This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No.

REFEREE: Real Value of Energy Efficiency

Functioning technology diffusion models and multiple benefit indicator processes

Project Acronym	REFEREE
Project Title	Real Value of Energy Efficiency
Grant Agreement No	101000136
Project Start Date	1 October 2020
Project End Date	31 March 2024
Call Identifier	LC-SC3-EC-4-2020
Funding Scheme	Research and Innovation Action (RIA)
Project Website	http://refereetool.eu/

Deliverable Information

Deliverable No	D 3.4
Deliverable Title	Functioning technology diffusion models and multiple benefit indicator processes (Python code)
Work Package No	3
Work Package Lead	CE
Contributing Partners	
Deliverable Type	Report
Dissemination Level	
Author(s)	Jamie Pirie, Rosie Hayward, Ornella Dellacio, Pim Vercoulen, Isha Dwesar, Sachin Gulati
Contributors	Iakov Frizis, Jon Stenning
Contractual Deadline	30 September 2021
Delivery Date	19 October 2022

Version Management

Version	Date	Author	Description of Change
1.0	18/10/2022	Jamie Pirie, Rosie Hayward, Ornella Dellacio, Pim Vercoulen, Isha Dwesar, Sachin Gulati	First complete version for submission

Partners

Partner	**Short name**	**Principal Investigator**
Cambridge Econometrics	CE	Jon Stenning

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No.

Introduction and summary

This note presents the functioning technology diffusion models and multiple benefit indicator process. It includes the modelling code, programmed in Python, which i) assesses the technology diffusions and ii) calculates the multiple benefits of energy efficiency.

For accessibility purposes and in line with the grant agreement this deliverable is submitted in pdf format. Thus, the code is not executable. To receive a working version of specific sections of the code developed for the purposes of the REFEREE project, please contact Iakov Frizis (IF@camecon.com).

In line with deliverable 3.1 (D3.1), the FTT model framework has been applied to a number of areas across the energy system to model how various sectors will decarbonise. In the course of the REFEREE project, four models have been updated or created: Household heating (FTT-Heat) – *updated,* *P*assenger road transport (FTT-Transport) – *updated,* Road freight transport (FTT-Freight) – *newly created,* Industrial heating processes (FTT-Industrial Heat) – *newly created.* It is the code describing these FTT models that is included in the pages below.

```
# -*- coding: utf-8 -*-
"""
=========================================
ftt_fr_main.py
=========================================
Freight transport FTT module.
#############################

This is the main file for FTT: Freight, which models technological
diffusion of freight vehicle types due to simulated consumer decision making.
Consumers compare the **levelised cost of freight**, which leads to changes in the
market shares of different technologies.

The outputs of this module include market shares, fuel use, and emissions.

Local library imports:

    FTT: Freight functions:

    - `get_lcof <ftt_fr_lcof.html>`__
        Levelised cost calculation

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros
    - `estimation <econometrics_functions.html>`__
        Predict future values according to the estimated coefficients.

Functions included:
    - solve
        Main solution function for the module
    - get_lcof
```

```
        Calculate levelised cost of freight

"""

from math import sqrt
import os
import copy
import sys
import warnings

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

from ftt_fr_lcof import get_lcof

#Main function

def solve(data, time_lag, iter_lag, titles, histend, year, domain):
    """
    Main solution function for the module.
    This function simulates investor decision making in the freight sector.
    Levelised costs (from the lcof function) are taken and market shares
    for each vehicle type are simulated to ensure demand is met.


    Parameters
    -----------
    data: dictionary of NumPy arrays
```

```
            Model variables for the given year of solution
        time_lag: type
            Description
        iter_lag: type
            Description
        titles: dictionary of lists
            Dictionary containing all title classification
        histend: dict of integers
            Final year of histrorical data by variable
        year: int
            Curernt/active year of solution
        Domain: dictionary of lists
            Pairs variables to domains

        Returns
        ----------
        data: dictionary of NumPy arrays
            Model variables for the given year of solution

        """

        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it

        #T_Scal = 5      # Time scaling factor used in the share dynamics

        c6ti = {category: index for index, category in enumerate(titles['C6TI'])}

        sector='freight'
        #Creating variables

        zjet=copy.deepcopy(data['ZJET'][0, :, :])
```

```
    emis_corr = np.zeros([len(titles['RTI']), len(titles['FTTI'])])

    if year <= histend["RVKZ"]:

        #U (ZEWG) is number of vehicles by technology
        data['ZEWG'][:,:,0] = data['ZEWS'][:,:,0]*data['RFLZ'][:, np.newaxis, 0,0]

#
#
#        #I (ZEWY) is new sales, positive changes in U
#        data['ZEWY'][r,:,0] = ((data['ZEWG'][r,:,0] - data['ZEWG'][r,:,0])/dt)*((data['ZEWG'][r,:,0] - data['ZEWG'][r,:,0])>0)


#
##        for veh in range(len(titles['FTTI'])):
##            for fuel in range(len(titles['JTI'])):
##                if titles['JTI'][fuel] == '5 Middle distillates' and data['ZJET'][0, veh, fuel] ==1:  # Middle distillates
##
##                            # Mix with biofuels if there's a biofuel mandate
##                    zjet[veh, fuel] = zjet[veh, fuel] * (1.0 - data['ZBFM'][r, 0, 0])
##
##                            # Emission correction factor
##                    emis_corr[r, veh] = 1.0 - data['ZBFM'][r, 0, 0]
##
##                elif titles['JTI'][fuel] == '11 Biofuels'  and data['ZJET'][0, veh, fuel] == 1:
##
##                    zjet[veh, fuel] = data['ZJET'][0, veh, fuel] * data['ZBFM'][r, 0, 0]
#
#        data['ZJNJ'][r, :, 0] = (np.matmul(np.transpose(zjet), data['ZEVV'][r, :, 0]*\
#                                        data['ZCET'][r, :, c6ti['9 energy use (MJ/vkm)']]))/0.041868
#
```

```
#        #Emissions, E is ZEWE
#        data['ZEWE'][r,:,0]=data['ZEVV'][r,:,0]*data['ZCET'][r,:,c6ti['14 CO2Emissions (gCO2/km)']]*(1-data['ZBFM'][r,0,0])/(1**6)
#
#
#        #Set cumulative capacities variable
#        data['ZEWW'][0,:,0]=data['ZCET'][0,:,c6ti['11 Cumulative seats']]


    "Model Dynamics"

    #Endogenous calculation starts here
    if year > histend['RVKZ']:

        data_dt = {}
        data_dt['ZWIY'] = np.zeros([len(titles['RTI']), len(titles['VTTI']), 1])

        for var in time_lag.keys():

            if var.startswith("R"):

                data_dt[var] = copy.deepcopy(time_lag[var])

        for var in time_lag.keys():

            if var.startswith("Z"):

                data_dt[var] = copy.deepcopy(time_lag[var])

        #find if there is a regulation and if it is exceeded

        isReg =  np.zeros([len(titles['RTI']), len(titles['FTTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['FTTI'])])
```

```
        division = divide((data_dt['RVKZ'][:, :, 0] - data['ZREG'][:, :, 0]),
                          data_dt['ZREG'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['ZREG'][:, :, 0] == 0.0] = 1.0
        isReg[data['ZREG'][:, :, 0] == -1.0] = 0.0


        for t in range(1, no_it+1):
    #Interpolations to avoid staircase profile

            RTCO = time_lag['RZCO'][:, :, :] + (data['RZCO'][:, :, :] - time_lag['RZCO'][:, :, :]) * t * dt
            FuT = time_lag['RTFZ0'][:, :, :] + (data['RTFZ0'][:, :, :] - time_lag['RTFZ0'][:, :, :]) * t * dt
            #TJET = time_lag['ZJET'][:, :, :] + (data['ZJET'][:, :, :] - time_lag['ZJET'][:, :, :]) * t * dt
            D = time_lag['RVKZ'][:, :, :] + (data['RVKZ'][:, :, :] - time_lag['RVKZ'][:, :, :]) * t * dt
            Utot = time_lag['RFLZ'][:, :, :] + (data['RFLZ'][:, :, :] - time_lag['RFLZ'][:, :, :]) * t * dt
            BFM = time_lag['ZBFM'][:, :, :] + (data['ZBFM'][:, :, :] - time_lag['ZBFM'][:, :, :]) * t * dt

            for r in range(len(titles['RTI'])):

                if D[r] == 0.0:
                    continue

                # DSiK contains the change in shares
                dSik = np.zeros([len(titles['FTTI']), len(titles['FTTI'])])

                # F contains the preferences
                F = np.ones([len(titles['FTTI']), len(titles['FTTI'])])*0.5

                # -----------------------------------------------------
                # Step 1: Endogenous EOL replacements
                # -----------------------------------------------------
                for b1 in range(len(titles['FTTI'])):

```

```
if  not (data_dt['ZEWS'][r, b1, 0] > 0.0 and
         data_dt['ZTLL'][r, b1, 0] != 0.0 and
         data_dt['ZTDD'][r, b1, 0] != 0.0):
    continue

S_i = data_dt['ZEWS'][r, b1, 0]

#Aki = 0.5 * data['IHEL'][r, b1, 0] / time_lag['IHUD'][r, b1, 0]

for b2 in range(b1):

    if  not (data_dt['ZEWS'][r, b2, 0] > 0.0 and
             data_dt['ZTLL'][r, b2, 0] != 0.0 and
             data_dt['ZTDD'][r, b2, 0] != 0.0):
        continue


    S_k = data_dt['ZEWS'][r, b2, 0]

    Aik = data['ZEWA'][0, b1, b2]
    Aki = data['ZEWA'][0, b2, b1]

    # Propagating width of variations in perceived costs
    dFik = sqrt(2) * sqrt((data_dt['ZTDD'][r, b1, 0]*data_dt['ZTDD'][r, b1, 0] + data_dt['ZTDD'][r, b2, 0]*data_dt['ZTDD'][r, b2, 0]))

    # Consumer preference incl. uncertainty
    Fik = 0.5*(1+np.tanh(1.25*(data_dt['ZTLL'][r, b2, 0]-data_dt['ZTLL'][r, b1, 0])/dFik))
    Fki = 1-Fik

    # Preferences are then adjusted for regulations
    F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
```

```
                        F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])


                        #Runge-Kutta market share dynamiccs
                        k_1 = S_i*S_k * (Aik*F[b1, b2] - Aki*F[b2, b1])
                        k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2] - Aki*F[b2, b1])
                        k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2] - Aki*F[b2, b1])
                        k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2] - Aki*F[b2, b1])

                        #This method currently applies RK4 to the shares, but all other components of the equation are calculated for the overall time step
                        #We must assume the the LCOE does not change significantly in a time step dt, so we can focus on the shares.

                        dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6#*data['ZCEZ'][r,0,0]
                        dSik[b2, b1] = -dSik[b1, b2]

                        # Market share dynamics
#                        dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2] - Aki*F[b2,b1])*dt#*data['ZCEZ'][r,0,0]
#                        dSik[b2, b1] = -dSik[b1, b2]

                    #Check share changes sum to zero goes here, this is under time and region loop
                    #ALso includes share equation
                dSk = np.zeros([len(titles['FTTI'])])
                data['ZEWS'][r, :, 0] = data_dt['ZEWS'][r, :, 0] + np.sum(dSik, axis=1) +dSk

                if ~np.isclose(np.sum(data['ZEWS'][r, :, 0]), 1.0, atol=1e-5):
                    msg = """Sector: {} - Region: {} - Year: {}
                    Sum of market shares do not add to 1.0 (instead: {})
                    """.format(sector, titles['RTI'][r], year, np.sum(data['ZEWS'][r, :, 0]))
                    warnings.warn(msg)

```

```
                if np.any(data['ZEWS'][r, :, 0] < 0.0):
                    msg = """Sector: {} - Region: {} - Year: {}
                    Negative market shares detected! Critical error!
                    """.format(sector, titles['RTI'][r], year)
                    warnings.warn(msg)


            for r in range(len(titles['RTI'])):
                #Copy over costs that dont change
                data['ZCET'][:, :, 1:20] = data_dt['ZCET'][:, :, 1:20]

                #G1 is Total service
                #G1[r,:,0]=D[r]/data['ZLOD'][r,0,0]

                data['ZESG'][r,:,0]=D[r,0,0]/data['ZLOD'][r,0,0]

                #Sd is share difference between small and large trucks
                data['ZESD'][r,0,0] = data['ZEWS'][r,0,0]+data['ZEWS'][r,2,0]+data['ZEWS'][r,4,0]+\
                data['ZEWS'][r,6,0]+data['ZEWS'][r,8,0]+data['ZEWS'][r,10,0]+data['ZEWS'][r,12,0]+\
                data['ZEWS'][r,14,0]+data['ZEWS'][r,16,0]+data['ZEWS'][r,18,0]

                data['ZESD'][r,1,0] = 1 - data['ZESD'][r,0,0]


                for x in range(0,20,2):
                    data['ZESA'][r,x,0]=data['ZEWS'][r,x,0]/data['ZESD'][r,0,0]
                    data['ZEVV'][r,x,0]=data['ZESG'][r,x,0]*data['ZESA'][r,x,0]/(1-1/(data['ZSLR'][r,0,0]+1))
                    data['ZEST'][r,x,0]=data['ZEVV'][r,x,0]*data['ZLOD'][r,1,0]
                for x in range(1,21,2):
                    data['ZESA'][r,x,0]=data['ZEWS'][r,x,0]/data['ZESD'][r,1,0]
                    data['ZEVV'][r,x,0]=data['ZESG'][r,x,0]*data['ZESA'][r,x,0]/(1/(data['ZSLR'][r,0,0]+1))
                    data['ZEST'][r,x,0]=data['ZEVV'][r,x,0]*data['ZLOD'][r,1,0]

```

```
#T is total service generated by small trucks in MTkm

#This is number of trucks by technology
#data['ZEWG'][r,:,0] = data['ZEWS'][r,:,0]*data['RFLZ'][r,0,0]
data['ZEWG'][r,:,0] = data['ZEWS'][r,:,0]*Utot[r,0,0]
#Investment (sales) = new capacity created

veh_diff = data['ZEWG'][r,:,0] - data_dt['ZEWG'][r,:,0]
veh_dprctn = data_dt['ZEWG'][r,:,0] / data['ZCET'][r,:,c6ti['8 service lifetime (y)']]
data['ZEWY'][r,:,0] = np.where(veh_diff>0.0,
                               veh_diff + veh_dprctn,
                               veh_dprctn)



#Emissions
data['ZEWE'][r,:,0]=data['ZEVV'][r,:,0]*data['ZCET'][r,:,c6ti['14 CO2Emissions (gCO2/km)']]*(1-data['ZBFM'][r,0,0])/(1E6)


zjet=copy.deepcopy(data['ZJET'][0, :, :])
for veh in range(len(titles['FTTI'])):
    for fuel in range(len(titles['JTI'])):
        if titles['JTI'][fuel] == '5 Middle distillates' and data['ZJET'][0, veh, fuel] ==1:  # Middle distillates

        # Mix with biofuels if there's a biofuel mandate
            zjet[veh, fuel] = zjet[veh, fuel] * (1.0 - data['ZBFM'][r, 0, 0])

        # Emission correction factor
            emis_corr[r, veh] = 1.0 - data['ZBFM'][r, 0, 0]

        elif titles['JTI'][fuel] == '11 Biofuels'  and data['ZJET'][0, veh, fuel] == 1:
```

```

                            zjet[veh, fuel] = data['ZJET'][0, veh, fuel] * data['ZBFM'][r, 0, 0]

                #Convert TJ to ktoe, therefore divide by 0.041868
                data['ZJNJ'][r, :, 0] = (np.matmul(np.transpose(zjet), data['ZEVV'][r, :, 0]*\
                                    data['ZCET'][r, :, c6ti['9 energy use (MJ/vkm)']]))/0.041868


                #Cumulative investment, not in region loop as it is global

            bi = np.zeros((len(titles['RTI']),len(titles['FTTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['ZEWB'][0, :, :],data['ZEWY'][r, :, 0])
            dw = np.sum(bi, axis=0)
            data['ZEWW'][0,:,0] = data_dt['ZEWW'][0,:,0] + dw

#                data['ZCET'][:,:,c6ti['11 Cumulative seats']]=data_dt['ZCET'][:,:,c6ti['11 Cumulative seats']]
#                +np.sum(data['ZEWB'][0,:,:]*data['ZEWY'][:,:,0],axis=1)*dt
                #reopen region loop,

                #learning curves and LCOF

            for r in range(len(titles['RTI'])):
                data['ZCET'][r,:,c6ti['1 Price of vehicles (USD/vehicle)']]=\
                    data_dt['ZCET'][r,:,c6ti['1 Price of vehicles (USD/vehicle)']]\
                    +data['ZLER'][0,:,0]*((data['ZEWW'][0,:,0]-\
                    data_dt['ZEWW'][0,:,0])/data['ZEWW'][0,:,0])*\
                    data_dt['ZCET'][r,:,c6ti['1 Price of vehicles (USD/vehicle)']]


            #Calculate total investment by technology in terms of truck purchases
            for r in range(len(titles['RTI'])):
                data['ZWIY'][r,:,0]=data_dt['ZWIY'][r,:,0] + \
```

```
                    data['ZEWY'][r, :, 0]*dt*data['ZCET'][r, :, c6ti['1 Price of vehicles (USD/vehicle)']]*1.263

                #Calculate levelised cost again
                data = get_lcof(data, titles)


                #Update time loop variables:
                for var in time_lag.keys():

                    if var.startswith("R"):

                        data_dt[var] = copy.deepcopy(time_lag[var])

                for var in time_lag.keys():

                    if var.startswith("Z"):

                        data_dt[var] = copy.deepcopy(time_lag[var])


        return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_h_main.py
=========================================
Residential heating sector FTT module.
######################################

This is the main file for FTT: Heat, which models technological
diffusion of residential heating technologies due to simulated consumer decision making.
Consumers compare the **levelised cost of heat**, which leads to changes in the
market shares of different technologies.

The outputs of this module include changes in final energy demand and boiler sales.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros
    - `estimation <econometrics_functions.html>`__
        Predict future values according to the estimated coefficients.

Functions included:
    - solve
        Main solution function for the module
    - get_lcoh
        Calculate levelised cost of residential heating

"""
# Standard library imports
from math import sqrt
```

```python
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# --------------------------- LCOH function ------------------------------------
# -----------------------------------------------------------------------------
def get_lcoh(data, titles):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of heat in 2014 Euros/kWh per
    boiler type. It includes intangible costs (gamma values) and together
    determines the investor preferences.
    """
    # Categories for the cost matrix (BHTC)
    c4ti = {category: index for index, category in enumerate(titles['C4TI'])}

    for r in range(len(titles['RTI'])):

        # Cost matrix
        bhtc = data['BHTC'][r, :, :]
```

```python

        # Boiler lifetime
        lt = bhtc[:, c4ti['5 Lifetime']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['HTTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
        lt_mat = np.where(mask, lt_mat, 0)

        # Capacity factor
        cf = bhtc[:, c4ti['13 Capacity factor mean'], np.newaxis]

        # Conversion efficiency
        ce = bhtc[:, c4ti['9 Conversion efficiency'], np.newaxis]

        # Discount rate
        # dr = bhtc[6]
        dr = bhtc[:, c4ti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['HTTI']), int(max_lt)])
        it[:, 0, np.newaxis] = divide(bhtc[:, c4ti['1 Investment cost mean'], np.newaxis], (cf*1000))

        # Standard deviation of investment cost
        dit = np.zeros([len(titles['HTTI']), int(max_lt)])
        dit[:, 0, np.newaxis] = divide(bhtc[:, c4ti['2 Investment cost SD'], np.newaxis], (cf*1000))

        # Upfront subsidy/tax at purchase time
        st = np.zeros([len(titles['HTTI']), int(max_lt)])
        st[:, 0, np.newaxis] = it[:, 0, np.newaxis] * data['HTVS'][r, :, 0, np.newaxis]

```

```
        # Average fuel costs
        ft = np.ones([len(titles['HTTI']), int(max_lt)])
        ft = ft * divide(data['HEWP'][r, :, 0, np.newaxis], ce)
        #ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['HTTI']), int(max_lt)])
        dft = ft * divide(bhtc[:, c4ti['11 Fuel cost SD'], np.newaxis], ce)
        dft = np.where(mask, dft, 0)

        # Fuel tax costs
        fft = np.ones([len(titles['HTTI']), int(max_lt)])
        fft = ft * data['HTRT'][r, :, 0, np.newaxis]
        fft = np.where(mask, fft, 0)

        # Average operation & maintenance cost
        omt = np.ones([len(titles['HTTI']), int(max_lt)])
        omt = omt * divide(bhtc[:, c4ti['3 O&M cost mean'], np.newaxis], (cf*1000))
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['HTTI']), int(max_lt)])
        domt = omt * divide(bhtc[:, c4ti['4 O&M cost SD'], np.newaxis], (cf*1000))
        domt = np.where(mask, domt, 0)

        # Feed-in-Tariffs
        fit = np.ones([len(titles['HTTI']), int(max_lt)])
        fit = fit * data['HEFI'][r, :, 0, np.newaxis]
        fit = np.where(mask, fit, 0)

        # Net present value calculations
        # Discount rate
        denominator = (1+dr)**lt_mat
```

```

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+fft+omt-fit)/denominator
        # 1.3-Only policy costs
        npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT
        lcoh = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
        # 1.2-LCOT including policy costs
        tlcoh = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        lcoh_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # Standard deviation of LCOT
        dlcoh = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

        # LCOT augmented with non-pecuniary costs
        tlcohg = tlcoh + data['HGAM'][r, :, 0]

        # Pay-back thresholds
        # TODO: Titles for FTT:Heat's cost categories are wrong
        pb = bhtc[:, c4ti['16 Empty']]
        dpb = bhtc[:, c4ti['17 Empty']]
```

```python

        # Marginal costs of existing units
        tmc = ft[:, 0] + omt[:, 0] + fft[:, 0] - fit[:, 0]
        dtmc = np.sqrt(dft[:, 0] + domt[:, 0])

        # Total pay-back costs of potential alternatives
        tpb = tmc + (it[:, 0] + st[:, 0])/pb
        dtpb = np.sqrt(dft[:, 0]**2 + domt[:, 0]**2 +
                       divide(dit[:, 0]**2, pb**2) +
                       divide(it[:, 0]**2, pb**4)*dpb**2)

        # Add gamma values
        tmc = tmc + data['HGAM'][r, :, 0]
        tpb = tpb + data['HGAM'][r, :, 0]

        # Pass to variables that are stored outside.
        data['HEWC'][r, :, 0] = lcoh           # The real bare LCOH without taxes
        data['HETC'][r, :, 0] = tlcoh          # The real bare LCOH with taxes
        data['HGC1'][r, :, 0] = tlcohg        # As seen by consumer (generalised cost)
        data['HWCD'][r, :, 0] = dlcoh         # Variation on the LCOH distribution
        data['HGC2'][r, :, 0] = tmc            # Total marginal costs
        data['HGD2'][r, :, 0] = dtmc         # SD of Total marginal costs
        data['HGC3'][r, :, 0] = tpb            # Total payback costs
        data['HGD3'][r, :, 0] = dtpb         # SD of Total payback costs

    return data

# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, specs):
    """
    Main solution function for the module.
```

```

    Add an extended description in the future.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
        Description
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution

    Notes
    ---------
    This function should be broken up into more elements in development.
    """

    # Categories for the cost matrix (BHTC)
    c4ti = {category: index for index, category in enumerate(titles['C4TI'])}
    jti = {category: index for index, category in enumerate(titles['JTI'])}
```

```python

    fuelvars = ['FR_1', 'FR_2', 'FR_3', 'FR_4', 'FR_5', 'FR_6',
                'FR_7', 'FR_8', 'FR_9', 'FR_10', 'FR_11', 'FR_12']

    sector = 'residential'
    #sector_index = titles['Sectors_short'].index(sector)



    # %% First initialise if necessary
    # Initialise in case of stock solution specification
    #if np.any(specs[sector]) < 5:

    # Up to the last year of historical useful energy demand by boiler
    if year <= histend['HEWF']:
    # Historical data ends in 2014, so we need to initialise data
    # when it's 2015 to make sure the model runs.
    # At some point we need to change the start year of the simulation and
    # Change the timelines in ALL of the csv's
#        if year == 2015:

        for r in range(len(titles['RTI'])):

            # Useful energy demand by boilers
            # The historical data contains final energy demand
            data['HEWG'][r, :, 0] = data['HEWF'][r, :, 0] * data['BHTC'][r, :, c4ti["9 Conversion efficiency"]]

            # Total useful heat demand
            # This is the demand driver for FTT:Heat
            data['RHUD'][r, 0, 0] = np.sum(data['HEWG'][r, :, 0])



```

```python


            if data['RHUD'][r, 0, 0] > 0.0:

                # Market shares (based on useful energy demand)
                data['HEWS'][r, :, 0] = data['HEWG'][r, :, 0] / data['RHUD'][r, 0, 0]

                # CORRECTION TO MARKET SHARES
                # Sometimes historical market shares do not add up to 1.0
                if (~np.isclose(np.sum(data['HEWS'][r, :, 0]), 0.0, atol=1e-9)
                        and np.sum(data['HEWS'][r, :, 0]) > 0.0):
                    data['HEWS'][r, :, 0] = np.divide(data['HEWS'][r, :, 0],
                                                       np.sum(data['HEWS'][r, :, 0]))
            # Capacity by boiler
            data['HEWK'][:, :, 0] = divide(data['HEWG'][:, :, 0],
                                  data['BHTC'][:, :, c4ti["13 Capacity factor mean"]])/1000
            # Emissions
            # TODO: Cost titles are wrong
            data['HEWE'][r, :, 0] = data['HEWF'][r, :, 0] * data['BHTC'][r, :, c4ti["15 Empty"]]

            # Final energy demand by energy carrier
            for fuel in range(len(titles['JTI'])):
                # Fuel use for heating
                data['HJHF'][r, fuel, 0] = np.sum(data['HEWF'][r, :, 0] * data['HJET'][0, :, fuel])
                # Fuel use for total residential sector
                if data['HJFC'][r, fuel, 0] > 0.0:
                    data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]

    if year == histend['HEWF']:
        # Historical data ends in 2014, so we need to initialise data
        # when it's 2015 to make sure the model runs.
        # At some point we need to change the start year of the simulation and
        # Change the timelines in ALL of the csv's
```

```
        # Endogenous price rates
        endog_price_rate = divide(data['MEWP'][:, :, 0],
                                  time_lag['MEWP'][:, :, 0])

        # If switch is set to 1, then an exogenous price rate is used
        # Otherwise, the price rates are set to endogenous

        #data['HFPR'][:, :, 0] = copy.deepcopy(data['HFFC'][:, :, 0])

        # Now transform price rates by fuel to price rates by boiler
        data['HEWP'][:, :, 0] = np.matmul(data['HFFC'][:, :, 0], data['HJET'][0, :, :].T)

        for r in range(len(titles['RTI'])):

            # Sales are the difference between fleet sizes and the addition of scrapped vehicles
            for tech in range(len(titles['HTTI'])):
                if (data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]) > 0:
                    data['HEWI'][r, tech, 0] = data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]

            data['HEWI'][r, :, 0] += divide(data['HEWK'][r, :, 0], data['HETR'][r, :, 0])

                             # Final energy demand by energy carrier
            for fuel in range(len(titles['JTI'])):

                # Fuel use for heating
                data['HJHF'][r, fuel, 0] = np.sum(data['HEWF'][r, :, 0] * data['HJET'][0, :, fuel])

                # Fuel use for total residential sector #HFUX is missing
                if data['HJFC'][r, fuel, 0] > 0.0:
                    data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]

        # Calculate the LCOT for each vehicle type.
        # Call the function
```

```
        data = get_lcoh(data, titles)
# %% Simulation of stock and energy specs
#    t0 = time.time()
    # Stock based solutions first
#    if np.any(specs[sector] < 5):

    # Endogenous calculation takes over from here
    if year > histend['HEWF']:

        # Create a local dictionary for timeloop variables
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():
            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable
        isReg = np.zeros([len(titles['RTI']), len(titles['HTTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['HTTI'])])
        division = divide((data_dt['HEWS'][:, :, 0] - data['HREG'][:, :, 0]),
                          data_dt['HREG'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['HREG'][:, :, 0] == 0.0] = 1.0
        isReg[data['HREG'][:, :, 0] == -1.0] = 0.0

        # Factor used to create quarterly data from annual figures
        no_it = 1
        dt = 1 / no_it

        ############## Computing new shares #################

        #Start the computation of shares
```

```
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            rhudt = time_lag['RHUD'][:, :, :] + (data['RHUD'][:, :, :] - time_lag['RHUD'][:, :, :]) * t * dt

            for r in range(len(titles['RTI'])):

                if rhudt[r] == 0.0:
                    continue

            ############################ FTT ##################################
#                         t3 = time.time()
#                         print("Solving {}".format(titles["RTI"][r]))
                # Initialise variables related to market share dynamics
                # DSiK contains the change in shares
                dSik = np.zeros([len(titles['HTTI']), len(titles['HTTI'])])

                # F contains the preferences
                F = np.ones([len(titles['HTTI']), len(titles['HTTI'])])*0.5

                # -----------------------------------------------------
                # Step 1: Endogenous EOL replacements
                # -----------------------------------------------------
                for b1 in range(len(titles['HTTI'])):

                    if  not (data_dt['HEWS'][r, b1, 0] > 0.0 and
                             data_dt['HGC1'][r, b1, 0] != 0.0 and
                             data_dt['HWCD'][r, b1, 0] != 0.0):
                        continue

                    S_i = data_dt['HEWS'][r, b1, 0]

                    for b2 in range(b1):
```

```
                            if  not (data_dt['HEWS'][r, b2, 0] > 0.0 and
                                     data_dt['HGC1'][r, b2, 0] != 0.0 and
                                     data_dt['HWCD'][r, b2, 0] != 0.0):
                                continue

                            S_k = data_dt['HEWS'][r, b2, 0]

                            # Propagating width of variations in perceived costs
                            dFik = sqrt(2) * sqrt((data_dt['HWCD'][r, b1, 0]*data_dt['HWCD'][r, b1, 0] + data_dt['HWCD'][r, b2, 0]*data_dt['HWCD'][r, b2, 0]))

                            # Consumer preference incl. uncertainty
                            Fik = 0.5*(1+np.tanh(1.25*(data_dt['HGC1'][r, b2, 0]-data_dt['HGC1'][r, b1, 0])/dFik))

                            # Preferences are then adjusted for regulations
                            F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
                            F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])

                            #Runge-Kutta market share dynamiccs
                            k_1 = S_i*S_k * (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
                            k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
                            k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
                            k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])

                            # Market share dynamics
                            #dSik[b1, b2] = S_i*S_k* (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data
```

```
                      ['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])*dt
                    dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                    dSik[b2, b1] = -dSik[b1, b2]

            # -------------------------------------------------------
            # Step 2: Endogenous premature replacements
            # -------------------------------------------------------
            # Initialise variables related to market share dynamics
            # DSiK contains the change in shares
            dSEik = np.zeros([len(titles['HTTI']), len(titles['HTTI'])])

            # F contains the preferences
            FE = np.ones([len(titles['HTTI']), len(titles['HTTI'])])*0.5

            # Intermediate shares: add the EoL effects before continuing
            # intermediate_shares = data_dt['HEWS'][r, :, 0] + np.sum(dSik, axis=1)

            # Scrappage rate
            SR = divide(np.ones([len(titles['HTTI'])]),
                        data['BHTC'][r, :, c4ti["16 Empty"]]) - data['HETR'][r, :, 0]
            SR = np.where(SR<0.0, 0.0, SR)

            for b1 in range(len(titles['HTTI'])):

                if not (data_dt['HEWS'][r, b1, 0] > 0.0 and
                        data_dt['HGC2'][r, b1, 0] != 0.0 and
                        data_dt['HGD2'][r, b1, 0] != 0.0 and
                        data_dt['HGC3'][r, b1, 0] != 0.0 and
                        data_dt['HGD3'][r, b1, 0] != 0.0):
                    continue

                SE_i = data_dt['HEWS'][r, b1, 0]

```

```
                    for b2 in range(b1):

                        if not (data_dt['HEWS'][r, b2, 0] > 0.0 and
                                data_dt['HGC2'][r, b2, 0] != 0.0 and
                                data_dt['HGD2'][r, b2, 0] != 0.0 and
                                data_dt['HGC3'][r, b2, 0] != 0.0 and
                                data_dt['HGD3'][r, b2, 0] != 0.0):
                            continue

                        SE_k = data_dt['HEWS'][r, b2, 0]

                        # NOTE: Premature replacements are optional for
                        # consumers. It is possible that NO premature
                        # replacements take place

                        # Propagating width of variations in perceived costs
                        dFEik = sqrt(2) * sqrt((data_dt['HGD3'][r, b1, 0]*data_dt['HGD3'][r, b1, 0] + data_dt['HGD2'][r, b2, 0]*data_dt['HGD2'][r, b2, 0]))
                        dFEki = sqrt(2) * sqrt((data_dt['HGD2'][r, b1, 0]*data_dt['HGD2'][r, b1, 0] + data_dt['HGC3'][r, b2, 0]*data_dt['HGC3'][r, b2, 0]))

                        # Consumer preference incl. uncertainty
                        FEik = 0.5*(1+np.tanh(1.25*(data_dt['HGC2'][r, b2, 0]-data_dt['HGC3'][r, b1, 0])/dFEik))
                        FEki = 0.5*(1+np.tanh(1.25*(data_dt['HGC2'][r, b1, 0]-data_dt['HGC3'][r, b2, 0])/dFEki))

                        # Preferences are then adjusted for regulations
                        FE[b1, b2] = FEik*(1.0-isReg[r, b1])
                        FE[b2, b1] = FEki*(1.0-isReg[r, b2])

                        # Market share dynamics
                        dSEik[b1, b2] = SE_i*SE_k* (data['HEWA'][0,b1, b2]*SR[b2]*FE[b1,b2] - data['HEWA'][0,b2, b1]*SR[b1]*FE[b2,b1])*dt
                        dSEik[b2, b1] = -dSEik[b1, b2]
```

```
                        # TODO: Calculate additional EOL values

                # ------------------------------------------------------
                # Step 3: Exogenous sales additions
                # ------------------------------------------------------
                # Add in exogenous sales figures. These are blended with
                # endogenous result! Note that it's different from the
                # ExogSales specification!
                Utot = rhudt[r]
                dSk = np.zeros([len(titles['HTTI'])])
                dUk = np.zeros([len(titles['HTTI'])])
                dUkTK = np.zeros([len(titles['HTTI'])])
                dUkREG = np.zeros([len(titles['HTTI'])])


                # Check that exogenous share changes add to zero
                dUkTK = data['HWSA'][r, :, 0]
                if (data['HWSA'][r, :, 0].sum() > 0.0):
                    dUkTK[0] = dUkTK[0] - data['HWSA'][r, :, 0].sum()

                # Correct for regulations #TODO Does this actually make sense?

                if time_lag['RHUD'][r, 0, 0] > 0.0 and rhudt[r] > 0.0 and (rhudt[r] - time_lag['RHUD'][r, 0, 0]) > 0.0:

                    dUkREG = -data_dt['HEWG'][r, :, 0] * ( (rhudt[r] - time_lag['RHUD'][r, 0, 0]) /
                                        time_lag['RHUD'][r, 0, 0]) * isReg[r, :].reshape([len(titles['HTTI'])])


                # Sum effect of exogenous sales additions (if any) with
                # effect of regulations
                dUk = dUkREG
```

```
                dUtot = np.sum(dUk)

                # Convert to market shares and make sure sum is zero
                # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
                dSk = np.divide(dUk, Utot) - data_dt['HEWG'][r, :, 0]*np.divide(dUtot, (Utot*Utot)) + dUkTK

#                        soel = np.sum(dSik, axis=1)
#                        st_1 = data_dt['TP_MS'][r, :, 0]
                # New market shares
                # Implement check that market shares sum to 1
#                        print(np.sum(dSik, axis=1))
                data['HEWS'][r, :, 0] = data_dt['HEWS'][r, :, 0] + np.sum(dSik, axis=1) + np.sum(dSEik, axis=1) + dSk[0, :]

                if ~np.isclose(np.sum(data['HEWS'][r, :, 0]), 1.0, atol=1e-2):
                    msg = """Sector: {} - Region: {} - Year: {}
                    Sum of market shares do not add to 1.0 (instead: {})
                    """.format(sector, titles['RTI'][r], year, np.sum(data['HEWS'][r, :, 0]))
                    warnings.warn(msg)

                if np.any(data['HEWS'][r, :, 0] < 0.0):
                    msg = """Sector: {} - Region: {} - Year: {}
                    Negative market shares detected! Critical error!
                    """.format(sector, titles['RTI'][r], year)
                    warnings.warn(msg)
#                        t4 = time.time()
#                        print("Share equation takes {}".format(t4-t3))

            ############## Update variables ##################
            # Useful heat by boiler
            data['HEWG'][:, :, 0] = data['HEWS'][:, :, 0] * rhudt[:, 0, 0, np.newaxis]

            # Final energy by boiler
```

```
            data['HEWF'][:, :, 0] = divide(data['HEWG'][:, :, 0],
                                           data['BHTC'][:, :, c4ti["9 Conversion efficiency"]])

            # Capacity by boiler
            data['HEWK'][:, :, 0] = divide(data['HEWG'][:, :, 0],
                                            data['BHTC'][:, :, c4ti["13 Capacity factor mean"]])/1000

            # Emissions
            # TODO: Cost titles are wrong
            data['HEWE'][:, :, 0] = data['HEWF'][:, :, 0] * data['BHTC'][:, :, c4ti["15 Empty"]]

            #data['HFPR'][:, :, 0] = copy.deepcopy(data['HFFC'][:, :, 0])

            # Now transform price rates by fuel to price rates by boiler
            data['HEWP'][:, :, 0] = np.matmul(data['HFFC'][:, :, 0], data['HJET'][0, :, :].T)

            # Sales are the difference between fleet sizes and the addition of scrapped vehicles
            for r in range(len(titles['RTI'])):
                for tech in range(len(titles['HTTI'])):
                    if (data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]) > 0:
                        data['HEWI'][r, tech, 0] = data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]
            data['HEWI'][:, :, 0] += divide(data['HEWK'][:, :, 0], data['HETR'][:, :, 0])

            # Corrections to sales and EOL when sales are negative.
            #condition = data['HEWI'][:, :, 0] < 0.0
            # data['RH_EOL'][:, :, 0] = np.where(condition,
            #                                     data['RH_EOL'][:, :, 0] - data['HEWI'][:, :, 0],
            #                                     data['RH_EOL'][:, :, 0])

            data['HEWI'][:, :, 0] = np.where(data['HEWI'][:, :, 0] < 0.0,
                                             0.0,
                                             data['HEWI'][:, :, 0])

```

```
            # Final energy demand for heating purposes
            data['HJHF'][:, :, 0] = np.matmul(data['HEWF'][:, :, 0], data['HJET'][0, :, :])

            # Final energy demand of the residential sector (incl. non-heat)
            # For the time being, this is calculated as a simply scale-up
            for fuel in range(len(titles['JTI'])):
                if data['HJFC'][r, fuel, 0] > 0.0:
                    data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]

            ############## Learning-by-doing #################

            # Cumulative global learning
            # Using a technological spill-over matrix (HEWB) together with capacity
            # additions (HEWI) we can estimate total global spillover of similar
            # technologies
            bi = np.zeros((len(titles['RTI']),len(titles['HTTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['HEWB'][0, :, :],data['HEWI'][r, :, 0])
            dw = np.sum(bi, axis=0)*dt


            # Cumulative capacity incl. learning spill-over effects
            data['HEWW'][0, :, 0] = data_dt['HEWW'][0, :, 0] + dw

            # Copy over the technology cost categories that do not change (all except prices which are updated
              through learning-by-doing below)
            data['BHTC'] = copy.deepcopy(data_dt['BHTC'])

            # Learning-by-doing effects on investment
            for b in range(len(titles['HTTI'])):

                if data['HEWW'][0, b, 0] > 0.1:

```

```
                    data['BHTC'][:, b, c4ti['1 Investment cost mean']] = data_dt['BHTC'][:, b, c4ti['1 Investment cost mean']] * \
                                                                                            (1.0 + data['BHTC'][:, b, c4ti['7 Learning rate']] * dw[b]/data['HEWW'][0, b, 0])



            ############## Final output ##################

            # Get LCOT
            if t ==1:

                data = get_lcoh(data, titles)

                # Update time loop variables:
                for var in data_dt.keys():

                    data_dt[var] = copy.deepcopy(data[var])


    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_chi_main.py
=========================================
Industrial chemical sector FTT module.
######################################


This is the main file for FTT: Industrial Heat - CHI, which models technological
diffusion of industrial heat processes within the chemical sector due
to simulated investor decision making. Investors compare the **levelised cost of
industrial heat**, which leads to changes in the market shares of different technologies.

The outputs of this module include changes in final energy demand and emissions due
chemical heat processes for the EU28.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros

Functions included:

    - solve
        Main solution function for the module
    - get_lcoih
        Calculates the levelised cost of industrial heat

"""
# Standard library imports
```

```python
from math import sqrt
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
#from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# -------------------------- LCOH function -----------------------------------
# -----------------------------------------------------------------------------
def get_lcoih(data, titles, year):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of industrial heat in 2019 Euros
    It includes intangible costs (gamma values) and together
    determines the investor preferences.

    Parameters
    -----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) for all
        variables. Variable names are keys and the values are 3D NumPy arrays.
    titles: dictionary
```

```
        Titles is a container of all permissible dimension titles of the model.

    Returns
    ----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) data for
        all variables.
        Variable names are keys and the values are 3D NumPy arrays.
        The values inside the container are updated and returned to the main
        routine.

    Notes
    ---------
    Additional notes if required.
    """

    # Categories for the cost matrix (BIC1)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    for r in range(len(titles['RTI'])):
        if data['IUD1'][r, :, 0].sum(axis=0)==0:
            continue


        lt = data['BIC1'][r,:, ctti['5 Lifetime (years)']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
        lt_mat = np.where(mask, lt_mat, 0)


```

```
        # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
        cf = data['BIC1'][r,:, ctti['13 Capacity factor mean'], np.newaxis]

        #conversion efficiency
        ce = data['BIC1'][r,:, ctti['9 Conversion efficiency'], np.newaxis]

        # Trap for very low CF
        cf[cf<0.000001] = 0.000001

        # Factor to transfer cost components in terms of capacity to generation
        conv = 1/(cf)/8766 #number of hours in a year

        # Discount rate
        # dr = BIC1[6]
        dr = data['BIC1'][r,:, ctti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['ITTI']), int(max_lt)])
        it[:, 0, np.newaxis] =  data['BIC1'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]*
          (1000000)*conv



        # Standard deviation of investment cost
        dit = np.zeros([len(titles['ITTI']), int(max_lt)])
        dit[:, 0, np.newaxis] =  data['BIC1'][r,:, ctti['2 Investment cost SD'], np.newaxis] *(1000000)*conv


        # Subsidies as a percentage of investment cost
        st = np.zeros([len(titles['ITTI']), int(max_lt)])
        st[:, 0, np.newaxis] = (data['BIC1'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
             * data['ISB1'][r, :, 0,np.newaxis] *conv)*(1000000)
```

```python


        # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
        ft = np.ones([len(titles['ITTI']), int(max_lt)])
        ft = ft * data['BIC1'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
        ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['ITTI']), int(max_lt)])
        dft = dft * data['BIC1'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
        dft = np.where(mask, dft, 0)

        #fuel tax/subsidies
        #fft = np.ones([len(titles['ITTI']), int(max_lt)])
#         fft = ft * data['PG_FUELTAX'][r, :, :]
#         fft = np.where(lt_mask, ft, 0)

        # Fixed operation & maintenance cost - variable O&M available but not included
        omt = np.ones([len(titles['ITTI']), int(max_lt)])
        omt = omt * data['BIC1'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) in a year
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['ITTI']), int(max_lt)])
        domt = domt * data['BIC1'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
        domt = np.where(mask, domt, 0)

        # Net present value calculations
        # Discount rate
        denominator = (1+dr)**(lt_mat)

        # 1-Expenses
```

```
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+omt)/denominator
        # 1.3-Only policy costs
        #npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1

        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT

        lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)

        # 1.2-LCOT including policy costs
        tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
        # 1.3 LCOE excluding policy, including co2 price
        #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
        # Standard deviation of LCOT
        dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

        # LCOE augmented with gamma values, no gamma values yet
        tlcoeg = tlcoe+data['IAM1'][r, :, 0]

        # Pass to variables that are stored outside.
```

```
        data['ILC1'][r, :, 0] = lcoe            # The real bare LCOT without taxes (euros/mwh)
        #data['IHLT'][r, :, 0] = tlcoe           # The real bare LCOE with taxes
        data['ILG1'][r, :, 0] = tlcoeg         # As seen by consumer (generalised cost)
        data['ILD1'][r, :, 0] = dlcoe          # Variation on the LCOT distribution

    return data



# %% main function
# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
    """

    Main solution function for the module.

    Simulates investor decision making.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
        Description
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
```

```
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution

    """

    # Categories for the cost matrix (BIC1)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    sector = 'chemical'


    data = get_lcoih(data, titles, year)

    # Endogenous calculation takes over from here
    if year > histend['IUD1']:

        # Create a local dictionary for timeloop variables
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():

            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
        isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
```

```
        division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = divide((data_dt['IWK1'][:, :, 0] - data['IRG1'][:, :, 0]),
                          data_dt['IRG1'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['IRG1'][:, :, 0] == 0.0] = 1.0
        isReg[data['IRG1'][:, :, 0] == -1.0] = 0.0


        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it
        kappa = 10 #tech substitution constant

        ############## Computing total useful energy demand ##################

        IUD1tot = data['IUD1'][:, :, 0].sum(axis=1)

        #Start the computation of shares
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            #Time lagged UED plus change in UED * (no of iterations) * dt

            IUD1t = time_lag['IUD1'][:, :, 0].sum(axis=1) + (IUD1tot - time_lag['IUD1'][:, :, 0].sum(axis=1)) * t * dt

            for r in range(len(titles['RTI'])):

                if IUD1t[r] == 0.0:
                    continue



```

```
        ############################ FTT ##################################

            # DSiK contains the change in shares
            dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])

            # F contains the preferences
            F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5

            # Market share constraints
            Gijmax = np.ones(len(titles['ITTI']))
            #Gijmin = np.ones((t2ti))

            # -------------------------------------------------------
            # Step 1: Endogenous EOL replacements
            # -------------------------------------------------------
            for b1 in range(len(titles['ITTI'])):

                if  not (data_dt['IWS1'][r, b1, 0] > 0.0 and
                         data_dt['ILG1'][r, b1, 0] != 0.0 and
                         data_dt['ILD1'][r, b1, 0] != 0.0):
                    continue

                #TODO: create market share constraints
                Gijmax[b1] = np.tanh(1.25*(data_dt['ISC1'][0, b1, 0] - data_dt['IWS1'][r, b1, 0])/0.1)
                #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)



                S_i = data_dt['IWS1'][r, b1, 0]


                for b2 in range(b1):

```

```python
if  not (data_dt['IWS1'][r, b2, 0] > 0.0 and
         data_dt['ILG1'][r, b2, 0] != 0.0 and
         data_dt['ILD1'][r, b2, 0] != 0.0):
    continue

S_k = data_dt['IWS1'][r,b2, 0]
Aik = data['IWA1'][0,b1 , b2]*kappa
Aki = data['IWA1'][0,b2, b1]*kappa

# Propagating width of variations in perceived costs
dFik = sqrt(2) * sqrt((data_dt['ILD1'][r, b1, 0]*data_dt['ILD1'][r, b1, 0] + data_dt['ILD1'][r, b2, 0]*data_dt['ILD1'][r, b2, 0]))

# Consumer preference incl. uncertainty
Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG1'][r, b2, 0]-data_dt['ILG1'][r, b1, 0])/dFik))

# Preferences are then adjusted for regulations
F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])


#Runge-Kutta market share dynamiccs
k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])

#This method currently applies RK4 to the shares, but all other components of the equation are calculated for the overall time step
#We must assume the the LCOE does not change significantly in a time step dt, so we can focus on the shares.
```

```
                    dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                    dSik[b2, b1] = -dSik[b1, b2]

                    #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
                    #dSik[b2, b1] = -dSik[b1, b2]


            # -------------------------------------------------------
            # Step 3: Exogenous sales additions
            # -------------------------------------------------------
            # Add in exogenous sales figures. These are blended with endogenous result!


            # Add in exogenous sales figures. These are blended with
            # endogenous result! Note that it's different from the
            # ExogSales specification!
            Utot = IUD1t[r]
            iud_lag = time_lag['IUD1'][:, :, 0].sum(axis=1)
            dSk = np.zeros((len(titles['ITTI'])))
            dUk = np.zeros((len(titles['ITTI'])))
            dUkTK = np.zeros((len(titles['ITTI'])))
            dUkREG = np.zeros((len(titles['ITTI'])))

            # Check that exogenous share changes add to zero
            dUkTK = data['IXS1'][r, :, 0]
            if (data['IXS1'][r, :, 0].sum() > 0.0):
                dUkTK[0] = dUkTK[0] - data['IXS1'][r, :, 0].sum()

            # Correct for regulations

            if iud_lag[r] > 0.0 and IUD1t[r] > 0.0 and (IUD1t[r] - iud_lag[r]) > 0.0:

```

```
                        dUkREG = -data_dt['IUD1'][r, :, 0] * ( (IUD1t[r] - iud_lag[r]) /
                                        iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])


                    # Sum effect of exogenous sales additions (if any) with
                    # effect of regulations
                    dUk = copy.deepcopy(dUkREG)
                    dUtot = np.sum(dUk)

                    # Convert to market shares and make sure sum is zero
                    # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
                    dSk = np.divide(dUk, Utot) - time_lag['IWS1'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK


                    # New market shares
                    # check that market shares sum to 1

                    data['IWS1'][r, :, 0] = data_dt['IWS1'][r, :, 0] + np.sum(dSik, axis=1) + dSk

                    if ~np.isclose(np.sum(data['IWS1'][r, :, 0]), 1.0, atol=1e-5):
                        msg = """Sector: {} - Region: {} - Year: {}
                        Sum of market shares do not add to 1.0 (instead: {})
                        """.format(sector, titles['RTI'][r], year, np.sum(data['IWS1'][r, :, 0]))
                        warnings.warn(msg)

                    if np.any(data['IWS1'][r, :, 0] < 0.0):
                        msg = """Sector: {} - Region: {} - Year: {}
                        Negative market shares detected! Critical error!
                        """.format(sector, titles['RTI'][r], year)
                        warnings.warn(msg)



```

```python

        # ============================================================
        #  Update variables
        # ============================================================



        #Useful heat by technology, calculate based on new market shares #Regional totals
        data['IUD1'][:, :, 0] = data['IWS1'][:, :, 0]* IUD1t[:, np.newaxis]

        # Capacity by technology
        data['IWK1'][:, :, 0] = divide(data['IUD1'][:, :, 0],
                                          data['BIC1'][:, :, ctti["13 Capacity factor mean"]]*8766)
        #add number of devices replaced due to breakdowns = IWK1_lagged/lifetime to yearly capacity additions
        #note some values of IWI1 negative

        data["IWI1"][:, :, 0] = 0
        for r in range(len(titles['RTI'])):
            for tech in range(len(titles['ITTI'])):
                if(data['IWK1'][r, tech, 0]-time_lag['IWK1'][r, tech, 0]) > 0:
                    data["IWI1"][r, tech, 0] = (data['IWK1'][r, tech, 0]-time_lag['IWK1'][r, tech, 0])

        data["IWI1"][:, :, 0] = data["IWI1"][:, :, 0] + np.where(data['BIC1'][:, :, ctti['5 Lifetime (years)']] !=0.0,
                                                        divide(time_lag['IWK1'][:, :, 0],
                                                        data['BIC1'][:, :, ctti['5 Lifetime (years)']]),0.0)

        #Update emissions
        #IHW1 is the global average emissions per unit of UED (GWh). IHW1 has units of kt of CO2/GWh
        for r in range(len(titles['RTI'])):
            data['IWE1'][r, :, 0] = data['IUD1'][r, :, 0] * data['IHW1'][0, :, 0]


```

```
            #Final energy by technology
            data['IFD1'][:, :, 0] = np.where(data['BIC1'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
                                             divide(data['IUD1'][:, :, 0],
                                                    data['BIC1'][:, :, ctti["9 Conversion efficiency"]]),0.0)



            # =============================================================
            # Learning-by-doing
            # =============================================================

            # Cumulative global learning
            # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
            # additions (IWI1 Capacity additions) we can estimate total global spillover of similar
            # techicals



            bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['IWB1'][0, :, :],data['IWI1'][r, :, 0])
            dw = np.sum(bi, axis=0)*dt

            # # Cumulative capacity incl. learning spill-over effects
            data["IWW1"][0, :, 0] = data_dt['IWW1'][0, :, 0] + dw
            #
            # # Copy over the technology cost categories that do not change (all except prices which are updated through learning-by-doing below)
            data['BIC1'] = copy.deepcopy(data_dt['BIC1'])
            #
            # # Learning-by-doing effects on investment
            for tech in range(len(titles['ITTI'])):

```

```
            if data['IWW1'][0, tech, 0] > 0.1:

                data['BIC1'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC1'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] * \
                                                                           (1.0 + data['BIC1'][:, tech, ctti['15 Learning exponent']] * dw[tech]/data['IWW1'][0, tech, 0])

        # ===================================================================
        # Update the time-loop variables
        # ===================================================================

        #Calculate levelised cost again
        data = get_lcoih(data, titles, year)

        #Update time loop variables:
        for var in data_dt.keys():

            data_dt[var] = copy.deepcopy(data[var])


    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_fbt_main.py
=========================================
Industrial food, beverages, and tobacco sector FTT module.
##########################################################


This is the main file for FTT: Industrial Heat - FBT, which models technological
diffusion of industrial heat processes within the food, beverages, and tobacco sector due
to simulated investor decision making. Investors compare the **levelised cost of
industrial heat**, which leads to changes in the market shares of different technologies.

The outputs of this module include changes in final energy demand and emissions due
chemical heat processes for the EU28.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros

Functions included:

    - solve
        Main solution function for the module
    - get_lcoih
        Calculates the levelised cost of industrial heat

"""
# Standard library imports
```

```python
from math import sqrt
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# -------------------------- LCOH function ------------------------------------
# -----------------------------------------------------------------------------
def get_lcoih(data, titles, year):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of industrial heat in 2019 Euros
    It includes intangible costs (gamma values) and together
    determines the investor preferences.

    Parameters
    -----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) for all
        variables. Variable names are keys and the values are 3D NumPy arrays.
    titles: dictionary
```

```
        Titles is a container of all permissible dimension titles of the model.

    Returns
    ----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) data for
        all variables.
        Variable names are keys and the values are 3D NumPy arrays.
        The values inside the container are updated and returned to the main
        routine.

    Notes
    ---------
    Additional notes if required.
    """

    # Categories for the cost matrix (BIC2)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    for r in range(len(titles['RTI'])):
        if data['IUD2'][r, :, 0].sum(axis=0)==0:
            continue

        # Cost matrix
        #BIC2 = data['BIC2'][r, :, :]

        lt = data['BIC2'][r,:, ctti['5 Lifetime (years)']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
        lt_mat = np.where(mask, lt_mat, 0)
```

```python



        # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
        cf = data['BIC2'][r,:, ctti['13 Capacity factor mean'], np.newaxis]

        #conversion efficiency
        ce = data['BIC2'][r,:, ctti['9 Conversion efficiency'], np.newaxis]

        # Trap for very low CF
        cf[cf<0.000001] = 0.000001

        # Factor to transfer cost components in terms of capacity to generation
#        ones = np.ones([len(titles['ITTI']), 1])
        conv = 1/(cf)/8766 #number of hours in a year

        # Discount rate
        # dr = data['BIC2'][r,6]
        dr = data['BIC2'][r,:, ctti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['ITTI']), int(max_lt)])
        it[:, 0, np.newaxis] =  data['BIC2'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis] * conv*(1000000)


        # Standard deviation of investment cost
        dit = np.zeros([len(titles['ITTI']), int(max_lt)])
        dit[:, 0, np.newaxis] =  data['BIC2'][r,:, ctti['2 Investment cost SD'], np.newaxis] * conv*(1000000)


        # Subsidies as a percentage of investment cost
```

```python
        st = np.zeros([len(titles['ITTI']), int(max_lt)])
        st[:, 0, np.newaxis] = (data['BIC2'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
             * data['ISB2'][r, :, 0,np.newaxis] * conv)*(1000000)


        # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
        ft = np.ones([len(titles['ITTI']), int(max_lt)])
        ft = ft * data['BIC2'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
        ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['ITTI']), int(max_lt)])
        dft = dft * data['BIC2'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
        dft = np.where(mask, dft, 0)

        #fuel tax/subsidies
        #fft = np.ones([len(titles['ITTI']), int(max_lt)])
#        fft = ft * data['PG_FUELTAX'][r, :, :]
#        fft = np.where(lt_mask, ft, 0)

        # Fixed operation & maintenance cost - variable O&M available but not included
        omt = np.ones([len(titles['ITTI']), int(max_lt)])
        omt = omt * data['BIC2'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) in a year
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['ITTI']), int(max_lt)])
        domt = domt * data['BIC2'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
        domt = np.where(mask, domt, 0)
```

```
        # Net present value calculations
        # Discount rate
        denominator = (1+dr)**(lt_mat)

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+omt)/denominator
        # 1.3-Only policy costs
        #npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max but keep t=0 as 1
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT

        lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)

        # 1.2-LCOT including policy costs
        tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
        # 1.3 LCOE excluding policy, including co2 price
        #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
        # Standard deviation of LCOT
        dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

```

```
        # LCOE augmented with gamma values, no gamma values yet
        tlcoeg = tlcoe+data['IAM2'][r, :, 0]

        # Pass to variables that are stored outside.
        data['ILC2'][r, :, 0] = lcoe            # The real bare LCOT without taxes (euros/mwh)
        #data['IHLT'][r, :, 0] = tlcoe           # The real bare LCOE with taxes
        data['ILG2'][r, :, 0] = tlcoeg         # As seen by consumer (generalised cost)
        data['ILD2'][r, :, 0] = dlcoe          # Variation on the LCOT distribution



    return data

#Final energy demand has to match IEA

# %% main function
# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
    """

    Main solution function for the module.

    Simulates investor decision making.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
```

```
        Description
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution


    """

    # Categories for the cost matrix (BIC2)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    sector = 'Food, beverages and tobacco'

    #Get fuel prices from E3ME and add them to the data for this code
    #Initialise everything #TODO

    #Calculate or read in FED
    #Calculate historical emissions
    data = get_lcoih(data, titles, year)

    # Endogenous calculation takes over from here
    if year > histend['IUD2']:

```

```
        # Create a local dictionary for timeloop variables
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():


            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
        isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = divide((data_dt['IWK2'][:, :, 0] - data['IRG2'][:, :, 0]),
                          data_dt['IRG2'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['IRG2'][:, :, 0] == 0.0] = 1.0
        isReg[data['IRG2'][:, :, 0] == -1.0] = 0.0


        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it
        kappa = 10 #tech substitution constant

        ############## Computing new shares #################

        IUD2tot = data['IUD2'][:, :, 0].sum(axis=1)
        #Start the computation of shares
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            #Time lagged UED plus change in UED * (no of iterations) * dt
```

```
            IUD2t = time_lag['IUD2'][:, :, 0].sum(axis=1) + (IUD2tot - time_lag['IUD2'][:, :, 0].sum(axis=1)) * t * dt

            for r in range(len(titles['RTI'])):

                if IUD2t[r] == 0.0:
                    continue



            ############################ FTT ##################################

                # DSiK contains the change in shares
                dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])

                # F contains the preferences
                F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5

                # Market share constraints
                Gijmax = np.ones(len(titles['ITTI']))
                #Gijmin = np.ones((t2ti))

                # -------------------------------------------------------
                # Step 1: Endogenous EOL replacements
                # -------------------------------------------------------
                for b1 in range(len(titles['ITTI'])):

                    if  not (data_dt['IWS2'][r, b1, 0] > 0.0 and
                             data_dt['ILG2'][r, b1, 0] != 0.0 and
                             data_dt['ILD2'][r, b1, 0] != 0.0):
                        continue

```

```
                #TODO: create market share constraints
                Gijmax[b1] = np.tanh(1.25*(data_dt['ISC2'][0, b1, 0] - data_dt['IWS2'][r, b1, 0])/0.1)
                #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)



                S_i = data_dt['IWS2'][r, b1, 0]


                for b2 in range(b1):

                    if  not (data_dt['IWS2'][r, b2, 0] > 0.0 and
                             data_dt['ILG2'][r, b2, 0] != 0.0 and
                             data_dt['ILD2'][r, b2, 0] != 0.0):
                        continue

                    S_k = data_dt['IWS2'][r,b2, 0]
                    Aik = data['IWA2'][0,b1 , b2]*kappa
                    Aki = data['IWA2'][0,b2, b1]*kappa

                    # Propagating width of variations in perceived costs
                    dFik = sqrt(2) * sqrt((data_dt['ILD2'][r, b1, 0]*data_dt['ILD2'][r, b1, 0] + data_dt['ILD2'][r, b2, 0]*data_dt['ILD2'][r, b2, 0]))

                    # Consumer preference incl. uncertainty
                    Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG2'][r, b2, 0]-data_dt['ILG2'][r, b1, 0])/dFik))

                    # Preferences are then adjusted for regulations
                    F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
                    F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])

```

```

                    #Runge-Kutta market share dynamiccs
                    k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                    k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                    k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                    k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])

                    #This method currently applies RK4 to the shares, but all other components of the equation
                      are calculated for the overall time step
                    #We must assume the the LCOE does not change significantly in a time step dt, so we can
                      focus on the shares.

                    dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                    dSik[b2, b1] = -dSik[b1, b2]

                    #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
                    #dSik[b2, b1] = -dSik[b1, b2]


            # -------------------------------------------------------
            # Step 3: Exogenous sales additions
            # -------------------------------------------------------
            # Add in exogenous sales figures. These are blended with endogenous result!


            # Add in exogenous sales figures. These are blended with
            # endogenous result! Note that it's different from the
            # ExogSales specification!
            Utot = IUD2t[r]
            iud_lag = time_lag['IUD2'][:, :, 0].sum(axis=1)
            dSk = np.zeros((len(titles['ITTI'])))
            dUk = np.zeros((len(titles['ITTI'])))
            dUkTK = np.zeros((len(titles['ITTI'])))
```

```
                    dUkREG = np.zeros((len(titles['ITTI'])))

                    # Check that exogenous share changes add to zero
                    dUkTK = data['IXS2'][r, :, 0]
                    if (data['IXS2'][r, :, 0].sum() > 0.0):
                        dUkTK[0] = dUkTK[0] - data['IXS2'][r, :, 0].sum()

                    # Correct for regulations #TODO Does this actually make sense?

                    if iud_lag[r] > 0.0 and IUD2t[r] > 0.0 and (IUD2t[r] - iud_lag[r]) > 0.0:

                        dUkREG = -data_dt['IUD2'][r, :, 0] * ( (IUD2t[r] - iud_lag[r]) /
                                        iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])


                    # Sum effect of exogenous sales additions (if any) with
                    # effect of regulations
                    dUk = copy.deepcopy(dUkREG)
                    dUtot = np.sum(dUk)

                    # Convert to market shares and make sure sum is zero
                    # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
                    dSk = np.divide(dUk, Utot) - time_lag['IWS2'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK


                    # New market shares
                    # check that market shares sum to 1
                            #print(np.sum(dSik, axis=1))
                    data['IWS2'][r, :, 0] = data_dt['IWS2'][r, :, 0] + np.sum(dSik, axis=1) + dSk

                    if ~np.isclose(np.sum(data['IWS2'][r, :, 0]), 1.0, atol=1e-5):
                        msg = """Sector: {} - Region: {} - Year: {}
                        Sum of market shares do not add to 1.0 (instead: {})
```

```
                """.format(sector, titles['RTI'][r], year, np.sum(data['IWS2'][r, :, 0]))
                warnings.warn(msg)

            if np.any(data['IWS2'][r, :, 0] < 0.0):
                msg = """Sector: {} - Region: {} - Year: {}
                Negative market shares detected! Critical error!
                """.format(sector, titles['RTI'][r], year)
                warnings.warn(msg)




        # ============================================================
        #  Update variables
        # ============================================================

        #TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change

        #Useful heat by technology, calculate based on new market shares #Regional totals
        data['IUD2'][:, :, 0] = data['IWS2'][:, :, 0]* IUD2t[:, np.newaxis]

        # Capacity by technology
        data['IWK2'][:, :, 0] = divide(data['IUD2'][:, :, 0],
                                          data['BIC2'][:, :, ctti["13 Capacity factor mean"]]*8766)
        #add number of devices replaced due to breakdowns = IWK2_lagged/lifetime to yearly capacity additions
        #note some values of IWI2 negative

        data["IWI2"][:, :, 0] = 0
        for r in range(len(titles['RTI'])):
            for tech in range(len(titles['ITTI'])):
                if(data['IWK2'][r, tech, 0]-time_lag['IWK2'][r, tech, 0]) > 0:
                    data["IWI2"][r, tech, 0] = (data['IWK2'][r, tech, 0]-time_lag['IWK2'][r, tech, 0])
        data["IWI2"][:, :, 0] = data["IWI2"][:, :, 0] + np.where(data['BIC2'][:, :, ctti['5 Lifetime
```

```
    (years)']] !=0.0,
                                                                      divide(time_lag['IWK2'][:, :, 0],data
                   ['BIC2'][:, :, ctti['5 Lifetime (years)']]),
                                                                      0.0)

            #Update emissions
            #IHW2 is the global average emissions per unit of UED (GWh). IHW2 has units of kt of CO2/GWh
            for r in range(len(titles['RTI'])):
                data['IWE2'][r, :, 0] = data['IUD2'][r, :, 0] * data['IHW2'][0, :, 0]


            #Final energy by technology
            data['IFD2'][:, :, 0] = np.where(data['BIC2'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
                                             divide(data['IUD2'][:, :, 0],
                                                    data['BIC2'][:, :, ctti["9 Conversion efficiency"]]),0.0)



            # =============================================================
            # Learning-by-doing
            # =============================================================

            # Cumulative global learning
            # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
            # additions (IWI2 Capacity additions) we can estimate total global spillover of similar
            # techicals



            bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['IWB2'][0, :, :],data['IWI2'][r, :, 0])
            dw = np.sum(bi, axis=0)*dt
```

```
            # # Cumulative capacity incl. learning spill-over effects
            data["IWW2"][0, :, 0] = data_dt['IWW2'][0, :, 0] + dw
            #
            # # Copy over the technology cost categories that do not change (all except prices which are updated
              through learning-by-doing below)
            data['BIC2'] = copy.deepcopy(data_dt['BIC2'])
            #
            # # Learning-by-doing effects on investment
            for tech in range(len(titles['ITTI'])):

                if data['IWW2'][0, tech, 0] > 0.1:

                    data['BIC2'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC2'][:, tech,
                       ctti['1 Investment cost mean (MEuro per MW)']] * \
                                                                          (1.0 + data['BIC2'][:, tech, ctti['15
                             Learning exponent']] * dw[tech]/data['IWW2'][0, tech, 0])

            # ===================================================================
            # Update the time-loop variables
            # ===================================================================

            #Calculate levelised cost again
            data = get_lcoih(data, titles, year)

            #Update time loop variables:
            for var in data_dt.keys():


                data_dt[var] = copy.deepcopy(data[var])


    return data
```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_mtm_main.py
=========================================
Industrial non-ferrous metals, machinery, and transport equipment sector FTT module.
####################################################################################


This is the main file for FTT: Industrial Heat - MTM, which models technological
diffusion of industrial heat processes within the non-ferrous metals, machinery, and
transport equipment sector due to simulated investor decision making. Investors compare
the **levelised cost of industrial heat**, which leads to changes in the market shares of
different technologies.

The outputs of this module include changes in final energy demand and emissions due
chemical heat processes for the EU28.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros

Functions included:

    - solve
        Main solution function for the module
    - get_lcoih
        Calculates the levelised cost of industrial heat

"""
```

```
# Standard library imports
from math import sqrt
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# ---------------------------- LCOH function ----------------------------------
# -----------------------------------------------------------------------------
def get_lcoih(data, titles, year):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of industrial heat in 2019 Euros
    It includes intangible costs (gamma values) and together
    determines the investor preferences.

    Parameters
    -----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) for all
        variables. Variable names are keys and the values are 3D NumPy arrays.
```

```
    titles: dictionary
        Titles is a container of all permissible dimension titles of the model.

    Returns
    ----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) data for
        all variables.
        Variable names are keys and the values are 3D NumPy arrays.
        The values inside the container are updated and returned to the main
        routine.

    Notes
    ---------
    Additional notes if required.
    """

    # Categories for the cost matrix (BIC3)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    for r in range(len(titles['RTI'])):
        if data['IUD3'][r, :, 0].sum(axis=0)==0:
            continue

        # Cost matrix
        #BIC3 = data['BIC3'][r, :, :]

        lt = data['BIC3'][r,:, ctti['5 Lifetime (years)']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
```

```
        lt_mat = np.where(mask, lt_mat, 0)

        # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
        cf = data['BIC3'][r,:, ctti['13 Capacity factor mean'], np.newaxis]

        #conversion efficiency
        ce = data['BIC3'][r,:, ctti['9 Conversion efficiency'], np.newaxis]

        # Trap for very low CF
        cf[cf<0.000001] = 0.000001

        # Factor to transfer cost components in terms of capacity to generation
#         ones = np.ones([len(titles['ITTI']), 1])
        conv = 1/(cf)/8766 #number of hours in a year

        # Discount rate
        # dr = data['BIC3'][r,6]
        dr = data['BIC3'][r,:, ctti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['ITTI']), int(max_lt)])
        it[:, 0, np.newaxis] =  data['BIC3'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis] *
          conv*(1*10^6)


        # Standard deviation of investment cost
        dit = np.zeros([len(titles['ITTI']), int(max_lt)])
        dit[:, 0, np.newaxis] =  data['BIC3'][r,:, ctti['2 Investment cost SD'], np.newaxis] * conv*(1*10^6)


        # Subsidies as a percentage of investment cost
        st = np.zeros([len(titles['ITTI']), int(max_lt)])
```

```
        st[:, 0, np.newaxis] = (data['BIC3'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
             * data['ISB3'][r, :, 0,np.newaxis] * conv)*(1*10^6)


        # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
        ft = np.ones([len(titles['ITTI']), int(max_lt)])
        ft = ft * data['BIC3'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
        ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['ITTI']), int(max_lt)])
        dft = dft * data['BIC3'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
        dft = np.where(mask, dft, 0)

        #fuel tax/subsidies
        #fft = np.ones([len(titles['ITTI']), int(max_lt)])
#         fft = ft * data['PG_FUELTAX'][r, :, :]
#         fft = np.where(lt_mask, ft, 0)

        # Fixed operation & maintenance cost - variable O&M available but not included
        omt = np.ones([len(titles['ITTI']), int(max_lt)])
        omt = omt * data['BIC3'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) in a year
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['ITTI']), int(max_lt)])
        domt = domt * data['BIC3'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
        domt = np.where(mask, domt, 0)



        # Net present value calculations
```

```
        # Discount rate
        denominator = (1+dr)**lt_mat

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+omt)/denominator
        # 1.3-Only policy costs
        #npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT

        lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)

        # 1.2-LCOT including policy costs
        tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
        # 1.3 LCOE excluding policy, including co2 price
        #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
        # Standard deviation of LCOT
        dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

        # LCOE augmented with gamma values, no gamma values yet
```

```
        tlcoeg = tlcoe+data['IAM3'][r, :, 0]

        # Pass to variables that are stored outside.
        data['ILC3'][r, :, 0] = lcoe            # The real bare LCOT without taxes (euros/mwh)
        #data['IHLT'][r, :, 0] = tlcoe           # The real bare LCOE with taxes
        data['ILG3'][r, :, 0] = tlcoeg         # As seen by consumer (generalised cost)
        data['ILD3'][r, :, 0] = dlcoe          # Variation on the LCOT distribution



    return data

#Final energy demand has to match IEA

# %% main function
# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
    """

    Main solution function for the module.

    Simulates investor decision making.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
        Description
```

```
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution


    """

    # Categories for the cost matrix (BIC3)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    sector = 'Metals, transport and machinery equipment'

    #Get fuel prices from E3ME and add them to the data for this code
    #Initialise everything #TODO

    #Calculate or read in FED
    #Calculate historical emissions
    data = get_lcoih(data, titles, year)

    # Endogenous calculation takes over from here
    if year > histend['IUD3']:

        # Create a local dictionary for timeloop variables
```

```
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():


            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
        isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = divide((data_dt['IWK3'][:, :, 0] - data['IRG3'][:, :, 0]),
                          data_dt['IRG3'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['IRG3'][:, :, 0] == 0.0] = 1.0
        isReg[data['IRG3'][:, :, 0] == -1.0] = 0.0


        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it
        kappa = 10 #tech substitution constant

        ############## Computing new shares #################
        IUD3tot = data['IUD3'][:, :, 0].sum(axis=1)
        #Start the computation of shares
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            #Time lagged UED plus change in UED * (no of iterations) * dt

            IUD3t = time_lag['IUD3'][:, :, 0].sum(axis=1) + (IUD3tot - time_lag['IUD3'][:, :, 0].sum(axis=1)) * t *
```

```
                    dt

                for r in range(len(titles['RTI'])):

                    if IUD3t[r] == 0.0:
                        continue



                ############################ FTT ##################################

                    # DSiK contains the change in shares
                    dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])

                    # F contains the preferences
                    F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5

                    # Market share constraints
                    Gijmax = np.ones(len(titles['ITTI']))
                    #Gijmin = np.ones((t2ti))

                    # -----------------------------------------------------
                    # Step 1: Endogenous EOL replacements
                    # -----------------------------------------------------
                    for b1 in range(len(titles['ITTI'])):

                        if  not (data_dt['IWS3'][r, b1, 0] > 0.0 and
                                 data_dt['ILG3'][r, b1, 0] != 0.0 and
                                 data_dt['ILD3'][r, b1, 0] != 0.0):
                            continue

                        #TODO: create market share constraints
                        Gijmax[b1] = np.tanh(1.25*(data_dt['ISC3'][0, b1, 0] - data_dt['IWS3'][r, b1, 0])/0.1)
```

```
                    #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)



                    S_i = data_dt['IWS3'][r, b1, 0]


                    for b2 in range(b1):

                        if  not (data_dt['IWS3'][r, b2, 0] > 0.0 and
                                 data_dt['ILG3'][r, b2, 0] != 0.0 and
                                 data_dt['ILD3'][r, b2, 0] != 0.0):
                            continue

                        S_k = data_dt['IWS3'][r,b2, 0]
                        Aik = data['IWA3'][0,b1 , b2]*kappa
                        Aki = data['IWA3'][0,b2, b1]*kappa

                        # Propagating width of variations in perceived costs
                        dFik = sqrt(2) * sqrt((data_dt['ILD3'][r, b1, 0]*data_dt['ILD3'][r, b1, 0] + data_dt['ILD3'][r, b2, 0]*data_dt['ILD3'][r, b2, 0]))

                        # Consumer preference incl. uncertainty
                        Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG3'][r, b2, 0]-data_dt['ILG3'][r, b1, 0])/dFik))

                        # Preferences are then adjusted for regulations
                        F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
                        F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])


                        #Runge-Kutta market share dynamiccs
```

```
                        k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                        k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                        k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                        k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])

                        #This method currently applies RK4 to the shares, but all other components of the equation are calculated for the overall time step
                        #We must assume the the LCOE does not change significantly in a time step dt, so we can focus on the shares.

                        dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                        dSik[b2, b1] = -dSik[b1, b2]

                        #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
                        #dSik[b2, b1] = -dSik[b1, b2]


                # -----------------------------------------------------
                # Step 3: Exogenous sales additions
                # -----------------------------------------------------
                # Add in exogenous sales figures. These are blended with endogenous result!


                # Add in exogenous sales figures. These are blended with
                # endogenous result! Note that it's different from the
                # ExogSales specification!
                Utot = IUD3t[r]
                iud_lag = time_lag['IUD3'][:, :, 0].sum(axis=1)
                dSk = np.zeros((len(titles['ITTI'])))
                dUk = np.zeros((len(titles['ITTI'])))
                dUkTK = np.zeros((len(titles['ITTI'])))
                dUkREG = np.zeros((len(titles['ITTI'])))

```

```
            # Check that exogenous share changes add to zero
            dUkTK = data['IXS3'][r, :, 0]
            if (data['IXS3'][r, :, 0].sum() > 0.0):
                dUkTK[0] = dUkTK[0] - data['IXS3'][r, :, 0].sum()

            # Correct for regulations #TODO Does this actually make sense?

            if iud_lag[r] > 0.0 and IUD3t[r] > 0.0 and (IUD3t[r] - iud_lag[r]) > 0.0:

                dUkREG = -data_dt['IUD3'][r, :, 0] * ( (IUD3t[r] - iud_lag[r]) /
                                iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])


            # Sum effect of exogenous sales additions (if any) with
            # effect of regulations
            dUk = copy.deepcopy(dUkREG)
            dUtot = np.sum(dUk)

            # Convert to market shares and make sure sum is zero
            # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
            dSk = np.divide(dUk, Utot) - time_lag['IWS3'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK


            # New market shares
            # check that market shares sum to 1
                    #print(np.sum(dSik, axis=1))
            data['IWS3'][r, :, 0] = data_dt['IWS3'][r, :, 0] + np.sum(dSik, axis=1) + dSk

            if ~np.isclose(np.sum(data['IWS3'][r, :, 0]), 1.0, atol=1e-5):
                msg = """Sector: {} - Region: {} - Year: {}
                Sum of market shares do not add to 1.0 (instead: {})
                """.format(sector, titles['RTI'][r], year, np.sum(data['IWS3'][r, :, 0]))
                warnings.warn(msg)
```

```
            if np.any(data['IWS3'][r, :, 0] < 0.0):
                msg = """Sector: {} - Region: {} - Year: {}
                Negative market shares detected! Critical error!
                """.format(sector, titles['RTI'][r], year)
                warnings.warn(msg)




        # ============================================================
        #  Update variables
        # ============================================================

        #TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change

        #Useful heat by technology, calculate based on new market shares #Regional totals
        data['IUD3'][:, :, 0] = data['IWS3'][:, :, 0]* IUD3t[:, np.newaxis]

        # Capacity by technology
        data['IWK3'][:, :, 0] = divide(data['IUD3'][:, :, 0],
                                          data['BIC3'][:, :, ctti["13 Capacity factor mean"]]*8766)
        #add number of devices replaced due to breakdowns = IWK4_lagged/lifetime to yearly capacity additions
        #note some values of IWI4 negative
        data["IWI3"][:, :, 0] = 0
        for r in range(len(titles['RTI'])):
            for tech in range(len(titles['ITTI'])):
                if(data['IWK3'][r, tech, 0]-time_lag['IWK3'][r, tech, 0]) > 0:
                    data['IWI3'][r, tech, 0] = (data['IWK3'][r, tech, 0]-time_lag['IWK3'][r, tech, 0])

        data["IWI3"][:, :, 0] = data["IWI3"][:, :, 0] + np.where(data['BIC3'][:, :, ctti['5 Lifetime (years)']] !=0.0,
                                                        divide(time_lag['IWK3'][:, :, 0],
```

```
                                                         data['BIC3'][:, :, ctti['5 Lifetime (years)']]),0.0)
                #Update emissions
                #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
                for r in range(len(titles['RTI'])):
                    data['IWE3'][r, :, 0] = data['IUD3'][r, :, 0] * data['IHW3'][0, :, 0]


                #Final energy by technology
                data['IFD3'][:, :, 0] = np.where(data['BIC3'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
                                                 divide(data['IUD3'][:, :, 0],
                                                        data['BIC3'][:, :, ctti["9 Conversion efficiency"]]),
                                                0.0)



                # ==============================================================
                # Learning-by-doing
                # ==============================================================

                # Cumulative global learning
                # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
                # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
                # techicals




                bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
                for r in range(len(titles['RTI'])):
                    bi[r,:] = np.matmul(data['IWB3'][0, :, :],data['IWI3'][r, :, 0])
                dw = np.sum(bi, axis=0)*dt

                # # Cumulative capacity incl. learning spill-over effects
```

```
            data['IWW3'][0, :, 0] = data_dt['IWW3'][0, :, 0] + dw
            #
            # # Copy over the technology cost categories that do not change (all except prices which are updated through learning-by-doing below)
            data['BIC3'] = copy.deepcopy(data_dt['BIC3'])
            #
            # # Learning-by-doing effects on investment
            for tech in range(len(titles['ITTI'])):

                if data['IWW3'][0, tech, 0] > 0.1:

                    data['BIC3'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC3'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] * \
                                                                           (1.0 + data['BIC3'][:, tech, ctti['15 Learning exponent']] * dw[tech]/data['IWW3'][0, tech, 0])

            # ===================================================================
            # Update the time-loop variables
            # ===================================================================

            #Calculate levelised cost again
            data = get_lcoih(data, titles, year)

            #Update time loop variables:
            for var in data_dt.keys():


                data_dt[var] = copy.deepcopy(data[var])


    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_nmm_main.py
=========================================
Industrial non-metallic minerals sector FTT module.
###################################################


This is the main file for FTT: Industrial Heat - NMM, which models technological
diffusion of industrial heat processes within the non-metallic minerals sector due
to simulated investor decision making. Investors compare the **levelised cost of
industrial heat**, which leads to changes in the market shares of different technologies.

The outputs of this module include changes in final energy demand and emissions due
chemical heat processes for the EU28.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros

Functions included:

    - solve
        Main solution function for the module
    - get_lcoih
        Calculates the levelised cost of industrial heat

"""
# Standard library imports
```

```python
from math import sqrt
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# --------------------------- LCOH function -----------------------------------
# -----------------------------------------------------------------------------
def get_lcoih(data, titles, year):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of industrial heat in 2019 Euros
    It includes intangible costs (gamma values) and together
    determines the investor preferences.

    Parameters
    -----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) for all
        variables. Variable names are keys and the values are 3D NumPy arrays.
    titles: dictionary
```

```
        Titles is a container of all permissible dimension titles of the model.

    Returns
    ----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) data for
        all variables.
        Variable names are keys and the values are 3D NumPy arrays.
        The values inside the container are updated and returned to the main
        routine.

    Notes
    ---------
    Additional notes if required.
    """

    # Categories for the cost matrix (BIC4)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    for r in range(len(titles['RTI'])):
        if data['IUD4'][r, :, 0].sum(axis=0)==0:
            continue

        # Cost matrix
        #BIC4 = data['BIC4'][r, :, :]

        lt = data['BIC4'][r,:, ctti['5 Lifetime (years)']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
        lt_mat = np.where(mask, lt_mat, 0)
```

```python


        # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
        cf = data['BIC4'][r,:, ctti['13 Capacity factor mean'], np.newaxis]

        #conversion efficiency
        ce = data['BIC4'][r,:, ctti['9 Conversion efficiency'], np.newaxis]

        # Trap for very low CF
        cf[cf<0.000001] = 0.000001

        # Factor to transfer cost components in terms of capacity to generation
#        ones = np.ones([len(titles['ITTI']), 1])
        conv = 1/(cf)/8766 #number of hours in a year

        # Discount rate
        # dr = data['BIC4'][r,6]
        dr = data['BIC4'][r,:, ctti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['ITTI']), int(max_lt)])
        it[:, 0, np.newaxis] =  data['BIC4'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis] * conv*(1*10^6)


        # Standard deviation of investment cost
        dit = np.zeros([len(titles['ITTI']), int(max_lt)])
        dit[:, 0, np.newaxis] =  data['BIC4'][r,:, ctti['2 Investment cost SD'], np.newaxis] * conv*(1*10^6)


        # Subsidies as a percentage of investment cost
        st = np.zeros([len(titles['ITTI']), int(max_lt)])
```

```
        st[:, 0, np.newaxis] = (data['BIC4'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
            * data['ISB4'][r, :, 0,np.newaxis] * conv)*(1*10^6)


        # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
        ft = np.ones([len(titles['ITTI']), int(max_lt)])
        ft = ft * data['BIC4'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
        ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['ITTI']), int(max_lt)])
        dft = dft * data['BIC4'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
        dft = np.where(mask, dft, 0)

        #fuel tax/subsidies
        #fft = np.ones([len(titles['ITTI']), int(max_lt)])
#        fft = ft * data['PG_FUELTAX'][r, :, :]
#        fft = np.where(lt_mask, ft, 0)

        # Fixed operation & maintenance cost - variable O&M available but not included
        omt = np.ones([len(titles['ITTI']), int(max_lt)])
        omt = omt * data['BIC4'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) in a year
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['ITTI']), int(max_lt)])
        domt = domt * data['BIC4'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
        domt = np.where(mask, domt, 0)



        # Net present value calculations
```

```
        # Discount rate
        denominator = (1+dr)**lt_mat

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+omt)/denominator
        # 1.3-Only policy costs
        #npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT

        lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)

        # 1.2-LCOT including policy costs
        tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
        # 1.3 LCOE excluding policy, including co2 price
        #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
        # Standard deviation of LCOT
        dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

        # LCOE augmented with gamma values, no gamma values yet
```

```python
        tlcoeg = tlcoe+data['IAM4'][r, :, 0]

        # Pass to variables that are stored outside.
        data['ILC4'][r, :, 0] = lcoe            # The real bare LCOT without taxes (euros/mwh)
        #data['IHLT'][r, :, 0] = tlcoe           # The real bare LCOE with taxes
        data['ILG4'][r, :, 0] = tlcoeg         # As seen by consumer (generalised cost)
        data['ILD4'][r, :, 0] = dlcoe          # Variation on the LCOT distribution



    return data

#Final energy demand has to match IEA

# %% main function
# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
    """
    Main solution function for the module.

    Simulates investor decision making.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
        Description
    titles: dictionary of lists
```

```
            Dictionary containing all title classification
        histend: dict of integers
            Final year of histrorical data by variable
        year: int
            Curernt/active year of solution
        specs: dictionary of NumPy arrays
            Function specifications for each region and module

        Returns
        ----------
        data: dictionary of NumPy arrays
            Model variables for the given year of solution


        """

        # Categories for the cost matrix (BIC4)
        ctti = {category: index for index, category in enumerate(titles['CTTI'])}

        sector = 'Non-metallic minerals'

        #Get fuel prices from E3ME and add them to the data for this code
        #Initialise everything #TODO

        #Calculate or read in FED
        #Calculate historical emissions
        data = get_lcoih(data, titles, year)

        # Endogenous calculation takes over from here
        if year > histend['IUD4']:

            # Create a local dictionary for timeloop variables
            # It contains values between timeloop interations in the FTT core
```

```
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():

            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
        isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = divide((data_dt['IWK4'][:, :, 0] - data['IRG4'][:, :, 0]),
                          data_dt['IRG4'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['IRG4'][:, :, 0] == 0.0] = 1.0
        isReg[data['IRG4'][:, :, 0] == -1.0] = 0.0


        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it
        kappa = 10 #tech substitution constant

        ############## Computing new shares #################
        IUD4tot = data['IUD4'][:, :, 0].sum(axis=1)
        #Start the computation of shares
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            #Time lagged UED plus change in UED * (no of iterations) * dt

            IUD4t = time_lag['IUD4'][:, :, 0].sum(axis=1) + (IUD4tot - time_lag['IUD4'][:, :, 0].sum(axis=1)) * t *
               dt

```

```
            for r in range(len(titles['RTI'])):

                if IUD4t[r] == 0.0:
                    continue



            ############################ FTT ##################################

                # DSiK contains the change in shares
                dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])

                # F contains the preferences
                F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5

                # Market share constraints
                Gijmax = np.ones(len(titles['ITTI']))
                #Gijmin = np.ones((t2ti))

                # -----------------------------------------------------
                # Step 1: Endogenous EOL replacements
                # -----------------------------------------------------
                for b1 in range(len(titles['ITTI'])):

                    if  not (data_dt['IWS4'][r, b1, 0] > 0.0 and
                             data_dt['ILG4'][r, b1, 0] != 0.0 and
                             data_dt['ILD4'][r, b1, 0] != 0.0):
                        continue

                    #TODO: create market share constraints
                    Gijmax[b1] = np.tanh(1.25*(data_dt['ISC4'][0, b1, 0] - data_dt['IWS4'][r, b1, 0])/0.1)
                    #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)

```

```

            S_i = data_dt['IWS4'][r, b1, 0]


            for b2 in range(b1):

                if  not (data_dt['IWS4'][r, b2, 0] > 0.0 and
                         data_dt['ILG4'][r, b2, 0] != 0.0 and
                         data_dt['ILD4'][r, b2, 0] != 0.0):
                    continue

                S_k = data_dt['IWS4'][r,b2, 0]
                Aik = data['IWA4'][0,b1 , b2]*kappa
                Aki = data['IWA4'][0,b2, b1]*kappa

                # Propagating width of variations in perceived costs
                dFik = sqrt(2) * sqrt((data_dt['ILD4'][r, b1, 0]*data_dt['ILD4'][r, b1, 0] + data_dt['ILD4'][r, b2, 0]*data_dt['ILD4'][r, b2, 0]))

                # Consumer preference incl. uncertainty
                Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG4'][r, b2, 0]-data_dt['ILG4'][r, b1, 0])/dFik))

                # Preferences are then adjusted for regulations
                F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
                F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])


                #Runge-Kutta market share dynamiccs
                k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
```

```
                        k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                        k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])

                        #This method currently applies RK4 to the shares, but all other components of the equation are calculated for the overall time step
                        #We must assume the the LCOE does not change significantly in a time step dt, so we can focus on the shares.

                        dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                        dSik[b2, b1] = -dSik[b1, b2]

                        #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
                        #dSik[b2, b1] = -dSik[b1, b2]


                # -----------------------------------------------------
                # Step 3: Exogenous sales additions
                # -----------------------------------------------------
                # Add in exogenous sales figures. These are blended with endogenous result!


                # Add in exogenous sales figures. These are blended with
                # endogenous result! Note that it's different from the
                # ExogSales specification!
                Utot = IUD4t[r]
                iud_lag = time_lag['IUD4'][:, :, 0].sum(axis=1)
                dSk = np.zeros((len(titles['ITTI'])))
                dUk = np.zeros((len(titles['ITTI'])))
                dUkTK = np.zeros((len(titles['ITTI'])))
                dUkREG = np.zeros((len(titles['ITTI'])))

                # Check that exogenous share changes add to zero
                dUkTK = data['IXS4'][r, :, 0]
```

```
                    if (data['IXS4'][r, :, 0].sum() > 0.0):
                        dUkTK[0] = dUkTK[0] - data['IXS4'][r, :, 0].sum()

                    # Correct for regulations #TODO Does this actually make sense?

                    if iud_lag[r] > 0.0 and IUD4t[r] > 0.0 and (IUD4t[r] - iud_lag[r]) > 0.0:

                        dUkREG = -data_dt['IUD4'][r, :, 0] * ( (IUD4t[r] - iud_lag[r]) /
                                        iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])


                    # Sum effect of exogenous sales additions (if any) with
                    # effect of regulations
                    dUk = copy.deepcopy(dUkREG)
                    dUtot = np.sum(dUk)

                    # Convert to market shares and make sure sum is zero
                    # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
                    dSk = np.divide(dUk, Utot) - time_lag['IWS4'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK


                    # New market shares
                    # check that market shares sum to 1
                            #print(np.sum(dSik, axis=1))
                    data['IWS4'][r, :, 0] = data_dt['IWS4'][r, :, 0] + np.sum(dSik, axis=1) + dSk

                    if ~np.isclose(np.sum(data['IWS4'][r, :, 0]), 1.0, atol=1e-5):
                        msg = """Sector: {} - Region: {} - Year: {}
                        Sum of market shares do not add to 1.0 (instead: {})
                        """.format(sector, titles['RTI'][r], year, np.sum(data['IWS4'][r, :, 0]))
                        warnings.warn(msg)

                    if np.any(data['IWS4'][r, :, 0] < 0.0):
```

```
                    msg = """Sector: {} - Region: {} - Year: {}
                    Negative market shares detected! Critical error!
                    """.format(sector, titles['RTI'][r], year)
                    warnings.warn(msg)




            # =============================================================
            #  Update variables
            # =============================================================

            #TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change

            #Useful heat by technology, calculate based on new market shares #Regional totals
            data['IUD4'][:, :, 0] = data['IWS4'][:, :, 0]* IUD4t[:, np.newaxis]

            # Capacity by technology
            data['IWK4'][:, :, 0] = divide(data['IUD4'][:, :, 0],
                                                  data['BIC4'][:, :, ctti["13 Capacity factor mean"]]*8766)
            #add number of devices replaced due to breakdowns = IWK4_lagged/lifetime to yearly capacity additions
            #note some values of IWI4 negative
            data["IWI4"][:, :, 0] = 0
            for r in range(len(titles['RTI'])):
                for tech in range(len(titles['ITTI'])):
                    if(data['IWK4'][r, tech, 0]-time_lag['IWK4'][r, tech, 0]) > 0:
                        data["IWI4"][r, tech, 0] = (data['IWK4'][r, tech, 0]-time_lag['IWK4'][r, tech, 0])
            data["IWI4"][:, :, 0] = data["IWI4"][:, :, 0] + np.where(data['BIC4'][:, :, ctti['5 Lifetime (years)']] !=0.0,
                                                                        divide(time_lag['IWK4'][:, :, 0],data['BIC4'][:, :, ctti['5 Lifetime (years)']]),
                                                                        0.0)

```

```
        #Update emissions
        #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
        for r in range(len(titles['RTI'])):
            data['IWE4'][r, :, 0] = data['IUD4'][r, :, 0] * data['IHW4'][0, :, 0]


        #Final energy by technology
        data['IFD4'][:, :, 0] = np.where(data['BIC4'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
                                         divide(data['IUD4'][:, :, 0],
                                                data['BIC4'][:, :, ctti["9 Conversion efficiency"]]),
                                        0.0)



        # =============================================================
        # Learning-by-doing
        # =============================================================

        # Cumulative global learning
        # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
        # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
        # techicals





        bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
        for r in range(len(titles['RTI'])):
            bi[r,:] = np.matmul(data['IWB4'][0, :, :],data['IWI4'][r, :, 0])
        dw = np.sum(bi, axis=0)*dt

        # # Cumulative capacity incl. learning spill-over effects
```

```
            data["IWW4"][0, :, 0] = data_dt['IWW4'][0, :, 0] + dw
            #
            # # Copy over the technology cost categories that do not change (all except prices which are updated through learning-by-doing below)
            data['BIC4'] = copy.deepcopy(data_dt['BIC4'])
            #
            # # Learning-by-doing effects on investment
            for tech in range(len(titles['ITTI'])):

                if data['IWW4'][0, tech, 0] > 0.1:

                    data['BIC4'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC4'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] * \
                                                                          (1.0 + data['BIC4'][:, tech, ctti['15 Learning exponent']] * dw[tech]/data['IWW4'][0, tech, 0])

            # ==================================================================
            # Update the time-loop variables
            # ==================================================================

            #Calculate levelised cost again
            data = get_lcoih(data, titles, year)

            #Update time loop variables:
            for var in data_dt.keys():


                data_dt[var] = copy.deepcopy(data[var])


    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_ois_main.py
=========================================
Industrial other sectors FTT module.
###################################################


This is the main file for FTT: Industrial Heat - OIS, which models technological
diffusion of industrial heat processes within the other sectors due
to simulated investor decision making. Investors compare the **levelised cost of
industrial heat**, which leads to changes in the market shares of different technologies.

The outputs of this module include changes in final energy demand and emissions due
chemical heat processes for the EU28.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros

Functions included:

    - solve
        Main solution function for the module
    - get_lcoih
        Calculates the levelised cost of industrial heat

"""
# Standard library imports
```

```python
from math import sqrt
import os
import copy
import sys
import warnings
import time

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation

# %% lcoh
# -----------------------------------------------------------------------------
# -------------------------- LCOH function ------------------------------------
# -----------------------------------------------------------------------------
def get_lcoih(data, titles, year):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of industrial heat in 2019 Euros
    It includes intangible costs (gamma values) and together
    determines the investor preferences.

    Parameters
    -----------
    data: dictionary
        Data is a container that holds all cross-sectional (of time) for all
        variables. Variable names are keys and the values are 3D NumPy arrays.
    titles: dictionary
```

```
            Titles is a container of all permissible dimension titles of the model.

        Returns
        ----------
        data: dictionary
            Data is a container that holds all cross-sectional (of time) data for
            all variables.
            Variable names are keys and the values are 3D NumPy arrays.
            The values inside the container are updated and returned to the main
            routine.

        Notes
        ---------
        Additional notes if required.
        """

        # Categories for the cost matrix (BIC5)
        ctti = {category: index for index, category in enumerate(titles['CTTI'])}

        for r in range(len(titles['RTI'])):
            if data['IUD5'][r, :, 0].sum(axis=0)==0:
                continue

            # Cost matrix
            #BIC5 = data['BIC5'][r, :, :]

            lt = data['BIC5'][r,:, ctti['5 Lifetime (years)']]

            max_lt = int(np.max(lt))
            lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
                                 num=max_lt, axis=1, endpoint=True)
            lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
            mask = lt_mat < lt_max_mat
```

```
        lt_mat = np.where(mask, lt_mat, 0)


        # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
        cf = data['BIC5'][r,:, ctti['13 Capacity factor mean'], np.newaxis]

        #conversion efficiency
        ce = data['BIC5'][r,:, ctti['9 Conversion efficiency'], np.newaxis]

        # Trap for very low CF
        cf[cf<0.000001] = 0.000001

        # Factor to transfer cost components in terms of capacity to generation
#        ones = np.ones([len(titles['ITTI']), 1])
        conv = 1/(cf)/8766 #number of hours in a year

        # Discount rate
        # dr = data['BIC5'][r,6]
        dr = data['BIC5'][r,:, ctti['8 Discount rate'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
        it = np.zeros([len(titles['ITTI']), int(max_lt)])
        it[:, 0, np.newaxis] =  data['BIC5'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis] * conv*(1*10^6)


        # Standard deviation of investment cost
        dit = np.zeros([len(titles['ITTI']), int(max_lt)])
        dit[:, 0, np.newaxis] =  data['BIC5'][r,:, ctti['2 Investment cost SD'], np.newaxis] * conv*(1*10^6)


        # Subsidies as a percentage of investment cost
```

```
        st = np.zeros([len(titles['ITTI']), int(max_lt)])
        st[:, 0, np.newaxis] = (data['BIC5'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
             * data['ISB5'][r, :, 0,np.newaxis] * conv)*(1*10^6)


        # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
        ft = np.ones([len(titles['ITTI']), int(max_lt)])
        ft = ft * data['BIC5'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
        ft = np.where(mask, ft, 0)

        # Standard deviation of fuel costs
        dft = np.ones([len(titles['ITTI']), int(max_lt)])
        dft = dft * data['BIC5'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
        dft = np.where(mask, dft, 0)

        #fuel tax/subsidies
        #fft = np.ones([len(titles['ITTI']), int(max_lt)])
#        fft = ft * data['PG_FUELTAX'][r, :, :]
#        fft = np.where(lt_mask, ft, 0)

        # Fixed operation & maintenance cost - variable O&M available but not included
        omt = np.ones([len(titles['ITTI']), int(max_lt)])
        omt = omt * data['BIC5'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) in a year
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
        domt = np.ones([len(titles['ITTI']), int(max_lt)])
        domt = domt * data['BIC5'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
        domt = np.where(mask, domt, 0)
```

```
        # Net present value calculations
        # Discount rate
        denominator = (1+dr)**lt_mat

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+st+ft+omt)/denominator
        # 1.3-Only policy costs
        #npv_expenses3 = (st+fft-fit)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT

        lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)

        # 1.2-LCOT including policy costs
        tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
        # 1.3 LCOE excluding policy, including co2 price
        #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # 1.3-LCOT of policy costs
        # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
        # Standard deviation of LCOT
        dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

```

```
        # LCOE augmented with gamma values, no gamma values yet
        tlcoeg = tlcoe+data['IAM5'][r, :, 0]

        # Pass to variables that are stored outside.
        data['ILC5'][r, :, 0] = lcoe            # The real bare LCOT without taxes (euros/mwh)
        #data['IHLT'][r, :, 0] = tlcoe           # The real bare LCOE with taxes
        data['ILG5'][r, :, 0] = tlcoeg         # As seen by consumer (generalised cost)
        data['ILD5'][r, :, 0] = dlcoe          # Variation on the LCOT distribution



    return data

#Final energy demand has to match IEA

# %% main function
# -----------------------------------------------------------------------------
# ----------------------------- Main ------------------------------------------
# -----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
    """

    Main solution function for the module.

    Simulates investor decision making.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
```

```
        Description
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution


    """

    # Categories for the cost matrix (BIC5)
    ctti = {category: index for index, category in enumerate(titles['CTTI'])}

    sector = 'Metals, transport and machinery equipment'

    #Get fuel prices from E3ME and add them to the data for this code
    #Initialise everything #TODO

    #Calculate or read in FED
    #Calculate historical emissions
    data = get_lcoih(data, titles, year)

    # Endogenous calculation takes over from here
    if year > histend['IUD5']:

```

```
        # Create a local dictionary for timeloop variables
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():

            data_dt[var] = copy.deepcopy(time_lag[var])

        # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
        isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
        division = divide((data_dt['IWK5'][:, :, 0] - data['IRG5'][:, :, 0]),
                          data_dt['IRG5'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['IRG5'][:, :, 0] == 0.0] = 1.0
        isReg[data['IRG5'][:, :, 0] == -1.0] = 0.0


        # Factor used to create quarterly data from annual figures
        no_it = 4
        dt = 1 / no_it
        kappa = 10 #tech substitution constant

        ############## Computing new shares #################
        IUD5tot = data['IUD5'][:, :, 0].sum(axis=1)
        #Start the computation of shares
        for t in range(1, no_it+1):

            # Interpolate to prevent staircase profile.
            #Time lagged UED plus change in UED * (no of iterations) * dt

            IUD5t = time_lag['IUD5'][:, :, 0].sum(axis=1) + (IUD5tot - time_lag['IUD5'][:, :, 0].sum(axis=1)) * t *
```

```
            dt

        for r in range(len(titles['RTI'])):

            if IUD5t[r] == 0.0:
                continue



        ############################ FTT ##################################

            # DSiK contains the change in shares
            dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])

            # F contains the preferences
            F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5

            # Market share constraints
            Gijmax = np.ones(len(titles['ITTI']))
            #Gijmin = np.ones((t2ti))

            # -----------------------------------------------------
            # Step 1: Endogenous EOL replacements
            # -----------------------------------------------------
            for b1 in range(len(titles['ITTI'])):

                if  not (data_dt['IWS5'][r, b1, 0] > 0.0 and
                         data_dt['ILG5'][r, b1, 0] != 0.0 and
                         data_dt['ILD5'][r, b1, 0] != 0.0):
                    continue

                #TODO: create market share constraints
                Gijmax[b1] = np.tanh(1.25*(data_dt['ISC5'][0, b1, 0] - data_dt['IWS5'][r, b1, 0])/0.1)
```

```
#Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)



S_i = data_dt['IWS5'][r, b1, 0]


for b2 in range(b1):

    if  not (data_dt['IWS5'][r, b2, 0] > 0.0 and
             data_dt['ILG5'][r, b2, 0] != 0.0 and
             data_dt['ILD5'][r, b2, 0] != 0.0):
        continue

    S_k = data_dt['IWS5'][r,b2, 0]
    Aik = data['IWA5'][0,b1 , b2]*kappa
    Aki = data['IWA5'][0,b2, b1]*kappa

    # Propagating width of variations in perceived costs
    dFik = sqrt(2) * sqrt((data_dt['ILD5'][r, b1, 0]*data_dt['ILD5'][r, b1, 0] + data_dt['ILD5'][r, b2, 0]*data_dt['ILD5'][r, b2, 0]))

    # Consumer preference incl. uncertainty
    Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG5'][r, b2, 0]-data_dt['ILG5'][r, b1, 0])/dFik))

    # Preferences are then adjusted for regulations
    F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1]) + 0.5*(isReg[r, b1]*isReg[r, b2])
    F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg[r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])


    #Runge-Kutta market share dynamiccs
```

```
                k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
                k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])

                #This method currently applies RK4 to the shares, but all other components of the equation are calculated for the overall time step
                #We must assume the the LCOE does not change significantly in a time step dt, so we can focus on the shares.

                dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                dSik[b2, b1] = -dSik[b1, b2]

                #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
                #dSik[b2, b1] = -dSik[b1, b2]


        # ------------------------------------------------------
        # Step 3: Exogenous sales additions
        # ------------------------------------------------------
        # Add in exogenous sales figures. These are blended with endogenous result!


        # Add in exogenous sales figures. These are blended with
        # endogenous result! Note that it's different from the
        # ExogSales specification!
        Utot = IUD5t[r]
        iud_lag = time_lag['IUD5'][:, :, 0].sum(axis=1)
        dSk = np.zeros((len(titles['ITTI'])))
        dUk = np.zeros((len(titles['ITTI'])))
        dUkTK = np.zeros((len(titles['ITTI'])))
        dUkREG = np.zeros((len(titles['ITTI'])))

```

```
# Check that exogenous share changes add to zero
dUkTK = data['IXS5'][r, :, 0]
if (data['IXS5'][r, :, 0].sum() > 0.0):
    dUkTK[0] = dUkTK[0] - data['IXS5'][r, :, 0].sum()

# Correct for regulations #TODO Does this actually make sense?

if iud_lag[r] > 0.0 and IUD5t[r] > 0.0 and (IUD5t[r] - iud_lag[r]) > 0.0:

    dUkREG = -data_dt['IUD5'][r, :, 0] * ( (IUD5t[r] - iud_lag[r]) /
                    iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])


# Sum effect of exogenous sales additions (if any) with
# effect of regulations
dUk = copy.deepcopy(dUkREG)
dUtot = np.sum(dUk)

# Convert to market shares and make sure sum is zero
# dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
dSk = np.divide(dUk, Utot) - time_lag['IWS5'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK


# New market shares
# check that market shares sum to 1
        #print(np.sum(dSik, axis=1))
data['IWS5'][r, :, 0] = data_dt['IWS5'][r, :, 0] + np.sum(dSik, axis=1) + dSk

if ~np.isclose(np.sum(data['IWS5'][r, :, 0]), 1.0, atol=1e-5):
    msg = """Sector: {} - Region: {} - Year: {}
    Sum of market shares do not add to 1.0 (instead: {})
    """.format(sector, titles['RTI'][r], year, np.sum(data['IWS5'][r, :, 0]))
    warnings.warn(msg)
```

```
    if np.any(data['IWS5'][r, :, 0] < 0.0):
        msg = """Sector: {} - Region: {} - Year: {}
        Negative market shares detected! Critical error!
        """.format(sector, titles['RTI'][r], year)
        warnings.warn(msg)




# ============================================================
#  Update variables
# ============================================================

#TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change

#Useful heat by technology, calculate based on new market shares #Regional totals
data['IUD5'][:, :, 0] = data['IWS5'][:, :, 0]* IUD5t[:, np.newaxis]

# Capacity by technology
data['IWK5'][:, :, 0] = divide(data['IUD5'][:, :, 0],
                                  data['BIC5'][:, :, ctti["13 Capacity factor mean"]]*8766)
#add number of devices replaced due to breakdowns = IWK4_lagged/lifetime to yearly capacity additions
#note some values of IWI4 negative
data["IWI1"][:, :, 0] = 0
for r in range(len(titles['RTI'])):
    for tech in range(len(titles['ITTI'])):
        if(data['IWK5'][r, tech, 0]-time_lag['IWK5'][r, tech, 0]) > 0:
            data['IWI5'][r, tech, 0] = (data['IWK5'][r, tech, 0]-time_lag['IWK5'][r, tech, 0])
data['IWI5'][:, :, 0] = data['IWI5'][:, :, 0] + np.where(data['BIC5'][:, :, ctti['5 Lifetime (years)']] !=0.0,
                                                            divide(time_lag['IWK5'][:, :, 0],data['BIC5'][:, :, ctti['5 Lifetime (years)']]),
```

```
                                                                                  0.0)

            #Update emissions
            #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
            for r in range(len(titles['RTI'])):
                data['IWE5'][r, :, 0] = data['IUD5'][r, :, 0] * data['IHW5'][0, :, 0]


            #Final energy by technology
            data['IFD5'][:, :, 0] = np.where(data['BIC5'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
                                             divide(data['IUD5'][:, :, 0],
                                                    data['BIC5'][:, :, ctti["9 Conversion efficiency"]]),
                                             0.0)



            # =============================================================
            # Learning-by-doing
            # =============================================================

            # Cumulative global learning
            # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
            # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
            # techicals





            bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['IWB5'][0, :, :],data['IWI5'][r, :, 0])
            dw = np.sum(bi, axis=0)*dt
```

```

            # # Cumulative capacity incl. learning spill-over effects
            data['IWW5'][0, :, 0] = data_dt['IWW5'][0, :, 0] + dw
            #
            # # Copy over the technology cost categories that do not change (all except prices which are updated through learning-by-doing below)
            data['BIC5'] = copy.deepcopy(data_dt['BIC5'])
            #
            # # Learning-by-doing effects on investment
            for tech in range(len(titles['ITTI'])):

                if data['IWW5'][0, tech, 0] > 0.1:

                    data['BIC5'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC5'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] * \
                                                                           (1.0 + data['BIC5'][:, tech, ctti['15 Learning exponent']] * dw[tech]/data['IWW5'][0, tech, 0])

            # ===================================================================
            # Update the time-loop variables
            # ===================================================================

            #Calculate levelised cost again
            data = get_lcoih(data, titles, year)

            #Update time loop variables:
            for var in data_dt.keys():

                data_dt[var] = copy.deepcopy(data[var])


    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
ftt_tr_main.py
=========================================
Passenger road transport FTT module.
####################################

This is the main file for FTT: Transport, which models technological
diffusion of passenger vehicle types due to simulated consumer decision making.
Consumers compare the **levelised cost of heat**, which leads to changes in the
market shares of different technologies.

The outputs of this module include sales, fuel use, and emissions.

Local library imports:

    Support functions:

    - `divide <divide.html>`__
        Bespoke element-wise divide which replaces divide-by-zeros with zeros
    - `estimation <econometrics_functions.html>`__
        Predict future values according to the estimated coefficients.

Functions included:
    - solve
        Main solution function for the module
    - get_lcot
        Calculate levelised cost of transport
"""

# Standard library imports
from math import sqrt
```

```python
import os
import copy
import sys
import warnings

# Third party imports
import pandas as pd
import numpy as np

# Local library imports
from support.divide import divide
from support.econometrics_functions import estimation


# %% lcot
# -----------------------------------------------------------------------------
# --------------------------- LCOT function -----------------------------------
# -----------------------------------------------------------------------------
def get_lcot(data, titles):
    """
    Calculate levelized costs.

    The function calculates the levelised cost of transport in 2012$/p-km per
    vehicle type. It includes intangible costs (gamma values) and together
    determines the investor preferences.
    """

    # Categories for the cost matrix (BTTC)
    c3ti = {category: index for index, category in enumerate(titles['C3TI'])}

    tf = np.ones([len(titles['VTTI']), 1])
    tf[12:15] = 0
    tf[18:21] = 0
```

```

    for r in range(len(titles['RTI'])):

        # Cost matrix
        bttc = data['BTTC'][r, :, :]

        # Vehicle lifetime
        lt = bttc[:, c3ti['8 lifetime']]
        max_lt = int(np.max(lt))
        lt_mat = np.linspace(np.zeros(len(titles['VTTI'])), max_lt-1,
                             num=max_lt, axis=1, endpoint=True)
        lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
        mask = lt_mat < lt_max_mat
        lt_mat = np.where(mask, lt_mat, 0)

        # Capacity factor used in decisions (constant), not actual capacity factor
        cf = bttc[:, c3ti['12 Cap_F (Mpkm/kseats-y)'], np.newaxis]

        # Discount rate
        # dr = bttc[6]
        dr = bttc[:, c3ti['7 Discount rate'], np.newaxis]

        # Occupancy rates
        ff = bttc[:, c3ti['11 occupancy rate p/sea'], np.newaxis]

        # Number of seats
        ns = bttc[:, c3ti['15 Seats/Veh'], np.newaxis]

        # Energy use
        en = bttc[:, c3ti['9 energy use (MJ/km)'], np.newaxis]

        # Initialse the levelised cost components
        # Average investment cost
```

```
        it = np.zeros([len(titles['VTTI']), int(max_lt)])
        it[:, 0, np.newaxis] = bttc[:, c3ti['1 Prices cars (USD/veh)'], np.newaxis]/ns/ff/cf/1000

        # Standard deviation of investment cost
        dit = np.zeros([len(titles['VTTI']), int(max_lt)])
        dit[:, 0, np.newaxis] = bttc[:, c3ti['2 Std of price'], np.newaxis]/ns/ff/cf/1000

        # Vehicle tax at purchase
        vtt = np.zeros([len(titles['VTTI']), int(max_lt)])
        vtt[:, 0, np.newaxis] = (data['TTVT'][r, :, 0, np.newaxis]+data['RTCO'][r, 0, 0]*bttc[:,c3ti['14 CO2Emissions'], np.newaxis])/ns/ff/cf/1000

        # Average fuel costs
        ft = np.ones([len(titles['VTTI']), int(max_lt)])
        ft = ft * bttc[:,c3ti['3 fuel cost (USD/km)'], np.newaxis]/ns/ff
        ft = np.where(mask, ft, 0)

        # Stadard deviation of fuel costs
        dft = np.ones([len(titles['VTTI']), int(max_lt)])
        dft = dft * bttc[:, c3ti['4 std fuel cost'], np.newaxis]
        dft = np.where(mask, dft, 0)

        # Fuel tax costs
        fft = np.ones([len(titles['VTTI']), int(max_lt)])
        fft = fft * data['RTFT'][r, 0, 0]*en/ns/ff*tf
        fft = np.where(mask, fft, 0)

        # Average operation & maintenance cost
        omt = np.ones([len(titles['VTTI']), int(max_lt)])
        omt = omt * bttc[:, c3ti['5 O&M costs (USD/km)'], np.newaxis]/ns/ff
        omt = np.where(mask, omt, 0)

        # Standard deviation of operation & maintenance cost
```

```
        domt = np.ones([len(titles['VTTI']), int(max_lt)])
        domt = omt * bttc[:, c3ti['6 std O&M'], np.newaxis]/ns
        domt = np.where(mask, domt, 0)

        # Road tax cost
        rtt = np.ones([len(titles['VTTI']), int(max_lt)])
        rtt = rtt * data['TTRT'][r, :, 0, np.newaxis]/cf/ns/ff/1000
        rtt = np.where(mask, rtt, 0)

        # Net present value calculations
        # Discount rate
        denominator = (1+dr)**lt_mat

        # 1-Expenses
        # 1.1-Without policy costs
        npv_expenses1 = (it+ft+omt)/denominator
        # 1.2-With policy costs
        npv_expenses2 = (it+vtt+ft+fft+omt+rtt)/denominator
        # 1.3-Only policy costs
        npv_expenses3 = (vtt+fft+rtt)/denominator
        # 2-Utility
        npv_utility = 1/denominator
        #Remove 1s for tech with small lifetime than max
        npv_utility[npv_utility==1] = 0
        npv_utility[:,0] = 1
        # 3-Standard deviation (propagation of error)
        npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator

        # 1-levelised cost variants in $/pkm
        # 1.1-Bare LCOT
        lcot = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
        # 1.2-LCOT including policy costs
        tlcot = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)
```

```
        # 1.3-LCOT of policy costs
        lcot_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
        # Standard deviation of LCOT
        dlcot = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)

        # LCOT augmented with non-pecuniary costs
        tlcotg = tlcot*(1+data['TGAM'][r, :, 0])

        # Transform into lognormal space
        logtlcot = np.log(tlcot*tlcot/np.sqrt(dlcot*dlcot + tlcot*tlcot)) + data['TGAM'][r, :, 0]
        dlogtlcot = np.sqrt(np.log10(1.0 + dlcot*dlcot/(tlcot*tlcot)))

        # Pass to variables that are stored outside.
        data['TEWC'][r, :, 0] = lcot            # The real bare LCOT without taxes
        data['TETC'][r, :, 0] = tlcot           # The real bare LCOE with taxes
        data['TEGC'][r, :, 0] = tlcotg         # As seen by consumer (generalised cost)
        data['TELC'][r, :, 0] = logtlcot      # In lognormal space
        data['TECD'][r, :, 0] = dlcot          # Variation on the LCOT distribution
        data['TLCD'][r, :, 0] = dlogtlcot     # Log variation on the LCOT distribution

    return data
# %% Fleet size - under development
# -----------------------------------------------------------------------------
# ----------------- Gompertz equation for fleet size --------------------------
# -----------------------------------------------------------------------------
# def fleet_size(data, titles):
#
#     return print("Hello")


# %% survival function
# -----------------------------------------------------------------------------
# -------------------------- Survival calcultion ------------------------------
```

```python
# ----------------------------------------------------------------------------
def survival_function(data, time_lag, histend, year, titles):
    # In this function we calculate scrappage, sales, tracking of age, and
    # average efficiency.
    # Categories for the cost matrix (BTTC)
    c3ti = {category: index for index, category in enumerate(titles['C3TI'])}


    # Create a generic matrix of fleet-stock by age
    # Assume uniform distribution, but only do so when we still have historical
    # market share data. Afterwards it becomes endogeous
    if year < histend['TEWS']:

        # TODO: This needs to be replaced with actual data
        correction = np.linspace(1/(3*len(titles['VYTI'])), 3/len(titles['VYTI']), len(titles['VYTI'])) * 0.6

        for age in range(len(titles['VYTI'])):

            data['RLTA'][:, :, age] = correction[age] *  data['TEWK'][:, :, 0]

    else:
        # Once we start to calculate the market shares and total fleet sizes
        # endogenously, we can update the vehicle stock by age matrix and
        # calculate scrappage, sales, average age, and average efficiency.
        for r in range(len(titles['RTI'])):

            for veh in range(len(titles['VTTI'])):

                # Move all vehicles one year up:
                # New sales will get added to the age-tracking matrix in the main
                # routine.
                data['RLTA'][r, veh, :-1] = copy.deepcopy(time_lag['RLTA'][r, veh, 1:])

```

```
                    # Current age-tracking matrix:
                    # Only retain the fleet that survives
                    data['RLTA'][r, veh, :] = data['RLTA'][r, veh, :] * data['TESF'][r, 0, :]

                    # Total amount of vehicles that survive:
                    survival = np.sum(data['RLTA'][r, veh, :])

                    # EoL scrappage: previous year's stock minus what survived
                    if time_lag['TEWK'][r, veh, 0] > survival:

                        data['REVS'][r, veh, 0] = time_lag['TEWK'][r, veh, 0] - survival

                    elif time_lag['TEWK'][r, veh, 0] < survival:
                        if year > 2016:
                            msg = """
                            Erronous outcome!
                            Check year {}, region {} - {}, vehicle {} - {}
                            Vehicles that survived are greater than what was in the fleet before:
                            {} versus {}
                            """.format(year, r, titles['RTI'][r], veh,
                                       titles['VTTI'][veh], time_lag['TEWK'][r, veh, 0], survival)
#                            print(msg)

    # calculate fleet size
    return data

# %% main function
# ----------------------------------------------------------------------------
# ------------------------------ Main ----------------------------------------
# ----------------------------------------------------------------------------
def solve(data, time_lag, iter_lag, titles, histend, year, specs):
    """
    Main solution function for the module.
```

```

    Add an extended description in the future.

    Parameters
    -----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution
    time_lag: type
        Description
    iter_lag: type
        Description
    titles: dictionary of lists
        Dictionary containing all title classification
    histend: dict of integers
        Final year of histrorical data by variable
    year: int
        Curernt/active year of solution
    specs: dictionary of NumPy arrays
        Function specifications for each region and module

    Returns
    ----------
    data: dictionary of NumPy arrays
        Model variables for the given year of solution

    Notes
    ---------
    This function should be broken up into more elements in development.
    """
    # Categories for the cost matrix (BTTC)
    c3ti = {category: index for index, category in enumerate(titles['C3TI'])}
    jti = {category: index for index, category in enumerate(titles['JTI'])}

```

```
    fuelvars = ['FR_1', 'FR_2', 'FR_3', 'FR_4', 'FR_5', 'FR_6',
                'FR_7', 'FR_8', 'FR_9', 'FR_10', 'FR_11', 'FR_12']

    sector = "tr_road_pass"
    sector_index = 0
    sector_index = 15 #titles['FUTI'].index('16 Road Transport')

    # Store fuel prices and convert to $2013/toe
    # It's actually in current$/toe
    # TODO: Temporary deflator values
    data['TE3P'][:, jti["5 Middle distillates"], 0] = iter_lag['PFRM'][:, sector_index, 0] / 1.33
    data['TE3P'][:, jti["7 Natural gas"], 0] = iter_lag['PFRG'][:, sector_index, 0] / 1.33
    data['TE3P'][:, jti["8 Electricity"], 0] = iter_lag['PFRE'][:, sector_index, 0] / 1.33
    data['TE3P'][:, jti["11 Biofuels"], 0] = iter_lag['PFRB'][:, sector_index, 0] / 1.33
#    data['TE3P'][:, "12 Hydrogen", 0] = data['PFRE'][:, sector_index, 0] * 2.0
    # %% First initialise if necessary


    # Up to the last year of historical market share data
    if year <= histend['TEWS']:

        for r in range(len(titles['RTI'])):

            # CORRECTION TO MARKET SHARES
            # Sometimes historical market shares do not add up to 1.0
            if (~np.isclose(np.sum(data['TEWS'][r, :, 0]), 0.0, atol=1e-9)
                    and np.sum(data['TEWS'][r, :, 0]) > 0.0):
                data['TEWS'][r, :, 0] = np.divide(data['TEWS'][r, :, 0],
                                                  np.sum(data['TEWS'][r, :, 0]))

        # Computes initial values for the capacity factor, numbers of
        # vehicles by technology and distance driven
        # "Capacity factor", defined as km driven per vehicle
```

```
        #data['TEWL'][:, :, 0] = data['RVKM'][:, 0, 0, np.newaxis]

        # "Capacities", defined as 1000 vehicles
        data['TEWK'][:, :, 0] = data['TEWS'][:, :, 0] * data['RFLT'][:, 0, 0, np.newaxis]

        # "Generation", defined as total km driven
        data['TEWG'][:, :, 0] = data['TEWK'][:, :, 0] * data['RVKM'][:, 0, 0, np.newaxis] * 1e-3

        # Call the survival function routine.
        data = survival_function(data, time_lag, histend, year, titles)

        if year == histend['TEWS']:

            # Sales are the difference between fleet sizes and the addition of scrapped vehicles
            data['TEWI'][:, :, 0] = data['TEWK'][:, :, 0] - time_lag['TEWK'][:, :, 0] + data['REVS'][:, :, 0]

            # Corrections to sales and EOL when sales are negative.
            condition = data['TEWI'][:, :, 0] < 0.0
            data['REVS'][:, :, 0] = np.where(condition,
                                             data['REVS'][:, :, 0] - data['TEWI'][:, :, 0],
                                             data['REVS'][:, :, 0])

            data['TEWI'][:, :, 0] = np.where(condition,
                                             0.0,
                                             data['TEWI'][:, :, 0])


            # Add sales to the age tracking matrix
            data['RLTA'][:, :, -1] = data['TEWI'][:, :, 0]

            # Local variable to calculate age related weights:
            share_by_age = np.zeros_like(data['RLTA'][:, :, :])
            # Average vehicle age and average relative efficiency factor
```

```
            for a, age in enumerate(titles['VYTI']):

                share_by_age[:, :, a] = divide(data['RLTA'][:, :, a],
                                               np.sum(data['RLTA'][:, :, :], axis=2))
            # Age (NOT USED)
            #data['TP_VAGE'][:, :, 0] = np.sum(share_by_age * np.asarray(titles['VYTI']), axis=2)

            # Efficiency factor #is this TETH?
            data['TEFF'][:, :, 0] = np.sum(share_by_age * data['TETH'][0, 0, :], axis=2)

            # Fuel use
            # Compute fuel use as distance driven times energy use, corrected by the biofuel mandate.
            emis_corr = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
            fuel_converter = np.zeros([len(titles['VTTI']), len(titles['JTI'])])
            fuel_converter = copy.deepcopy(data['TJET'][0, :, :])

            for r in range(len(titles['RTI'])):

                for fuel in range(len(titles['JTI'])):

                    for veh in range(len(titles['VTTI'])):

                        if titles['JTI'][fuel] == '5 Middle distillates' and data['TJET'][0, veh, fuel] ==1:  # Middle distillates

                            # Mix with biofuels if there's a biofuel mandate
                            fuel_converter[veh, fuel] = fuel_converter[veh, fuel] * (1.0 - data['RBFM'][r, 0, 0])

                            # Emission correction factor
                            emis_corr[r, veh] = 1.0 - data['RBFM'][r, 0, 0]

                        elif titles['JTI'][fuel] == '11 Biofuels'  and data['TJET'][0, veh, fuel] == 1:

```

```
                            fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * data['RBFM'][r, 0, 0]
                    #TODO this is broken
                    # Calculate fuel use - passenger car only! Therefore this will
                    # differ from E3ME results
                    # TEWG:                          km driven
                    # Convert energy unit (1/41.868) ktoe/MJ
                    # Energy demand (BBTC)           MJ/km

                    data['TJEF'][r, :, 0] = (np.matmul(np.transpose(fuel_converter), data['TEWG'][r, :, 0]*\
                                        data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']]*data['TEFF'][r, :,
                            0]/41.868))


                    data['TVFP'][r, :, 0] = (np.matmul(fuel_converter, data['TE3P'][r, :, 0]))*\
                                          data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] * \
                                                      data['TEFF'][r, :, 0]/ 41868

                # Emissions, unit MtCO2 - using vehicle emission factors, corrected by the biofuel mandate
                data['TEWE'][:, :, 0] = data['TEWG'][:, :, 0] * \
                                        data['BTTC'][:, :, c3ti['14 CO2Emissions']] * \
                                        data['TEFF'][:, :, 0] * 1e-6 * 1.2 * emis_corr

                #(NOT USED)
                #data['TP_VEC'][:, :, 0] = copy.deepcopy(data['BTTC'][:, :, c3ti['3 fuel cost (USD/km)']])

            # Calculate the LCOT for each vehicle type.
            # Call the function
            data = get_lcot(data, titles)

        # %% Simulation of stock and energy specs

        if year > histend['TEWS']:
            # TODO: Implement survival function to get a more accurate depiction of
```

```
        # vehicles being phased out and to be able to track the age of the fleet.
        # This means that a new variable will need to be implemented which is
        # basically TP_VFLT with a third dimension (vehicle age in years- up to 23y)
        # Reduced efficiences can then be tracked properly as well.

        # Create a local dictionary for timeloop variables
        # It contains values between timeloop interations in the FTT core
        data_dt = {}

        # First, fill the time loop variables with the their lagged equivalents
        for var in time_lag.keys():

            data_dt[var] = copy.deepcopy(time_lag[var])

        data_dt['TWIY'] = np.zeros([len(titles['RTI']), len(titles['VTTI']), 1])

        #Create the regulation variable
        isReg = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
        division = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
        division = divide((data_dt['TEWS'][:, :, 0] - data['TREG'][:, :, 0]),
                           data_dt['TREG'][:, :, 0])
        isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
        isReg[data['TREG'][:, :, 0] == 0.0] = 1.0
        isReg[data['TREG'][:, :, 0] == -1.0] = 0.0

        # Call the survival function routine.
        data = survival_function(data, time_lag, histend, year, titles)

        # Total number of scrapped vehicles: #TP_TEOL changed to RVTS #Is this really needed?
        #data['RVTS'][:, 0, 0] = np.sum(data['REVS'][:, :, 0], axis=1)

        # Factor used to create quarterly data from annual figures
        no_it = 4
```

```
        dt = 1 / no_it

        ############## Computing new shares ##################

        #Start the computation of shares
        for t in range(1, no_it+1):

            # Both rvkm and RFLT are exogenous at the moment
            # Interpolate to prevent staircase profile.
            rvkmt = time_lag['RVKM'][:, 0, 0] + (data['RVKM'][:, 0, 0] - time_lag['RVKM'][:, 0, 0]) * t * dt
            rfltt = time_lag['RFLT'][:, 0, 0] + (data['RFLT'][:, 0, 0] - time_lag['RFLT'][:, 0, 0]) * t * dt

            for r in range(len(titles['RTI'])):

                if rfltt[r] == 0.0:
                    continue

                ############################ FTT ###################################
                # Initialise variables related to market share dynamics
                # DSiK contains the change in shares
                dSik = np.zeros([len(titles['VTTI']), len(titles['VTTI'])])

                # F contains the preferences
                F = np.ones([len(titles['VTTI']), len(titles['VTTI'])])*0.5

#                     if int(data['TDA1'][r]) < year:

                # TODO: Check Specs dimensions
                #if np.any(specs[sector][r, :] == 1):  # FTT Specification

                for v1 in range(len(titles['VTTI'])):

                    if not (data_dt['TEWS'][r, v1, 0] > 0.0 and
```

```
                    data_dt['TELC'][r, v1, 0] != 0.0 and
                    data_dt['TLCD'][r, v1, 0] != 0.0 and
                    data['TWSA'][r, v1, 0] < 0.0):
                continue

            S_veh_i = data_dt['TEWS'][r, v1, 0]
            Aki = 0.5 * data['REVS'][r, v1, 0] / time_lag['TEWK'][r, v1, 0]

            for v2 in range(v1):

                if not (data_dt['TEWS'][r, v2, 0] > 0.0 and
                        data_dt['TELC'][r, v2, 0] != 0.0 and
                        data_dt['TLCD'][r, v2, 0] != 0.0 and
                        data['TWSA'][r, v2, 0] < 0.0):
                    continue

                S_veh_k = data_dt['TEWS'][r, v2, 0]
                Aik = 0.5 * data['REVS'][r, v2, 0] / time_lag['TEWK'][r, v2, 0] #Not using TWEA?

                # Propagating width of variations in perceived costs
                dFik = sqrt(2) * sqrt((data_dt['TLCD'][r, v1, 0]*data_dt['TLCD'][r, v1, 0] + data_dt['TLCD'][r, v2, 0]*data_dt['TLCD'][r, v2, 0]))

                # Consumer preference incl. uncertainty
                Fik = 0.5*(1+np.tanh(1.25*(data_dt['TELC'][r, v2, 0]-data_dt['TELC'][r, v1, 0])/dFik))

                # Preferences are then adjusted for regulations
                F[v1, v2] = Fik*(1.0-isReg[r, v1]) * (1.0 - isReg[r, v2]) + isReg[r, v2]*(1.0-isReg[r, v1]) + 0.5*(isReg[r, v1]*isReg[r, v2])
                F[v2, v1] = (1.0-Fik)*(1.0-isReg[r, v2]) * (1.0 - isReg[r, v1]) + isReg[r, v1]*(1.0-isReg[r, v2]) + 0.5*(isReg[r, v2]*isReg[r, v1])

                #Runge-Kutta market share dynamiccs
```

```
                        k_1 = S_veh_i*S_veh_k* (Aik*F[v1,v2] - Aki*F[v2,v1])
                        k_2 = (S_veh_i+dt*k_1/2)*(S_veh_k-dt*k_1/2)* (Aik*F[v1,v2] - Aki*F[v2,v1])
                        k_3 = (S_veh_i+dt*k_2/2)*(S_veh_k-dt*k_2/2) * (Aik*F[v1,v2] - Aki*F[v2,v1])
                        k_4 = (S_veh_i+dt*k_3)*(S_veh_k-dt*k_3) * (Aik*F[v1,v2] - Aki*F[v2,v1])

                        # Market share dynamics
                        #dSik[v1, v2] = S_veh_i*S_veh_k* (Aik*F[v1,v2] - Aki*F[v2,v1])*dt
                        dSik[v1, v2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
                        dSik[v2, v1] = -dSik[v1, v2]

                # Add in exogenous sales figures. These are blended with
                # endogenous result! Note that it's different from the
                # ExogSales specification!
                Utot = rfltt[r]
                dSk = np.zeros([len(titles['VTTI'])])
                dUk = np.zeros([len(titles['VTTI'])])
                dUkTK = np.zeros([len(titles['VTTI'])])
                dUkREG = np.zeros([len(titles['VTTI'])])

                # PV: Added a term to check that exogenous capacity is smaller than rgulated capacity.
                # Regulations have priority over exogenous capacity
                reg_vs_exog = ((data['TWSA'][r, :, 0] + data_dt['TEWK'][r, :, 0]) > data['TREG'][r, :, 0]) & (data['TREG'][r, :, 0] >= 0.0)
                data['TWSA'][r, :, 0] = np.where(reg_vs_exog, 0.0, data['TWSA'][r, :, 0])
                TWSA_scalar = 1.0

                # Check that exogenous sales additions aren't too large
                # As a proxy it can't be greater than 80% of the fleet size
                # divided by 13 (the average lifetime of vehicles)
                if (data['TWSA'][r, :, 0].sum() > 0.8 * rfltt[r] / 13):

                    TWSA_scalar = data['TWSA'][r, :, 0].sum() / (0.8 * rfltt[r] / 13)

```

```
                TWSA_gt_null = data['TWSA'][r, :, 0] >= 0.0
                dUkTK = np.where(TWSA_gt_null, data['TWSA'][r, :, 0] * TWSA_scalar, 0.0)

                # Correct for regulations
                #Share of UED * change in UED * isReg i.e. change in UED split into technologies times isReg
                if time_lag['RFLT'][r, 0, 0] > 0.0 and rfltt[r] > 0.0 and (rfltt[r] - time_lag['RFLT'][r, 0, 0]) > 0.0:

                    dUkREG = data_dt['TEWK'][r, :, 0] * ( (rfltt[r] - time_lag['RFLT'][r, 0, 0]) /
                                 time_lag['RFLT'][r, 0, 0]) * isReg[r, :].reshape([len(titles['VTTI'])])
                # Sum effect of exogenous sales additions (if any) with
                # effect of regulations
                dUk = dUkTK + dUkREG
                dUtot = np.sum(dUk)
                # Convert to market shares and make sure sum is zero
                # dSk = dUk/Utot - Uk dUtot/Utot^2  (Chain derivative)
                dSk = np.divide(dUk, Utot) - data_dt['TEWK'][r, :, 0]*np.divide(dUtot, (Utot*Utot))
#                        soel = np.sum(dSik, axis=1)
#                        st_1 = data_dt['TEWS'][r, :, 0]

                # New market shares

#                        print(np.sum(dSik, axis=1))
                data['TEWS'][r, :, 0] = data_dt['TEWS'][r, :, 0] + np.sum(dSik, axis=1) + dSk

                if ~np.isclose(np.sum(data['TEWS'][r, :, 0]), 1.0, atol=1e-3):
                    msg = """Sector: {} - Region: {} - Year: {}
                    Sum of market shares do not add to 1.0 (instead: {})
                    """.format(sector, titles['RTI'][r], year, np.sum(data['TEWS'][r, :, 0]))
                    warnings.warn(msg)

                if np.any(data['TEWS'][r, :, 0] < 0.0):
                    msg = """Sector: {} - Region: {} - Year: {}
```

```
                Negative market shares detected! Critical error!
                """.format(sector, titles['RTI'][r], year)
                warnings.warn(msg)



        ############## Update variables ##################

        # Update demand for driving (in km/ veh/ y) - exogenous atm
        #data['TEWL'][:, :, 0] = rvkmt[:, 0, 0]
        # Vehicle composition
        data['TEWK'][:, :, 0] = data['TEWS'][:, :, 0] * rfltt[:, np.newaxis]
        # Total distance driven per vehicle type
        data['TEWG'][:, :, 0] = data['TEWK'][:, :, 0] * rvkmt[:, np.newaxis] * 1e-3

        # Sales are the difference between fleet sizes and the addition of scrapped vehicles
        data['TEWI'][:, :, 0] = data['TEWK'][:, :, 0] - time_lag['TEWK'][:, :, 0] + data['REVS'][:, :, 0]

        # Corrections to sales and EOL when sales are negative.
        condition = data['TEWI'][:, :, 0] < 0.0
        data['REVS'][:, :, 0] = np.where(condition,
                                           data['REVS'][:, :, 0] - data['TEWI'][:, :, 0],
                                           data['REVS'][:, :, 0])

        data['TEWI'][:, :, 0] = np.where(condition,
                                           0.0,
                                           data['TEWI'][:, :, 0])


        # Add sales to the age tracking matrix #Changing TP_VFLTA to RLTA
        data['RLTA'][:, :, -1] = data['TEWI'][:, :, 0]

        # Local variable to calculate age related weights:
```

```
            share_by_age = np.zeros_like(data['RLTA'][:, :, :])
            # Average vehicle age and average relative efficiency factor
            for a, age in enumerate(titles['VYTI']):

                share_by_age[:, :, a] = divide(data['RLTA'][:, :, a],
                                               np.sum(data['RLTA'][:, :, :], axis=2))
            # Age (NOT USED)
            #data['TP_VAGE'][:, :, 0] = np.sum(share_by_age * np.asarray(titles['VYTI']), axis=2)

            # Efficiency factor #TETH is age by region, this changes it to region by technology
            data['TEFF'][:, :, 0] = np.sum(share_by_age * data['TETH'][0, 0, :], axis=2)


            # Fuel use
            # Compute fuel use as distance driven times energy use, corrected by the biofuel mandate.
            emis_corr = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
            fuel_converter = copy.deepcopy(data['TJET'][0, :, :])
            for r in range(len(titles['RTI'])):


                for fuel in range(len(titles['JTI'])):

                    for veh in range(len(titles['VTTI'])):

                        if titles['JTI'][fuel] == '5 Middle distillates' and data['TJET'][0, veh, fuel] ==1:  # Middle distillates

                            # Mix with biofuels if there's a biofuel mandate
                            fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * (1.0 - data['RBFM'][r, 0, 0])

                            # Emission correction factor
                            emis_corr[r, veh] = 1.0 - data['RBFM'][r, 0, 0]

```

```
                        elif titles['JTI'][fuel] == '11 Biofuels'  and data['TJET'][0, veh, fuel] == 1:

                            fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * data['RBFM'][r, 0, 0]

                #TODO this is broken

                # Calculate fuel use - passenger car only! Therefore this will
                # differ from E3ME results
                # TP_FD: (TEWG)                                     ktoe
                # TP_VTDD:                                     km/veh
                # Convert energy unit (1/41.868) ktoe/MJ
                # Energy demand (BBTC)            MJ/km
                # data['TJEF'][r, :, 0] = np.matmul(data['TEWG'][r, :, 0] *
                #                                    data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] *
                #                                    data['TEFF'][r, :, 0],
                #                                    fuel_converter) / 41.868
                #
                # data['TVFP'][r, :, 0] = np.matmul(data['TE3P'][r, :, 0],
                #                                       fuel_converter.T) / 41868 *\
                #                                       data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] * \
                #                                       data['TEFF'][r, :, 0]

            # Emissions, unit MtCO2 - using vehicle emission factors, corrected by the biofuel mandate
            data['TEWE'][:, :, 0] = data['TEWG'][:, :, 0] * \
                                    data['BTTC'][:, :, c3ti['14 CO2Emissions']] * \
                                    data['TEFF'][:, :, 0] * 1e-6 * 1.2 * emis_corr

            # Since there's no dynamic fuel price calculation yet, keep it constant (NO USED)
            #data['TP_VEC'][:, :, 0] = copy.deepcopy(data['BTTC'][:, :, c3ti['3 fuel cost (USD/km)']])

        ############## Learning-by-doing ##################

            # Cumulative global learning
```

```
            # Using a technological spill-over matrix (TEWB) together with capacity
            # additions (TEWI) we can estimate total global spillover of similar
            # vehicals
#            bi = np.matmul(data['TEWI'][:, :, 0], data['TEWB'][0, :, :])
#            dw = np.sum(bi, axis=0)*dt

            bi = np.zeros((len(titles['RTI']),len(titles['VTTI'])))
            for r in range(len(titles['RTI'])):
                bi[r,:] = np.matmul(data['TEWB'][0, :, :],data['TEWI'][r, :, 0])
            dw = np.sum(bi, axis=0)*dt

            # Cumulative capacity incl. learning spill-over effects
            data['TEWW'][0, :, 0] = data_dt['TEWW'][0, :, 0] + dw

            # Copy over the technology cost categories that do not change (all except prices which are updated through learning-by-doing below)
            data['BTTC'] = copy.deepcopy(data_dt['BTTC'])

            # Learning-by-doing effects on investment
            for veh in range(len(titles['VTTI'])):

                if data['TEWW'][0, veh, 0] > 0.1:

                    data['BTTC'][:, veh, c3ti['1 Prices cars (USD/veh)']] = data_dt['BTTC'][:, veh, c3ti['1 Prices cars (USD/veh)']] * \
                                                                             (1.0 + data['BTTC'][:, veh, c3ti['16 Learning exponent']] * dw[veh]/data['TEWW'][0, veh, 0])

            # Investment in terms of car purchases:
            for r in range(len(titles['RTI'])):

                data['TWIY'][r, :, 0] = data_dt['TWIY'][r, :, 0] + data['TEWI'][r, :, 0]*dt*data['BTTC'][r, :, c3ti['1 Prices cars (USD/veh)']]/1.33
```

```


        ############## Final output ##################

            # Get LCOT
            if t ==1:
                data = get_lcot(data, titles)

            # Update time loop variables:
            for var in data_dt.keys():

                data_dt[var] = copy.deepcopy(data[var])

    return data

```

```python
# -*- coding: utf-8 -*-
"""
=========================================
Multiple benefits.py
=========================================
Quntification of multiple benefits of energy efficiency.
######################################

This is the main file for the quantification of the multiple benefits of
energy efficiency, resulting from the implementation of different policy options
to enhance energy efficiency in Europe. The program processes results from
E3ME LITE and performs off-model calculations for the quantification of beneits
that are not estimated within E3ME lite.

This program performs the following tasks:
    1) Reads mock data from E3ME LITE for the Baseline and the selected Scenario
    2) Processes mock data from E3ME LITE to convert them to the most suitable format
    3) Estimates multiple benefits of energy efficiency that are not covered by E3ME LITE
    4) Calculates absolute and percentage differences from the Baseline
    5) Writes the final results to pickle files

Multiple benefits quantified in this program include:

Air pollution & Emissions, Air pollution Damage Costs, Employment,
Energy Cost Impact, Energy Intensity, Fossil Fuel Consumption, Fuel imports,
GDP, Gross Value Added, International competitiveness, Labour Productivity.
Material Use, Public budget as share of GDP, Share of energy consumption,
Water used in electricity generation.

@author: od
"""


```

```
# Standard library imports
import pandas as pd
import numpy as np
from celib import DB1
import copy
import pickle
import os
import glob
from pathlib import Path
import time

# Record start time of the program
start = time.time()

directory = 'J:\Projects\DG Research\REFEREE (P1451)\Multiple benefits quantification\Python'
path = 'J:\Projects\DG Research\REFEREE (P1451)\Deliverables\FTT and E3ME info\E3ME export\Final_Workbooks'
names = ['Baseline','Scenario']
add = 'Country'

# Relevant variables to call at the data processing stage
aggregate_var_loc_1 = 550
gva_loc_1 = 20
gva_loc_2 = 479
employment_loc_1 = 92
employment_loc_2 = 407
enercon_loc_1 = 164
enercon_loc_2 = 380
price_loc_1 = 191
price_loc_2 = 353
expenditure_loc_1 = 218
expenditure_loc_2 = 307
energy_prices_loc_1 = 433
energy_prices_loc_2 = 3
```

```
output_loc_1 = 289
output_loc_2 = 210
generation_loc_1 = 263
generation_loc_2 = 282
material_loc_1 = 9
material_loc_2 = 7
imports_loc_1 = 361
imports_loc_2 = 138
coeff_loc_1 = 4
coeff_loc_2 = 5
lista_loc_1 = 20
lista_loc_2 = 479
deflating_factor = 0.929440948/0.874115426 # Source: GDP deflator World Bank

cols_mat = ['Country','Indicator', 2021, 2022, 2023, 2024, 2025, 2026,2027,
                2028,2029,2030,2031,2032,2033,2034,2035,2036,2037,2038,2039,
                2040,2041,2042, 2043,2044,2045,2046,2047,2048,2049,2050,2051,
                2052,2053,2054,2055,2056,2057,2058,2059,2060,2061,2062,2063,
                2064,2065,2066,2067,2068,2069,2070]

cols_ = cols_final = ['Country','Indicator', 2022, 2023, 2024, 2025, 2026,2027,
                2028,2029,2030,2031,2032,2033,2034,2035,2036,2037,2038,2039,
                2040,2041,2042, 2043,2044,2045,2046,2047,2048,2049,2050,2051,
                2052,2053,2054,2055,2056,2057,2058,2059,2060,2061,2062,2063,
                2064,2065,2066,2067,2068,2069,2070]
cols_final = ['Country','Indicator','Level of Disaggregation', 2022, 2023, 2024, 2025, 2026,2027,
                2028,2029,2030,2031,2032,2033,2034,2035,2036,2037,2038,2039,
                2040,2041,2042, 2043,2044,2045,2046,2047,2048,2049,2050,2051,
                2052,2053,2054,2055,2056,2057,2058,2059,2060,2061,2062,2063,
                2064,2065,2066,2067,2068,2069,2070]
cols_final_ = ['Pillar','Indicator','Level of Disaggregation', 'Country', 'Unit',
               2022, 2023, 2024, 2025, 2026,2027,
                2028,2029,2030,2031,2032,2033,2034,2035,2036,2037,2038,2039,
```

```
                 2040,2041,2042, 2043,2044,2045,2046,2047,2048,2049,2050,2051,
                 2052,2053,2054,2055,2056,2057,2058,2059,2060,2061,2062,2063,
                 2064,2065,2066,2067,2068,2069,2070]
dicto = {0:'Electricity', 1: 'Gas', 2: 'Coal', 3: 'Petrol', 4: 'Oil'}
pillars = {'Gross Value Added':'Industrial Productivity',
           'Energy Intensity':'Industrial Productivity',
         'Energy Cost Impact': 'Industrial Productivity',
         'International competitiveness':'Industrial Productivity',
         'Labour Productivity':'Industrial Productivity',
         'GDP': 'Socioeconomic Development',
         'Employment':'Socioeconomic Development',
         'Public budget as share of GDP': 'Socioeconomic Development',
         'Price' : 'Socioeconomic Development',
         'Consumer Expenditure' : 'Socioeconomic Development',
         'Share of energy consumption_Q1' : 'Socioeconomic Development',
         'Share of energy consumption_Q2' : 'Socioeconomic Development',
         'Share of energy consumption_Q3' : 'Socioeconomic Development',
         'Share of energy consumption_Q4' : 'Socioeconomic Development',
         'Share of energy consumption_Q5' : 'Socioeconomic Development',
         'Share of total space heat demand' : 'Socioeconomic Development',
         'Air pollution Damage Costs':'Air quality and wellbeing',
         'Air pollution & Emissions':'Environment & Climate',
         'Fossil Fuel Consumption':'Environment & Climate',
         'Water used in electricity generation' : 'Environment & Climate',
         'Fuel imports' : 'Environment & Climate',
         'Material Use' : 'Environment & Climate'}
fuels = ['Electricity', 'Gas', 'Other Fuels', 'Liquid Fuels']
agri =  ['1 Crop production', '2 Forestry', ' 3 Fishing']
keep = ['BIOGAS', 'BIOMASS', 'COAL GASES', 'GEO-THERMAL', 'HARD COAL',
       'HEAVY FUEL OIL', 'HYDRO', 'NATURAL GAS', 'NUCLEAR']

# %%
# ----------------------------------------------------------------------------
```

```python
# --------------------------- READ DATA INPUTS --------------------------------
# ----------------------------------------------------------------------------
if __name__ == '__main__':
    os.chdir(directory)

############################ NON-E3ME LITE DATA ############################
#%%
    # Metadata
    print('Reading data inputs')
    metadata_units = pd.read_excel(r'Data\Units of measurement.xlsx', sheet_name='Sheet2').to_dict('records')
    metadata_units = metadata_units[0]
    ## Data Converters
    # E3ME electricity sources to WRI classification (Unit: GWh -> kwh)
    converter_wri = pd.read_excel('Data\Converter_E3ME electricity sources to WRI.xlsx', index_col=0).fillna(0)
    # 70 E3ME sectors to broad sectors
    converter_sectors = pd.read_csv('J:\Projects\DG Research\REFEREE (P1451)\Multiple benefits quantification\Python\Data\E3ME 70 Sector to Broad sector.csv',
                         index_col=0).fillna(0)
    # Country codes lookup
    country_lookup = pd.read_excel('Data\EU country code lookup.xlsx', sheet_name = 'E3ME', index_col=0)
    country_lookup_bsm = pd.read_excel('Data\EU country code lookup.xlsx', sheet_name = 'BSM', index_col=0)
    conv = pd.read_excel('Data\BSM_data.xlsx', sheet_name='Converter').fillna(0).set_index('Archetype')
    # Damage costs (Unit: Euro per kg)
    damage_costs = pd.read_excel('Data\Damage costs.xlsx',
                             index_col=0).div(0.001 )# convert from EUR per kg to EUR per tonne
    damage_costs = damage_costs * deflating_factor # deflate prices to EUR 2010
    # Water withdrawal coefficients (Unit: gal/kWh)
    water_coefficients = pd.read_excel('Data\Water coefficients.xlsx',
                                   sheet_name = 'Sheet1', index_col=0)
    water_coefficients = water_coefficients.loc[water_coefficients.index.intersection(list(damage_costs.index)+['UK'])]
    water_coefficients = water_coefficients.apply(lambda x: x.fillna(x.mean()),axis = 0)
    water_coefficients = water_coefficients.reset_index()
```

```
    water_coefficients_long = pd.melt(water_coefficients, id_vars = ['index'],
                                      var_name ='Level of Disaggregation', value_name='values')
    water_coefficients_long = water_coefficients_long.rename(columns = {'index':'Country'})
    # Eurostat shares of overall consumption spent on energy
    shares = pd.read_excel('Data\Shares.xls', sheet_name= 'hbs_str_t223',
                           skiprows = 3, index_col=0)
    split = pd.read_excel('Data\Energy consumption split.xlsx')
    split = split.set_index(['Unnamed: 0', 'Unnamed: 1'])
    # Break down Eurostat shares by energy source using split dataframe
    e = shares.mul(split.loc[('Italy', 'Electricity')])
    e['Level of Disaggregation'] = 'Electricity'
    g = shares.mul(split.loc[('Italy', 'Gas')])
    g['Level of Disaggregation'] = 'Gas'
    lf = shares.mul(split.loc[('Italy', 'Liquid fuels')])
    lf['Level of Disaggregation'] = 'Liquid Fuels'
    of = shares.mul(split.loc[('Italy', 'Other fuels')])
    of['Level of Disaggregation'] = 'Other Fuels'
    detailed_shares = pd.concat([e,g,lf,of],axis=0).reset_index()
    detailed_shares = detailed_shares.set_index(['index', 'Level of Disaggregation'])
    # Reshape shares of energy consumption to a more useful format
    detailed_shares_q1 = pd.DataFrame(detailed_shares.loc[:,'First quintile'])
    detailed_shares_q2 = pd.DataFrame(detailed_shares.loc[:,'Second quintile'])
    detailed_shares_q3 = pd.DataFrame(detailed_shares.loc[:,'Third quintile'])
    detailed_shares_q4 = pd.DataFrame(detailed_shares.loc[:,'Fourth quintile'])
    detailed_shares_q5 = pd.DataFrame(detailed_shares.loc[:,'Fifth quintile'])

# %%
    # Read coefficients from material flo accounts tool
    coeff = pd.read_excel('Data\EarlyEstimates Tool_V9_NEW.xlsb.xlsm',
                                      sheet_name = 'Results Set',
                                      skiprows = coeff_loc_1, skipfooter= coeff_loc_2,
                                      usecols = 'B:E', index_col = 0).dropna()
    coeff['Country'] = coeff.index.map(country_lookup['Country'])
```

```
    coeff = coeff.reset_index(drop=True)
    coeff = coeff.rename(columns={'Unnamed: 2':'Material',
                                  'Unnamed: 3':'Coefficient',
                                  'Unnamed: 4':'Value'})
    c = ['Material', 'Coefficient', 'Country', 'Value']
    coeff = coeff[c]

    # %%
    # Check detailed_shares sum to shares
#    detailed_shares = detailed_shares.reset_index()
#    detailed_shares = [g for _,g in detailed_shares.groupby('index')]
#    detailed_shares_check = []
#    for i in detailed_shares:
#        i = i.set_index('Level of Disaggregation')
#        i.loc['Total'] = i.sum(numeric_only=True, axis=0)
#        i = i.fillna(method='ffill')
#        i = round(i,3)
#        detailed_shares_check.append(i)
#    detailed_shares_check = pd.concat(detailed_shares_check).reset_index()
#    detailed_shares_check = detailed_shares_check[detailed_shares_check['Level of Disaggregation'] == 'Total']
#    detailed_shares_check = detailed_shares_check.set_index('index')
#    detailed_shares_check = detailed_shares_check.reindex(shares.index)
#
#    detailed_shares_check.iloc[:,1:] == shares

# %% ############################ BUILDING STOCK MODEL DATA ################

    bsm = pd.read_excel('Data\BSM_data.xlsx').rename(columns = {'Unnamed: 0':'helper'}).dropna()
    cntry = []
    for i in list(set(bsm.loc[:,'helper'])):
        if len(i) == 2:
            cntry.append(i)
    cntry.sort()
```

```
    df_bsm = pd.DataFrame()
    for cn in cntry:

        ind = bsm[bsm.loc[:,'helper']==cn].index[0]
        start = ind+1
        end = ind+21
        df_sto = bsm.loc[start:end,:]
        df_sto = df_sto.copy()
        df_sto.loc[:,'Country code'] = cn
        df_bsm = pd.concat([df_bsm,df_sto]).reset_index(drop=True)
    df_bsm=df_bsm.rename(columns={'helper':'Age'})

# %%
############################ E3ME LITE DATA ###########################

    # -----------------------------------------------------
    # Aggregate variables
    # -----------------------------------------------------

    print('Beginning data processing')
    print('Step 1')
    final_dataset = {}
    for name in names:
        dataset = {}
        aggregate_var = pd.concat([
                pd.read_excel(file, sheet_name=name, skipfooter = aggregate_var_loc_1,
                              usecols = 'A,N:BJ')
                .assign(Country=file.stem)
                .rename(columns={'Unnamed: 0':'Variable'})
        for file in Path(path).glob('*.xlsx')]).dropna()
        cols = list(aggregate_var.columns)
        cols = [cols[-1]] + cols[:-1]
        aggregate_var = aggregate_var[cols]
```

```
        aggregate_var.index = np.arange(0, len(aggregate_var))
        aggregate_var = aggregate_var.rename(columns={'Variable' : 'Indicator'})
        aggregate_var = aggregate_var[aggregate_var.Country != 'World']

# %%
        # -----------------------------------------------------
        # Gross Value Added
        # -----------------------------------------------------
        print('Step 2')
        gva_list = ([pd.read_excel(file, sheet_name=name, skiprows = gva_loc_1,
                                   skipfooter = gva_loc_2, usecols = 'A,N:BJ', index_col = 0).T
                                    .dot(converter_sectors).T
                                    .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])

        for v in gva_list:
            v.loc['Total'] = v.sum(axis=0, numeric_only=True)
            v.ffill(inplace=True)
        gva = pd.concat(gva_list).reset_index()

        # Order columns
        cols = list(gva.columns)
        cols = [cols[-1]] + cols[:-1]
        gva = gva[cols]
        gva.columns = aggregate_var.columns
        gva = gva.rename(columns={'Indicator':'Level of Disaggregation'})
        gva['Indicator'] = 'Gross Value Added'
        gva = gva[cols_final]
        metadata_sectors = gva.iloc[:, 0:3]

        #Extract Total GVA by Member State
        gva_tot = gva[gva['Level of Disaggregation'] == 'Total'].reset_index(drop=True)
        world_gva = gva_tot.iloc[29,3:]
```

```
        gva = gva[gva.Country != 'World']
        dataset['Gross Value Added']= gva
    # %%
        # -------------------------------------------------------
        # Employment
        # -------------------------------------------------------
        print('Step 3')
        employment_list = ([
        pd.read_excel(file,
                      sheet_name=name, skiprows = employment_loc_1,
                      skipfooter = employment_loc_2,
                      usecols = 'A,N:BJ', index_col=0).T
            .dot(converter_sectors).T
            .assign(Country=file.stem)
            #.rename(columns={'Unnamed: 0':'Variable'})
        for file in Path(path).glob('*.xlsx')])

        employment = pd.concat(employment_list).reset_index()

        # Order columns
        cols = list(employment.columns)
        cols = [cols[-1]] + cols[:-1]
        employment = employment[cols]
        employment.columns = aggregate_var.columns
        employment = employment.rename(columns={'Indicator':'Level of Disaggregation'})
        employment['Indicator'] = 'Employment'
        employment = employment[cols_final].reset_index(drop=True)
        employment = employment[employment.Country != 'World']
        dataset['Employment'] = employment

    # %%
        # -------------------------------------------------------
        # Energy consumption
```

```
        # ----------------------------------------------------------------------

        print('Step 4')
        enercon_list = ([pd.read_excel(file, sheet_name=name, skiprows = enercon_loc_1,
                                       skipfooter = enercon_loc_2, usecols = 'A,N:BJ', index_col = 0)
                                        .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])
        energy_consumption = pd.concat(enercon_list).reset_index()

        # Order columns
        cols = list(energy_consumption.columns)
        cols = [cols[-1]] + cols[:-1]
        energy_consumption = energy_consumption[cols]
        energy_consumption.columns = aggregate_var.columns
        energy_consumption = energy_consumption.rename(columns={'Indicator':'Level of Disaggregation'})
        energy_consumption['Indicator'] = 'Energy Consumption'
        energy_consumption = energy_consumption[cols_final].reset_index(drop=True)
        energy_consumption = energy_consumption[energy_consumption.Country != 'World']

    # %%
        # ----------------------------------------------------------------------
        # Prices
        # ----------------------------------------------------------------------
        print('Step 5')
        price_list = ([pd.read_excel(file, sheet_name=name, skiprows = price_loc_1,
                                     skipfooter = price_loc_2, usecols = 'A,N:BJ', index_col = 0)
                                      .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])
        average_price = pd.concat(price_list).reset_index()

        # Order columns
        cols = list(average_price.columns)
        cols = [cols[-1]] + cols[:-1]
```

```
        average_price = average_price[cols]
        average_price.columns = aggregate_var.columns
        average_price = average_price.rename(columns={'Indicator':'Level of Disaggregation'})
        average_price['Indicator'] = 'Average prices'
        average_price = average_price[cols_final].reset_index(drop=True)
        average_price = average_price[average_price.Country != 'World']

    # %%
        # -----------------------------------------------------
        # Derive average energy cist from Prices
        # -----------------------------------------------------
        print('Step 6')
        energy_cost = energy_consumption.iloc[:,3:].mul(average_price.iloc[:,3:])
        energy_cost = pd.concat([energy_consumption.iloc[:,0:3],energy_cost], axis=1)
        energy_cost['Indicator'] = 'Energy Cost'

        energy_cost_tot = energy_cost.groupby(by=['Country']).sum()
        energy_cost_tot['Indicator'] = 'Total Energy Cost'
        cols = list(energy_cost_tot.columns)
        cols = [cols[-1]] + cols[:-1]
        energy_cost_tot = energy_cost_tot[cols].reset_index()
        energy_cost_tot = energy_cost_tot[energy_cost_tot['Country'] != 'World']

    # %%
        # -----------------------------------------------------
        # Estimate total energy consumption
        # -----------------------------------------------------
        print('Step 7')
        for v in enercon_list:
            v.loc['Total'] = v.sum(axis=0, numeric_only=True)
            v.ffill(inplace=True)
        energy_consumption = pd.concat(enercon_list).reset_index()

```

```
        # Order columns
        cols = list(energy_consumption.columns)
        cols = [cols[-1]] + cols[:-1]
        energy_consumption = energy_consumption[cols]
        energy_consumption.columns = aggregate_var.columns
        energy_consumption = energy_consumption.rename(columns={'Indicator':'Level of Disaggregation'})
        energy_consumption['Indicator'] = 'Energy Consumption'
        energy_consumption = energy_consumption[cols_final]

        # Extract data on Total Energy Consumption by Member State
        energy_consumption_tot = energy_consumption[energy_consumption['Level of Disaggregation'] ==
          'Total'].reset_index(drop=True)
        energy_consumption_tot = energy_consumption_tot[energy_consumption_tot['Country'] != 'World']
    # %%
        # -----------------------------------------------------
        # Consumer Expenditure
        # -----------------------------------------------------

        print('Step 8')
        consumer_expenditure = pd.concat([
        pd.read_excel(file,
                      sheet_name=name, skiprows = expenditure_loc_1,
                      skipfooter = expenditure_loc_2,
                      usecols = 'A,N:BJ')
            .assign(Country=file.stem)  # We may also want file.name here
            .rename(columns={'Unnamed: 0':'Variable'})
        for file in Path(path).glob('*.xlsx')]).dropna()

        # Order columns
        cols = list(consumer_expenditure.columns)
        cols = [cols[-1]] + cols[:-1]
        consumer_expenditure = consumer_expenditure[cols]
        consumer_expenditure.columns = aggregate_var.columns
```

```
        consumer_expenditure = consumer_expenditure.rename(columns={'Indicator':'Level of Disaggregation'})
        consumer_expenditure['Indicator'] = 'Consumer Expenditure'
        consumer_expenditure = consumer_expenditure[cols_final]
        consumer_expenditure = consumer_expenditure[consumer_expenditure.Country != 'World']

        # Extract consumer expenditure for electricity, gas, liquid fuels and other fuels
        electricity = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '9 Electricity'].reset_index(drop=True)
        electricity['Level of Disaggregation'] = 'Electricity'

        gas = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '10 Gas'].reset_index(drop=True)
        gas['Level of Disaggregation'] = 'Gas'

        liquid_f = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '11 Liquid Fuels'].reset_index(drop=True)
        liquid_f['Level of Disaggregation'] = 'Liquid Fuels'

        other_f = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '12 Other Fuels'].reset_index(drop=True)
        other_f['Level of Disaggregation'] = 'Other Fuels'

        con_exp_ener = pd.concat([electricity, gas, liquid_f, other_f], axis=0)
        dataset['Consumer Expenditure'] = con_exp_ener

    # %%
        # -----------------------------------------------------
        # Gross Domestic Product
        # ----------------------------------------------------

        print('Step 9')
        gdp = aggregate_var.loc[aggregate_var['Indicator'] == 'GDP'].reset_index(drop=True)
        dataset['Gross Domestic Product'] = gdp
```

```python

        # Extract fossil fuel consumption
        coal = aggregate_var.loc[aggregate_var['Indicator'] == 'Coal fuel consumption'].reset_index(drop=True)
        gas = aggregate_var.loc[aggregate_var['Indicator'] == 'Gas fuel consumption'].reset_index(drop=True)
        oil = aggregate_var.loc[aggregate_var['Indicator'] == 'Oil Consumption (Crude oil, Heavy fuel Oil and Middle distallates'].reset_index(drop=True)
        fossil_fuels = pd.concat([coal, gas, oil])
        fossil_fuels = fossil_fuels.rename(columns={'Indicator':'Level of Disaggregation'})
        fossil_fuels['Indicator'] = 'Fossil Fuel Consumption'
        fossil_fuels = fossil_fuels[cols_final]
        dataset['Fossil Fuel Consumption'] = fossil_fuels

    # %%
        # -----------------------------------------------------
        # Energy prices
        # -----------------------------------------------------

        print('Step 10')
        elec_list = ([pd.read_excel(file, sheet_name=name, skiprows = energy_prices_loc_1,
                                   skipfooter = energy_prices_loc_2, usecols = 'A,N:BJ', index_col = 0)
                                    .assign(Country=file.stem)

        for file in Path(path).glob('*.xlsx')])
        extracted_cols = list(aggregate_var.iloc[:,2:].columns)
        extracted_cols.append(add)
        new = []
        for i in elec_list:
            n = i.loc['19 Households']
            n.columns = extracted_cols
            w = n.iloc[3:,:].mean()
            n = n.append(w, ignore_index=True).fillna(method='ffill').drop([3,4], axis=0).reset_index(drop=True)
            n['Level of Disaggregation'] = fuels
            n['Indicator'] = 'Price'
```

```
            n = n[cols_final]
            new.append(n)
        new_dataset = pd.concat(new)
        dataset['Price'] = new_dataset

 # %%
        # -----------------------------------------------------
        # Gross output (Unit: €2010m)
        # -----------------------------------------------------

        print('Step 11')
        output_list = ([pd.read_excel(file, sheet_name=name, skiprows = output_loc_1,
                                      skipfooter = output_loc_2, usecols = 'A,N:BJ', index_col = 0)
                                       .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])

        for v in output_list:
            v.loc['Total'] = v.sum(axis=0, numeric_only=True)
            v.ffill(inplace=True)
        output = pd.concat(output_list).reset_index()

        # Order columns
        cols = list(output.columns)
        cols = [cols[-1]] + cols[:-1]
        output = output[cols]
        output.columns = aggregate_var.columns
        output = output.rename(columns={'Indicator':'Level of Disaggregation'})
        output['Indicator'] = 'Gross Output'
        output = output[output['Country'] != 'World']
        output = output[cols_final]
# %%
        # -----------------------------------------------------
        # Electricity generation
```

```
        # -----------------------------------------------------

        print('Step 12')
        generation_list = ([pd.read_excel(file, sheet_name=name, skiprows = generation_loc_1,
                                   skipfooter = generation_loc_2, usecols = 'A,N:BJ', index_col = 0).T
                                    .dot(converter_wri).T
                                    .mul(1000000) #Conversion fro GWh to kwh
                                    .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])
        generation = pd.concat(generation_list).reset_index()

        # Order columns
        cols = list(generation.columns)
        cols = [cols[-1]] + cols[:-1]
        generation = generation[cols]
        generation.columns = aggregate_var.columns
        generation = generation.rename(columns={'Indicator':'Level of Disaggregation'})
        generation['Indicator'] = 'Electricity generation'
        generation = generation[cols_final]
        generation = generation[(generation.Country != 'World') & (generation.Country != 'EU27')]
        generation = generation[generation['Level of Disaggregation'].isin(keep)]
#%%
        # -----------------------------------------------------
        # Material Use for 2021 from Eurostat (Unit: Thousands tonnes)
        # -----------------------------------------------------

        print('Step 13')
        material_use_eurostat = pd.read_excel('Data\env_ac_mfa__custom_3549625_page_spreadsheet.xlsx',
                                              sheet_name = 'Sheet 1',
                                              skiprows = material_loc_1, skipfooter = material_loc_2,
                                              usecols = 'A:C', index_col = 0)
        material_use_eurostat = material_use_eurostat.loc[material_use_eurostat.index.isin(list
          (damage_costs.index) +['UK'])] #add loc here? copy warning
```

```
#%%

        # -----------------------------------------------------
        # Energy Imports (Unit: €2010m)
        # -----------------------------------------------------

        print('Step 14')
        imports = ([pd.read_excel(file,
                      sheet_name=name, skiprows = imports_loc_1, skipfooter = imports_loc_2,
                      usecols = 'A,N:BJ', index_col=0)
            .assign(Country=file.stem)
            #.rename(columns={'Unnamed: 0':'Variable'})
        for file in Path(path).glob('*.xlsx')])

        imports = pd.concat(imports).reset_index()

        # Order columns
        cols = list(imports.columns)
        cols = [cols[-1]] + cols[:-1]
        imports = imports[cols]
        imports.columns = aggregate_var.columns
        imports = imports.rename(columns={'Indicator':'Level of Disaggregation'})
        imports['Indicator'] = 'Fuel imports'
        imports = imports[cols_final].reset_index(drop=True)
        imports = imports[imports.Country != 'World']
        imports = imports[(imports['Level of Disaggregation'].isin(['4 Coal',
                           '5 Oil and Gas','12 Manufactured fuels']))].reset_index(drop=True)
# %% main function
# -----------------------------------------------------------------------------
# ---------------------QUANTIFICATION OF MULTIPLE BENEFITS --------------------
# -----------------------------------------------------------------------------

        # =============================================================
```

```
        #  Labour productivity by country (Unit: €2010m per worker)
        # =============================================================

        print('Beginning of the multiple benefits quantifications')
        print('Step 15')
        use_gva = gva[gva['Level of Disaggregation'] != 'Total']
        use_gva = use_gva.reset_index(drop=True)
        productivity = use_gva.iloc[:, 3:].div((employment.iloc[:, 3:]*1000))
        productivity = pd.concat([employment.iloc[:,0:3], productivity], axis=1)
        productivity['Indicator'] = 'Labour Productivity'

        metadata_sectors = metadata_sectors[metadata_sectors['Level of Disaggregation'] != 'Total']
        dataset['Labour Productivity'] = productivity

    # %%
        # =============================================================
        #  Energy intensity (Unit: ktoe per €2010m)
        # =============================================================

        print('Step 15')
        energy_intensity = energy_consumption_tot.iloc[:,3:].div(gva_tot.iloc[0:29, 3:])
        energy_intensity = pd.concat([energy_consumption_tot.iloc[:, 0:3],
                                      energy_intensity], axis=1)
        energy_intensity['Indicator'] = 'Energy Intensity'
        energy_intensity = energy_intensity[energy_intensity['Country'] != 'World']
        dataset['Energy Intensity'] = energy_intensity
    # %%
        # =============================================================
        #  Energy cost impact (Unit: €2010m)
        # =============================================================

        print('Step 16')
        energy_cost_impact = energy_cost_tot.iloc[:, 2:].div(gva_tot.iloc[0:29, 3:])
```

```
        energy_cost_impact = pd.concat([energy_consumption_tot.iloc[:, 0:3],
                                        energy_cost_impact], axis=1)
        energy_cost_impact['Indicator'] = 'Energy Cost Impact'
        dataset['Energy Cost Impact'] = energy_cost_impact
    # %%
        # ==============================================================
        #  International competitiveness (Unit: ratio)
        # ==============================================================

        print('Step 17')
        int_comp = gva_tot.iloc[0:29,3:]/gva_tot.iloc[29,3:]
        int_comp = pd.concat([gva_tot.iloc[0:29,0:3], int_comp], axis=1)
        int_comp['Indicator'] = 'International competitiveness'
        dataset['International Competitiveness'] = int_comp
    # %%
        # ==============================================================
        #  Public budget as share of GDP (Unit: ratio)
        # ==============================================================

        print('Step 18')
        gov_exp = aggregate_var.loc[aggregate_var['Indicator'] == 'Total government expenditure'].reset_index(drop=True)

        # Finally compute public budget as share of GDP
        public_budget = gov_exp.iloc[:,2:].div(gdp.iloc[:,2:])
        public_budget = pd.concat([gdp.iloc[:,0:2], public_budget], axis=1)
        public_budget = public_budget.rename(columns={'Variable' : 'Indicator'})
        public_budget['Indicator'] = 'Public budget as share of GDP'
        public_budget['Level of Disaggregation'] = 'Total'
        dataset['Public budget as share of GDP'] = public_budget

    # %%
        # ==============================================================
```

```
        #  Energy indipendence
        # =============================================================

        print('Step 19')
        #Extract Total GVA by Member State
        output_tot = output[output['Level of Disaggregation'] == 'Total'].reset_index(drop=True)

        # Finally, estimate energy imporst as a share of gross output (Unit: %)
        imports_shares = imports.set_index('Country').iloc[:,2:].div(output_tot.set_index('Country').iloc[:,2:])
        imports_shares = imports_shares.reset_index()
        imports_shares['Indicator'] = 'Fuel imports'
        imports_shares = imports_shares.join(imports['Level of Disaggregation'])
        imports_shares =  imports_shares[cols_final]
        dataset['Fuel imports'] = imports_shares

    # %%
        # =============================================================
        #  Water used for electricity generation (Unit: gal)
        # =============================================================

        print('Step 20')
        #Put electricity generation data in the right format for matrix moltiplication
        water_coefficients_long =water_coefficients_long[(water_coefficients_long['Level of Disaggregation'].isin(keep))]

        water_coefficients_long_clean = water_coefficients_long.set_index(['Country','Level of Disaggregation'])
        generation_clean=generation.drop('Indicator', axis=1)
        generation_clean = generation_clean.set_index(['Country','Level of Disaggregation'])
        for i in generation_clean.columns:
            water_coefficients_long_clean[i] = water_coefficients_long_clean['values']
        water_coefficients_long_clean=water_coefficients_long_clean.drop('values', axis=1)

        # Estimate water use per unit of electricity generated by source (Unit: gallon)
```

```
        water_use = generation_clean * water_coefficients_long_clean
        water_use = water_use.reset_index()
        water_use['Indicator'] = 'Water used in electricity generation'
        water_use = water_use[cols_final]
        dataset['Water used in electricity generation'] = water_use

        # Note: Zero velues in the water use dataset show that
        # either no electricity is produced form X source in Y country (1) or
        # electricity generation from X source in Y country is not associated with any water withdrawal (2)

    #%%
        # ============================================================
        #  Air pollution costs
        # ============================================================

        print('Step 21')
        nox = aggregate_var.loc[aggregate_var.Indicator == 'Nox Emissions']
        so2 = aggregate_var.loc[aggregate_var.Indicator == 'SO2 Emissions']
        ch4 = aggregate_var.loc[aggregate_var.Indicator == 'CH4 Emissions']
        voc = aggregate_var.loc[aggregate_var.Indicator == 'VOC Emissions']
        pm = aggregate_var.loc[aggregate_var.Indicator == 'PM10 Emissions']
        co2 = aggregate_var.loc[aggregate_var.Indicator == 'CO2 Emissions']
        emissions = pd.concat([nox,so2, ch4, voc, pm, co2])
        emissions = emissions.rename(columns={'Indicator':'Level of Disaggregation'})
        emissions['Indicator'] =  'Air pollution & Emissions'
        emissions = emissions[cols_final]
        dataset['Air pollution & Emissions'] = emissions

        #NOx Damage costs (Unit: Euro)
        nox = nox.drop('Indicator', axis=1)
        nox = nox[0:28].set_index('Country').mul(1000)
        nox_cost = nox.multiply(damage_costs['nox'], axis=0).reset_index()
        nox_cost = nox_cost.rename(columns={'index' : 'Country'})
```

```
        nox_cost['Level of Disaggregation'] = 'NOX Damage Cost'
        nox_cost['Indicator'] = 'Air pollution Damage Costs'
        nox_cost = nox_cost[cols_final]
        #SO2 Damage costs (Unit: Euro)
        so2 = so2.drop('Indicator', axis=1)
        so2 = so2[0:28].set_index('Country').mul(1000)
        so2_cost = so2.multiply(damage_costs['so2'], axis=0).reset_index()
        so2_cost = so2_cost.rename(columns={'index' : 'Country'})
        so2_cost['Level of Disaggregation'] = 'SO2 Damage Cost'
        so2_cost['Indicator'] = 'Air pollution Damage Costs'
        so2_cost = so2_cost[cols_final]

        #VOC Damage costs (Unit: Euro)
        voc = voc.drop('Indicator', axis=1)
        voc = voc[0:28].set_index('Country').mul(1000)
        voc_cost = voc.multiply(damage_costs['voc'], axis=0).reset_index()
        voc_cost = voc_cost.rename(columns={'index' : 'Country'})
        voc_cost['Level of Disaggregation'] = 'VOC Damage Cost'
        voc_cost['Indicator'] = 'Air pollution Damage Costs'
        voc_cost = voc_cost[cols_final]

        #PM Damage costs (Unit: Euro)
        pm = pm.drop('Indicator', axis=1)
        pm = pm[0:28].set_index('Country').mul(1000)
        pm_cost = pm.multiply(damage_costs['pm'], axis=0).reset_index()
        pm_cost = pm_cost.rename(columns={'index' : 'Country'})
        pm_cost['Level of Disaggregation'] = 'PM Damage Cost'
        pm_cost['Indicator'] = 'Air pollution Damage Costs'
        pm_cost = pm_cost[cols_final]

        damages = pd.concat([nox_cost, so2_cost, voc_cost, pm_cost])
        dataset['Air pollution Damages'] = damages

```

```
# %%
        # ============================================================
        #  Domestic material extraction
        # ============================================================

        print('Step 22')
        # Read GVA for relevant sectors
        lista = ([pd.read_excel(file, sheet_name=name, skiprows = lista_loc_1,
                               skipfooter = lista_loc_2, usecols = 'A,M:BJ', index_col = 0)
                                .assign(Country=file.stem)
        for file in Path(path).glob('*.xlsx')])

        mat = pd.concat(lista)
        mat = mat.reset_index()
        cols = list(mat.columns)
        cols = [cols[-1]] + cols[:-1]
        mat = mat.loc[:,cols]
        mat =  mat.set_axis(cols_mat, axis=1)

        lista2 = []
        for i in lista:
            j = i.iloc[0:3]
            j.loc['Total'] = j.sum(axis=0, numeric_only=True)
            j.ffill(inplace=True)
            j = j.reset_index()
            j = (j[cols]).set_axis(cols_mat, axis=1)
            lista2.append(j)

        lista3 = []
        for i in lista:
            w = i.iloc[3:6]
            w.loc['Total'] = w.sum(axis=0, numeric_only=True)
            w.ffill(inplace=True)
```

```
            w = w.reset_index()
            w = (w[cols]).set_axis(cols_mat, axis=1)
            lista3.append(w)

        mf1 = (pd.concat(lista2))
        mf1 = mf1[mf1['Indicator'].isin(['Total'])].reset_index(drop=True).drop('Indicator', axis=1)
        mf1_p = mf1.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
        mf2 = mat[mat['Indicator'].isin(['18 Metal products'])].reset_index(drop=True).drop('Indicator', axis=1)
        mf2_p = mf2.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
        mf3 = mat[mat['Indicator'].isin(['30 Construction'])].drop('Indicator', axis=1)
        mf3_p = mf3.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
        mf4 = (pd.concat(lista3))
        mf4 = mf4[mf4['Indicator'].isin(['Total'])].reset_index(drop=True).drop('Indicator', axis=1)
        mf4_p = mf4.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
# %%
        # Estimate material use

        # Biomass (Code: MF1)
        print('Step 23')
        alpha_mf1 = coeff[(coeff['Material'] == 'MF1') & (coeff['Coefficient'] == 'Intercept')].drop
          (['Coefficient', 'Material'], axis=1).set_index(['Country'])
        beta_mf1 = coeff[(coeff['Material'] == 'MF1') & (coeff['Coefficient'] == 'P1')].drop(['Coefficient',
          'Material'], axis=1).set_index(['Country'])
        eurostat_mf1 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Biomass'].drop
          ('Level of Disaggregation', axis=1)

        relative_change_mf1 = pd.DataFrame()
        mf1_p = mf1_p.reindex(beta_mf1.index)
        for i in mf1_p.columns:
            relative_change_mf1[i] =alpha_mf1['Value'].add((beta_mf1['Value'].mul(mf1_p[i])))
        relative_change_mf1 = relative_change_mf1.drop(2021, axis=1)

        years_ = list(relative_change_mf1.columns)
```

```

        final_mf1 = eurostat_mf1.copy()
        for ye in years_:

            year = ye

            year_1 = int(ye)-1

            euro = final_mf1[year_1]
            rel_ch = 1+relative_change_mf1[year]
            calc = euro*rel_ch
            calc = pd.DataFrame(calc)
            calc.columns = [year]
            final_mf1[year] = calc[year]
            final_mf1['Level of Disaggregation'] = 'Biomass'
    # %%
         # Metal Ores (Code: MF2)
        print('Step 24')
        alpha_mf2 = coeff[(coeff['Material'] == 'MF2') & (coeff['Coefficient'] == 'Intercept')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        beta_mf2 = coeff[(coeff['Material'] == 'MF2') & (coeff['Coefficient'] == 'P10')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        eurostat_mf2 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Metal ores (gross ores)'].drop('Level of Disaggregation', axis=1)

        relative_change_mf2 = pd.DataFrame()
        mf2_p = mf2_p.reindex(beta_mf2.index)
        for i in mf2_p.columns:
            relative_change_mf2[i] =alpha_mf2['Value'].add((beta_mf2['Value'].mul(mf2_p[i])))
        relative_change_mf2 = relative_change_mf2.drop(2021, axis=1)

        final_mf2 = eurostat_mf2.copy()
        for ye in years_:
```

```python

            year = ye

            year_1 = int(ye)-1

            euro = final_mf2[year_1]
            rel_ch = 1+relative_change_mf2[year]
            calc = euro*rel_ch
            calc = pd.DataFrame(calc)
            calc.columns = [year]
            final_mf2[year] = calc[year]
            final_mf2['Level of Disaggregation'] = 'Metal Ores'

    # %%
        # Non-metallic minerals (Code: MF3)
        print('Step 25')
        alpha_mf3 = coeff[(coeff['Material'] == 'MF3') & (coeff['Coefficient'] == 'Intercept')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        beta_mf3 = coeff[(coeff['Material'] == 'MF3') & (coeff['Coefficient'] == 'P14')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        eurostat_mf3 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Non-metallic minerals'].drop('Level of Disaggregation', axis=1)

        relative_change_mf3 = pd.DataFrame()
        mf3_p = mf3_p.reindex(beta_mf3.index)
        for i in mf3_p.columns:
            relative_change_mf3[i] =alpha_mf3['Value'].add((beta_mf3['Value'].mul(mf3_p[i])))
        relative_change_mf3 = relative_change_mf3.drop(2021, axis=1)

        final_mf3 = eurostat_mf3.copy()
        for ye in years_:

            year = ye
```

```

            year_1 = int(ye)-1

            euro = final_mf3[year_1]
            rel_ch = 1+relative_change_mf3[year]
            calc = euro*rel_ch
            calc = pd.DataFrame(calc)
            calc.columns = [year]
            final_mf3[year] = calc[year]
            final_mf3['Level of Disaggregation'] = 'Non-metallic minerals'


   # %%
        # Fossil energy material/carriers (Code: MF4)
        print('Step 26')
        alpha_mf4 = coeff[(coeff['Material'] == 'MF4') & (coeff['Coefficient'] == 'Intercept')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        beta_mf4 = coeff[(coeff['Material'] == 'MF4') & (coeff['Coefficient'] == 'P17')].drop(['Coefficient', 'Material'], axis=1).set_index(['Country'])
        eurostat_mf4 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Fossil energy materials/carriers'].drop('Level of Disaggregation', axis=1)
#%%
        relative_change_mf4 = pd.DataFrame()
        mf4_p = mf4_p.reindex(beta_mf4.index)
        for i in mf4_p.columns:
            relative_change_mf4[i] =alpha_mf4['Value'].add((beta_mf4['Value'].mul(mf4_p[i])))
        relative_change_mf4 = relative_change_mf4.drop(2021, axis=1)

        final_mf4 = eurostat_mf4.copy()
        for ye in years_:

            year = ye

```

```
            year_1 = int(ye)-1

            euro = final_mf4[year_1]
            rel_ch = 1+relative_change_mf4[year]
            calc = euro*rel_ch
            calc = pd.DataFrame(calc)
            calc.columns = [year]
            final_mf4[year] = calc[year]
            final_mf4['Level of Disaggregation'] = 'Fossil energy material/carrier'

# %%
        print('Step 28')
        material_final = pd.concat([final_mf1, final_mf2, final_mf3, final_mf4]).reset_index()
        material_final['Indicator'] = 'Material Use'
        material_final = material_final[cols_final]
        dataset['Material Use'] = material_final


# %%
        # ==============================================================
        #  Garther relevant indicators into a dictionary
        #  with reluts from the Baseline and the Scenario
        # ==============================================================

        # Create dataframe for Baseline and Scenario with thelevant indicators
        print('Step 29')
        final_dataset[name]=dataset

        # Check if datafarmes in final_dataset contains NaNs
        print("Checking for any nan values in the Dataframes")
        for i, j in final_dataset.items():
            check_for_nan = {}
            for k, v in j.items():
                _nan = v.isnull().values.any()
```

```
                print(_nan)
                check_for_nan[k] = _nan
                # if value if True, there are NaNs value to investigate
# %%

    #Concatenate all indicators in a single dataframe
    print('Step 30')
    concatenated = {}
    for i,v in final_dataset.items():
        df = pd.DataFrame()
        for j, l in v.items():
            df = pd.concat([df,l])
            df['Level of Disaggregation'] = df['Level of Disaggregation'].replace(np.nan, 'Total')
            df['Pillar'] = df['Indicator'].map(pillars)
            df['Unit'] = df['Indicator'].map(metadata_units)
            df = df[cols_final_]

        concatenated[i]=df.reset_index(drop=True)

# %%
# -----------------------------------------------------------------------------
# ------------------- CALCULATING DIFFERENCES FROM THE BASELINE----------------
# -----------------------------------------------------------------------------

# Calculate absolute difference from the Baseline for each indicator
    print('Step 31')
    abs_diff = concatenated['Scenario'].iloc[:,5:] - concatenated['Baseline'].iloc[:,5:]
    abs_diff_ = pd.concat([concatenated['Scenario'].iloc[:,0:5],abs_diff], axis=1)

    # Check for NaNs
    any_nan = abs_diff_.isnull()
    print("Checking for any nan values in the absolute differences from the baseline scenario:\
     True confirms the existance of NaN values in the Dataframe")
```

```
    print(any_nan) # True confirms the existance of NaN values in the Dataframe
# %%
    # Calculate percentage difference from the Baseline for each indicator
    print('Step 32')
    factor1 = abs_diff_.iloc[:, 5:].astype(float)
    factor2 = concatenated['Baseline'].iloc[:,5:].astype(float)

    pct_diff = factor1.div(factor2)
    pct_diff_ = pd.concat([concatenated['Scenario'].iloc[:,0:5],pct_diff], axis=1)
# %%
# ----------------------------------------------------------------------------
# --------------- CALCULATING DISTRIBUTIONAL IMPACTS -------------------------
# ----------------------------------------------------------------------------

################################# PART A ####################################
#  Distributional impacts by MS and by income quintile...
#  Distributional impacts refer to the share of overall consumption spent on energy
#  Results are broken down by sourge of energy, i.e., electricity, gas, liquid fuels and other fuels
#  *************** Note: missing data for Italy in this dataset *************

    print('Step 33')
    # Extract results on energy consumption change from the Baseline
    pct_energy = pct_diff_[pct_diff_.Indicator == 'Consumer Expenditure'].reset_index(drop=True)
    pct_energy = pct_energy[pct_energy['Country'] != 'UK']
    pct_energy = pct_energy.drop(['Pillar', 'Indicator', 'Unit'], axis=1)
    pct_energy = pct_energy.set_index(['Country', 'Level of Disaggregation'])

#     check = round(e + g + lf + of,3)
#     (check == round(shares,3)).all()
    q1 = pd.DataFrame()
    q2 = pd.DataFrame()
    q3 = pd.DataFrame()
    q4 = pd.DataFrame()
```

```
    q5 = pd.DataFrame()

    # Estimate energy reduction effect
    for i in pct_energy.columns:
        q1[i] = detailed_shares_q1['First quintile'] + (pct_energy[i].mul(detailed_shares_q1['First quintile']))
        q2[i] = detailed_shares_q2['Second quintile'] + (pct_energy[i].mul(detailed_shares_q2['Second quintile']))
        q3[i] = detailed_shares_q3['Third quintile'] + (pct_energy[i].mul(detailed_shares_q3['Third quintile']))
        q4[i] = detailed_shares_q4['Fourth quintile'] + (pct_energy[i].mul(detailed_shares_q4['Fourth quintile']))
        q5[i] = detailed_shares_q5['Fifth quintile'] + (pct_energy[i].mul(detailed_shares_q5['Fifth quintile']))

    # Extrapolate change in prices from Baseline
    p_effect = pct_diff_[pct_diff_.Indicator == 'Price'].reset_index(drop=True)
    p_effect = p_effect[(p_effect['Country'] != 'UK') & (p_effect['Country'] != 'World')]
    p_effect = p_effect.drop(['Pillar', 'Indicator', 'Unit'], axis=1)
    p_effect = p_effect.set_index(['Country', 'Level of Disaggregation']).reindex(detailed_shares_q1.index)

    q1_p = pd.DataFrame()
    q2_p = pd.DataFrame()
    q3_p = pd.DataFrame()
    q4_p = pd.DataFrame()
    q5_p = pd.DataFrame()

    # Incorporate the price effect into the previously esimated distributional impacts
    for i in p_effect.columns:
        q1_p[i] = q1[i] + p_effect[i].mul(detailed_shares_q1['First quintile'])
        q2_p[i] = q2[i] + p_effect[i].mul(detailed_shares_q2['Second quintile'])
        q3_p[i] = q3[i] + p_effect[i].mul(detailed_shares_q3['Third quintile'])
        q4_p[i] = q4[i] + p_effect[i].mul(detailed_shares_q4['Fourth quintile'])
        q5_p[i] = q5[i] + p_effect[i].mul(detailed_shares_q5['Fifth quintile'])

#  ******************************** WARNING *********************************
#  Zero value in consumer expenditure for energy (i.e., Cyprus, Croatia, Malta)
#  return zero values for the distributional impacts indicators as well.
```

```
#  This is due to use of mock data rather than final E3ME LITE results.

# %%
    # Calculate percentage difference from Baseline

    print('Step 34')
    q1_pct = pd.DataFrame()
    q2_pct = pd.DataFrame()
    q3_pct = pd.DataFrame()
    q4_pct = pd.DataFrame()
    q5_pct = pd.DataFrame()

    for i in q1.columns:
        q1_pct[i] = (q1_p[i].subtract(detailed_shares_q1['First quintile'])).div(detailed_shares_q1['First
          quintile'], fill_value=0)
        q2_pct[i] = (q2_p[i].subtract(detailed_shares_q2['Second quintile'])).div(detailed_shares_q2['Second
          quintile'], fill_value=0)
        q3_pct[i] = (q3_p[i].subtract(detailed_shares_q3['Third quintile'])).div(detailed_shares_q3['Third
          quintile'], fill_value=0)
        q4_pct[i] = (q4_p[i].subtract(detailed_shares_q4['Fourth quintile'])).div(detailed_shares_q4['Fourth
          quintile'], fill_value=0)
        q5_pct[i] = (q5_p[i].subtract(detailed_shares_q5['Fifth quintile'])).div(detailed_shares_q5['Fifth
          quintile'], fill_value=0)

    q1_pct = q1_pct.reset_index().rename(columns={'index':'Country'})
    q1_pct['Indicator'] = 'Share of energy consumption_Q1'
    q2_pct = q2_pct.reset_index().rename(columns={'index':'Country'})
    q2_pct['Indicator'] = 'Share of energy consumption_Q2'
    q3_pct = q3_pct.reset_index().rename(columns={'index':'Country'})
    q3_pct['Indicator'] = 'Share of energy consumption_Q3'
    q4_pct = q4_pct.reset_index().rename(columns={'index':'Country'})
    q4_pct['Indicator'] = 'Share of energy consumption_Q4'
    q5_pct = q5_pct.reset_index().rename(columns={'index':'Country'})
```

```
    q5_pct['Indicator'] = 'Share of energy consumption_Q5'

    #Concatenate quintile data frames in a unique dataframe
    energy_shares = pd.concat([q1_pct,q2_pct,q3_pct,q4_pct,q5_pct], axis=0)
    energy_shares['Pillar'] = energy_shares['Indicator'].map(pillars)
    energy_shares['Unit'] = energy_shares['Indicator'].map(metadata_units)
    energy_shares = energy_shares[cols_final_]

    # Append quintile data to pct_diff_ dataframe
    pct_diff_ = pct_diff_.append(energy_shares)
    pct_diff_ = pct_diff_[(pct_diff_['Indicator'] != 'Price') & (pct_diff_['Indicator'] != 'Consumer
      Expenditure')]
    abs_diff_ = abs_diff_[(abs_diff_['Indicator'] != 'Price') & (abs_diff_['Indicator'] != 'Consumer
      Expenditure')]

# %%
    ################################# PART B ###################################
    #  Distributional impacts by MS and by dwelling archetype
    # Process building stock model data (df_bsf)...
    # ...and estimate the share of total speace heat demand...
    # ...of each dwelling archetype for each country

    print('Step 35')
    # Baseline
    df_bsm_list = [g for _,g in df_bsm.groupby('Country code')]
    baseline = []
    for i in df_bsm_list:
        c = i['Country code']
        i = i.set_index('Age')
        i.loc['Total']= i.sum(numeric_only=True, axis=0)
        i = i.fillna(method='ffill')
        i = (i.iloc[:,0:30]).T
        i = i.dot(conv).T
```

```
            hh = i.div(i.loc['Total'])
            hh = pd.concat([c,hh], axis=1).fillna(method='ffill').dropna()
            hh['Country'] = hh['Country code'].map(country_lookup_bsm['Country'])
            hh = hh.drop(['Number of dwellings', 'Country code'], axis=1)
            hh = hh.drop('Total')
            baseline.append(hh)

        bsm_baseline = pd.concat(baseline).reset_index().set_index('Country').sort_values(by=['Country'])
        archetype = bsm_baseline['index'].drop('United Kingdom', axis=0)
        #bsm_baseline = bsm_baseline.drop('index', axis=1)
        bsm_baseline = bsm_baseline.drop('United Kingdom', axis=0)

    # %%
        # Scenario
        # Estimate reduction in heat demand shares by archetypes..
        #...based on E3ME results on consumer expenditure in energy.

        print('Step 36')
        # Extract reduction in consumer expenditure on energy from E3ME LITE results
        pct_energy = pct_energy.reset_index().set_index('Level of Disaggregation')
        pct_energy_ = [g for _,g in pct_energy.groupby('Country')]
        for i in pct_energy_:
            i.loc['Total'] = i.sum(numeric_only=True, axis=0)
            i['Country'] = i['Country'].fillna(method='ffill')
        pct_energy_conct = pd.concat(pct_energy_).reset_index()
        pct_energy_conct =  pct_energy_conct[ pct_energy_conct['Level of Disaggregation'] == 'Total']
        pct_energy_conct = pct_energy_conct.set_index('Country').drop('Level of Disaggregation',axis=1)
        pct_energy_conct = pct_energy_conct.drop('EU27', axis=0)

        bsm_baseline_list = list(set(bsm_baseline.index))
        bsm_baseline_list = sorted(bsm_baseline_list)
        pct_energy_conct = pct_energy_conct.reindex(bsm_baseline_list)

```

```
    # Apply % reduction in consumer expenditure on energy to bsm_baseline
    bsm_scenario_change = bsm_baseline.iloc[:,1:].mul(pct_energy_conct.iloc[:,0:29], axis=0).reset_index
      ().set_index('Country').dropna()
    bsm_scenario = bsm_baseline.iloc[:, 1:].add(bsm_scenario_change).reset_index()
    archetype = archetype.reset_index()
    bsm_scenario = pd.concat([archetype['index'], bsm_scenario], axis=1)
    bsm_scenario = bsm_scenario.rename(columns={'index': 'Level of Disaggregation'})
    bsm_scenario = bsm_scenario[bsm_scenario['Level of Disaggregation'] != 'Total']
# %%
    # Estimate absolute & percentage difference from the Baseline
    print('Step 37')
    bsm_baseline = bsm_baseline.reset_index().set_index(['Country', 'index'])
    bsm_scenario = bsm_scenario.set_index(['Country', 'Level of Disaggregation'])
#%%
    print('Step 38')
    bsm_abs_diff = bsm_scenario.subtract(bsm_baseline)
    bsm_pct_diff = bsm_abs_diff.div(bsm_baseline).reset_index()
    bsm_pct_diff['Indicator'] = 'Share of total space heat demand'
    bsm_pct_diff['Pillar'] = bsm_pct_diff['Indicator'].map(pillars)
    bsm_pct_diff['Unit'] = bsm_pct_diff['Indicator'].map(metadata_units)
    bsm_pct_diff = bsm_pct_diff[cols_final_[0:32]]
#%%
    print('Step 39')
    bsm_abs_diff = bsm_abs_diff.reset_index()
    bsm_abs_diff['Indicator'] = 'Share of total space heat demand'
    bsm_abs_diff['Pillar'] = bsm_abs_diff['Indicator'].map(pillars)
    bsm_abs_diff['Unit'] = bsm_abs_diff['Indicator'].map(metadata_units)
    bsm_abs_diff = bsm_abs_diff[cols_final_[0:32]]

# %%
    # Append to pct_diff dataframe
    print('Step 40')
    abs_diff_ = abs_diff_.append(bsm_abs_diff).reset_index(drop=True)
```

```
    pct_diff_ = pct_diff_.append(bsm_pct_diff).reset_index(drop=True)
    # Warning: this will generate missing values
    # as the distributional impacts by dwelling archetype are only
    # available between 2022 and 2050

# %%
# -----------------------------------------------------------------------------
# ------------------------------ FINAL RESULTS --------------------------------
# -----------------------------------------------------------------------------
    print('Writing final results to a pickle file')
    # Write final results to pickle
    abs_diff_ = abs_diff_[(abs_diff_['Country'] != 'UK') & (abs_diff_['Country'] != 'EU27')]
    abs_diff_.to_pickle("Outputs/Absolute difference from the baseline.pkl")

    pct_diff_ = pct_diff_[(pct_diff_['Country'] != 'UK') & (pct_diff_['Country'] != 'EU27')]
    pct_diff_.to_pickle("Outputs/Percentage difference from the baseline.pkl")

    #checking a result: RH
    #print(final_dataset['Scenario']['Labour Productivity'].head())

    #%%
    # Carry out last few checks on pct_diff_

    pct_diff_.isnull().values.any()
    pct_diff_.iloc[0:1772].isnull().values.any()
    indicators_pct = list(set(pct_diff_['Indicator']))
    indicators_abs = list(set(abs_diff_['Indicator']))

    print('Checking the number of countries for each indicator')
    pct_diff_list = [g for _,g in pct_diff_.groupby('Indicator')]
    for i in pct_diff_list:
        country_list = len(list(set(i.loc[:,'Country'])))
        elem = sorted(list(set(i.loc[:,'Country'])))
```

```
            first_elem = sorted(list(set(pct_diff_list[0].loc[:,'Country'])))
            print(elem == first_elem)
            print(country_list)

        print('End of the script')

#%%

    print('End of the script')

    # Record end time of the program
    end = time.time()

    print("The time of execution of above program is :", (end-start))
```