('\chi', 'Interpreter', 'tex');
548     ylabel('V_{rxtr} [L]', 'Interpreter', 'tex')
549     hold off
550 end
551
552 function plot_struct = set_plot_row(plot_struct, plot_row)
553
554     if isnan(plot_row.conversion)
555         return
556     end
557
558     fields = {'carbon_dioxide', 'ethylene_oxide', 'methanol', 'ethylene_carbonate',↙
```

```
'ethylene_glycol', 'methoxy_ethanol', 'dimethyl_carbonate'};
559     subfields = {'mol', 'kta', 'x'};
560
561     % Iterate over F_fresh
562     for i = 1:length(fields)
563         for j = 1:length(subfields)
564             plot_struct.data.F_fresh.(fields{i}).(subfields{j})(plot_row.row_number,↙
1) = plot_row.F_fresh.(fields{i}).(subfields{j});
565         end
566     end
567
568     % Iterate over F_rxtr
569     for i = 1:length(fields)
570         for j = 1:length(subfields)
571             plot_struct.data.F_rxtr.(fields{i}).(subfields{j})(plot_row.row_number,↙
1) = plot_row.F_rxtr.(fields{i}).(subfields{j});
572         end
573     end
574
575     % Iterate over F_out
576     for i = 1:length(fields)
577         for j = 1:length(subfields)
578             plot_struct.data.F_out.(fields{i}).(subfields{j})(plot_row.row_number, 1)↙
= plot_row.F_out.(fields{i}).(subfields{j});
579         end
580     end
581
582     % Iterate over R
583     for i = 1:length(fields)
584         for j = 1:length(subfields)
585             plot_struct.data.R.(fields{i}).(subfields{j})(plot_row.row_number, 1) =↙
plot_row.R.(fields{i}).(subfields{j});
586         end
587     end
588
589     % reactor data
590     plot_struct.data.conversion(plot_row.row_number) = plot_row.conversion;
591     plot_struct.data.tau(plot_row.row_number) = plot_row.tau;
592     plot_struct.data.V_rxtr(plot_row.row_number) = plot_row.V_rxtr;
593     plot_struct.data.npv(plot_row.row_number) = plot_row.npv;
594
595 end
596
597 function plot_struct = get_plot_structure(T, P, opt)
598     user = get_user_inputs();
599     % Thermodynamic state
600     plot_struct.T = T;
601     plot_struct.P = P;
602     plot_struct.opt = opt;
603
604     fields = {'carbon_dioxide', 'ethylene_oxide', 'methanol', 'ethylene_carbonate',↙
'ethylene_glycol', 'methoxy_ethanol', 'dimethyl_carbonate'};
605     subfields = {'mol', 'kta'};
606     categories = {'F_fresh', 'F_rxtr', 'F_out', 'R'};
607
608     % Initialize the fields with get_empty_vector
609     for c = 1:length(categories)
610         for i = 1:length(fields)
```

```
611             for j = 1:length(subfields)
612                 plot_struct.data.(categories{c}).(fields{i}).(subfields{j}) =↙
get_empty_vector(length(user.level3.tau_range));
613             end
614         end
615     end
616 
617     % Initialize NaN values for the fields
618     for c = 1:length(categories)
619         for i = 1:length(fields)
620             for j = 1:length(subfields)
621                 plot_struct.data.(categories{c}).(fields{i}).(subfields{j})(:) = NaN;
622             end
623         end
624     end
625 
626     % Initialize reactor data
627     plot_struct.data.conversion = get_empty_vector(length(user.level3.tau_range));
628     plot_struct.data.tau = get_empty_vector(length(user.level3.tau_range));
629     plot_struct.data.V_rxtr = get_empty_vector(length(user.level3.tau_range));
630     plot_struct.data.npv = get_empty_vector(length(user.level3.tau_range));
631 
632     % Reactor Data
633     plot_struct.data.conversion(:) = NaN;
634     plot_struct.data.tau(:) = NaN;
635     plot_struct.data.V_rxtr(:) = NaN;
636     plot_struct.data.npv(:) = NaN;
637 
638     % Plot info (maybe delete ??)
639     plot_struct.name.title = '';
640     plot_struct.name.y = '';
641     plot_struct.name.x = '';
642     % plot_struct.length = NaN;
643     plot_struct.color = '';
644 end
645 
646 % function plot_struct = get_plot_structure()
647 %     plot_struct.name.title = '';
648 %     plot_struct.name.y = '';
649 %     plot_struct.name.x = '';
650 %     plot_struct.length = NaN;
651 %     plot_struct.color = '';
652 % end
653 
654 function vector = get_empty_vector(length)
655     vector = zeros(length, 1);
656 
657 end
658 
```

```
 1
 2 function Cp_avg = avg_heat_capacity(F)
 3     global HEAT_CAPACITY_HYDROGEN HEAT_CAPACITY_METHANE HEAT_CAPACITY_ETHANE ...
 4         HEAT_CAPACITY_ETHYLENE HEAT_CAPACITY_PROPANE HEAT_CAPACITY_BUTANE ...
 5         HEAT_CAPACITY_WATER
 6
 7     % weighted average of Cp's
 8     F_tot = total_molar_flowrate(F);
 9
10     Cp_avg = (molar_flowrate(F, 'hydrogen') / F_tot) * HEAT_CAPACITY_HYDROGEN + ...
11         (molar_flowrate(F, 'methane') / F_tot) * HEAT_CAPACITY_METHANE + ...
12         (molar_flowrate(F, 'ethane') / F_tot) * HEAT_CAPACITY_ETHANE + ...
13         (molar_flowrate(F, 'ethylene') / F_tot) * HEAT_CAPACITY_ETHYLENE + ...
14         (molar_flowrate(F, 'propane') / F_tot) * HEAT_CAPACITY_PROPANE + ...
15         (molar_flowrate(F, 'butane') / F_tot) * HEAT_CAPACITY_BUTANE + ...
16         (molar_flowrate(F, 'water') / F_tot) * HEAT_CAPACITY_WATER;
17
18 end
19
20 function sep = hex(sep, T_out)
21     % Temperatures must be in kelvin
22     global GJ_PER_KJ
23
24     % GJ/yr     =  (mol / yr)              * (kJ / mol K )              * ( K    -↙
K   ) * (GJ / kJ)
25     sep.heat = total_molar_flowrate(sep.F) * avg_heat_capacity(sep.F) * (T_out - sep.↙
T) * GJ_PER_KJ;
26     sep.T = T_out;
27 end
28
29 function sep = hex_e101(sep)
30     global GJ_PER_KJ
31
32     % user inputs
33     T_out = 25 + 273.15;    % [ K ]
34
35     % GJ/yr     =  (mol / yr)              * (kJ / mol K )              * ( K    -↙
K   ) * (GJ / kJ)
36     sep.heat = total_molar_flowrate(sep.F) * avg_heat_capacity(sep.F) * (T_out - sep.↙
T) * GJ_PER_KJ;
37     sep.T = T_out;
38 end
39
40 function [combusted_fuel_flowrates, heatflux_left] = fuel_combustion(heat_flux,↙
flowrates)
41     global HYDROGEN METHANE ETHYLENE PROPANE BUTANE;
42     global ENTHALPY_METHANE ENTHALPY_PROPANE ENTHALPY_BUTANE HEAT_CAPACITY_ETHANE;
43     global MT_PER_KT G_PER_KT GJ_PER_KJ KJ_PER_GJ MOLMASS_METHANE KT_PER_G↙
MOLMASS_BUTANE ...
44             MOLMASS_PROPANE PSA_TOGGLE ENTHALPY_HYDROGEN MOLMASS_HYDROGEN
45
46     % Note! : Longest Chain Hydrocarbons are cheapest to combust
47
48     % initialize all values in the array to be zero
49     combusted_fuel_flowrates = flowrates * 0;
50
51     % LOGIC : Goes through each heat source in order, returns if the heat flux↙
supplied is sufficient.
```

```
52      heatflux_left = heat_flux;
53
54      % (GJ / yr)           = (kt / yr)          * (g / kt) * (kJ / g)        * (GJ /↙
kJ)
55      Q_combust_all_hydrogen = flowrates(HYDROGEN) * G_PER_KT * ENTHALPY_HYDROGEN *↙
GJ_PER_KJ;
56
57      if (~PSA_TOGGLE)
58          % Hydrogen
59          if (heatflux_left > Q_combust_all_hydrogen)
60              combusted_fuel_flowrates(HYDROGEN) = flowrates(HYDROGEN);
61              heatflux_left = heatflux_left - Q_combust_all_hydrogen;
62          else
63              % (kt / yr)                        = ((GJ)                 ) * (KJ / GJ) *
64              combusted_fuel_flowrates(HYDROGEN) = (heatflux_left) * KJ_PER_GJ * ...
65                  ... % (mol / KJ)        * (g / mol)       * (kt / g)
66                  ( 1 / ENTHALPY_HYDROGEN) * MOLMASS_HYDROGEN * KT_PER_G;
67              heatflux_left = 0;
68              return
69          end
70      end
71
72      % (GJ / yr)           = (kt / yr)          * (g / kt) * (kJ / g)        * (GJ /↙
kJ)
73      Q_combust_all_methane = flowrates(METHANE) * G_PER_KT * ENTHALPY_METHANE *↙
GJ_PER_KJ;
74
75      % Methane
76      if (heatflux_left > Q_combust_all_methane)
77          combusted_fuel_flowrates(METHANE) = flowrates(METHANE);
78          heatflux_left = heatflux_left - Q_combust_all_methane;
79      else
80          % (kt / yr)                        = ((GJ)                 ) * (KJ / GJ) *
81          combusted_fuel_flowrates(METHANE) = (heatflux_left) * KJ_PER_GJ * ...
82              ... % (mol / KJ)        * (g / mol)       * (kt / g)
83              ( 1 / ENTHALPY_METHANE) * MOLMASS_METHANE * KT_PER_G;
84          heatflux_left = 0;
85          return
86      end
87  end
88
```

```
 1
 2 function user = get_user_inputs()
 3
 4     % Level 3 | hard coded
 5     user.level3.molar_ratio_methanol_EO = 15;
 6     user.level3.molar_ratio_methanol_EC = 15;
 7     user.level3.molar_ratio_carbon_dioxide_EO = 13;
 8     user.level3.molar_ratio_carbon_dioxide_EC = 12;
 9         % one mol equiv is consumed in the virtual reactor
10     user.level3.tau_precision = 500;
11     % user.level3.tau_min = 50;
12     % user.level3.tau_max = 500;
13
14     % isothermal tau ranges
15     user.level3.isothermal.tau_min = 500;
16     user.level3.isothermal.tau_max = 10^4;
17     user.level3.isothermal.P_specify.tau_min = 100;
18     user.level3.isothermal.P_specify.tau_max = 10^4;
19     user.level3.isobaric.tau_min = 100;
20     user.level3.isobaric.tau_max = 10^5;
21
22
23     user.level3.temp_precision = 10;
24     user.level3.temp_min.C = 80;
25     user.level3.temp_max.C = 140;
26
27     user.level3.pressure_precision = 5;
28     user.level3.press_min.bar = 50;
29     user.level3.press_max.bar = 150;
30
31     user.level3.isothermal_temp.C = 140;
32     user.level3.isobaric_press.bar = 150;
33     user.level3.fsolveOpt = optimoptions('fsolve', 'Display', 'off');
34     % user.level3.fsolveOpt = optimoptions('fsolve', ...
35     %     'Algorithm', 'trust-region-dogleg', ...       % Choose the algorithm
36     %     'Display', 'off', ...                         % Control display level
37     %     'FunctionTolerance', 1e-3, ...                % Termination tolerance on↙
the function value
38     %     'StepTolerance', 1e-3, ...                    % Termination tolerance on↙
the step size
39     %     'MaxFunctionEvaluations', 10^4, ...           % Maximum number of function↙
evaluations allowed
40     %     'MaxIterations', 10^3, ...                     % Maximum number of↙
iterations allowed
41     %     'OptimalityTolerance', 1e-8, ...              % Termination tolerance on↙
the first-order optimality
42     %     'CheckGradients', true, ...                  % Check gradients numerically
43     %     'FiniteDifferenceStepSize', 1e-8, ...         % Step size for finite↙
differences
44     %     'FiniteDifferenceType', 'forward', ...        % Finite difference type↙
('forward' or 'central')
45     %     'FunctionTolerance', 1e-3, ...                % Termination tolerance on↙
the function value
46     %     'PlotFcn', [], ...                            % Plot functions
47     %     'OutputFcn', []);                             % Output functions
48     user.dmc_production_rate = 100; % [kta]
49
50     % Plotting
```

```
 51     user.plot.image_dpi = '-r600'; % [600 dpi]
 52     user.plot.isothermal.x_point = 0.85;
 53     user.plot.isothermal.y_point = -100;
 54     user.plot.supercritical_c02_pressure = 73.8;    %  Bar
 55
 56     % Level 3 | Soft coded
 57     % user.level3.tau_range = ...
 58                 % linspace(user.level3.isobaric.tau_min, user.level3.isobaric.↙
tau_max, user.level3.tau_precision);
 59
 60     user.level3.tau_range.isobaric = ...
 61                 linspace(user.level3.isobaric.tau_min, user.level3.isobaric.tau_max,↙
user.level3.tau_precision);
 62
 63     user.level3.tau_range.isothermal = ...
 64                 linspace(user.level3.isothermal.tau_min, user.level3.isothermal.↙
tau_max, user.level3.tau_precision);
 65
 66     user.level3.tau_range.P_specify.isothermal = ...
 67         linspace(user.level3.isothermal.P_specify.tau_min, user.level3.isothermal.↙
P_specify.tau_max, user.level3.tau_precision);
 68
 69
 70
 71         % user.level3.tau_range.isothermal.plot_npv_all_pressures = ...
 72         % linspace(500, user.level3.isothermal.tau_max, user.level3.tau_precision);
 73
 74
 75
 76
 77
 78     user.level3.temp_range = ...
 79         linspace(user.level3.temp_min.C, user.level3.temp_max.C, ...
 80                     user.level3.temp_precision);
 81     user.level3.press_range = ...
 82         linspace(user.level3.press_min.bar, user.level3.press_max.bar, ...
 83                     user.level3.pressure_precision);
 84
 85     % NPV
 86     user.npv.enterprise_rate = 0.15;                % per year
 87     user.npv.total_tax_rate = 0.27;                 % per year
 88     user.npv.admin_and_general_services = 0.05;     % 5% of annual revenues
 89     user.npv.construction_period = 3;               % years
 90     user.npv.period_plant_operation = 12;           % years
 91     user.npv.project_life = ...
 92         user.npv.construction_period + user.npv.period_plant_operation;
 93                                                     % years (constr + op yrs)
 94     user.depreciation_schedule = 10;                % years, linear deprec.
 95     user.marshall_and_swift_index = 1800;           % M&S
 96     user.salvage_value = 0.05;                      % 5% of fixed capital investment
 97
 98     % user.npv.discount_rate =
 99
100
101     % Aspen
102     user.aspen.reactor_volumes = [ 30, 120, 210, 300];
103     user.aspen.conversions = [ 0.55, 0.65, 0.85, 0.95];
104 end
```

105

```
 1
 2 function const = get_constants()
 3     const.units = get_unit_conversions();
 4     const.molar_mass = get_molar_masses();
 5     const.stoich = get_stoichiometric_coeff();
 6     const.thermo = get_thermodynamic_constants();
 7     const.econ = get_economic_constants();
 8     const.densities = get_species_densities();
 9     % const.heat_cap = get_heat_capacities();
10     const.pi = 3.14159;
11
12 end
13
14
15 function rho = get_species_densities();
16     rho.units = "kg / m^3 ";
17
18     rho.ethylene_carbonate = 1.3214; % g / ml
19         % https://pubchem.ncbi.nlm.nih.gov/compound/Ethylene-↙
carbonate#section=Melting-Point
20         % kg / m3 = (g/ml) * (1000ml / L) * (kg / 1000 g) * (1000L / m^3)
21     rho.ethylene_carbonate = rho.ethylene_carbonate * 1000;
22
23     rho.dimethyl_carbonate = 1.069; % g / ml
24         % https://www.sigmaaldrich.com/US/en/product/sial/517127
25     rho.dimethyl_carbonate = rho.dimethyl_carbonate * 1000;
26
27     rho.ethylene_glycol = 1.113; % g / ml
28         % https://www.sigmaaldrich.com/US/en/product/sial/324558
29     rho.ethylene_glycol = rho.ethylene_glycol * 1000;
30
31     rho.methoxy_ethanol = 0.965; % g / ml
32         % https://www.sigmaaldrich.com/US/en/product/sial/284467
33     rho.methoxy_ethanol = rho.methoxy_ethanol * 1000;
34
35     rho.ethylene_oxide = 0.882; % g / ml
36         % https://www.sigmaaldrich.com/US/en/product/aldrich/387614
37     rho.ethylene_oxide = rho.ethylene_oxide * 1000;
38
39 end
40
41
42 function econ = get_economic_constants()
43
44     econ.value.units = "$ / MT";
45     econ.value.dimethyl_carbonate = 1100;
46     econ.value.ethylene_glycol = 500;
47     econ.value.methanol = 600;
48     econ.value.ethylene_oxide = 1250;
49     econ.value.carbon_dioxide_feedstock = 45;
50
51     econ.value.fuel.units = "$ / GJ";
52     econ.value.fuel.natural_gas = 3;
53
54     % econ.value
55 end
56
57 function thermo = get_thermodynamic_constants()
```

```
58      thermo.R = 8.314;                   % [ J / mol K ]
59      % thermo.heat_cap = get_heat_capacities();
60      % thermo.rate_const = get_rate_constants();
61      % thermo.rate = get_reaction_rates();
62      % thermo.rate_const = get_isothermal_rate_constants();
63      thermo.enthalpy = get_reaction_enthalpies();
64  end
65
66  function enthalpy = get_reaction_enthalpies()
67      enthalpy.e1.kj_per_mol = -60.8;
68      enthalpy.e2.kj_per_mol = -53.5;
69      enthalpy.e3.kj_per_mol = -55.3;
70
71  end
72
73  function stoich = get_stoichiometric_coeff()
74      % WARNING : DOES NOT GIVE NEGATIVE VALUES
75
76      stoich.r1.ethylene_oxide = 1;
77      stoich.r1.carbon_dioxide = 1;
78      stoich.r1.ethylene_carbonate = 1;
79
80      stoich.r2.ethylene_carbonate = 1;
81      stoich.r2.methanol = 2;
82      stoich.r2.dimethyl_carbonate = 1;
83      stoich.r2.ethylene_glycol = 1;
84
85      stoich.r3.ethylene_carbonate = 1;
86      stoich.r3.methanol = 1;
87      stoich.r3.methoxy_ethanol = 1;
88      stoic.r3.carbon_dioxide = 1;
89  end
90
91  function units = get_unit_conversions()
92
93      % Mass
94      units.mass.mt_per_kt = 10^3;
95      units.mass.g_per_kt = 10^9;
96      units.mass.kt_per_g = 10^-9;
97      units.mass.kg_per_kt =  10^6;
98      units.mass.mt_per_g = 10^-6;
99      units.mass.kg_per_g = 10^-3;
100     units.mass.g_per_kg = 10^3;
101
102     % Energy
103     units.energy.gj_per_kj = 10^-6;
104     units.energy.kj_per_gj = 10^6;
105     units.energy.gj_per_j = 10^-9;
106
107     % Temperature
108     units.temperature.c_to_k = @(T_C) T_C + 273.15;
109     units.temperature.k_to_c = @(T_K) T_K - 273.15;
110
111     % Value
112     units.value.mmdolla_per_dolla = 10^-6;    % [ $ MM / $]
113     units.value.dolla_per_mmdolla = 10^6;    % [ $ / $ MM ]
114
115     % Pressure
```

```
116     units.pressure.bar_per_psia = 0.0689476;
117     units.pressure.psia_per_bar = 1 / units.pressure.bar_per_psia ;
118
119     % Time
120     units.time.yr_per_sec = 1 / (3.154 * 10^7);
121     units.time.sec_per_yr = 3.154 * 10^7;
122     units.time.yr_per_hr = (1/8760 );
123     units.time.hr_per_yr = 8760;
124
125     % Volumes
126     units.volume.m3_per_l = 0.001;
127     units.volume.l_per_m3 = 1000;
128
129     % Length
130     units.length.ft_per_m =  3.28084;
131     units.length.m_per_ft = 1 / units.length.ft_per_m;
132
133     % heat
134     units.heat.millionbtu_per_gj = 1.0551;       % [ ]
135 end
136
137 function molar_mass = get_molar_masses()
138     molar_mass.units = "g / mol";
139
140     molar_mass.ethylene_oxide = 44.0526;            % [ g / mol ]
141         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=C75218&Mask=80
142     molar_mass.carbon_dioxide =  44.0095;           % [ g / mol ]
143         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=124-38-9
144     molar_mass.ethylene_carbonate =  88.0621;       % [ g / mol ]
145         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=C96491&Mask=200
146     molar_mass.methanol = 32.0419;                  % [ g / mol ]
147         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=67-56-1
148     molar_mass.dimethyl_carbonate = 90.0779;        % [ g / mol ]
149         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=C616386&Mask=200
150     molar_mass.ethylene_glycol =62.0678;            % [ g / mol ]
151         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=107-21-1
152     molar_mass.methoxy_ethanol = 76.10;              % [ g / mol ]
153         % source: https://webbook.nist.gov/cgi/cbook.cgi?ID=109-86-4
154
155     molar_mass.aniline = 93.13;
156     molar_mass.water = 18;
157 end
158
159
160
```

```
 1
 2
 3
 4 function console = get_console()
 5
 6     console.divider = ...
 7     "_____________________________________________________________________";
 8
 9     console.section = @console_section;
10     console.subsection = @console_subsection;
11 end
12
13 function void = console_section(section_name)
14
15     divider =↙
"_____________________________________________________________________";
16
17     fprintf("%s%s\n", section_name, divider);
18 end
19
20
21 function void = console_subsection(section_name, indent)
22
23     divider =↙
"_____________________________________________________________________";
24
25     tabs = "";
26     for i = 1:indent
27         tabs = tabs + "\t";
28     end
29
30     fprintf(tabs + "%s%s\n",section_name, divider);
31 end
```

```
 1
 2 function fxns = flowrate_fxns()
 3     fxns.set_F_kta = @set_F_kta;
 4     fxns.set_F_mol = @set_F_mol;
 5     fxns.get_blank_flowstream = @get_blank_flowstream;
 6     fxns.get_basis_feed_flowrates = @get_basis_feed_flowrates;
 7     fxns.set_mol_fractions = @set_mol_fractions;
 8     fxns.get_total_flowrate = @get_total_flowrate;
 9 end
10
11 function F_total = get_total_flowrate(F, opt)
12
13     if strcmp('mol', opt)
14         fieldNames = fieldnames(F);
15         F_total = 0;
16         for i = 1:length(fieldNames)
17             species = fieldNames{i};
18             F_total = F_total + F.(species).mol;
19         end
20     elseif strcmp('kta', opt)
21         % Find total molar flow rate
22         fieldNames = fieldnames(F);
23         F_total = 0;
24         for i = 1:length(fieldNames)
25             species = fieldNames{i};
26             F_total = F_total + F.(species).kta;
27         end
28     else
29         disp("ERROR : get_total_flowrate : invalid option")
30         F_total = NaN;
31     end
32 end
33
34
35 function F = get_basis_feed_flowrates()
36     user = get_user_inputs();
37
38     F = get_blank_flowstream();
39
40     % Define the basis ratios for the REAL reactor input flows
41     F.ethylene_carbonate.mol = 1;
42     F.methanol.mol = F.ethylene_carbonate.mol * user.level3.molar_ratio_methanol_EC;
43     F.carbon_dioxide.mol = F.ethylene_carbonate.mol * user.level3.↙
molar_ratio_carbon_dioxide_EC;
44
45     % set the flowrates
46     F = set_F_mol(F);
47
48     % update the mol fractions
49     F = set_mol_fractions(F);
50 end
51
52 function F = get_blank_flowstream()
53     F.carbon_dioxide.mol = 0;
54     F.ethylene_oxide.mol = 0;
55     F.methanol.mol = 0;
56     F.ethylene_carbonate.mol = 0;
57     F.ethylene_glycol.mol = 0;
```

```
 58     F.methoxy_ethanol.mol = 0;
 59     F.dimethyl_carbonate.mol = 0;
 60
 61     F = set_F_mol(F);
 62
 63     % update the mol fractions
 64     F = set_mol_fractions(F);
 65 end
 66
 67 function F = set_F_kta(F)
 68     % Calculates the molar flow rates in mol / s for a given flow rates in kta
 69     const = get_constants();
 70
 71     % Calculate the mol / s flowrate for given kta flowrate inputs
 72     fieldNames = fieldnames(F);
 73     for i = 1:length(fieldNames)
 74         % mol / s            = (kt / yr )           * (g / kt)
 75         F.(fieldNames{i}).mol = F.(fieldNames{i}).kta * const.units.mass.g_per_kt *↙
...
 76             ... %  (yr / sec)      * (mol / g)
 77             const.units.time.yr_per_sec * (1 / const.molar_mass.(fieldNames{i}));
 78     end
 79
 80     % update the mol fractions with new calculated molar flowrates
 81     F = set_mol_fractions(F);
 82 end
 83
 84 function F = set_F_mol(F)
 85     % Calculates the the molar flow rates in kta for given flow rates in mol / s
 86     const = get_constants();
 87
 88     % Set the kta flowrates, for given flowrates in mol / s
 89     fieldNames = fieldnames(F);
 90     for i = 1:length(fieldNames)
 91         % kt / yr            = (mol / s)            * (g / mol)
 92         F.(fieldNames{i}).kta = F.(fieldNames{i}).mol * const.molar_mass.(fieldNames↙
{i}) * ...
 93             ... % (kt / g)           * (s / yr)
 94             const.units.mass.kt_per_g * const.units.time.sec_per_yr;
 95     end
 96
 97    % update the mol fractions
 98     F = set_mol_fractions(F);
 99 end
100
101 function F = set_mol_fractions(F)
102
103     % Find total molar flow rate
104     fieldNames = fieldnames(F);
105     F_total = 0;
106     for i = 1:length(fieldNames)
107         species = fieldNames{i};
108         F_total = F_total + F.(species).mol;
109     end
110
111     % Calculate the mol fractions
112     for i = 1:length(fieldNames)
113         if F_total
```

```
114             F.(fieldNames{i}).x = F.(fieldNames{i}).mol / F_total;
115         else % For blank flowstreams
116             F.(fieldNames{i}).x = 0;
117         end
118     end
119
120 end
```

```
%% MASTER FUNCTION

% aspen_datapoints();

function fxns = get_economic_functions()
    fxns.get_npv = @get_npv;
    fxns.get_work_min_npv = @get_work_min_npv;
    fxns.get_osbl = @get_osbl;
    fxns.get_isbl = @get_isbl;
    fxns.get_aspen_datapoints = @get_aspen_datapoints;

end

function energy = get_energy_plant()

    energy = 0;
end

function Fp = get_closest_Fp(P)
    % Define the table values
    pressures = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000];
    Fp_values = [1.00, 1.05, 1.15, 1.20, 1.35, 1.45, 1.60, 1.80, 1.90, 2.30, 2.50];

    % Find the closest pressure in the table
    [~, idx] = min(abs(pressures - P));

    % Get the corresponding Fp value
    Fp = Fp_values(idx);
end

function F = get_aspen_Vrxtr_data()
    user = get_user_inputs();

    % Values to be stored
    values = [72.1761, 3.2673, 0.0000, 112.1209, 644.8418, 509.2977, 2.0989, 0.5246,↙
0.0000];

    % Names of the species
    species_names = {'ethylene_glycol', 'ethylene_carbonate', 'ethylene_oxide', ...
                     'dimethyl_carbonate', 'carbon_dioxide', 'methanol', ...
                     'methoxy_ethanol', 'aniline', 'water'};

    % Initialize the structure
    species_data = flowrate_fxns().get_blank_flowstream;

    % Store the values in the structure
    for i = 1:length(species_names)
        F.(species_names{i}).kta = values(i);
    end
    F = flowrate_fxns().set_F_kta(F);


end

function chi = get_aspen_dataset_conversion(V)
    conversion = get_user_inputs().aspen.conversions;
    switch V
        case 30
```

```
 58             chi = conversion(1);
 59         case 120
 60             chi = conversion(2);
 61         case 210
 62             chi = conversion(3);
 63         case 300
 64             chi = conversion(4);
 65         otherwise
 66             disp("ERROR | aspen_dataset_conversion ")
 67     end
 68 end
 69
 70
 71 function aspen_data = get_aspen_datapoints()
 72     user = get_user_inputs();
 73
 74
 75     % function lifetime_npv = get_npv(npv)
 76         % USER_INPUTS | All inputs are in units of $MM
 77             % npv.mainProductRevenue = value_ethylene(P_ethylene);
 78             % npv.byProductRevenue = value_h2_chem(P_hydrogen - combusted_hydrogen);
 79             % npv.rawMaterialsCost = value_ethane(F_fresh_ethane);
 80             % npv.utilitiesCost = cost_steam(F_steam, COST_RATES_STEAM(STEAM_CHOICE,↙
STEAM_COST_COL));
 81             % npv.CO2sustainabilityCharge = tax_C02(combusted_fuel_flow_rates,↙
F_natural_gas);
 82             % npv.conversion = conversion(i);
 83             % npv.isbl = cost_rxt_vec + cost_separation_system(P_flowrates, F_steam,↙
R_ethane);
 84             % npv.isbl = cost_rxt_vec + cost_separation_system(P_flowrates, F_steam,↙
R_ethane);
 85     % lifetime_npv = get_npv_aspen(npv, osbl)
 86     % lifetime_npv = get_npv_aspen(npv, osbl)
 87     P_f = 1;
 88     for i = 1:length(user.aspen.reactor_volumes)
 89         V = user.aspen.reactor_volumes(i);
 90         F = get_aspen_Vrxtr_data();
 91         npv_params.mainProductRevenue = get_main_product_revenue(F);
 92         npv_params.byProductRevenue = get_byproduct_revenue(F);
 93         npv_params.rawMaterialsCost = get_raw_material_cost(F);
 94         npv_params.utilitiesCost = get_utilities_cost(F, T, P, P_f);
 95         npv_params.CO2sustainabilityCharge = get_CO2_sustainability_charge();
 96         npv_params.conversion = get_aspen_dataset_conversion(V);
 97         isbl = get_aspen_isbl_osbl(V).isbl;
 98 %       npv_params.  = get_aspen_isbl_osbl(V).osbl;
 99         y_npv(i) = get_npv_aspen(npv_params, isbl, get_aspen_isbl_osbl(V).osbl);
100     end
101
102
103     aspen_data.x = get_constants().npv.aspen.converions;
104     aspen_data.y = y_npv;
105
106
107 end
108
109
110
111
```

```
112
113
114 function installed_cost = get_cost_reactor(V, P)
115     % V = volume of reactor in m^3
116     % P = bar
117     user = get_user_inputs();
118     const = get_constants();
119     coeff = (user.marshall_and_swift_index / 280) * 101.9;
120     F_m = 1.0;
121         % Assumption : Carbon Steel is the material ??
122     P = P * const.units.pressure.psia_per_bar;
123     F_p = get_closest_Fp(P );
124     F_c = F_m * F_p;
125     k = 2.942 / 4.413; % D / H ratio is assumed constant
126     D.m = (4 * k * V / const.pi)^(1/3);
127     H.m = D.m / k;
128     D.ft = D.m * const.units.length.ft_per_m;
129     H.ft = H.m * const.units.length.ft_per_m;
130     installed_cost = coeff * (D.ft)^1.066 * H.ft^0.82 * (2.18 + F_c);
131     installed_cost = installed_cost * 1.5;
132     V;
133 end
134
135 function data = get_aspen_isbl_osbl(V_rxtr)
136
137     if V_rxtr == 30
138         operating_costs = 37.401 * 10^6;
139         capital_costs  = 19.575 * 10^6;
140     elseif V_rxtr == 120
141         operating_costs = 37.554 * 10^6;
142         capital_costs = 19.342 * 10^6;
143     elseif V_rxtr == 210
144         operating_costs = 37.237 * 10^6;
145         capital_costs = 19.897 * 10^6;
146     elseif V_rxtr == 300
147         operating_costs = 37.197 * 10^6;
148         capital_costs = 20.325 * 10^6;
149     else
150         disp("ERROR | get_aspen_isbl_osbl | invalid V_rxtr choice")
151         data = NaN;
152         return
153     end
154
155     data.isbl = capital_costs;
156     data.osbl = operating_costs;
157 end
158
159 function isbl_capital_cost = get_isbl(V_rxtr, P, opt)
160     if strcmp(opt, 'aspen')
161         isbl_capital_cost = get_cost_reactor(V_rxtr, P);
162         data = get_aspen_isbl_osbl(V_rxtr);
163
164         isbl_capital_cost = isbl_capital_cost + data.isbl;
165     elseif strcmp(opt, 'matlab')
166         isbl_capital_cost = get_cost_reactor(V_rxtr, P);
167     else
168         isbl = NaN;
169         disp("ERROR | get_ISBL_data | opt not valid");
```

```
170     end
171 end
172
173 function osbl_capital_cost = get_osbl(V_rxtr, P, opt)
174     if strcmp(opt, 'aspen')
175         osbl_capital_cost = get_cost_reactor(V_rxtr, P);
176         data = get_aspen_isbl_osbl(V_rxtr);
177
178         osbl_capital_cost = osbl_capital_cost + data.osbl;
179     elseif strcmp(opt, 'matlab')
180         osbl_capital_cost = get_cost_reactor(V_rxtr, P);
181     else
182         osbl = NaN;
183         disp("ERROR | get_OSBL_data | opt not valid");
184     end
185 end
186
187
188
189 function charge = get_CO2_sustainability_charge()
190     charge = 0;
191 end
192
193 function cost = get_separation_system_cost()
194     cost = 0;
195 end
196
197 function lifetime_npv = get_work_min_npv(F, T, P, V_rxtr, conversion, opt)
198     P_f = 1;
199     sep_fxns = separation_fxns();
200     w_min = sep_fxns.get_work_min(F, T);
201
202     npv_params.mainProductRevenue = get_main_product_revenue(F);
203     npv_params.byProductRevenue = get_byproduct_revenue(F);
204     npv_params.rawMaterialsCost = get_raw_material_cost(F);
205     energy = get_energy_plant();
206     npv_params.utilitiesCost = get_utilities_cost(F, T, P, P_f);
207     npv_params.conversion = conversion;
208     % npv_params.ISBLcapitalCost = get_cost_reactor(V_rxtr, P);
209     if strcmp(opt,'aspen')
210         npv_params.ISBLcapitalCost = get_isbl(V_rxtr, P, opt);
211     else
212         npv_params.ISBLcapitalCost = get_cost_reactor(V_rxtr, P);
213     end
214
215     npv_params.CO2sustainabilityCharge = get_CO2_sustainability_charge();
216
217     if strcmp(opt, 'aspen')
218         lifetime_npv = get_npv_aspen(npv_params, npv_params.ISBLcapitalCost, ...
219                                                  get_osbl(V_rxtr, P, opt));
220     else
221         lifetime_npv = get_npv(npv_params) / 10^6 ;
222     end
223 end
224
225 function value = chemical_value(F, species)
226
227     switch true
```

```
228         case strcmp(species, 'dimethyl_carbonate')
229             value = 1100 * 10^3 * F.dimethyl_carbonate.kta;
230         case strcmp(species, 'ethylene_glycol')
231             value = 500 * 10^3 * F.ethylene_glycol.kta;
232         case strcmp(species, 'methanol')
233             value = 600 * 10^3 * F.methanol.kta;
234         case strcmp(species, 'ethylene_oxide')
235             value = 1250 * 10^3 * F.ethylene_glycol.kta;
236         case strcmp(species, 'carbon_dioxide')
237             value = 45 * 10^3 * F.carbon_dioxide.kta;
238         otherwise
239             value = NaN;
240             fprintf("ERROR | chemical_value | invalid case | %s\n", species);
241     end
242 end
243
244 function value = get_main_product_revenue(F)
245     % Products are the value of DMC
246     value = chemical_value(F, 'dimethyl_carbonate');
247 end
248
249 function value = get_byproduct_revenue(F)
250     % byproducts are EG
251     value = chemical_value(F, 'ethylene_glycol');
252
253 end
254
255 function cost = get_raw_material_cost(F)
256     % raw material costs are EO, MeOH, C02, water
257     % ask TJ how to cost the water for the sep unit
258
259     cost = 0;
260     cost = cost + chemical_value(F, 'ethylene_oxide');
261     cost = cost + chemical_value(F, 'methanol');
262     cost = cost + chemical_value(F, 'carbon_dioxide');
263     % cost = cost +
264         % water ??
265 end
266
267 function cost = get_utilities_cost(F,T, P, P_f)
268     % utilities are electricity, fuel??
269     rate.dollar_per_GJ = 3;
270
271     R = get_constants().thermo.R;
272
273
274     F_tot = flowrate_fxns().get_total_flowrate(F, 'mol');
275
276 %  W = compressor_work_TJ(sep, P_f)
277     % R = 8.314;        % [ J / mol K]
278     % n = total_molar_flowrate(sep.F);
279     % T = sep.T;
280     % P_0 = sep.P;
281
282     W = - F_tot * R * T * log10(P_f / P);   % ?? I think that it's base 10
283     W = get_constants().units.energy.gj_per_j;
284     cost = rate.dollar_per_GJ * W;
285 end
```

```
286
287 function value = get_CO2_sustainability_value()
288     % 45 / MT, confirm with TJ
289
290     value = 0;
291
292 end
293
294 function lifetime_npv = get_npv_aspen(npv, isbl, osbl)
295     % USER_INPUTS | All inputs are in units of $MM
296         % npv.mainProductRevenue = value_ethylene(P_ethylene);
297         % npv.byProductRevenue = value_h2_chem(P_hydrogen - combusted_hydrogen);
298         % npv.rawMaterialsCost = value_ethane(F_fresh_ethane);
299         % npv.utilitiesCost = cost_steam(F_steam, COST_RATES_STEAM(STEAM_CHOICE,↙
STEAM_COST_COL));
300         % npv.CO2sustainabilityCharge = tax_C02(combusted_fuel_flow_rates,↙
F_natural_gas);
301         % npv.conversion = conversion(i);
302         % npv.isbl = cost_rxt_vec + cost_separation_system(P_flowrates, F_steam,↙
R_ethane);
303
304         % look at photos screenshot, cost of electricity, some shit
305     user = get_user_inputs();
306
307     YEARS_IN_OPERATION = user.npv.period_plant_operation;
308
309     % % Economic Assumptions
310     % npv.discountRate = 0.15;      % [ % in decimal ]
311     % npv.taxRate = 0.27;               % [ % in decimal ]
312     % npv.salvageValue = 0.05;      % [ % in decimal ]
313
314     npv.discountRate = user.npv.enterprise_rate;    % [ % in decimal ]
315     npv.taxRate = user.npv.total_tax_rate;      % [ % in decimal ]
316     npv.salvageValue = user.salvage_value;      % [ % in decimal ]
317
318     WORKING_CAP_PERCENT_OF_FCI = 0.15;      % [ % in decimal ]
319     STARTUP_COST_PERCENT_OF_FCI = 0.10;     % [ % in decimal ]
320     LENGTH_CONSTRUCTION_TABLE = 6;
321     LAST_ROW_CONSTRUCTION = LENGTH_CONSTRUCTION_TABLE;
322     YEARS_OF_CONSTUCTION = user.npv.construction_period;
323
324     % Revenues & Production Costs
325     npv.consummablesCost = 0;
326     npv.VCOP = npv.rawMaterialsCost + npv.utilitiesCost + ...
327                 npv.consummablesCost + npv.CO2sustainabilityCharge - ...
328                                                         npv.byProductRevenue;
329     npv.salaryAndOverhead = 0;
330     npv.maintenenace = 0;
331     % npv.interest = 15;
332     npv.interest = user.npv.enterprise_rate;
333
334     % npv.AGS = (npv.mainProductRevenue + npv.byProductRevenue)*0.05;       % ~5%↙
revenue
335     npv.AGS = (npv.mainProductRevenue + npv.byProductRevenue) * ...
336         user.npv.admin_and_general_services;
337
338     npv.FCOP = npv.salaryAndOverhead + npv.maintenenace +...
339                             npv.AGS + npv.interest;
```

```
340
341     % Capital Costs
342     npv.ISBLcapitalCost = isbl;
343         % modification
344     npv.OSBLcapitalCost = osbl;
345         % ??  npv.ISBLcapitalCost, npv.OSBLcapitalCost)
346     npv.contingency = (npv.ISBLcapitalCost + npv.OSBLcapitalCost) * 0.25;
347     npv.indirectCost = (npv.ISBLcapitalCost + npv.OSBLcapitalCost + ...
348                                                 npv.contingency) * 0.30;
349     npv.totalFixedCapitalCost = npv.ISBLcapitalCost + ...
350                                 npv.OSBLcapitalCost + ...
351                                 npv.indirectCost + ...
352                                 npv.contingency;
353
354     npv.workingCapital = npv.totalFixedCapitalCost * WORKING_CAP_PERCENT_OF_FCI;
355     npv.startupCost = npv.totalFixedCapitalCost * STARTUP_COST_PERCENT_OF_FCI;
356     npv.land = 10;
357     npv.totalCapitalInvestment = npv.totalFixedCapitalCost + ...
358                                     npv.workingCapital + ...
359                                     npv.startupCost + ...
360                                     npv.land;
361
362
363     % CONSTRUCTION SCHEDULE INDICIES
364     YEAR = 1;
365     FC = 2;
366     WC = 3;
367     SU = 4;
368     FCOP = 5;
369     VCOP = 6;
370     construction_matrix = zeros(LENGTH_CONSTRUCTION_TABLE + 1, VCOP);
371
372     % Generate the construction schedule matrix
373     for yr = 0:LENGTH_CONSTRUCTION_TABLE
374         row = yr + 1;
375         if yr > 0 && yr < 4
376             construction_matrix(row, FC) = 0.33;
377         end
378         if yr == 3
379             construction_matrix(row, WC) = 1.00;
380             construction_matrix(row, SU) = 1.00;
381         end
382         if yr > 3 && yr <= 6
383             construction_matrix(row, FCOP) = 1.00;
384             construction_matrix(row, VCOP) = 1.00;
385         end
386     end
387
388     % NPV COLUMN INDICIES
389     YEAR = 1;
390     CAPITAL_EXPENSE = 2;
391     REVENUE = 3;
392     COM = 4;
393     GROSS_PROFIT = 5;
394     DEPRECIATION = 6;
395     TAXABLE_INC = 7;
396     TAXES_PAID = 8;
397     CASH_FLOW = 9;
```

```
398     CUM_CASH_FLOW = 10;
399     PV_OF_CF = 11;
400     CUM_PV_OF_CF = 12;
401     NPV = 13;
402     cash_flow_matrix = zeros(YEARS_IN_OPERATION + 1, NPV);
403     LAST_ROW_CASHFLOW = YEARS_IN_OPERATION + 1;
404
405
406     for yr = 0:YEARS_IN_OPERATION
407         row = yr + 1;
408         cash_flow_matrix(row, YEAR) = yr;
409
410         % Capital Expenses Column
411         if yr == 0
412             cash_flow_matrix(row, CAPITAL_EXPENSE) = npv.land;
413         elseif yr >= 1 && yr <= 5
414             cash_flow_matrix(row, CAPITAL_EXPENSE) ...
415                 = npv.totalFixedCapitalCost * construction_matrix(row,FC) + ...
416                   npv.workingCapital * construction_matrix(row, WC) + ...
417                   npv.startupCost * construction_matrix(row, SU) ;
418         elseif yr == YEARS_IN_OPERATION
419             cash_flow_matrix(row, CAPITAL_EXPENSE) = - npv.salvageValue * npv.↙
totalFixedCapitalCost;
420         else
421             cash_flow_matrix(row, CAPITAL_EXPENSE) ...
422                 = npv.totalFixedCapitalCost * construction_matrix↙
(LAST_ROW_CONSTRUCTION,FC) + ...
423                   npv.workingCapital * construction_matrix(LAST_ROW_CONSTRUCTION, WC)↙
+ ...
424                   npv.startupCost * construction_matrix(LAST_ROW_CONSTRUCTION, SU) ;
425         end
426
427         % Revenue Column
428         if yr <= LENGTH_CONSTRUCTION_TABLE % ??
429             cash_flow_matrix(row, REVENUE) = npv.mainProductRevenue *↙
construction_matrix(row, VCOP);
430         else
431             cash_flow_matrix(row, REVENUE) = npv.mainProductRevenue *↙
construction_matrix(LAST_ROW_CONSTRUCTION, VCOP);
432         end
433
434         % COM Column
435         if yr <= LENGTH_CONSTRUCTION_TABLE
436             cash_flow_matrix(row, COM) = npv.VCOP * construction_matrix(row, VCOP) +↙
...
437                                          npv.FCOP * construction_matrix(row,↙
FCOP);
438         else
439             cash_flow_matrix(row, COM) = npv.VCOP * construction_matrix↙
(LAST_ROW_CONSTRUCTION, VCOP) + ...
440                                      npv.FCOP * construction_matrix↙
(LAST_ROW_CONSTRUCTION, FCOP);
441         end
442
443         % Gross Profit
444         cash_flow_matrix(row, GROSS_PROFIT) = cash_flow_matrix(row,REVENUE) -↙
cash_flow_matrix(row, COM);
445
```

```
446         % Depreciation
447         if yr >= YEARS_OF_CONSTUCTION
448             cash_flow_matrix(row, DEPRECIATION) = 0.1*(npv.totalFixedCapitalCost +↙
npv.startupCost - 0.05*npv.totalFixedCapitalCost);
449         end
450
451         % Taxable Inc
452         if yr >= YEARS_OF_CONSTUCTION
453             cash_flow_matrix(row, TAXABLE_INC) = cash_flow_matrix(row, GROSS_PROFIT)↙
- cash_flow_matrix(row,DEPRECIATION);
454         end
455
456         % Taxes Paid
457         if yr >= YEARS_OF_CONSTUCTION
458             cash_flow_matrix(row, TAXES_PAID) = cash_flow_matrix(row, TAXABLE_INC) *↙
npv.taxRate;
459         end
460
461         % Cash Flow
462         cash_flow_matrix(row, CASH_FLOW) = -cash_flow_matrix(row, CAPITAL_EXPENSE) +↙
...
463                 ( cash_flow_matrix(row,REVENUE) ...
464                     - cash_flow_matrix(row, COM) ...
465                     - cash_flow_matrix(row, DEPRECIATION) ...
466                 ) * ( 1 - npv.taxRate) + cash_flow_matrix(row, DEPRECIATION);
467
468         % Cummulative Cash Flow
469         cash_flow_matrix(row, CUM_CASH_FLOW) = sum( cash_flow_matrix( 1 : row,↙
CASH_FLOW) );
470
471         % PV of CF
472         cash_flow_matrix(row, PV_OF_CF) = cash_flow_matrix(row, CASH_FLOW) / ( 1 +↙
npv.discountRate)^yr;
473
474         % Cummulative PV of CF
475         cash_flow_matrix(row , CUM_PV_OF_CF) = sum( cash_flow_matrix(1:row, PV_OF_CF)↙
);
476
477         % NPV
478         if row > 1
479             cash_flow_matrix(row , NPV) = cash_flow_matrix(row - 1, NPV) +↙
cash_flow_matrix(row, PV_OF_CF);
480         else
481             cash_flow_matrix(row, NPV) = cash_flow_matrix(row, PV_OF_CF);
482         end
483     end
484
485     % RETURN
486     cf.matrix = cash_flow_matrix;
487     cf.lifetime_npv = cash_flow_matrix(LAST_ROW_CASHFLOW, NPV);
488
489     lifetime_npv = cf.lifetime_npv;
490 end
491
492 function lifetime_npv = get_npv(npv)
493     % USER_INPUTS | All inputs are in units of $MM
494         % npv.mainProductRevenue = value_ethylene(P_ethylene);
495         % npv.byProductRevenue = value_h2_chem(P_hydrogen - combusted_hydrogen);
```

```
496         % npv.rawMaterialsCost = value_ethane(F_fresh_ethane);
497         % npv.utilitiesCost = cost_steam(F_steam, COST_RATES_STEAM(STEAM_CHOICE,↙
STEAM_COST_COL));
498         % npv.CO2sustainabilityCharge = tax_C02(combusted_fuel_flow_rates,↙
F_natural_gas);
499         % npv.conversion = conversion(i);
500         % npv.isbl = cost_rxt_vec + cost_separation_system(P_flowrates, F_steam,↙
R_ethane);
501
502         % look at photos screenshot, cost of electricity, some shit
503     user = get_user_inputs();
504
505     YEARS_IN_OPERATION = user.npv.period_plant_operation;
506
507     % % Economic Assumptions
508     % npv.discountRate = 0.15;      % [ % in decimal ]
509     % npv.taxRate = 0.27;              % [ % in decimal ]
510     % npv.salvageValue = 0.05;      % [ % in decimal ]
511
512     npv.discountRate = user.npv.enterprise_rate;    % [ % in decimal ]
513     npv.taxRate = user.npv.total_tax_rate;      % [ % in decimal ]
514     npv.salvageValue = user.salvage_value;      % [ % in decimal ]
515
516     WORKING_CAP_PERCENT_OF_FCI = 0.15;      % [ % in decimal ]
517     STARTUP_COST_PERCENT_OF_FCI = 0.10;     % [ % in decimal ]
518     LENGTH_CONSTRUCTION_TABLE = 6;
519     LAST_ROW_CONSTRUCTION = LENGTH_CONSTRUCTION_TABLE;
520     YEARS_OF_CONSTUCTION = user.npv.construction_period;
521
522     % Revenues & Production Costs
523     npv.consummablesCost = 0;
524     npv.VCOP = npv.rawMaterialsCost + npv.utilitiesCost + ...
525                 npv.consummablesCost + npv.CO2sustainabilityCharge - ...
526                                                         npv.byProductRevenue;
527     npv.salaryAndOverhead = 0;
528     npv.maintenenace = 0;
529     % npv.interest = 15;
530     npv.interest = user.npv.enterprise_rate;
531
532     % npv.AGS = (npv.mainProductRevenue + npv.byProductRevenue)*0.05;       % ~5%↙
revenue
533     npv.AGS = (npv.mainProductRevenue + npv.byProductRevenue) * ...
534         user.npv.admin_and_general_services;
535
536     npv.FCOP = npv.salaryAndOverhead + npv.maintenenace +...
537                           npv.AGS + npv.interest;
538
539     % Capital Costs
540     npv.OSBLcapitalCost = npv.ISBLcapitalCost * 0.40;
541         % ??  npv.ISBLcapitalCost, npv.OSBLcapitalCost)
542     npv.contingency = (npv.ISBLcapitalCost + npv.OSBLcapitalCost) * 0.25;
543     npv.indirectCost = (npv.ISBLcapitalCost + npv.OSBLcapitalCost + ...
544                                                     npv.contingency) * 0.30;
545     npv.totalFixedCapitalCost = npv.ISBLcapitalCost + ...
546                                 npv.OSBLcapitalCost + ...
547                                 npv.indirectCost + ...
548                                 npv.contingency;
549
```

```
550     npv.workingCapital = npv.totalFixedCapitalCost * WORKING_CAP_PERCENT_OF_FCI;
551     npv.startupCost = npv.totalFixedCapitalCost * STARTUP_COST_PERCENT_OF_FCI;
552     npv.land = 10;
553     npv.totalCapitalInvestment = npv.totalFixedCapitalCost + ...
554                                  npv.workingCapital + ...
555                                  npv.startupCost + ...
556                                  npv.land;
557
558
559     % CONSTRUCTION SCHEDULE INDICIES
560     YEAR = 1;
561     FC = 2;
562     WC = 3;
563     SU = 4;
564     FCOP = 5;
565     VCOP = 6;
566     construction_matrix = zeros(LENGTH_CONSTRUCTION_TABLE + 1, VCOP);
567
568     % Generate the construction schedule matrix
569     for yr = 0:LENGTH_CONSTRUCTION_TABLE
570         row = yr + 1;
571         if yr > 0 && yr < 4
572             construction_matrix(row, FC) = 0.33;
573         end
574         if yr == 3
575             construction_matrix(row, WC) = 1.00;
576             construction_matrix(row, SU) = 1.00;
577         end
578         if yr > 3 && yr <= 6
579             construction_matrix(row, FCOP) = 1.00;
580             construction_matrix(row, VCOP) = 1.00;
581         end
582     end
583
584     % NPV COLUMN INDICIES
585     YEAR = 1;
586     CAPITAL_EXPENSE = 2;
587     REVENUE = 3;
588     COM = 4;
589     GROSS_PROFIT = 5;
590     DEPRECIATION = 6;
591     TAXABLE_INC = 7;
592     TAXES_PAID = 8;
593     CASH_FLOW = 9;
594     CUM_CASH_FLOW = 10;
595     PV_OF_CF = 11;
596     CUM_PV_OF_CF = 12;
597     NPV = 13;
598     cash_flow_matrix = zeros(YEARS_IN_OPERATION + 1, NPV);
599     LAST_ROW_CASHFLOW = YEARS_IN_OPERATION + 1;
600
601
602     for yr = 0:YEARS_IN_OPERATION
603         row = yr + 1;
604         cash_flow_matrix(row, YEAR) = yr;
605
606         % Capital Expenses Column
607         if yr == 0
```

```
608             cash_flow_matrix(row, CAPITAL_EXPENSE) = npv.land;
609         elseif yr >= 1 && yr <= 5
610             cash_flow_matrix(row, CAPITAL_EXPENSE) ...
611                 = npv.totalFixedCapitalCost * construction_matrix(row,FC) + ...
612                   npv.workingCapital * construction_matrix(row, WC) + ...
613                   npv.startupCost * construction_matrix(row, SU) ;
614         elseif yr == YEARS_IN_OPERATION
615             cash_flow_matrix(row, CAPITAL_EXPENSE) = - npv.salvageValue * npv.↙
totalFixedCapitalCost;
616         else
617             cash_flow_matrix(row, CAPITAL_EXPENSE) ...
618                 = npv.totalFixedCapitalCost * construction_matrix↙
(LAST_ROW_CONSTRUCTION,FC) + ...
619                   npv.workingCapital * construction_matrix(LAST_ROW_CONSTRUCTION, WC)↙
+ ...
620                   npv.startupCost * construction_matrix(LAST_ROW_CONSTRUCTION, SU) ;
621         end
622
623         % Revenue Column
624         if yr <= LENGTH_CONSTRUCTION_TABLE % ??
625             cash_flow_matrix(row, REVENUE) = npv.mainProductRevenue *↙
construction_matrix(row, VCOP);
626         else
627             cash_flow_matrix(row, REVENUE) = npv.mainProductRevenue *↙
construction_matrix(LAST_ROW_CONSTRUCTION, VCOP);
628         end
629
630         % COM Column
631         if yr <= LENGTH_CONSTRUCTION_TABLE
632             cash_flow_matrix(row, COM) = npv.VCOP * construction_matrix(row, VCOP) +↙
...
633                                          npv.FCOP * construction_matrix(row,↙
FCOP);
634         else
635             cash_flow_matrix(row, COM) = npv.VCOP * construction_matrix↙
(LAST_ROW_CONSTRUCTION, VCOP) + ...
636                                      npv.FCOP * construction_matrix↙
(LAST_ROW_CONSTRUCTION, FCOP);
637         end
638
639         % Gross Profit
640         cash_flow_matrix(row, GROSS_PROFIT) = cash_flow_matrix(row,REVENUE) -↙
cash_flow_matrix(row, COM);
641
642         % Depreciation
643         if yr >= YEARS_OF_CONSTUCTION
644             cash_flow_matrix(row, DEPRECIATION) = 0.1*(npv.totalFixedCapitalCost +↙
npv.startupCost - 0.05*npv.totalFixedCapitalCost);
645         end
646
647         % Taxable Inc
648         if yr >= YEARS_OF_CONSTUCTION
649             cash_flow_matrix(row, TAXABLE_INC) = cash_flow_matrix(row, GROSS_PROFIT)↙
- cash_flow_matrix(row,DEPRECIATION);
650         end
651
652         % Taxes Paid
653         if yr >= YEARS_OF_CONSTUCTION
```

```
654             cash_flow_matrix(row, TAXES_PAID) = cash_flow_matrix(row, TAXABLE_INC) *↙
npv.taxRate;
655         end
656
657         % Cash Flow
658         cash_flow_matrix(row, CASH_FLOW) = -cash_flow_matrix(row, CAPITAL_EXPENSE) +↙
...
659                 ( cash_flow_matrix(row,REVENUE) ...
660                     - cash_flow_matrix(row, COM) ...
661                     - cash_flow_matrix(row, DEPRECIATION) ...
662                 ) * ( 1 - npv.taxRate) + cash_flow_matrix(row, DEPRECIATION);
663
664         % Cummulative Cash Flow
665         cash_flow_matrix(row, CUM_CASH_FLOW) = sum( cash_flow_matrix( 1 : row,↙
CASH_FLOW) );
666
667         % PV of CF
668         cash_flow_matrix(row, PV_OF_CF) = cash_flow_matrix(row, CASH_FLOW) / ( 1 +↙
npv.discountRate)^yr;
669
670         % Cummulative PV of CF
671         cash_flow_matrix(row , CUM_PV_OF_CF) = sum( cash_flow_matrix(1:row, PV_OF_CF)↙
);
672
673         % NPV
674         if row > 1
675             cash_flow_matrix(row , NPV) = cash_flow_matrix(row - 1, NPV) +↙
cash_flow_matrix(row, PV_OF_CF);
676         else
677             cash_flow_matrix(row, NPV) = cash_flow_matrix(row, PV_OF_CF);
678         end
679     end
680
681     % RETURN
682     cf.matrix = cash_flow_matrix;
683     cf.lifetime_npv = cash_flow_matrix(LAST_ROW_CASHFLOW, NPV);
684
685     lifetime_npv = cf.lifetime_npv;
686 end
687
```

I Team Member Work Statement

Group 10

Team Member Work Statement

Name of team member here **Wesley Johanson**

State here what you contributed to the design project and report

I did all of the MATLAB coding and simulations. Also helping with higher level conceptual design and interpretation of results from the simulations to help direct the direction of the design.

Name of team member here **Isaiah Huma**

State here what you contributed to the design project and report

I did all of the Level 2 and 3 mole balance derivations, derivations of the system of equations for the reactor design, Aspen Hysys flowsheets at 2 isobaric and 2 isothermal conditions, 4 Aspen Hysys flowsheets at various volumes, equipment sizing and costing, NPV calculations, sensitivity analysis calculations, tornado plot, stream tables, equipment tables, appendices, and HAZOP table.

Name of team member here **Thejas (TJ) Ravish**

State here what you contributed to the design project and report

I did all research for group for plant design, aided in Aspen Hysys flowsheets and costing/NPV, and did most of the writing of report as well as PFD design

Print Name and Sign: Wesley Johanson Date: 6/5/2024

Print Name and Sign Isaiah Huma Date: 6/5/24

Isaiah Huma

Print Name and Sign Thejas Ravish Date: 6/5/24

Thejas Ravish

Rating of Team Members for Design Project

Please rate each group member's contributions in the categories below:
1-2 - unsatisfactory, 3 - acceptable/adequate, 4 – very good, 5 - excellent
Each member fills out one form and signs the bottom.

Name :	1) Wesley Johanson	2) Isaiah Huma	3) Thejas Ravish
Quality of work presented	5	5	5
Quantity of work performed	5	5	5
Effort	5	5	5
Punctuality (meetings and deadlines)	5	5	5
Knowledge of design methods	5	5	5
Class attendance	5	5	5
Communication	5	5	5

Do you feel that each member of the group deserves the same grade? If not, who does not and why?

Yes

It's important to note that differences in performance will not necessarily affect individual grades; however, large discrepancies may result in differences in grades.

Additional comments:

Chemical Engineering is insane…

Print Name and Sign:  Date: 6/5/2024

Wesley Johanson

Rating of Team Members for Design Project

Please rate each group member's contributions in the categories below:
1-2 - unsatisfactory, 3 - acceptable/adequate, 4 – very good, 5 - excellent
Each member fills out one form and signs the bottom.

Name :	1) Wesley Johanson	2) Isaiah Huma	3) Thejas Ravish
Quality of work presented	5	5	5
Quantity of work performed	5	5	5
Effort	5	5	5
Punctuality (meetings and deadlines)	5	5	5
Knowledge of design methods	5	5	5
Class attendance	5	5	5
Communication	5	5	5

Do you feel that each member of the group deserves the same grade? If not, who does not and why?

Yes

It's important to note that differences in performance will not necessarily affect individual grades; however, large discrepancies may result in differences in grades.

Additional comments:

Print Name and Sign: Thejas Ravish Date: 6/5/24

Thejas Ravish

Rating of Team Members for Design Project

Please rate each group member's contributions in the categories below:
1-2 - unsatisfactory, 3 - acceptable/adequate, 4 – very good, 5 - excellent
Each member fills out one form and signs the bottom.

Name: 1) ___Isaiah Huma_____ 2) __Wesley Johanson__ 3) _Thejas Ravish_____

Quality of work presented	__5___	__4___	__5___
Quantity of work performed	__5___	__5___	__4___
Effort	__5__	__4__	__5___
Punctuality (meetings and deadlines)	__5___	__5___	__5___
Knowledge of design methods	__5__	__5___	__5___
Class attendance	__5___	__5___	__5__
Communication	__4__	__5___	__5___

Do you feel that each member of the group deserves the same grade? If not, who does not and why?

Yes

It's important to note that differences in performance will not necessarily affect individual grades; however, large discrepancies may result in differences in grades.

Additional comments:

I had a lot of fun working in this group and could not have asked for better teammates and friends

Print Name and Sign: ____Isaiah Huma__Isaiah Huma_____ Date: ___6/5/24___