energies

MDPI

Supplementary Materials: Source Code

LAEND: A Model for the Multi-Objective Investment Optimization of Residential Quarters Considering Costs and ILCD Midpoint Indicators

Ingela Tietze *, Lukas Lazar, Heidi Hottenroth and Steffen Lewerenz

Institute for Industrial Ecology (INEC), Pforzheim University, Tiefenbronner Str. 65, D-75175 Pforzheim, Germany; lukas.lazar@hs-pforzheim.de (L.L.); heidi.hottenroth@hs-pforzheim.de (H.H.); steffen.lewerenz@hs-pforzheim.de (S.L.)

* Correspondence: ingela.tietze@hs-pforzheim.de; Tel.: +49-7231-28-6361

Received: 20 December 2019; Accepted: 23 January 2020; Published: 1 February 2020

LAEND source code

```
'LAEND'
##############################################################################
# Imports, logging, global data frames
##############################################################################
import sys
import olca
import pickle
import logging
import pandas as pd
import pprint as pp
import numpy as np
import seaborn as sns
import oemof_visio as oev
import oemof.solph as solph
import oemof.outputlib as outputlib
from mpl_toolkits.mplot3d import Axes3D

from pathlib import Path
from oemof.network import Node
from oemof.tools import logger
from oemof.tools import helpers
from scipy.stats import pearsonr
from oemof.tools import economics
from matplotlib import pyplot as plt
from oemof.outputlib import processing, views
from oemof.solph import (EnergySystem, Bus, Source, Sink, Flow,
                         Model, Investment, components)

# feedinlib
import feedinlib.models as models
import feedinlib.weather as weather
import feedinlib.powerplants as plants
from windpowerlib import basicmodel as windmodel
```

```
# demandlib
import demandlib.bdew as bdew
import datetime

# Logging
import warnings
warnings.simplefilter(action="ignore", category=RuntimeWarning)
logging.getLogger().setLevel(logging.INFO)

# Global data frames
system_arch = pd.DataFrame()
system_impacts = pd.DataFrame(index= [
        'System costs',
        'Climate change, biogenic',
        'Climate change, fossil',
        'Climate change, land use',
        'Climate change, total',
        'Acidification Potential',
        'Ecotoxicity',
        'Eutrophication, freshwater',
        'Eutrophication, marine',
        'Eutrophication, terrestrial',
        'Human toxicity, carcinogenic',
        'Ionising radiation',
        'Human toxicity, non-carcinogenic',
        'Ozone depletion potential',
        'Photochemical Ozone creation',
        'Respiratory Effects',
        'Water, dissipated',
        'Resources, fossil',
        'Land use',
        'Minerals and Metals',
        'JRCII',
        'EnvCosts',
        'Equilibrium'
        ])

##########################################################################
# System settings
##########################################################################
year_hours ={
            2020:       8784,
            2021:       8760,
            2022:       8760,
            2023:       8760,
            2024:       8784,
            2025:       8760,
            2026:       8760,
            2027:       8760,
```

```
        2028:       8784,
        2029:       8760,
        2030:       8760,
        2031:       8760,
        2032:       8784,
        2033:       8760,
        2034:       8760,
        2035:       8760,
        2036:       8784,
        2037:       8760,
        2038:       8760,
        2039:       8760,
        2040:       8784,
        2041:  8760,
        2042:  8760,
        2043:  8760,
        2044:  8784,
        2045:  8760,
        2046:  8760,
        2047:  8760,
        2048:  8784,
        2049:  8760,
        2050:  8760
            }

# Start and end year of optimization
start_year=2020
end_year=2050

# Scenarios, objective function
scenario = [
        'Costs',
        'Climate change, biogenic',
        'Climate change, fossil',
        'Climate change, land use',
        'Climate change, total',
        'Acidification Potential',
        'Ecotoxicity',
        'Eutrophication, freshwater',
        'Eutrophication, marine',
        'Eutrophication, terrestrial',
        'Human toxicity, carcinogenic',
        'Ionising radiation',
        'Human toxicity, non-carcinogenic',
        'Ozone depletion potential',
        'Photochemical Ozone creation',
        'Respiratory Effects',
        'Water, dissipated',
        'Resources, fossil',
        'Land use',
```

```
        'Minerals and Metals',
        'JRCII',
        'EnvCosts',
        'Equilibrium'
        ]

# Sensitivity analysis
sensitivity_analysis = True

# Weight of costs
g_c_envcosts = 0 # Condition 0.5 for EnvCosts is set in oemofrun


##############################################################################
# ESM Data Input: Demand, Supply, Costs, Impacts, Invest decision
##############################################################################
print('# READING DEMAND, PV SUPPLY AND INVESTMENT TIMES')

# ----------------------Imports Demand, Supply, Supporting----------------
# Electricity demand, 74 households
def demandappend(start_year, end_year):
    data_d = pd.read_csv(Path(
                    'in/Profiles_electricity_germany.csv'))
    cd = {}
    for i in range (start_year, end_year+1):
        if year_hours[i] == 8760:
            print(i, ' Regular year appended')
            cd[i] = data_d.iloc[0:8760]
            #data_d.append(data_d.iloc[0:8760])
        else:
            print(i, ' Leapyear appended')
            cd[i] = data_d.iloc[0:8784]
            #data_d.append(data_d.iloc[0:8784])
    #pd.DataFrame(cd)
    data_d = pd.concat(cd)

    #data_d = data_d.drop(data_d.columns[0], axis=1)
    #data_d = data_d.drop(data_d.columns[[0,1]], axis='columns')
    data_d.to_csv('in/Profiles_electricity_germany_20_50.csv')
    return data_d

demandappend(2020, 2050)

data_d = pd.read_csv(Path(
                    'in/Profiles_electricity_germany_20_50.csv'))
data_d = data_d.loc[:,'p1':]
print(data_d.sum(axis=1))

# PV, location: Hochschule Pforzheim
timeseries_pv_hspf = pd.read_csv(Path(
```

```
                                'in/timeseries_pv_hspf.csv'))

# Investment hours for years
def hours_calc (start_year, end_year):
    hours_invest=0
    for i in range (start_year, end_year+1):
        hours_invest += year_hours[i]
    return hours_invest

def invest (start_year, end_year, lifetime):
    if (start_year, lifetime) in invest_dict:
        print(f'Investment times for {start_year}, {lifetime} have already been calculated, times from
dictionary are taken instead')
        return invest_dict[start_year, lifetime]
    else:
        print(f'Calculating investment times for {start_year}, {lifetime}')
        invest_np = []
        for i in range (0, hours_calc(2020, start_year-1)):
            invest_np.append(0)
        if (start_year-1+lifetime) <= 2050:
            power_hours = hours_calc(start_year, (start_year-1+lifetime))
        else:
            power_hours = hours_calc(start_year, 2050)
        for i in range (0, power_hours):
            invest_np.append(1)
        for i in range (0, hours_calc(start_year-1+lifetime+1, 2050)):
            invest_np.append(0)
        invest_dict[start_year, lifetime] = invest_np
        return invest_np

invest_dict = {}

# ---------------------------------Wind supply----------------------------
# Specification of the weather data set, height conversion in [m]
#description of names:
#name_dc = {
#    'your diffuse horizontal radiation': 'dhi',
#    'your direct horizontal radiation': 'dirhi',
#    'your pressure data set': 'pressure',
#    'your ambient temperature': 'temp_air',
#    'your wind speed': 'v_wind',
#    'your roughness length': 'z0'}

print('# CALCULATING WIND ENERGY')

heightofdata = {
    'dhi': 0,
    'dirhi': 0,
    'pressure': 244,
    'temp_air': 377,
```

```
    'v_wind': 377,
    'Z0': 0}

# Specifications of the wind turbines in [m]
nordexN50 = {
    'h_hub': 70,
    'd_rotor': 50,
    'wind_conv_type': 'NORDEX N50 800'}

def ready_example_data(filename, datetime_column='Unnamed: 0'):
    df = pd.read_csv(filename)
    return df.set_index(pd.to_datetime(df[datetime_column])).tz_localize(
        'UTC').tz_convert('Europe/Berlin').drop(datetime_column, 1)

#Passing all data to the weather class
#https://cdc.dwd.de/portal/201809260905/index.html
weather_df = ready_example_data(Path('in/weather_pf.csv'))
my_weather = weather.FeedinWeather(
    data=weather_df,
    timezone='Europe/Berlin',
    latitude=49,
    longitude=8,
    data_height=heightofdata)

# Initialise different power plants
# So far there is only one model available. So you do not have to pass a
# model (s. E126). If model is passed the default model is used.
# We hope that there will be different models in future versions. You can
# also write your own model an pass it to the powerplant.

N50_power_plant = plants.WindPowerPlant(model=models.SimpleWindTurbine,
                                        **nordexN50)

# Create a feedin series for a specific powerplant under specific weather
# conditions. One can define the number of turbines or the overall capacity.
# If no multiplier is set, the time series will be for one turbine.
N50_feedin = N50_power_plant.feedin(weather=my_weather, number=1)
N50_feedin=N50_feedin /1000 #1W --> kW
N50_feedin.name = 'N50'

# Show wind turbine feedin plot
print('')
print('********* Wind turbine *********')
if plt:
    print('Load profile of the wind turbine')
    N50_feedin.plot(legend=True, drawstyle='steps-post', figsize=(10,5))
    plt.show()
else:
    print('No wind turbine')
```

```
# Write turbine feedin to CSV
N50_lp=N50_feedin.to_frame(name='wind')
N50_lp.reset_index()
# Conversion of wind turbine feedin to profile per 1 kW
N50_lp['wind'] = N50_lp['wind']/800
N50_lp.to_csv(Path('out/Nordex_N50_load_profile.csv'))
N50_lp = pd.read_csv(Path('out/Nordex_N50_load_profile.csv'))

for i in range (start_year, end_year+1):
    invest_years = invest(i, end_year, 20)
    print(len(invest_years))
    N50_lp[f'wind_{i}']=N50_lp['wind']*invest_years

print(N50_lp)

# Wind model
n50 = windmodel.SimpleWindTurbine(**nordexN50)

# cp curve
if plt:
    print('cp curve of the wind turbine')
    n50.cp_values.plot(style='*')
    plt.show()
else:
    # The value for 8 m/s
    print(n50.cp_values.loc[8])

logging.info('Done with wind turbine calculation!')

# ---------------------------------PV supply-----------------------------
print('# PV DATA')
print('********* PV *********')
print('Load profile PV')

timeseries_pv_hspf['EPV']=timeseries_pv_hspf['EPV']/1000 # [W] --> [kW]

# Plot timeseries for PV at Hochschule Pforzheim
timeseries_pv_hspf['EPV'].plot(legend=True, drawstyle='steps-post',
                                        figsize=(10,5))
plt.show()

# Adjust investment times to feedin data
lt_PV=30
for i in range (start_year, end_year+1):
    invest_years = invest(i, end_year, lt_PV)
    timeseries_pv_hspf[f'PV_{i}']=timeseries_pv_hspf['EPV'].mul(
            invest_years)

writer = pd.ExcelWriter(Path(
      f'out/PV_timeseries.xlsx'))
```

```
timeseries_pv_hspf.to_excel(writer,'PV_timeseries')
writer.save()


#############################################################################
# Life Cycle Assessment
#############################################################################

# oLCA calculation
def LCIA(ps):
    client = olca.Client(8080)
    setup = olca.CalculationSetup()
    setup.calculation_type = olca.CalculationType.SIMPLE_CALCULATION
    setup.impact_method = client.find(olca.ImpactMethod,
                                      'ILCD 2.0 2018 midpoint')

    setup.product_system = client.find(olca.ProductSystem, ps)
    setup.amount = 1.0

# calculate the results and export it to an Excel file
    result = client.calculate(setup)
    client.excel_export(result, Path(f'LCA/{ps}.xlsx'))
    client.dispose(result)
    print(ps, ' Done')

# ---------------------------- Call openLCA -------------------------------
# Call product system of function in LCA module
def LCIAcall ():
    print('LCA calculation enabled,connecting to openLCA by client')
    LCIA('biogas, burned in micro gas turbine 100kWe, 1kWh')
    LCIA('electricity, high voltage, production mix, 1kWh')
    LCIA('electricity photovoltaic, 3kWp slanted-roof installation, '
         'multi-Si, 1kWh')
    LCIA('electricity photovoltaic, 3kWp slanted-roof installation, '
         'single-Si, 1kWh')
    LCIA('electricity production, wind, sml 1MW, onshore, 1kWh')
    LCIA('battery, Li-ion LFP-LTO, 37.9Wh')
    LCIA('battery, VRF, 19.4Wh')
    LCIA('market for battery, Li-ion, rechargeable, prismatic, 0.114kWh')
    LCIA('market for battery, NaCl, 0.116kWh')
    LCIA('market for battery, NiMH, rechargeable, prismatic, 0,08kWh')
    LCIA('market for methane, 96% by volume, from biogas, low pressure, '
         'at user, 1MJ')
    LCIA('market for natural gas, low pressure, 1m3')
    LCIA('micro gas turbine production, 100kW electrical, storage, 10000
         l')
    LCIA('natural gas, burned in micro gas turbine, 100kWe, 1kWh')
    LCIA('photovoltaic, 3kWp slanted-roof installation, multi-Si, 1kWp')
    LCIA('photovoltaic, 3kWp slanted-roof installation, single-Si, 1kWp')
    LCIA('wind Nordex N50, 1kW')
```

LCIAcall ()

```
###########################################################################
# Scenarios
###########################################################################
logging.info('# START SCENARIOS')

def oemofrun (start_year, end_year, scenario, g_c_envcosts):
    # *************************************************************************
    # ********** PART 1 - Define and optimise the energy system
*****************************************************************************

    # Solver
    solver = 'cplex'
    debug = False
    solver_verbose = True  # show/hide solver output
    solver_options_on = False
    solver_options = {
    } if solver_options_on == True else {}

    # Timesteps
    hours=0
    for i in range (start_year, end_year+1):
        hours += year_hours[i]
    print(f'Optimisation hours:{hours}')
    number_of_time_steps = hours

    # initiate the logger (see the API docs for more information)
    logger.define_logging(logfile='oemof.log',
                          screen_level=logging.INFO,
                          file_level=logging.DEBUG)
    logging.info('Initialize the energy system')
    date_time_index = pd.date_range('1/1/%s'%start_year,
                                    periods=number_of_time_steps,
                                    freq='H')
    energysystem = EnergySystem(timeindex=date_time_index)
    Node.registry = energysystem


        # ----------------------------LCIA data feedin, normalised---------
    oLCA={}

    # CHP
    oLCA['CHPg','op'] = pd.read_excel(Path( 'LCA/natural gas, burned in
        micro gas turbine, 100kWe, 1kWh.xlsx'),
        sheet_name='Impacts')
    oLCA['CHPg','inv'] = pd.read_excel(Path('LCA/micro gas turbine
        production, 100kW electrical, storage, 10000 l.xlsx'),
        sheet_name='Impacts')
```

```
oLCA['CHPg','inv']['Result'] =oLCA['CHPg','inv']['Result']/100/20
   # 100kW turbine, 20 years lifetime
oLCA['CHPb','op'] = pd.read_excel(Path(
   'LCA/biogas, burned in micro gas turbine 100kWe, 1kWh.xlsx'),
   sheet_name='Impacts')
oLCA['CHPb','inv'] = pd.read_excel(Path(
   'LCA/micro gas turbine production, 100kW electrical, storage, 10000
    l.xlsx'),
   sheet_name='Impacts')
oLCA['CHPb','inv']['Result'] = oLCA['CHPb','inv']['Result']/100/20
   # 100kW turbine, 20 years lifetime

# Gas
oLCA['gas','op'] = pd.read_excel(Path(
   'LCA/market for natural gas, low pressure, 1m3.xlsx'),
   sheet_name='Impacts')
oLCA['gas','op']['Result'] = oLCA['gas','op']['Result']/39*3.6
   #39 MJ/m3 ecoinvent inquiry --> kWh

oLCA['biogas','op'] = pd.read_excel(Path(
   'LCA/market for methane, 96% by volume, from biogas, low pressure,
    at user, 1MJ.xlsx'),
   sheet_name='Impacts')
oLCA['biogas','op']['Result'] = oLCA['biogas','op']['Result']*3.6
   # 1 MJ -->kWh

# Grid
oLCA['grid','op'] = pd.read_excel(Path(
   'LCA/electricity, high voltage, production mix, 1kWh.xlsx'),
   sheet_name='Impacts')
# PV
oLCA['PVm','op'] = pd.read_excel(Path(
   'LCA/electricity photovoltaic, 3kWp slanted-roof installation,
    multi-Si, 1kWh.xlsx'),
   sheet_name='Impacts')
oLCA['PVm','inv'] = pd.read_excel(Path(
   'LCA/photovoltaic, 3kWp slanted-roof installation, multi-Si,
    1kWp.xlsx'),
   sheet_name='Impacts')
oLCA['PVm','inv']['Result'] = oLCA['PVm','inv']['Result']/30
    # 30 years lifetime

# Wind
oLCA['wind','op'] = pd.read_excel(Path(
   'LCA/electricity production, wind, sml 1MW, onshore, 1kWh.xlsx'),
   sheet_name='Impacts')
oLCA['wind','inv'] = pd.read_excel(Path(
   'LCA/wind Nordex N50, 1kW.xlsx'), sheet_name='Impacts')
oLCA['wind','inv']['Result'] = oLCA['wind','inv']['Result']/20
    # 20 years lifetime
```

```
# Battery storage
oLCA['LFP','inv'] = pd.read_excel(Path(
    'LCA/battery, Li-ion LFP-LTO, 37.9Wh.xlsx'),
    sheet_name='Impacts')
oLCA['LFP','inv']['Result'] =
            oLCA['LFP','inv']['Result']*0.9/(37.9)*1000/15
   #37.9 Wh effektiv, n=0.9, 8000 Cycles, lifetime = 15 y (assumption)
oLCA['VRF','inv'] = pd.read_excel(Path(
    'LCA/battery, VRF, 19.4Wh.xlsx'),
    sheet_name='Impacts')
oLCA['VRF','inv']['Result'] =
    oLCA['VRF','inv']['Result']*0.75/(19.4)*1000/15
   #19.4 Wh eff.,n=0,75, 10000 Cycles, lifetime = 15 y (assumption)
print(oLCA)
LCIA_index=oLCA['CHPg','inv']['Impact category']
print(LCIA_index)

LCIA_ES={}
for key in oLCA.keys():
    LCIA_ES[key] = oLCA[key]['Result']

LCIA_ES = pd.DataFrame(LCIA_ES, index=LCIA_index)


#LCIA_ES = pd.DataFrame.from_dict(oLCA)
LCIA_ES.to_excel(Path('LCA/LCIA_ES.xlsx'))

logging.info('# LCA Impact Assessment was calculated and fed into the
ESM.')

#########################################################################
# Weighting and Normalisation Procedure of Costs and Environmental
  Impacts ##############################################################################
logging.info('# WEIGHTING AND NORMALISATON.')

# Weighting factors
g_c_envcosts = 0.5 if scenario == 'EnvCosts' else g_c_envcosts
n_c_envcosts = 75093.38e9 # Global GDP 2018, IMF
g_env_envcosts = 1-g_c_envcosts
if scenario == 'Costs':
    g_c=1          # Costs (economical)
    g_env=0
else:
    g_c = 0
    g_env=1
if scenario in {'EnvCosts', 'Equilibrium'}:
    g_c = g_c_envcosts
    n_c = n_c_envcosts
    g_c = g_c/n_c
```

```
    g_env = g_env_envcosts

# Read Weighting and normalisation factors
G_in = pd.read_excel(Path(
    'in/Normalisation and Weighting.xlsx'), sheet_name='Weighting')
N_in = pd.read_excel(Path(
    'in/Normalisation and Weighting.xlsx'), sheet_name='Normalisation')
if scenario != 'Costs':
    G = np.array(G_in[scenario])
    G = G_in[scenario]
    print('********* Weighting Factors *********')
    print(G)
    print('')
    # Normalisation factors
    N = np.array(N_in[scenario])
    N = N_in[scenario]
    print('********* Normalisation Factors *********')
    print(N)
    print('')
else:
    print('Costs optimisation, no weighting and normalisation
    required')
    G = 0
    N = 0
    print(f'Weighting factor: {G}')
    print(f'Normalisation factor: {N}')


#------------------------ Costs and Environmental Impacts-------------

variable_costs={}
ep_costs={}
variable_env={}
ep_env={}
cost_decrease=0.01 # Annual cost decrease (technical progress)
wacc=0.01 # weighted average cost of capital

# Call environmental impacts according to process and cost type
# Variable
def varenv(p):
    if np.all(G==0):
        variable_env[p]=0
    else:
        variable_env[p] = G*oLCA[p,'op'].iloc[:,4].values/N
        condition = (variable_env[p]!=0)
        variable_env[p] = np.extract(condition, variable_env[p])
        variable_env[p] = np.sum(variable_env[p])

# Investment, equivalent periodical
def epenv(p):
```

```
    if np.all(G==0):
        ep_env[f'{p}']=0
    else:
        ep_env[f'{p}'] = G*oLCA[f'{p}','inv'].iloc[:,4].values/N
        condition = (ep_env[f'{p}']!=0)
        ep_env[f'{p}'] = np.extract(condition, ep_env[f'{p}'])
        ep_env[f'{p}'] = np.sum(ep_env[f'{p}'])

# Define costs and environmental impacts per kWh or kW
# Natural Gas
aexchp = 0.73 # allocation factor (see also ecoinvent methodology)
variable_costs['gas']=0.06*aexchp
varenv('gas')

# Biogas
variable_costs['biogas']=0.10*aexchp
varenv('biogas')

# CHPg
lt_CHP=20
variable_costs['CHPg']=0.01*aexchp
ep_costs['CHPg']=economics.annuity(capex=(760*(1+0.06+0.45)*aexchp),
n=lt_CHP, wacc=wacc)
varenv('CHPg')
epenv('CHPg')

# CHPb
variable_costs['CHPb']=0.02*aexchp
ep_costs['CHPb']=economics.annuity(capex=(764*(1+0.06+0.45)*aexchp),
n=lt_CHP,
                                   wacc=wacc)
varenv('CHPb')
epenv('CHPb')

# Electricity mix from grid, Germany
variable_costs['grid'] = 0.14 #[EUR/kWh]
varenv('grid')
variable_env['grid']=variable_env['grid']/0.98
    # Efficiency high/low voltage (according to ecoinvent)

#PVm
lt_PV=30
variable_costs['PVm'] = 0 #[EUR/kWh]
ep_costs['PVm'] = (1300*0.01+economics.annuity(capex=1300, n=lt_PV,
                   wacc=wacc))
varenv('PVm')
epenv('PVm')
r_PV=1 # impact reduction
ep_costs['PVm'] *=r_PV
ep_env['PVm'] *= r_PV
```

```
# Wind Nordex N50
lt_wind=20
variable_costs['wind'] = 0.
ep_costs['wind'] = (59+economics.annuity(capex=1558, n=lt_wind,
                                         wacc=wacc)) # per kW
varenv('wind')
epenv('wind')

# Storage - Lithium-ion LFP
reduce = 1 # for sensitivity analysis < 1
lt_storage=15
variable_costs['LFP'] = 0
ep_costs['LFP'] =
reduce*(25+economics.annuity(capex=374+500+125+50+900,
                             n=lt_storage, wacc=wacc))
variable_env['LFP']=0
epenv('LFP')

# Storage - VRF
variable_costs['VRF'] = 0 # Baumann, 2017, 2018, 10000 Cycles
ep_costs['VRF'] =
reduce*(40+economics.annuity(capex=374+500+125+50+458,
                             n=lt_storage, wacc=wacc))
variable_env['VRF']=0
epenv('VRF')


# ----------Investment costs and environment in future----------
end_year = end_year + 1
ep_costs_suml = {}
ep_env_suml = {}

c_weight=1
if scenario in {'Human toxicity, carcinogenic',
          'Human toxicity, non-carcinogenic',
          'Ozone depletion potential',
          'Respiratory Effects'}:
   c_weight=1e6
if scenario == 'Minerals and Metals':
   c_weight=1e3
if scenario in {'JRCII', 'EnvCosts', 'Equilibrium'}:
   c_weight=1e10


print('Corrected weight for solver: ', c_weight)

def ep_costs_sum (lt, i, p):
   ltr = (end_year - i) if (end_year - i)<lt else lt
```

```
        ep_costs_sum = ep_costs[f'{p}']/(1+cost_decrease)**(i-
        start_year)*ltr
        ep_costs_suml[f'{p}_{i}'] = ep_costs_sum
        ep_costs_sum *= g_c
        print('')
        print(f'Investment costs, {p}_{i}: {ep_costs_sum}')
        return ep_costs_sum*c_weight

    def ep_env_sum (lt, i, p):
        ltr = (end_year - i) if (end_year - i)<lt else lt
        ep_env_sum = ep_env[f'{p}']*ltr
        ep_env_suml[f'{p}_{i}'] = ep_env_sum
        ep_env_sum *= g_env
        print(f'Investment environment, {p}_{i}: {ep_env_sum}')
        try:
            oLCA[p, 'inv']['Result', i] =oLCA[p, 'inv']['Result']*ltr
        except KeyError:
            print( p, 'is not available')
        return ep_env_sum*c_weight

    # Investment step
    investment_step = 5
    print('Investment step: ', investment_step)

    # Variable costs and env weighting
    variable_costs_sum = pd.Series(variable_costs)*g_c*c_weight
    variable_env_sum = pd.Series(variable_env)*g_env*c_weight
    print('Cost: ', variable_costs_sum)
    print('Env: ', variable_env_sum)
    print('')

    ##########################################################################
    # Create oemof objects
###########################################################################

    logging.info('Create oemof objects')

    #Busses
    bel = Bus(label='electricity')
    bgas = Bus(label='gas')
    bbgas = Bus(label='biogas')
    bth = Bus(label='heat')

    # Demand
    Sink(label='houses',
        inputs={
            bel: Flow(
                actual_value=data_d.sum(axis=1),
                fixed=True,
                nominal_value=1),
```

```
            })

    # Overproduction sink
    Sink(label='excess_bel',
        inputs={
            bel: Flow(
                variable_costs=0,
                variable_env=0)})

    # Overproduction sink heat
    Sink(label='excess_bth',
        inputs={
            bth: Flow()})

    #Resource gas/bio gas transformers
    print(i)
    Source(label=f'gas_{start_year}', outputs={bgas: solph.Flow(
                            max = invest(i, end_year, lt_CHP),

                           variable_costs=variable_costs_sum['gas'],
                           variable_env=variable_env_sum['gas'],
                            )})

    Source(label=f'biogas_{start_year}', outputs={bbgas: solph.Flow(
                        max = invest(i, end_year, lt_CHP),

                                                variable_costs=variable_costs_sum['biogas'],
                        variable_env=variable_env_sum['biogas']
                        )})


# ---------------------------------Electricity-----------------------------
    # Electricity mix Germany
    Source(label=f'grid_{start_year}',
      outputs={
            bel: Flow(
                variable_costs= variable_costs_sum['grid'],
                variable_env = variable_env_sum['grid'],
                )})


    # PV
    for i in range (start_year, end_year, investment_step):
       Source(label=f'PVm_{i}',
          outputs={bel: Flow(
                actual_value=timeseries_pv_hspf[f'PV_{i}'],
                fixed=True,
                variable_costs=variable_costs_sum['PVm'],
                variable_env=variable_env_sum['PVm'],
                investment=Investment(ep_costs=
```

```
                        ep_costs_sum(lt_PV, i, 'PVm'),
                        ep_env=
                        ep_env_sum(lt_PV, i,'PVm'))))})

# Wind
for i in range (start_year, end_year, investment_step):
    Source(label=f'wind_{i}',
        outputs={bel: Flow(
                actual_value=N50_lp[f'wind_{i}'],
                #nominal_value=1,
                fixed=True,
                variable_costs=variable_costs_sum['wind'],
                variable_env=variable_env_sum['wind'],
                #min=0.1,
                investment=Investment(ep_costs=
                        ep_costs_sum(lt_wind, i,
                                'wind'),
                        ep_env=
                        ep_env_sum(lt_wind, i,
                                'wind'))))})

# CHPgas
for i in range (start_year, end_year, investment_step):
    solph.Transformer(label=f'CHPg_{i}',
        inputs={bgas: Flow()},
        outputs={bel: Flow(
                max =  invest(i, end_year, lt_CHP),
                                        #invest(i, end_year, lt_CHP),
                variable_costs=variable_costs_sum['CHPg'],
                #invest_years_variable[f'invest_{i}'],
                variable_env = variable_env_sum['CHPg'],
                #invest_years_variable[f'invest_{i}'],
                investment=Investment(
                        ep_costs=
                        ep_costs_sum(lt_CHP, i,
                        'CHPg'),
                        ep_env=
                        ep_env_sum(lt_CHP, i,
                        'CHPg'))),
            bth: Flow()},
        conversion_factors={bel: 0.37, bth: 0.49})

# CHPbiogas
for i in range (start_year, end_year, investment_step):
    solph.Transformer(label=f'CHPb_{i}',
        inputs={bbgas: Flow()},
        outputs={bel: Flow(
                max = invest(i, end_year, lt_CHP),
                variable_costs=variable_costs_sum['CHPb'],
                variable_env = variable_env_sum['CHPb'],
```

```
                investment=Investment(ep_costs=
                                ep_costs_sum(lt_CHP, i,
                                'CHPb'),
                                ep_env=
                                ep_env_sum(lt_CHP, i,
                                'CHPb'))),
            bth: Flow()},
        conversion_factors={bel: 0.39, bth: 0.49}) # ASUE Kenndaten 2014, 2011 (thermischer n)


# Battery storage - LFP
  for i in range (start_year, end_year, investment_step):
    components.GenericStorage(label=f'LFP_{i}',
      inputs={bel: Flow(
              max = invest(i, end_year, lt_storage),
              summed_max=8000
              )},
      outputs={bel: Flow(
            max = invest(i, end_year, lt_storage),
            summed_max=8000, # Cycles
            variable_costs=variable_costs_sum['LFP'],
            variable_env=variable_env_sum['LFP']
            )},
      initial_capacity=0.6,
      balanced = True,
      loss_rate = 1-0.9999,
        # Wang 2016: Characteristic Analysis of Lithium Titanate
          Battery
      invest_relation_input_capacity=1,
                    # c-rate, degr. dep., Stroe 2018
      invest_relation_output_capacity=1,
                    # c-rate, degr. dep., Stroe 2018
      inflow_conversion_factor=1,
      outflow_conversion_factor=0.9,
      investment=Investment(ep_costs=
                    ep_costs_sum(lt_CHP, i, 'LFP'),
                    ep_env=
                    ep_env_sum(lt_CHP, i, 'LFP')
                    ))

  print(ep_env_sum)

 # Battery storage - VRF
  for i in range (start_year, end_year, investment_step):
    components.GenericStorage(label=f'VRF_{i}',
      inputs={bel: Flow(
              max = invest(i, end_year, lt_storage),
              summed_max=10000
      outputs={bel: Flow(
            max = invest(i, end_year, lt_storage),
```

```
                summed_max = 10000, # Cycles
                variable_costs=variable_costs_sum['VRF'],
                #invest_years_variable_10[f'invest_{i}'],
                variable_env=variable_env_sum['VRF']
                #invest_years_variable_10[f'invest_{i}']
                )},
            initial_capacity=0.6,
            loss_rate = 1-0.9308,
            balanced = True,
            invest_relation_input_capacity=1,
            invest_relation_output_capacity=1,
            inflow_conversion_factor=1,
            outflow_conversion_factor=0.75,
            investment=Investment(ep_costs=
                            ep_costs_sum(lt_storage, i, 'VRF'),
                            ep_env=
                            ep_env_sum(lt_storage, i, 'VRF')))

    ##########################################################################
    # Optimise the energy system and debug
##########################################################################

    #Optimisation
    logging.info('Optimise the energy system')

    # initialise the operational model
    model = Model(energysystem)

    # This is for debugging only. It is not(!) necessary to solve the
      problem and
    # should be set to False to save time and disc space in normal use. For
    # debugging the timesteps should be set to 3, to increase the readability of
    # the lp-file.
    if debug:
        filename = Path(
            helpers.extend_basic_path('lp_files'), 'basic_example.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        model.write(filename, io_options={'symbolic_solver_labels': True})

    # if tee_switch is true solver messages will be displayed
    logging.info('Solve the optimization problem')
    model.solve(solver=solver, solve_kwargs={'tee': solver_verbose,},
                cmdline_options=solver_options)
    logging.info('Store the energy system with the results.')

    # The processing module of the outputlib can be used to extract the
      results
    # from the model transfer them into a homogeneous structured
      dictionary.
    # add results to the energy system to make it possible to store them.
```

```
energysystem.results['main'] = outputlib.processing.results(model)
energysystem.results['meta'] = outputlib.processing.meta_results(model)

# The default path is the '.oemof' folder in your $HOME directory.
# The default filename is 'es_dump.oemof'.
# You can omit the attributes (as None is the default value) for
  testing
# cases. You should use unique names/folders for valuable results to
  avoid
# overwriting.

# store energy system with results
energysystem.dump(dpath=None, filename=None)

# store costs and impacts
ep_costs_suml_pd = pd.Series(ep_costs_suml)
ep_env_suml_pd = pd.Series(ep_env_suml)
writer = pd.ExcelWriter(Path(
      f'out/{scenario}_Costs_and_Impacts_for_calculation.xlsx'))
variable_costs_sum.to_excel(writer,'Variable_costs')
variable_env_sum.to_excel(writer,'Variable_env')
ep_costs_suml_pd.to_excel(writer, 'EP_costs')
ep_env_suml_pd.to_excel(writer, 'EP_env')
writer.save()

##########################################################################
# Results  ###############################################################################

# print the solver results
print('********* Meta results *********')
pp.pprint(energysystem.results['meta'])
print('')
c_weight_pd = pd.Series(c_weight)
energysystem_results_meta_pd =
pd.DataFrame(energysystem.results['meta'])
energysystem_meta=pd.concat([energysystem_results_meta_pd,
c_weight_pd],
keys=['objective result', 'corrected weight'])

print('')
print('********* Results *********')
logging.info('Restore the energy system and the results.')
energysystem = solph.EnergySystem()
energysystem.restore(dpath=None, filename=None)

# define an alias for shorter calls below (optional)
results = energysystem.results['main']
#storage = energysystem.groups['storage']

# Getting results and views
```

```
results = processing.results(model)
   #custom_storage = views.node(results, 'storage')
electricity_bus = views.node(results, 'electricity')
heat_bus = views.node(results, 'heat')
gas_bus = views.node(results, 'gas')
biogas_bus = views.node(results, 'biogas')

# Print the sums of the flows around the electricity bus
print('********* Main results, electricity *********')
print(electricity_bus['sequences'].sum(axis=0))
print(electricity_bus['scalars'])
electricity_bus['scalars'].sum(axis=0))

 # Print the sums of the flows around the heat bus
print('********* Main results, heat *********')
print(heat_bus['sequences'].sum(axis=0))

# Create aggregated results data frame
electricity_bus_agg = pd.DataFrame()
electricity_bus_agg['demand'] = electricity_bus['sequences'].filter(
                    like='house', axis=1).sum(axis=1)
electricity_bus_agg['wind'] = electricity_bus['sequences'].filter(
                    like='wind', axis=1).sum(axis=1)
electricity_bus_agg['PV'] = electricity_bus['sequences'].filter(
                    like='PV', axis=1).sum(axis=1)
electricity_bus_agg['CHPg'] = electricity_bus['sequences'].filter(
                    like='CHPg', axis=1).sum(axis=1)
electricity_bus_agg['CHPb'] = electricity_bus['sequences'].filter(
                    like='CHPb', axis=1).sum(axis=1)
electricity_bus_agg['grid'] = electricity_bus['sequences'].filter(
                    like='grid', axis=1).sum(axis=1)
electricity_bus_agg['storage'] = (electricity_bus['sequences'].filter(
                    like='LFP', axis=1).sum(axis=1))+(

electricity_bus['sequences'].filter(
                      like='VRF', axis=1).sum(axis=1))

print('')
print('********* Results, aggregated *********')
print(electricity_bus_agg.sum(axis=0))

# Write bus results to Excel files
writer = pd.ExcelWriter(Path(
      f'out/{scenario}_results.xlsx'))
energysystem_meta.to_excel(writer,'meta')
electricity_bus['sequences'].to_excel(writer,'el_sequences')
electricity_bus['scalars'].to_excel(writer, 'el_scalars')
electricity_bus_agg.to_excel(writer, 'el_aggregated')
heat_bus['sequences'].to_excel(writer,'h_sequences')
gas_bus['sequences'].to_excel(writer,'g_sequences')
```

```
biogas_bus['sequences'].to_excel(writer,'b_sequences')
writer.save()


# --------Build total costs and total environmental intervention-------
print('********* Total impacts and costs *********')
inv_costs = {}
inv_env = {}
var_costs = {}
var_env = {}
esystems = {'electricity' : ['grid', 'CHPg', 'CHPb', 'PVm', 'wind',
            'LFP', 'VRF'],
           'heat' : [],
           'gas' : ['gas'],
           'biogas' : ['biogas']}

# Investment Costs (scalars)
for i in esystems['electricity']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            inv_costs[f'{i}_{j}', 'electricity'] = (
                electricity_bus['scalars'][(
                    (f'{i}_{j}', 'electricity'),
                    'invest')]*ep_costs_suml[f'{i}_{j}'])
    except KeyError:
        print(f'In investment costs {i}_{j} has been excepted')
        continue

for i in esystems['heat']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            inv_costs[f'{i}_{j}', 'heat'] = (heat_bus['scalars']
                [((f'{i}_{j}', 'heat'),
                'invest')]*ep_costs_suml[f'{i}_{j}'])
    except KeyError:
        print(f'In investment costs {i}_{j} has been excepted')
        continue
inv_costs = pd.Series(inv_costs)
print('********* Investment costs *********')
print(inv_costs)

# Investment Environment (scalars)
for i in esystems['electricity']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            inv_env[f'{i}_{j}', 'electricity'] =
            (electricity_bus['scalars']
                [((f'{i}_{j}', 'electricity'), 'invest')]*oLCA[i,
                 'inv']['Result', j])
    except KeyError:
```

```
        print(f'In investment environment {i}_{j}_electricity has been
              excepted')
        continue

for i in esystems['heat']:
    try:
        inv_env[i, 'heat'] =heat_bus['scalars'].filter(
              like=i).sum(axis=0) * oLCA[i, 'inv']['Result', j]
    except KeyError:
        print(f'In investment environment {i}_{j}_heat has been
             excepted')
        continue

inv_env = pd.DataFrame(inv_env)
print('********* Investment Environment *********')
print(inv_env)

# Variable Costs (sequences)

for i in esystems['electricity']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_costs[f'{i}_{j}', 'electricity'] =
            electricity_bus['sequences'][
                  ((f'{i}_{j}', 'electricity'),
                   'flow')].sum(axis=0)*variable_costs[i]
    except KeyError:
        print(f'In variable costs {i}_{j}_electricity has been
             excepted')
        continue

for i in esystems['heat']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_costs[f'{i}_{j}', 'heat'] = heat_bus['sequences'][
                  ((f'{i}_{j}', 'heat'),
                   'flow')].sum(axis=0)*variable_costs[i]
    except KeyError:
        print(f'In variable costs {i}_{j}_heat has been excepted')
        continue

for i in esystems['gas']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_costs[f'{i}_{j}', 'gas'] = gas_bus['sequences'][
                  ((f'{i}_{j}', 'gas'),
                   'flow')].sum(axis=0)*variable_costs[i]
    except KeyError:
        print(f'In variable costs {i}_{j}_gas has been excepted')
        continue
```

```
for i in esystems['biogas']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_costs[f'{i}_{j}', 'biogas'] = biogas_bus['sequences'][
                ((f'{i}_{j}', 'biogas'),
                 'flow')].sum(axis=0)*variable_costs[i]
    except KeyError:
        print(f'In variable costs {i}_{j}_biogas has been excepted')
        continue

var_costs = pd.Series(var_costs)
print('********* Variable costs *********')
print(var_costs)

# Variable Environment (sequences)
for i in esystems['electricity']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_env[f'{i}_{j}', 'electricity'] =
            electricity_bus['sequences'][
                ((f'{i}_{j}', 'electricity'),
                'flow')].sum(axis=0)*oLCA[i, 'op']['Result']
    except KeyError:
        print(f'In variable environment {i}_{j}_electricity has been
            excepted')
        continue

for i in esystems['heat']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_env[f'{i}_{j}', 'heat'] = heat_bus['sequences'][
                ((f'{i}_{j}', 'heat'), 'flow')].sum(axis=0)*oLCA[i,
                 'op']['Result']
    except KeyError:
        print(f'In variable environment {i}_{j}_heat has been
            excepted')
        continue

for i in esystems['gas']:
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_env[f'{i}_{j}', 'gas'] = gas_bus['sequences'][
                ((f'{i}_{j}', 'gas'), 'flow')].sum(axis=0)*oLCA[i,
                  'op']['Result']
    except KeyError:
        print(f'In variable environment {i}_{j}_gas has been excepted')
        continue

for i in esystems['biogas']:
```

```
    try:
        for j in range (start_year, end_year+1, investment_step):
            var_env[f'{i}_{j}', 'biogas'] = biogas_bus['sequences'][
                ((f'{i}_{j}', 'biogas'),
                 'flow')].sum(axis=0)*oLCA[i, 'op']['Result']
    except KeyError:
        print(f'In variable environment {i}_{j}_biogas has been
              excepted')
        continue

var_env = pd.DataFrame(var_env)
print('********* Variable Environment *********')
print(var_env)

# Build sums and multi criteria scenarios
cenv_costs = pd.Series()
cenv_env = pd.DataFrame(oLCA['PVm', 'op']['Impact category'])
cenv_costs['Investment costs'] = inv_costs.sum()
cenv_costs['Variable_costs'] = var_costs.sum()
cenv_costs['System_costs'] = inv_costs.sum() + var_costs.sum()
cenv_env['Investment_environmental_impacts'] = inv_env.sum(axis=1)
cenv_env['Variable_environmental_impacts'] = var_env.sum(axis=1)
cenv_env['System_environmental_impacts'] = inv_env.sum(axis=1) +
var_env.sum(axis=1)
cenv_costs['JRCII'] =
cenv_env['System_environmental_impacts']/N_in['JRCII']*G_in['JRCII']
cenv_costs['JRCII_sum'] = cenv_costs['JRCII'].sum()
cenv_costs['EnvCosts'] =
cenv_env['System_environmental_impacts']/N_in['EnvCosts']*
G_in['EnvCosts']
cenv_costs['EnvCosts_sum'] =
cenv_costs['EnvCosts'].sum()*g_env_envcosts +
cenv_costs['System_costs']/n_c_envcosts*g_c_envcosts
cenv_costs['Equilibrium'] =
cenv_env['System_environmental_impacts']/N_in['Equilibrium']*
G_in['Equilibrium']
cenv_costs['Equilibrium_sum'] =
cenv_costs['Equilibrium'].sum()*g_env_envcosts +
cenv_costs['System_costs']/n_c_envcosts*g_c_envcosts

system_impacts[scenario] = 0
system_impacts.loc[['System costs'], [scenario]] =
cenv_costs['System_costs']
print(system_impacts)

        system_impacts[scenario].iloc[1:20]=cenv_env[
            'System_environmental_impacts'].values
system_impacts.loc[['JRCII'], [scenario]] = cenv_costs['JRCII_sum']
system_impacts.loc[['EnvCosts'], [scenario]] =
cenv_costs['EnvCosts_sum']
```

```
system_impacts.loc[['Equilibrium'], [scenario]] =
cenv_costs['Equilibrium_sum']

pickle.dump(system_impacts[scenario],
open(Path(f'temp/{scenario}_impacts.p'), 'wb'))
print(system_impacts)

# --------Write Costs and impacts to Excel-----------

# equivalent periodical costs
ep_costs_out_pd = pd.DataFrame.from_dict(
      ep_costs_suml, orient='index')
ep_env_out_pd = pd.DataFrame.from_dict(
      ep_env_suml, orient='index')
writer = pd.ExcelWriter(Path(
      f'out/{scenario}_ep_costs_env.xlsx'))
ep_costs_out_pd.to_excel(writer,'ep_costs')
ep_env_out_pd.to_excel(writer,'ep_env')
writer.save()

# Total Costs and Impacts
writer = pd.ExcelWriter(Path(
      f'out/{scenario}_Costs_and_Impacts.xlsx'))
inv_costs.to_excel(writer,'Investment_costs')
inv_env.to_excel(writer,'Investment_environment')
var_costs.to_excel(writer,'Variable_costs')
var_env.to_excel(writer,'Variable_environment')
cenv_costs.to_excel(writer,'Sum_Costs_and_mc')
cenv_env.to_excel(writer,'Sum_Environment')
writer.save()


# Prepare All Results
system_arch[scenario] = electricity_bus_agg.sum(axis=0)
pickle.dump(system_arch[scenario],
open(Path(f'temp/{scenario}_arch.p'), 'wb'))

# --------Plots-----------

print('')
print('')
print('********* Plots *********')

# Electricity bus
ax = electricity_bus['sequences'].plot(kind='line', drawstyle='steps-
post',
                                #color=colors,
                                 figsize=(10, 5))
ax.set_title('Electricity mix')
plt.savefig(Path
```

```
        (f'out/png/{scenario}_electricity_timeline'))
plt.show()

# Input flows
in_cols = oev.plot.divide_bus_columns(
   'electricity', electricity_bus['sequences'].columns)['in_cols']
ax = electricity_bus['sequences'][in_cols].plot(kind='line',
                                    drawstyle='steps-post',
                                    figsize=(10, 5))
ax.set_title('Input flows')
plt.savefig(Path(
      f'out/png/{scenario}_electricity_input_timeline'))
plt.show()


# Heat bus
ax = heat_bus['sequences'].plot(kind='line', drawstyle='steps-post',
                                #color=colors,
                                 figsize=(10, 5))
ax.set_title('Heat mix')
plt.savefig(Path(
      f'out/png/{scenario}_heat_timeline'))
plt.show()

# Gas bus
ax = gas_bus['sequences'].plot(kind='line', drawstyle='steps-post',
                                #color=colors,
                                 figsize=(10, 5))
ax.set_title('Gas mix')
plt.savefig(Path(f'out/png/{scenario}_gas_timeline'))
plt.show()

# Bgas bus
ax = biogas_bus['sequences'].plot(kind='line', drawstyle='steps-post',
                                #color=colors,
                                 figsize=(20,5))
ax.set_title('Biogas mix')
plt.savefig(Path(f'out/png/{scenario}_biogas_timeline'))
plt.show()

# Results aggregated
ax = electricity_bus_agg[['wind', 'PV', 'CHPg', 'CHPb', 'grid']].plot(
      kind='line', figsize=(10, 5))
ax.set_ylabel('Electricity in kW')
ax.set_title('Electricity mix, aggregated')
plt.savefig(Path(f'out/png/{scenario}_electricity_total_timeline'))
plt.show()

# Demand
ax = electricity_bus_agg[['demand']].plot(kind='line', figsize=(10, 5))
```

```
    ax.set_ylabel('Electricity in kW')
    ax.set_title('Demand')
    plt.savefig(Path(f'out/png/{scenario}_demand_electricity'))
    plt.show()

    # Bar chart, totals
    ax = (electricity_bus_agg.sum(axis=0)/1000).plot.barh(stacked=True)
    ax.set_title('Total electricity demand and generation')
    ax.set_xlabel('Power in MWh')
    plt.savefig(Path(f'out/png/{scenario}_electricity_total_bar'))
    plt.show()
    print('')

    # Pie Chart with storage
    ax = electricity_bus_agg[['wind', 'PV', 'CHPg', 'CHPb', 'grid',
    'storage']].sum(
            axis=0).plot.pie(figsize=(6, 6))
    ax.set_title('Shares of electricity generation with storage')
    ax.set_ylabel('Share')
    plt.savefig(Path(f'out/png/{scenario}_electricity_total_w_stor_pie'))
    plt.show()
    print('')

    # Pie Chart without storage
    ax = electricity_bus_agg[['wind', 'PV', 'CHPg', 'CHPb', 'grid']].sum(
            axis=0).plot.pie(figsize=(6, 6))
    ax.set_title('Shares of electricity generation w/o storage')
    ax.set_ylabel('Share')
    plt.savefig(Path(f'out/png/{scenario}_electricity_total_wo_stor_pie'))
    plt.show()

    print('********* Optimisation ended *********')

scenario_len = len(scenario)
for i in range (0, scenario_len):
    oemofrun(start_year, end_year, scenario[i], g_c_envcosts)

def result_handling(sens):
    scenario = [
                'Costs',
                'Climate change, biogenic',
                'Climate change, fossil',
                'Climate change, land use',
                'Climate change, total',
                'Acidification Potential',
                'Ecotoxicity',
                'Eutrophication, freshwater',
                'Eutrophication, marine',
                'Eutrophication, terrestrial',
                'Human toxicity, carcinogenic',
```

```
            'Ionising radiation',
            'Human toxicity, non-carcinogenic',
            'Ozone depletion potential',
            'Photochemical Ozone creation',
            'Respiratory Effects',
            'Water, dissipated',
            'Resources, fossil',
            'Land use',
            'Minerals and Metals',
            'JRCII',
            'EnvCosts',
            'Equilibrium'
            ]

scenario_len = len(scenario)
# Read in Electricity_bus cache
for i in range (0, scenario_len):
    f = open(Path(f'temp/{scenario[i]}_arch.p'), 'rb')
    system_arch[scenario[i]] = pickle.load(f)
    f.close()
    f = open(Path(f'temp/{scenario[i]}_impacts.p'), 'rb')
    system_impacts[scenario[i]] = pickle.load(f)
    f.close()

print(system_arch)
print(system_impacts)

# Plot system results
    #With storage

ax = system_arch.iloc[1:].T.plot(
        kind='barh', stacked=True)
ax.set_title('Electricity generation, all scenarios')
ax.set_xlabel('Power in MWh')
plt.savefig(Path('out/png/All_scenarios_electricity.svg'),
                format = 'svg')
plt.show()

    # Without storage
ax = system_arch.iloc[1:5].T.plot(
        kind='barh', stacked=True)
ax.set_title('Electricity generation, all scenarios')
ax.set_xlabel('Power in MWh')
plt.savefig(Path('out/png/All_scenarios_electricity_wo_storage.svg'),
                format = 'svg')
plt.show()


# Correlation
corr_p = pd.DataFrame()
```

```
    corr_k = pd.DataFrame()
    corr_s = pd.DataFrame()
    corr_p = system_impacts.corr(method = 'pearson')
    corr_k = system_impacts.corr(method = 'kendall')
    corr_s = system_impacts.corr(method = 'spearman')

    # Arrange system plots
    arch = system_arch
    impc = system_impacts
    archimpc = pd.concat([arch, impc])
    impc.to_excel('IMPC.xlsx')
    archimpc = archimpc.sort_values(by='System costs', axis=1)
    archimpc.to_excel('ARCHIMPC.xlsx')
    systems = ['wind', 'PV', 'CHPg', 'CHPb', 'grid', 'storage']

    sum_cc = pd.Series()
    sum_cc = 0
    for i in systems:
        sum_cc += archimpc.loc[i]

    cc = []
    cc_d = {}
    for i in systems:
        share = archimpc.loc[i]/1000        cc_d[i] = share
    cc_df = pd.DataFrame.from_dict(cc_d)


    # Arrange system expansion table
    scalars = {}

    scenario_len = len(scenario)
    for i in range (0, scenario_len):
        print(scenario[i])
        scalars[scenario[i]] = pd.read_excel(Path(f'out/{scenario[i]}_results.xlsx'),
sheet_name='el_scalars', squeeze=True)
    print(scalars)
    investt = {}
    esystems = ['CHPb', 'CHPg', 'LFP', 'PVm', 'VRF', 'wind']
    investment_step = 5
    for i in range (0, scenario_len):
        investt[scenario[i]]={}
        for j in range (start_year, end_year+1, investment_step):
            investt[scenario [i]][j] = {}
            for k in esystems:
                try:
                    investt[scenario[i]][j][k] =
                    scalars[scenario[i]].at[f'"(('{k}_{j}',
                    'electricity'), 'invest')"']
                except KeyError:
                    print(k, j, ' not available')
```

```
print(investt)

system_expan = pd.DataFrame.from_dict({(i,j): investt[i][j]
                    for i in investt.keys()
                    for j in investt[i].keys()
                    #for k in investt[i][j].keys()
                },
                orient='index')


writer = pd.ExcelWriter(Path(
    f'System_expan.xlsx'))
system_expan.to_excel(writer, 'System_expan.xlsx')
writer.save()

#write Climate change in tons, Costs in Mill. EUR for following Excel
 plots
cc_df['Climate change, total'] = archimpc.loc['Climate change,
total']/1000
cc_df['System costs'] = archimpc.loc['System costs']/1e6

# Write System architecture, impacts, correlation and p-values to Excel
writer = pd.ExcelWriter(Path(
    f'out/All_Scenarios_{sens}.xlsx'))
system_arch.to_excel(writer, 'Architecture')
system_expan.to_excel(writer, 'Expansion')
system_impacts.to_excel(writer, 'Impacts')
corr_p.to_excel(writer, 'Pearson')
corr_k.to_excel(writer, 'Kendall')
corr_s.to_excel(writer, 'Spearman')
cc_df.to_excel(writer, sheet_name='Plot')
writer.save()


# Climate change, cost, storage Plots
print('Climate change, costs')
cc_df.iloc[:,0:6].plot(kind='bar', stacked=True, figsize = (20,5),
logy=True)

plt.savefig(Path('out/png/System_CC_Cost.svg'),
            format = 'svg')
plt.show()

print('Costs')
archimpc.loc['System costs'].plot(kind='line', figsize =(20,5), color =
'red')

plt.savefig(Path('out/png/System_costs.svg'),
            format = 'svg')
plt.show()
```

```
    print('Climate change, total')
    archimpc.loc['Climate change, total'].plot(kind='line', figsize
    =(20,5), color = 'blue')

    plt.savefig(Path('out/png/System_Climate_Change.svg'),
                format = 'svg')
    plt.show()

    print('storage')
    archimpc.loc['storage'].plot(kind='line', figsize =(20,5), color =
    'green')
    plt.savefig(Path('out/png/System_storage.svg'),
                format = 'svg')
    plt.show()

result_handling(0.5)

# Sensitivity Analysis
g_c_envcosts_lst = [0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9]
if sensitivity_analysis == True:
    for j in g_c_envcosts_lst:
        oemofrun(start_year, end_year, 'EnvCosts', j)
        result_handling(j)

print('')
print('********* THE END *********')

# Logfile end
if logfile == True:
    sys.stdout.close()
    sys.stdout=stdoutOrigin
```

Oemof source code modifications

Original code by oemof (https://oemof.org/), see also: Hilpert, S.; Kaldemeyer, C.; Krien, U.; Günther, S.; Wingenbach, C.; Plessmann, G. The Open Energy Modelling Framework (oemof)-A new approach to facilitate open science in energy system modelling. Energy Strategy Reviews 2018, 22, 16–25.

In solph blocks.py, class Flow(SimpleBlock):

```
def _objective_expression(self):
        r""" Objective expression for all standard flows with fixed costs
        and variable costs.
        """
        m = self.parent_block()
```

```
    variable_costs = 0
    gradient_costs = 0

    for i, o in m.FLOWS:
        if m.flows[i, o].variable_costs[0] is not None:
            for t in m.TIMESTEPS:
                variable_costs += (m.flow[i, o, t] * m.objective_weighting[t] *
                                   (m.flows[i, o].variable_costs[t] +
                                    m.flows[i, o].variable_env[t]))

        if m.flows[i, o].positive_gradient['ub'][0] is not None:
            for t in m.TIMESTEPS:
                gradient_costs += (self.positive_gradient[i, o, t] *
                                   m.flows[i, o].positive_gradient[
                                       'costs'])

        if m.flows[i, o].negative_gradient['ub'][0] is not None:
            for t in m.TIMESTEPS:
                gradient_costs += (self.negative_gradient[i, o, t] *
                                   m.flows[i, o].negative_gradient[
                                       'costs'])

    return variable_costs + gradient_costs
```

In solph blocks.py, class InvestmentFlow(SimpleBlock):

```
def _objective_expression(self):
    r""" Objective expression for flows with investment attribute of type
    class:`.Investment`. The returned costs are fixed, variable and
    investment costs.
    """
    if not hasattr(self, 'FLOWS'):
        return 0

    m = self.parent_block()
    investment_costs = 0

    for i, o in self.FLOWS:
        if m.flows[i, o].investment.ep_costs is not None:
            investment_costs += (self.invest[i, o] *
                                 (m.flows[i, o].investment.ep_costs+
                                  m.flows[i, o].investment.ep_env))
        else:
            raise ValueError("Missing value for investment costs!")

    self.investment_costs = Expression(expr=investment_costs)
    return investment_costs
```

In solph components.py, class GenericInvestmentStorageBlock(SimpleBlock):

```
def _objective_expression(self):
    """Objective expression with fixed and investement costs."""
    if not hasattr(self, 'INVESTSTORAGES'):
        return 0

    investment_costs = 0

    for n in self.INVESTSTORAGES:
        if n.investment.ep_costs is not None:
            investment_costs += self.invest[n] * n.investment.ep_costs
        if n.investment.ep_env is not None:
            investment_costs += self.invest[n] * n.investment.ep_env
        else:
            raise ValueError("Missing value for investment costs!")

    self.investment_costs = Expression(expr=investment_costs)

    return investment_costs
```

In solph network.py, within class Flow:

```
def __init__(self, **kwargs):
    # TODO: Check if we can inherit from pyomo.core.base.var _VarData
    # then we need to create the var object with
    # pyomo.core.base.IndexedVarWithDomain before any Flow is created.
    # E.g. create the variable in the energy system and populate with
    # information afterwards when creating objects.

    scalars = ['nominal_value', 'summed_max', 'summed_min',
               'investment', 'nonconvex', 'integer', 'fixed']
    sequences = ['actual_value', 'variable_costs', 'variable_env',
                 'min', 'max']
    dictionaries = ['positive_gradient', 'negative_gradient']
    defaults = {'fixed': False, 'min': 0, 'max': 1, 'variable_costs': 0,
                'variable_env':0,
                'positive_gradient': {'ub': None, 'costs': 0},
                'negative_gradient': {'ub': None, 'costs': 0},
                }
```

In solph options.py, class Investment:

```
def __init__(self, maximum=float('+inf'), minimum=0, ep_costs=0,
             existing=0, ep_env=0):
    self.maximum = maximum
    self.minimum = minimum
    self.ep_costs = ep_costs
    self.existing = existing
```

```
        self.ep_env = ep_env
```