

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

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*, Passenger 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.

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_fr_main.py
5 =====
6 Freight transport FTT module.
7 #####
8
9 This is the main file for FTT: Freight, which models technological
10 diffusion of freight vehicle types due to simulated consumer decision making.
11 Consumers compare the levelised cost of freight, which leads to changes in the
12 market shares of different technologies.
13
14 The outputs of this module include market shares, fuel use, and emissions.
15
16 Local library imports:
17
18     FTT: Freight functions:
19
20     - `get_lcof <ftt_fr_lcof.html>`__
21       Levelised cost calculation
22
23     Support functions:
24
25     - `divide <divide.html>`__
26       Bespoke element-wise divide which replaces divide-by-zeros with zeros
27     - `estimation <econometrics_functions.html>`__
28       Predict future values according to the estimated coefficients.
29
30 Functions included:
31     - solve
32       Main solution function for the module
33     - get_lcof
```

```
34     Calculate levelised cost of freight
35
36     """
37
38     from math import sqrt
39     import os
40     import copy
41     import sys
42     import warnings
43
44     # Third party imports
45     import pandas as pd
46     import numpy as np
47
48     # Local library imports
49     from support.divide import divide
50     from support.econometrics_functions import estimation
51
52     from ftt_fr_lcof import get_lcof
53
54     #Main function
55
56     def solve(data, time_lag, iter_lag, titles, histend, year, domain):
57         """
58         Main solution function for the module.
59         This function simulates investor decision making in the freight sector.
60         Levelised costs (from the lcof function) are taken and market shares
61         for each vehicle type are simulated to ensure demand is met.
62
63
64         Parameters
65         -----
66         data: dictionary of NumPy arrays
```

```
67     Model variables for the given year of solution
68     time_lag: type
69     Description
70     iter_lag: type
71     Description
72     titles: dictionary of lists
73     Dictionary containing all title classification
74     histend: dict of integers
75     Final year of historical data by variable
76     year: int
77     Current/active year of solution
78     Domain: dictionary of lists
79     Pairs variables to domains
80
81     Returns
82     -----
83     data: dictionary of NumPy arrays
84     Model variables for the given year of solution
85
86     """
87
88     # Factor used to create quarterly data from annual figures
89     no_it = 4
90     dt = 1 / no_it
91
92     #T_Scal = 5      # Time scaling factor used in the share dynamics
93
94     c6ti = {category: index for index, category in enumerate(titles['C6TI'])}
95
96     sector='freight'
97     #Creating variables
98
99     zjet=copy.deepcopy(data['ZJET'][0, :, :])
```

```
100     emis_corr = np.zeros([len(titles['RTI']), len(titles['FTTI'])])
101
102     if year <= histend["RVKZ"]:
103
104         #U (ZEWG) is number of vehicles by technology
105         data['ZEWG'][:, :, 0] = data['ZEWS'][:, :, 0]*data['RFLZ'][:, np.newaxis, 0, 0]
106
107     #
108     #
109     #     #I (ZEWY) is new sales, positive changes in U
110     #     data['ZEWY'][r, :, 0] = ((data['ZEWG'][r, :, 0] - data['ZEWG'][r, :, 0])/dt)*((data['ZEWG'][r, :, 0] - data
111     #     ['ZEWG'][r, :, 0])>0)
112
113     #
114     ##         for veh in range(len(titles['FTTI'])):
115     ##             for fuel in range(len(titles['JTI'])):
116     ##                 if titles['JTI'][fuel] == '5 Middle distillates' and data['ZJET'][0, veh, fuel] == 1: # Middle
117     ##                     distillates
118     ##                         # Mix with biofuels if there's a biofuel mandate
119     ##                         zjet[veh, fuel] = zjet[veh, fuel] * (1.0 - data['ZBFM'][r, 0, 0])
120     ##
121     ##                         # Emission correction factor
122     ##                         emis_corr[r, veh] = 1.0 - data['ZBFM'][r, 0, 0]
123     ##
124     ##                 elif titles['JTI'][fuel] == '11 Biofuels' and data['ZJET'][0, veh, fuel] == 1:
125     ##
126     ##                     zjet[veh, fuel] = data['ZJET'][0, veh, fuel] * data['ZBFM'][r, 0, 0]
127     #
128     #     data['ZJNJ'][r, :, 0] = (np.matmul(np.transpose(zjet), data['ZEVV'][r, :, 0]*\
129     #     data['ZCET'][r, :, c6ti['9 energy use (MJ/vkm)']]))/0.041868
130     #
```

```
131 #           #Emissions, E is ZEWE
132 #           data['ZEWE'][r,:,0]=data['ZEVV'][r,:,0]*data['ZCET'][r,:,c6ti['14 CO2Emissions (gCO2/km)']]*(1-data
    ['ZBFM'][r,0,0])/(1**6)
133 #
134 #
135 #           #Set cumulative capacities variable
136 #           data['ZEWV'][0,:,0]=data['ZCET'][0,:,c6ti['11 Cumulative seats']]
137
138
139 "Model Dynamics"
140
141 #Endogenous calculation starts here
142 if year > histend['RVKZ']:
143
144     data_dt = {}
145     data_dt['ZWIY'] = np.zeros([len(titles['RTI']), len(titles['VTTI']), 1])
146
147     for var in time_lag.keys():
148
149         if var.startswith("R"):
150
151             data_dt[var] = copy.deepcopy(time_lag[var])
152
153     for var in time_lag.keys():
154
155         if var.startswith("Z"):
156
157             data_dt[var] = copy.deepcopy(time_lag[var])
158
159     #find if there is a regulation and if it is exceeded
160
161     isReg = np.zeros([len(titles['RTI']), len(titles['FTTI'])])
162     division = np.zeros([len(titles['RTI']), len(titles['FTTI'])])
```

```
163     division = divide((data_dt['RVKZ'][:, :, 0] - data['ZREG'][:, :, 0]),
164                       data_dt['ZREG'][:, :, 0])
165     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
166     isReg[data['ZREG'][:, :, 0] == 0.0] = 1.0
167     isReg[data['ZREG'][:, :, 0] == -1.0] = 0.0
168
169
170     for t in range(1, no_it+1):
171         #Interpolations to avoid staircase profile
172
173         RTCO = time_lag['RZCO'][:, :, :] + (data['RZCO'][:, :, :] - time_lag['RZCO'][:, :, :]) * t * dt
174         FuT = time_lag['RTFZ0'][:, :, :] + (data['RTFZ0'][:, :, :] - time_lag['RTFZ0'][:, :, :]) * t * dt
175         #TJET = time_lag['ZJET'][:, :, :] + (data['ZJET'][:, :, :] - time_lag['ZJET'][:, :, :]) * t * dt
176         D = time_lag['RVKZ'][:, :, :] + (data['RVKZ'][:, :, :] - time_lag['RVKZ'][:, :, :]) * t * dt
177         Utot = time_lag['RFLZ'][:, :, :] + (data['RFLZ'][:, :, :] - time_lag['RFLZ'][:, :, :]) * t * dt
178         BFM = time_lag['ZBFM'][:, :, :] + (data['ZBFM'][:, :, :] - time_lag['ZBFM'][:, :, :]) * t * dt
179
180         for r in range(len(titles['RTI'])):
181
182             if D[r] == 0.0:
183                 continue
184
185             # DSik contains the change in shares
186             dSik = np.zeros([len(titles['FTTI']), len(titles['FTTI'])])
187
188             # F contains the preferences
189             F = np.ones([len(titles['FTTI']), len(titles['FTTI'])])*0.5
190
191             # -----
192             # Step 1: Endogenous EOL replacements
193             # -----
194             for b1 in range(len(titles['FTTI'])):
195
```



```

196         if not (data_dt['ZEWS'][r, b1, 0] > 0.0 and
197                 data_dt['ZTLL'][r, b1, 0] != 0.0 and
198                 data_dt['ZTDD'][r, b1, 0] != 0.0):
199             continue
200
201         S_i = data_dt['ZEWS'][r, b1, 0]
202
203         #Aki = 0.5 * data['IHEL'][r, b1, 0] / time_lag['IHUD'][r, b1, 0]
204
205         for b2 in range(b1):
206
207             if not (data_dt['ZEWS'][r, b2, 0] > 0.0 and
208                     data_dt['ZTLL'][r, b2, 0] != 0.0 and
209                     data_dt['ZTDD'][r, b2, 0] != 0.0):
210                 continue
211
212
213             S_k = data_dt['ZEWS'][r, b2, 0]
214
215             Aik = data['ZEWA'][0, b1, b2]
216             Aki = data['ZEWA'][0, b2, b1]
217
218             # Propagating width of variations in perceived costs
219             dFik = sqrt(2) * sqrt((data_dt['ZTDD'][r, b1, 0]*data_dt['ZTDD'][r, b1, 0] + data_dt
220                                     ['ZTDD'][r, b2, 0]*data_dt['ZTDD'][r, b2, 0]))
221
222             # Consumer preference incl. uncertainty
223             Fik = 0.5*(1+np.tanh(1.25*(data_dt['ZTLL'][r, b2, 0]-data_dt['ZTLL'][r, b1, 0])/dFik))
224             Fki = 1-Fik
225
226             # Preferences are then adjusted for regulations
227             F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
228                 + 0.5*(isReg[r, b1]*isReg[r, b2])

```

```

227 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])
228
229
230 #Runge-Kutta market share dynamiccs
231 k_1 = S_i*S_k * (Aik*F[b1, b2] - Aki*F[b2, b1])
232 k_2 = (S_i+dt*k_1/2)*(S_k-dt*k_1/2)* (Aik*F[b1, b2] - Aki*F[b2, b1])
233 k_3 = (S_i+dt*k_2/2)*(S_k-dt*k_2/2) * (Aik*F[b1, b2] - Aki*F[b2, b1])
234 k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2] - Aki*F[b2, b1])
235
236 #This method currently applies RK4 to the shares, but all other components of the equation ↗
      are calculated for the overall time step
237 #We must assume the the LCOE does not change significantly in a time step dt, so we can ↗
      focus on the shares.
238
239 dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6#*data['ZCEZ'][r,0,0]
240 dSik[b2, b1] = -dSik[b1, b2]
241
242 # Market share dynamics
243 # dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2] - Aki*F[b2,b1])*dt#*data['ZCEZ'][r,0,0]
244 # dSik[b2, b1] = -dSik[b1, b2]
245
246 #Check share changes sum to zero goes here, this is under time and region loop
247 #Also includes share equation
248 dSk = np.zeros([len(titles['FTTI'])])
249 data['ZEWS'][r, :, 0] = data_dt['ZEWS'][r, :, 0] + np.sum(dSik, axis=1) +dSk
250
251 if ~np.isclose(np.sum(data['ZEWS'][r, :, 0]), 1.0, atol=1e-5):
252     msg = ""Sector: {} - Region: {} - Year: {}
253     Sum of market shares do not add to 1.0 (instead: {})
254     """.format(sector, titles['RTI'][r], year, np.sum(data['ZEWS'][r, :, 0]))
255     warnings.warn(msg)
256

```

```
257     if np.any(data['ZEWS'][r, :, 0] < 0.0):
258         msg = ""Sector: {} - Region: {} - Year: {}
259         Negative market shares detected! Critical error!
260         """.format(sector, titles['RTI'][r], year)
261         warnings.warn(msg)
262
263
264     for r in range(len(titles['RTI'])):
265         #Copy over costs that dont change
266         data['ZCET'][:, :, 1:20] = data_dt['ZCET'][:, :, 1:20]
267
268         #G1 is Total service
269         #G1[r, :, 0]=D[r]/data['ZLOD'][r, 0, 0]
270
271         data['ZESG'][r, :, 0]=D[r, 0, 0]/data['ZLOD'][r, 0, 0]
272
273         #Sd is share difference between small and large trucks
274         data['ZESD'][r, 0, 0] = data['ZEWS'][r, 0, 0]+data['ZEWS'][r, 2, 0]+data['ZEWS'][r, 4, 0]+
275         data['ZEWS'][r, 6, 0]+data['ZEWS'][r, 8, 0]+data['ZEWS'][r, 10, 0]+data['ZEWS'][r, 12, 0]+
276         data['ZEWS'][r, 14, 0]+data['ZEWS'][r, 16, 0]+data['ZEWS'][r, 18, 0]
277
278         data['ZESD'][r, 1, 0] = 1 - data['ZESD'][r, 0, 0]
279
280
281         for x in range(0, 20, 2):
282             data['ZESA'][r, x, 0]=data['ZEWS'][r, x, 0]/data['ZESD'][r, 0, 0]
283             data['ZEVV'][r, x, 0]=data['ZESG'][r, x, 0]*data['ZESA'][r, x, 0]/(1-1/(data['ZSLR'][r, 0, 0]+1))
284             data['ZEST'][r, x, 0]=data['ZEVV'][r, x, 0]*data['ZLOD'][r, 1, 0]
285         for x in range(1, 21, 2):
286             data['ZESA'][r, x, 0]=data['ZEWS'][r, x, 0]/data['ZESD'][r, 1, 0]
287             data['ZEVV'][r, x, 0]=data['ZESG'][r, x, 0]*data['ZESA'][r, x, 0]/(1/(data['ZSLR'][r, 0, 0]+1))
288             data['ZEST'][r, x, 0]=data['ZEVV'][r, x, 0]*data['ZLOD'][r, 1, 0]
289
```

```
290     #T is total service generated by small trucks in MTKm
291
292     #This is number of trucks by technology
293     #data['ZEWG'][r,:,0] = data['ZEWS'][r,:,0]*data['RFLZ'][r,0,0]
294     data['ZEWG'][r,:,0] = data['ZEWS'][r,:,0]*Utot[r,0,0]
295     #Investment (sales) = new capacity created
296
297     veh_diff = data['ZEWG'][r,:,0] - data_dt['ZEWG'][r,:,0]
298     veh_dprctn = data_dt['ZEWG'][r,:,0] / data['ZCET'][r,:,c6ti['8 service lifetime (y)']]
299     data['ZEZY'][r,:,0] = np.where(veh_diff>0.0,
300                                   veh_diff + veh_dprctn,
301                                   veh_dprctn)
302
303
304
305     #Emissions
306     data['ZEWE'][r,:,0]=data['ZEVV'][r,:,0]*data['ZCET'][r,:,c6ti['14 CO2Emissions (gCO2/km)']]*(1-data_
307     ['ZBFM'][r,0,0])/(1E6)
308
309
310     zjet=copy.deepcopy(data['ZJET'][0, :, :])
311     for veh in range(len(titles['FTTI'])):
312         for fuel in range(len(titles['JTI'])):
313             if titles['JTI'][fuel] == '5 Middle distillates' and data['ZJET'][0, veh, fuel] ==1: # ↗
314                 Middle distillates
315
316                 # Mix with biofuels if there's a biofuel mandate
317                 zjet[veh, fuel] = zjet[veh, fuel] * (1.0 - data['ZBFM'][r, 0, 0])
318
319                 # Emission correction factor
320                 emis_corr[r, veh] = 1.0 - data['ZBFM'][r, 0, 0]
321
322     elif titles['JTI'][fuel] == '11 Biofuels' and data['ZJET'][0, veh, fuel] == 1:
```

```
321
322         zjet[veh, fuel] = data['ZJET'][0, veh, fuel] * data['ZBFM'][r, 0, 0]
323
324         #Convert TJ to ktoe, therefore divide by 0.041868
325         data['ZJNJ'][r, :, 0] = (np.matmul(np.transpose(zjet), data['ZEVV'][r, :, 0]*\
326             data['ZCET'][r, :, c6ti['9 energy use (MJ/vkm)']]))/0.041868
327
328
329         #Cumulative investment, not in region loop as it is global
330
331         bi = np.zeros((len(titles['RTI']),len(titles['FTTI'])))
332         for r in range(len(titles['RTI'])):
333             bi[r,:] = np.matmul(data['ZEWB'][0, :, :],data['ZEWY'][r, :, 0])
334         dw = np.sum(bi, axis=0)
335         data['ZEWV'][0,:,0] = data_dt['ZEWV'][0,:,0] + dw
336
337         #         data['ZCET'][:, :, c6ti['11 Cumulative seats']] = data_dt['ZCET'][:, :, c6ti['11 Cumulative seats']]
338         #         + np.sum(data['ZEWB'][0, :, :] * data['ZEWY'][:, :, 0], axis=1) * dt
339         #reopen region loop,
340
341         #learning curves and LCOF
342
343         for r in range(len(titles['RTI'])):
344             data['ZCET'][r, :, c6ti['1 Price of vehicles (USD/vehicle)']] = \
345                 data_dt['ZCET'][r, :, c6ti['1 Price of vehicles (USD/vehicle)']] \
346                 + data['ZLER'][0, :, 0] * ((data['ZEWV'][0, :, 0] - \
347                     data_dt['ZEWV'][0, :, 0]) / data['ZEWV'][0, :, 0]) * \
348                     data_dt['ZCET'][r, :, c6ti['1 Price of vehicles (USD/vehicle)']]
349
350
351         #Calculate total investment by technology in terms of truck purchases
352         for r in range(len(titles['RTI'])):
353             data['ZWIY'][r, :, 0] = data_dt['ZWIY'][r, :, 0] + \
```

```
354         data['ZEWY'][r, :, 0]*dt*data['ZCET'][r, :, c6ti['1 Price of vehicles (USD/vehicle)']]*1.263
355
356     #Calculate levelised cost again
357     data = get_lcof(data, titles)
358
359
360     #Update time loop variables:
361     for var in time_lag.keys():
362
363         if var.startswith("R"):
364
365             data_dt[var] = copy.deepcopy(time_lag[var])
366
367     for var in time_lag.keys():
368
369         if var.startswith("Z"):
370
371             data_dt[var] = copy.deepcopy(time_lag[var])
372
373
374     return data
375
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_h_main.py
5 =====
6 Residential heating sector FTT module.
7 #####
8
9 This is the main file for FTT: Heat, which models technological
10 diffusion of residential heating technologies due to simulated consumer decision making.
11 Consumers compare the **levelised cost of heat**, which leads to changes in the
12 market shares of different technologies.
13
14 The outputs of this module include changes in final energy demand and boiler sales.
15
16 Local library imports:
17
18     Support functions:
19
20     - `divide <divide.html>`__
21       Bespoke element-wise divide which replaces divide-by-zeros with zeros
22     - `estimation <econometrics_functions.html>`__
23       Predict future values according to the estimated coefficients.
24
25 Functions included:
26     - solve
27       Main solution function for the module
28     - get_lcoh
29       Calculate levelised cost of residential heating
30
31 """
32 # Standard library imports
33 from math import sqrt
```

```
34 import os
35 import copy
36 import sys
37 import warnings
38 import time
39
40 # Third party imports
41 import pandas as pd
42 import numpy as np
43
44 # Local library imports
45 from support.divide import divide
46 from support.econometrics_functions import estimation
47
48 # %% lcoh
49 # -----
50 # ----- LCOH function -----
51 # -----
52 def get_lcoh(data, titles):
53     """
54     Calculate levelized costs.
55
56     The function calculates the levelised cost of heat in 2014 Euros/kWh per
57     boiler type. It includes intangible costs (gamma values) and together
58     determines the investor preferences.
59     """
60     # Categories for the cost matrix (BHTC)
61     c4ti = {category: index for index, category in enumerate(titles['C4TI'])}
62
63     for r in range(len(titles['RTI'])):
64
65         # Cost matrix
66         bhtc = data['BHTC'][r, :, :]
```



```
67
68     # Boiler lifetime
69     lt = bhtc[:, c4ti['5 Lifetime']]
70     max_lt = int(np.max(lt))
71     lt_mat = np.linspace(np.zeros(len(titles['HTTI'])), max_lt-1,
72                          num=max_lt, axis=1, endpoint=True)
73     lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
74     mask = lt_mat < lt_max_mat
75     lt_mat = np.where(mask, lt_mat, 0)
76
77     # Capacity factor
78     cf = bhtc[:, c4ti['13 Capacity factor mean']], np.newaxis]
79
80     # Conversion efficiency
81     ce = bhtc[:, c4ti['9 Conversion efficiency']], np.newaxis]
82
83     # Discount rate
84     # dr = bhtc[6]
85     dr = bhtc[:, c4ti['8 Discount rate']], np.newaxis]
86
87     # Initialise the levelised cost components
88     # Average investment cost
89     it = np.zeros([len(titles['HTTI']), int(max_lt)])
90     it[:, 0, np.newaxis] = divide(bhtc[:, c4ti['1 Investment cost mean']], np.newaxis], (cf*1000))
91
92     # Standard deviation of investment cost
93     dit = np.zeros([len(titles['HTTI']), int(max_lt)])
94     dit[:, 0, np.newaxis] = divide(bhtc[:, c4ti['2 Investment cost SD']], np.newaxis], (cf*1000))
95
96     # Upfront subsidy/tax at purchase time
97     st = np.zeros([len(titles['HTTI']), int(max_lt)])
98     st[:, 0, np.newaxis] = it[:, 0, np.newaxis] * data['HTVS'][r, :, 0, np.newaxis]
99
```

```
100     # Average fuel costs
101     ft = np.ones([len(titles['HTTI']), int(max_lt)])
102     ft = ft * divide(data['HEWP'][r, :, 0, np.newaxis], ce)
103     #ft = np.where(mask, ft, 0)
104
105     # Standard deviation of fuel costs
106     dft = np.ones([len(titles['HTTI']), int(max_lt)])
107     dft = ft * divide(bhtc[:, c4ti['11 Fuel cost SD'], np.newaxis], ce)
108     dft = np.where(mask, dft, 0)
109
110     # Fuel tax costs
111     fft = np.ones([len(titles['HTTI']), int(max_lt)])
112     fft = ft * data['HTRT'][r, :, 0, np.newaxis]
113     fft = np.where(mask, fft, 0)
114
115     # Average operation & maintenance cost
116     omt = np.ones([len(titles['HTTI']), int(max_lt)])
117     omt = omt * divide(bhtc[:, c4ti['3 O&M cost mean'], np.newaxis], (cf*1000))
118     omt = np.where(mask, omt, 0)
119
120     # Standard deviation of operation & maintenance cost
121     domt = np.ones([len(titles['HTTI']), int(max_lt)])
122     domt = omt * divide(bhtc[:, c4ti['4 O&M cost SD'], np.newaxis], (cf*1000))
123     domt = np.where(mask, domt, 0)
124
125     # Feed-in-Tariffs
126     fit = np.ones([len(titles['HTTI']), int(max_lt)])
127     fit = fit * data['HEFI'][r, :, 0, np.newaxis]
128     fit = np.where(mask, fit, 0)
129
130     # Net present value calculations
131     # Discount rate
132     denominator = (1+dr)**lt_mat
```

```
133
134     # 1-Expenses
135     # 1.1-Without policy costs
136     npv_expenses1 = (it+ft+omt)/denominator
137     # 1.2-With policy costs
138     npv_expenses2 = (it+st+ft+fft+omt-fit)/denominator
139     # 1.3-Only policy costs
140     npv_expenses3 = (st+fft-fit)/denominator
141     # 2-Utility
142     npv_utility = 1/denominator
143     #Remove 1s for tech with small lifetime than max
144     npv_utility[npv_utility==1] = 0
145     npv_utility[:,0] = 1
146     # 3-Standard deviation (propagation of error)
147     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
148
149     # 1-levelised cost variants in $/pkm
150     # 1.1-Bare LCOT
151     lcoh = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
152     # 1.2-LCOT including policy costs
153     tlcoh = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)
154     # 1.3-LCOT of policy costs
155     lcoh_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
156     # Standard deviation of LCOT
157     dlcoh = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
158
159     # LCOT augmented with non-pecuniary costs
160     tlcohg = tlcoh + data['HGAM'][r, :, 0]
161
162     # Pay-back thresholds
163     # TODO: Titles for FTT:Heat's cost categories are wrong
164     pb = bhtc[:, c4ti['16 Empty']]
165     dpb = bhtc[:, c4ti['17 Empty']]
```

```
166
167     # Marginal costs of existing units
168     tmc = ft[:, 0] + omt[:, 0] + fft[:, 0] - fit[:, 0]
169     dtmc = np.sqrt(dft[:, 0] + domt[:, 0])
170
171     # Total pay-back costs of potential alternatives
172     tpb = tmc + (it[:, 0] + st[:, 0])/pb
173     dtpb = np.sqrt(dft[:, 0]**2 + domt[:, 0]**2 +
174                 divide(dit[:, 0]**2, pb**2) +
175                 divide(it[:, 0]**2, pb**4)*dpb**2)
176
177     # Add gamma values
178     tmc = tmc + data['HGAM'][r, :, 0]
179     tpb = tpb + data['HGAM'][r, :, 0]
180
181     # Pass to variables that are stored outside.
182     data['HEWC'][r, :, 0] = lcoh           # The real bare LCOH without taxes
183     data['HETC'][r, :, 0] = tlcoh        # The real bare LCOH with taxes
184     data['HGC1'][r, :, 0] = tlcohg      # As seen by consumer (generalised cost)
185     data['HWCD'][r, :, 0] = dlcoh       # Variation on the LCOH distribution
186     data['HGC2'][r, :, 0] = tmc         # Total marginal costs
187     data['HGD2'][r, :, 0] = dtmc       # SD of Total marginal costs
188     data['HGC3'][r, :, 0] = tpb        # Total payback costs
189     data['HGD3'][r, :, 0] = dtpb       # SD of Total payback costs
190
191     return data
192
193 # -----
194 # ----- Main -----
195 # -----
196 def solve(data, time_lag, iter_lag, titles, histend, year, specs):
197     """
198     Main solution function for the module.
```

```
199
200     Add an extended description in the future.
201
202     Parameters
203     -----
204     data: dictionary of NumPy arrays
205           Model variables for the given year of solution
206     time_lag: type
207           Description
208     iter_lag: type
209           Description
210     titles: dictionary of lists
211           Dictionary containing all title classification
212     histend: dict of integers
213           Final year of historical data by variable
214     year: int
215           Current/active year of solution
216     specs: dictionary of NumPy arrays
217           Function specifications for each region and module
218
219     Returns
220     -----
221     data: dictionary of NumPy arrays
222           Model variables for the given year of solution
223
224     Notes
225     -----
226     This function should be broken up into more elements in development.
227     """
228
229     # Categories for the cost matrix (BHTC)
230     c4ti = {category: index for index, category in enumerate(titles['C4TI'])}
231     jti = {category: index for index, category in enumerate(titles['JTI'])}
```

```
232
233     fuelvars = ['FR_1', 'FR_2', 'FR_3', 'FR_4', 'FR_5', 'FR_6',
234                'FR_7', 'FR_8', 'FR_9', 'FR_10', 'FR_11', 'FR_12']
235
236     sector = 'residential'
237     #sector_index = titles['Sectors_short'].index(sector)
238
239
240
241     # %% First initialise if necessary
242     # Initialise in case of stock solution specification
243     #if np.any(specs[sector]) < 5:
244
245     # Up to the last year of historical useful energy demand by boiler
246     if year <= histend['HEWF']:
247     # Historical data ends in 2014, so we need to initialise data
248     # when it's 2015 to make sure the model runs.
249     # At some point we need to change the start year of the simulation and
250     # Change the timelines in ALL of the csv's
251     #     if year == 2015:
252
253         for r in range(len(titles['RTI'])):
254
255             # Useful energy demand by boilers
256             # The historical data contains final energy demand
257             data['HEWG'][r, :, 0] = data['HEWF'][r, :, 0] * data['BHTC'][r, :, c4ti["9 Conversion efficiency"]]
258
259             # Total useful heat demand
260             # This is the demand driver for FTT:Heat
261             data['RHUD'][r, 0, 0] = np.sum(data['HEWG'][r, :, 0])
262
263
264
```

```
265
266
267     if data['RHUD'][r, 0, 0] > 0.0:
268
269         # Market shares (based on useful energy demand)
270         data['HEWS'][r, :, 0] = data['HEWG'][r, :, 0] / data['RHUD'][r, 0, 0]
271
272         # CORRECTION TO MARKET SHARES
273         # Sometimes historical market shares do not add up to 1.0
274         if (~np.isclose(np.sum(data['HEWS'][r, :, 0]), 0.0, atol=1e-9)
275             and np.sum(data['HEWS'][r, :, 0]) > 0.0):
276             data['HEWS'][r, :, 0] = np.divide(data['HEWS'][r, :, 0],
277                                               np.sum(data['HEWS'][r, :, 0]))
278
279         # Capacity by boiler
280         data['HEWK'][:, :, 0] = divide(data['HEWG'][:, :, 0],
281                                     data['BHTC'][:, :, c4ti["13 Capacity factor mean"]])/1000
282
283         # Emissions
284         # TODO: Cost titles are wrong
285         data['HEWE'][r, :, 0] = data['HEWF'][r, :, 0] * data['BHTC'][r, :, c4ti["15 Empty"]]
286
287         # Final energy demand by energy carrier
288         for fuel in range(len(titles['JTI'])):
289             # Fuel use for heating
290             data['HJHF'][r, fuel, 0] = np.sum(data['HEWF'][r, :, 0] * data['HJET'][0, :, fuel])
291             # Fuel use for total residential sector
292             if data['HJFC'][r, fuel, 0] > 0.0:
293                 data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]
294
295     if year == histend['HEWF']:
296         # Historical data ends in 2014, so we need to initialise data
297         # when it's 2015 to make sure the model runs.
298         # At some point we need to change the start year of the simulation and
299         # Change the timelines in ALL of the csv's
```

```
298     # Endogenous price rates
299     endog_price_rate = divide(data['MEWP'][:, :, 0],
300                             time_lag['MEWP'][:, :, 0])
301
302     # If switch is set to 1, then an exogenous price rate is used
303     # Otherwise, the price rates are set to endogenous
304
305     #data['HFPR'][:, :, 0] = copy.deepcopy(data['HFFC'][:, :, 0])
306
307     # Now transform price rates by fuel to price rates by boiler
308     data['HEWP'][:, :, 0] = np.matmul(data['HFFC'][:, :, 0], data['HJET'][0, :, :].T)
309
310     for r in range(len(titles['RTI'])):
311
312         # Sales are the difference between fleet sizes and the addition of scrapped vehicles
313         for tech in range(len(titles['HTTI'])):
314             if (data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]) > 0:
315                 data['HEWI'][r, tech, 0] = data['HEWK'][r, tech, 0] - time_lag['HEWK'][r, tech, 0]
316
317         data['HEWI'][r, :, 0] += divide(data['HEWK'][r, :, 0], data['HETR'][r, :, 0])
318
319         # Final energy demand by energy carrier
320         for fuel in range(len(titles['JTI'])):
321
322             # Fuel use for heating
323             data['HJHF'][r, fuel, 0] = np.sum(data['HEWF'][r, :, 0] * data['HJET'][0, :, fuel])
324
325             # Fuel use for total residential sector #HFUX is missing
326             if data['HJFC'][r, fuel, 0] > 0.0:
327                 data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]
328
329     # Calculate the LCOT for each vehicle type.
330     # Call the function
```



```
331     data = get_lcoh(data, titles)
332 # %% Simulation of stock and energy specs
333 #     t0 = time.time()
334 # Stock based solutions first
335 #     if np.any(specs[sector] < 5):
336
337 # Endogenous calculation takes over from here
338 if year > histend['HEWF']:
339
340     # Create a local dictionary for timeloop variables
341     # It contains values between timeloop iterations in the FTT core
342     data_dt = {}
343
344     # First, fill the time loop variables with the their lagged equivalents
345     for var in time_lag.keys():
346         data_dt[var] = copy.deepcopy(time_lag[var])
347
348     # Create the regulation variable
349     isReg = np.zeros([len(titles['RTI']), len(titles['HTTI'])])
350     division = np.zeros([len(titles['RTI']), len(titles['HTTI'])])
351     division = divide((data_dt['HEWS'][:, :, 0] - data['HREG'][:, :, 0]),
352                     data_dt['HREG'][:, :, 0])
353     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
354     isReg[data['HREG'][:, :, 0] == 0.0] = 1.0
355     isReg[data['HREG'][:, :, 0] == -1.0] = 0.0
356
357     # Factor used to create quarterly data from annual figures
358     no_it = 1
359     dt = 1 / no_it
360
361     ##### Computing new shares #####
362
363     #Start the computation of shares
```

```
364     for t in range(1, no_it+1):
365
366         # Interpolate to prevent staircase profile.
367         rhudt = time_lag['RHUD'][:, :, :] + (data['RHUD'][:, :, :] - time_lag['RHUD'][:, :, :]) * t * dt
368
369         for r in range(len(titles['RTI'])):
370
371             if rhudt[r] == 0.0:
372                 continue
373
374             ##### FTT #####
375 #                 t3 = time.time()
376 #                 print("Solving {}".format(titles["RTI"][r]))
377 #                 # Initialise variables related to market share dynamics
378 #                 # DSik contains the change in shares
379                 dSik = np.zeros([len(titles['HTTI']), len(titles['HTTI'])])
380
381                 # F contains the preferences
382                 F = np.ones([len(titles['HTTI']), len(titles['HTTI'])])*0.5
383
384                 # -----
385                 # Step 1: Endogenous EOL replacements
386                 # -----
387                 for b1 in range(len(titles['HTTI'])):
388
389                     if not (data_dt['HEWS'][r, b1, 0] > 0.0 and
390                             data_dt['HGC1'][r, b1, 0] != 0.0 and
391                             data_dt['HWCD'][r, b1, 0] != 0.0):
392                         continue
393
394                     S_i = data_dt['HEWS'][r, b1, 0]
395
396                     for b2 in range(b1):
```

```
397
398         if not (data_dt['HEWS'][r, b2, 0] > 0.0 and
399                 data_dt['HGC1'][r, b2, 0] != 0.0 and
400                 data_dt['HWCD'][r, b2, 0] != 0.0):
401             continue
402
403         S_k = data_dt['HEWS'][r, b2, 0]
404
405         # Propagating width of variations in perceived costs
406         dFik = sqrt(2) * sqrt((data_dt['HWCD'][r, b1, 0]*data_dt['HWCD'][r, b1, 0] + data_dt
407                                ['HWCD'][r, b2, 0]*data_dt['HWCD'][r, b2, 0]))
408
409         # Consumer preference incl. uncertainty
410         Fik = 0.5*(1+np.tanh(1.25*(data_dt['HGC1'][r, b2, 0]-data_dt['HGC1'][r, b1, 0])/dFik))
411
412         # Preferences are then adjusted for regulations
413         F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
414         + 0.5*(isReg[r, b1]*isReg[r, b2])
415         F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
416                                [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
417
418         #Runge-Kutta market share dynamiccs
419         k_1 = S_i*S_k * (data['HEWA'][0,b1, b2]*F[b1,b2]*data['HETR'][r,b2, 0]- data['HEWA'][0,b2,
420                                b1]*F[b2,b1]*data['HETR'][r,b1, 0])
421         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,
422                                0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
423         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,
424                                0]- data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
425         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]-
426                                data['HEWA'][0,b2, b1]*F[b2,b1]*data['HETR'][r,b1, 0])
427
428         # Market share dynamics
429         #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
423     dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
424     dSik[b2, b1] = -dSik[b1, b2]
425
426     # -----
427     # Step 2: Endogenous premature replacements
428     # -----
429     # Initialise variables related to market share dynamics
430     # DSik contains the change in shares
431     dSEik = np.zeros([len(titles['HTTI']), len(titles['HTTI'])])
432
433     # F contains the preferences
434     FE = np.ones([len(titles['HTTI']), len(titles['HTTI'])])*0.5
435
436     # Intermediate shares: add the EoL effects before continuing
437     # intermediate_shares = data_dt['HEWS'][r, :, 0] + np.sum(dSik, axis=1)
438
439     # Scrappage rate
440     SR = divide(np.ones([len(titles['HTTI'])]),
441                data['BHTC'][r, :, c4ti["16 Empty"]]) - data['HETR'][r, :, 0]
442     SR = np.where(SR<0.0, 0.0, SR)
443
444     for b1 in range(len(titles['HTTI'])):
445
446         if not (data_dt['HEWS'][r, b1, 0] > 0.0 and
447                data_dt['HGC2'][r, b1, 0] != 0.0 and
448                data_dt['HGD2'][r, b1, 0] != 0.0 and
449                data_dt['HGC3'][r, b1, 0] != 0.0 and
450                data_dt['HGD3'][r, b1, 0] != 0.0):
451             continue
452
453     SE_i = data_dt['HEWS'][r, b1, 0]
454
```

```
455         for b2 in range(b1):
456
457             if not (data_dt['HEWS'][r, b2, 0] > 0.0 and
458                   data_dt['HGC2'][r, b2, 0] != 0.0 and
459                   data_dt['HGD2'][r, b2, 0] != 0.0 and
460                   data_dt['HGC3'][r, b2, 0] != 0.0 and
461                   data_dt['HGD3'][r, b2, 0] != 0.0):
462                 continue
463
464             SE_k = data_dt['HEWS'][r, b2, 0]
465
466             # NOTE: Premature replacements are optional for
467             # consumers. It is possible that NO premature
468             # replacements take place
469
470             # Propagating width of variations in perceived costs
471             dFEik = sqrt(2) * sqrt((data_dt['HGD3'][r, b1, 0]*data_dt['HGD3'][r, b1, 0] + data_dt
472                                     ['HGD2'][r, b2, 0]*data_dt['HGD2'][r, b2, 0]))
473
474             dFEki = sqrt(2) * sqrt((data_dt['HGD2'][r, b1, 0]*data_dt['HGD2'][r, b1, 0] + data_dt
475                                     ['HGC3'][r, b2, 0]*data_dt['HGC3'][r, b2, 0]))
476
477             # Consumer preference incl. uncertainty
478             FEik = 0.5*(1+np.tanh(1.25*(data_dt['HGC2'][r, b2, 0]-data_dt['HGC3'][r, b1, 0])/dFEik))
479             FEki = 0.5*(1+np.tanh(1.25*(data_dt['HGC2'][r, b1, 0]-data_dt['HGC3'][r, b2, 0])/dFEki))
480
481             # Preferences are then adjusted for regulations
482             FE[b1, b2] = FEik*(1.0-isReg[r, b1])
483             FE[b2, b1] = FEki*(1.0-isReg[r, b2])
484
485             # Market share dynamics
486             dSEik[b1, b2] = SE_i*SE_k* (data['HEWA'][0,b1, b2]*SR[b2]*FE[b1,b2] - data['HEWA'][0,b2,
487                                     b1]*SR[b1]*FE[b2,b1])*dt
488             dSEik[b2, b1] = -dSEik[b1, b2]
```

```
485
486         # TODO: Calculate additional EOL values
487
488         # -----
489         # Step 3: Exogenous sales additions
490         # -----
491         # Add in exogenous sales figures. These are blended with
492         # endogenous result! Note that it's different from the
493         # ExogSales specification!
494         Utot = rhudt[r]
495         dSk = np.zeros([len(titles['HTTI'])])
496         dUk = np.zeros([len(titles['HTTI'])])
497         dUkTK = np.zeros([len(titles['HTTI'])])
498         dUkREG = np.zeros([len(titles['HTTI'])])
499
500
501         # Check that exogenous share changes add to zero
502         dUkTK = data['HWSA'][r, :, 0]
503         if (data['HWSA'][r, :, 0].sum() > 0.0):
504             dUkTK[0] = dUkTK[0] - data['HWSA'][r, :, 0].sum()
505
506         # Correct for regulations #TODO Does this actually make sense?
507
508         if time_lag['RHUD'][r, 0, 0] > 0.0 and rhudt[r] > 0.0 and (rhudt[r] - time_lag['RHUD'][r, 0, 0]) > ↗
509             0.0:
510                 dUkREG = -data_dt['HEWG'][r, :, 0] * ( rhudt[r] - time_lag['RHUD'][r, 0, 0]) /
511                     time_lag['RHUD'][r, 0, 0] * isReg[r, :].reshape([len(titles['HTTI'])])
512
513
514         # Sum effect of exogenous sales additions (if any) with
515         # effect of regulations
516         dUk = dUkREG
```

```
517         dUtot = np.sum(dUk)
518
519         # Convert to market shares and make sure sum is zero
520         # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
521         dSk = np.divide(dUk, Utot) - data_dt['HEWG'][r, :, 0]*np.divide(dUtot, (Utot*Utot)) + dUkTK
522
523 #         soel = np.sum(dSik, axis=1)
524 #         st_1 = data_dt['TP_MS'][r, :, 0]
525 # New market shares
526 # Implement check that market shares sum to 1
527 #         print(np.sum(dSik, axis=1))
528 data['HEWS'][r, :, 0] = data_dt['HEWS'][r, :, 0] + np.sum(dSik, axis=1) + np.sum(dSEik, axis=1) + ↗
        dSk[0, :]
529
530 if ~np.isclose(np.sum(data['HEWS'][r, :, 0]), 1.0, atol=1e-2):
531     msg = ""Sector: {} - Region: {} - Year: {}
532     Sum of market shares do not add to 1.0 (instead: {})
533     """.format(sector, titles['RTI'][r], year, np.sum(data['HEWS'][r, :, 0]))
534     warnings.warn(msg)
535
536 if np.any(data['HEWS'][r, :, 0] < 0.0):
537     msg = ""Sector: {} - Region: {} - Year: {}
538     Negative market shares detected! Critical error!
539     """.format(sector, titles['RTI'][r], year)
540     warnings.warn(msg)
541 #         t4 = time.time()
542 #         print("Share equation takes {}".format(t4-t3))
543
544 ##### Update variables #####
545 # Useful heat by boiler
546 data['HEWG'][:, :, 0] = data['HEWS'][:, :, 0] * rhudt[:, 0, 0, np.newaxis]
547
548 # Final energy by boiler
```



```
582 # Final energy demand for heating purposes
583 data['HJHF'][:, :, 0] = np.matmul(data['HEWF'][:, :, 0], data['HJET'][0, :, :])
584
585 # Final energy demand of the residential sector (incl. non-heat)
586 # For the time being, this is calculated as a simply scale-up
587 for fuel in range(len(titles['JTI'])):
588     if data['HJFC'][r, fuel, 0] > 0.0:
589         data['HJEF'][r, fuel, 0] = data['HJHF'][r, fuel, 0] / data['HJFC'][r, fuel, 0]
590
591 ##### Learning-by-doing #####
592
593 # Cumulative global learning
594 # Using a technological spill-over matrix (HEWB) together with capacity
595 # additions (HEWI) we can estimate total global spillover of similar
596 # technologies
597 bi = np.zeros((len(titles['RTI']),len(titles['HTTI'])))
598 for r in range(len(titles['RTI'])):
599     bi[r,:] = np.matmul(data['HEWB'][0, :, :],data['HEWI'][r, :, 0])
600 dw = np.sum(bi, axis=0)*dt
601
602
603 # Cumulative capacity incl. learning spill-over effects
604 data['HEWW'][0, :, 0] = data_dt['HEWW'][0, :, 0] + dw
605
606 # Copy over the technology cost categories that do not change (all except prices which are updated
607 data['BHTC'] = copy.deepcopy(data_dt['BHTC'])
608
609 # Learning-by-doing effects on investment
610 for b in range(len(titles['HTTI'])):
611
612     if data['HEWW'][0, b, 0] > 0.1:
613
```

```
614 data['BHTC'][:, b, c4ti['1 Investment cost mean']] = data_dt['BHTC'][:, b, c4ti['1 Investment  
cost mean']] * \  
615 (1.0 + data['BHTC'][:, b, c4ti['7  
Learning rate']] * dw[b]/data['HEWW'][0, b, 0])  
616  
617  
618  
619 ##### Final output #####  
620  
621 # Get LCOT  
622 if t ==1:  
623  
624 data = get_lcoh(data, titles)  
625  
626 # Update time loop variables:  
627 for var in data_dt.keys():  
628  
629 data_dt[var] = copy.deepcopy(data[var])  
630  
631  
632 return data  
633
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_chi_main.py
5 =====
6 Industrial chemical sector FTT module.
7 #####
8
9
10 This is the main file for FTT: Industrial Heat - CHI, which models technological
11 diffusion of industrial heat processes within the chemical sector due
12 to simulated investor decision making. Investors compare the **levelised cost of
13 industrial heat**, which leads to changes in the market shares of different technologies.
14
15 The outputs of this module include changes in final energy demand and emissions due
16 chemical heat processes for the EU28.
17
18 Local library imports:
19
20     Support functions:
21
22     - `divide <divide.html>`__
23         Bespoke element-wise divide which replaces divide-by-zeros with zeros
24
25 Functions included:
26
27     - solve
28         Main solution function for the module
29     - get_lcoih
30         Calculates the levelised cost of industrial heat
31
32 """
33 # Standard library imports
```

```

34 from math import sqrt
35 import os
36 import copy
37 import sys
38 import warnings
39 import time
40
41 # Third party imports
42 import pandas as pd
43 import numpy as np
44
45 # Local library imports
46 from support.divide import divide
47 #from support.econometrics_functions import estimation
48
49 # %% lcoh
50 # -----
51 # ----- LCOH function -----
52 # -----
53 def get_lcoih(data, titles, year):
54     """
55     Calculate levelized costs.
56
57     The function calculates the levelised cost of industrial heat in 2019 Euros
58     It includes intangible costs (gamma values) and together
59     determines the investor preferences.
60
61     Parameters
62     -----
63     data: dictionary
64         Data is a container that holds all cross-sectional (of time) for all
65         variables. Variable names are keys and the values are 3D NumPy arrays.
66     titles: dictionary

```

```
67     Titles is a container of all permissible dimension titles of the model.
68
69     Returns
70     -----
71     data: dictionary
72         Data is a container that holds all cross-sectional (of time) data for
73         all variables.
74         Variable names are keys and the values are 3D NumPy arrays.
75         The values inside the container are updated and returned to the main
76         routine.
77
78     Notes
79     -----
80     Additional notes if required.
81     """
82
83     # Categories for the cost matrix (BIC1)
84     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
85
86     for r in range(len(titles['RTI'])):
87         if data['IUD1'][r, :, 0].sum(axis=0)==0:
88             continue
89
90
91         lt = data['BIC1'][r, :, ctti['5 Lifetime (years)']]
92         max_lt = int(np.max(lt))
93         lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
94                             num=max_lt, axis=1, endpoint=True)
95         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
96         mask = lt_mat < lt_max_mat
97         lt_mat = np.where(mask, lt_mat, 0)
98
99
```

```
100     # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
101     cf = data['BIC1'][r,:, ctti['13 Capacity factor mean']], np.newaxis]
102
103     #conversion efficiency
104     ce = data['BIC1'][r,:, ctti['9 Conversion efficiency']], np.newaxis]
105
106     # Trap for very low CF
107     cf[cf<0.000001] = 0.000001
108
109     # Factor to transfer cost components in terms of capacity to generation
110     conv = 1/(cf)/8766 #number of hours in a year
111
112     # Discount rate
113     # dr = BIC1[6]
114     dr = data['BIC1'][r,:, ctti['8 Discount rate']], np.newaxis]
115
116     # Initialise the levelised cost components
117     # Average investment cost
118     it = np.zeros([len(titles['ITTI']), int(max_lt)])
119     it[:, 0, np.newaxis] = data['BIC1'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis]*
120         (1000000)*conv
121
122
123     # Standard deviation of investment cost
124     dit = np.zeros([len(titles['ITTI']), int(max_lt)])
125     dit[:, 0, np.newaxis] = data['BIC1'][r,:, ctti['2 Investment cost SD']], np.newaxis] *(1000000)*conv
126
127
128     # Subsidies as a percentage of investment cost
129     st = np.zeros([len(titles['ITTI']), int(max_lt)])
130     st[:, 0, np.newaxis] = (data['BIC1'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis]
131         * data['ISB1'][r, :, 0,np.newaxis] *conv)*(1000000)
```

```
132
133
134     # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
135     ft = np.ones([len(titles['ITTI']), int(max_lt)])
136     ft = ft * data['BIC1'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
137     ft = np.where(mask, ft, 0)
138
139     # Standard deviation of fuel costs
140     dft = np.ones([len(titles['ITTI']), int(max_lt)])
141     dft = dft * data['BIC1'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
142     dft = np.where(mask, dft, 0)
143
144     #fuel tax/subsidies
145     #fft = np.ones([len(titles['ITTI']), int(max_lt)])
146 #     fft = ft * data['PG_FUELTAX'][r, :, :]
147 #     fft = np.where(lt_mask, ft, 0)
148
149     # Fixed operation & maintenance cost - variable O&M available but not included
150     omt = np.ones([len(titles['ITTI']), int(max_lt)])
151     omt = omt * data['BIC1'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) ↗
152     # in a year
153     omt = np.where(mask, omt, 0)
154
155     # Standard deviation of operation & maintenance cost
156     domt = np.ones([len(titles['ITTI']), int(max_lt)])
157     domt = domt * data['BIC1'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
158     domt = np.where(mask, domt, 0)
159
160     # Net present value calculations
161     # Discount rate
162     denominator = (1+dr)**(lt_mat)
163
164     # 1-Expenses
```

```
164     # 1.1-Without policy costs
165     npv_expenses1 = (it+ft+omt)/denominator
166     # 1.2-With policy costs
167     npv_expenses2 = (it+st+ft+omt)/denominator
168     # 1.3-Only policy costs
169     #npv_expenses3 = (st+fft-fit)/denominator
170     # 2-Utility
171     npv_utility = 1/denominator
172     #Remove 1s for tech with small lifetime than max
173     npv_utility[npv_utility==1] = 0
174     npv_utility[:,0] = 1
175
176     # 3-Standard deviation (propagation of error)
177     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
178
179     # 1-levelised cost variants in $/pkm
180     # 1.1-Bare LCOT
181
182     lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
183
184     # 1.2-LCOT including policy costs
185     tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)+data['IEFI'][r, :, 0]
186     # 1.3 LCOE excluding policy, including co2 price
187     #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
188     # 1.3-LCOT of policy costs
189     # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
190     # Standard deviation of LCOT
191     dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
192
193     # LCOE augmented with gamma values, no gamma values yet
194     tlcoeg = tlcoe+data['IAM1'][r, :, 0]
195
196     # Pass to variables that are stored outside.
```



```
197     data['ILC1'][r, :, 0] = lcoe           # The real bare LCOT without taxes (euros/mwh)
198     #data['IHLT'][r, :, 0] = tlcoe         # The real bare LCOE with taxes
199     data['ILG1'][r, :, 0] = tlcoeg        # As seen by consumer (generalised cost)
200     data['ILD1'][r, :, 0] = dlcoe         # Variation on the LCOT distribution
201
202     return data
203
204
205
206 # %% main function
207 # -----
208 # ----- Main -----
209 # -----
210 def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
211     """
212
213     Main solution function for the module.
214
215     Simulates investor decision making.
216
217     Parameters
218     -----
219     data: dictionary of NumPy arrays
220         Model variables for the given year of solution
221     time_lag: type
222         Description
223     iter_lag: type
224         Description
225     titles: dictionary of lists
226         Dictionary containing all title classification
227     histend: dict of integers
228         Final year of historical data by variable
229     year: int
```



```
263     division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
264     division = divide((data_dt['IWK1'][:, :, 0] - data['IRG1'][:, :, 0]),
265                     data_dt['IRG1'][:, :, 0])
266     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
267     isReg[data['IRG1'][:, :, 0] == 0.0] = 1.0
268     isReg[data['IRG1'][:, :, 0] == -1.0] = 0.0
269
270
271     # Factor used to create quarterly data from annual figures
272     no_it = 4
273     dt = 1 / no_it
274     kappa = 10 #tech substitution constant
275
276     ##### Computing total useful energy demand #####
277
278     IUD1tot = data['IUD1'][:, :, 0].sum(axis=1)
279
280     #Start the computation of shares
281     for t in range(1, no_it+1):
282
283         # Interpolate to prevent staircase profile.
284         #Time lagged UED plus change in UED * (no of iterations) * dt
285
286         IUD1t = time_lag['IUD1'][:, :, 0].sum(axis=1) + (IUD1tot - time_lag['IUD1'][:, :, 0].sum(axis=1)) * t * dt
287
288         for r in range(len(titles['RTI'])):
289
290             if IUD1t[r] == 0.0:
291                 continue
292
293
294
```

```
295 ##### FTT #####
296
297 # DSik contains the change in shares
298 dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])
299
300 # F contains the preferences
301 F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5
302
303 # Market share constraints
304 Gijmax = np.ones(len(titles['ITTI']))
305 #Gijmin = np.ones((t2ti))
306
307 # -----
308 # Step 1: Endogenous EOL replacements
309 # -----
310 for b1 in range(len(titles['ITTI'])):
311
312     if not (data_dt['IWS1'][r, b1, 0] > 0.0 and
313            data_dt['ILG1'][r, b1, 0] != 0.0 and
314            data_dt['ILD1'][r, b1, 0] != 0.0):
315         continue
316
317     #TODO: create market share constraints
318     Gijmax[b1] = np.tanh(1.25*(data_dt['ISC1'][0, b1, 0] - data_dt['IWS1'][r, b1, 0])/0.1)
319     #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)
320
321
322
323     S_i = data_dt['IWS1'][r, b1, 0]
324
325
326     for b2 in range(b1):
327
```

```
328         if not (data_dt['IWS1'][r, b2, 0] > 0.0 and
329                 data_dt['ILG1'][r, b2, 0] != 0.0 and
330                 data_dt['ILD1'][r, b2, 0] != 0.0):
331             continue
332
333         S_k = data_dt['IWS1'][r,b2, 0]
334         Aik = data['IWA1'][0,b1 , b2]*kappa
335         Aki = data['IWA1'][0,b2, b1]*kappa
336
337         # Propagating width of variations in perceived costs
338         dFik = sqrt(2) * sqrt((data_dt['ILD1'][r, b1, 0]*data_dt['ILD1'][r, b1, 0] + data_dt
339                                ['ILD1'][r, b2, 0]*data_dt['ILD1'][r, b2, 0]))
340
341         # Consumer preference incl. uncertainty
342         Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG1'][r, b2, 0]-data_dt['ILG1'][r, b1, 0])/dFik))
343
344         # Preferences are then adjusted for regulations
345         F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
346         + 0.5*(isReg[r, b1]*isReg[r, b2])
347         F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
348         [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
349
350         #Runge-Kutta market share dynamics
351         k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
352         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])
353         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])
354         k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
355
356         #This method currently applies RK4 to the shares, but all other components of the equation
357         #are calculated for the overall time step
358         #We must assume the the LCOE does not change significantly in a time step dt, so we can
359         #focus on the shares.
```

```
356
357         dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
358         dSik[b2, b1] = -dSik[b1, b2]
359
360         #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
361         #dSik[b2, b1] = -dSik[b1, b2]
362
363
364     # -----
365     # Step 3: Exogenous sales additions
366     # -----
367     # Add in exogenous sales figures. These are blended with endogenous result!
368
369
370     # Add in exogenous sales figures. These are blended with
371     # endogenous result! Note that it's different from the
372     # ExogSales specification!
373     Utot = IUD1t[r]
374     iud_lag = time_lag['IUD1'][:, :, 0].sum(axis=1)
375     dSk = np.zeros((len(titles['ITTI'])))
376     dUk = np.zeros((len(titles['ITTI'])))
377     dUkTK = np.zeros((len(titles['ITTI'])))
378     dUkREG = np.zeros((len(titles['ITTI'])))
379
380     # Check that exogenous share changes add to zero
381     dUkTK = data['IXS1'][r, :, 0]
382     if (data['IXS1'][r, :, 0].sum() > 0.0):
383         dUkTK[0] = dUkTK[0] - data['IXS1'][r, :, 0].sum()
384
385     # Correct for regulations
386
387     if iud_lag[r] > 0.0 and IUD1t[r] > 0.0 and (IUD1t[r] - iud_lag[r]) > 0.0:
388
```

```
389         dUKREG = -data_dt['IUD1'][r, :, 0] * ( (IUD1t[r] - iud_lag[r]) /
390             iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])
391
392
393     # Sum effect of exogenous sales additions (if any) with
394     # effect of regulations
395     dUk = copy.deepcopy(dUKREG)
396     dUtot = np.sum(dUk)
397
398     # Convert to market shares and make sure sum is zero
399     # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
400     dSk = np.divide(dUk, Utot) - time_lag['IWS1'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUKTK
401
402
403     # New market shares
404     # check that market shares sum to 1
405
406     data['IWS1'][r, :, 0] = data_dt['IWS1'][r, :, 0] + np.sum(dSik, axis=1) + dSk
407
408     if ~np.isclose(np.sum(data['IWS1'][r, :, 0]), 1.0, atol=1e-5):
409         msg = ""Sector: {} - Region: {} - Year: {}
410             Sum of market shares do not add to 1.0 (instead: {})
411             """.format(sector, titles['RTI'][r], year, np.sum(data['IWS1'][r, :, 0]))
412         warnings.warn(msg)
413
414     if np.any(data['IWS1'][r, :, 0] < 0.0):
415         msg = ""Sector: {} - Region: {} - Year: {}
416             Negative market shares detected! Critical error!
417             """.format(sector, titles['RTI'][r], year)
418         warnings.warn(msg)
419
420
421
```

```
422
423     # =====
424     # Update variables
425     # =====
426
427
428
429     #Useful heat by technology, calculate based on new market shares #Regional totals
430     data['IUD1'][:, :, 0] = data['IWS1'][:, :, 0]* IUD1t[:, np.newaxis]
431
432     # Capacity by technology
433     data['IWK1'][:, :, 0] = divide(data['IUD1'][:, :, 0],
434                                  data['BIC1'][:, :, ctti["13 Capacity factor mean"]]*8766)
435     #add number of devices replaced due to breakdowns = IWK1_lagged/lifetime to yearly capacity additions
436     #note some values of IWI1 negative
437
438     data["IWI1"][:, :, 0] = 0
439     for r in range(len(titles['RTI'])):
440         for tech in range(len(titles['ITTI'])):
441             if(data['IWK1'][r, tech, 0]-time_lag['IWK1'][r, tech, 0]) > 0:
442                 data["IWI1"][r, tech, 0] = (data['IWK1'][r, tech, 0]-time_lag['IWK1'][r, tech, 0])
443
444     data["IWI1"][:, :, 0] = data["IWI1"][:, :, 0] + np.where(data['BIC1'][:, :, ctti['5 Lifetime
445                                     (years)']] !=0.0,
446                                                         divide(time_lag['IWK1'][:, :, 0],
447                                                         data['BIC1'][:, :, ctti['5 Lifetime (years)']] ),0.0)
448
449     #Update emissions
450     #IHW1 is the global average emissions per unit of UED (GWh). IHW1 has units of kt of CO2/GWh
451     for r in range(len(titles['RTI'])):
452         data['IWE1'][r, :, 0] = data['IUD1'][r, :, 0] * data['IHW1'][0, :, 0]
453
```



```
454     #Final energy by technology
455     data['IFD1'][:, :, 0] = np.where(data['BIC1'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
456                                     divide(data['IUD1'][:, :, 0],
457                                             data['BIC1'][:, :, ctti["9 Conversion efficiency"]]),0.0)
458
459
460
461     # =====
462     # Learning-by-doing
463     # =====
464
465     # Cumulative global learning
466     # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
467     # additions (IWI1 Capacity additions) we can estimate total global spillover of similar
468     # techicals
469
470
471
472     bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
473     for r in range(len(titles['RTI'])):
474         bi[r,:] = np.matmul(data['IWB1'][0, :, :],data['IWI1'][r, :, 0])
475     dw = np.sum(bi, axis=0)*dt
476
477     # # Cumulative capacity incl. learning spill-over effects
478     data["IWW1"][0, :, 0] = data_dt['IWW1'][0, :, 0] + dw
479     #
480     # # Copy over the technology cost categories that do not change (all except prices which are updated ↗
481     #     through learning-by-doing below)
482     data['BIC1'] = copy.deepcopy(data_dt['BIC1'])
483     #
484     # # Learning-by-doing effects on investment
485     for tech in range(len(titles['ITTI'])):
```

```
486         if data['IWW1'][0, tech, 0] > 0.1:
487
488             data['BIC1'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC1'][:, tech, ↵
                ctti['1 Investment cost mean (MEuro per MW)']] * \
489
                (1.0 + data['BIC1'][:, tech, ctti['15 ↵
                    Learning exponent']] * dw[tech]/data['IWW1'][0, tech, 0])
490
491         # =====
492         # Update the time-loop variables
493         # =====
494
495         #Calculate levelised cost again
496         data = get_lcoih(data, titles, year)
497
498         #Update time loop variables:
499         for var in data_dt.keys():
500
501             data_dt[var] = copy.deepcopy(data[var])
502
503
504     return data
505
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_fbt_main.py
5 =====
6 Industrial food, beverages, and tobacco sector FTT module.
7 #####
8
9
10 This is the main file for FTT: Industrial Heat - FBT, which models technological
11 diffusion of industrial heat processes within the food, beverages, and tobacco sector due
12 to simulated investor decision making. Investors compare the **levelised cost of
13 industrial heat**, which leads to changes in the market shares of different technologies.
14
15 The outputs of this module include changes in final energy demand and emissions due
16 chemical heat processes for the EU28.
17
18 Local library imports:
19
20     Support functions:
21
22     - `divide <divide.html>`__
23         Bespoke element-wise divide which replaces divide-by-zeros with zeros
24
25 Functions included:
26
27     - solve
28         Main solution function for the module
29     - get_lcoih
30         Calculates the levelised cost of industrial heat
31
32 """
33 # Standard library imports
```

```
34 from math import sqrt
35 import os
36 import copy
37 import sys
38 import warnings
39 import time
40
41 # Third party imports
42 import pandas as pd
43 import numpy as np
44
45 # Local library imports
46 from support.divide import divide
47 from support.econometrics_functions import estimation
48
49 # %% lcoh
50 # -----
51 # ----- LCOH function -----
52 # -----
53 def get_lcoih(data, titles, year):
54     """
55     Calculate levelized costs.
56
57     The function calculates the levelised cost of industrial heat in 2019 Euros
58     It includes intangible costs (gamma values) and together
59     determines the investor preferences.
60
61     Parameters
62     -----
63     data: dictionary
64         Data is a container that holds all cross-sectional (of time) for all
65         variables. Variable names are keys and the values are 3D NumPy arrays.
66     titles: dictionary
```

```
67     Titles is a container of all permissible dimension titles of the model.
68
69     Returns
70     -----
71     data: dictionary
72         Data is a container that holds all cross-sectional (of time) data for
73         all variables.
74         Variable names are keys and the values are 3D NumPy arrays.
75         The values inside the container are updated and returned to the main
76         routine.
77
78     Notes
79     -----
80     Additional notes if required.
81     """
82
83     # Categories for the cost matrix (BIC2)
84     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
85
86     for r in range(len(titles['RTI'])):
87         if data['IUD2'][r, :, 0].sum(axis=0)==0:
88             continue
89
90         # Cost matrix
91         #BIC2 = data['BIC2'][r, :, :]
92
93         lt = data['BIC2'][r, :, ctti['5 Lifetime (years)']]
94         max_lt = int(np.max(lt))
95         lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
96                             num=max_lt, axis=1, endpoint=True)
97         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
98         mask = lt_mat < lt_max_mat
99         lt_mat = np.where(mask, lt_mat, 0)
```

```
100
101
102
103     # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
104     cf = data['BIC2'][r,:, ctti['13 Capacity factor mean']], np.newaxis]
105
106     #conversion efficiency
107     ce = data['BIC2'][r,:, ctti['9 Conversion efficiency']], np.newaxis]
108
109     # Trap for very low CF
110     cf[cf<0.000001] = 0.000001
111
112     # Factor to transfer cost components in terms of capacity to generation
113 #     ones = np.ones([len(titles['ITTI']), 1])
114     conv = 1/(cf)/8766 #number of hours in a year
115
116     # Discount rate
117     # dr = data['BIC2'][r,6]
118     dr = data['BIC2'][r,:, ctti['8 Discount rate']], np.newaxis]
119
120     # Initialise the levelised cost components
121     # Average investment cost
122     it = np.zeros([len(titles['ITTI']), int(max_lt)])
123     it[:, 0, np.newaxis] = data['BIC2'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis] *
124         conv*(1000000)
125
126     # Standard deviation of investment cost
127     dit = np.zeros([len(titles['ITTI']), int(max_lt)])
128     dit[:, 0, np.newaxis] = data['BIC2'][r,:, ctti['2 Investment cost SD']], np.newaxis] * conv*(1000000)
129
130
131     # Subsidies as a percentage of investment cost
```

```
132     st = np.zeros([len(titles['ITTI']), int(max_lt)])
133     st[:, 0, np.newaxis] = (data['BIC2'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
134         * data['ISB2'][r, :, 0,np.newaxis] * conv)*(1000000)
135
136
137     # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
138     ft = np.ones([len(titles['ITTI']), int(max_lt)])
139     ft = ft * data['BIC2'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
140     ft = np.where(mask, ft, 0)
141
142     # Standard deviation of fuel costs
143     dft = np.ones([len(titles['ITTI']), int(max_lt)])
144     dft = dft * data['BIC2'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
145     dft = np.where(mask, dft, 0)
146
147     #fuel tax/subsidies
148     #fft = np.ones([len(titles['ITTI']), int(max_lt)])
149 #     fft = ft * data['PG_FUELTAX'][r, :, :]
150 #     fft = np.where(lt_mask, ft, 0)
151
152     # Fixed operation & maintenance cost - variable O&M available but not included
153     omt = np.ones([len(titles['ITTI']), int(max_lt)])
154     omt = omt * data['BIC2'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) ↗
155         in a year
156     omt = np.where(mask, omt, 0)
157
158     # Standard deviation of operation & maintenance cost
159     domt = np.ones([len(titles['ITTI']), int(max_lt)])
160     domt = domt * data['BIC2'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
161     domt = np.where(mask, domt, 0)
162
163
```

```
164     # Net present value calculations
165     # Discount rate
166     denominator = (1+dr)**(lt_mat)
167
168     # 1-Expenses
169     # 1.1-Without policy costs
170     npv_expenses1 = (it+ft+omt)/denominator
171     # 1.2-With policy costs
172     npv_expenses2 = (it+st+ft+omt)/denominator
173     # 1.3-Only policy costs
174     #npv_expenses3 = (st+fft-fit)/denominator
175     # 2-Utility
176     npv_utility = 1/denominator
177     #Remove 1s for tech with small lifetime than max but keep t=0 as 1
178     npv_utility[npv_utility==1] = 0
179     npv_utility[:,0] = 1
180     # 3-Standard deviation (propagation of error)
181     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
182
183     # 1-levelised cost variants in $/pkm
184     # 1.1-Bare LCOT
185
186     lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
187
188     # 1.2-LCOT including policy costs
189     tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
190     # 1.3 LCOE excluding policy, including co2 price
191     #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
192     # 1.3-LCOT of policy costs
193     # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
194     # Standard deviation of LCOT
195     dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
196
```



```
197     # LCOE augmented with gamma values, no gamma values yet
198     tlcoeg = tlcoe+data['IAM2'][r, :, 0]
199
200     # Pass to variables that are stored outside.
201     data['ILC2'][r, :, 0] = lcoe           # The real bare LCOT without taxes (euros/mwh)
202     #data['IHLT'][r, :, 0] = tlcoe        # The real bare LCOE with taxes
203     data['ILG2'][r, :, 0] = tlcoeg       # As seen by consumer (generalised cost)
204     data['ILD2'][r, :, 0] = dlcoe        # Variation on the LCOT distribution
205
206
207
208     return data
209
210 #Final energy demand has to match IEA
211
212 # %% main function
213 # -----
214 # ----- Main -----
215 # -----
216 def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
217     """
218
219     Main solution function for the module.
220
221     Simulates investor decision making.
222
223     Parameters
224     -----
225     data: dictionary of NumPy arrays
226           Model variables for the given year of solution
227     time_lag: type
228           Description
229     iter_lag: type
```

```
230     Description
231     titles: dictionary of lists
232         Dictionary containing all title classification
233     histend: dict of integers
234         Final year of historical data by variable
235     year: int
236         Current/active year of solution
237     specs: dictionary of NumPy arrays
238         Function specifications for each region and module
239
240     Returns
241     -----
242     data: dictionary of NumPy arrays
243         Model variables for the given year of solution
244
245     """
246
247     # Categories for the cost matrix (BIC2)
248     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
249
250
251     sector = 'Food, beverages and tobacco'
252
253     #Get fuel prices from E3ME and add them to the data for this code
254     #Initialise everything #TODO
255
256     #Calculate or read in FED
257     #Calculate historical emissions
258     data = get_lcoih(data, titles, year)
259
260     # Endogenous calculation takes over from here
261     if year > histend['IUD2']:
262
```

```
263     # Create a local dictionary for timeloop variables
264     # It contains values between timeloop iterations in the FTT core
265     data_dt = {}
266
267     # First, fill the time loop variables with the their lagged equivalents
268     for var in time_lag.keys():
269
270
271         data_dt[var] = copy.deepcopy(time_lag[var])
272
273     # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
274     isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
275     division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
276     division = divide((data_dt['IWK2'][:, :, 0] - data['IRG2'][:, :, 0]),
277                     data_dt['IRG2'][:, :, 0])
278     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
279     isReg[data['IRG2'][:, :, 0] == 0.0] = 1.0
280     isReg[data['IRG2'][:, :, 0] == -1.0] = 0.0
281
282
283     # Factor used to create quarterly data from annual figures
284     no_it = 4
285     dt = 1 / no_it
286     kappa = 10 #tech substitution constant
287
288     ##### Computing new shares #####
289
290     IUD2tot = data['IUD2'][:, :, 0].sum(axis=1)
291     #Start the computation of shares
292     for t in range(1, no_it+1):
293
294         # Interpolate to prevent staircase profile.
295         #Time lagged UED plus change in UED * (no of iterations) * dt
```

```
296
297     IUD2t = time_lag['IUD2'][:, :, 0].sum(axis=1) + (IUD2tot - time_lag['IUD2'][:, :, 0].sum(axis=1)) * t * dt
298
299     for r in range(len(titles['RTI'])):
300
301         if IUD2t[r] == 0.0:
302             continue
303
304
305
306     ##### FTT #####
307
308     # DSik contains the change in shares
309     dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])
310
311     # F contains the preferences
312     F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5
313
314     # Market share constraints
315     Gijmax = np.ones(len(titles['ITTI']))
316     #Gijmin = np.ones((t2ti))
317
318     # -----
319     # Step 1: Endogenous EOL replacements
320     # -----
321     for b1 in range(len(titles['ITTI'])):
322
323         if not (data_dt['IWS2'][r, b1, 0] > 0.0 and
324                data_dt['ILG2'][r, b1, 0] != 0.0 and
325                data_dt['ILD2'][r, b1, 0] != 0.0):
326             continue
327
```

```
328 #TODO: create market share constraints
329 Gijmax[b1] = np.tanh(1.25*(data_dt['ISC2'][0, b1, 0] - data_dt['IWS2'][r, b1, 0])/0.1)
330 #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)
331
332
333
334 S_i = data_dt['IWS2'][r, b1, 0]
335
336
337 for b2 in range(b1):
338
339     if not (data_dt['IWS2'][r, b2, 0] > 0.0 and
340             data_dt['ILG2'][r, b2, 0] != 0.0 and
341             data_dt['ILD2'][r, b2, 0] != 0.0):
342         continue
343
344     S_k = data_dt['IWS2'][r, b2, 0]
345     Aik = data['IWA2'][0, b1, b2]*kappa
346     Aki = data['IWA2'][0, b2, b1]*kappa
347
348     # Propagating width of variations in perceived costs
349     dFik = sqrt(2) * sqrt((data_dt['ILD2'][r, b1, 0]*data_dt['ILD2'][r, b1, 0] + data_dt
350                            ['ILD2'][r, b2, 0]*data_dt['ILD2'][r, b2, 0]))
351
352     # Consumer preference incl. uncertainty
353     Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG2'][r, b2, 0]-data_dt['ILG2'][r, b1, 0])/dFik))
354
355     # Preferences are then adjusted for regulations
356     F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
357     + 0.5*(isReg[r, b1]*isReg[r, b2])
358     F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
359     [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
```

```

358
359         #Runge-Kutta market share dynamics
360         k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
361         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])
362         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])
363         k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
364
365         #This method currently applies RK4 to the shares, but all other components of the equation ↗
366         #are calculated for the overall time step
367         #We must assume the the LCOE does not change significantly in a time step dt, so we can ↗
368         #focus on the shares.
369
370         dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
371         dSik[b2, b1] = -dSik[b1, b2]
372
373         #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
374         #dSik[b2, b1] = -dSik[b1, b2]
375
376         # -----
377         # Step 3: Exogenous sales additions
378         # -----
379         # Add in exogenous sales figures. These are blended with endogenous result!
380
381         # Add in exogenous sales figures. These are blended with
382         # endogenous result! Note that it's different from the
383         # ExogSales specification!
384         Utot = IUD2t[r]
385         iud_lag = time_lag['IUD2'][:, :, 0].sum(axis=1)
386         dSk = np.zeros((len(titles['ITTI'])))
387         dUk = np.zeros((len(titles['ITTI'])))
388         dUkTK = np.zeros((len(titles['ITTI'])))

```

```
389     dUkREG = np.zeros((len(titles['ITTI'])))
390
391     # Check that exogenous share changes add to zero
392     dUkTK = data['IXS2'][r, :, 0]
393     if (data['IXS2'][r, :, 0].sum() > 0.0):
394         dUkTK[0] = dUkTK[0] - data['IXS2'][r, :, 0].sum()
395
396     # Correct for regulations #TODO Does this actually make sense?
397
398     if iud_lag[r] > 0.0 and IUD2t[r] > 0.0 and (IUD2t[r] - iud_lag[r]) > 0.0:
399
400         dUkREG = -data_dt['IUD2'][r, :, 0] * ( (IUD2t[r] - iud_lag[r]) /
401             iud_lag[r]) * isReg[r, :].reshape((len(titles['ITTI'])))
402
403
404     # Sum effect of exogenous sales additions (if any) with
405     # effect of regulations
406     dUk = copy.deepcopy(dUkREG)
407     dUtot = np.sum(dUk)
408
409     # Convert to market shares and make sure sum is zero
410     # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
411     dSk = np.divide(dUk, Utot) - time_lag['IWS2'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK
412
413
414     # New market shares
415     # check that market shares sum to 1
416     #print(np.sum(dSik, axis=1))
417     data['IWS2'][r, :, 0] = data_dt['IWS2'][r, :, 0] + np.sum(dSik, axis=1) + dSk
418
419     if ~np.isclose(np.sum(data['IWS2'][r, :, 0]), 1.0, atol=1e-5):
420         msg = ""Sector: {} - Region: {} - Year: {}
421             Sum of market shares do not add to 1.0 (instead: {})
```

```
422         """ .format(sector, titles['RTI'][r], year, np.sum(data['IWS2'][r, :, 0]))
423         warnings.warn(msg)
424
425     if np.any(data['IWS2'][r, :, 0] < 0.0):
426         msg = ""Sector: {} - Region: {} - Year: {}
427         Negative market shares detected! Critical error!
428         """ .format(sector, titles['RTI'][r], year)
429         warnings.warn(msg)
430
431
432
433
434     # =====
435     # Update variables
436     # =====
437
438     #TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change
439
440     #Useful heat by technology, calculate based on new market shares #Regional totals
441     data['IUD2'][:, :, 0] = data['IWS2'][:, :, 0]* IUD2t[:, np.newaxis]
442
443     # Capacity by technology
444     data['IWK2'][:, :, 0] = divide(data['IUD2'][:, :, 0],
445                                 data['BIC2'][:, :, ctti["13 Capacity factor mean"]]*8766)
446     #add number of devices replaced due to breakdowns = IWK2_lagged/lifetime to yearly capacity additions
447     #note some values of IWI2 negative
448
449     data["IWI2"][:, :, 0] = 0
450     for r in range(len(titles['RTI'])):
451         for tech in range(len(titles['ITTI'])):
452             if(data['IWK2'][r, tech, 0]-time_lag['IWK2'][r, tech, 0]) > 0:
453                 data["IWI2"][r, tech, 0] = (data['IWK2'][r, tech, 0]-time_lag['IWK2'][r, tech, 0])
454     data["IWI2"][:, :, 0] = data["IWI2"][:, :, 0] + np.where(data['BIC2'][:, :, ctti['5 Lifetime
```



```
(years)']] !=0.0,
455         divide(time_lag['IWK2'][:, :, 0],data
        ['BIC2'][:, :, ctti['5 Lifetime (years)']]),
456         0.0)
457
458 #Update emissions
459 #IHW2 is the global average emissions per unit of UED (GWh). IHW2 has units of kt of CO2/GWh
460 for r in range(len(titles['RTI'])):
461     data['IWE2'][r, :, 0] = data['IUD2'][r, :, 0] * data['IHW2'][0, :, 0]
462
463
464 #Final energy by technology
465 data['IFD2'][:, :, 0] = np.where(data['BIC2'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
466     divide(data['IUD2'][:, :, 0],
467     data['BIC2'][:, :, ctti["9 Conversion efficiency"]]),0.0)
468
469
470
471 # =====
472 # Learning-by-doing
473 # =====
474
475 # Cumulative global learning
476 # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
477 # additions (IWI2 Capacity additions) we can estimate total global spillover of similar
478 # techicals
479
480
481
482 bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
483 for r in range(len(titles['RTI'])):
484     bi[r,:] = np.matmul(data['IWB2'][0, :, :],data['IWI2'][r, :, 0])
485 dw = np.sum(bi, axis=0)*dt
```

```
486
487     # # Cumulative capacity incl. learning spill-over effects
488     data["IWW2"][0, :, 0] = data_dt['IWW2'][0, :, 0] + dw
489     #
490     # # Copy over the technology cost categories that do not change (all except prices which are updated ↗
         through learning-by-doing below)
491     data['BIC2'] = copy.deepcopy(data_dt['BIC2'])
492     #
493     # # Learning-by-doing effects on investment
494     for tech in range(len(titles['ITTI'])):
495
496         if data['IWW2'][0, tech, 0] > 0.1:
497
498             data['BIC2'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC2'][:, tech, ↗
                 ctti['1 Investment cost mean (MEuro per MW)']] * \
499                 (1.0 + data['BIC2'][:, tech, ctti['15 ↗
                 Learning exponent']] * dw[tech]/data['IWW2'][0, tech, 0])
500
501     # =====
502     # Update the time-loop variables
503     # =====
504
505     #Calculate levelised cost again
506     data = get_lcoih(data, titles, year)
507
508     #Update time loop variables:
509     for var in data_dt.keys():
510
511
512         data_dt[var] = copy.deepcopy(data[var])
513
514
515     return data
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_mtm_main.py
5 =====
6 Industrial non-ferrous metals, machinery, and transport equipment sector FTT module.
7 #####
8
9
10 This is the main file for FTT: Industrial Heat - MTM, which models technological
11 diffusion of industrial heat processes within the non-ferrous metals, machinery, and
12 transport equipment sector due to simulated investor decision making. Investors compare
13 the **levelised cost of industrial heat**, which leads to changes in the market shares of
14 different technologies.
15
16 The outputs of this module include changes in final energy demand and emissions due
17 chemical heat processes for the EU28.
18
19 Local library imports:
20
21     Support functions:
22
23     - `divide <divide.html>`__
24         Bespoke element-wise divide which replaces divide-by-zeros with zeros
25
26 Functions included:
27
28     - solve
29         Main solution function for the module
30     - get_lcoih
31         Calculates the levelised cost of industrial heat
32
33 """
```

```
34 # Standard library imports
35 from math import sqrt
36 import os
37 import copy
38 import sys
39 import warnings
40 import time
41
42 # Third party imports
43 import pandas as pd
44 import numpy as np
45
46 # Local library imports
47 from support.divide import divide
48 from support.econometrics_functions import estimation
49
50 # %% lcoh
51 # -----
52 # ----- LCOH function -----
53 # -----
54 def get_lcoih(data, titles, year):
55     """
56     Calculate levelized costs.
57
58     The function calculates the levelised cost of industrial heat in 2019 Euros
59     It includes intangible costs (gamma values) and together
60     determines the investor preferences.
61
62     Parameters
63     -----
64     data: dictionary
65         Data is a container that holds all cross-sectional (of time) for all
66         variables. Variable names are keys and the values are 3D NumPy arrays.
```

```
67     titles: dictionary
68         Titles is a container of all permissible dimension titles of the model.
69
70     Returns
71     -----
72     data: dictionary
73         Data is a container that holds all cross-sectional (of time) data for
74         all variables.
75         Variable names are keys and the values are 3D NumPy arrays.
76         The values inside the container are updated and returned to the main
77         routine.
78
79     Notes
80     -----
81     Additional notes if required.
82     """
83
84     # Categories for the cost matrix (BIC3)
85     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
86
87     for r in range(len(titles['RTI'])):
88         if data['IUD3'][r, :, 0].sum(axis=0)==0:
89             continue
90
91         # Cost matrix
92         #BIC3 = data['BIC3'][r, :, :]
93
94         lt = data['BIC3'][r, :, ctti['5 Lifetime (years)']]
95         max_lt = int(np.max(lt))
96         lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
97                             num=max_lt, axis=1, endpoint=True)
98         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
99         mask = lt_mat < lt_max_mat
```

```
100     lt_mat = np.where(mask, lt_mat, 0)
101
102     # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
103     cf = data['BIC3'][r,:, ctti['13 Capacity factor mean']], np.newaxis]
104
105     #conversion efficiency
106     ce = data['BIC3'][r,:, ctti['9 Conversion efficiency']], np.newaxis]
107
108     # Trap for very low CF
109     cf[cf<0.000001] = 0.000001
110
111     # Factor to transfer cost components in terms of capacity to generation
112 #     ones = np.ones([len(titles['ITTI']), 1])
113     conv = 1/(cf)/8766 #number of hours in a year
114
115     # Discount rate
116     # dr = data['BIC3'][r,6]
117     dr = data['BIC3'][r,:, ctti['8 Discount rate']], np.newaxis]
118
119     # Initialise the levelised cost components
120     # Average investment cost
121     it = np.zeros([len(titles['ITTI']), int(max_lt)])
122     it[:, 0, np.newaxis] = data['BIC3'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis] *
        conv*(1*10^6)
123
124
125     # Standard deviation of investment cost
126     dit = np.zeros([len(titles['ITTI']), int(max_lt)])
127     dit[:, 0, np.newaxis] = data['BIC3'][r,:, ctti['2 Investment cost SD']], np.newaxis] * conv*(1*10^6)
128
129
130     # Subsidies as a percentage of investment cost
131     st = np.zeros([len(titles['ITTI']), int(max_lt)])
```

```
132     st[:, 0, np.newaxis] = (data['BIC3'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
133         * data['ISB3'][r, :, 0,np.newaxis] * conv)*(1*10^6)
134
135
136     # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
137     ft = np.ones([len(titles['ITTI']), int(max_lt)])
138     ft = ft * data['BIC3'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
139     ft = np.where(mask, ft, 0)
140
141     # Standard deviation of fuel costs
142     dft = np.ones([len(titles['ITTI']), int(max_lt)])
143     dft = dft * data['BIC3'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
144     dft = np.where(mask, dft, 0)
145
146     #fuel tax/subsidies
147     #fft = np.ones([len(titles['ITTI']), int(max_lt)])
148 #     fft = ft * data['PG_FUELTAX'][r, :, :]
149 #     fft = np.where(lt_mask, ft, 0)
150
151     # Fixed operation & maintenance cost - variable O&M available but not included
152     omt = np.ones([len(titles['ITTI']), int(max_lt)])
153     omt = omt * data['BIC3'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) ↗
154         in a year
155     omt = np.where(mask, omt, 0)
156
157     # Standard deviation of operation & maintenance cost
158     domt = np.ones([len(titles['ITTI']), int(max_lt)])
159     domt = domt * data['BIC3'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
160     domt = np.where(mask, domt, 0)
161
162
163     # Net present value calculations
```

```
164     # Discount rate
165     denominator = (1+dr)**lt_mat
166
167     # 1-Expenses
168     # 1.1-Without policy costs
169     npv_expenses1 = (it+ft+omt)/denominator
170     # 1.2-With policy costs
171     npv_expenses2 = (it+st+ft+omt)/denominator
172     # 1.3-Only policy costs
173     #npv_expenses3 = (st+fft-fit)/denominator
174     # 2-Utility
175     npv_utility = 1/denominator
176     #Remove 1s for tech with small lifetime than max
177     npv_utility[npv_utility==1] = 0
178     npv_utility[:,0] = 1
179     # 3-Standard deviation (propagation of error)
180     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
181
182     # 1-levelised cost variants in $/pkm
183     # 1.1-Bare LCOT
184
185     lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
186
187     # 1.2-LCOT including policy costs
188     tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)+data['IEFI'][r, :, 0]
189     # 1.3 LCOE excluding policy, including co2 price
190     #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
191     # 1.3-LCOT of policy costs
192     # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
193     # Standard deviation of LCOT
194     dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
195
196     # LCOE augmented with gamma values, no gamma values yet
```



```
197     tlcoeg = tlcoe+data['IAM3'][r, :, 0]
198
199     # Pass to variables that are stored outside.
200     data['ILC3'][r, :, 0] = lcoe           # The real bare LCOT without taxes (euros/mwh)
201     #data['IHLT'][r, :, 0] = tlcoe       # The real bare LCOE with taxes
202     data['ILG3'][r, :, 0] = tlcoeg      # As seen by consumer (generalised cost)
203     data['ILD3'][r, :, 0] = dlcoe       # Variation on the LCOT distribution
204
205
206
207     return data
208
209 #Final energy demand has to match IEA
210
211 # %% main function
212 # -----
213 # ----- Main -----
214 # -----
215 def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
216     """
217
218     Main solution function for the module.
219
220     Simulates investor decision making.
221
222     Parameters
223     -----
224     data: dictionary of NumPy arrays
225           Model variables for the given year of solution
226     time_lag: type
227           Description
228     iter_lag: type
229           Description
```

```
230     titles: dictionary of lists
231         Dictionary containing all title classification
232     histend: dict of integers
233         Final year of historical data by variable
234     year: int
235         Current/active year of solution
236     specs: dictionary of NumPy arrays
237         Function specifications for each region and module
238
239     Returns
240     -----
241     data: dictionary of NumPy arrays
242         Model variables for the given year of solution
243
244     """
245
246     # Categories for the cost matrix (BIC3)
247     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
248
249     sector = 'Metals, transport and machinery equipment'
250
251     #Get fuel prices from E3ME and add them to the data for this code
252     #Initialise everything #TODO
253
254     #Calculate or read in FED
255     #Calculate historical emissions
256     data = get_lcoih(data, titles, year)
257
258     # Endogenous calculation takes over from here
259     if year > histend['IUD3']:
260
261         # Create a local dictionary for timeloop variables
262
```

```
263     # It contains values between timeloop iterations in the FTT core
264     data_dt = {}
265
266     # First, fill the time loop variables with the their lagged equivalents
267     for var in time_lag.keys():
268
269         data_dt[var] = copy.deepcopy(time_lag[var])
270
271
272     # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
273     isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
274     division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
275     division = divide((data_dt['IWK3'][:, :, 0] - data['IRG3'][:, :, 0]),
276                      data_dt['IRG3'][:, :, 0])
277     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
278     isReg[data['IRG3'][:, :, 0] == 0.0] = 1.0
279     isReg[data['IRG3'][:, :, 0] == -1.0] = 0.0
280
281
282     # Factor used to create quarterly data from annual figures
283     no_it = 4
284     dt = 1 / no_it
285     kappa = 10 #tech substitution constant
286
287     ##### Computing new shares #####
288     IUD3tot = data['IUD3'][:, :, 0].sum(axis=1)
289     #Start the computation of shares
290     for t in range(1, no_it+1):
291
292         # Interpolate to prevent staircase profile.
293         #Time lagged UED plus change in UED * (no of iterations) * dt
294
295         IUD3t = time_lag['IUD3'][:, :, 0].sum(axis=1) + (IUD3tot - time_lag['IUD3'][:, :, 0].sum(axis=1)) * t * ↗
```

```
dt
296
297     for r in range(len(titles['RTI'])):
298
299         if IUD3t[r] == 0.0:
300             continue
301
302
303
304     ##### FTT #####
305
306     # DSik contains the change in shares
307     dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])
308
309     # F contains the preferences
310     F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5
311
312     # Market share constraints
313     Gijmax = np.ones(len(titles['ITTI']))
314     #Gijmin = np.ones((t2ti))
315
316     # -----
317     # Step 1: Endogenous EOL replacements
318     # -----
319     for b1 in range(len(titles['ITTI'])):
320
321         if not (data_dt['IWS3'][r, b1, 0] > 0.0 and
322                data_dt['ILG3'][r, b1, 0] != 0.0 and
323                data_dt['ILD3'][r, b1, 0] != 0.0):
324             continue
325
326     #TODO: create market share constraints
327     Gijmax[b1] = np.tanh(1.25*(data_dt['ISC3'][0, b1, 0] - data_dt['IWS3'][r, b1, 0])/0.1)
```

```
328 #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)
329
330
331
332 S_i = data_dt['IWS3'][r, b1, 0]
333
334
335 for b2 in range(b1):
336
337     if not (data_dt['IWS3'][r, b2, 0] > 0.0 and
338             data_dt['ILG3'][r, b2, 0] != 0.0 and
339             data_dt['ILD3'][r, b2, 0] != 0.0):
340         continue
341
342     S_k = data_dt['IWS3'][r, b2, 0]
343     Aik = data['IWA3'][0, b1, b2]*kappa
344     Aki = data['IWA3'][0, b2, b1]*kappa
345
346     # Propagating width of variations in perceived costs
347     dFik = sqrt(2) * sqrt((data_dt['ILD3'][r, b1, 0]*data_dt['ILD3'][r, b1, 0] + data_dt
348         ['ILD3'][r, b2, 0]*data_dt['ILD3'][r, b2, 0]))
349
350     # Consumer preference incl. uncertainty
351     Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG3'][r, b2, 0]-data_dt['ILG3'][r, b1, 0])/dFik))
352
353     # Preferences are then adjusted for regulations
354     F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
355     + 0.5*(isReg[r, b1]*isReg[r, b2])
356     F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
357     [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
358
359     #Runge-Kutta market share dynamics
```

```

358     k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
359     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])
360     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])
361     k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
362
363     #This method currently applies RK4 to the shares, but all other components of the equation ↗
        are calculated for the overall time step
364     #We must assume the the LCOE does not change significantly in a time step dt, so we can ↗
        focus on the shares.

365
366     dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
367     dSik[b2, b1] = -dSik[b1, b2]
368
369     #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
370     #dSik[b2, b1] = -dSik[b1, b2]
371
372
373     # -----
374     # Step 3: Exogenous sales additions
375     # -----
376     # Add in exogenous sales figures. These are blended with endogenous result!
377
378
379     # Add in exogenous sales figures. These are blended with
380     # endogenous result! Note that it's different from the
381     # ExogSales specification!
382     Utot = IUD3t[r]
383     iud_lag = time_lag['IUD3'][:, :, 0].sum(axis=1)
384     dSk = np.zeros((len(titles['ITTI'])))
385     dUk = np.zeros((len(titles['ITTI'])))
386     dUkTK = np.zeros((len(titles['ITTI'])))
387     dUkREG = np.zeros((len(titles['ITTI'])))
388

```

```
389     # Check that exogenous share changes add to zero
390     dUkTK = data['IXS3'][r, :, 0]
391     if (data['IXS3'][r, :, 0].sum() > 0.0):
392         dUkTK[0] = dUkTK[0] - data['IXS3'][r, :, 0].sum()
393
394     # Correct for regulations #TODO Does this actually make sense?
395
396     if iud_lag[r] > 0.0 and IUD3t[r] > 0.0 and (IUD3t[r] - iud_lag[r]) > 0.0:
397
398         dUkREG = -data_dt['IUD3'][r, :, 0] * ( (IUD3t[r] - iud_lag[r]) /
399             iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])
400
401
402     # Sum effect of exogenous sales additions (if any) with
403     # effect of regulations
404     dUk = copy.deepcopy(dUkREG)
405     dUtot = np.sum(dUk)
406
407     # Convert to market shares and make sure sum is zero
408     # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
409     dSk = np.divide(dUk, Utot) - time_lag['IWS3'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK
410
411
412     # New market shares
413     # check that market shares sum to 1
414     #print(np.sum(dSik, axis=1))
415     data['IWS3'][r, :, 0] = data_dt['IWS3'][r, :, 0] + np.sum(dSik, axis=1) + dSk
416
417     if ~np.isclose(np.sum(data['IWS3'][r, :, 0]), 1.0, atol=1e-5):
418         msg = ""Sector: {} - Region: {} - Year: {}
419             Sum of market shares do not add to 1.0 (instead: {})
420             """.format(sector, titles['RTI'][r], year, np.sum(data['IWS3'][r, :, 0]))
421         warnings.warn(msg)
```



```
454 data['BIC3'][:, :, ctti['5 Lifetime (years)']],0.0)
455 #Update emissions
456 #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
457 for r in range(len(titles['RTI'])):
458     data['IWE3'][r, :, 0] = data['IUD3'][r, :, 0] * data['IHW3'][0, :, 0]
459
460
461 #Final energy by technology
462 data['IFD3'][:, :, 0] = np.where(data['BIC3'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
463     divide(data['IUD3'][:, :, 0],
464     data['BIC3'][:, :, ctti["9 Conversion efficiency"]]),
465     0.0)
466
467
468
469 # =====
470 # Learning-by-doing
471 # =====
472
473 # Cumulative global learning
474 # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
475 # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
476 # techicals
477
478
479
480
481 bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
482 for r in range(len(titles['RTI'])):
483     bi[r,:] = np.matmul(data['IWB3'][0, :, :],data['IWI3'][r, :, 0])
484 dw = np.sum(bi, axis=0)*dt
485
486 # # Cumulative capacity incl. learning spill-over effects
```

```
487     data['IWW3'][0, :, 0] = data_dt['IWW3'][0, :, 0] + dw
488     #
489     # # Copy over the technology cost categories that do not change (all except prices which are updated
490     # through learning-by-doing below)
491     data['BIC3'] = copy.deepcopy(data_dt['BIC3'])
492     #
493     # # Learning-by-doing effects on investment
494     for tech in range(len(titles['ITTI'])):
495         if data['IWW3'][0, tech, 0] > 0.1:
496             data['BIC3'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC3'][:, tech,
497             ctti['1 Investment cost mean (MEuro per MW)']] * \
498                 (1.0 + data['BIC3'][:, tech, ctti['15
499                 Learning exponent']] * dw[tech]/data['IWW3'][0, tech, 0])
500     # =====
501     # Update the time-loop variables
502     # =====
503
504     #Calculate levelised cost again
505     data = get_lcoih(data, titles, year)
506
507     #Update time loop variables:
508     for var in data_dt.keys():
509
510
511         data_dt[var] = copy.deepcopy(data[var])
512
513
514     return data
515
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_nmm_main.py
5 =====
6 Industrial non-metallic minerals sector FTT module.
7 #####
8
9
10 This is the main file for FTT: Industrial Heat - NMM, which models technological
11 diffusion of industrial heat processes within the non-metallic minerals sector due
12 to simulated investor decision making. Investors compare the **levelised cost of
13 industrial heat**, which leads to changes in the market shares of different technologies.
14
15 The outputs of this module include changes in final energy demand and emissions due
16 chemical heat processes for the EU28.
17
18 Local library imports:
19
20     Support functions:
21
22     - `divide <divide.html>`__
23         Bespoke element-wise divide which replaces divide-by-zeros with zeros
24
25 Functions included:
26
27     - solve
28         Main solution function for the module
29     - get_lcoih
30         Calculates the levelised cost of industrial heat
31
32 """
33 # Standard library imports
```

```
34 from math import sqrt
35 import os
36 import copy
37 import sys
38 import warnings
39 import time
40
41 # Third party imports
42 import pandas as pd
43 import numpy as np
44
45 # Local library imports
46 from support.divide import divide
47 from support.econometrics_functions import estimation
48
49 # %% lcoh
50 # -----
51 # ----- LCOH function -----
52 # -----
53 def get_lcoih(data, titles, year):
54     """
55     Calculate levelized costs.
56
57     The function calculates the levelised cost of industrial heat in 2019 Euros
58     It includes intangible costs (gamma values) and together
59     determines the investor preferences.
60
61     Parameters
62     -----
63     data: dictionary
64         Data is a container that holds all cross-sectional (of time) for all
65         variables. Variable names are keys and the values are 3D NumPy arrays.
66     titles: dictionary
```

```
67     Titles is a container of all permissible dimension titles of the model.
68
69     Returns
70     -----
71     data: dictionary
72         Data is a container that holds all cross-sectional (of time) data for
73         all variables.
74         Variable names are keys and the values are 3D NumPy arrays.
75         The values inside the container are updated and returned to the main
76         routine.
77
78     Notes
79     -----
80     Additional notes if required.
81     """
82
83     # Categories for the cost matrix (BIC4)
84     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
85
86     for r in range(len(titles['RTI'])):
87         if data['IUD4'][r, :, 0].sum(axis=0)==0:
88             continue
89
90         # Cost matrix
91         #BIC4 = data['BIC4'][r, :, :]
92
93         lt = data['BIC4'][r, :, ctti['5 Lifetime (years)']]
94         max_lt = int(np.max(lt))
95         lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
96                             num=max_lt, axis=1, endpoint=True)
97         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
98         mask = lt_mat < lt_max_mat
99         lt_mat = np.where(mask, lt_mat, 0)
```

```
100
101
102     # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
103     cf = data['BIC4'][r,:, ctti['13 Capacity factor mean']], np.newaxis]
104
105     #conversion efficiency
106     ce = data['BIC4'][r,:, ctti['9 Conversion efficiency']], np.newaxis]
107
108     # Trap for very low CF
109     cf[cf<0.000001] = 0.000001
110
111     # Factor to transfer cost components in terms of capacity to generation
112 #     ones = np.ones([len(titles['ITTI']), 1])
113     conv = 1/(cf)/8766 #number of hours in a year
114
115     # Discount rate
116     # dr = data['BIC4'][r,6]
117     dr = data['BIC4'][r,:, ctti['8 Discount rate']], np.newaxis]
118
119     # Initialise the levelised cost components
120     # Average investment cost
121     it = np.zeros([len(titles['ITTI']), int(max_lt)])
122     it[:, 0, np.newaxis] = data['BIC4'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis] *
        conv*(1*10^6)
123
124
125     # Standard deviation of investment cost
126     dit = np.zeros([len(titles['ITTI']), int(max_lt)])
127     dit[:, 0, np.newaxis] = data['BIC4'][r,:, ctti['2 Investment cost SD']], np.newaxis] * conv*(1*10^6)
128
129
130     # Subsidies as a percentage of investment cost
131     st = np.zeros([len(titles['ITTI']), int(max_lt)])
```

```
132     st[:, 0, np.newaxis] = (data['BIC4'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
133         * data['ISB4'][r, :, 0,np.newaxis] * conv)*(1*10^6)
134
135
136     # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
137     ft = np.ones([len(titles['ITTI']), int(max_lt)])
138     ft = ft * data['BIC4'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
139     ft = np.where(mask, ft, 0)
140
141     # Standard deviation of fuel costs
142     dft = np.ones([len(titles['ITTI']), int(max_lt)])
143     dft = dft * data['BIC4'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
144     dft = np.where(mask, dft, 0)
145
146     #fuel tax/subsidies
147     #fft = np.ones([len(titles['ITTI']), int(max_lt)])
148 #     fft = ft * data['PG_FUELTAX'][r, :, :]
149 #     fft = np.where(lt_mask, ft, 0)
150
151     # Fixed operation & maintenance cost - variable O&M available but not included
152     omt = np.ones([len(titles['ITTI']), int(max_lt)])
153     omt = omt * data['BIC4'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) ↗
154         in a year
154     omt = np.where(mask, omt, 0)
155
156     # Standard deviation of operation & maintenance cost
157     domt = np.ones([len(titles['ITTI']), int(max_lt)])
158     domt = domt * data['BIC4'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
159     domt = np.where(mask, domt, 0)
160
161
162
163     # Net present value calculations
```

```
164     # Discount rate
165     denominator = (1+dr)**lt_mat
166
167     # 1-Expenses
168     # 1.1-Without policy costs
169     npv_expenses1 = (it+ft+omt)/denominator
170     # 1.2-With policy costs
171     npv_expenses2 = (it+st+ft+omt)/denominator
172     # 1.3-Only policy costs
173     #npv_expenses3 = (st+fft-fit)/denominator
174     # 2-Utility
175     npv_utility = 1/denominator
176     #Remove 1s for tech with small lifetime than max
177     npv_utility[npv_utility==1] = 0
178     npv_utility[:,0] = 1
179     # 3-Standard deviation (propagation of error)
180     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
181
182     # 1-levelised cost variants in $/pkm
183     # 1.1-Bare LCOT
184
185     lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
186
187     # 1.2-LCOT including policy costs
188     tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)+data['IEFI'][r, :, 0]
189     # 1.3 LCOE excluding policy, including co2 price
190     #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
191     # 1.3-LCOT of policy costs
192     # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
193     # Standard deviation of LCOT
194     dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
195
196     # LCOE augmented with gamma values, no gamma values yet
```



```
197     tlcoeg = tlcoe+data['IAM4'][r, :, 0]
198
199     # Pass to variables that are stored outside.
200     data['ILC4'][r, :, 0] = lcoe           # The real bare LCOT without taxes (euros/mwh)
201     #data['IHLT'][r, :, 0] = tlcoe       # The real bare LCOE with taxes
202     data['ILG4'][r, :, 0] = tlcoeg      # As seen by consumer (generalised cost)
203     data['ILD4'][r, :, 0] = dlcoe       # Variation on the LCOT distribution
204
205
206
207     return data
208
209 #Final energy demand has to match IEA
210
211 # %% main function
212 # -----
213 # ----- Main -----
214 # -----
215 def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
216     """
217     Main solution function for the module.
218
219     Simulates investor decision making.
220
221     Parameters
222     -----
223     data: dictionary of NumPy arrays
224           Model variables for the given year of solution
225     time_lag: type
226             Description
227     iter_lag: type
228             Description
229     titles: dictionary of lists
```

```
230     Dictionary containing all title classification
231     histend: dict of integers
232     Final year of historical data by variable
233     year: int
234     Current/active year of solution
235     specs: dictionary of NumPy arrays
236     Function specifications for each region and module
237
238     Returns
239     -----
240     data: dictionary of NumPy arrays
241         Model variables for the given year of solution
242
243
244     """
245
246     # Categories for the cost matrix (BIC4)
247     ctti = {category: index for index, category in enumerate(enumerate('CTTI'))}
248
249     sector = 'Non-metallic minerals'
250
251     #Get fuel prices from E3ME and add them to the data for this code
252     #Initialise everything #TODO
253
254     #Calculate or read in FED
255     #Calculate historical emissions
256     data = get_lcoih(data, titles, year)
257
258     # Endogenous calculation takes over from here
259     if year > histend['IUD4']:
260
261         # Create a local dictionary for timelooop variables
262         # It contains values between timelooop interations in the FTT core
```

```
263     data_dt = {}
264
265     # First, fill the time loop variables with the their lagged equivalents
266     for var in time_lag.keys():
267
268         data_dt[var] = copy.deepcopy(time_lag[var])
269
270     # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
271     isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
272     division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
273     division = divide((data_dt['IWK4'][:, :, 0] - data['IRG4'][:, :, 0]),
274                      data_dt['IRG4'][:, :, 0])
275     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
276     isReg[data['IRG4'][:, :, 0] == 0.0] = 1.0
277     isReg[data['IRG4'][:, :, 0] == -1.0] = 0.0
278
279
280     # Factor used to create quarterly data from annual figures
281     no_it = 4
282     dt = 1 / no_it
283     kappa = 10 #tech substitution constant
284
285     ##### Computing new shares #####
286     IUD4tot = data['IUD4'][:, :, 0].sum(axis=1)
287     #Start the computation of shares
288     for t in range(1, no_it+1):
289
290         # Interpolate to prevent staircase profile.
291         #Time lagged UED plus change in UED * (no of iterations) * dt
292
293         IUD4t = time_lag['IUD4'][:, :, 0].sum(axis=1) + (IUD4tot - time_lag['IUD4'][:, :, 0].sum(axis=1)) * t * dt
294
```

```
295     for r in range(len(titles['RTI'])):
296
297         if IUD4t[r] == 0.0:
298             continue
299
300
301
302     ##### FTT #####
303
304     # DSik contains the change in shares
305     dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])
306
307     # F contains the preferences
308     F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5
309
310     # Market share constraints
311     Gijmax = np.ones(len(titles['ITTI']))
312     #Gijmin = np.ones((t2ti))
313
314     # -----
315     # Step 1: Endogenous EOL replacements
316     # -----
317     for b1 in range(len(titles['ITTI'])):
318
319         if not (data_dt['IWS4'][r, b1, 0] > 0.0 and
320                data_dt['ILG4'][r, b1, 0] != 0.0 and
321                data_dt['ILD4'][r, b1, 0] != 0.0):
322             continue
323
324     #TODO: create market share constraints
325     Gijmax[b1] = np.tanh(1.25*(data_dt['ISC4'][0, b1, 0] - data_dt['IWS4'][r, b1, 0])/0.1)
326     #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)
327
```

```
328
329
330     S_i = data_dt['IWS4'][r, b1, 0]
331
332
333     for b2 in range(b1):
334
335         if not (data_dt['IWS4'][r, b2, 0] > 0.0 and
336               data_dt['ILG4'][r, b2, 0] != 0.0 and
337               data_dt['ILD4'][r, b2, 0] != 0.0):
338             continue
339
340         S_k = data_dt['IWS4'][r,b2, 0]
341         Aik = data['IWA4'][0,b1 , b2]*kappa
342         Aki = data['IWA4'][0,b2, b1]*kappa
343
344         # Propagating width of variations in perceived costs
345         dFik = sqrt(2) * sqrt((data_dt['ILD4'][r, b1, 0]*data_dt['ILD4'][r, b1, 0] + data_dt
346                                ['ILD4'][r, b2, 0]*data_dt['ILD4'][r, b2, 0]))
347
348         # Consumer preference incl. uncertainty
349         Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG4'][r, b2, 0]-data_dt['ILG4'][r, b1, 0])/dFik))
350
351         # Preferences are then adjusted for regulations
352         F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
353         + 0.5*(isReg[r, b1]*isReg[r, b2])
354
355         F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
356                                [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
357
358
359         #Runge-Kutta market share dynamics
360         k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
361         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])
```

```
358     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])
359     k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
360
361     #This method currently applies RK4 to the shares, but all other components of the equation ↗
        are calculated for the overall time step
362     #We must assume the the LCOE does not change significantly in a time step dt, so we can ↗
        focus on the shares.
363
364     dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
365     dSik[b2, b1] = -dSik[b1, b2]
366
367     #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
368     #dSik[b2, b1] = -dSik[b1, b2]
369
370
371     # -----
372     # Step 3: Exogenous sales additions
373     # -----
374     # Add in exogenous sales figures. These are blended with endogenous result!
375
376
377     # Add in exogenous sales figures. These are blended with
378     # endogenous result! Note that it's different from the
379     # ExogSales specification!
380     Utot = IUD4t[r]
381     iud_lag = time_lag['IUD4'][:, :, 0].sum(axis=1)
382     dSk = np.zeros((len(titles['ITTI'])))
383     dUk = np.zeros((len(titles['ITTI'])))
384     dUkTK = np.zeros((len(titles['ITTI'])))
385     dUkREG = np.zeros((len(titles['ITTI'])))
386
387     # Check that exogenous share changes add to zero
388     dUkTK = data['IXS4'][r, :, 0]
```

```
389     if (data['IXS4'][r, :, 0].sum() > 0.0):
390         dUkTK[0] = dUkTK[0] - data['IXS4'][r, :, 0].sum()
391
392     # Correct for regulations #TODO Does this actually make sense?
393
394     if iud_lag[r] > 0.0 and IUD4t[r] > 0.0 and (IUD4t[r] - iud_lag[r]) > 0.0:
395
396         dUkREG = -data_dt['IUD4'][r, :, 0] * ( (IUD4t[r] - iud_lag[r]) /
397             iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])
398
399
400     # Sum effect of exogenous sales additions (if any) with
401     # effect of regulations
402     dUk = copy.deepcopy(dUkREG)
403     dUtot = np.sum(dUk)
404
405     # Convert to market shares and make sure sum is zero
406     # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
407     dSk = np.divide(dUk, Utot) - time_lag['IWS4'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK
408
409
410     # New market shares
411     # check that market shares sum to 1
412     #print(np.sum(dSik, axis=1))
413     data['IWS4'][r, :, 0] = data_dt['IWS4'][r, :, 0] + np.sum(dSik, axis=1) + dSk
414
415     if ~np.isclose(np.sum(data['IWS4'][r, :, 0]), 1.0, atol=1e-5):
416         msg = ""Sector: {} - Region: {} - Year: {}
417             Sum of market shares do not add to 1.0 (instead: {})
418             """.format(sector, titles['RTI'][r], year, np.sum(data['IWS4'][r, :, 0]))
419         warnings.warn(msg)
420
421     if np.any(data['IWS4'][r, :, 0] < 0.0):
```



```
453     #Update emissions
454     #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
455     for r in range(len(titles['RTI'])):
456         data['IWE4'][r, :, 0] = data['IUD4'][r, :, 0] * data['IHW4'][0, :, 0]
457
458
459     #Final energy by technology
460     data['IFD4'][:, :, 0] = np.where(data['BIC4'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
461                                     divide(data['IUD4'][:, :, 0],
462                                             data['BIC4'][:, :, ctti["9 Conversion efficiency"]]),
463                                     0.0)
464
465
466     # =====
467     # Learning-by-doing
468     # =====
469
470
471     # Cumulative global learning
472     # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
473     # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
474     # techicals
475
476
477
478
479
480     bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
481     for r in range(len(titles['RTI'])):
482         bi[r,:] = np.matmul(data['IWB4'][0, :, :],data['IWI4'][r, :, 0])
483     dw = np.sum(bi, axis=0)*dt
484
485     # # Cumulative capacity incl. learning spill-over effects
```

```
486     data["IWW4"][0, :, 0] = data_dt['IWW4'][0, :, 0] + dw
487     #
488     # # Copy over the technology cost categories that do not change (all except prices which are updated
489     #     through learning-by-doing below)
490     data['BIC4'] = copy.deepcopy(data_dt['BIC4'])
491     #
492     # # Learning-by-doing effects on investment
493     for tech in range(len(titles['ITTI'])):
494         if data['IWW4'][0, tech, 0] > 0.1:
495             data['BIC4'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC4'][:, tech,
496             ctti['1 Investment cost mean (MEuro per MW)']] * \
497             (1.0 + data['BIC4'][:, tech, ctti['15
498             Learning exponent']]) * dw[tech]/data['IWW4'][0, tech, 0])
499     # =====
500     # Update the time-loop variables
501     # =====
502
503     #Calculate levelised cost again
504     data = get_lcoih(data, titles, year)
505
506     #Update time loop variables:
507     for var in data_dt.keys():
508
509
510         data_dt[var] = copy.deepcopy(data[var])
511
512
513     return data
514
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_ois_main.py
5 =====
6 Industrial other sectors FTT module.
7 #####
8
9
10 This is the main file for FTT: Industrial Heat - OIS, which models technological
11 diffusion of industrial heat processes within the other sectors due
12 to simulated investor decision making. Investors compare the **levelised cost of
13 industrial heat**, which leads to changes in the market shares of different technologies.
14
15 The outputs of this module include changes in final energy demand and emissions due
16 chemical heat processes for the EU28.
17
18 Local library imports:
19
20     Support functions:
21
22     - `divide <divide.html>`__
23         Bespoke element-wise divide which replaces divide-by-zeros with zeros
24
25 Functions included:
26
27     - solve
28         Main solution function for the module
29     - get_lcoih
30         Calculates the levelised cost of industrial heat
31
32 """
33 # Standard library imports
```

```

34 from math import sqrt
35 import os
36 import copy
37 import sys
38 import warnings
39 import time
40
41 # Third party imports
42 import pandas as pd
43 import numpy as np
44
45 # Local library imports
46 from support.divide import divide
47 from support.econometrics_functions import estimation
48
49 # %% lcoh
50 # -----
51 # ----- LCOH function -----
52 # -----
53 def get_lcoih(data, titles, year):
54     """
55     Calculate levelized costs.
56
57     The function calculates the levelised cost of industrial heat in 2019 Euros
58     It includes intangible costs (gamma values) and together
59     determines the investor preferences.
60
61     Parameters
62     -----
63     data: dictionary
64         Data is a container that holds all cross-sectional (of time) for all
65         variables. Variable names are keys and the values are 3D NumPy arrays.
66     titles: dictionary

```

```
67     Titles is a container of all permissible dimension titles of the model.
68
69     Returns
70     -----
71     data: dictionary
72         Data is a container that holds all cross-sectional (of time) data for
73         all variables.
74         Variable names are keys and the values are 3D NumPy arrays.
75         The values inside the container are updated and returned to the main
76         routine.
77
78     Notes
79     -----
80     Additional notes if required.
81     """
82
83     # Categories for the cost matrix (BIC5)
84     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
85
86     for r in range(len(titles['RTI'])):
87         if data['IUD5'][r, :, 0].sum(axis=0)==0:
88             continue
89
90         # Cost matrix
91         #BIC5 = data['BIC5'][r, :, :]
92
93         lt = data['BIC5'][r, :, ctti['5 Lifetime (years)']]
94
95         max_lt = int(np.max(lt))
96         lt_mat = np.linspace(np.zeros(len(titles['ITTI'])), max_lt-1,
97                             num=max_lt, axis=1, endpoint=True)
98         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
99         mask = lt_mat < lt_max_mat
```

```
100     lt_mat = np.where(mask, lt_mat, 0)
101
102
103     # Capacity factor used in decisions (constant), not actual capacity factor #TODO ask about this
104     cf = data['BIC5'][r,:, ctti['13 Capacity factor mean']], np.newaxis]
105
106     #conversion efficiency
107     ce = data['BIC5'][r,:, ctti['9 Conversion efficiency']], np.newaxis]
108
109     # Trap for very low CF
110     cf[cf<0.000001] = 0.000001
111
112     # Factor to transfer cost components in terms of capacity to generation
113 #     ones = np.ones([len(titles['ITTI']), 1])
114     conv = 1/(cf)/8766 #number of hours in a year
115
116     # Discount rate
117     # dr = data['BIC5'][r,6]
118     dr = data['BIC5'][r,:, ctti['8 Discount rate']], np.newaxis]
119
120     # Initialise the levelised cost components
121     # Average investment cost
122     it = np.zeros([len(titles['ITTI']), int(max_lt)])
123     it[:, 0, np.newaxis] = data['BIC5'][r,:, ctti['1 Investment cost mean (MEuro per MW)']], np.newaxis] *
        conv*(1*10^6)
124
125
126     # Standard deviation of investment cost
127     dit = np.zeros([len(titles['ITTI']), int(max_lt)])
128     dit[:, 0, np.newaxis] = data['BIC5'][r,:, ctti['2 Investment cost SD']], np.newaxis] * conv*(1*10^6)
129
130
131     # Subsidies as a percentage of investment cost
```

```
132     st = np.zeros([len(titles['ITTI']), int(max_lt)])
133     st[:, 0, np.newaxis] = (data['BIC5'][r,:, ctti['1 Investment cost mean (MEuro per MW)'], np.newaxis]
134         * data['ISB5'][r, :, 0,np.newaxis] * conv)*(1*10^6)
135
136
137     # Average fuel costs 2010Euros/toe to euros/MWh 1 toe = 11.63 MWh
138     ft = np.ones([len(titles['ITTI']), int(max_lt)])
139     ft = ft * data['BIC5'][r,:, ctti['10 Fuel cost mean'], np.newaxis]/11.63/ce
140     ft = np.where(mask, ft, 0)
141
142     # Standard deviation of fuel costs
143     dft = np.ones([len(titles['ITTI']), int(max_lt)])
144     dft = dft * data['BIC5'][r,:, ctti['11 Fuel cost SD'], np.newaxis]/11.63/ce
145     dft = np.where(mask, dft, 0)
146
147     #fuel tax/subsidies
148     #fft = np.ones([len(titles['ITTI']), int(max_lt)])
149 #     fft = ft * data['PG_FUELTAX'][r, :, :]
150 #     fft = np.where(lt_mask, ft, 0)
151
152     # Fixed operation & maintenance cost - variable O&M available but not included
153     omt = np.ones([len(titles['ITTI']), int(max_lt)])
154     omt = omt * data['BIC5'][r,:, ctti['3 O&M cost mean (Euros/MJ/s/year)'], np.newaxis]*conv #(euros per MW) ↗
155         in a year
156     omt = np.where(mask, omt, 0)
157
158     # Standard deviation of operation & maintenance cost
159     domt = np.ones([len(titles['ITTI']), int(max_lt)])
160     domt = domt * data['BIC5'][r,:, ctti['4 O&M cost SD'], np.newaxis]*conv
161     domt = np.where(mask, domt, 0)
162
163
```

```
164     # Net present value calculations
165     # Discount rate
166     denominator = (1+dr)**lt_mat
167
168     # 1-Expenses
169     # 1.1-Without policy costs
170     npv_expenses1 = (it+ft+omt)/denominator
171     # 1.2-With policy costs
172     npv_expenses2 = (it+st+ft+omt)/denominator
173     # 1.3-Only policy costs
174     #npv_expenses3 = (st+fft-fit)/denominator
175     # 2-Utility
176     npv_utility = 1/denominator
177     #Remove 1s for tech with small lifetime than max
178     npv_utility[npv_utility==1] = 0
179     npv_utility[:,0] = 1
180     # 3-Standard deviation (propagation of error)
181     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
182
183     # 1-levelised cost variants in $/pkm
184     # 1.1-Bare LCOT
185
186     lcoe = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
187
188     # 1.2-LCOT including policy costs
189     tlcoe = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)#+data['IEFI'][r, :, 0]
190     # 1.3 LCOE excluding policy, including co2 price
191     #lcoeco2 = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
192     # 1.3-LCOT of policy costs
193     # lcoe_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)+data['MEFI'][r, :, 0]
194     # Standard deviation of LCOT
195     dlcoe = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
196
```



```
197     # LCOE augmented with gamma values, no gamma values yet
198     tlcoeg = tlcoe+data['IAM5'][r, :, 0]
199
200     # Pass to variables that are stored outside.
201     data['ILC5'][r, :, 0] = lcoe           # The real bare LCOT without taxes (euros/mwh)
202     #data['IHLT'][r, :, 0] = tlcoe       # The real bare LCOE with taxes
203     data['ILG5'][r, :, 0] = tlcoeg      # As seen by consumer (generalised cost)
204     data['ILD5'][r, :, 0] = dlcoe       # Variation on the LCOT distribution
205
206
207
208     return data
209
210 #Final energy demand has to match IEA
211
212 # %% main function
213 # -----
214 # ----- Main -----
215 # -----
216 def solve(data, time_lag, iter_lag, titles, histend, year, domain):#, #specs, converter, coefficients):
217     """
218
219     Main solution function for the module.
220
221     Simulates investor decision making.
222
223     Parameters
224     -----
225     data: dictionary of NumPy arrays
226           Model variables for the given year of solution
227     time_lag: type
228           Description
229     iter_lag: type
```

```
230     Description
231     titles: dictionary of lists
232         Dictionary containing all title classification
233     histend: dict of integers
234         Final year of historical data by variable
235     year: int
236         Current/active year of solution
237     specs: dictionary of NumPy arrays
238         Function specifications for each region and module
239
240     Returns
241     -----
242     data: dictionary of NumPy arrays
243         Model variables for the given year of solution
244
245     """
246
247     # Categories for the cost matrix (BIC5)
248     ctti = {category: index for index, category in enumerate(titles['CTTI'])}
249
250
251     sector = 'Metals, transport and machinery equipment'
252
253     #Get fuel prices from E3ME and add them to the data for this code
254     #Initialise everything #TODO
255
256     #Calculate or read in FED
257     #Calculate historical emissions
258     data = get_lcoih(data, titles, year)
259
260     # Endogenous calculation takes over from here
261     if year > histend['IUD5']:
262
```

```
263     # Create a local dictionary for timeloop variables
264     # It contains values between timeloop iterations in the FTT core
265     data_dt = {}
266
267     # First, fill the time loop variables with the their lagged equivalents
268     for var in time_lag.keys():
269
270         data_dt[var] = copy.deepcopy(time_lag[var])
271
272     # Create the regulation variable #Regulate capacity #no regulations yet, isReg full of zeros
273     isReg = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
274     division = np.zeros([len(titles['RTI']), len(titles['ITTI'])])
275     division = divide((data_dt['IWK5'][:, :, 0] - data['IRG5'][:, :, 0]),
276                      data_dt['IRG5'][:, :, 0])
277     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
278     isReg[data['IRG5'][:, :, 0] == 0.0] = 1.0
279     isReg[data['IRG5'][:, :, 0] == -1.0] = 0.0
280
281
282     # Factor used to create quarterly data from annual figures
283     no_it = 4
284     dt = 1 / no_it
285     kappa = 10 #tech substitution constant
286
287     ##### Computing new shares #####
288     IUD5tot = data['IUD5'][:, :, 0].sum(axis=1)
289     #Start the computation of shares
290     for t in range(1, no_it+1):
291
292         # Interpolate to prevent staircase profile.
293         #Time lagged UED plus change in UED * (no of iterations) * dt
294
295         IUD5t = time_lag['IUD5'][:, :, 0].sum(axis=1) + (IUD5tot - time_lag['IUD5'][:, :, 0].sum(axis=1)) * t * ↗
```

```
dt
296
297     for r in range(len(titles['RTI'])):
298
299         if IUD5t[r] == 0.0:
300             continue
301
302
303
304     ##### FTT #####
305
306     # DSik contains the change in shares
307     dSik = np.zeros([len(titles['ITTI']), len(titles['ITTI'])])
308
309     # F contains the preferences
310     F = np.ones([len(titles['ITTI']), len(titles['ITTI'])])*0.5
311
312     # Market share constraints
313     Gijmax = np.ones(len(titles['ITTI']))
314     #Gijmin = np.ones((t2ti))
315
316     # -----
317     # Step 1: Endogenous EOL replacements
318     # -----
319     for b1 in range(len(titles['ITTI'])):
320
321         if not (data_dt['IWS5'][r, b1, 0] > 0.0 and
322                data_dt['ILG5'][r, b1, 0] != 0.0 and
323                data_dt['ILD5'][r, b1, 0] != 0.0):
324             continue
325
326     #TODO: create market share constraints
327     Gijmax[b1] = np.tanh(1.25*(data_dt['ISC5'][0, b1, 0] - data_dt['IWS5'][r, b1, 0])/0.1)
```

```
328         #Gijmin[b1] = np.tanh(1.25*(-mes2_dt[r, b1, 0] + mews_dt[r, b1, 0])/0.1)
329
330
331
332         S_i = data_dt['IWS5'][r, b1, 0]
333
334
335         for b2 in range(b1):
336
337             if not (data_dt['IWS5'][r, b2, 0] > 0.0 and
338                     data_dt['ILG5'][r, b2, 0] != 0.0 and
339                     data_dt['ILD5'][r, b2, 0] != 0.0):
340                 continue
341
342             S_k = data_dt['IWS5'][r, b2, 0]
343             Aik = data['IWA5'][0, b1, b2]*kappa
344             Aki = data['IWA5'][0, b2, b1]*kappa
345
346             # Propagating width of variations in perceived costs
347             dFik = sqrt(2) * sqrt((data_dt['ILD5'][r, b1, 0]*data_dt['ILD5'][r, b1, 0] + data_dt
348                                     ['ILD5'][r, b2, 0]*data_dt['ILD5'][r, b2, 0]))
349
350             # Consumer preference incl. uncertainty
351             Fik = 0.5*(1+np.tanh(1.25*(data_dt['ILG5'][r, b2, 0]-data_dt['ILG5'][r, b1, 0])/dFik))
352
353             # Preferences are then adjusted for regulations
354             F[b1, b2] = Fik*(1.0-isReg[r, b1]) * (1.0 - isReg[r, b2]) + isReg[r, b2]*(1.0-isReg[r, b1])
355             + 0.5*(isReg[r, b1]*isReg[r, b2])
356             F[b2, b1] = (1.0-Fik)*(1.0-isReg[r, b2]) * (1.0 - isReg[r, b1]) + isReg[r, b1]*(1.0-isReg
357                 [r, b2]) + 0.5*(isReg[r, b2]*isReg[r, b1])
358
359             #Runge-Kutta market share dynamics
```

```

358     k_1 = S_i*S_k * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
359     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])
360     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])
361     k_4 = (S_i+dt*k_3)*(S_k-dt*k_3) * (Aik*F[b1, b2]*Gijmax[b1] - Aki*F[b2, b1]*Gijmax[b2])
362
363     #This method currently applies RK4 to the shares, but all other components of the equation ↗
364     #are calculated for the overall time step
365     #We must assume the the LCOE does not change significantly in a time step dt, so we can ↗
366     #focus on the shares.
367
368     dSik[b1, b2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
369     dSik[b2, b1] = -dSik[b1, b2]
370
371     #dSik[b1, b2] = S_i*S_k* (Aik*F[b1,b2]*Gijmax[b1] - Aki*F[b2,b1]*Gijmax[b2])*dt
372     #dSik[b2, b1] = -dSik[b1, b2]
373
374     # -----
375     # Step 3: Exogenous sales additions
376     # -----
377     # Add in exogenous sales figures. These are blended with endogenous result!
378
379     # Add in exogenous sales figures. These are blended with
380     # endogenous result! Note that it's different from the
381     # ExogSales specification!
382     Utot = IUD5t[r]
383     iud_lag = time_lag['IUD5'][:, :, 0].sum(axis=1)
384     dSk = np.zeros((len(titles['ITTI'])))
385     dUk = np.zeros((len(titles['ITTI'])))
386     dUkTK = np.zeros((len(titles['ITTI'])))
387     dUkREG = np.zeros((len(titles['ITTI'])))
388

```

```
389     # Check that exogenous share changes add to zero
390     dUkTK = data['IXS5'][r, :, 0]
391     if (data['IXS5'][r, :, 0].sum() > 0.0):
392         dUkTK[0] = dUkTK[0] - data['IXS5'][r, :, 0].sum()
393
394     # Correct for regulations #TODO Does this actually make sense?
395
396     if iud_lag[r] > 0.0 and IUD5t[r] > 0.0 and (IUD5t[r] - iud_lag[r]) > 0.0:
397
398         dUkREG = -data_dt['IUD5'][r, :, 0] * ( (IUD5t[r] - iud_lag[r]) /
399             iud_lag[r]) * isReg[r, :].reshape([len(titles['ITTI'])])
400
401
402     # Sum effect of exogenous sales additions (if any) with
403     # effect of regulations
404     dUk = copy.deepcopy(dUkREG)
405     dUtot = np.sum(dUk)
406
407     # Convert to market shares and make sure sum is zero
408     # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
409     dSk = np.divide(dUk, Utot) - time_lag['IWS5'][r, :, 0]*Utot*np.divide(dUtot, (Utot*Utot)) + dUkTK
410
411
412     # New market shares
413     # check that market shares sum to 1
414     #print(np.sum(dSik, axis=1))
415     data['IWS5'][r, :, 0] = data_dt['IWS5'][r, :, 0] + np.sum(dSik, axis=1) + dSk
416
417     if ~np.isclose(np.sum(data['IWS5'][r, :, 0]), 1.0, atol=1e-5):
418         msg = ""Sector: {} - Region: {} - Year: {}
419         Sum of market shares do not add to 1.0 (instead: {})
420         """.format(sector, titles['RTI'][r], year, np.sum(data['IWS5'][r, :, 0]))
421         warnings.warn(msg)
```

```
422
423     if np.any(data['IWS5'][r, :, 0] < 0.0):
424         msg = ""Sector: {} - Region: {} - Year: {}
425         Negative market shares detected! Critical error!
426         "".format(sector, titles['RTI'][r], year)
427         warnings.warn(msg)
428
429
430
431
432     # =====
433     # Update variables
434     # =====
435
436     #TODO: what else needs to go here? TODO calculate new capacity and new yearly capacity change
437
438     #Useful heat by technology, calculate based on new market shares #Regional totals
439     data['IUD5'][:, :, 0] = data['IWS5'][:, :, 0]* IUD5t[:, np.newaxis]
440
441     # Capacity by technology
442     data['IWK5'][:, :, 0] = divide(data['IUD5'][:, :, 0],
443                                 data['BIC5'][:, :, ctti["13 Capacity factor mean"]]*8766)
444     #add number of devices replaced due to breakdowns = IWK4_lagged/lifetime to yearly capacity additions
445     #note some values of IWI4 negative
446     data["IWI1"][:, :, 0] = 0
447     for r in range(len(titles['RTI'])):
448         for tech in range(len(titles['ITTI'])):
449             if(data['IWK5'][r, tech, 0]-time_lag['IWK5'][r, tech, 0]) > 0:
450                 data['IWI5'][r, tech, 0] = (data['IWK5'][r, tech, 0]-time_lag['IWK5'][r, tech, 0])
451     data['IWI5'][:, :, 0] = data['IWI5'][:, :, 0] + np.where(data['BIC5'][:, :, ctti['5 Lifetime
452                                     (years)']] !=0.0,
                                                         divide(time_lag['IWK5'][:, :, 0],data
                                                         ['BIC5'][:, :, ctti['5 Lifetime (years)']]))),
```



```
453                                     0.0)
454
455     #Update emissions
456     #IHW4 is the global average emissions per unit of UED (GWh). IHW4 has units of kt of CO2/GWh
457     for r in range(len(titles['RTI'])):
458         data['IWE5'][r, :, 0] = data['IUD5'][r, :, 0] * data['IHW5'][0, :, 0]
459
460
461     #Final energy by technology
462     data['IFD5'][:, :, 0] = np.where(data['BIC5'][:, :, ctti["9 Conversion efficiency"]] !=0.0,
463                                     divide(data['IUD5'][:, :, 0],
464                                             data['BIC5'][:, :, ctti["9 Conversion efficiency"]]),
465                                     0.0)
466
467
468
469     # =====
470     # Learning-by-doing
471     # =====
472
473     # Cumulative global learning
474     # Using a technological spill-over matrix (IEWB spillover matrix) together with capacity
475     # additions (IWI4 Capacity additions) we can estimate total global spillover of similar
476     # techicals
477
478
479
480
481
482     bi = np.zeros((len(titles['RTI']),len(titles['ITTI'])))
483     for r in range(len(titles['RTI'])):
484         bi[r,:] = np.matmul(data['IWB5'][0, :, :],data['IWI5'][r, :, 0])
485     dw = np.sum(bi, axis=0)*dt
```

```
486
487     # # Cumulative capacity incl. learning spill-over effects
488     data['IWW5'][0, :, 0] = data_dt['IWW5'][0, :, 0] + dw
489     #
490     # # Copy over the technology cost categories that do not change (all except prices which are updated ↗
491     #     through learning-by-doing below)
492     data['BIC5'] = copy.deepcopy(data_dt['BIC5'])
493     #
494     # # Learning-by-doing effects on investment
495     for tech in range(len(titles['ITTI'])):
496         if data['IWW5'][0, tech, 0] > 0.1:
497             data['BIC5'][:, tech, ctti['1 Investment cost mean (MEuro per MW)']] = data_dt['BIC5'][:, tech, ↗
498                 ctti['1 Investment cost mean (MEuro per MW)']] * \
499                 (1.0 + data['BIC5'][:, tech, ctti['15 ↗
500                     Learning exponent']] * dw[tech]/data['IWW5'][0, tech, 0])
501     # =====
502     # Update the time-loop variables
503     # =====
504
505     #Calculate levelised cost again
506     data = get_lcoih(data, titles, year)
507
508     #Update time loop variables:
509     for var in data_dt.keys():
510
511         data_dt[var] = copy.deepcopy(data[var])
512
513
514     return data
515
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 ftt_tr_main.py
5 =====
6 Passenger road transport FTT module.
7 #####
8
9 This is the main file for FTT: Transport, which models technological
10 diffusion of passenger vehicle types due to simulated consumer decision making.
11 Consumers compare the **levelised cost of heat**, which leads to changes in the
12 market shares of different technologies.
13
14 The outputs of this module include sales, fuel use, and emissions.
15
16 Local library imports:
17
18     Support functions:
19
20     - `divide <divide.html>`__
21       Bespoke element-wise divide which replaces divide-by-zeros with zeros
22     - `estimation <econometrics_functions.html>`__
23       Predict future values according to the estimated coefficients.
24
25 Functions included:
26     - solve
27       Main solution function for the module
28     - get_lcot
29       Calculate levelised cost of transport
30 """
31
32 # Standard library imports
33 from math import sqrt
```

```
34 import os
35 import copy
36 import sys
37 import warnings
38
39 # Third party imports
40 import pandas as pd
41 import numpy as np
42
43 # Local library imports
44 from support.divide import divide
45 from support.econometrics_functions import estimation
46
47
48 # %% lcot
49 # -----
50 # ----- LCOT function -----
51 # -----
52 def get_lcot(data, titles):
53     """
54     Calculate levelized costs.
55
56     The function calculates the levelised cost of transport in 2012$/p-km per
57     vehicle type. It includes intangible costs (gamma values) and together
58     determines the investor preferences.
59     """
60
61     # Categories for the cost matrix (BTTC)
62     c3ti = {category: index for index, category in enumerate(titles['C3TI'])}
63
64     tf = np.ones([len(titles['VTTI']), 1])
65     tf[12:15] = 0
66     tf[18:21] = 0
```

```
67
68     for r in range(len(titles['RTI'])):
69
70         # Cost matrix
71         bttc = data['BTTC'][r, :, :]
72
73         # Vehicle lifetime
74         lt = bttc[:, c3ti['8 lifetime']]
75         max_lt = int(np.max(lt))
76         lt_mat = np.linspace(np.zeros(len(titles['VTI'])), max_lt-1,
77                             num=max_lt, axis=1, endpoint=True)
78         lt_max_mat = np.concatenate(int(max_lt)*[lt[:, np.newaxis]], axis=1)
79         mask = lt_mat < lt_max_mat
80         lt_mat = np.where(mask, lt_mat, 0)
81
82         # Capacity factor used in decisions (constant), not actual capacity factor
83         cf = bttc[:, c3ti['12 Cap_F (Mpkm/kseats-y)']], np.newaxis]
84
85         # Discount rate
86         # dr = bttc[6]
87         dr = bttc[:, c3ti['7 Discount rate']], np.newaxis]
88
89         # Occupancy rates
90         ff = bttc[:, c3ti['11 occupancy rate p/sea']], np.newaxis]
91
92         # Number of seats
93         ns = bttc[:, c3ti['15 Seats/Veh']], np.newaxis]
94
95         # Energy use
96         en = bttc[:, c3ti['9 energy use (MJ/km)']], np.newaxis]
97
98         # Initialise the levelised cost components
99         # Average investment cost
```

```
100     it = np.zeros([len(titles['VTTI']), int(max_lt)])
101     it[:, 0, np.newaxis] = btcc[:, c3ti['1 Prices cars (USD/veh)'], np.newaxis]/ns/ff/cf/1000
102
103     # Standard deviation of investment cost
104     dit = np.zeros([len(titles['VTTI']), int(max_lt)])
105     dit[:, 0, np.newaxis] = btcc[:, c3ti['2 Std of price'], np.newaxis]/ns/ff/cf/1000
106
107     # Vehicle tax at purchase
108     vtt = np.zeros([len(titles['VTTI']), int(max_lt)])
109     vtt[:, 0, np.newaxis] = (data['TTVT'][r, :, 0, np.newaxis]+data['RTCO'][r, 0, 0]*btcc[:,c3ti['14
        CO2Emissions'], np.newaxis])/ns/ff/cf/1000
110
111     # Average fuel costs
112     ft = np.ones([len(titles['VTTI']), int(max_lt)])
113     ft = ft * btcc[:,c3ti['3 fuel cost (USD/km)'], np.newaxis]/ns/ff
114     ft = np.where(mask, ft, 0)
115
116     # Stadarnd deviation of fuel costs
117     dft = np.ones([len(titles['VTTI']), int(max_lt)])
118     dft = dft * btcc[:, c3ti['4 std fuel cost'], np.newaxis]
119     dft = np.where(mask, dft, 0)
120
121     # Fuel tax costs
122     fft = np.ones([len(titles['VTTI']), int(max_lt)])
123     fft = fft * data['RTFT'][r, 0, 0]*en/ns/ff*tf
124     fft = np.where(mask, fft, 0)
125
126     # Average operation & maintenance cost
127     omt = np.ones([len(titles['VTTI']), int(max_lt)])
128     omt = omt * btcc[:, c3ti['5 O&M costs (USD/km)'], np.newaxis]/ns/ff
129     omt = np.where(mask, omt, 0)
130
131     # Standard deviation of operation & maintenance cost
```

```
132     domt = np.ones([len(titles['VTTI']), int(max_lt)])
133     domt = omt * btte[:, c3ti['6 std O&M']], np.newaxis]/ns
134     domt = np.where(mask, domt, 0)
135
136     # Road tax cost
137     rtt = np.ones([len(titles['VTTI']), int(max_lt)])
138     rtt = rtt * data['TTRT'][r, :, 0, np.newaxis]/cf/ns/ff/1000
139     rtt = np.where(mask, rtt, 0)
140
141     # Net present value calculations
142     # Discount rate
143     denominator = (1+dr)**lt_mat
144
145     # 1-Expenses
146     # 1.1-Without policy costs
147     npv_expenses1 = (it+ft+omt)/denominator
148     # 1.2-With policy costs
149     npv_expenses2 = (it+vtt+ft+fft+omt+rtt)/denominator
150     # 1.3-Only policy costs
151     npv_expenses3 = (vtt+fft+rtt)/denominator
152     # 2-Utility
153     npv_utility = 1/denominator
154     #Remove 1s for tech with small lifetime than max
155     npv_utility[npv_utility==1] = 0
156     npv_utility[:,0] = 1
157     # 3-Standard deviation (propagation of error)
158     npv_std = np.sqrt(dit**2 + dft**2 + domt**2)/denominator
159
160     # 1-levelised cost variants in $/pkm
161     # 1.1-Bare LCOT
162     lcot = np.sum(npv_expenses1, axis=1)/np.sum(npv_utility, axis=1)
163     # 1.2-LCOT including policy costs
164     tlcot = np.sum(npv_expenses2, axis=1)/np.sum(npv_utility, axis=1)
```

```

165     # 1.3-LCOT of policy costs
166     lcot_pol = np.sum(npv_expenses3, axis=1)/np.sum(npv_utility, axis=1)
167     # Standard deviation of LCOT
168     dlcot = np.sum(npv_std, axis=1)/np.sum(npv_utility, axis=1)
169
170     # LCOT augmented with non-pecuniary costs
171     tlcotg = tlcot*(1+data['TGAM'][r, :, 0])
172
173     # Transform into lognormal space
174     logtlcot = np.log(tlcot*tlcot/np.sqrt(dlcot*dlcot + tlcot*tlcot)) + data['TGAM'][r, :, 0]
175     dlogtlcot = np.sqrt(np.log10(1.0 + dlcot*dlcot/(tlcot*tlcot)))
176
177     # Pass to variables that are stored outside.
178     data['TEWC'][r, :, 0] = lcot           # The real bare LCOT without taxes
179     data['TETC'][r, :, 0] = tlcot         # The real bare LCOE with taxes
180     data['TEGC'][r, :, 0] = tlcotg       # As seen by consumer (generalised cost)
181     data['TELC'][r, :, 0] = logtlcot     # In lognormal space
182     data['TECD'][r, :, 0] = dlcot        # Variation on the LCOT distribution
183     data['TLCD'][r, :, 0] = dlogtlcot    # Log variation on the LCOT distribution
184
185     return data
186 # %% Fleet size - under development
187 # -----
188 # ----- Gompertz equation for fleet size -----
189 # -----
190 # def fleet_size(data, titles):
191 #
192 #     return print("Hello")
193
194
195 # %% survival function
196 # -----
197 # ----- Survival calculation -----

```



```

198 # -----
199 def survival_function(data, time_lag, histend, year, titles):
200     # In this function we calculate scrappage, sales, tracking of age, and
201     # average efficiency.
202     # Categories for the cost matrix (BTTC)
203     c3ti = {category: index for index, category in enumerate(titles['C3TI'])}
204
205
206     # Create a generic matrix of fleet-stock by age
207     # Assume uniform distribution, but only do so when we still have historical
208     # market share data. Afterwards it becomes endogenous
209     if year < histend['TEWS']:
210
211         # TODO: This needs to be replaced with actual data
212         correction = np.linspace(1/(3*len(titles['VYTI'])), 3/len(titles['VYTI']), len(titles['VYTI'])) * 0.6
213
214         for age in range(len(titles['VYTI'])):
215
216             data['RLTA'][:, :, age] = correction[age] * data['TEWK'][:, :, 0]
217
218     else:
219         # Once we start to calculate the market shares and total fleet sizes
220         # endogenously, we can update the vehicle stock by age matrix and
221         # calculate scrappage, sales, average age, and average efficiency.
222         for r in range(len(titles['RTI'])):
223
224             for veh in range(len(titles['VTTI'])):
225
226                 # Move all vehicles one year up:
227                 # New sales will get added to the age-tracking matrix in the main
228                 # routine.
229                 data['RLTA'][r, veh, :-1] = copy.deepcopy(time_lag['RLTA'][r, veh, 1:])
230

```

```
231     # Current age-tracking matrix:
232     # Only retain the fleet that survives
233     data['RLTA'][r, veh, :] = data['RLTA'][r, veh, :] * data['TESF'][r, 0, :]
234
235     # Total amount of vehicles that survive:
236     survival = np.sum(data['RLTA'][r, veh, :])
237
238     # EoL scrappage: previous year's stock minus what survived
239     if time_lag['TEWK'][r, veh, 0] > survival:
240
241         data['REVS'][r, veh, 0] = time_lag['TEWK'][r, veh, 0] - survival
242
243     elif time_lag['TEWK'][r, veh, 0] < survival:
244         if year > 2016:
245             msg = """
246             Erronous outcome!
247             Check year {}, region {} - {}, vehicle {} - {}
248             Vehicles that survived are greater than what was in the fleet before:
249             {} versus {}
250             """.format(year, r, titles['RTI'][r], veh,
251                       titles['VTI'][veh], time_lag['TEWK'][r, veh, 0], survival)
252         #         print(msg)
253
254     # calculate fleet size
255     return data
256
257 # %% main function
258 # -----
259 # ----- Main -----
260 # -----
261 def solve(data, time_lag, iter_lag, titles, histend, year, specs):
262     """
263     Main solution function for the module.
```

```
264
265     Add an extended description in the future.
266
267     Parameters
268     -----
269     data: dictionary of NumPy arrays
270         Model variables for the given year of solution
271     time_lag: type
272         Description
273     iter_lag: type
274         Description
275     titles: dictionary of lists
276         Dictionary containing all title classification
277     histend: dict of integers
278         Final year of historical data by variable
279     year: int
280         Current/active year of solution
281     specs: dictionary of NumPy arrays
282         Function specifications for each region and module
283
284     Returns
285     -----
286     data: dictionary of NumPy arrays
287         Model variables for the given year of solution
288
289     Notes
290     -----
291     This function should be broken up into more elements in development.
292     """
293     # Categories for the cost matrix (BTTC)
294     c3ti = {category: index for index, category in enumerate(titles['C3TI'])}
295     jti = {category: index for index, category in enumerate(titles['JTI'])}
296
```

```
297 fuelvars = ['FR_1', 'FR_2', 'FR_3', 'FR_4', 'FR_5', 'FR_6',
298             'FR_7', 'FR_8', 'FR_9', 'FR_10', 'FR_11', 'FR_12']
299
300 sector = "tr_road_pass"
301 sector_index = 0
302 sector_index = 15 #titles['FUTI'].index('16 Road Transport')
303
304 # Store fuel prices and convert to $2013/toe
305 # It's actually in current$/toe
306 # TODO: Temporary deflator values
307 data['TE3P'][:, jti["5 Middle distillates"], 0] = iter_lag['PFRM'][:, sector_index, 0] / 1.33
308 data['TE3P'][:, jti["7 Natural gas"], 0] = iter_lag['PFRG'][:, sector_index, 0] / 1.33
309 data['TE3P'][:, jti["8 Electricity"], 0] = iter_lag['PFRE'][:, sector_index, 0] / 1.33
310 data['TE3P'][:, jti["11 Biofuels"], 0] = iter_lag['PFRB'][:, sector_index, 0] / 1.33
311 # data['TE3P'][:, "12 Hydrogen", 0] = data['PFRE'][:, sector_index, 0] * 2.0
312 # %% First initialise if necessary
313
314
315 # Up to the last year of historical market share data
316 if year <= histend['TEWS']:
317
318     for r in range(len(titles['RTI'])):
319
320         # CORRECTION TO MARKET SHARES
321         # Sometimes historical market shares do not add up to 1.0
322         if (~np.isclose(np.sum(data['TEWS'][r, :, 0]), 0.0, atol=1e-9)
323             and np.sum(data['TEWS'][r, :, 0]) > 0.0):
324             data['TEWS'][r, :, 0] = np.divide(data['TEWS'][r, :, 0],
325                                             np.sum(data['TEWS'][r, :, 0]))
326
327         # Computes initial values for the capacity factor, numbers of
328         # vehicles by technology and distance driven
329         # "Capacity factor", defined as km driven per vehicle
```

```
330     #data['TEWL'][:, :, 0] = data['RVKM'][:, 0, 0, np.newaxis]
331
332     # "Capacities", defined as 1000 vehicles
333     data['TEWK'][:, :, 0] = data['TEWS'][:, :, 0] * data['RFLT'][:, 0, 0, np.newaxis]
334
335     # "Generation", defined as total km driven
336     data['TEWG'][:, :, 0] = data['TEWK'][:, :, 0] * data['RVKM'][:, 0, 0, np.newaxis] * 1e-3
337
338     # Call the survival function routine.
339     data = survival_function(data, time_lag, histend, year, titles)
340
341     if year == histend['TEWS']:
342
343         # Sales are the difference between fleet sizes and the addition of scrapped vehicles
344         data['TEWI'][:, :, 0] = data['TEWK'][:, :, 0] - time_lag['TEWK'][:, :, 0] + data['REVS'][:, :, 0]
345
346         # Corrections to sales and EOL when sales are negative.
347         condition = data['TEWI'][:, :, 0] < 0.0
348         data['REVS'][:, :, 0] = np.where(condition,
349                                     data['REVS'][:, :, 0] - data['TEWI'][:, :, 0],
350                                     data['REVS'][:, :, 0])
351
352         data['TEWI'][:, :, 0] = np.where(condition,
353                                     0.0,
354                                     data['TEWI'][:, :, 0])
355
356
357         # Add sales to the age tracking matrix
358         data['RLTA'][:, :, -1] = data['TEWI'][:, :, 0]
359
360         # Local variable to calculate age related weights:
361         share_by_age = np.zeros_like(data['RLTA'][:, :, :])
362         # Average vehicle age and average relative efficiency factor
```



```

395         fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * data['RBFM'][r, 0, 0]
396         #TODO this is broken
397         # Calculate fuel use - passenger car only! Therefore this will
398         # differ from E3ME results
399         # TEWG:                km driven
400         # Convert energy unit (1/41.868) ktoe/MJ
401         # Energy demand (BBTC)    MJ/km
402
403         data['TJEF'][r, :, 0] = (np.matmul(np.transpose(fuel_converter), data['TEWG'][r, :, 0])*\
404             data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']]*data['TEFF'][r, :,
0]/41.868))
405
406
407         data['TVFP'][r, :, 0] = (np.matmul(fuel_converter, data['TE3P'][r, :, 0]))*\
408             data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] * \
409             data['TEFF'][r, :, 0]/ 41868
410
411         # Emissions, unit MtCO2 - using vehicle emission factors, corrected by the biofuel mandate
412         data['TEWE'][:, :, 0] = data['TEWG'][:, :, 0] * \
413             data['BTTC'][:, :, c3ti['14 CO2Emissions']] * \
414             data['TEFF'][:, :, 0] * 1e-6 * 1.2 * emis_corr
415
416         #(NOT USED)
417         #data['TP_VEC'][:, :, 0] = copy.deepcopy(data['BTTC'][:, :, c3ti['3 fuel cost (USD/km)']])
418
419         # Calculate the LCOT for each vehicle type.
420         # Call the function
421         data = get_lcot(data, titles)
422
423         # %% Simulation of stock and energy specs
424
425         if year > histend['TEWS']:
426             # TODO: Implement survival function to get a more accurate depiction of

```

```
427     # vehicles being phased out and to be able to track the age of the fleet.
428     # This means that a new variable will need to be implemented which is
429     # basically TP_VFLT with a third dimension (vehicle age in years- up to 23y)
430     # Reduced efficiencies can then be tracked properly as well.
431
432     # Create a local dictionary for timeloop variables
433     # It contains values between timeloop iterations in the FTT core
434     data_dt = {}
435
436     # First, fill the time loop variables with the their lagged equivalents
437     for var in time_lag.keys():
438
439         data_dt[var] = copy.deepcopy(time_lag[var])
440
441     data_dt['TWIY'] = np.zeros([len(titles['RTI']), len(titles['VTTI']), 1])
442
443     #Create the regulation variable
444     isReg = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
445     division = np.zeros([len(titles['RTI']), len(titles['VTTI'])])
446     division = divide((data_dt['TEWS'][:, :, 0] - data['TREG'][:, :, 0]),
447                      data_dt['TREG'][:, :, 0])
448     isReg = 0.5 + 0.5*np.tanh(2*1.25*division)
449     isReg[data['TREG'][:, :, 0] == 0.0] = 1.0
450     isReg[data['TREG'][:, :, 0] == -1.0] = 0.0
451
452     # Call the survival function routine.
453     data = survival_function(data, time_lag, histend, year, titles)
454
455     # Total number of scrapped vehicles: #TP_TEOL changed to RVTS #Is this really needed?
456     #data['RVTS'][:, 0, 0] = np.sum(data['REVS'][:, :, 0], axis=1)
457
458     # Factor used to create quarterly data from annual figures
459     no_it = 4
```



```
460     dt = 1 / no_it
461
462     ##### Computing new shares #####
463
464     #Start the computation of shares
465     for t in range(1, no_it+1):
466
467         # Both rvkm and RFLT are exogenous at the moment
468         # Interpolate to prevent staircase profile.
469         rvkmt = time_lag['RVKM'][:, 0, 0] + (data['RVKM'][:, 0, 0] - time_lag['RVKM'][:, 0, 0]) * t * dt
470         rfltt = time_lag['RFLT'][:, 0, 0] + (data['RFLT'][:, 0, 0] - time_lag['RFLT'][:, 0, 0]) * t * dt
471
472         for r in range(len(titles['RTI'])):
473
474             if rfltt[r] == 0.0:
475                 continue
476
477             ##### FTT #####
478             # Initialise variables related to market share dynamics
479             # DSiK contains the change in shares
480             dSiK = np.zeros([len(titles['VTI']), len(titles['VTI'])])
481
482             # F contains the preferences
483             F = np.ones([len(titles['VTI']), len(titles['VTI'])])*0.5
484
485             #
486                 if int(data['TDA1'][r]) < year:
487
488                     # TODO: Check Specs dimensions
489                     #if np.any(specs[sector][r, :] == 1): # FTT Specification
490
491                     for v1 in range(len(titles['VTI'])):
492
493                         if not (data_dt['TEWS'][r, v1, 0] > 0.0 and
```

```
493         data_dt['TELC'][r, v1, 0] != 0.0 and
494         data_dt['TLCD'][r, v1, 0] != 0.0 and
495         data['TWSA'][r, v1, 0] < 0.0):
496     continue
497
498     S_veh_i = data_dt['TEWS'][r, v1, 0]
499     Aki = 0.5 * data['REVS'][r, v1, 0] / time_lag['TEWK'][r, v1, 0]
500
501     for v2 in range(v1):
502
503         if not (data_dt['TEWS'][r, v2, 0] > 0.0 and
504               data_dt['TELC'][r, v2, 0] != 0.0 and
505               data_dt['TLCD'][r, v2, 0] != 0.0 and
506               data['TWSA'][r, v2, 0] < 0.0):
507             continue
508
509             S_veh_k = data_dt['TEWS'][r, v2, 0]
510             Aik = 0.5 * data['REVS'][r, v2, 0] / time_lag['TEWK'][r, v2, 0] #Not using TWEA?
511
512             # Propagating width of variations in perceived costs
513             dFik = sqrt(2) * sqrt((data_dt['TLCD'][r, v1, 0]*data_dt['TLCD'][r, v1, 0] + data_dt
514                                ['TLCD'][r, v2, 0]*data_dt['TLCD'][r, v2, 0]))
515
516             # Consumer preference incl. uncertainty
517             Fik = 0.5*(1+np.tanh(1.25*(data_dt['TELC'][r, v2, 0]-data_dt['TELC'][r, v1, 0])/dFik))
518
519             # Preferences are then adjusted for regulations
520             F[v1, v2] = Fik*(1.0-isReg[r, v1]) * (1.0 - isReg[r, v2]) + isReg[r, v2]*(1.0-isReg[r, v1])
521             + 0.5*(isReg[r, v1]*isReg[r, v2])
522             F[v2, v1] = (1.0-Fik)*(1.0-isReg[r, v2]) * (1.0 - isReg[r, v1]) + isReg[r, v1]*(1.0-isReg
523             [r, v2]) + 0.5*(isReg[r, v2]*isReg[r, v1])
524
525             #Runge-Kutta market share dynamiccs
```

```

523     k_1 = S_veh_i*S_veh_k* (Aik*F[v1,v2] - Aki*F[v2,v1])
524     k_2 = (S_veh_i+dt*k_1/2)*(S_veh_k-dt*k_1/2)* (Aik*F[v1,v2] - Aki*F[v2,v1])
525     k_3 = (S_veh_i+dt*k_2/2)*(S_veh_k-dt*k_2/2) * (Aik*F[v1,v2] - Aki*F[v2,v1])
526     k_4 = (S_veh_i+dt*k_3)*(S_veh_k-dt*k_3) * (Aik*F[v1,v2] - Aki*F[v2,v1])
527
528     # Market share dynamics
529     #dSik[v1, v2] = S_veh_i*S_veh_k* (Aik*F[v1,v2] - Aki*F[v2,v1])*dt
530     dSik[v1, v2] = dt*(k_1+2*k_2+2*k_3+k_4)/6
531     dSik[v2, v1] = -dSik[v1, v2]
532
533     # Add in exogenous sales figures. These are blended with
534     # endogenous result! Note that it's different from the
535     # ExogSales specification!
536     Utot = rfltt[r]
537     dSk = np.zeros([len(titles['VTTI'])])
538     dUk = np.zeros([len(titles['VTTI'])])
539     dUkTK = np.zeros([len(titles['VTTI'])])
540     dUkREG = np.zeros([len(titles['VTTI'])])
541
542     # PV: Added a term to check that exogenous capacity is smaller than regulated capacity.
543     # Regulations have priority over exogenous capacity
544     reg_vs_exog = ((data['TWSA'][r, :, 0] + data_dt['TEWK'][r, :, 0]) > data['TREG'][r, :, 0]) & (data
545     ['TREG'][r, :, 0] >= 0.0)
546     data['TWSA'][r, :, 0] = np.where(reg_vs_exog, 0.0, data['TWSA'][r, :, 0])
547     TWSA_scalar = 1.0
548
549     # Check that exogenous sales additions aren't too large
550     # As a proxy it can't be greater than 80% of the fleet size
551     # divided by 13 (the average lifetime of vehicles)
552     if (data['TWSA'][r, :, 0].sum() > 0.8 * rfltt[r] / 13):
553         TWSA_scalar = data['TWSA'][r, :, 0].sum() / (0.8 * rfltt[r] / 13)
554

```

```
555 TWSA_gt_null = data['TWSA'][r, :, 0] >= 0.0
556 dUkTK = np.where(TWSA_gt_null, data['TWSA'][r, :, 0] * TWSA_scalar, 0.0)
557
558 # Correct for regulations
559 #Share of UED * change in UED * isReg i.e. change in UED split into technologies times isReg
560 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:
561
562     dUkREG = data_dt['TEWK'][r, :, 0] * (rfltt[r] - time_lag['RFLT'][r, 0, 0]) /
563         time_lag['RFLT'][r, 0, 0]) * isReg[r, :].reshape([len(titles['VTTI'])])
564 # Sum effect of exogenous sales additions (if any) with
565 # effect of regulations
566 dUk = dUkTK + dUkREG
567 dUtot = np.sum(dUk)
568 # Convert to market shares and make sure sum is zero
569 # dSk = dUk/Utot - Uk dUtot/Utot^2 (Chain derivative)
570 dSk = np.divide(dUk, Utot) - data_dt['TEWK'][r, :, 0]*np.divide(dUtot, (Utot*Utot))
571 #     soel = np.sum(dSik, axis=1)
572 #     st_1 = data_dt['TEWS'][r, :, 0]
573
574 # New market shares
575
576 #     print(np.sum(dSik, axis=1))
577 data['TEWS'][r, :, 0] = data_dt['TEWS'][r, :, 0] + np.sum(dSik, axis=1) + dSk
578
579 if ~np.isclose(np.sum(data['TEWS'][r, :, 0]), 1.0, atol=1e-3):
580     msg = ""Sector: {} - Region: {} - Year: {}
581         Sum of market shares do not add to 1.0 (instead: {})
582         """.format(sector, titles['RTI'][r], year, np.sum(data['TEWS'][r, :, 0]))
583     warnings.warn(msg)
584
585 if np.any(data['TEWS'][r, :, 0] < 0.0):
586     msg = ""Sector: {} - Region: {} - Year: {}
```

```
587         Negative market shares detected! Critical error!
588         """.format(sector, titles['RTI'][r], year)
589         warnings.warn(msg)
590
591
592
593     ##### Update variables #####
594
595     # Update demand for driving (in km/ veh/ y) - exogenous atm
596     #data['TEWL'][:, :, 0] = rvkmt[:, 0, 0]
597     # Vehicle composition
598     data['TEWK'][:, :, 0] = data['TEWS'][:, :, 0] * rfltt[:, np.newaxis]
599     # Total distance driven per vehicle type
600     data['TEWG'][:, :, 0] = data['TEWK'][:, :, 0] * rvkmt[:, np.newaxis] * 1e-3
601
602     # Sales are the difference between fleet sizes and the addition of scrapped vehicles
603     data['TEWI'][:, :, 0] = data['TEWK'][:, :, 0] - time_lag['TEWK'][:, :, 0] + data['REVS'][:, :, 0]
604
605     # Corrections to sales and EOL when sales are negative.
606     condition = data['TEWI'][:, :, 0] < 0.0
607     data['REVS'][:, :, 0] = np.where(condition,
608                                     data['REVS'][:, :, 0] - data['TEWI'][:, :, 0],
609                                     data['REVS'][:, :, 0])
610
611     data['TEWI'][:, :, 0] = np.where(condition,
612                                     0.0,
613                                     data['TEWI'][:, :, 0])
614
615
616     # Add sales to the age tracking matrix #Changing TP_VFLTA to RLTA
617     data['RLTA'][:, :, -1] = data['TEWI'][:, :, 0]
618
619     # Local variable to calculate age related weights:
```

```
620     share_by_age = np.zeros_like(data['RLTA'][:, :, :])
621     # Average vehicle age and average relative efficiency factor
622     for a, age in enumerate(enumerate(titles['VYTI'])):
623
624         share_by_age[:, :, a] = divide(data['RLTA'][:, :, a],
625                                        np.sum(data['RLTA'][:, :, :], axis=2))
626
627     # Age (NOT USED)
628     #data['TP_VAGE'][:, :, 0] = np.sum(share_by_age * np.asarray(enumerate(titles['VYTI']), axis=2))
629
630     # Efficiency factor #TETH is age by region, this changes it to region by technology
631     data['TEFF'][:, :, 0] = np.sum(share_by_age * data['TETH'][0, 0, :], axis=2)
632
633     # Fuel use
634     # Compute fuel use as distance driven times energy use, corrected by the biofuel mandate.
635     emis_corr = np.zeros([len(enumerate(titles['RTI'])), len(enumerate(titles['VTTI']))])
636     fuel_converter = copy.deepcopy(data['TJET'][0, :, :])
637     for r in range(len(enumerate(titles['RTI']))):
638
639
640         for fuel in range(len(enumerate(titles['JTI']))):
641
642             for veh in range(len(enumerate(titles['VTTI']))):
643
644                 if titles['JTI'][fuel] == '5 Middle distillates' and data['TJET'][0, veh, fuel] ==1: # ↗
645                     Middle distillates
646
647                     # Mix with biofuels if there's a biofuel mandate
648                     fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * (1.0 - data['RBFM'][r, 0, 0])
649
650                     # Emission correction factor
651                     emis_corr[r, veh] = 1.0 - data['RBFM'][r, 0, 0]
```

```

652         elif titles['JTI'][fuel] == '11 Biofuels' and data['TJET'][0, veh, fuel] == 1:
653
654             fuel_converter[veh, fuel] = data['TJET'][0, veh, fuel] * data['RBFM'][r, 0, 0]
655
656             #TODO this is broken
657
658             # Calculate fuel use - passenger car only! Therefore this will
659             # differ from E3ME results
660             # TP_FD: (TEWG)                ktoe
661             # TP_VTDD:                    km/veh
662             # Convert energy unit (1/41.868) ktoe/MJ
663             # Energy demand (BBTC)       MJ/km
664             # data['TJEF'][r, :, 0] = np.matmul(data['TEWG'][r, :, 0] *
665             #                                     data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] *
666             #                                     data['TEFF'][r, :, 0],
667             #                                     fuel_converter) / 41.868
668             #
669             # data['TVFP'][r, :, 0] = np.matmul(data['TE3P'][r, :, 0],
670             #                                     fuel_converter.T) / 41868 * \
671             #                                     data['BTTC'][r, :, c3ti['9 energy use (MJ/km)']] * \
672             #                                     data['TEFF'][r, :, 0]
673
674             # Emissions, unit MtCO2 - using vehicle emission factors, corrected by the biofuel mandate
675             data['TEWE'][:, :, 0] = data['TEWG'][:, :, 0] * \
676                 data['BTTC'][:, :, c3ti['14 CO2Emissions']] * \
677                 data['TEFF'][:, :, 0] * 1e-6 * 1.2 * emis_corr
678
679             # Since there's no dynamic fuel price calculation yet, keep it constant (NO USED)
680             #data['TP_VEC'][:, :, 0] = copy.deepcopy(data['BTTC'][:, :, c3ti['3 fuel cost (USD/km)']])
681
682             ##### Learning-by-doing #####
683
684             # Cumulative global learning

```

```

685     # Using a technological spill-over matrix (TEWB) together with capacity
686     # additions (TEWI) we can estimate total global spillover of similar
687     # vehicals
688 #     bi = np.matmul(data['TEWI'][:, :, 0], data['TEWB'][0, :, :])
689 #     dw = np.sum(bi, axis=0)*dt
690
691     bi = np.zeros((len(titles['RTI']),len(titles['VTTI'])))
692     for r in range(len(titles['RTI'])):
693         bi[r,:] = np.matmul(data['TEWB'][0, :, :],data['TEWI'][r, :, 0])
694     dw = np.sum(bi, axis=0)*dt
695
696     # Cumulative capacity incl. learning spill-over effects
697     data['TEWW'][0, :, 0] = data_dt['TEWW'][0, :, 0] + dw
698
699     # Copy over the technology cost categories that do not change (all except prices which are updated
        # through learning-by-doing below)
700     data['BTTC'] = copy.deepcopy(data_dt['BTTC'])
701
702     # Learning-by-doing effects on investment
703     for veh in range(len(titles['VTTI'])):
704
705         if data['TEWW'][0, veh, 0] > 0.1:
706
707             data['BTTC'][:, veh, c3ti['1 Prices cars (USD/veh)']] = data_dt['BTTC'][:, veh, c3ti['1 Prices
                cars (USD/veh)']] * \
708                                     (1.0 + data['BTTC'][:, veh, c3ti['16
                Learning exponent']] * dw[veh]/data['TEWW'][0, veh, 0])
709
710     # Investment in terms of car purchases:
711     for r in range(len(titles['RTI'])):
712
713         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

```



```
714
715
716     ##### Final output #####
717
718     # Get LCOT
719     if t ==1:
720         data = get_lcot(data, titles)
721
722     # Update time loop variables:
723     for var in data_dt.keys():
724
725         data_dt[var] = copy.deepcopy(data[var])
726
727     return data
728
```

```
1 # -*- coding: utf-8 -*-
2 """
3 =====
4 Multiple benefits.py
5 =====
6 Quntification of multiple benefits of energy efficiency.
7 #####
8
9 This is the main file for the quantification of the multiple benefits of
10 energy efficiency, resulting from the implementation of different policy options
11 to enhance energy efficiency in Europe. The program processes results from
12 E3ME LITE and performs off-model calculations for the quantification of beneits
13 that are not estimated within E3ME lite.
14
15 This program performs the following tasks:
16     1) Reads mock data from E3ME LITE for the Baseline and the selected Scenario
17     2) Processes mock data from E3ME LITE to convert them to the most suitable format
18     3) Estimates multiple benefits of energy efficiency that are not covered by E3ME LITE
19     4) Calculates absolute and percentage differences from the Baseline
20     5) Writes the final results to pickle files
21
22 Multiple benefits quantified in this program include:
23
24 Air pollution & Emissions, Air pollution Damage Costs, Employment,
25 Energy Cost Impact, Energy Intensity, Fossil Fuel Consumption, Fuel imports,
26 GDP, Gross Value Added, International competitiveness, Labour Productivity.
27 Material Use, Public budget as share of GDP, Share of energy consumption,
28 Water used in electricity generation.
29
30 @author: od
31 """
32
33
```

```
34 # Standard library imports
35 import pandas as pd
36 import numpy as np
37 from celib import DB1
38 import copy
39 import pickle
40 import os
41 import glob
42 from pathlib import Path
43 import time
44
45 # Record start time of the program
46 start = time.time()
47
48 directory = 'J:\Projects\DG Research\REFEREE (P1451)\Multiple benefits quantification\Python'
49 path = 'J:\Projects\DG Research\REFEREE (P1451)\Deliverables\FTT and E3ME info\E3ME export\Final_Workbooks'
50 names = ['Baseline', 'Scenario']
51 add = 'Country'
52
53 # Relevant variables to call at the data processing stage
54 aggregate_var_loc_1 = 550
55 gva_loc_1 = 20
56 gva_loc_2 = 479
57 employment_loc_1 = 92
58 employment_loc_2 = 407
59 enercon_loc_1 = 164
60 enercon_loc_2 = 380
61 price_loc_1 = 191
62 price_loc_2 = 353
63 expenditure_loc_1 = 218
64 expenditure_loc_2 = 307
65 energy_prices_loc_1 = 433
66 energy_prices_loc_2 = 3
```

```
67 output_loc_1 = 289
68 output_loc_2 = 210
69 generation_loc_1 = 263
70 generation_loc_2 = 282
71 material_loc_1 = 9
72 material_loc_2 = 7
73 imports_loc_1 = 361
74 imports_loc_2 = 138
75 coeff_loc_1 = 4
76 coeff_loc_2 = 5
77 lista_loc_1 = 20
78 lista_loc_2 = 479
79 deflating_factor = 0.929440948/0.874115426 # Source: GDP deflator World Bank
80
81 cols_mat = ['Country', 'Indicator', 2021, 2022, 2023, 2024, 2025, 2026, 2027,
82           2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039,
83           2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051,
84           2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063,
85           2064, 2065, 2066, 2067, 2068, 2069, 2070]
86
87 cols_ = cols_final = ['Country', 'Indicator', 2022, 2023, 2024, 2025, 2026, 2027,
88                    2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039,
89                    2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051,
90                    2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063,
91                    2064, 2065, 2066, 2067, 2068, 2069, 2070]
92 cols_final = ['Country', 'Indicator', 'Level of Disaggregation', 2022, 2023, 2024, 2025, 2026, 2027,
93             2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039,
94             2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051,
95             2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063,
96             2064, 2065, 2066, 2067, 2068, 2069, 2070]
97 cols_final_ = ['Pillar', 'Indicator', 'Level of Disaggregation', 'Country', 'Unit',
98              2022, 2023, 2024, 2025, 2026, 2027,
99              2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039,
```

```

100         2040,2041,2042, 2043,2044,2045,2046,2047,2048,2049,2050,2051,
101         2052,2053,2054,2055,2056,2057,2058,2059,2060,2061,2062,2063,
102         2064,2065,2066,2067,2068,2069,2070]
103 dicto = {0:'Electricity', 1: 'Gas', 2: 'Coal', 3: 'Petrol', 4: 'Oil'}
104 pillars = {'Gross Value Added':'Industrial Productivity',
105            'Energy Intensity':'Industrial Productivity',
106            'Energy Cost Impact': 'Industrial Productivity',
107            'International competitiveness':'Industrial Productivity',
108            'Labour Productivity':'Industrial Productivity',
109            'GDP': 'Socioeconomic Development',
110            'Employment':'Socioeconomic Development',
111            'Public budget as share of GDP': 'Socioeconomic Development',
112            'Price' : 'Socioeconomic Development',
113            'Consumer Expenditure' : 'Socioeconomic Development',
114            'Share of energy consumption_Q1' : 'Socioeconomic Development',
115            'Share of energy consumption_Q2' : 'Socioeconomic Development',
116            'Share of energy consumption_Q3' : 'Socioeconomic Development',
117            'Share of energy consumption_Q4' : 'Socioeconomic Development',
118            'Share of energy consumption_Q5' : 'Socioeconomic Development',
119            'Share of total space heat demand' : 'Socioeconomic Development',
120            'Air pollution Damage Costs':'Air quality and wellbeing',
121            'Air pollution & Emissions':'Environment & Climate',
122            'Fossil Fuel Consumption':'Environment & Climate',
123            'Water used in electricity generation' : 'Environment & Climate',
124            'Fuel imports' : 'Environment & Climate',
125            'Material Use' : 'Environment & Climate'}
126 fuels = ['Electricity', 'Gas', 'Other Fuels', 'Liquid Fuels']
127 agri = ['1 Crop production', '2 Forestry', ' 3 Fishing']
128 keep = ['BIOGAS', 'BIOMASS', 'COAL GASES', 'GEO-THERMAL', 'HARD COAL',
129         'HEAVY FUEL OIL', 'HYDRO', 'NATURAL GAS', 'NUCLEAR']
130
131 # %%
132 # -----

```

```

133 # ----- READ DATA INPUTS -----
134 # -----
135 if __name__ == '__main__':
136     os.chdir(directory)
137
138 ##### NON-E3ME LITE DATA #####
139 #%%
140 # Metadata
141 print('Reading data inputs')
142 metadata_units = pd.read_excel(r'Data\Units of measurement.xlsx', sheet_name='Sheet2').to_dict('records')
143 metadata_units = metadata_units[0]
144 ## Data Converters
145 # E3ME electricity sources to WRI classification (Unit: GWh -> kWh)
146 converter_wri = pd.read_excel('Data\Converter_E3ME electricity sources to WRI.xlsx', index_col=0).fillna(0)
147 # 70 E3ME sectors to broad sectors
148 converter_sectors = pd.read_csv('J:\Projects\DG Research\REFEREE (P1451)\Multiple benefits quantification
    \Python\Data\E3ME 70 Sector to Broad sector.csv',
149     index_col=0).fillna(0)
150 # Country codes lookup
151 country_lookup = pd.read_excel('Data\EU country code lookup.xlsx', sheet_name = 'E3ME', index_col=0)
152 country_lookup_bsm = pd.read_excel('Data\EU country code lookup.xlsx', sheet_name = 'BSM', index_col=0)
153 conv = pd.read_excel('Data\BSM_data.xlsx', sheet_name='Converter').fillna(0).set_index('Archetype')
154 # Damage costs (Unit: Euro per kg)
155 damage_costs = pd.read_excel('Data\Damage costs.xlsx',
156     index_col=0).div(0.001 )# convert from EUR per kg to EUR per tonne
157 damage_costs = damage_costs * deflating_factor # deflate prices to EUR 2010
158 # Water withdrawal coefficients (Unit: gal/kWh)
159 water_coefficients = pd.read_excel('Data\Water coefficients.xlsx',
160     sheet_name = 'Sheet1', index_col=0)
161 water_coefficients = water_coefficients.loc[water_coefficients.index.intersection(list(damage_costs.index)
162     +['UK'])]
162 water_coefficients = water_coefficients.apply(lambda x: x.fillna(x.mean()),axis = 0)
163 water_coefficients = water_coefficients.reset_index()

```

```
164 water_coefficients_long = pd.melt(water_coefficients, id_vars = ['index'],
165                                   var_name = 'Level of Disaggregation', value_name='values')
166 water_coefficients_long = water_coefficients_long.rename(columns = {'index':'Country'})
167 # Eurostat shares of overall consumption spent on energy
168 shares = pd.read_excel('Data\Shares.xls', sheet_name= 'hbs_str_t223',
169                       skiprows = 3, index_col=0)
170 split = pd.read_excel('Data\Energy consumption split.xlsx')
171 split = split.set_index(['Unnamed: 0', 'Unnamed: 1'])
172 # Break down Eurostat shares by energy source using split dataframe
173 e = shares.mul(split.loc[('Italy', 'Electricity')])
174 e['Level of Disaggregation'] = 'Electricity'
175 g = shares.mul(split.loc[('Italy', 'Gas')])
176 g['Level of Disaggregation'] = 'Gas'
177 lf = shares.mul(split.loc[('Italy', 'Liquid fuels')])
178 lf['Level of Disaggregation'] = 'Liquid Fuels'
179 of = shares.mul(split.loc[('Italy', 'Other fuels')])
180 of['Level of Disaggregation'] = 'Other Fuels'
181 detailed_shares = pd.concat([e,g,lf,of],axis=0).reset_index()
182 detailed_shares = detailed_shares.set_index(['index', 'Level of Disaggregation'])
183 # Reshape shares of energy consumption to a more useful format
184 detailed_shares_q1 = pd.DataFrame(detailed_shares.loc[:, 'First quintile'])
185 detailed_shares_q2 = pd.DataFrame(detailed_shares.loc[:, 'Second quintile'])
186 detailed_shares_q3 = pd.DataFrame(detailed_shares.loc[:, 'Third quintile'])
187 detailed_shares_q4 = pd.DataFrame(detailed_shares.loc[:, 'Fourth quintile'])
188 detailed_shares_q5 = pd.DataFrame(detailed_shares.loc[:, 'Fifth quintile'])
189
190 # %%
191 # Read coefficients from material flo accounts tool
192 coeff = pd.read_excel('Data\EarlyEstimates Tool_V9_NEW.xlsb.xlsm',
193                     sheet_name = 'Results Set',
194                     skiprows = coeff_loc_1, skipfooter= coeff_loc_2,
195                     usecols = 'B:E', index_col = 0).dropna()
196 coeff['Country'] = coeff.index.map(country_lookup['Country'])
```

```

197     coeff = coeff.reset_index(drop=True)
198     coeff = coeff.rename(columns={'Unnamed: 2':'Material',
199                                 'Unnamed: 3':'Coefficient',
200                                 'Unnamed: 4':'Value'})
201     c = ['Material', 'Coefficient', 'Country', 'Value']
202     coeff = coeff[c]
203
204     # %%
205     # Check detailed_shares sum to shares
206 #     detailed_shares = detailed_shares.reset_index()
207 #     detailed_shares = [g for _,g in detailed_shares.groupby('index')]
208 #     detailed_shares_check = []
209 #     for i in detailed_shares:
210 #         i = i.set_index('Level of Disaggregation')
211 #         i.loc['Total'] = i.sum(numeric_only=True, axis=0)
212 #         i = i.fillna(method='ffill')
213 #         i = round(i,3)
214 #         detailed_shares_check.append(i)
215 #     detailed_shares_check = pd.concat(detailed_shares_check).reset_index()
216 #     detailed_shares_check = detailed_shares_check[detailed_shares_check['Level of Disaggregation'] == 'Total']
217 #     detailed_shares_check = detailed_shares_check.set_index('index')
218 #     detailed_shares_check = detailed_shares_check.reindex(shares.index)
219 #
220 #     detailed_shares_check.iloc[:,1:] == shares
221
222 # %% ##### BUILDING STOCK MODEL DATA #####
223
224     bsm = pd.read_excel('Data\BSM_data.xlsx').rename(columns = {'Unnamed: 0':'helper'}).dropna()
225     centry = []
226     for i in list(set(bsm.loc[:, 'helper'])):
227         if len(i) == 2:
228             centry.append(i)
229     centry.sort()

```



```
230 df_bsm = pd.DataFrame()
231 for cn in cntry:
232     ind = bsm[bsm.loc[:, 'helper']==cn].index[0]
233     start = ind+1
234     end = ind+21
235     df_sto = bsm.loc[start:end, :]
236     df_sto = df_sto.copy()
237     df_sto.loc[:, 'Country code'] = cn
238     df_bsm = pd.concat([df_bsm, df_sto]).reset_index(drop=True)
239 df_bsm=df_bsm.rename(columns={'helper': 'Age'})
240
241
242 # %%
243 ##### E3ME LITE DATA #####
244
245 # -----
246 # Aggregate variables
247 # -----
248
249 print('Beginning data processing')
250 print('Step 1')
251 final_dataset = {}
252 for name in names:
253     dataset = {}
254     aggregate_var = pd.concat([
255         pd.read_excel(file, sheet_name=name, skipfooter = aggregate_var_loc_1,
256                       usecols = 'A,N:BJ')
257         .assign(Country=file.stem)
258         .rename(columns={'Unnamed: 0': 'Variable'})
259     for file in Path(path).glob('*.*xlsx')]).dropna()
260     cols = list(aggregate_var.columns)
261     cols = [cols[-1]] + cols[:-1]
262     aggregate_var = aggregate_var[cols]
```

```
263     aggregate_var.index = np.arange(0, len(aggregate_var))
264     aggregate_var = aggregate_var.rename(columns={'Variable' : 'Indicator'})
265     aggregate_var = aggregate_var[aggregate_var.Country != 'World']
266
267 # %%
268 # -----
269 # Gross Value Added
270 # -----
271 print('Step 2')
272 gva_list = ([pd.read_excel(file, sheet_name=name, skiprows = gva_loc_1,
273                          skipfooter = gva_loc_2, usecols = 'A,N:BJ', index_col = 0).T
274              .dot(converter_sectors).T
275                .assign(Country=file.stem)
276                for file in Path(path).glob('*.xlsx')])
277
278 for v in gva_list:
279     v.loc['Total'] = v.sum(axis=0, numeric_only=True)
280     v.ffill(inplace=True)
281 gva = pd.concat(gva_list).reset_index()
282
283 # Order columns
284 cols = list(gva.columns)
285 cols = [cols[-1]] + cols[:-1]
286 gva = gva[cols]
287 gva.columns = aggregate_var.columns
288 gva = gva.rename(columns={'Indicator':'Level of Disaggregation'})
289 gva['Indicator'] = 'Gross Value Added'
290 gva = gva[cols_final]
291 metadata_sectors = gva.iloc[:, 0:3]
292
293 #Extract Total GVA by Member State
294 gva_tot = gva[gva['Level of Disaggregation'] == 'Total'].reset_index(drop=True)
295 world_gva = gva_tot.iloc[29,3:]
```

```

296     gva = gva[gva.Country != 'World']
297     dataset['Gross Value Added']= gva
298     # %%
299     # -----
300     # Employment
301     # -----
302     print('Step 3')
303     employment_list = ([
304     pd.read_excel(file,
305                   sheet_name=name, skiprows = employment_loc_1,
306                   skipfooter = employment_loc_2,
307                   usecols = 'A,N:BJ', index_col=0).T
308     .dot(converter_sectors).T
309     .assign(Country=file.stem)
310     #.rename(columns={'Unnamed: 0': 'Variable'})
311     for file in Path(path).glob('*.xlsx')])
312
313     employment = pd.concat(employment_list).reset_index()
314
315     # Order columns
316     cols = list(employment.columns)
317     cols = [cols[-1]] + cols[:-1]
318     employment = employment[cols]
319     employment.columns = aggregate_var.columns
320     employment = employment.rename(columns={'Indicator': 'Level of Disaggregation'})
321     employment['Indicator'] = 'Employment'
322     employment = employment[cols_final].reset_index(drop=True)
323     employment = employment[employment.Country != 'World']
324     dataset['Employment'] = employment
325
326     # %%
327     # -----
328     # Energy consumption

```

```
329 # -----
330
331 print('Step 4')
332 enercon_list = ([pd.read_excel(file, sheet_name=name, skiprows = enercon_loc_1,
333                             skipfooter = enercon_loc_2, usecols = 'A,N:BJ', index_col = 0)
334                  .assign(Country=file.stem)
335 for file in Path(path).glob('*xlsx')])
336 energy_consumption = pd.concat(enercon_list).reset_index()
337
338 # Order columns
339 cols = list(energy_consumption.columns)
340 cols = [cols[-1]] + cols[:-1]
341 energy_consumption = energy_consumption[cols]
342 energy_consumption.columns = aggregate_var.columns
343 energy_consumption = energy_consumption.rename(columns={'Indicator':'Level of Disaggregation'})
344 energy_consumption['Indicator'] = 'Energy Consumption'
345 energy_consumption = energy_consumption[cols_final].reset_index(drop=True)
346 energy_consumption = energy_consumption[energy_consumption.Country != 'World']
347
348 # %%
349 # -----
350 # Prices
351 # -----
352 print('Step 5')
353 price_list = ([pd.read_excel(file, sheet_name=name, skiprows = price_loc_1,
354                             skipfooter = price_loc_2, usecols = 'A,N:BJ', index_col = 0)
355               .assign(Country=file.stem)
356 for file in Path(path).glob('*xlsx')])
357 average_price = pd.concat(price_list).reset_index()
358
359 # Order columns
360 cols = list(average_price.columns)
361 cols = [cols[-1]] + cols[:-1]
```

```
362     average_price = average_price[cols]
363     average_price.columns = aggregate_var.columns
364     average_price = average_price.rename(columns={'Indicator':'Level of Disaggregation'})
365     average_price['Indicator'] = 'Average prices'
366     average_price = average_price[cols_final].reset_index(drop=True)
367     average_price = average_price[average_price.Country != 'World']
368
369     # %%
370     # -----
371     # Derive average energy cist from Prices
372     # -----
373     print('Step 6')
374     energy_cost = energy_consumption.iloc[:,3:].mul(average_price.iloc[:,3:])
375     energy_cost = pd.concat([energy_consumption.iloc[:,0:3],energy_cost], axis=1)
376     energy_cost['Indicator'] = 'Energy Cost'
377
378     energy_cost_tot = energy_cost.groupby(by=['Country']).sum()
379     energy_cost_tot['Indicator'] = 'Total Energy Cost'
380     cols = list(energy_cost_tot.columns)
381     cols = [cols[-1]] + cols[:-1]
382     energy_cost_tot = energy_cost_tot[cols].reset_index()
383     energy_cost_tot = energy_cost_tot[energy_cost_tot['Country'] != 'World']
384
385     # %%
386     # -----
387     # Estimate total energy consumption
388     # -----
389     print('Step 7')
390     for v in enercon_list:
391         v.loc['Total'] = v.sum(axis=0, numeric_only=True)
392         v.ffill(inplace=True)
393     energy_consumption = pd.concat(enercon_list).reset_index()
394
```

```
395     # Order columns
396     cols = list(energy_consumption.columns)
397     cols = [cols[-1]] + cols[:-1]
398     energy_consumption = energy_consumption[cols]
399     energy_consumption.columns = aggregate_var.columns
400     energy_consumption = energy_consumption.rename(columns={'Indicator':'Level of Disaggregation'})
401     energy_consumption['Indicator'] = 'Energy Consumption'
402     energy_consumption = energy_consumption[cols_final]
403
404     # Extract data on Total Energy Consumption by Member State
405     energy_consumption_tot = energy_consumption[energy_consumption['Level of Disaggregation'] ==
406         'Total'].reset_index(drop=True)
407     energy_consumption_tot = energy_consumption_tot[energy_consumption_tot['Country'] != 'World']
408 # %%
409 # -----
410 # Consumer Expenditure
411 # -----
412
413 print('Step 8')
414 consumer_expenditure = pd.concat([
415     pd.read_excel(file,
416         sheet_name=name, skiprows = expenditure_loc_1,
417         skipfooter = expenditure_loc_2,
418         usecols = 'A,N:BJ')
419     .assign(Country=file.stem) # We may also want file.name here
420     .rename(columns={'Unnamed: 0':'Variable'})
421     for file in Path(path).glob('*.xlsx')]).dropna()
422
423 # Order columns
424 cols = list(consumer_expenditure.columns)
425 cols = [cols[-1]] + cols[:-1]
426 consumer_expenditure = consumer_expenditure[cols]
427 consumer_expenditure.columns = aggregate_var.columns
```

```
427     consumer_expenditure = consumer_expenditure.rename(columns={'Indicator':'Level of Disaggregation'})
428     consumer_expenditure['Indicator'] = 'Consumer Expenditure'
429     consumer_expenditure = consumer_expenditure[cols_final]
430     consumer_expenditure = consumer_expenditure[consumer_expenditure.Country != 'World']
431
432     # Extract consumer expenditure for electricity, gas, liquid fuels and other fuels
433     electricity = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '9'           ↗
         Electricity'].reset_index(drop=True)
434     electricity['Level of Disaggregation'] = 'Electricity'
435
436     gas = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '10 Gas'].reset_index   ↗
         (drop=True)
437     gas['Level of Disaggregation'] = 'Gas'
438
439     liquid_f = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '11 Liquid       ↗
         Fuels'].reset_index(drop=True)
440     liquid_f['Level of Disaggregation'] = 'Liquid Fuels'
441
442     other_f = consumer_expenditure[consumer_expenditure['Level of Disaggregation'] == '12 Other       ↗
         Fuels'].reset_index(drop=True)
443     other_f['Level of Disaggregation'] = 'Other Fuels'
444
445     con_exp_ener = pd.concat([electricity, gas, liquid_f, other_f], axis=0)
446     dataset['Consumer Expenditure'] = con_exp_ener
447
448     # %%
449     # -----
450     # Gross Domestic Product
451     # -----
452
453     print('Step 9')
454     gdp = aggregate_var.loc[aggregate_var['Indicator'] == 'GDP'].reset_index(drop=True)
455     dataset['Gross Domestic Product'] = gdp
```

```
456
457     # Extract fossil fuel consumption
458     coal = aggregate_var.loc[aggregate_var['Indicator'] == 'Coal fuel consumption'].reset_index(drop=True)
459     gas = aggregate_var.loc[aggregate_var['Indicator'] == 'Gas fuel consumption'].reset_index(drop=True)
460     oil = aggregate_var.loc[aggregate_var['Indicator'] == 'Oil Consumption (Crude oil, Heavy fuel Oil and
         Middle distallates)'].reset_index(drop=True)
461     fossil_fuels = pd.concat([coal, gas, oil])
462     fossil_fuels = fossil_fuels.rename(columns={'Indicator': 'Level of Disaggregation'})
463     fossil_fuels['Indicator'] = 'Fossil Fuel Consumption'
464     fossil_fuels = fossil_fuels[cols_final]
465     dataset['Fossil Fuel Consumption'] = fossil_fuels
466
467     # %%
468     # -----
469     # Energy prices
470     # -----
471
472     print('Step 10')
473     elec_list = ([pd.read_excel(file, sheet_name=name, skiprows = energy_prices_loc_1,
474                             skipfooter = energy_prices_loc_2, usecols = 'A,N:BJ', index_col = 0)
475                  .assign(Country=file.stem)
476
477     for file in Path(path).glob('*.*xlsx')])
478     extracted_cols = list(aggregate_var.iloc[:,2:].columns)
479     extracted_cols.append(add)
480     new = []
481     for i in elec_list:
482         n = i.loc['19 Households']
483         n.columns = extracted_cols
484         w = n.iloc[3:,:].mean()
485         n = n.append(w, ignore_index=True).fillna(method='ffill').drop([3,4], axis=0).reset_index(drop=True)
486         n['Level of Disaggregation'] = fuels
487         n['Indicator'] = 'Price'
```



```

488         n = n[cols_final]
489         new.append(n)
490         new_dataset = pd.concat(new)
491         dataset['Price'] = new_dataset
492
493 # %%
494 # -----
495 # Gross output (Unit: €2010m)
496 # -----
497
498 print('Step 11')
499 output_list = ([pd.read_excel(file, sheet_name=name, skiprows = output_loc_1,
500                             skipfooter = output_loc_2, usecols = 'A,N:BJ', index_col = 0)
501                 .assign(Country=file.stem)
502                 for file in Path(path).glob('*.*xlsx')])
503
504 for v in output_list:
505     v.loc['Total'] = v.sum(axis=0, numeric_only=True)
506     v.ffill(inplace=True)
507 output = pd.concat(output_list).reset_index()
508
509 # Order columns
510 cols = list(output.columns)
511 cols = [cols[-1]] + cols[:-1]
512 output = output[cols]
513 output.columns = aggregate_var.columns
514 output = output.rename(columns={'Indicator':'Level of Disaggregation'})
515 output['Indicator'] = 'Gross Output'
516 output = output[output['Country'] != 'World']
517 output = output[cols_final]
518 # %%
519 # -----
520 # Electricity generation

```

```

521 # -----
522
523 print('Step 12')
524 generation_list = ([pd.read_excel(file, sheet_name=name, skiprows = generation_loc_1,
525                               skipfooter = generation_loc_2, usecols = 'A,N:BJ', index_col = 0).T
526                               .dot(converter_wri).T
527                               .mul(1000000) #Conversion fro GWh to kwh
528                               .assign(Country=file.stem)
529 for file in Path(path).glob('*.*xlsx')])
530 generation = pd.concat(generation_list).reset_index()
531
532 # Order columns
533 cols = list(generation.columns)
534 cols = [cols[-1]] + cols[:-1]
535 generation = generation[cols]
536 generation.columns = aggregate_var.columns
537 generation = generation.rename(columns={'Indicator':'Level of Disaggregation'})
538 generation['Indicator'] = 'Electricity generation'
539 generation = generation[cols_final]
540 generation = generation[(generation.Country != 'World') & (generation.Country != 'EU27')]
541 generation = generation[generation['Level of Disaggregation'].isin(keep)]
542 ###
543 # -----
544 # Material Use for 2021 from Eurostat (Unit: Thousands tonnes)
545 # -----
546
547 print('Step 13')
548 material_use_eurostat = pd.read_excel('Data\env_ac_mfa__custom_3549625_page_spreadsheet.xlsx',
549                                     sheet_name = 'Sheet 1',
550                                     skiprows = material_loc_1, skipfooter = material_loc_2,
551                                     usecols = 'A:C', index_col = 0)
552 material_use_eurostat = material_use_eurostat.loc[material_use_eurostat.index.isin(list
    (damage_costs.index) +['UK'])] #add loc here? copy warning

```



```

553  ###
554
555      # -----
556      # Energy Imports (Unit: €2010m)
557      # -----
558
559      print('Step 14')
560      imports = ([pd.read_excel(file,
561                             sheet_name=name, skiprows = imports_loc_1, skipfooter = imports_loc_2,
562                             usecols = 'A,N:BJ', index_col=0)
563                 .assign(Country=file.stem)
564                 #.rename(columns={'Unnamed: 0': 'Variable'}))
565      for file in Path(path).glob('*.xlsx')]
566
567      imports = pd.concat(imports).reset_index()
568
569      # Order columns
570      cols = list(imports.columns)
571      cols = [cols[-1]] + cols[:-1]
572      imports = imports[cols]
573      imports.columns = aggregate_var.columns
574      imports = imports.rename(columns={'Indicator': 'Level of Disaggregation'})
575      imports['Indicator'] = 'Fuel imports'
576      imports = imports[cols_final].reset_index(drop=True)
577      imports = imports[imports.Country != 'World']
578      imports = imports[(imports['Level of Disaggregation'].isin(['4 Coal',
579                                                                '5 Oil and Gas', '12 Manufactured fuels']))].reset_index(drop=True)
580  # %% main function
581  # -----
582  # -----QUANTIFICATION OF MULTIPLE BENEFITS -----
583  # -----
584
585      # =====

```

```

586     # Labour productivity by country (Unit: €2010m per worker)
587     # =====
588
589     print('Beginning of the multiple benefits quantifications')
590     print('Step 15')
591     use_gva = gva[gva['Level of Disaggregation'] != 'Total']
592     use_gva = use_gva.reset_index(drop=True)
593     productivity = use_gva.iloc[:, 3:].div((employment.iloc[:, 3:]*1000))
594     productivity = pd.concat([employment.iloc[:,0:3], productivity], axis=1)
595     productivity['Indicator'] = 'Labour Productivity'
596
597     metadata_sectors = metadata_sectors[metadata_sectors['Level of Disaggregation'] != 'Total']
598     dataset['Labour Productivity'] = productivity
599
600 # %%
601 # =====
602 # Energy intensity (Unit: ktoe per €2010m)
603 # =====
604
605     print('Step 15')
606     energy_intensity = energy_consumption_tot.iloc[:,3:].div(gva_tot.iloc[0:29, 3:])
607     energy_intensity = pd.concat([energy_consumption_tot.iloc[:, 0:3],
608                                   energy_intensity], axis=1)
609     energy_intensity['Indicator'] = 'Energy Intensity'
610     energy_intensity = energy_intensity[energy_intensity['Country'] != 'World']
611     dataset['Energy Intensity'] = energy_intensity
612 # %%
613 # =====
614 # Energy cost impact (Unit: €2010m)
615 # =====
616
617     print('Step 16')
618     energy_cost_impact = energy_cost_tot.iloc[:, 2:].div(gva_tot.iloc[0:29, 3:])

```

```

619     energy_cost_impact = pd.concat([energy_consumption_tot.iloc[:, 0:3],
620                                   energy_cost_impact], axis=1)
621     energy_cost_impact['Indicator'] = 'Energy Cost Impact'
622     dataset['Energy Cost Impact'] = energy_cost_impact
623     # %%
624     # =====
625     # International competitiveness (Unit: ratio)
626     # =====
627
628     print('Step 17')
629     int_comp = gva_tot.iloc[0:29,3:]/gva_tot.iloc[29,3:]
630     int_comp = pd.concat([gva_tot.iloc[0:29,0:3], int_comp], axis=1)
631     int_comp['Indicator'] = 'International competitiveness'
632     dataset['International Competitiveness'] = int_comp
633     # %%
634     # =====
635     # Public budget as share of GDP (Unit: ratio)
636     # =====
637
638     print('Step 18')
639     gov_exp = aggregate_var.loc[aggregate_var['Indicator'] == 'Total government expenditure'].reset_index
640     (drop=True)
641
642     # Finally compute public budget as share of GDP
643     public_budget = gov_exp.iloc[:,2:].div(gdp.iloc[:,2:])
644     public_budget = pd.concat([gdp.iloc[:,0:2], public_budget], axis=1)
645     public_budget = public_budget.rename(columns={'Variable' : 'Indicator'})
646     public_budget['Indicator'] = 'Public budget as share of GDP'
647     public_budget['Level of Disaggregation'] = 'Total'
648     dataset['Public budget as share of GDP'] = public_budget
649
650     # %%
651     # =====

```

```

651 # Energy independence
652 # =====
653
654 print('Step 19')
655 #Extract Total GVA by Member State
656 output_tot = output[output['Level of Disaggregation'] == 'Total'].reset_index(drop=True)
657
658 # Finally, estimate energy imporst as a share of gross output (Unit: %)
659 imports_shares = imports.set_index('Country').iloc[:,2:].div(output_tot.set_index('Country').iloc[:,2:])
660 imports_shares = imports_shares.reset_index()
661 imports_shares['Indicator'] = 'Fuel imports'
662 imports_shares = imports_shares.join(imports['Level of Disaggregation'])
663 imports_shares = imports_shares[cols_final]
664 dataset['Fuel imports'] = imports_shares
665
666 # %%
667 # =====
668 # Water used for electricity generation (Unit: gal)
669 # =====
670
671 print('Step 20')
672 #Put electricity generation data in the right format for matrix multiplication
673 water_coefficients_long =water_coefficients_long[(water_coefficients_long['Level of Disaggregation'].isin
        (keep))]
674
675 water_coefficients_long_clean = water_coefficients_long.set_index(['Country','Level of Disaggregation'])
676 generation_clean=generation.drop('Indicator', axis=1)
677 generation_clean = generation_clean.set_index(['Country','Level of Disaggregation'])
678 for i in generation_clean.columns:
679     water_coefficients_long_clean[i] = water_coefficients_long_clean['values']
680 water_coefficients_long_clean=water_coefficients_long_clean.drop('values', axis=1)
681
682 # Estimate water use per unit of electricity generated by source (Unit: gallon)

```

```
683     water_use = generation_clean * water_coefficients_long_clean
684     water_use = water_use.reset_index()
685     water_use['Indicator'] = 'Water used in electricity generation'
686     water_use = water_use[cols_final]
687     dataset['Water used in electricity generation'] = water_use
688
689     # Note: Zero values in the water use dataset show that
690     # either no electricity is produced from X source in Y country (1) or
691     # electricity generation from X source in Y country is not associated with any water withdrawal (2)
692
693     ##%
694     # =====
695     # Air pollution costs
696     # =====
697
698     print('Step 21')
699     nox = aggregate_var.loc[aggregate_var.Indicator == 'Nox Emissions']
700     so2 = aggregate_var.loc[aggregate_var.Indicator == 'SO2 Emissions']
701     ch4 = aggregate_var.loc[aggregate_var.Indicator == 'CH4 Emissions']
702     voc = aggregate_var.loc[aggregate_var.Indicator == 'VOC Emissions']
703     pm = aggregate_var.loc[aggregate_var.Indicator == 'PM10 Emissions']
704     co2 = aggregate_var.loc[aggregate_var.Indicator == 'CO2 Emissions']
705     emissions = pd.concat([nox, so2, ch4, voc, pm, co2])
706     emissions = emissions.rename(columns={'Indicator': 'Level of Disaggregation'})
707     emissions['Indicator'] = 'Air pollution & Emissions'
708     emissions = emissions[cols_final]
709     dataset['Air pollution & Emissions'] = emissions
710
711     #NOx Damage costs (Unit: Euro)
712     nox = nox.drop('Indicator', axis=1)
713     nox = nox[0:28].set_index('Country').mul(1000)
714     nox_cost = nox.multiply(damage_costs['nox'], axis=0).reset_index()
715     nox_cost = nox_cost.rename(columns={'index' : 'Country'})
```

```
716     nox_cost['Level of Disaggregation'] = 'NOX Damage Cost'
717     nox_cost['Indicator'] = 'Air pollution Damage Costs'
718     nox_cost = nox_cost[cols_final]
719     #SO2 Damage costs (Unit: Euro)
720     so2 = so2.drop('Indicator', axis=1)
721     so2 = so2[0:28].set_index('Country').mul(1000)
722     so2_cost = so2.multiply(damage_costs['so2'], axis=0).reset_index()
723     so2_cost = so2_cost.rename(columns={'index' : 'Country'})
724     so2_cost['Level of Disaggregation'] = 'SO2 Damage Cost'
725     so2_cost['Indicator'] = 'Air pollution Damage Costs'
726     so2_cost = so2_cost[cols_final]
727
728     #VOC Damage costs (Unit: Euro)
729     voc = voc.drop('Indicator', axis=1)
730     voc = voc[0:28].set_index('Country').mul(1000)
731     voc_cost = voc.multiply(damage_costs['voc'], axis=0).reset_index()
732     voc_cost = voc_cost.rename(columns={'index' : 'Country'})
733     voc_cost['Level of Disaggregation'] = 'VOC Damage Cost'
734     voc_cost['Indicator'] = 'Air pollution Damage Costs'
735     voc_cost = voc_cost[cols_final]
736
737     #PM Damage costs (Unit: Euro)
738     pm = pm.drop('Indicator', axis=1)
739     pm = pm[0:28].set_index('Country').mul(1000)
740     pm_cost = pm.multiply(damage_costs['pm'], axis=0).reset_index()
741     pm_cost = pm_cost.rename(columns={'index' : 'Country'})
742     pm_cost['Level of Disaggregation'] = 'PM Damage Cost'
743     pm_cost['Indicator'] = 'Air pollution Damage Costs'
744     pm_cost = pm_cost[cols_final]
745
746     damages = pd.concat([nox_cost, so2_cost, voc_cost, pm_cost])
747     dataset['Air pollution Damages'] = damages
748
```



```
749 # %%
750 # =====
751 # Domestic material extraction
752 # =====
753
754 print('Step 22')
755 # Read GVA for relevant sectors
756 lista = ([pd.read_excel(file, sheet_name=name, skiprows = lista_loc_1,
757                      skipfooter = lista_loc_2, usecols = 'A,M:BJ', index_col = 0)
758           .assign(Country=file.stem)
759           for file in Path(path).glob('*.xlsx')])
760
761 mat = pd.concat(lista)
762 mat = mat.reset_index()
763 cols = list(mat.columns)
764 cols = [cols[-1]] + cols[:-1]
765 mat = mat.loc[:,cols]
766 mat = mat.set_axis(cols_mat, axis=1)
767
768 lista2 = []
769 for i in lista:
770     j = i.iloc[0:3]
771     j.loc['Total'] = j.sum(axis=0, numeric_only=True)
772     j.ffill(inplace=True)
773     j = j.reset_index()
774     j = (j[cols]).set_axis(cols_mat, axis=1)
775     lista2.append(j)
776
777 lista3 = []
778 for i in lista:
779     w = i.iloc[3:6]
780     w.loc['Total'] = w.sum(axis=0, numeric_only=True)
781     w.ffill(inplace=True)
```

```
782     w = w.reset_index()
783     w = (w[cols]).set_axis(cols_mat, axis=1)
784     lista3.append(w)
785
786     mf1 = (pd.concat(lista2))
787     mf1 = mf1[mf1['Indicator'].isin(['Total'])].reset_index(drop=True).drop('Indicator', axis=1)
788     mf1_p = mf1.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
789     mf2 = mat[mat['Indicator'].isin(['18 Metal products'])].reset_index(drop=True).drop('Indicator', axis=1)
790     mf2_p = mf2.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
791     mf3 = mat[mat['Indicator'].isin(['30 Construction'])].drop('Indicator', axis=1)
792     mf3_p = mf3.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
793     mf4 = (pd.concat(lista3))
794     mf4 = mf4[mf4['Indicator'].isin(['Total'])].reset_index(drop=True).drop('Indicator', axis=1)
795     mf4_p = mf4.set_index('Country').pct_change(axis=1).drop(['EU27', 'UK', 'World'], axis=0)
796 # %%
797     # Estimate material use
798
799     # Biomass (Code: MF1)
800     print('Step 23')
801     alpha_mf1 = coeff[(coeff['Material'] == 'MF1') & (coeff['Coefficient'] == 'Intercept')].drop      ↗
802     (['Coefficient', 'Material'], axis=1).set_index(['Country'])
803     beta_mf1 = coeff[(coeff['Material'] == 'MF1') & (coeff['Coefficient'] == 'P1')].drop(['Coefficient',      ↗
804     'Material'], axis=1).set_index(['Country'])
805     eurostat_mf1 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Biomass'].drop      ↗
806     ('Level of Disaggregation', axis=1)
807
808     relative_change_mf1 = pd.DataFrame()
809     mf1_p = mf1_p.reindex(beta_mf1.index)
810     for i in mf1_p.columns:
811         relative_change_mf1[i] =alpha_mf1['Value'].add((beta_mf1['Value'].mul(mf1_p[i])))
812     relative_change_mf1 = relative_change_mf1.drop(2021, axis=1)
813
814     years_ = list(relative_change_mf1.columns)
```

```
812
813     final_mf1 = eurostat_mf1.copy()
814     for ye in years_:
815
816         year = ye
817
818         year_1 = int(ye)-1
819
820         euro = final_mf1[year_1]
821         rel_ch = 1+relative_change_mf1[year]
822         calc = euro*rel_ch
823         calc = pd.DataFrame(calc)
824         calc.columns = [year]
825         final_mf1[year] = calc[year]
826         final_mf1['Level of Disaggregation'] = 'Biomass'
827     # %%
828     # Metal Ores (Code: MF2)
829     print('Step 24')
830     alpha_mf2 = coeff[(coeff['Material'] == 'MF2') & (coeff['Coefficient'] == 'Intercept')].drop      ↗
831     (['Coefficient', 'Material'], axis=1).set_index(['Country'])
832     beta_mf2 = coeff[(coeff['Material'] == 'MF2') & (coeff['Coefficient'] == 'P10')].drop(['Coefficient',      ↗
833     'Material'], axis=1).set_index(['Country'])
834     eurostat_mf2 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Metal ores      ↗
835     (gross ores)'].drop('Level of Disaggregation', axis=1)
836
837     relative_change_mf2 = pd.DataFrame()
838     mf2_p = mf2_p.reindex(beta_mf2.index)
839     for i in mf2_p.columns:
840         relative_change_mf2[i] =alpha_mf2['Value'].add((beta_mf2['Value'].mul(mf2_p[i])))
841     relative_change_mf2 = relative_change_mf2.drop(2021, axis=1)
842
843     final_mf2 = eurostat_mf2.copy()
844     for ye in years_:
```

```
842
843     year = ye
844
845     year_1 = int(ye)-1
846
847     euro = final_mf2[year_1]
848     rel_ch = 1+relative_change_mf2[year]
849     calc = euro*rel_ch
850     calc = pd.DataFrame(calc)
851     calc.columns = [year]
852     final_mf2[year] = calc[year]
853     final_mf2['Level of Disaggregation'] = 'Metal Ores'
854
855 # %%
856 # Non-metallic minerals (Code: MF3)
857 print('Step 25')
858 alpha_mf3 = coeff[(coeff['Material'] == 'MF3') & (coeff['Coefficient'] == 'Intercept')].drop      ↗
859             (['Coefficient', 'Material'], axis=1).set_index(['Country'])
860 beta_mf3 = coeff[(coeff['Material'] == 'MF3') & (coeff['Coefficient'] == 'P14')].drop(['Coefficient',      ↗
861             'Material'], axis=1).set_index(['Country'])
862 eurostat_mf3 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Non-metallic      ↗
863             minerals'].drop('Level of Disaggregation', axis=1)
864
865 relative_change_mf3 = pd.DataFrame()
866 mf3_p = mf3_p.reindex(beta_mf3.index)
867 for i in mf3_p.columns:
868     relative_change_mf3[i] =alpha_mf3['Value'].add((beta_mf3['Value'].mul(mf3_p[i])))
869 relative_change_mf3 = relative_change_mf3.drop(2021, axis=1)
870
871 final_mf3 = eurostat_mf3.copy()
872 for ye in years_:
873     year = ye
```

```
872
873     year_1 = int(ye)-1
874
875     euro = final_mf3[year_1]
876     rel_ch = 1+relative_change_mf3[year]
877     calc = euro*rel_ch
878     calc = pd.DataFrame(calc)
879     calc.columns = [year]
880     final_mf3[year] = calc[year]
881     final_mf3['Level of Disaggregation'] = 'Non-metallic minerals'
882
883
884 # %%
885 # Fossil energy material/carriers (Code: MF4)
886 print('Step 26')
887 alpha_mf4 = coeff[(coeff['Material'] == 'MF4') & (coeff['Coefficient'] == 'Intercept')].drop      ↗
888   (['Coefficient', 'Material'], axis=1).set_index(['Country'])
889 beta_mf4 = coeff[(coeff['Material'] == 'MF4') & (coeff['Coefficient'] == 'P17')].drop(['Coefficient',      ↗
890   'Material'], axis=1).set_index(['Country'])
891 eurostat_mf4 = material_use_eurostat[material_use_eurostat['Level of Disaggregation'] == 'Fossil energy      ↗
892   materials/carriers'].drop('Level of Disaggregation', axis=1)
893
894 #%%
895 relative_change_mf4 = pd.DataFrame()
896 mf4_p = mf4_p.reindex(beta_mf4.index)
897 for i in mf4_p.columns:
898     relative_change_mf4[i] =alpha_mf4['Value'].add((beta_mf4['Value'].mul(mf4_p[i])))
899 relative_change_mf4 = relative_change_mf4.drop(2021, axis=1)
900
901 final_mf4 = eurostat_mf4.copy()
902 for ye in years_:
903     year = ye
```

```
902     year_1 = int(ye)-1
903
904     euro = final_mf4[year_1]
905     rel_ch = 1+relative_change_mf4[year]
906     calc = euro*rel_ch
907     calc = pd.DataFrame(calc)
908     calc.columns = [year]
909     final_mf4[year] = calc[year]
910     final_mf4['Level of Disaggregation'] = 'Fossil energy material/carrier'
911
912 # %%
913 print('Step 28')
914 material_final = pd.concat([final_mf1, final_mf2, final_mf3, final_mf4]).reset_index()
915 material_final['Indicator'] = 'Material Use'
916 material_final = material_final[cols_final]
917 dataset['Material Use'] = material_final
918
919 # %%
920 # =====
921 # Garther relevant indicators into a dictionary
922 # with reluts from the Baseline and the Scenario
923 # =====
924
925 # Create dataframe for Baseline and Scenario with thelevant indicators
926 print('Step 29')
927 final_dataset[name]=dataset
928
929 # Check if dataframes in final_dataset contains NaNs
930 print("Checking for any nan values in the Dataframes")
931 for i, j in final_dataset.items():
932     check_for_nan = {}
933     for k, v in j.items():
934         _nan = v.isnull().values.any()
```

```

935         print(_nan)
936         check_for_nan[k] = _nan
937         # if value if True, there are NaNs value to investigate
938 # %%
939
940 #Concatenate all indicators in a single dataframe
941 print('Step 30')
942 concatenated = {}
943 for i,v in final_dataset.items():
944     df = pd.DataFrame()
945     for j, l in v.items():
946         df = pd.concat([df,l])
947         df['Level of Disaggregation'] = df['Level of Disaggregation'].replace(np.nan, 'Total')
948         df['Pillar'] = df['Indicator'].map(pillars)
949         df['Unit'] = df['Indicator'].map(metadata_units)
950         df = df[cols_final_]
951
952     concatenated[i]=df.reset_index(drop=True)
953
954 # %%
955 # -----
956 # ----- CALCULATING DIFFERENCES FROM THE BASELINE-----
957 # -----
958
959 # Calculate absolute difference from the Baseline for each indicator
960 print('Step 31')
961 abs_diff = concatenated['Scenario'].iloc[:,5:] - concatenated['Baseline'].iloc[:,5:]
962 abs_diff_ = pd.concat([concatenated['Scenario'].iloc[:,0:5],abs_diff], axis=1)
963
964 # Check for NaNs
965 any_nan = abs_diff_.isnull()
966 print("Checking for any nan values in the absolute differences from the baseline scenario:\
967     True confirms the existance of NaN values in the Dataframe")

```

```

968     print(any_nan) # True confirms the existance of NaN values in the Dataframe
969 # %%
970     # Calculate percentage difference from the Baseline for each indicator
971     print('Step 32')
972     factor1 = abs_diff_.iloc[:, 5:].astype(float)
973     factor2 = concatenated['Baseline'].iloc[:,5:].astype(float)
974
975     pct_diff = factor1.div(factor2)
976     pct_diff_ = pd.concat([concatenated['Scenario'].iloc[:,0:5],pct_diff], axis=1)
977 # %%
978 # -----
979 # ----- CALCULATING DISTRIBUTIONAL IMPACTS -----
980 # -----
981
982 ##### PART A #####
983 # Distributional impacts by MS and by income quintile...
984 # Distributional impacts refer to the share of overall consumption spent on energy
985 # Results are broken down by source of energy, i.e., electricity, gas, liquid fuels and other fuels
986 # ***** Note: missing data for Italy in this dataset *****
987
988     print('Step 33')
989     # Extract results on energy consumption change from the Baseline
990     pct_energy = pct_diff_[pct_diff_.Indicator == 'Consumer Expenditure'].reset_index(drop=True)
991     pct_energy = pct_energy[pct_energy['Country'] != 'UK']
992     pct_energy = pct_energy.drop(['Pillar', 'Indicator', 'Unit'], axis=1)
993     pct_energy = pct_energy.set_index(['Country', 'Level of Disaggregation'])
994
995 #     check = round(e + g + lf + of,3)
996 #     (check == round(shares,3)).all()
997     q1 = pd.DataFrame()
998     q2 = pd.DataFrame()
999     q3 = pd.DataFrame()
1000    q4 = pd.DataFrame()

```



```
1001     q5 = pd.DataFrame()
1002
1003     # Estimate energy reduction effect
1004     for i in pct_energy.columns:
1005         q1[i] = detailed_shares_q1['First quintile'] + (pct_energy[i].mul(detailed_shares_q1['First quintile']))
1006         q2[i] = detailed_shares_q2['Second quintile'] + (pct_energy[i].mul(detailed_shares_q2['Second quintile']))
1007         q3[i] = detailed_shares_q3['Third quintile'] + (pct_energy[i].mul(detailed_shares_q3['Third quintile']))
1008         q4[i] = detailed_shares_q4['Fourth quintile'] + (pct_energy[i].mul(detailed_shares_q4['Fourth quintile']))
1009         q5[i] = detailed_shares_q5['Fifth quintile'] + (pct_energy[i].mul(detailed_shares_q5['Fifth quintile']))
1010
1011     # Extrapolate change in prices from Baseline
1012     p_effect = pct_diff_[pct_diff_.Indicator == 'Price'].reset_index(drop=True)
1013     p_effect = p_effect[(p_effect['Country'] != 'UK') & (p_effect['Country'] != 'World')]
1014     p_effect = p_effect.drop(['Pillar', 'Indicator', 'Unit'], axis=1)
1015     p_effect = p_effect.set_index(['Country', 'Level of Disaggregation']).reindex(detailed_shares_q1.index)
1016
1017     q1_p = pd.DataFrame()
1018     q2_p = pd.DataFrame()
1019     q3_p = pd.DataFrame()
1020     q4_p = pd.DataFrame()
1021     q5_p = pd.DataFrame()
1022
1023     # Incorporate the price effect into the previously esimated distributional impacts
1024     for i in p_effect.columns:
1025         q1_p[i] = q1[i] + p_effect[i].mul(detailed_shares_q1['First quintile'])
1026         q2_p[i] = q2[i] + p_effect[i].mul(detailed_shares_q2['Second quintile'])
1027         q3_p[i] = q3[i] + p_effect[i].mul(detailed_shares_q3['Third quintile'])
1028         q4_p[i] = q4[i] + p_effect[i].mul(detailed_shares_q4['Fourth quintile'])
1029         q5_p[i] = q5[i] + p_effect[i].mul(detailed_shares_q5['Fifth quintile'])
1030
1031     # ***** WARNING *****
1032     # Zero value in consumer expenditure for energy (i.e., Cyprus, Croatia, Malta)
1033     # return zero values for the distributional impacts indicators as well.
```

```
1034 # This is due to use of mock data rather than final E3ME LITE results.
1035
1036 # %%
1037     # Calculate percentage difference from Baseline
1038
1039     print('Step 34')
1040     q1_pct = pd.DataFrame()
1041     q2_pct = pd.DataFrame()
1042     q3_pct = pd.DataFrame()
1043     q4_pct = pd.DataFrame()
1044     q5_pct = pd.DataFrame()
1045
1046     for i in q1.columns:
1047         q1_pct[i] = (q1_p[i].subtract(detailed_shares_q1['First quintile'])).div(detailed_shares_q1['First
1048             quintile'], fill_value=0)
1049         q2_pct[i] = (q2_p[i].subtract(detailed_shares_q2['Second quintile'])).div(detailed_shares_q2['Second
1050             quintile'], fill_value=0)
1051         q3_pct[i] = (q3_p[i].subtract(detailed_shares_q3['Third quintile'])).div(detailed_shares_q3['Third
1052             quintile'], fill_value=0)
1053         q4_pct[i] = (q4_p[i].subtract(detailed_shares_q4['Fourth quintile'])).div(detailed_shares_q4['Fourth
1054             quintile'], fill_value=0)
1055         q5_pct[i] = (q5_p[i].subtract(detailed_shares_q5['Fifth quintile'])).div(detailed_shares_q5['Fifth
1056             quintile'], fill_value=0)
1057
1058     q1_pct = q1_pct.reset_index().rename(columns={'index': 'Country'})
1059     q1_pct['Indicator'] = 'Share of energy consumption_Q1'
1060     q2_pct = q2_pct.reset_index().rename(columns={'index': 'Country'})
1061     q2_pct['Indicator'] = 'Share of energy consumption_Q2'
1062     q3_pct = q3_pct.reset_index().rename(columns={'index': 'Country'})
1063     q3_pct['Indicator'] = 'Share of energy consumption_Q3'
1064     q4_pct = q4_pct.reset_index().rename(columns={'index': 'Country'})
1065     q4_pct['Indicator'] = 'Share of energy consumption_Q4'
1066     q5_pct = q5_pct.reset_index().rename(columns={'index': 'Country'})
```

```
1062     q5_pct['Indicator'] = 'Share of energy consumption_Q5'
1063
1064     #Concatenate quintile data frames in a unique dataframe
1065     energy_shares = pd.concat([q1_pct,q2_pct,q3_pct,q4_pct,q5_pct], axis=0)
1066     energy_shares['Pillar'] = energy_shares['Indicator'].map(pillars)
1067     energy_shares['Unit'] = energy_shares['Indicator'].map(metadata_units)
1068     energy_shares = energy_shares[cols_final_]
1069
1070     # Append quintile data to pct_diff_ dataframe
1071     pct_diff_ = pct_diff_.append(energy_shares)
1072     pct_diff_ = pct_diff_[(pct_diff_['Indicator'] != 'Price') & (pct_diff_['Indicator'] != 'Consumer Expenditure')]
1073     abs_diff_ = abs_diff_[(abs_diff_['Indicator'] != 'Price') & (abs_diff_['Indicator'] != 'Consumer Expenditure')]
1074
1075     # %%
1076     ##### PART B #####
1077     # Distributional impacts by MS and by dwelling archetype
1078     # Process building stock model data (df_bsf)...
1079     # ...and estimate the share of total space heat demand...
1080     # ...of each dwelling archetype for each country
1081
1082     print('Step 35')
1083     # Baseline
1084     df_bsm_list = [g for _,g in df_bsm.groupby('Country code')]
1085     baseline = []
1086     for i in df_bsm_list:
1087         c = i['Country code']
1088         i = i.set_index('Age')
1089         i.loc['Total']= i.sum(numeric_only=True, axis=0)
1090         i = i.fillna(method='ffill')
1091         i = (i.iloc[:,0:30]).T
1092         i = i.dot(conv).T
```

```
1093     hh = i.div(i.loc['Total'])
1094     hh = pd.concat([c,hh], axis=1).fillna(method='ffill').dropna()
1095     hh['Country'] = hh['Country code'].map(country_lookup_bsm['Country'])
1096     hh = hh.drop(['Number of dwellings', 'Country code'], axis=1)
1097     hh = hh.drop('Total')
1098     baseline.append(hh)
1099
1100     bsm_baseline = pd.concat(baseline).reset_index().set_index('Country').sort_values(by=['Country'])
1101     archetype = bsm_baseline['index'].drop('United Kingdom', axis=0)
1102     #bsm_baseline = bsm_baseline.drop('index', axis=1)
1103     bsm_baseline = bsm_baseline.drop('United Kingdom', axis=0)
1104
1105 # %%
1106 # Scenario
1107 # Estimate reduction in heat demand shares by archetypes..
1108 #...based on E3ME results on consumer expenditure in energy.
1109
1110 print('Step 36')
1111 # Extract reduction in consumer expenditure on energy from E3ME LITE results
1112 pct_energy = pct_energy.reset_index().set_index('Level of Disaggregation')
1113 pct_energy_ = [g for _,g in pct_energy.groupby('Country')]
1114 for i in pct_energy_:
1115     i.loc['Total'] = i.sum(numeric_only=True, axis=0)
1116     i['Country'] = i['Country'].fillna(method='ffill')
1117     pct_energy_conct = pd.concat(pct_energy_).reset_index()
1118     pct_energy_conct = pct_energy_conct[pct_energy_conct['Level of Disaggregation'] == 'Total']
1119     pct_energy_conct = pct_energy_conct.set_index('Country').drop('Level of Disaggregation',axis=1)
1120     pct_energy_conct = pct_energy_conct.drop('EU27', axis=0)
1121
1122     bsm_baseline_list = list(set(bsm_baseline.index))
1123     bsm_baseline_list = sorted(bsm_baseline_list)
1124     pct_energy_conct = pct_energy_conct.reindex(bsm_baseline_list)
1125
```

```
1126 # Apply % reduction in consumer expenditure on energy to bsm_baseline
1127 bsm_scenario_change = bsm_baseline.iloc[:,1:].mul(pct_energy_conct.iloc[:,0:29], axis=0).reset_index
    (.set_index('Country')).dropna()
1128 bsm_scenario = bsm_baseline.iloc[:, 1:].add(bsm_scenario_change).reset_index()
1129 archetype = archetype.reset_index()
1130 bsm_scenario = pd.concat([archetype['index'], bsm_scenario], axis=1)
1131 bsm_scenario = bsm_scenario.rename(columns={'index': 'Level of Disaggregation'})
1132 bsm_scenario = bsm_scenario[bsm_scenario['Level of Disaggregation'] != 'Total']
1133 # %%
1134 # Estimate absolute & percentage difference from the Baseline
1135 print('Step 37')
1136 bsm_baseline = bsm_baseline.reset_index().set_index(['Country', 'index'])
1137 bsm_scenario = bsm_scenario.set_index(['Country', 'Level of Disaggregation'])
1138 #%%
1139 print('Step 38')
1140 bsm_abs_diff = bsm_scenario.subtract(bsm_baseline)
1141 bsm_pct_diff = bsm_abs_diff.div(bsm_baseline).reset_index()
1142 bsm_pct_diff['Indicator'] = 'Share of total space heat demand'
1143 bsm_pct_diff['Pillar'] = bsm_pct_diff['Indicator'].map(pillars)
1144 bsm_pct_diff['Unit'] = bsm_pct_diff['Indicator'].map(metadata_units)
1145 bsm_pct_diff = bsm_pct_diff[cols_final_[0:32]]
1146 #%%
1147 print('Step 39')
1148 bsm_abs_diff = bsm_abs_diff.reset_index()
1149 bsm_abs_diff['Indicator'] = 'Share of total space heat demand'
1150 bsm_abs_diff['Pillar'] = bsm_abs_diff['Indicator'].map(pillars)
1151 bsm_abs_diff['Unit'] = bsm_abs_diff['Indicator'].map(metadata_units)
1152 bsm_abs_diff = bsm_abs_diff[cols_final_[0:32]]
1153
1154 # %%
1155 # Append to pct_diff dataframe
1156 print('Step 40')
1157 abs_diff_ = abs_diff_.append(bsm_abs_diff).reset_index(drop=True)
```

```
1158     pct_diff_ = pct_diff_.append(bsm_pct_diff).reset_index(drop=True)
1159     # Warning: this will generate missing values
1160     # as the distributional impacts by dwelling archetype are only
1161     # available between 2022 and 2050
1162
1163     # %%
1164     # -----
1165     # ----- FINAL RESULTS -----
1166     # -----
1167     print('Writing final results to a pickle file')
1168     # Write final results to pickle
1169     abs_diff_ = abs_diff_[abs_diff_['Country'] != 'UK'] & (abs_diff_['Country'] != 'EU27')
1170     abs_diff_.to_pickle("Outputs/Absolute difference from the baseline.pkl")
1171
1172     pct_diff_ = pct_diff_[pct_diff_['Country'] != 'UK'] & (pct_diff_['Country'] != 'EU27')
1173     pct_diff_.to_pickle("Outputs/Percentage difference from the baseline.pkl")
1174
1175     #checking a result: RH
1176     #print(final_dataset['Scenario']['Labour Productivity'].head())
1177
1178     #%%
1179     # Carry out last few checks on pct_diff_
1180
1181     pct_diff_.isnull().values.any()
1182     pct_diff_.iloc[0:1772].isnull().values.any()
1183     indicators_pct = list(set(pct_diff_['Indicator']))
1184     indicators_abs = list(set(abs_diff_['Indicator']))
1185
1186     print('Checking the number of countries for each indicator')
1187     pct_diff_list = [g for _,g in pct_diff_.groupby('Indicator')]
1188     for i in pct_diff_list:
1189         country_list = len(list(set(i.loc[:, 'Country'])))
1190         elem = sorted(list(set(i.loc[:, 'Country'])))
```

```
1191     first_elem = sorted(list(set(pct_diff_list[0].loc[:, 'Country'])))
1192     print(elem == first_elem)
1193     print(country_list)
1194
1195     print('End of the script')
1196
1197     #%%
1198
1199     print('End of the script')
1200
1201     # Record end time of the program
1202     end = time.time()
1203
1204     print("The time of execution of above program is :", (end-start))
```