ENGSCI 700A/B

# Research Compendium

*Connor McDowall*
*cmcd398*
*530913386*

October 31, 2020

# Contents

# List of Figures

# List of Tables

# 1 Programming

These section contains all programming scripts for the project.

## 1.1 GOCPI NZ Energy Systems Example

The GOCPI NZ Energy Systems Example is the processing script for designing NZ and AUS Energy Systems

## 1.2 GOCPI Module

### 1.2.1 Navigation

### 1.2.2 Energysystems

### 1.2.3 CreateCases

### 1.2.4 Forecasting

### 1.2.5 Optimisation

```python
# GOCPI_NZ_Example.gyp is an exemplar script in how to build a
# data case for the Model

#
   ###############################################################################

# This is a major input script for creating data files.
#
   ###############################################################################

# Import all necessary python packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sc
import sklearn as skl
import csv as csv
import openpyxl
import pathlib
import os
from pathlib import Path
from openpyxl import load_workbook
import GOCPI as GF
import cplex as cp
import docplex as dp

# Sets sets (All must be one word)
# Creates a New Zealand Energy System Scenario using the CreateCases
   Module
nz_energy_system = GF.CreateCases()

# Set Definitions
#
   ###############################################################################
```

```python
# ##############################################################################

# Defines the forecast period
nz_energy_system.set_year(2020, 2030, 1)

# Defines the regions
REGION = ['NEWZEALAND', 'AUSTRALIA']
nz_energy_system.set_region(REGION)

# Defines the Emissions
EMISSION = ['CO2', 'NOX', 'CO', 'METHANE']
nz_energy_system.set_emission(EMISSION)

# Technology
# ##############################################################################
# ##############################################################################
# Defines the technology set (MBIE Energy Statistics Energy Supply and Demand -
# Gross PJ (Higher Heating Value))
Production = [
    'Indigenous_Production', 'Imports', 'Exports', 'Stock_Change',
    'International_Transport'
]
Conversion = [
    'Electricity_Generation', 'Cogeneration', 'Fuel_Production',
    'Other_Transformation', 'Losses_and_Own_Use'
]
Non_Energy = ['Non_Energy_Use']
Consumption = [
    'Agriculture', 'Forestry_and_Logging', 'Fishing', 'Mining',
    'Food_Processing', 'Textiles', 'Wood_Pulp_Paper_and_Printing', 'Chemicals',
    'Non_Metallic_Minerals', 'Basic_Metals',
    'Mechanical_Electrical_Equippment', 'Building_and_Construction',
    'Unallocated', 'Commercial', 'Transport', 'Residential'
]
Statistical_Differences = ['Statistical_Differences']
TECHNOLOGY_ALL = [
    Production, Conversion, Non_Energy, Consumption, Statistical_Differences
]
TECHNOLOGY = []
for tech in TECHNOLOGY_ALL:
    for i in range(0, len(tech), 1):
        TECHNOLOGY.append(tech[i])

# Sets the technology set
nz_energy_system.set_technology(TECHNOLOGY)

# Sets capacity technologies for energy production
CAPACITY_TECHNOLOGY = Conversion
CONSUMPTION_TECHNOLOGY = Consumption
```

```python
nz_energy_system.set_capacity_technology(TECHNOLOGY)
nz_energy_system.set_availability_technology(TECHNOLOGY)
# Sets the Conversion Sets

# ##############################################################################
# Calculates Energy Balances Base Year
# ##############################################################################

# Sets names for the energy balance sheets
NZ_energy_balances = GF.Forecasting()
root_energy_balance = pathlib.Path(
    '/Users/connor/Google Drive/Documents/University/Courses/2020/ENGSCI 700A&B/GOCPI/data/Energy Balances'
)
IEA_World_Energy_Balances_A2K = 'IEAWorldEnergyBalances2017A-K.csv'
IEA_World_Energy_Balances_L2Z = 'IEAWorldEnergyBalances2017L-Z.csv'
create_excel_spreadsheet = True
output_file = "Geo EB.xlsx"

# Creates the geography dataframe
outputs = NZ_energy_balances.energy_balance_base(
    root_energy_balance, IEA_World_Energy_Balances_A2K,
    IEA_World_Energy_Balances_L2Z, create_excel_spreadsheet, output_file)

# ##############################################################################
# Calculates Fuels
# ##############################################################################
# Defines the fuel set (MBIE Energy Statistics Energy Supply and Demand - Gross PJ (Higher Heating Value))
Coal = ['Bituminous', 'Sub_Bitumious', 'Lignite']
Oil = [
    'Crude_Feedstocks_NGL', 'LPG', 'Petrol', 'Diesel', 'Fuel_Oil',
    'Aviation_Fuel_and_Kerosine', 'Oil_Other'
]
Natural_Gas = ['Natural_Gas']
Renewables = [
    'Hydro', 'Geothermal', 'Solar', 'Wind', 'Liquid_Biofuels', 'Biogas', 'Wood'
]
Electricity = ['Electricity']
Waste_Heat = ['Waste_Heat']

FUEL_ALL = [Coal, Oil, Natural_Gas, Renewables, Electricity, Waste_Heat]
FUEL = []
for fuel_type in FUEL_ALL:
    for i in range(0, len(fuel_type), 1):
        FUEL.append(fuel_type[i])

```

```
# Sets Specified Fuels
SPECIFIED_FUEL_ALL = [
    Coal, Oil, Natural_Gas, Renewables, Electricity, Waste_Heat
]
SPECIFIED_FUEL = []
for fuel_type in SPECIFIED_FUEL_ALL:
    for i in range(0, len(fuel_type), 1):
        SPECIFIED_FUEL.append(fuel_type[i])

# Sets Accumulated Fuels
ACCUMULATED_FUEL_ALL = [
    Coal, Oil, Natural_Gas, Renewables, Electricity, Waste_Heat
]
ACCUMULATED_FUEL = []
for fuel_type in ACCUMULATED_FUEL_ALL:
    for i in range(0, len(fuel_type), 1):
        ACCUMULATED_FUEL.append(fuel_type[i])

# Sets the total fuels
nz_energy_system.set_fuel(FUEL)
nz_energy_system.set_specified_fuel(FUEL)
nz_energy_system.set_accumulated_fuel(FUEL)
#
    ##############################################################################

# Continues defining sets
#
    ##############################################################################

# Defines timeslices
TIMESLICE = [
    'DAY_SUMMER', 'NIGHT_SUMMER', 'DAY_WINTER', 'NIGHT_WINTER',
    'DAY_INTERMEDIATE', 'NIGHT_INTERMEDIATE'
]
nz_energy_system.set_timeslice(TIMESLICE)

# Defines Modes of Operation
nz_energy_system.set_mode_of_operation(1)

# Defines the storage set
STORAGE = ['DAM']
nz_energy_system.set_storage(STORAGE)

# Defines the daytype (numbers represent different daytypes)
# 1 = Weekday (Mon - Fri), 2 = Weekend (Sat & Sun)
nz_energy_system.set_daytype(2)

# Defines the seasons
# (Three seasons (Summer (1), Winter (2) and Intermediate (3)))
nz_energy_system.set_season(3)

# Defines the dailytimebracket (Number of distinct periods in a day)
# 4 = Morning (6hrs), Afternoon (6hrs), Evening (6hrs), Night (6hrs)
nz_energy_system.set_daily_time_bracket(4)

#
    ##############################################################################
```

```
# Defines Global Parameters
# ##############################################################################
# Defines the YearSplit parameter
# Creates Dictionary for number of days
days = {
    'January': 31,
    'February': 28,
    'March': 31,
    'April': 30,
    'May': 31,
    'June': 30,
    'July': 31,
    'August': 31,
    'September': 30,
    'October': 31,
    'November': 30,
    'December': 31
}

# Combines summer, winter and intermediate nights
days_summer = days['January'] + days['February'] + days['December']
days_winter = days['June'] + days['July'] + days['August']
days_intermediate = days['April'] + days['May'] + days['March'] + days['September'] + days['October'] + days['November']
days_total = days_summer + days_winter + days_intermediate

# Creates fractions and stores values in a dictionary
day_summer = (0.5 * days_summer / days_total)
night_summer = (0.5 * days_summer / days_total)
day_winter = (0.5 * days_winter / days_total)
night_winter = (0.5 * days_winter / days_total)
day_intermediate = (0.5 * days_intermediate / days_total)
night_intermediate = (0.5 * days_intermediate / days_total)

# Dictionaries
splits = {
    'DAY_SUMMER': day_summer,
    'NIGHT_SUMMER': night_summer,
    'DAY_WINTER': day_winter,
    'NIGHT_WINTER': night_winter,
    'DAY_INTERMEDIATE': day_intermediate,
    'NIGHT_INTERMEDIATE': night_intermediate
}
# Creates the YearSplit parameter 2D Matrix
nz_energy_system.set_year_split(TIMESLICE, nz_energy_system.year, splits)

# Imports S&P NZX:50 and S&P ASX:200 Indices Arrays to calculate market returns
root = '/Users/connor/Google Drive/Documents/University/Courses/2020/ENGSCI 700A&B/GOCPI/data/Inputs/GOCPI OseMOSYS'
file_root = Path(root)
file_spreadsheet = 'Returns.xls'
location = GF.Navigation(file_root, file_spreadsheet)
market_returns = location.Find_File()
nz_df = pd.read_excel(market_returns, sheet_name='NZ')
```

```
aus_df = pd.read_excel(market_returns, sheet_name='AUS')
nz_index = nz_df[["Monthly_Returns"]].to_numpy()
aus_index = aus_df[["Monthly_Returns"]].to_numpy()

# Defines the Dictionaries required for Region. All regions should have the same names
# Creates a dictionary of market indices
market_index = {'NEWZEALAND': nz_index, 'AUSTRALIA': aus_index}
# Tresury Equity Balances as at 2019
# (Australia has negative equity, New Zealand has $139746000000)
# However, Governments do not have market equity so should be zer for both
equity = {'NEWZEALAND': 0, 'AUSTRALIA': 0}
# Tresury Debt Balance as at 2019
debt = {'NEWZEALAND': 110477000000, 'AUSTRALIA': 619219000000}
# Tresury Finance Cost(Interest Expenses on Debt as at 2019
cost_of_debt_pre_tax = {'NEWZEALAND': 4059000000, 'AUSTRALIA': 17088000000}
# Preference Equity (None for governments)
preference_equity = {'NEWZEALAND': 0, 'AUSTRALIA': 0}
market_value_preference_shares = {'NEWZEALAND': 1, 'AUSTRALIA': 1}
# (Set to zero if none otherwise you get an error)
preference_dividends = {'NEWZEALAND': 0, 'AUSTRALIA': 0}
# Calculated from 10 Year Treasury Bonds (10 Year Average)
risk_free_rate = {'NEWZEALAND': 0.0360, 'AUSTRALIA': 0.0335}
# Company Tax Rates
effective_tax_rate = {'NEWZEALAND': 0.28, 'AUSTRALIA': 0.30}
# Beta for region modelled
market_risk_coefficient = {'NEWZEALAND': 0, 'AUSTRALIA': 0}

# Sets the discount rates
nz_energy_system.set_discount_rate(equity, debt, market_index,
                                   cost_of_debt_pre_tax, risk_free_rate,
                                   effective_tax_rate, preference_equity,
                                   market_value_preference_shares,
                                   preference_dividends,
                                   market_risk_coefficient)

# Creates Dictionary of day splits (assumes constant accross years)
# Preserve the order of the split.
hour_split = {"1": 6, "2": 6, "3": 6, "4": 6}
num_days = 365
num_hours = 24
nz_energy_system.set_day_split(nz_energy_system.dailytimebracket,
                               nz_energy_system.year, hour_split, num_days,
                               num_hours)

# Sets a dictionary to match the timeslice with season
link_ls = {
    "DAY_SUMMER": "1",
    "NIGHT_SUMMER": "1",
    "DAY_WINTER": "2",
    "NIGHT_WINTER": "2",
    "DAY_INTERMEDIATE": "3",
    "NIGHT_INTERMEDIATE": "3"
```

```python
}
nz_energy_system.set_conversion_ls(nz_energy_system.timeslice,
                                   nz_energy_system.season, link_ls)
# Sets a dictionary to match the timeslice with daytype
# Daytypes: 1 = Weekday (Mon - Fri), 2 = Weekend (Sat & Sun)
# Order must be preserved
link_ld = {
    "DAY_SUMMER": np.ones((1, 2)),
    "NIGHT_SUMMER": np.ones((1, 2)),
    "DAY_WINTER": np.ones((1, 2)),
    "NIGHT_WINTER": np.ones((1, 2)),
    "DAY_INTERMEDIATE": np.ones((1, 2)),
    "NIGHT_INTERMEDIATE": np.ones((1, 2))
}
nz_energy_system.set_conversion_ld(nz_energy_system.timeslice,
                                   nz_energy_system.daytype, link_ld)
# Sets a dictionary to match the timeslice with daytype
# 1). Morning (6hrs), 2).Afternoon (6hrs), 3).Evening (6hrs), 4).Night (6hrs)
# Order must be preserved in the arrays
link_lh = {
    "DAY_SUMMER": np.array([1, 1, 0, 0]),
    "NIGHT_SUMMER": np.array([0, 0, 1, 1]),
    "DAY_WINTER": np.array([1, 1, 0, 0]),
    "NIGHT_WINTER": np.array([0, 0, 1, 1]),
    "DAY_INTERMEDIATE": np.array([1, 1, 0, 0]),
    "NIGHT_INTERMEDIATE": np.array([0, 0, 1, 1])
}
override_conversionlh = None
# Sets the Conversionlh parameter

nz_energy_system.set_conversion_lh(nz_energy_system.timeslice,
                                   nz_energy_system.dailytimebracket, link_lh,
                                   override_conversionlh)
# Creates season dictionary for daytypes (Assumed to be the same each year)
link_dtdt = {
    "1": np.array([5, 2]),
    "2": np.array([5, 2]),
    "3": np.array([5, 2])
}
override_dtdt = None
# Sets the DaysInDayType parameter
nz_energy_system.set_days_in_day_type(nz_energy_system.season,
                                      nz_energy_system.daytype,
                                      nz_energy_system.year, link_dtdt,
                                      override_dtdt)

# Creates trade relationships using an 2D numpy array
# Must [NEWZEALAND, AUSTRALIA],[NEWZEALAND, AUSTRALIA]
# Hypothetically, you can model any trade relationship for any fuel in any year
# FUELS = As above
# YEAR = 2020 - 2030 (11)
trade = np.zeros((len(nz_energy_system.region), len(nz_energy_system.region),
                  len(nz_energy_system.fuel), len(nz_energy_system.year)))
trade_all_fuels = np.array([[0, 1], [1, 0]])
for i in range(0, len(nz_energy_system.fuel), 1):
    for j in range(0, len(nz_energy_system.year), 1):
        trade[:, :, i, j] = trade_all_fuels
nz_energy_system.set_trade_route(trade)

# Creates depreciation methods dictionary
depreciation_methods = {"NEWZEALAND": 2, "AUSTRALIA": 2}
override_depreciation = None
nz_energy_system.set_depreciation_method(nz_energy_system.region,
                                         depreciation_methods,
                                         override_depreciation)

# ##############################################################################
# Initialisation and Definition of demand parameters (Including forecasting)
# ##############################################################################
# Sets dictionaries to calculate CAGR for Fuels Forecasts
nz_cagr_fuels = {}
aus_cagr_fuels = {}
cagr_dictionaries_regions = [nz_cagr_fuels, aus_cagr_fuels]
# Initialises cagr parameters
nz_start_year_fuels = {}
nz_end_year_fuels = {}
nz_start_value_fuels = {}
nz_end_value_fuels = {}
aus_start_year_fuels = {}
aus_end_year_fuels = {}
aus_start_value_fuels = {}
aus_end_value_fuels = {}
nz_cagr_dictionaries_parameters = [
    nz_start_year_fuels, nz_end_year_fuels, nz_start_value_fuels,
    nz_end_value_fuels
]
aus_cagr_dictionaries_parameters = [
    aus_start_year_fuels, aus_end_year_fuels, aus_start_value_fuels,
    aus_end_value_fuels
]

# Populates regional dictionaries with new entry, all fuel types with default cagr values
for region_fuels in cagr_dictionaries_regions:
    for i in range(0, len(nz_energy_system.fuel), 1):
        region_fuels[nz_energy_system.fuel[i]] = 0.05

# Populates regional dictionaries with new entry, all fuel types with default values
for parameters in nz_cagr_dictionaries_parameters:
    for i in range(0, len(nz_energy_system.fuel), 1):
        region_fuels[nz_energy_system.fuel[i]] = 1

for parameters in nz_cagr_dictionaries_parameters:
```

```python
    for i in range(0, len(nz_energy_system.fuel), 1):
        region_fuels[nz_energy_system.fuel[i]] = 1
# Loads demand data to the parameter dictionaries (Energy units are in PJs)
# New Zealand

nz_start_years = np.zeros(len(nz_energy_system.fuel))
nz_start_years[:] = 2010
nz_end_years = np.zeros(len(nz_energy_system.fuel))
nz_end_years[:] = 2018
nz_start_values = np.array([
    7.23, 13.24, 4.19, 0, 7.11, 110.43, 106.09, 7.11, 14.62, 0, 60.29, 0, 9.21,
    0.35, 0, 0, 0.33, 55.89, 146.49, 0
])
nz_end_values = np.zeros(len(nz_energy_system.fuel))
nz_end_values = np.array([
    3.07, 16.26, 5.14, 0, 8.71, 113.22, 138.79, 5.82, 16.23, 0, 73.97, 0, 8.03,
    0.36, 0, 0, 0.33, 56.61, 142.87, 0
])
# Australia
aus_start_years = np.zeros(len(nz_energy_system.fuel))
aus_start_years[:] = 2017
aus_end_years = np.zeros(len(nz_energy_system.fuel))
aus_end_years[:] = 2018
aus_start_values = np.array([
    104.9, 9.0, 0.5, 2.3, 72.4, 847.9724, 1038.76619, 42.39862, 190.79379, 0.0,
    0.0, 0.0, 0, 15.7, 0.0, 8.4, 94.7, 79.2, 821.8, 0
])
aus_end_values = np.zeros(len(nz_energy_system.fuel))
aus_end_values = np.array([
    104.445, 8.737, 0.38, 2.019, 67.499, 904.7584, 1108.32904, 45.23792,
    135.71376, 0.35788, 942.965, 0, 0, 16.56, 0, 8.642, 83.592, 76.81, 835.439,
    0
])
# Assign values to the dictionary
for i in range(0, len(nz_energy_system.fuel), 1):
    aus_start_year_fuels[nz_energy_system.fuel[i]] = aus_start_years[i]
    aus_end_year_fuels[nz_energy_system.fuel[i]] = aus_end_years[i]
    aus_start_value_fuels[nz_energy_system.fuel[i]] = aus_start_values[i]
    aus_end_value_fuels[nz_energy_system.fuel[i]] = aus_end_values[i]
    nz_start_year_fuels[nz_energy_system.fuel[i]] = nz_start_years[i]
    nz_end_year_fuels[nz_energy_system.fuel[i]] = nz_end_years[i]
    nz_start_value_fuels[nz_energy_system.fuel[i]] = nz_start_values[i]
    nz_end_value_fuels[nz_energy_system.fuel[i]] = nz_end_values[i]

print("nz_start_year_fuels", nz_start_year_fuels)
print("nz_end_year_fuels", nz_end_year_fuels)
print("nz_start_value_fuels", nz_start_value_fuels)
print("nz_end_value_fuels", nz_end_value_fuels)
print("aus_start_year_fuels", aus_start_year_fuels)
print("aus_end_year_fuels", aus_end_year_fuels)
print("aus_start_value_fuels", aus_start_value_fuels)
print("aus_end_value_fuels", aus_end_value_fuels)

# Calculates the cagr dictionary
forecasting_functions = GF.Forecasting()
for fuel in nz_cagr_fuels:
    nz_cagr_fuels[
        fuel] = forecasting_functions.
   calculate_constant_average_growth_rate(
            nz_start_year_fuels[fuel], nz_end_year_fuels[fuel],
            nz_start_value_fuels[fuel], nz_end_value_fuels[fuel])
for fuel in aus_cagr_fuels:
    aus_cagr_fuels[
        fuel] = forecasting_functions.
   calculate_constant_average_growth_rate(
            aus_start_year_fuels[fuel], aus_end_year_fuels[fuel],
            aus_start_value_fuels[fuel], aus_end_value_fuels[fuel])

# Calculates NZ CAGR forecasts
nz_fuel_forecast = forecasting_functions.calculate_cagr_forecasts(
    nz_cagr_fuels, nz_end_value_fuels, nz_energy_system.fuel,
    nz_energy_system.year)

# Calculates AUS CAGR forecasts
aus_fuel_forecast = forecasting_functions.calculate_cagr_forecasts(
    aus_cagr_fuels, aus_end_value_fuels, nz_energy_system.fuel,
    nz_energy_system.year)

fuel_forecasts = [nz_fuel_forecast, aus_fuel_forecast]

# Creates the forecast 3D array
forecast = np.zeros((len(nz_energy_system.region), len(nz_energy_system
   .fuel),
                     len(nz_energy_system.year)))

# Sets the forecast 3D array with CAGR forecast values
for i in range(0, len(fuel_forecasts), 1):
    forecast[i, :, :] = fuel_forecasts[i]

# Sets the Specified Demand Profiles
# nz_energy_system.set_specified_annual_demand(forecast[:, 0:-1, :])
nz_energy_system.set_specified_annual_demand(forecast[:, :, :])
# Sets the Accumulated Demand Profiles (Hack to make sure 3D Array)
acc_forecast = np.zeros(
    (len(nz_energy_system.region), len(nz_energy_system.
   accumulated_fuel),
     len(nz_energy_system.year)))
acc_forecast[:, 0, :] = forecast[:, -1, :]

# Make adjustments to the accumumulated fuel forecasts
nz_energy_system.set_accumulated_annual_demand(forecast[:, :, :])
# Sets linear profile for timeslices (In this example, is is assumed
   the fuel is consumed uniformally in time splits)
linear_profile = splits
override = None
# Sets the Specifief Demand Profiles
nz_energy_system.set_specified_demand_profile(
    nz_energy_system.SpecifiedAnnualDemand, nz_energy_system.region,
    nz_energy_system.specified_fuel, nz_energy_system.year,
    nz_energy_system.timeslice, linear_profile, override)

# Sets the Capacity to Activity Factors (Assume conversion of GW to PJ)
nz_capacity_to_activity = {}
aus_capacity_to_activity = {}
for tech in nz_energy_system.capacity_technology:
    nz_capacity_to_activity[tech] = 31.536
    aus_capacity_to_activity[tech] = 31.536

capacity_dictionaries = [nz_capacity_to_activity,
   aus_capacity_to_activity]
# Sets the CapacityToActivty Function
override = None
nz_energy_system.set_capacity_to_activity_unit(
    nz_energy_system.region, nz_energy_system.capacity_technology,
    capacity_dictionaries, override)
print(nz_energy_system.capacity_technology)
print(nz_energy_system.CapacityToActivityUnit)

# Sets capacity factor matrix to operate in every timeslice (Assumes
   operate 0.8 of the time).
capacity_factors = np.zeros(
    (len(nz_energy_system.region), len(nz_energy_system.
   capacity_technology),
     len(nz_energy_system.timeslice), len(nz_energy_system.year)))
capacity_factors[:, :, :, :] = 0.8

nz_energy_system.set_capacity_factor(capacity_factors)

# Set availability factors
availability_factors = np.zeros((len(nz_energy_system.region),
                                 len(nz_energy_system.
   availability_technology),
                                 len(nz_energy_system.year)))

availability_factors[:, :, :] = 1
nz_energy_system.set_availability_factor(availability_factors)

# Sets up operational life

#
#
# print(nz_energy_system.YearSplit)
# print(nz_energy_system.DiscountRate)
# print(nz_energy_system.DaySplit)
# print(nz_energy_system.Conversionld)
# print(nz_energy_system.Conversionls)
# print(nz_energy_system.Conversionlh)
# print(nz_energy_system.TradeRoute)
# print(nz_energy_system.DaysInDayType)
# print(nz_energy_system.DepreciationMethod)

# Initialises yet to be written parameters to check progress / load
   Parameters (Delete later)
ly = len(nz_energy_system.year)
lr = len(nz_energy_system.region)
le = len(nz_energy_system.emission)
lt = len(nz_energy_system.technology)
```

```
lf = len(nz_energy_system.fuel)
ll = len(nz_energy_system.timeslice)
lm = len(nz_energy_system.mode_of_operation)
ls = len(nz_energy_system.storage)
lld = len(nz_energy_system.daytype)
lls = len(nz_energy_system.season)
llh = len(nz_energy_system.dailytimebracket)

#nz_energy_system.YearSplit = np.ones((ll, ly))
#nz_energy_system.DiscountRate = np.ones((lr))
#nz_energy_system.DaySplit = np.ones((llh, ly))
#nz_energy_system.Conversionls = np.ones((ll, lls))
#nz_energy_system.Conversionld = np.ones((ll, lld))
#nz_energy_system.Conversionlh = np.ones((ll, llh))
#nz_energy_system.DaysInDayType = np.ones((lls, lld, ly))
#nz_energy_system.TradeRoute = np.ones((lr, lr, lf, ly))
#nz_energy_system.DepreciationMethod = np.ones((lr))
#nz_energy_system.SpecifiedAnnualDemand = np.ones((lr, lf, ly))
#nz_energy_system.SpecifiedDemandProfile = np.ones((lr, lf, ll, ly))
#nz_energy_system.AccumulatedAnnualDemand = np.ones((lr, lf, ly))
#nz_energy_system.CapacityToActivityUnit = np.ones((lr, lt))
#nz_energy_system.CapacityFactor = np.ones((lr, lt, ll, ly))
#nz_energy_system.AvailabilityFactor = np.ones((lr, lt, ly))
nz_energy_system.OperationalLife = np.ones((lr, lt))
nz_energy_system.ResidualCapacity = np.ones((lr, lt, ly))
nz_energy_system.InputActivityRatio = np.ones((lr, lt, lf, lm, ly))
nz_energy_system.OutputActivityRatio = np.ones((lr, lt, lf, lm, ly))
nz_energy_system.CapitalCost = np.ones((lr, lt, ly))
nz_energy_system.VariableCost = np.ones((lr, lt, lm, ly))
nz_energy_system.FixedCost = np.ones((lr, lt, ly))
nz_energy_system.TechnologyToStorage = np.ones((lr, lt, ls, lm))
nz_energy_system.TechnologyFromStorage = np.ones((lr, lt, ls, lm))
nz_energy_system.StorageLevelStart = np.ones((lr, ls))
nz_energy_system.StorageMaxChargeRate = np.ones((lr, ls))
nz_energy_system.StorageMaxDischargeRate = np.ones((lr, ls))
nz_energy_system.MinStorageCharge = np.ones((lr, ls, ly))
nz_energy_system.OperationalLifeStorage = np.ones((lr, ls))
nz_energy_system.CapitalCostStorage = np.ones((lr, ls, ly))
nz_energy_system.ResidualStorageCapacity = np.ones((lr, ls, ly))
nz_energy_system.CapacityOfOneTechnologyUnit = np.ones((lr, lt, ly))
nz_energy_system.TotalAnnualMaxCapacity = np.ones((lr, lt, ly))
nz_energy_system.TotalAnnualMinCapacity = np.ones((lr, lt, ly))
nz_energy_system.TotalAnnualMaxCapacityInvestment = np.ones((lr, lt, ly))
nz_energy_system.TotalAnnualMinCapacityInvestment = np.ones((lr, lt, ly))
nz_energy_system.TotalTechnologyAnnualActivityLowerLimit = np.ones(
    (lr, lt, ly))
nz_energy_system.TotalTechnologyAnnualActivityUpperLimit = np.ones(
    (lr, lt, ly))
nz_energy_system.TotalTechnologyModelPeriodActivityUpperLimit = np.ones(
    (lr, lt))
nz_energy_system.TotalTechnologyModelPeriodActivityLowerLimit = np.ones(
    (lr, lt))
nz_energy_system.ReserveMarginTagTechnology = np.ones((lr, lt, ly))
nz_energy_system.ReserveMarginTagFuel = np.ones((lr, lf, ly))
nz_energy_system.ReserveMargin = np.ones((lr, ly))
nz_energy_system.RETagTechnology = np.ones((lr, lt, ly))
nz_energy_system.RETagFuel = np.ones((lr, lf, ly))
nz_energy_system.REMinProductionTarget = np.ones((lr, ly))
nz_energy_system.EmissionActivityRatio = np.ones((lr, lt, le, lm, ly))
nz_energy_system.EmissionsPenalty = np.ones((lr, le, ly))
nz_energy_system.AnnualExogenousEmission = np.ones((lr, le, ly))
nz_energy_system.AnnualEmissionLimit = np.ones((lr, le, ly))
nz_energy_system.ModelPeriodExogenousEmission = np.ones((lr, le))
nz_energy_system.ModelPeriodEmissionLimit = np.ones((lr, le))

# Sets the case (Toggle depending on the data set you choose to use)
case = nz_energy_system

# Initialises the energy system
system = GF.Energy_Systems(
    nz_energy_system.year, nz_energy_system.region, nz_energy_system.emission,
    nz_energy_system.technology, nz_energy_system.capacity_technology,
    nz_energy_system.availability_technology, nz_energy_system.fuel,
    nz_energy_system.specified_fuel, nz_energy_system.accumulated_fuel,
    nz_energy_system.timeslice, nz_energy_system.mode_of_operation,
    nz_energy_system.storage, nz_energy_system.daytype,
    nz_energy_system.season, nz_energy_system.dailytimebracket)

# Loads the datacase to the system
system.load_datacase(case, system)

# Sets up location information
data_txt = 'GOCPI_NZ_Example_Data.txt'
model_source_file = 'GOCPI_OseMOSYS_Structure.xlsx'
root = '/Users/connor/Google Drive/Documents/University/Courses/2020/ENGSCI 700A&B/GOCPI/data/Inputs/GOCPI OseMOSYS'
data_roots = Path(root)
data_location_1 = os.path.join(data_roots, data_txt)

# Sets the default parameters
default_parameters = {
    'YearSplit': 1,
    'DiscountRate': 0.05,
    'DaySplit': 1,
    'Conversionls': 1,
    'Conversionld': 1,
    'Conversionlh': 1,
    'DaysInDayType': 1,
    'TradeRoute': 1,
    'DepreciationMethod': 2,
    'SpecifiedAnnualDemand': 1,
    'SpecifiedDemandProfile': 1,
    'AccumulatedAnnualDemand': 1,
    'CapacityToActivityUnit': 1,
    'CapacityFactor': 1,
    'AvailabilityFactor': 1,
    'OperationalLife': 1,
    'ResidualCapacity': 1,
    'InputActivityRatio': 1,
    'OutputActivityRatio': 1,
    'CapitalCost': 1,
    'VariableCost': 1,
    'FixedCost': 1,
    'TechnologyToStorage': 1,
    'TechnologyFromStorage': 1,
    'StorageLevelStart': 1,
    'StorageMaxChargeRate': 1,
    'StorageMaxDischargeRate': 1,
    'MinStorageCharge': 1,
    'OperationalLifeStorage': 1,
    'CapitalCostStorage': 1,
    'ResidualStorageCapacity': 1,
    'CapacityOfOneTechnologyUnit': 1,
    'TotalAnnualMaxCapacity': 99999,
    'TotalAnnualMinCapacity': 1,
    'TotalAnnualMaxCapacityInvestment': 999999,
    'TotalAnnualMinCapacityInvestment': 0,
    'TotalTechnologyAnnualActivityLowerLimit': 0,
    'TotalTechnologyAnnualActivityUpperLimit': 999999,
    'TotalTechnologyModelPeriodActivityUpperLimit': 999999,
    'TotalTechnologyModelPeriodActivityLowerLimit': 0,
    'ReserveMarginTagTechnology': 1,
    'ReserveMarginTagFuel': 1,
    'ReserveMargin': 1,
    'RETagTechnology': 1,
    'RETagFuel': 1,
    'REMinProductionTarget': 1,
    'EmissionActivityRatio': 1,
    'EmissionsPenalty': 1,
    'AnnualExogenousEmission': 1,
    'AnnualEmissionLimit': 1,
    'ModelPeriodExogenousEmission': 1,
    'ModelPeriodEmissionLimit': 1
}

# Sets the default toggles (To only use defaults)
toggle_defaults = {
    'YearSplit': False,
    'DiscountRate': False,
    'DaySplit': False,
    'Conversionls': False,
    'Conversionld': False,
    'Conversionlh': False,
    'DaysInDayType': False,
    'TradeRoute': False,
    'DepreciationMethod': False,
    'SpecifiedAnnualDemand': False,
    'SpecifiedDemandProfile': False,
    'AccumulatedAnnualDemand': False,
    'CapacityToActivityUnit': False,
    'CapacityFactor': False,
    'AvailabilityFactor': False,
    'OperationalLife': False,
    'ResidualCapacity': False,
    'InputActivityRatio': False,
    'OutputActivityRatio': False,
    'CapitalCost': False,
    'VariableCost': False,
    'FixedCost': False,
    'TechnologyToStorage': False,
    'TechnologyFromStorage': False,
    'StorageLevelStart': False,
    'StorageMaxChargeRate': False,
    'StorageMaxDischargeRate': False,
    'MinStorageCharge': False,
    'OperationalLifeStorage': False,
    'CapitalCostStorage': False,
    'ResidualStorageCapacity': False,
    'CapacityOfOneTechnologyUnit': False,
    'TotalAnnualMaxCapacity': False,
    'TotalAnnualMinCapacity': False,
    'TotalAnnualMaxCapacityInvestment': False,
    'TotalAnnualMinCapacityInvestment': False,
    'TotalTechnologyAnnualActivityLowerLimit': False,
    'TotalTechnologyAnnualActivityUpperLimit': False,
    'TotalTechnologyModelPeriodActivityUpperLimit': False,
    'TotalTechnologyModelPeriodActivityLowerLimit': False,
    'ReserveMarginTagTechnology': False,
    'ReserveMarginTagFuel': False,
    'ReserveMargin': False,
    'RETagTechnology': False,
    'RETagFuel': False,
    'REMinProductionTarget': False,
    'EmissionActivityRatio': False,
    'EmissionsPenalty': False,
    'AnnualExogenousEmission': False,
    'AnnualEmissionLimit': False,
    'ModelPeriodExogenousEmission': False,
    'ModelPeriodEmissionLimit': False
}
# Sets the default toggles (To only use defaults)
# toggle_defaults = {
#     'YearSplit': False,
#     'DiscountRate': False,
#     'DaySplit': False,
#     'Conversionls': False,
#     'Conversionld': True,
#     'Conversionlh': True,
#     'DaysInDayType': True,
#     'TradeRoute': True,
#     'DepreciationMethod': True,
#     'SpecifiedAnnualDemand': True,
#     'SpecifiedDemandProfile': True,
#     'AccumulatedAnnualDemand': True,
#     'CapacityToActivityUnit': True,
#     'CapacityFactor': True,
#     'AvailabilityFactor': True,
#     'OperationalLife': True,
#     'ResidualCapacity': True,
#     'InputActivityRatio': True,
#     'OutputActivityRatio': True,
#     'CapitalCost': True,
#     'VariableCost': True,
#     'FixedCost': True,
#     'TechnologyToStorage': True,
#     'TechnologyFromStorage': True,
#     'StorageLevelStart': True,
#      'StorageMaxChargeRate': True,
#      'StorageMaxDischargeRate': True,
#      'MinStorageCharge': True,
#      'OperationalLifeStorage': True,
#      'CapitalCostStorage': True,
#      'ResidualStorageCapacity': True,
#      'CapacityOfOneTechnologyUnit': True,
#      'TotalAnnualMaxCapacity': True,
#      'TotalAnnualMinCapacity': True,
#      'TotalAnnualMaxCapacityInvestment': True,
#      'TotalAnnualMinCapacityInvestment': True,
#      'TotalTechnologyAnnualActivityLowerLimit': True,
#      'TotalTechnologyAnnualActivityUpperLimit': True,
#      'TotalTechnologyModelPeriodActivityUpperLimit': True,
#      'TotalTechnologyModelPeriodActivityLowerLimit': True,
#      'ReserveMarginTagTechnology': True,
#      'ReserveMarginTagFuel': True,
#      'ReserveMargin': True,
#      'RETagTechnology': True,
#      'RETagFuel': True,
#      'REMinProductionTarget': True,
#      'EmissionActivityRatio': False,
#      'EmissionsPenalty': False,
#      'AnnualExogenousEmission': False,
#      'AnnualEmissionLimit': False,
#      'ModelPeriodExogenousEmission': False,
#      'ModelPeriodEmissionLimit': False
# }

# Create the Data File
system.create_data_file(data_location_1, default_parameters,
   toggle_defaults)

# Cereate the Model File
system.create_model_file(root, model_source_file)
```

## 2 OseMOSYS

```
set YEAR;
set TECHNOLOGY;
set TIMESLICE;
set FUEL;
set EMISSION;
set MODE_OF_OPERATION;
set REGION;
set SEASON;
set DAYTYPE;
set DAILYTIMEBRACKET;
set STORAGE;
param YearSplit{l in TIMESLICE,y in YEAR};
param DiscountRate{r in REGION};
param DaySplit{lh in DAILYTIMEBRACKET,y in YEAR};
param Conversionls{l in TIMESLICE,ls in SEASON};
param Conversionld{l in TIMESLICE,ld in DAYTYPE};
param Conversionlh{l in TIMESLICE,lh in DAILYTIMEBRACKET};
param DaysInDayType{ls in SEASON ,ld in DAYTYPE,y in YEAR};
param TradeRoute{r in REGION,rr in REGION,f in FUEL,y in YEAR};
param DepreciationMethod{r in REGION};
param SpecifiedAnnualDemand{r in REGION,f in FUEL,y in YEAR};
param SpecifiedDemandProfile{r in REGION,f in FUEL,l in TIMESLICE,y in YEAR};
param AccumulatedAnnualDemand{r in REGION,f in FUEL,y in YEAR};
param CapacityToActivityUnit{r in REGION,t in TECHNOLOGY};
param CapacityFactor{r in REGION,t in TECHNOLOGY,l in TIMESLICE,y in YEAR};
param AvailabilityFactor{r in REGION,t in TECHNOLOGY,y in YEAR};
param OperationalLife{r in REGION,t in TECHNOLOGY};
param ResidualCapacity{r in REGION,t in TECHNOLOGY,y in YEAR};
param InputActivityRatio{r in REGION,t in TECHNOLOGY,f in FUEL,m in MODE_OF_OPERATION,y in YEAR};
param OutputActivityRatio{r in REGION,t in TECHNOLOGY,f in FUEL,m in MODE_OF_OPERATION,y in YEAR};
param CapitalCost{r in REGION,t in TECHNOLOGY,y in YEAR};
param VariableCost{r in REGION,t in TECHNOLOGY,m in MODE_OF_OPERATION,y in YEAR};
param FixedCost{r in REGION,t in TECHNOLOGY,y in YEAR};
param TechnologyToStorage{r in REGION,t in TECHNOLOGY,s in STORAGE,m in MODE_OF_OPERATION};
param TechnologyFromStorage{r in REGION,t in TECHNOLOGY,s in STORAGE,m in MODE_OF_OPERATION};
param StorageLevelStart{r in REGION,s in STORAGE};
param StorageMaxChargeRate{r in REGION,s in STORAGE};
param StorageMaxDischargeRate{r in REGION,s in STORAGE};
param MinStorageCharge{r in REGION,s in STORAGE,y in YEAR};
param OperationalLifeStorage{r in REGION, s in STORAGE};
param CapitalCostStorage{r in REGION,s in STORAGE,y in YEAR};
param ResidualStorageCapacity{r in REGION,s in STORAGE,y in YEAR};
param CapacityOfOneTechnologyUnit{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalAnnualMaxCapacity{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalAnnualMinCapacity{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalAnnualMaxCapacityInvestment{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalAnnualMinCapacityInvestment{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalTechnologyAnnualActivityUpperLimit{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalTechnologyAnnualActivityLowerLimit{r in REGION,t in TECHNOLOGY,y in YEAR};
param TotalTechnologyModelPeriodActivityUpperLimit{r in REGION,t in TECHNOLOGY};
param TotalTechnologyModelPeriodActivityLowerLimit{r in REGION,t in TECHNOLOGY};
param ReserveMarginTagTechnology{r in REGION,t in TECHNOLOGY,y in YEAR};
param ReserveMarginTagFuel{r in REGION,f in FUEL,y in YEAR};
param ReserveMargin{r in REGION,y in YEAR};
param RETagTechnology{r in REGION,t in TECHNOLOGY,y in YEAR};
param RETagFuel{r in REGION,f in FUEL,y in YEAR};
param REMinProductionTarget{r in REGION,y in YEAR};
param EmissionActivityRatio{r in REGION,t in TECHNOLOGY,e in EMISSION,m in MODE_OF_OPERATION,y in YEAR};
param EmissionsPenalty{r in REGION,e in EMISSION,y in YEAR};
param AnnualExogenousEmission{r in REGION,e in EMISSION,y in YEAR};
param AnnualEmissionLimit{r in REGION,e in EMISSION,y in YEAR};
param ModelPeriodExogenousEmission{r in REGION,e in EMISSION};
param ModelPeriodEmissionLimit{r in REGION,e in EMISSION};
var RateOfDemand{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR} >=0;
var Demand{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR}>=0;
var RateOfStorageCharge{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,lh in DAILYTIMEBRACKET,y in YEAR};
var RateOfStorageDischarge{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,lh in DAILYTIMEBRACKET,y in YEAR};
var NetChargeWithinYear{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,lh in DAILYTIMEBRACKET,y in YEAR};
var NetChargeWithinDay{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,lh in DAILYTIMEBRACKET,y in YEAR};
var StorageLevelYearStart{r in REGION,s in STORAGE,y in YEAR}>=0;
var StorageLevelYearFinish{r in REGION,s in STORAGE,y in YEAR}>=0;
var StorageLevelSeasonStart{r in REGION,s in STORAGE,ls in SEASON,y in YEAR}>=0;
var StorageLevelDayTypeStart{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,y in YEAR}>=0;
var StorageLevelDayTypeFinish{r in REGION,s in STORAGE,ls in SEASON,ld in DAYTYPE,y in YEAR}>=0;
var StorageLowerLimit{r in REGION,s in STORAGE,y in YEAR}>=0;
var StorageUpperLimit{r in REGION,s in STORAGE,y in YEAR}>=0;
var AccumulatedNewStorageCapacity{r in REGION,s in STORAGE,y in YEAR}>=0;
var NewStorageCapacity{r in REGION,s in STORAGE,y in YEAR}>=0;
var CapitalInvestmentStorage{r in REGION,s in STORAGE,y in YEAR}>=0;
var DiscountedCapitalInvestmentStorage{r in REGION,s in STORAGE,y in YEAR}>=0;
var SalvageValueStorage{r in REGION,s in STORAGE,y in YEAR}>=0;
var DiscountedSalvageValueStorage{r in REGION,s in STORAGE,y in YEAR}>=0;
var TotalDiscountedStorageCost{r in REGION,s in STORAGE,y in YEAR}>=0;
var NumberOfNewTechnologyUnits{r in REGION,t in TECHNOLOGY,y in YEAR}>=0, integer;
var NewCapacity{r in REGION,t in TECHNOLOGY,y in YEAR}>=0;
var AccumulatedNewCapacity{r in REGION,t in TECHNOLOGY,y in YEAR}>=0;
var TotalCapacityAnnual{r in REGION,t in TECHNOLOGY,y in YEAR}>=0;
var RateOfActivity{r in REGION,l in TIMESLICE,t in TECHNOLOGY,m in MODE_OF_OPERATION,y in YEAR} >=0;
var RateOfTotalActivity{r in REGION,t in TECHNOLOGY,l in TIMESLICE,y in YEAR} >=0;
var TotalTechnologyAnnualActivity{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var TotalAnnualTechnologyActivityByMode{r in REGION,t in TECHNOLOGY,m in MODE_OF_OPERATION,y in YEAR} >=0;
var TotalTechnologyModelPeriodActivity{r in REGION,t in TECHNOLOGY};
var RateOfProductionByTechnologyByMode{r in REGION,l in TIMESLICE,t in TECHNOLOGY,m in MODE_OF_OPERATION,f in FUEL,y in YEAR}>=0;
var RateOfProductionByTechnology{r in REGION,l in TIMESLICE,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var ProductionByTechnology{r in REGION,l in TIMESLICE,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var ProductionByTechnologyAnnual{r in REGION,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var RateOfProduction{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR}>=0;
var Production{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR} >=0;
var RateOfUseByTechnologyByMode{r in REGION,l in TIMESLICE,t in
TECHNOLOGY,m in MODE_OF_OPERATION,f in FUEL,y in YEAR} >=0;
var RateOfUseByTechnology{r in REGION,l in TIMESLICE,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var UseByTechnologyAnnual{r in REGION,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var UseByTechnology{r in REGION,l in TIMESLICE,t in TECHNOLOGY,f in FUEL,y in YEAR} >=0;
var RateOfUse{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR}>=0;
var Use{r in REGION,l in TIMESLICE,f in FUEL,y in YEAR} >=0;
var Trade{r in REGION,rr in REGION,l in TIMESLICE,f in FUEL,y in YEAR};
var TradeAnnual{r in REGION,rr in REGION,f in FUEL,y in YEAR};
var ProductionAnnual{r in REGION,f in FUEL,y in YEAR} >=0;
var UseAnnual{r in REGION,f in FUEL,y in YEAR}>=0;
var CapitalInvestment{r in REGION,t in TECHNOLOGY,y in YEAR}>=0;
var DiscountedCapitalInvestment{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var SalvageValue{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var DiscountedSalvageValue{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var OperatingCost{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var DiscountedOperatingCost{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var AnnualVariableOperatingCost{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var AnnualFixedOperatingCost{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var TotalDiscountedCostByTechnology{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var TotalDiscountedCost{r in REGION,y in YEAR} >=0;
var ModelPeriodCostByRegion{r in REGION}>=0;
var TotalCapacityInReserveMargin{r in REGION,y in YEAR} >=0;
var DemandNeedingReserveMargin{r in REGION,l in TIMESLICE,y in YEAR} >=0;
var TotalREProductionAnnual{r in REGION,y in YEAR};
var RETotalProductionOfTargetFuelAnnual{r in REGION,y in YEAR};
var AnnualTechnologyEmissionByMode{r in REGION,t in TECHNOLOGY,e in EMISSION,m in MODE_OF_OPERATION,y in YEAR} >=0;
var AnnualTechnologyEmission{r in REGION,t in TECHNOLOGY,e in EMISSION,y in YEAR} >=0;
var AnnualTechnologyEmissionPenaltyByEmission{r in REGION,t in TECHNOLOGY,e in EMISSION,y in YEAR} >=0;
var AnnualTechnologyEmissionsPenalty{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var DiscountedTechnologyEmissionsPenalty{r in REGION,t in TECHNOLOGY,y in YEAR} >=0;
var AnnualEmissions{r in REGION,e in EMISSION,y in YEAR} >=0;
var ModelPeriodEmissions{r in REGION,e in EMISSION} >=0;
minimize cost: sum{r in REGION, y in YEAR} TotalDiscountedCost[r,y];
s.t. EQ_SpecifiedDemand{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: SpecifiedAnnualDemand[r,f,y]*SpecifiedDemandProfile[r,f,l,y] / YearSplit[l,y]=RateOfDemand[r,l,f,y];
s.t. CAa1_TotalNewCapacity{r in REGION, t in TECHNOLOGY, y in YEAR}: AccumulatedNewCapacity[r,t,y] = sum{yy in YEAR: y-yy < OperationalLife[r,t] && y-yy>=0} NewCapacity[r,t,yy];
s.t. CAa2_TotalAnnualCapacity{r in REGION, t in TECHNOLOGY, y in YEAR}: AccumulatedNewCapacity[r,t,y]+ ResidualCapacity[r,t,y] = TotalCapacityAnnual[r,t,y];
s.t. CAa3_TotalActivityOfEachTechnology{r in REGION, t in TECHNOLOGY, l in TIMESLICE, y in YEAR}: sum{m in MODE_OF_OPERATION} RateOfActivity[r,l,t,m,y] = RateOfTotalActivity[r,t,l,y];
```

```
s.t. CAa4_Constraint_Capacity{r in REGION, l in TIMESLICE, t in TECHNOLOGY, y in YEAR}: RateOfTotalActivity[r,t,l,y] <= TotalCapacityAnnual[r,t,y] * CapacityFactor[r,t,l,y]*CapacityToActivityUnit[r,t];
s.t. CAa5_TotalNewCapacity{r in REGION, t in TECHNOLOGY, y in YEAR: CapacityOfOneTechnologyUnit[r,t,y]<>0}: CapacityOfOneTechnologyUnit[r,t,y]*NumberOfNewTechnologyUnits[r,t,y] = NewCapacity[r,t,y];
s.t. CAb1_PlannedMaintenance{r in REGION, t in TECHNOLOGY, y in YEAR}: sum{l in TIMESLICE} RateOfTotalActivity[r,t,l,y]*YearSplit[l,y] <= sum{l in TIMESLICE} (TotalCapacityAnnual[r,t,y]*CapacityFactor[r,t,l,y]*YearSplit[l,y])* AvailabilityFactor[r,t,y]*CapacityToActivityUnit[r,t];
s.t. EBa1_RateOfFuelProduction1{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, m in MODE_OF_OPERATION, y in YEAR: OutputActivityRatio[r,t,f,m,y] <>0}: RateOfActivity[r,l,t,m,y]*OutputActivityRatio[r,t,f,m,y]  = RateOfProductionByTechnologyByMode[r,l,t,m,f,y];
s.t. EBa2_RateOfFuelProduction2{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, y in YEAR}: sum{m in MODE_OF_OPERATION: OutputActivityRatio[r,t,f,m,y] <>0} RateOfProductionByTechnologyByMode[r,l,t,m,f,y] = RateOfProductionByTechnology[r,l,t,f,y];
s.t. EBa3_RateOfFuelProduction3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: sum{t in TECHNOLOGY} RateOfProductionByTechnology[r,l,t,f,y]  =  RateOfProduction[r,l,f,y];
s.t. EBa4_RateOfFuelUse1{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, m in MODE_OF_OPERATION, y in YEAR: InputActivityRatio[r,t,f,m,y]<>0}: RateOfActivity[r,l,t,m,y]*InputActivityRatio[r,t,f,m,y]  = RateOfUseByTechnologyByMode[r,l,t,m,f,y];
s.t. EBa5_RateOfFuelUse2{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, y in YEAR}: sum{m in MODE_OF_OPERATION: InputActivityRatio[r,t,f,m,y]<>0} RateOfUseByTechnologyByMode[r,l,t,m,f,y] = RateOfUseByTechnology[r,l,t,f,y];
s.t. EBa6_RateOfFuelUse3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: sum{t in TECHNOLOGY} RateOfUseByTechnology[r,l,t,f,y]  = RateOfUse[r,l,f,y];
s.t. EBa7_EnergyBalanceEachTS1{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: RateOfProduction[r,l,f,y]*YearSplit[l,y] = Production[r,l,f,y];
s.t. EBa8_EnergyBalanceEachTS2{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: RateOfUse[r,l,f,y]*YearSplit[l,y] = Use[r,l,f,y];
s.t. EBa9_EnergyBalanceEachTS3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: RateOfDemand[r,l,f,y]*YearSplit[l,y] = Demand[r,l,f,y];
s.t. EBa10_EnergyBalanceEachTS4{r in REGION, rr in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: Trade[r,rr,l,f,y] = -Trade[rr,r,l,f,y];
s.t. EBa11_EnergyBalanceEachTS5{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}: Production[r,l,f,y] >= Demand[r,l,f,y] + Use[r,l,f,y] + sum{rr in REGION} Trade[r,rr,l,f,y]*TradeRoute[r,rr,f,y];
s.t. EBb1_EnergyBalanceEachYear1{r in REGION, f in FUEL, y in YEAR}: sum{l in TIMESLICE} Production[r,l,f,y] = ProductionAnnual[r,f,y];
s.t. EBb2_EnergyBalanceEachYear2{r in REGION, f in FUEL, y in YEAR}: sum{l in TIMESLICE} Use[r,l,f,y] = UseAnnual[r,f,y];
s.t. EBb3_EnergyBalanceEachYear3{r in REGION, rr in REGION, f in FUEL, y in YEAR}: sum{l in TIMESLICE} Trade[r,rr,l,f,y] = TradeAnnual[r,rr,f,y];
s.t. EBb4_EnergyBalanceEachYear4{r in REGION, f in FUEL, y in YEAR}: ProductionAnnual[r,f,y] >= UseAnnual[r,f,y] + sum{rr in REGION}
```

```
TradeAnnual[r,rr,f,y]*TradeRoute[r,rr,f,y] + AccumulatedAnnualDemand[r,f,y];
s.t. Acc1_FuelProductionByTechnology{r in REGION, l in TIMESLICE, t in TECHNOLOGY, f in FUEL, y in YEAR}: RateOfProductionByTechnology[r,l,t,f,y] * YearSplit[l,y] = ProductionByTechnology[r,l,t,f,y];
s.t. Acc2_FuelUseByTechnology{r in REGION, l in TIMESLICE, t in TECHNOLOGY, f in FUEL, y in YEAR}: RateOfUseByTechnology[r,l,t,f,y] * YearSplit[l,y] = UseByTechnology[r,l,t,f,y];
s.t. Acc3_AverageAnnualRateOfActivity{r in REGION, t in TECHNOLOGY, m in MODE_OF_OPERATION, y in YEAR}: sum{l in TIMESLICE} RateOfActivity[r,l,t,m,y]*YearSplit[l,y] = TotalAnnualTechnologyActivityByMode[r,t,m,y];
s.t. Acc4_ModelPeriodCostByRegion{r in REGION}: sum{y in YEAR} TotalDiscountedCost[r,y] = ModelPeriodCostByRegion[r];
s.t. S1_RateOfStorageCharge{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: sum{t in TECHNOLOGY, m in MODE_OF_OPERATION, l in TIMESLICE:TechnologyToStorage[r,t,s,m]>0} RateOfActivity[r,l,t,m,y] * TechnologyToStorage[r,t,s,m] * Conversionls[l,ls] * Conversionld[l,ld] * Conversionlh[l,lh] = RateOfStorageCharge[r,s,ls,ld,lh,y];
s.t. S2_RateOfStorageDischarge{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: sum{t in TECHNOLOGY, m in MODE_OF_OPERATION, l in TIMESLICE:TechnologyFromStorage[r,t,s,m]>0} RateOfActivity[r,l,t,m,y] * TechnologyFromStorage[r,t,s,m] * Conversionls[l,ls] * Conversionld[l,ld] * Conversionlh[l,lh] = RateOfStorageDischarge[r,s,ls,ld,lh,y];
s.t. S3_NetChargeWithinYear{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: sum{l in TIMESLICE:Conversionls[l,ls]>0&&Conversionld[l,ld]>0&&Conversionlh[l,lh]>0}  (RateOfStorageCharge[r,s,ls,ld,lh,y] - RateOfStorageDischarge[r,s,ls,ld,lh,y]) * YearSplit[l,y] * Conversionls[l,ls] * Conversionld[l,ld] * Conversionlh[l,lh] = NetChargeWithinYear[r,s,ls,ld,lh,y];
s.t. S4_NetChargeWithinDay{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: (RateOfStorageCharge[r,s,ls,ld,lh,y] - RateOfStorageDischarge[r,s,ls,ld,lh,y]) * DaySplit[lh,y] = NetChargeWithinDay[r,s,ls,ld,lh,y];
s.t. S5_and_S6_StorageLevelYearStart{r in REGION, s in STORAGE, y in YEAR}: if y = min{yy in YEAR} min(yy) then StorageLevelStart[r,s] else StorageLevelYearStart[r,s,y-1] + sum{ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET} NetChargeWithinYear[r,s,ls,ld,lh,y-1] = StorageLevelYearStart[r,s,y];
s.t. S7_and_S8_StorageLevelYearFinish{r in REGION, s in STORAGE, y in YEAR}: if y < max{yy in YEAR} max(yy) then StorageLevelYearStart[r,s,y+1] else StorageLevelYearStart[r,s,y] + sum{ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET} NetChargeWithinYear[r,s,ls,ld,lh,y] = StorageLevelYearFinish[r,s,y];
s.t. S9_and_S10_StorageLevelSeasonStart{r in REGION, s in STORAGE, ls in SEASON, y in YEAR}: if ls = min{lsls in SEASON} min(lsls) then StorageLevelYearStart[r,s,y] else StorageLevelSeasonStart[r,s,ls-1,y] + sum{ld in DAYTYPE, lh in DAILYTIMEBRACKET} NetChargeWithinYear[r,s,ls-1,ld,lh,y] = StorageLevelSeasonStart[r,s,ls,y];
s.t. S11_and_S12_StorageLevelDayTypeStart{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, y in YEAR}: if ld = min{ldld in DAYTYPE} min(ldld) then StorageLevelSeasonStart[r,s,ls,y] else StorageLevelDayTypeStart[r,s,ls,ld-1,y] + sum{lh in DAILYTIMEBRACKET} NetChargeWithinDay[r,s,ls,ld-1,lh,y] * DaysInDayType[ls,ld-1,y] = StorageLevelDayTypeStart[r,s,ls,ld,y];
s.t. S13_and_S14_and_S15_StorageLevelDayTypeFinish{r in REGION, s in
```

```
    STORAGE, ls in SEASON, ld in DAYTYPE, y in YEAR}:  if ls = max{lsls
    in SEASON} max(lsls) && ld = max{ldld in DAYTYPE} max(ldld) then
    StorageLevelYearFinish[r,s,y] else if ld = max{ldld in DAYTYPE} max(
    ldld) then StorageLevelSeasonStart[r,s,ls+1,y] else
    StorageLevelDayTypeFinish[r,s,ls,ld+1,y] - sum{lh in
    DAILYTIMEBRACKET} NetChargeWithinDay[r,s,ls,ld+1,lh,y] *
    DaysInDayType[ls,ld+1,y] = StorageLevelDayTypeFinish[r,s,ls,ld,y];
s.t.
    SC1_LowerLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWeekCons
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: 0 <= (StorageLevelDayTypeStart[r,s,ls,
    ld,y]+sum{lhlh in DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s
    ,ls,ld,lhlh,y])-StorageLowerLimit[r,s,y];
s.t.
    SC1_UpperLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWeekCons
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: (StorageLevelDayTypeStart[r,s,ls,ld,y
    ]+sum{lhlh in DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s,ls,
    ld,lhlh,y])-StorageUpperLimit[r,s,y] <= 0;
s.t.
    SC2_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: 0 <= if ld > min{ldld in DAYTYPE} min(
    ldld) then (StorageLevelDayTypeStart[r,s,ls,ld,y]-sum{lhlh in
    DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,ls,ld-1,lhlh,y])-
    StorageLowerLimit[r,s,y];
s.t.
    SC2_UpperLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: if ld > min{ldld in DAYTYPE} min(ldld)
     then (StorageLevelDayTypeStart[r,s,ls,ld,y]-sum{lhlh in
    DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,ls,ld-1,lhlh,y])-
    StorageUpperLimit[r,s,y] <= 0;
s.t.
    SC3_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: 0 <= (StorageLevelDayTypeFinish[r,s,ls
    ,ld,y] - sum{lhlh in DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[
    r,s,ls,ld,lhlh,y])-StorageLowerLimit[r,s,y];
s.t.
    SC3_UpperLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: (StorageLevelDayTypeFinish[r,s,ls,ld,y
    ] - sum{lhlh in DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,
    ls,ld,lhlh,y])-StorageUpperLimit[r,s,y] <= 0;
s.t.
    SC4_LowerLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWeekCons
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: 0 <= if ld > min{ldld in DAYTYPE} min(
    ldld) then (StorageLevelDayTypeFinish[r,s,ls,ld-1,y]+sum{lhlh in
    DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s,ls,ld,lhlh,y])-
    StorageLowerLimit[r,s,y];
s.t.
    SC4_UpperLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWeekCons
    {r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in
    DAILYTIMEBRACKET, y in YEAR}: if ld > min{ldld in DAYTYPE} min(ldld)
     then (StorageLevelDayTypeFinish[r,s,ls,ld-1,y]+sum{lhlh in
    DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s,ls,ld,lhlh,y])-
```

```
StorageUpperLimit[r,s,y] <= 0;
s.t. SC5_MaxChargeConstraint{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: RateOfStorageCharge[r,s,ls,ld,lh,y] <= StorageMaxChargeRate[r,s];
s.t. SC6_MaxDischargeConstraint{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: RateOfStorageDischarge[r,s,ls,ld,lh,y] <= StorageMaxDischargeRate[r,s];
s.t. SI1_StorageUpperLimit{r in REGION, s in STORAGE, y in YEAR}: AccumulatedNewStorageCapacity[r,s,y]+ResidualStorageCapacity[r,s,y] = StorageUpperLimit[r,s,y];
s.t. SI2_StorageLowerLimit{r in REGION, s in STORAGE, y in YEAR}: MinStorageCharge[r,s,y]*StorageUpperLimit[r,s,y] = StorageLowerLimit[r,s,y];
s.t. SI3_TotalNewStorage{r in REGION, s in STORAGE, y in YEAR}: sum{yy in YEAR: y-yy < OperationalLifeStorage[r,s] && y-yy>=0} NewStorageCapacity[r,s,yy]=AccumulatedNewStorageCapacity[r,s,y];
s.t. SI4_UndiscountedCapitalInvestmentStorage{r in REGION, s in STORAGE, y in YEAR}: CapitalCostStorage[r,s,y] * NewStorageCapacity[r,s,y] = CapitalInvestmentStorage[r,s,y];
s.t. SI5_DiscountingCapitalInvestmentStorage{r in REGION, s in STORAGE, y in YEAR}: CapitalInvestmentStorage[r,s,y]/((1+DiscountRate[r])^(y-min{yy in YEAR} min(yy))) = DiscountedCapitalInvestmentStorage[r,s,y];
s.t. SI6_SalvageValueStorageAtEndOfPeriod1{r in REGION, s in STORAGE, y in YEAR: (y+OperationalLifeStorage[r,s]-1) <= (max{yy in YEAR} max(yy))}: 0 = SalvageValueStorage[r,s,y];
s.t. SI7_SalvageValueStorageAtEndOfPeriod2{r in REGION, s in STORAGE, y in YEAR: (DepreciationMethod[r]=1 && (y+OperationalLifeStorage[r,s]-1) > (max{yy in YEAR} max(yy)) && DiscountRate[r]=0) || (DepreciationMethod[r]=2 && (y+OperationalLifeStorage[r,s]-1) > (max{yy in YEAR} max(yy)))}: CapitalInvestmentStorage[r,s,y]*(1-(max{yy in YEAR} max(yy) - y+1)/OperationalLifeStorage[r,s]) = SalvageValueStorage[r,s,y];
s.t. SI8_SalvageValueStorageAtEndOfPeriod3{r in REGION, s in STORAGE, y in YEAR: DepreciationMethod[r]=1 && (y+OperationalLifeStorage[r,s]-1) > (max{yy in YEAR} max(yy)) && DiscountRate[r]>0}: CapitalInvestmentStorage[r,s,y]*(1-(((1+DiscountRate[r])^(max{yy in YEAR} max(yy) - y+1)-1)/((1+DiscountRate[r])^OperationalLifeStorage[r,s]-1))) = SalvageValueStorage[r,s,y];
s.t. SI9_SalvageValueStorageDiscountedToStartYear{r in REGION, s in STORAGE, y in YEAR}: SalvageValueStorage[r,s,y]/((1+DiscountRate[r])^(max{yy in YEAR} max(yy)-min{yy in YEAR} min(yy)+1)) = DiscountedSalvageValueStorage[r,s,y];
s.t. SI10_TotalDiscountedCostByStorage{r in REGION, s in STORAGE, y in YEAR}: DiscountedCapitalInvestmentStorage[r,s,y]-DiscountedSalvageValueStorage[r,s,y] = TotalDiscountedStorageCost[r,s,y];
s.t. CC1_UndiscountedCapitalInvestment{r in REGION, t in TECHNOLOGY, y in YEAR}: CapitalCost[r,t,y] * NewCapacity[r,t,y] = CapitalInvestment[r,t,y];
s.t. CC2_DiscountingCapitalInvestment{r in REGION, t in TECHNOLOGY, y in YEAR}: CapitalInvestment[r,t,y]/((1+DiscountRate[r])^(y-min{yy in YEAR} min(yy))) = DiscountedCapitalInvestment[r,t,y];
s.t. SV1_SalvageValueAtEndOfPeriod1{r in REGION, t in TECHNOLOGY, y in YEAR: DepreciationMethod[r]=1 && (y + OperationalLife[r,t]-1) > (max{yy in YEAR} max(yy)) && DiscountRate[r]>0}: SalvageValue[r,t,y] = CapitalCost[r,t,y]*NewCapacity[r,t,y]*(1-(((1+DiscountRate[r])^(max{
   yy in YEAR} max(yy) - y+1)-1)/((1+DiscountRate[r])^OperationalLife[r
   ,t]-1)));
s.t. SV2_SalvageValueAtEndOfPeriod2{r in REGION, t in TECHNOLOGY, y in
   YEAR: (DepreciationMethod[r]=1 && (y + OperationalLife[r,t]-1) > (
   max{yy in YEAR} max(yy)) && DiscountRate[r]=0) || (
   DepreciationMethod[r]=2 && (y + OperationalLife[r,t]-1) > (max{yy in
    YEAR} max(yy)))}: SalvageValue[r,t,y] = CapitalCost[r,t,y]*
   NewCapacity[r,t,y]*(1-(max{yy in YEAR} max(yy) - y+1)/
   OperationalLife[r,t]);
s.t. SV3_SalvageValueAtEndOfPeriod3{r in REGION, t in TECHNOLOGY, y in
   YEAR: (y + OperationalLife[r,t]-1) <= (max{yy in YEAR} max(yy))}:
   SalvageValue[r,t,y] = 0;
s.t. SV4_SalvageValueDiscountedToStartYear{r in REGION, t in TECHNOLOGY
   , y in YEAR}: DiscountedSalvageValue[r,t,y] = SalvageValue[r,t,y
   ]/((1+DiscountRate[r])^(1+max{yy in YEAR} max(yy)-min{yy in YEAR}
   min(yy)));
s.t. OC1_OperatingCostsVariable{r in REGION, t in TECHNOLOGY, l in
   TIMESLICE, y in YEAR}: sum{m in MODE_OF_OPERATION}
   TotalAnnualTechnologyActivityByMode[r,t,m,y]*VariableCost[r,t,m,y] =
    AnnualVariableOperatingCost[r,t,y];
s.t. OC2_OperatingCostsFixedAnnual{r in REGION, t in TECHNOLOGY, y in
   YEAR}: TotalCapacityAnnual[r,t,y]*FixedCost[r,t,y] =
   AnnualFixedOperatingCost[r,t,y];
s.t. OC3_OperatingCostsTotalAnnual{r in REGION, t in TECHNOLOGY, y in
   YEAR}: AnnualFixedOperatingCost[r,t,y]+AnnualVariableOperatingCost[r
   ,t,y] = OperatingCost[r,t,y];
s.t. OC4_DiscountedOperatingCostsTotalAnnual{r in REGION, t in
   TECHNOLOGY, y in YEAR}: OperatingCost[r,t,y]/((1+DiscountRate[r])^(y
   -min{yy in YEAR} min(yy)+0.5)) = DiscountedOperatingCost[r,t,y];
s.t. TDC1_TotalDiscountedCostByTechnology{r in REGION, t in TECHNOLOGY,
    y in YEAR}: DiscountedOperatingCost[r,t,y]+
   DiscountedCapitalInvestment[r,t,y]+
   DiscountedTechnologyEmissionsPenalty[r,t,y]-DiscountedSalvageValue[r
   ,t,y] = TotalDiscountedCostByTechnology[r,t,y];
s.t. TDC2_TotalDiscountedCost{r in REGION, y in YEAR}: sum{t in
   TECHNOLOGY} TotalDiscountedCostByTechnology[r,t,y]+sum{s in STORAGE}
    TotalDiscountedStorageCost[r,s,y] = TotalDiscountedCost[r,y];
s.t. TCC1_TotalAnnualMaxCapacityConstraint{r in REGION, t in TECHNOLOGY
   , y in YEAR}: TotalCapacityAnnual[r,t,y] <= TotalAnnualMaxCapacity[r
   ,t,y];
s.t. TCC2_TotalAnnualMinCapacityConstraint{r in REGION, t in TECHNOLOGY
   , y in YEAR: TotalAnnualMinCapacity[r,t,y]>0}: TotalCapacityAnnual[r
   ,t,y] >= TotalAnnualMinCapacity[r,t,y];
s.t. NCC1_TotalAnnualMaxNewCapacityConstraint{r in REGION, t in
   TECHNOLOGY, y in YEAR}: NewCapacity[r,t,y] <=
   TotalAnnualMaxCapacityInvestment[r,t,y];
s.t. NCC2_TotalAnnualMinNewCapacityConstraint{r in REGION, t in
   TECHNOLOGY, y in YEAR: TotalAnnualMinCapacityInvestment[r,t,y]>0}:
   NewCapacity[r,t,y] >= TotalAnnualMinCapacityInvestment[r,t,y];
s.t. AAC1_TotalAnnualTechnologyActivity{r in REGION, t in TECHNOLOGY, y
    in YEAR}: sum{l in TIMESLICE} RateOfTotalActivity[r,t,l,y]*
   YearSplit[l,y] = TotalTechnologyAnnualActivity[r,t,y];
s.t. AAC2_TotalAnnualTechnologyActivityUpperLimit{r in REGION, t in
   TECHNOLOGY, y in YEAR}: TotalTechnologyAnnualActivity[r,t,y] <=
   TotalTechnologyAnnualActivityUpperLimit[r,t,y] ;
s.t. AAC3_TotalAnnualTechnologyActivityLowerLimit{r in REGION, t in
   TECHNOLOGY, y in YEAR: TotalTechnologyAnnualActivityLowerLimit[r,t,y
   ]>0}: TotalTechnologyAnnualActivity[r,t,y] >=
TotalTechnologyAnnualActivityLowerLimit[r,t,y];
s.t. TAC1_TotalModelHorizonTechnologyActivity{r in REGION, t in TECHNOLOGY}: sum{y in YEAR} TotalTechnologyAnnualActivity[r,t,y] = TotalTechnologyModelPeriodActivity[r,t];
s.t. TAC2_TotalModelHorizonTechnologyActivityUpperLimit{r in REGION, t in TECHNOLOGY: TotalTechnologyModelPeriodActivityUpperLimit[r,t]>0}: TotalTechnologyModelPeriodActivity[r,t] <= TotalTechnologyModelPeriodActivityUpperLimit[r,t] ;
s.t. TAC3_TotalModelHorizenTechnologyActivityLowerLimit{r in REGION, t in TECHNOLOGY: TotalTechnologyModelPeriodActivityLowerLimit[r,t]>0}: TotalTechnologyModelPeriodActivity[r,t] >= TotalTechnologyModelPeriodActivityLowerLimit[r,t] ;
s.t. RM1_ReserveMargin_TechnologiesIncluded_In_Activity_Units{r in REGION, l in TIMESLICE, y in YEAR}: sum {t in TECHNOLOGY} TotalCapacityAnnual[r,t,y] * ReserveMarginTagTechnology[r,t,y] * CapacityToActivityUnit[r,t]       = TotalCapacityInReserveMargin[r,y];
s.t. RM2_ReserveMargin_FuelsIncluded{r in REGION, l in TIMESLICE, y in YEAR}: sum {f in FUEL} RateOfProduction[r,l,f,y] * ReserveMarginTagFuel[r,f,y] = DemandNeedingReserveMargin[r,l,y];
s.t. RM3_ReserveMargin_Constraint{r in REGION, l in TIMESLICE, y in YEAR}: DemandNeedingReserveMargin[r,l,y] * ReserveMargin[r,y]<= TotalCapacityInReserveMargin[r,y];
s.t. RE1_FuelProductionByTechnologyAnnual{r in REGION, t in TECHNOLOGY, f in FUEL, y in YEAR}: sum{l in TIMESLICE} ProductionByTechnology[r,l,t,f,y] = ProductionByTechnologyAnnual[r,t,f,y];
s.t. RE2_TechIncluded{r in REGION, y in YEAR}: sum{t in TECHNOLOGY, f in FUEL} ProductionByTechnologyAnnual[r,t,f,y]*RETagTechnology[r,t,y] = TotalREProductionAnnual[r,y];
s.t. RE3_FuelIncluded{r in REGION, y in YEAR}: sum{l in TIMESLICE, f in FUEL} RateOfProduction[r,l,f,y]*YearSplit[l,y]*RETagFuel[r,f,y] = RETotalProductionOfTargetFuelAnnual[r,y];
s.t. RE4_EnergyConstraint{r in REGION, y in YEAR}: REMinProductionTarget[r,y]*RETotalProductionOfTargetFuelAnnual[r,y] <= TotalREProductionAnnual[r,y];
s.t. RE5_FuelUseByTechnologyAnnual{r in REGION, t in TECHNOLOGY, f in FUEL, y in YEAR}: sum{l in TIMESLICE} RateOfUseByTechnology[r,l,t,f,y]*YearSplit[l,y] = UseByTechnologyAnnual[r,t,f,y];
s.t. E1_AnnualEmissionProductionByMode{r in REGION, t in TECHNOLOGY, e in EMISSION, m in MODE_OF_OPERATION, y in YEAR}: EmissionActivityRatio[r,t,e,m,y]*TotalAnnualTechnologyActivityByMode[r,t,m,y]=AnnualTechnologyEmissionByMode[r,t,e,m,y];
s.t. E2_AnnualEmissionProduction{r in REGION, t in TECHNOLOGY, e in EMISSION, y in YEAR}: sum{m in MODE_OF_OPERATION} AnnualTechnologyEmissionByMode[r,t,e,m,y] = AnnualTechnologyEmission[r,t,e,y];
s.t. E3_EmissionsPenaltyByTechAndEmission{r in REGION, t in TECHNOLOGY, e in EMISSION, y in YEAR}: AnnualTechnologyEmission[r,t,e,y]*EmissionsPenalty[r,e,y] = AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y];
s.t. E4_EmissionsPenaltyByTechnology{r in REGION, t in TECHNOLOGY, y in YEAR}: sum{e in EMISSION} AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y] = AnnualTechnologyEmissionsPenalty[r,t,y];
s.t. E5_DiscountedEmissionsPenaltyByTechnology{r in REGION, t in TECHNOLOGY, y in YEAR}: AnnualTechnologyEmissionsPenalty[r,t,y]/((1+DiscountRate[r])^(y-min{yy in YEAR} min(yy)+0.5)) = DiscountedTechnologyEmissionsPenalty[r,t,y];
s.t. E6_EmissionsAccounting1{r in REGION, e in EMISSION, y in YEAR}:
    sum{t in TECHNOLOGY} AnnualTechnologyEmission[r,t,e,y] =
    AnnualEmissions[r,e,y];
s.t. E7_EmissionsAccounting2{r in REGION, e in EMISSION}: sum{y in YEAR
    } AnnualEmissions[r,e,y] = ModelPeriodEmissions[r,e]-
    ModelPeriodExogenousEmission[r,e];
s.t. E8_AnnualEmissionsLimit{r in REGION, e in EMISSION, y in YEAR}:
    AnnualEmissions[r,e,y]+AnnualExogenousEmission[r,e,y] <=
    AnnualEmissionLimit[r,e,y];
s.t. E9_ModelPeriodEmissionsLimit{r in REGION, e in EMISSION}:
    ModelPeriodEmissions[r,e] <= ModelPeriodEmissionLimit[r,e];
solve;
end;
```

# 3 Project Log Book

Disclaimer: Contributions to the Project Log Book grew inconsistent toward the later stages of the project.

## January - February

- Began scoping energy related project during experience in the Commercial team at ExxonMobil Australia
- Emailed and Meet with Rosalind
- Decided to look at Carbon Pricing Initiatives to inform reinvestment and carbon pricing initiatives
- Rosalind tasked with with investigating GAMS

## March 1st - May 30th

- Coronavirus was classified a worldwide pandemic
- New Zealand was sent into lockdown
- Researched 30+ Academic reports, articles, websites for Literature Review
- Wrote 10 page Literature Review
- Scoped the project
- Submitted Mid-Semester Literature Review on May 5th
- Installed GAMS on my local device
- Began researching the construction of an energy system with Excel, VEDA FE, GAMS, VEDA BE, Python
- Created GOCPI Geographies.gyp script to combined cities, countries and continents while providing granularity to the modelling process
- Created GOCPI.html as a project display for selling the project
- Ran into a series of installation and usage issues with VEDA and GAMS
- Requested VM to work from home

- Installed VMware and GAMS on FlexIT systems
- Faced GAMS Licensing issues on FlexIT

### May 31st 2020

1. Installed Microsoft Remote Desktop and FortiClient VPN to access UoA Virtual Machine
2. Set up Virtual Machine

### June 1st 2020

1. Installed VEDA FE and VEDA FE on Virtual Machine
2. Downloaded 12 Demo Models to build my TIMES Model

### June 3rd

1. Begun testing the Model the Demo Models

### June 4th - June 10th

1. Meeting with Rosalind. Discussed set up and action points moving forward.
2. Showed VEDA-FE. Four assessments were discussed.
3. Continued researching how to use VEDA

### June 11th - Approximately 4 hours

1. Meeting with Rosalind at 10:30am via Zoom
2. Discussed action points moving forward.
3. Continued to adapt excel spreadsheets for Excel Data.
4. There is still an issue with GAMS Installation (Check with Tony. He knows a guy)
5. VEDA FE creates the necessary DD files. Continue to work through the DEMO Models to understand GAMS.

### June 16th - July 1st

- No Progress - Study Break and Exams for ACCTG 371, FINANCE 362 and ENGSCI 711

### July 2nd

- Last meeting in Rosalind's corner office. Discussed online exams, Chegg, cheating and project next steps.
- Agreed to adapt spreadsheets for user input and use BP's World Energy Outlook Statistics to determine production, conversion and consumption rates.

## July 3rd

- Began adapting Demo 12 model for custom inputs
- Began using the openpyxl python library to manipulate excel (GOCPI Input.gyp)

## July 4rd

- Continue to work on openpyxl adaptation with xls and xlsx excel sheets

## July 6th

- Created a proper file directory for managing the project
- Continued to adapt GOCPI Inputs.gyp to scale across multiple sheets
- Adapted GOCPI.html, GOCPI Inputs.gyp and GOCPI Geographies to work after rearranging the geographies
- Nearly had a heart attacked as I was led to believe issues with Github and Git meant I deleted my entire project
- Recovered entire project and reports

## July 7th

- Worked on file manipulation in Google Drive via Google Cloud APIs
- Discovered IEA Energy Balances on stats.OECD.org via Uni library databases
- Found 20GB csv on Energy Balances data
- Processed 20GB csv to create two 80MB csv for 2017 energy balance data using Microsoft Access

## July 7th

- Developed and resolved issues relating to git and Github
- Developed processing methods for Energy Balance statistics using pandas pivot table function

## July 17th

- Meeting with Kiti (NZ TIMES Energy Modeler)
- Discuss constraints associated TIMES and GAMS modelling
- Introduced to OseMOSYS (Open Source, Energy Modelling Tool)
- Introduced to MBIE,EECA (https://www.eeca.govt.nz/)
- Agreed to explore OseMOSYS and alternative datasources to build an alternative product.
- Agreed to keep Kiti updated on projec process moving forward.

## July 18th

- Downloaded MBIE Energy
- Research OseMOSYS energy modelling Approach
- Downloaded OseMOSYS energy modelling tools
- Tested Pyomo, GNU and GAMS approaches. GNU optimised using glpsol in conda environment. Progress works well.
- Decision: Move away from TIMES/GAMS modelling to using Osemosys.
- Began Scripting Sheet to generate model input text file

## July 19th

- Created excel spreadsheet to store OseMOSYS energy model inputs
- Began adapting sets, parameters, variables, equations and constraints to excel template.
- Researched more about OseMOSYS

## July 20th

- Continued to adapt 200+ lines of model code in the excel templates

## July 21st

- Learned to create custom python packages.
- Began working on adjustable sets

## July 22nd

-

## July 23rd

- Productive meeting with Rosalind, showed model output. (Rosalind said progress was really exciting)

## July 24th

- Continued creating a custom package for the GCOPI module.

## July 25th

- Started GOCPI module to create scalable data files

## July 26th

- Continued to adapt GOCPI custom package to create scalable data files (Completed)

## July 27th

- Edited report headings and created a structure for the Research Report.

## July 28th

- Investigated CPLEX Solvers
- Registered for the IBM Academic Initiative
- Downloaded and Installed IBM ILOG CPLEX Optimizer Studio
- Installed cplex and docplex Python APIs from the IBM ILOG CPLEX Optimizer Studio
- Added create model file model to GOCPI

## July 30th - August 9th

- Spent a day fixing git commit and push issues
- Installed GIT LFS and the functionality of .gitignore to prevent the committing .mp4 and .lp files
- Installed yapf in Microsoft Visual Studio Code to enable PEP-8 Autoformatting
- Wrote 4.5 pages for the technical, mainly focusing on the setup of Python, Anaconda, CPLEX, Git, GitHub, folder structure suggested by Wilson et al and the OseMOSYS methodology.
- Submitted the 4-6 page technical report.
- Created presentation structure

## August 10th

- Drafted and submitted four slide summary for presentation.
- Recorded and submitted 5 minute presentation

## August 12th

- Lockdown and Became Ill
- Went and got COVID-19 Testing (Stood in Queue for 4.5 hours)

### August 13th

- Very productive meeting with Rosalind
- Discussed project process, presentation and mid-year technical report
- Continuing doing what I am doing.
- Continued developing NZ Example
- Abandoned developing the NZ Example as faced severe limitations
- Continued developing the Navigation, Forecasting, Energysystems and CreateCases modules.

### September 2nd - September 30th

- IBM Cloud Installation and Application.
- Discussed project process, presentation and mid-year technical report
- Investigated adopting DOCPLEX optimisation technologies.
- Discovered limitations in the IBM Decision Optimisation service. This was no longer viable as imported to IBM Watson Machine Learning service.
- Began exploring the implementation of the IBM Watson Machine Learning service to engage with this pipeline.
- Developed the optimisation module to use

### October 1st - October 29th

- Systems week interfered with the construction of the report.
- Wrote the report
- Edited the report
- Reviewed the report
- Had three productive meetings with my supervisor about the report.

### October 30th

- Submitted the final report

## 4 Bibliography