

```
1 import os
2 import pandas as pd
3 import numpy as np
4 import scipy
5 import math
6 import matplotlib as mpl
7 import matplotlib
8 matplotlib.use('TkAgg')
9 # matplotlib.use('TkAgg')
10 import matplotlib.pyplot as plt
11 import datetime
12 import tables
13 import matplotlib.cm as cm
14 import matplotlib.colors as clr
15 from matplotlib.ticker import FormatStrFormatter
16 from mpl_toolkits.mplot3d import Axes3D
17 from matplotlib.colors import LogNorm
18 from matplotlib import cm
19 from matplotlib.colors import BoundaryNorm
20 from matplotlib.ticker import MaxNLocator
21 from matplotlib.mlab import griddata
22 from matplotlib.backends.backend_tkagg import
    FigureCanvasTkAgg, NavigationToolbar2Tk
23 from matplotlib.figure import Figure
24 import tkinter as tk
25 from tkinter import ttk
26 from tkinter.scrolledtext import ScrolledText
27 from tkinter import *
28 from tkinter.filedialog import askopenfilename
29 import time
30 import sys
31 from inspect import currentframe, getframeinfo
32 from tkinter import filedialog
33 import builtins
34
35 pd.set_option('display.max_columns', 500)
36 pd.set_option('display.width', 2000)
37 import datetime
38 import re
39 import tables
40 import os
41 import shutil
42 import pyarrow as pa
43 import pyarrow.parquet as pq
44 from apscheduler.schedulers.background import
```

```

44 BackgroundScheduler
45 # from simCase83 import Simulation
46 import threading
47 from multiprocessing import dummy as multithreading
48 import queue
49 from threading import Thread
50 from concurrent.futures import Future
51 from concurrent.futures import ThreadPoolExecutor
52 from concurrent.futures import ProcessPoolExecutor
53 import gc
54 # from first_class import StdoutToWidget
55 from mpl_toolkits.axes_grid1 import make_axes_locatable
56 from tkinter import messagebox
57 import random
58 from decimal import Decimal
59 import copy
60 from pandas.plotting import register_matplotlib_converters
61 register_matplotlib_converters()
62 from matplotlib.figure import Figure
63 import random
64
65 LARGE_FONT = ('Verdana', 12)
66 global_x = 0
67 global_x_label = []
68 global_sim_data = {}
69 global_sim_data_listbox = []
70 dict_param = {}
71 dict_paramv2 = {}
72 prep_pageone = {}
73 current_selection = 'None'
74 current_tab = 'None'
75 chosen_rows_alt, chosen_cols_alt = (None, None)
76
77
78 # -----
79 # START OF IMPORT / CONVERT FUNCTION
80 class Simulation(threading.Thread):
81     def __init__(self, save_loc, file_name):
82         threading.Thread.__init__(self)
83         self.home = save_loc
84         self.filepath = file_name
85
86     def convert(self):
87         filepath = self.filepath
88         start_time = datetime.datetime.now()

```

```

89         x_current_addition = """
90
91             dx, dy, dz, lx, ly, lz, nx, ny, nz, n_name,
92             n_title, n_temp, n_sim, n_ver, n_file, n_file_type,
93             pre_line, n_steps, keys, \
94                 check_grid01, pre_pre_line, df_grid_data,
95                 blacklist, x_check, y_check, z_check, k_check, ln,
96                 cells_grid, \
97                     cells_col, pre_pre_pre_line = ([] for ti in range
98 (31))
99
100
101             add_folder, name_of_file, core_path, new_path,
102             select_folder, print_now, keys2, comp_temp, reg_temp,
103             well_temp, \
104                 n_pressure, ndaysnow, ndates, complist,
105                 well_list_w, path_comp, path_reg, path_well, comp_list,
106                 temp_list, templistr, \
107                     temp_list_w, print_now_r, reg_list, n_reg,
108                     current, wellreads, read_now, current_f, nwells,
109                     n_well_name, ntypes, n_pv_well, \
110                         n_well_temp, well_temp_items, final_line,
111                         x_current, sum_qst, path_data, cell_index = ([] for ti in
112 range(40))
113
114             normal_length = 0
115
116             n_summary, getcomps, store_once_one,
117             store_data_once, firstvalues, start_main, x, check_title,
118             check_grid02, check_grid03, \
119                 key_search, indexing_go, = [0] * 12
120
121             start = "01-Jan-2010 00:00:00"
122             i_day = datetime.datetime.strptime(start, "%d-%b
123 -%Y %H:%M:%S")
124
125             temp_dict, cells_dict, grid_dim_dict,
126             temp_storage, dict_comp, keys2_dict, summary_info,
127             store_once, col_width, \
128                 current_dict, store_col, col_width_data,
129                 cells_data = ({}) for tj in range(13))
130
131             well_param = 0
132
133             for temp_items in ['DX', 'DY', 'DZ', 'XKEYS']:

```

```

115         temp_storage[temp_items] = []
116
117     def get_cells(*args):
118         length = len(args)
119         if length != 5:
120             return None
121         celli = args[0]
122         cellj = args[1]
123         cellk = args[2]
124
125         grid_dict_data = args[3]
126         option = args[4]
127
128         imax = int(grid_dict_data['NX'])
129         jmax = int(grid_dict_data['NY'])
130         kmax = int(grid_dict_data['NZ'])
131         cells_col_names = ['Cell', 'i', 'j', 'k', 'DX',
132                           'DY', 'DZ', 'X', 'Y', 'Z']
133         cell_list_dict = {}
134         for nok in cells_col_names:
135             cell_list_dict[nok] = []
136
137         if option == "single":
138             ncell = (cellj - 1) * imax + (cellk - 1)
139             * jmax * kmax + celli
140             return ncell
141         elif option == "full":
142             for nk in range(1, kmax + 1):
143                 for nj in range(1, jmax + 1):
144                     for ni in range(1, imax + 1):
145                         ncell = (nj - 1) * imax + (nk
146                         - 1) * jmax * imax + ni
147                         cell_list_dict['Cell'].append(
148                             int(ncell))
149                         cell_list_dict['i'].append(
150                             int(ni))
151                         cell_list_dict['j'].append(
152                             int(nj)))
153                         cell_list_dict['k'].append(
154                             int(nk)))
155                         cell_list_dict['NX'] = imax
156                         cell_list_dict['NY'] = jmax
157                         cell_list_dict['NZ'] = kmax
158             return cell_list_dict
159

```

```

153     def get_cell_dim(*args):
154         length = len(args)
155         if length != 2:
156             return None
157
158         first_value = 0
159         all_cells = args[0]
160         grid_info = args[1]
161
162         dim_dx = grid_info['DX']
163         dim_dy = grid_info['DY']
164         dim_dz = grid_info['DZ']
165
166         length_x, length_y, length_z = [0] * 3
167         dxi_prev, dyj_prev, dzk_prev = [0] * 3
168         ivalue_prev, jvalue_prev, kvalue_prev = [0] *
169             3
170
171         ivalue = list(np.unique(all_cells['i']))
172         jvalue = list(np.unique(all_cells['j']))
173         kvalue = list(np.unique(all_cells['k']))
174
175         length_z, dzk_prev = [0]*2
176         for kl in kvalue:
177             dzk = float(dim_dz[kl-1])
178             length_z = length_z + float(0.5) *
179                 (dzk_prev + dzk)
180             length_y, dyj_prev = [0] * 2
181             for jl in jvalue:
182                 dyj = float(dim_dy[jl-1])
183                 length_y = length_y + float(0.5) *
184                     (dyj_prev + dyj)
185             length_x, dxi_prev = [0]*2
186             for il in ivalue:
187                 dxi = float(dim_dx[il-1])
188                 length_x = length_x + float(0.5) *
189                     (dxi_prev + dxi))
190
191                 all_cells['DX'].append(dxi)
192                 all_cells['DY'].append(dyj)
193                 all_cells['DZ'].append(dzk)
194                 all_cells['X'].append(length_x)
195                 all_cells['Y'].append(length_y)
196                 all_cells['Z'].append(length_z)
197                 dxi_prev = dxi
198                 dyj_prev = dyj

```

```
194             dzk_prev = dzk
195
196         return all_cells
197
198     def get_keys(*args):
199         """You must specify: | Full XKEYS list | List
200         of XKEYS you want removed |"""
201
202         length = len(args)
203         if length != 2:
204             return None
205
206         keys_original = args[0]
207         keys_remove = args[1]
208         keys_remaining = []
209
210         for m in keys_original:
211             if any(m in s for s in keys_remove):
212                 pass
213             else:
214                 keys_remaining.append(m)
215
216
217     def create_directory(*args):
218         """You must specify: | Core Path | New Path
219         (+1 increment) |"""
220
221         length = len(args)
222         if length != 2:
223             return None
224
225         core = args[0]
226         new = args[1]
227
228         if os.path.exists(core):
229             if os.path.exists(new):
230                 pass
231             else:
232                 os.mkdir(new)
233
234             os.mkdir(core)
235
236     def write_to_file(*args):
```

```

237             column_width = {}
238             no_col_restraint = 0
239             length = len(args)
240             if length == 6:
241                 no_col_restraint = 1
242             elif length == 7:
243                 no_col_restraint = 0
244                 column_width = args[6]
245             else:
246                 return None
247
248             store_method = args[0]
249             dataframe = args[1]
250             path_original = args[2]
251             folder = args[3]
252             name = args[4]
253             id_unique = args[5]
254
255             datatype = ''
256             if store_method == 'parquet':
257                 path = os.path.join(path_original, folder
258 , name + '.parquet')
259                 table = pa.Table.from_pandas(dataframe)
260                 write_id = id_unique
261                 if write_id is None:
262                     write_id = pq.ParquetWriter(path,
263 table.schema)
264                     write_id.write_table(table=table)
265                     id_unique = write_id
266                     return id_unique
267
268             elif store_method == 'hdf5':
269                 path = os.path.join(path_original, folder
270 , name + '.h5')
271                 if no_col_restraint == 1:
272                     dataframe.to_hdf(path, key=name,
273 format='table', append=True)
274                 elif no_col_restraint == 0:
275                     dataframe.to_hdf(path, key=name,
276 format='table', append=True, data_columns=True, complevel
277 =9, complib='blosc',
278                                         min_itemsize=
279 column_width)
280             return None
281
282

```

```

275     store_method = 'parquet'
276     with open(filepath) as fw:
277         for final_line, line in enumerate(fw, 1):
278             pass
279
280     data_input = []
281     data_main = []
282     close01 = []
283     close02 = []
284     close03 = []
285     close04 = []
286     close05 = []
287     close06 = []
288     close07 = []
289     close08 = []
290     close09 = []
291     close10 = []
292     n_timestep = []
293     write_id_data = None
294     write_id_time = None
295     write_id_input = None
296     write_id_comp = None
297     write_id_param = None
298     write_id_reg = None
299     write_id_wells = None
300     write_id_wellparam = None
301
302     store_parameters = 0
303     n_count = 0
304     skip_line = 10
305     with open(filepath) as fp0:
306         for lineNumber, line in enumerate(fp0, 1):
307             x = round((lineNumber / final_line) * 100
308             , 2)
309             # print('line: ' + str(lineNumber
310             ) + ' ' + str(line))
311             if skip_line == 0:
312                 global global_x_label
313                 global_x_label.config(text=str(x) +
314                 '%')
315                 n_count2 = 5
316             else:
317                 skip_line -= 1
318             if not data_input:
319                 if not close01 and 'Input file' in

```

```

316 pre_line:
317                                         current = pre_pre_pre_line.split(
318                                         )
319                                         n_sim, n_ver = (current[0], float(
320                                         current[2]))
321                                         current = pre_line.split()[-1].
322                                         split('.')
323                                         n_file, n_file_type = (current[0]
324                                         , '.' + current[1])
325                                         add_folder = n_file
326                                         core_path = self.home
327                                         new_path = os.path.join(core_path
328                                         , add_folder)
329                                         create_directory(core_path,
330                                         new_path)
331                                         select_folder, close01 = (
332                                         add_folder, 1)
333                                         elif close01 and 'Run description' in
334                                         pre_line and not close02:
335                                         n_title, close02 = (line[:-1], 1)
336                                         elif close02 and 'Grid dimensions' in
337                                         pre_pre_line and not close03:
338                                         current = pre_line.split() + line
339                                         .split()
340                                         for i in current:
341                                         if i in ['NX', 'NY', 'NZ', '
342                                         LX', 'LY', 'LZ']:
343                                         grid_dim_dict[i] = float(
344                                         current[current.index(i) + 2])
345                                         n_count += 1
346                                         if n_count == len(
347                                         grid_dim_dict):
348                                         close03 = 1
349                                         elif close03 and ('DX' in pre_line or
350                                         'DY' not in line) and not close04:
351                                         if 'DY' in line:
352                                         close04 = 1
353                                         else:
354                                         current = line.split()
355                                         temp_storage['DX'] =
356                                         temp_storage['DX'] + current
357                                         elif close04 and ('DY' in pre_line or
358                                         'DZ' not in line) and not close05:
359                                         if 'DZ' in line:
360                                         close05 = 1

```

```

345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
            else:
                current = line.split()
                temp_storage['DY'] =
                    temp_storage['DY'] + current
            elif close05 and ('DZ' in pre_line or
'DZ' not in line) and not close06:
                if 'XKEYS' in line:
                    close06 = 1
                else:
                    current = line.split()
                    temp_storage['DZ'] =
                        temp_storage['DZ'] + current
            elif close06 and ("XKEYS" in pre_line
or "XKEYS" not in line) and not close07:
                current = line.split()
                for i in current:
                    if i == 'ZZZZE':
                        close07 = 1
                    else:
                        i = i[0:len(i) - 1]
                        temp_storage['XKEYS'].
append(i)
            elif close07 and 'TAXIS' in pre_line
and not close08:
                cells_dict = get_cells(1, 1, 1,
grid_dim_dict, 'full')
                cells_pos, close08 = (cells_dict[
'Cell'], 1)
            elif close08 and not close09:
                for x in temp_storage['XKEYS']:
                    if x in line:
                        x_current_addition = str(
line[:-1])
                        blacklist.append(x)
                        cells_dict[
x_current_addition], close10 = ([]), 1
                        break
                if 'Summary' in line and '
Timestep' in line:
                    close09, data_input = (1, 1)
                    keys = get_keys(temp_storage[
'XKEYS'], blacklist)
                    cells_data = get_cell_dim(
cells_dict, temp_storage)
                    df_grid_data = pd.DataFrame(

```

```

376 cells_data)
377                                     name_of_file, select_folder =
378                                         ('INPUT', add_folder)
379
380                                         grid_col = df_grid_data.
381                                         columns
382                                         grid_col_new = []
383                                         for p in range(len(grid_col)):
384                                         :
385                                         y = grid_col[p]
386                                         y = y.translate({ord(i):
387                                         None for i in ['-', '_', '[', ']', '#', '.']}))
388                                         grid_col_new.append(y)
389
390                                         df_grid_data.columns =
391                                         grid_col_new
392                                         write_id_parquet =
393                                         write_to_file(store_method, df_grid_data, core_path,
394                                         select_folder, name_of_file, write_id_input, None)
395                                         if write_id_parquet is not
396                                         None:
397                                         write_id_input =
398                                         write_id_parquet
399                                         elif (x_current_addition in
400                                         pre_pre_line or x_current_addition not in line) and
401                                         x_current_addition not in pre_line:
402                                         current = line.split()
403                                         cells_dict[x_current_addition]
404                                         ] = cells_dict[x_current_addition] + current
405                                         if data_input and not data_main:
406                                         if "Summary" in pre_pre_line and "
407                                         Timestep" in pre_pre_line:
408                                         name_of_file, n_summary = ('TIME'
409                                         , n_summary + 1)
410                                         current = pre_pre_line.replace(
411                                         ',', ' ').split()
412                                         ndaysnow = float(current[current.
413                                         index('Time') + 1])
414                                         ndates = i_day + datetime.
415                                         timedelta(days=float(ndaysnow))
416                                         n_timestep = int(current[current.
417                                         index('Timestep') + 1])
418                                         n_pv = float(current[current.
419                                         index('PV') + 2])
420                                         n_pv_tot = float(pre_line.split())

```

```

401 [2])
402 n_cpu = float(line.split()[2])
403
404 add_this = { 'nStep': [n_summary],
405 'nDays': [ndaysnow], 'nDate': [pd.to_datetime(ndates)],
406 'n_timestep': [
407 n_timestep], 'nPv': [n_pv], 'nPvtot': [n_pv_tot], 'nCPU':
408 [n_cpu]}
409
410 df_time = pd.DataFrame(add_this)
411 df_time = df_time.set_index(
412 df_time.columns[2])
413
414 write_id_parquet = write_to_file(
415 store_method, df_time, core_path, select_folder,
416 name_of_file, write_id_time, None)
417 if write_id_parquet is not None:
418     write_id_time =
419     write_id_parquet
420 elif "--" in pre_line and "Component
421 volume balance" in pre_pre_pre_line:
422     current = pre_pre_pre_line.split(
423 )
424     n_pressure = float(current[-2])
425     current = pre_pre_line.split()
426     pv_loc = current.index('PV=') + 3
427     name_of_file, print_now = ('COMP'
428 , 0)
429     complist = ['Time'] + ['Pressure'
430 ] + ['Component'] + current[pv_loc:-2] + [current[-2] +
431 ' ' + current[-1]]
432
433 while print_now == 0:
434     temp_list = []
435     current = line.split()
436     if "=" in line:
437         print_now = 1
438         getcomps = 1
439         comp_temp = pd.DataFrame(
440         comp_temp, columns=complist)
441         comp_temp = comp_temp.
442         set_index(comp_temp.columns[0])
443
444         write_id_parquet =
445         write_to_file(store_method, comp_temp, core_path,
446         select_folder, name_of_file, write_id_comp, None)

```

```

430                     if write_id_parquet is not
431                         None:
432                             write_id_parquet =
433                                 write_id_comp =
434                                     comp_temp = []
435                         else:
436                             try:
437                                 string01 = current[2]
438                                     float(string01)
439                                     string01 = current[1]
440                                         for r in current[2:]:
441                                             r = float(r)
442                                             temp_list.append(r)
443                                                 current = [pd.to_datetime
444 (ndates)] + [n_pressure] + [string01] + temp_list
445                                         except ValueError:
446                                             string01 = current[1] +
447                                                 current[2]
448                                         for r in current[3:]:
449                                             r = float(r)
450                                             temp_list.append(r)
451                                                 current = [pd.to_datetime
452 (ndates)] + [n_pressure] + [string01] + temp_list
453                                         comp_temp.append(current)
454                                         if getcomps == 0:
455                                             comp_list.append(string01
456                                         )
457                                         elif getcomps == 1:
458                                             pass
459                                         break
460                                         if "--" in pre_line and "Region" in
461                                         pre_pre_line:
462                                             current = pre_pre_line.split()
463                                             n_reg = float(current[1])
464                                             pv_loc = current.index('PV=') + 3
465                                             name_of_file, print_now_r = ('
466                                             REGION', 0)
467                                             reg_list = ['Time'] + ['Region']
468                                             + ['Component'] + current[pv_loc:-2] + [current[-2] + '_'
469                                             + current[-1]]
470                                             while print_now_r == 0:
471                                                 templistr = []
472                                                 if "Region" in line or "=" in
473                                                 line or line == '\n':
474                                                     print_now_r = 1

```

```

464                                     reg_temp = pd.DataFrame(
465             reg_temp, columns=reg_list)
466                                     reg_temp = reg_temp.set_index
467             (reg_temp.columns[0])
468                                     write_id_parquet =
469             write_to_file(store_method, reg_temp, core_path,
470             select_folder, name_of_file, write_id_reg, None)
471             if write_id_parquet is not
472             None:
473                 write_id_parquet
474                 write_id_reg =
475                 reg_temp = []
476             else:
477                 current = line.split()
478             try:
479                 string01 = current[2]
480                 float(string01)
481                 string01 = current[1]
482                 for z in current[2:]:
483                     z = float(z)
484                     templistr.append(z)
485                 current = [pd.to_datetime
486                 (ndates)] + [n_reg] + [string01] + templistr
487                 except ValueError:
488                     string01 = current[1] +
489                     current[2]
490                     for z in current[3:]:
491                         z = float(z)
492                         templistr.append(z)
493                     current = [pd.to_datetime
494                     (ndates)] + [n_reg] + [string01] + templistr
495                     reg_temp.append(current)
496                     break
497                     if "Well" in pre_line and "report" in
498                     pre_line:
499                         current = pre_line.split()
500                         nwells = int(current[1])
501                         pv_loc = current.index('PV') + 2
502                         n_pv_well = float(current[pv_loc]
503                         )
504                         current = line.split()
505                         ntypes = current[0].replace(',', '',
506                         '')
507                         n_well_name = current[1].replace(

```

```

496  ' . , ')
497
498             try:
499                 T_loc = current.index('
500                     temperature')
501
502                     try:
503                         n_well_temp = float(
504                             current[T_loc + 2])
505
506                     except ValueError:
507                         n_well_temp = float(
508                             current[T_loc + 1].replace('=', ''))
509
510                     except ValueError:
511                         n_well_temp = 999 #  

512                         Indicates well has been closed
513                         wellreads = 0
514
515             elif "Connection" in pre_pre_line and
516                 wellreads == 0:
517
518                 current = pre_pre_line.
519                     replace('block', '')
520
521                     current = current.split()
522                     current_n = pre_pre_line.split()[  

523                         1:]
524
525                     current_f = []
526                     well_param_names = []
527                     for u in range(len(current)):
528                         if well_param == 0:
529                             storethis = str(current[u]
530                                 ] + ' ' + str(current_n[u]))
531
532                             well_param_names.append(
533                                 storethis)
534
535                             current_f0 = current[u].
536                                 replace('/', 'per') + current_n[u].replace('/', 'per')
537
538                             current_f0 = current_f0.
539                                 translate({ord(i): None for i in ['(', ')', '^']})
540
541                             current_f.append(current_f0)
542
543                             if well_param == 0:
544                                 well_param_names = ['nPVwell'
545                                     , 'Temperature', 'Connection'] + well_param_names
546
547                                 well_col = pd.DataFrame(pd.
548                                     Series(well_param_names))
549
550                                 well_col.set_index(well_col.
551                                     columns[0])
552
553                                 name_of_file = 'WELLPARAM'
554
555                                 write_id_parquet =
556                                 write_to_file(store_method, well_col, core_path,

```

```

525 select_folder, name_of_file, write_id_wellparam)
526                                     if write_id_parquet is not
None:
527                                         write_id_wellparam =
528                                         well_param = 1
529                                         read_now = 0
530                                         while wellreads == 0 and read_now ==
531                                             0: # and line != '\n':
532                                             current = []
533                                             temp_list_w = []
534                                             if "----" in line:
535                                                 current = pre_line.split()
536                                                 for z in current[1:]:
537                                                     # print('current: ' + str
538                                                     (z))
539                                                     if normal_length == 0:
540                                                         normal_length = len(
541                                                             well_param_names) - 2
542                                                         current_length = len(
543                                                             current)
544                                                         if normal_length !=
545                                                             current_length:
546                                                             if '-' in z and not 'e-'
547                                                             in z:
548                                                               pass
549                                                               # print('yup,
550                                                               there is extra -, and no e-') # could be just negative
551                                                               number
552                                                               elif '-' in z and 'e
553                                                               -' in z:
554                                                               split_it = z.
555                                                               split('e-')
556                                                               base_number =
557                                                               split_it[0]
558                                                               if '-' in
559                                                               base_number:
560                                                               new_split =
561                                                               base_number.split('-')
562                                                               if len(
563                                                               new_split) > 1:
564                                                               first_number = float(new_split[0])
565                                                               temp_list_w.append(first_number)

```

```

552     second_number = float('-' + new_split[1] + 'e-' +
553     split_it[1])
554
555     temp_list_w.append(second_number)
556
557     else:
558         z = float(z)
559         temp_list_w.append(z)
560         string01 = current[0]
561         current = [pd.to_datetime(
562             ndates)] + [nwell] + [n_well_name] + [ntypes] + [
563             n_pv_well] + [n_well_temp] + [string01] + temp_list_w
564
565         elif 'Total' in pre_pre_line:
566             name_of_file = 'WELLS'
567             if store_once_one == 0:
568                 well_list_w = ['Time'] +
569                 ['nWell'] + ['nWellName'] + ['nType'] + ['nPVwell'] + [
570                 'nWellTemp'] + ['Connection'] + current_f
571
572             for e in well_list_w[1:]:
573                 col_width[str(e)] =
574
575                 50
576
577             store_once_one = 1
578             well_temp = pd.DataFrame(
579                 well_temp, columns=well_list_w)
580
581             well_temp = well_temp.
582             set_index(well_temp.columns[0])
583
584             write_id_parquet =
585             write_to_file(store_method, well_temp, core_path,
586             select_folder, name_of_file, write_id_wells, col_width)
587
588             if write_id_parquet is not
589             None:
590
591                 write_id_wells =
592
593                 write_id_parquet
594
595                 well_temp = []
596
597                 wellreads, read_now, sum_qst
598                 = (1, 1, 1)
599
600                 break
601
602                 if current:
603                     well_temp.append(current)
604
605                     break
606
607                 if wellreads == 1 and read_now == 1:
608
609                     if ("Summary" in line and "
610                         "Timestep" in line) or 'CPU summary report' in line:
611
612                         name_of_file, cell_number =
613
614                         (
```

```

580 'DATA', len(cells_dict['Cell']))
581                                     current_dict['Time'] = [pd.
582             to_datetime(ndates) * cell_number
583                                     current_dict['Days'] = [float
584             (ndaysnow) * cell_number
585                                     current_dict['Timestep'] = [
586             int(n_timestep) * cell_number
587                                     current_dict['Cell'] =
588             cells_data['Cell']
589                                     testing = pd.DataFrame(
590             current_dict)
591                                     cols = testing.columns.tolist()
592             ())
593                                     cols = ['Time', 'Days', 'Timestep', 'Cell'] + cols[:-4]
594                                     testing = testing[cols]
595                                     testing = testing.set_index(
596             testing.columns[testing.columns.tolist().index('Time')])
597                                     if store_data_once == 0:
598             df_col = testing.columns.
599             tolist()
600                                     for f in df_col:
601             col_width_data[str(f)]
602             ] = 50
603                                     store_data_once = 1
604                                     write_id_parquet =
605             write_to_file(store_method, testing, core_path,
606             select_folder, name_of_file, write_id_data,
607             col_width_data)
608                                     if write_id_parquet is not
609             None:
610             write_id_data =
611             write_id_parquet
612             wellreads, start_main,
613             current_dict = (0, 0, {})
614             else:
615             for xi in keys:
616             if xi in pre_pre_line and
617             start_main == 0:
618             start_main = 1
619             remove = pre_pre_line
620             .rstrip()
621

```

```

607                                     if len(store_col) !=
608             len(keys):
609                 remove.translate({ord(i): None for i in ['_', '[', ']',
610                   ' ', '(', ')', '-', '/', '%', '^']})
611             store_col[remove] = x_current
612             else:
613                 if
614                     store_parameters == 0:
615                         test_col = pd
616                             .DataFrame(pd.Series(store_col))
617                             set_index(test_col.columns[0])
618                             name_of_file
619                             = 'PARAMETERS'
620
621                         write_id_parquet = write_to_file(store_method, test_col,
622                           core_path, select_folder, name_of_file, write_id_param)
623                         if
624                             write_id_parquet is not None:
625                                 write_id_param = write_id_parquet
626
627                         store_parameters = 1
628                         x_current =
629                         store_col[remove]
630                         elif xi in line and
631                           start_main == 1:
632                               start_main = 0
633                               break
634
635                               while start_main == 1:
636                                   current = line.split()
637                                   if x_current not in
638                                       current_dict:
639                                           current_dict[
640                                             x_current] = list(map(float, current))
641                                           elif current:
642                                               current_dict[
643                                                 x_current] = current_dict[x_current] + list(map(float,
644                                                   current))
645                                               break
646                               pre_pre_pre_line = pre_pre_line

```

```

633                 pre_pre_line = pre_line
634                 pre_line = line
635
636         if write_id_data:
637             write_id_data.close()
638         if write_id_time:
639             write_id_time.close()
640         if write_id_input:
641             write_id_input.close()
642         if write_id_comp:
643             write_id_comp.close()
644         if write_id_param:
645             write_id_param.close()
646         if write_id_reg:
647             write_id_reg.close()
648         if write_id_wells:
649             write_id_wells.close()
650         if write_id_wellparam:
651             write_id_wellparam.close()
652         time_elapsed = datetime.datetime.now() -
653             start_time
654             # print('Time elapsed (hh:mm:ss.ms) {}' .format(
655             time_elapsed))
656
657 # END OF IMPORT / CONVERT FUNCTION
658 #
659 tp = ThreadPoolExecutor(1)
660
661
662 def threaded(fn):
663     def wrapper(*args, **kwargs):
664         return tp.submit(fn, *args, **kwargs)
665
666     return wrapper
667
668
669 class PopupWindow(object):
670     def __init__(self, master):
671         top = self.top = Toplevel(master)
672         self.l = Label(top, text='Rows: ', relief=SUNKEN)
673         self.l.pack(side=LEFT, padx=1, pady=3, ipadx=1,
674                     ipady=1)
675         self.e = Entry(top, width=4, relief=SUNKEN)

```

```

675         self.e.pack(side=LEFT, padx=3, pady=3, ipady=1)
676         self.l2 = Label(top, text='Columns: ', relief=SUNKEN)
677         self.l2.pack(side=LEFT, padx=1, pady=3, ipadx=1, ipady=1)
678         self.e2 = Entry(top, width=4, relief=SUNKEN)
679         self.e2.pack(side=LEFT, padx=3, pady=3, ipady=1)
680
681         self.b = ttk.Button(top, text='ok', command=self.cleanup)
682         self.b.pack(side=LEFT, padx=3, pady=3)
683         self.value = None
684         self.value2 = None
685
686     def cleanup(self):
687         self.value = self.e.get()
688         self.value2 = self.e2.get()
689         global chosen_rows_alt, chosen_cols_alt
690         chosen_rows_alt = self.value
691         chosen_cols_alt = self.value2
692         self.top.destroy()
693
694
695 root = tk.Tk
696
697
698 class SimPlotJIN(root):
699     def __init__(self, *args, **kwargs): # When you call
       the class, this will always run. Restart pc -> want
       something ie. explorer.exe, keyboard to load, etc..
700         tk.Tk.__init__(self, *args, **kwargs) # tkinter
       is now also initialized
701         tk.Tk.iconbitmap(self, default='gui_icon.ico')
702         tk.Tk.wm_title(self, 'SimPlotJIN')
703         tk.Tk.geometry(self, "1300x1000")
704         status = Label(self, text='..RAM usage', anchor='w',
       relief=SUNKEN)
705         status.pack(side=BOTTOM, fill='both')
706
707         self.nb = ttk.Notebook(self)
708         self.nb.pack(expand=1, fill='both')
709         self.frames = {}
710         labels = ['Start', 'Page One', 'Page Two', 'Page
       Three']
711         classes = [StartPage, PageOne, PageTwo, PageThree]

```

```

711 ]
712     for i in range(len(classes)):
713         page = classes[i]
714         frame = page(parent=self.nb, controller=self)
715         # Calls the class
716         self.frames[page] = frame
717         self.nb.add(frame, text=labels[i])
718
719     def prep_local_param(event):
720         selection = event.widget.select()
721         tab = event.widget.tab(selection, 'text')
722         global current_tab
723         current_tab = tab
724         current = global_sim_data
725         if tab == 'Page One' and current:
726             alls = list(global_sim_data_listbox.get(0
727             , END))
728             for num in reversed(range(len(alls))):
729                 keys = alls[num]
730                 if 'DATA' not in current[keys] or 'PARAMETERS' not in current[keys]:
731                     global_sim_data_listbox.delete(
732                         num)
733                 else:
734                     store_path = current[keys][0]
735                     path_param = os.path.join(
736                         store_path, 'PARAMETERS' + '.parquet')
737                     avail_param = pd.read_parquet(
738                         path_param)
739                     if 'WELLPARAM' in current[keys]:
740                         path_wellparam = os.path.join(
741                             store_path, 'WELLPARAM' + '.parquet')
742                         avail_wellparam = pd.
743                             read_parquet(path_wellparam)
744                         store_wellparam =
745                         avail_wellparam.iloc[:,0].tolist()
746                         path_wells = os.path.join(
747                             store_path, 'WELLS' + '.parquet')
748                         wells_col = pd.read_parquet(
749                             path_wells).columns.tolist()
750                         temp_dict = {}
751                         shown = store_wellparam
752                         hidden = wells_col[3:]
753                         for ik in list(range(len(
754                             shown))):
```

```

744                               temp_dict[shown[ik]] =
    hidden[ik]
745                               global dict_paramv2
746                               dict_paramv2[keys] =
    temp_dict
747                               store_param = avail_param.index.
    tolist()
748                               full_list = [store_path] +
    store_param
749                               global dict_param
750                               dict_param[keys] = full_list
751
752                               local_dict = {}
753                               for param_user in store_param:
754                                   param_backend = avail_param.
    loc[param_user, 0]
755                               local_dict[param_user] =
    param_backend
756                               global prep_pageone
757                               prep_pageone[keys] = local_dict
758
759                               self.nb.bind('<<NotebookTabChanged>>',
    prep_local_param)
760
761             def on_closing(self):
762                 if messagebox.askokcancel('Quit', 'Do you want to
    quit?'):
763                     SimPlotJIN().quit()
764
765
766 class StartPage(tk.Frame): # Creates a frame that we
    call the start page. then we can make more pages, and
    show them with show_frame method
767     def __init__(self, parent, controller):
768         self.controller = controller
769         self.parent = parent
770         tk.Frame.__init__(self, parent)
771         self.filename = '...'
772         self.list1 = []
773         self.collect_thread = []
774         self.text_here = ''
775         self.count = 0
776         self.home_location = os.path.join(os.path.
    expanduser('~'), 'Documents', 'ProjIORCoreSim')
777         self.read_location = self.home_location

```

```

778         self.simulation_data_found = {}
779         self.simulation_data_to_plot = {}
780         self.simulation_data_sorted = {}
781
782         data_full = {}
783
784         bigframe = Frame(self, bg='#CD3333')
785         bigframe.pack(expand=True, fill='both', padx=1,
786                       pady=1)
786
787         f1 = Frame(bigframe, bg='orange')
788         f2 = Frame(bigframe, bg='yellow', bd=3)
789         f2a = Frame(f2, bg='grey', bd=3)
790         f2b = Frame(f2, bg='black', bd=3)
791         f2c = Frame(f2, bg='blue', bd=3)
792         f3 = Frame(bigframe, bg='green')
793
794         f1.pack(side=TOP, expand=0, fill='both', padx=3,
795                     pady=3)
795         f2.pack(side=TOP, expand=0, fill='both', padx=3,
796                     pady=3)
796         f3.pack(side=TOP, expand=1, fill='both', padx=3,
797                     pady=3)
797
798         f2a.grid(column=0, row=0)
799         f2b.grid(column=1, row=0)
800         f2c.grid(column=2, row=0)
801
802         button_import = ttk.Button(f1, text='Import..',
803                                     command=lambda: self.load_file(f1))
803         button_import.grid(column=0, row=0, sticky='nw',
804                             padx=3, pady=3)
804         button_save = ttk.Button(f1, text='Save to..',
805                                     command=lambda: self.save_file(f1))
805         button_save.grid(column=0, row=1, sticky='nw',
806                             padx=3, pady=3)
806         self.button_convert = ttk.Button(f1, text='Convert',
807                                         command=lambda: self.convert_file(f1))
807         self.button_convert.grid(column=2, row=0)
808         button_read = ttk.Button(f1, text='Read from..',
809                                     command=lambda: self.read_folder(f1, f2a.list_parent))
809         button_read.grid(column=0, row=2, sticky='nw',
810                             padx=3, pady=3)
810         button_add = ttk.Button(f2b, text='Add', command=
811                               lambda: self.add_name(parent=f2a.list_parent, child=f2c.

```

```

810     list_child))
811         button_add.pack()
812         button_del = ttk.Button(f2b, text='Remove',
813         command=lambda: self.remove_name(child=f2c.list_child))
814         button_del.pack()
815
816         f1.label = Label(f1, text=self.filename, width=1,
817         relief=SUNKEN)
816         f1.label.grid(column=1, row=0, padx=3, pady=3,
817         ipadx=250, ipady=2)
817         f1.label4 = Label(f1, text=self.home_location,
817         width=1, relief=SUNKEN)
818         f1.label4.grid(column=1, row=1, padx=3, pady=3,
818         ipadx=250, ipady=2)
819         f1.label5 = Label(f1, text=self.read_location,
819         width=1, relief=SUNKEN)
820         f1.label5.grid(column=1, row=2, padx=3, pady=3,
820         ipadx=250, ipady=2)
821         global global_x_label
822         global_x_label = Label(f1, text='0.00 %', width=5
822 )
823         global_x_label.grid(column=3, row=0, padx=3, pady
823 =3, ipadx=15, ipady=2)
824
825         f2a.label6 = Label(f2a, text=' Available
825 Simulation Cases ')
826         f2a.label6.grid(column=0, row=0, padx=3, pady=3)
827         f2a.list_parent = Listbox(f2a, height=10,
827         selectmode=EXTENDED, relief=SUNKEN)
828         f2a.list_parent.grid(column=0, row=1, padx=3,
828         pady=3)
829         f2c.label7 = Label(f2c, text=' Cases available
829 for plotting ')
830         f2c.label7.grid(column=0, row=0, padx=3, pady=3)
831         f2c.list_child = Listbox(f2c, height=10,
831         selectmode=EXTENDED, relief=SUNKEN)
832         f2c.list_child.grid(column=0, row=1, padx=3, pady
832 =3)
833
834         self.local_simulations(path_to_check=self.
834         read_location, f2a_listbox=f2a.list_parent)
835
836     def local_simulations(self, path_to_check,
836     f2a_listbox):
837         store_list = []

```

```

838         for path, dirs, files in os.walk(path_to_check):
839             store_list = []
840             for i in files:
841                 current = i.split('.')
842                 if current[0] not in ['COMP', 'DATA', 'INPUT',
843 'PARAMETERS', 'REGION', 'TIME', 'WELLS', 'WELLPARAM']:
844                     break
845                 elif current[1] == 'parquet':
846                     store_list.append(current[0])
847                     if store_list:
848                         sim_folder = os.path.basename(path)
849                         combined = [path] + store_list
850                         f2a_listbox.insert(END, sim_folder)
851                         self.simulation_data_found[sim_folder] =
852                             combined
853
854             def add_name(self, parent, child):
855                 cursors = parent.curselection()
856                 alls = list(child.get(0, END))
857                 global global_sim_data_listbox
858                 for item in list(cursors):
859                     x_add = parent.get(item)
860                     self.simulation_data_to_plot[x_add] = self.
861                     simulation_data_found[x_add]
862                     if x_add not in alls:
863                         child.insert(END, x_add)
864                         global_sim_data_listbox.insert(END, x_add
865 )
866             global global_sim_data
867             global_sim_data = self.simulation_data_to_plot
868
869             def remove_name(self, child):
870                 cursors = child.curselection()
871                 global global_sim_data_listbox
872                 for item in reversed(cursors):
873                     x_del = child.get(item)
874                     self.simulation_data_to_plot.pop(x_del, None)
875                     child.delete(item)
876                     global_sim_data_listbox.delete(item)
877
878             def load_file(self, cont):
879                 self.filename = askopenfilename(title='Select .
880             out file', filetypes=(('OUT File', '*.out'),))
881             if self.filename:

```

```

877             cont.label['text'] = self.filename
878
879     def save_file(self, cont):
880         self.home_location = filedialog.askdirectory(
881             title='Select save folder')
882         if self.home_location:
883             cont.label4['text'] = self.home_location
884
885     def read_folder(self, frame, f2a_listbox):
886         path_read = filedialog.askdirectory(title='Select
887             read folder')
888         if path_read:
889             f2a_listbox.delete(0, END)
890             frame.label5['text'] = path_read
891             self.local_simulations(path_to_check=path_read,
892             f2a_listbox=f2a_listbox)
893             self.read_location = path_read
894
895     @threded
896     def convert_file(self, cont):
897         self.button_convert['state'] = 'disabled'
898         Simulation(save_loc=self.home_location, file_name
899 =self.filename).convert()
900         self.button_convert['state'] = 'normal'
901
902
903     class PageOne(tk.Frame):
904         def __init__(self, parent, controller):
905             self.controller = controller
906             self.parent = parent
907             tk.Frame.__init__(self, parent)
908             self.f1_input = Frame(self, bg='grey')
909             self.f1_input.pack(side=TOP, padx=3, pady=3,
910             expand=0, fill='both')
911             self.f2_toolkit = Frame(self)
912             self.f2_toolkit.pack(side=TOP, fill='both',
913             expand=False)
914             self.f2_plot = Frame(self)
915             self.f2_plot.pack(side=TOP, padx=10, pady=10,
916             expand=1, fill='both')
917             global global_sim_data_listbox
918             global_sim_data_listbox = Listbox(self.f1_input,
919             height=5, selectmode=SINGLE, relief=SUNKEN,
920             exportselection=False)
921             global_sim_data_listbox.pack(side=LEFT, padx=2,

```

```

912    pady=2, fill='y')
913        global_sim_data_listbox.bind('<<ListboxSelect>>',
914            self.get_selected_item_prep)
914        self.prep_sim_parameters = Listbox(self.f1_input,
915            height=5, selectmode=SINGLE, relief=SUNKEN, width=24,
916            exportselection=False)
915        self.prep_sim_parameters.pack(side=LEFT, padx=2,
916        pady=2, fill='y')
916        self.prep_sim_parameters.bind('<<ListboxSelect>>'
917            , self.get_folded_properties)
917        self.local_sim_parameters = Listbox(self.f1_input
918            , height=5, selectmode=EXTENDED, relief=SUNKEN)
918        self.local_sim_parameters.pack(side=LEFT, padx=2,
919        pady=2, fill='y')
919        self.local_sim_parameters.bind('<<ListboxSelect
920            >>', self.get_multiple_items)
920        self.f3 = Frame(self.f1_input)
921        self.f3.pack(side=LEFT, expand=0, fill='both')
922        self.f3a = Frame(self.f3)
923        self.f3b = Frame(self.f3)
924        self.f3c = Frame(self.f3)
925        self.f3a.pack(side=TOP, expand=1)
926        self.f3b.pack(side=TOP, expand=1)
927        self.f3c.pack(side=TOP, expand=1)
928        self.fontsize = 12
929        self.hold3 = IntVar()
930        self.hold_choice3 = Checkbutton(self.f3b, text='
931            Hold3', variable=self.hold3, onvalue=0, offvalue=1)
931        self.hold_choice3.pack()
932        self.grid_dropdown_font = ttk.Combobox(self.f3b,
933            height=1, width=7, state='readonly')
933        self.grid_dropdown_font.pack()
934        self.label_font = Label(self.f3b, text='Font size
935            : ' + str(self.fontsize), height=1, width=10)
935        self.label_font.pack()
936        self.grid_dropdown_font['values'] = list(range(1,
937            100+1, 1))
937        self.grid_dropdown_font.current(self.fontsize-1)
938        self.grid_dropdown_font.bind('<<ComboboxSelected
939            >>', self.get_fontsize)
939        self.button_add_plots = ttk.Button(self.f3b, text
940            ='Add', width=10, command=self.add_to_plot_list)
940        self.button_add_plots.pack()
941        self.button_remove_plots = ttk.Button(self.f3b,
941            text='Remove', width=10, command=self.

```

```

941     remove_from_plot_list)
942             self.button_remove_plots.pack()
943             self.button_remove_x = ttk.Button(self.f3b, text=
'Clear X', width=10, command=lambda: self.clear_xy(
typedata='X'))
944             self.button_remove_x.pack()
945             self.button_remove_y = ttk.Button(self.f3b, text=
'Clear Y', width=10, command=lambda: self.clear_xy(
typedata='Y'))
946             self.button_remove_y.pack()
947
948             self.listboxes_frame = Frame(self.f1_input, bg='
red')
949             self.listboxes_frame.pack(side=LEFT, padx=2, pady
=2, fill='both', expand=0)
950
951             self.xy_listbox_frame = Frame(self.
listboxes_frame, bg='white')
952             self.xy_listbox_frame.pack(side=TOP, padx=2, pady
=2, fill='both', expand=0)
953             self.x_listbox = Listbox(self.xy_listbox_frame,
height=1, selectmode=None, relief=SUNKEN)
954             self.x_listbox.grid(column=0, row=0, sticky='nw',
padx=2, pady=2, ipady=2)
955             self.x_button_frame = Frame(self.xy_listbox_frame
, width=50, height=25)
956             self.x_button_frame.grid(column=1, row=0, sticky=
'nw', padx=2, pady=2)
957             self.x_button_frame.pack_propagate(0)
958             self.x_button = ttk.Button(self.x_button_frame,
text='Add X', command=lambda: self.add_to_xy(typedata='X'
))
959             self.x_button.pack(expand=1, fill='both')
960             self.y_listbox = Listbox(self.xy_listbox_frame,
height=1, selectmode=None, relief=SUNKEN)
961             self.y_listbox.grid(column=0, row=1, sticky='nw',
padx=2, pady=2, ipady=2)
962             self.y_button_frame = Frame(self.xy_listbox_frame
, width=50, height=25)
963             self.y_button_frame.grid(column=1, row=1, sticky=
'nw', padx=2, pady=2)
964             self.y_button_frame.pack_propagate(0)
965             self.y_button = ttk.Button(self.y_button_frame,
text='Add Y', command=lambda: self.add_to_xy(typedata='Y'
))

```

```

966         self.y_button.pack(expand=1, fill='both')
967
968         self.z_listbox_frame = Frame(self.
969             listboxes_frame, bg='blue')
969             self.z_listbox_frame.pack(side=TOP, padx=2, pady
969 =2, fill='both', expand=1)
970             self.pageone_listbox_plot = Listbox(self.
970                 z_listbox_frame, height=5, selectmode=EXTENDED, relief=
970 SUNKEN)
971             self.pageone_listbox_plot.pack(side=LEFT, padx=2
971 , pady=2, fill='both', expand=1)
972             self.pageone_listbox_plot.bind('<<ListboxSelect
972 >>', self.get_plot_titles)
973
974         self.checkmarks = Frame(self.f1_input, bg='black
974 ')
975             self.checkmarks.pack(side=LEFT, expand=0, fill='
975 both')
976             self.checkmarks_a = Frame(self.checkmarks, bg='
976 green')
977             self.checkmarks_a.pack(side=TOP, expand=0, fill='
977 both')
978             self.checkmarks_b = Frame(self.checkmarks, bg='
978 white')
979             self.checkmarks_b.pack(side=TOP, expand=0, fill='
979 both')
980             self.checkmarks_c = Frame(self.checkmarks, bg='
980 orange')
981             self.checkmarks_c.pack(side=TOP, expand=1, fill='
981 both')
982             self.checkmarks_d = Frame(self.checkmarks, bg='
982 orange')
983             self.checkmarks_d.pack(side=TOP, expand=1, fill='
983 both')
984             self.figs = 0
985             self.label_figs = Label(self.checkmarks_a, text=
985 'Figures: ' + str(self.figs), height=1, relief=SUNKEN,
985 width=9)
986             self.label_figs.grid(column=0, row=0, sticky='nw
986 ', padx=3, pady=3, ipady=2)
987             self.grid_dropdown = ttk.Combobox(self.
987 checkmarks_a, height=1, width=4)
988             self.grid_dropdown.grid(column=1, row=0, sticky=
988 'nw', padx=3, pady=3, ipady=2)
989             self.grid_button = ttk.Button(self.checkmarks_a,

```

```

989     text='Row: Col:', command=self.popup)
990         self.grid_button.grid(column=2, row=0, sticky='
991             nw', padx=3, pady=3)
991             self.sharex = IntVar()
992             self.sharey = IntVar()
993             self.showtime = IntVar()
994             self.showsimcase = IntVar()
995             self.plottype = IntVar()
996             self.hold = IntVar()
997             self.hold2 = IntVar()
998             self.sharex_choice = Checkbutton(self.
checkmarks_b, text='Share X', variable=self.sharex,
onvalue=1, offvalue=0, bg='grey')
999                 self.sharex_choice.grid(column=0, row=0, sticky=
'nw', padx=3, pady=3)
1000                 self.sharey_choice = Checkbutton(self.
checkmarks_b, text='Share Y', variable=self.sharey,
onvalue=1, offvalue=0, bg='grey')
1001                     self.sharey_choice.grid(column=1, row=0, sticky=
'nw', padx=3, pady=3)
1002                     self.showtime_choice = Checkbutton(self.
checkmarks_b, text='Time', variable=self.showtime,
onvalue=0, offvalue=1, bg='grey')
1003                     self.showtime_choice.grid(column=0, row=1,
sticky='nw', padx=3, pady=3)
1004                     self.showsimcase_choice = Checkbutton(self.
checkmarks_b, text='Simcase', variable=self.showsimcase,
onvalue=0, offvalue=1, bg='grey')
1005                         self.showsimcase_choice.grid(column=1, row=1,
sticky='nw', padx=3, pady=3)
1006                         self.hold_choice = Checkbutton(self.checkmarks_b
, text='Hold', variable=self.hold, onvalue=0, offvalue=1
, bg='grey')
1007                         self.hold_choice.grid(column=2, row=0, sticky='
nw', padx=3, pady=3)
1008                         self.hold_choice2 = Checkbutton(self.
checkmarks_b, text='Hold2', variable=self.hold2, onvalue
=1, offvalue=0, bg='grey')
1009                         self.hold_choice2.grid(column=2, row=1, sticky='
nw', padx=3, pady=3)
1010                         self.xyz = IntVar()
1011                         self.xy = Radiobutton(self.checkmarks_c, text='
xy', variable=self.xyz, value=1, bg='grey', command=
lambda: self.set_xyz())
1012                         self.yx = Radiobutton(self.checkmarks_c, text='

```

```

1012     'yx', variable=self.xyz, value=2, bg='grey', command=
        lambda: self.set_xyz())
1013         self.xz = Radiobutton(self.checkmarks_c, text='
        'xz', variable=self.xyz, value=3, bg='grey', command=
        lambda: self.set_xyz())
1014         self.zx = Radiobutton(self.checkmarks_c, text='
        'zx', variable=self.xyz, value=4, bg='grey', command=
        lambda: self.set_xyz())
1015         self.zy = Radiobutton(self.checkmarks_c, text='
        'zy', variable=self.xyz, value=5, bg='grey', command=
        lambda: self.set_xyz())
1016         self.yz = Radiobutton(self.checkmarks_c, text='
        'yz', variable=self.xyz, value=6, bg='grey', command=
        lambda: self.set_xyz())
1017         self.plottype_choice = Checkbutton(self.
        checkmarks_c, text='Plot type', variable=self.plottype,
        onvalue=0, offvalue=1, bg='grey')
1018
1019         self.xyz_reset = Radiobutton(self.checkmarks_d,
        text='reset', variable=self.xyz, value=9, bg='grey',
        command=lambda: self.set_xyz())
1020         self.xdays = Radiobutton(self.checkmarks_d, text=
        'x-days', variable=self.xyz, value=7, bg='grey',
        command=lambda: self.set_xyz())
1021         self.ydays = Radiobutton(self.checkmarks_d, text=
        'y-days', variable=self.xyz, value=8, bg='grey',
        command=lambda: self.set_xyz())
1022
1023         self.xy.grid(column=0, row=1, sticky='nw', padx=
        3, pady=3)
1024         self.yx.grid(column=1, row=1, sticky='nw', padx=
        3, pady=3)
1025         self.xz.grid(column=2, row=1, sticky='nw', padx=
        3, pady=3)
1026         self.zx.grid(column=0, row=2, sticky='nw', padx=
        3, pady=3)
1027         self.zy.grid(column=1, row=2, sticky='nw', padx=
        3, pady=3)
1028         self.yz.grid(column=2, row=2, sticky='nw', padx=
        3, pady=3)
1029         self.plottype_choice.grid(column=3, row=1,
        sticky='nw', padx=3, pady=3)
1030         self.xyz_reset.grid(column=0, row=0, sticky='nw'
        , padx=3, pady=3)
1031         self.xdays.grid(column=1, row=0, sticky='nw',

```

```

1031    padx=3, pady=3)
1032        self.ydays.grid(column=2, row=0, sticky='nw',
1033        padx=3, pady=3)
1033
1034        self.slide_and_gelmod = Frame(self.f1_input, bg=
1035        'black')
1035        self.slide_and_gelmod.pack(side=LEFT, expand=0,
1036        fill='both')
1036        self.sliders = Frame(self.slide_and_gelmod, bg='
1037        red')
1037        self.sliders.pack(side=TOP, expand=0, fill='both
1038        ')
1038        self.prep_gelmods = Frame(self.slide_and_gelmod,
1039        bg='grey')
1039        self.prep_gelmods.pack(side=TOP, expand=1, fill=
1040        'both')
1040        self.gelmods = Frame(self.prep_gelmods, bg='grey
1041        ')
1041        self.gelmods.pack(side=LEFT, expand=1, fill='
1042        both')
1042        self.buttons_gelmods = Frame(self.prep_gelmods,
1043        bg='grey')
1043        self.buttons_gelmods.pack(side=LEFT, expand=1,
1044        fill='both')
1044
1045        # Component (1) - Na (ppm)
1046        self.compl = Text(self.gelmods, height=1, width=
1047        6)
1047        self.complmid = Text(self.gelmods, height=1,
1048        width=5)
1048        self.complend = Text(self.gelmods, height=1,
1049        width=6)
1049        self.compl_label = Label(self.gelmods, text='Na
1050        (ppm) : ', height=1, relief=SUNKEN, width=9)
1050        self.compl_label2 = Label(self.gelmods, text=': '
1051        , height=1)
1051        self.compl_label3 = Label(self.gelmods, text=': '
1052        , height=1)
1053        self.compl_label.grid(column=0, row=0, sticky='
1053        nw', padx=3, pady=3)
1054        self.compl.grid(column=1, row=0, sticky='nw',
1054        padx=3, pady=3)
1055        self.compl_label2.grid(column=2, row=0, sticky='
1055        nw', padx=3, pady=3)

```

```

1056         self.complmid.grid(column=3, row=0, sticky='nw',
1057             padx=3, pady=3)
1058         self.comp1_label3.grid(column=4, row=0, sticky='
1059             nw', padx=3, pady=3)
1060         self.complend.grid(column=5, row=0, sticky='nw',
1061             padx=3, pady=3)
1062
1063             # Component (2) - Ca (ppm)
1064             self.comp2 = Text(self.gelmods, height=1, width=
1065                 6)
1066             self.comp2mid = Text(self.gelmods, height=1,
1067                 width=5)
1068             self.comp2end = Text(self.gelmods, height=1,
1069                 width=6)
1070             self.comp2_label = Label(self.gelmods, text='Ca
1071                 (ppm) : ', height=1, relief=SUNKEN, width=9)
1072             self.comp2_label2 = Label(self.gelmods, text=':'
1073                 , height=1)
1074             self.comp2_label3 = Label(self.gelmods, text=':'
1075                 , height=1)
1076             self.comp2_label.grid(column=0, row=1, sticky='
1077                 nw', padx=3, pady=3)
1078             self.comp2.grid(column=1, row=1, sticky='nw',
1079                 padx=3, pady=3)
1080             self.comp2_label2.grid(column=2, row=1, sticky='
1081                 nw', padx=3, pady=3)
1082             self.comp2mid.grid(column=3, row=1, sticky='nw',
1083                 padx=3, pady=3)
1084             self.comp2label3.grid(column=4, row=1, sticky='
1085                 nw', padx=3, pady=3)
1086             self.comp2end.grid(column=5, row=1, sticky='nw',
1087                 padx=3, pady=3)
1088
1089             # Component (3) - T (°C)
1090             self.comp3 = Text(self.gelmods, height=1, width=
1091                 6)
1092             self.comp3mid = Text(self.gelmods, height=1,
1093                 width=5)
1094             self.comp3end = Text(self.gelmods, height=1,
1095                 width=6)
1096             self.comp3_label = Label(self.gelmods, text='
1097                 Temp (°C) : ', height=1, relief=SUNKEN, width=9)
1098             self.comp3_label2 = Label(self.gelmods, text=':'
1099                 , height=1)

```

```

1081         self.comp3_label3 = Label(self.gelmods, text=':')
1082         , height=1)
1083         self.comp3_label.grid(column=0, row=2, sticky='nw',
1084         padx=3, pady=3)
1085         self.comp3.grid(column=1, row=2, sticky='nw',
1086         padx=3, pady=3)
1087         self.comp3_label2.grid(column=2, row=2, sticky='nw',
1088         padx=3, pady=3)
1089         self.comp3mid.grid(column=3, row=2, sticky='nw',
1090         padx=3, pady=3)
1091         self.comp3_label3.grid(column=4, row=2, sticky='nw',
1092         padx=3, pady=3)
1093         self.comp3end.grid(column=5, row=2, sticky='nw',
1094         padx=3, pady=3)
1095         # self.complbutton.grid(column=0, row=0, sticky='nw',
1096         # padx=3, pady=3)
1097         self.complbutton = ttk.Button(self.
1098             buttons_gelmods, text='Set', command=lambda: self.
1099                 set_gelmod())
1100         self.complbutton.pack(side=TOP, expand=1, fill='both',
1101         padx=3, pady=1)
1102         self.complbutton2 = ttk.Button(self.
1103             buttons_gelmods, text='Reset', command=lambda: self.
1104                 reset_gelmod())
1105         self.complbutton2.pack(side=TOP, expand=1, fill='both',
1106         padx=3, pady=1)
1107         self.complbutton3 = ttk.Button(self.
1108             buttons_gelmods, text='Plot gelmod', command=lambda:
1109                 self.plot_gelmod())
1110         self.complbutton3.pack(side=TOP, expand=1, fill='both',
1111         padx=3, pady=1)
1112
1113         # SLIDERS
1114         self.slidex_label1, self.slidex_left, self.
1115             slidex_right, self.slidex_label2, self.valuex1, self.
1116             freezex1 = [None] * 6
1117         self.slidey_label1, self.slidey_left, self.
1118             slidey_right, self.slidey_label2, self.valuey1, self.
1119             freezey1 = [None] * 6
1120         self.slidez_label1, self.slidez_left, self.
1121             slidez_right, self.slidez_label2, self.valuez1, self.
1122             freezez1 = [None] * 6
1123         self.slidetime_label1, self.slidetime_left, self

```

```

1102 .slidetime_right, self.slidetime_label2, self.valuetime1
    , self.freezeftime1 = [None] * 6
1103         self.last_settings = {}
1104         self.last_settings_old = {}
1105         for dim in ['imin', 'imax', 'jmin', 'jmax', 'kmin',
1106             'kmax', 'tmin', 'tmax']:
1107             self.last_settings[dim] = 1
1108             self.x_input, self.x_time = (None, None)
1109             # SLIDERS
1110
1111             self.f6 = Frame(self.f1_input, bg='white')
1112             self.f6.pack(side=LEFT, expand=1, fill='both')
1113             self.plotlabels = Frame(self.f6)
1114             self.plotlabels.pack(side=TOP, expand=1, fill='both')
1115             self.plotlabels_toppart = Frame(self.plotlabels,
1116                 bg='black')
1117             self.plotlabels_toppart.pack(side=TOP, expand=0,
1118                 fill='x')
1119             self.plotlabels_labels = Frame(self.
1120                 plotlabels_toppart, bg='red')
1121             self.plotlabels_labels.pack(side=LEFT, expand=0,
1122                 fill='both')
1123             self.plotlabels_buttons = Frame(self.
1124                 plotlabels_toppart)
1125             self.plotlabels_buttons.pack(side=LEFT, expand=1
1126                 , fill='both')
1127             self.ptitle_label = Label(self.plotlabels_labels
1128                 , text='Plot title: ', height=1, relief=SUNKEN, width=9)
1129             self.pxtitle_label = Label(self.
1130                 plotlabels_labels, text='X label: ', height=1, relief=
1131                 SUNKEN, width=9)
1132             self.pytitle_label = Label(self.
1133                 plotlabels_labels, text='Y label: ', height=1, relief=
1134                 SUNKEN, width=9)
1135             self.ptitle = Entry(self.plotlabels_labels,
1136                 width=40)
1137             self.pxtitle = Entry(self.plotlabels_labels,
1138                 width=40)
1139             self.pytitle = Entry(self.plotlabels_labels,
1140                 width=40)
1141             self.ptitle_label.grid(column=0, row=0, sticky='
1142                 nw', padx=3, pady=3)
1143             self.ptitle.grid(column=1, row=0, sticky='nw',
1144                 padx=3, pady=3)

```

```

1128         self.title_set = ttk.Button(self.
1129             plotlabels_buttons, text='Set', command=lambda: self.
1130                 set_plot_labels())
1129         self.title_reset = ttk.Button(self.
1130             plotlabels_buttons, text='Reset', command=lambda: self.
1131                 reset_plot_labels())
1130         self.title_set.pack(anchor='nw', expand=1, fill=
1131             'y', padx=3, pady=3)
1131         self.title_reset.pack(anchor='nw', expand=1,
1132             fill='y', padx=3, pady=3)
1132         self.pxtitle_label.grid(column=0, row=1, sticky=
1133             'nw', padx=3, pady=3)
1133         self.pxtitle.grid(column=1, row=1, sticky='nw',
1134             padx=3, pady=3)
1134         self.pytitle_label.grid(column=0, row=2, sticky=
1135             'nw', padx=3, pady=3)
1135         self.pytitle.grid(column=1, row=2, sticky='nw',
1136             padx=3, pady=3)
1137         button1 = ttk.Button(self.f1_input, text='Plot
1138             it', command=self.plot_graphv2)
1138         button1.pack(padx=3, pady=3)
1139         button2 = ttk.Button(self.f1_input, text='Delete
1140             all', command=lambda: self.delete_figures(2))
1140         button2.pack(padx=3, pady=3)
1141         button3 = ttk.Button(self.f1_input, text='Save
1142             setup', command=lambda: self.store_settings())
1142         button3.pack(padx=3, pady=3)
1143         button4 = ttk.Button(self.f1_input, text='
1144             Restore', command=lambda: self.restore_settings())
1144         button4.pack(padx=3, pady=3)
1145         self.doitonce = 0
1146         self.properties_available = {}
1147         self.last_select = []
1148         self.properties_conversion = {}
1149         self.properties_plot_these = {}
1150         self.reference = {}
1151         self.final_plot_data = {}
1152         self.canvas = None
1153         self.toolbar = None
1154         self.fig_grid_size = {}
1155         self.chosen_rows_alt = None
1156         self.chosen_cols_alt = None
1157         self.w = None
1158         self.browse_days = {}

```

```

1159         self.tmin_stored, self.tmax_stored = (None, None)
1160
1161         self.current_selection_v2 = None
1162         self.merged_listbox_items = {}
1163         self.simcase_child = None
1164         self.simcase = None
1165         self.simcase_path = None
1166         self.data_conversion = {}
1167         self.plot_id = None
1168         self.plot_rdy = {}
1169         self.plot_x = {}
1170         self.plot_y = {}
1171         self.simcase_ijkt_count = {}
1172         self.plot_id_old = {}
1173         self.plot_id_hist = []
1174         self.settings_stored = 0
1175         self.tlimits = []
1176
1177         self.comp_na_start = 0
1178         self.comp_na_mid = 2000
1179         self.comp_na_end = 16000
1180         self.compl.insert(END, self.comp_na_start)
1181         self.complmid.insert(END, self.comp_na_mid)
1182         self.complend.insert(END, self.comp_na_end)
1183
1184         self.comp_ca_start = 0
1185         self.comp_ca_mid = 50
1186         self.comp_ca_end = 500
1187         self.comp2.insert(END, self.comp_ca_start)
1188         self.comp2mid.insert(END, self.comp_ca_mid)
1189         self.comp2end.insert(END, self.comp_ca_end)
1190
1191         self.comp_temp_start = 10
1192         self.comp_temp_mid = 10
1193         self.comp_temp_end = 140
1194         self.comp3.insert(END, self.comp_temp_start)
1195         self.comp3mid.insert(END, self.comp_temp_mid)
1196         self.comp3end.insert(END, self.comp_temp_end)
1197
1198         self.comp_na = list(range(self.comp_na_start,
1199             self.comp_na_end+1, self.comp_na_mid))
1198         self.comp_ca = list(range(self.comp_ca_start,
1199             self.comp_ca_end+1, self.comp_ca_mid))
1199         self.comp_temp = list(range(self.comp_temp_start
, self.comp_temp_end+1, self.comp_temp_mid))

```

```

1200
1201         self.change_plot_label_current = None
1202         self.shown_title_old = None
1203         self.plot_title = None
1204         self.plot_xlabel = None
1205         self.plot_ylabel = None
1206
1207     def set_plot_labels(self):
1208         if self.change_plot_label_current:
1209             value = self.change_plot_label_current
1210             title_input = self.ptitle.get()
1211             user_title = str(title_input)
1212             split_title = user_title.split(' ')
1213             new_title = ' '.join(split_title)
1214
1215             xlabel, ylabel = (None, None)
1216             user_xlabel, user_ylabel = (self.pxtitle.get(),
1217                                         self.pytitle.get())
1217             if user_xlabel:
1218                 xlabel = str(user_xlabel)
1219                 self.plot_xlabel = xlabel
1220             if user_ylabel:
1221                 ylabel = str(user_ylabel)
1222                 self.plot_ylabel = ylabel
1223             newvalues = {'shown_title': new_title,
1224                         'xlabel': xlabel, 'ylabel': ylabel}
1225             self.plot_rdy[value][1]['title'].update(
1226                 newvalues)
1227             else:
1228                 title_input = self.ptitle.get()
1229                 if title_input:
1230                     user_title = str(title_input)
1231                     split_title = user_title.split(' ')
1232                     new_title = ' '.join(split_title)
1233                     self.plot_title = new_title
1234                     xlabel, ylabel = (None, None)
1235                     user_xlabel, user_ylabel = (self.pxtitle.get(),
1236                                         self.pytitle.get())
1237                     if user_xlabel:
1238                         xlabel = str(user_xlabel)
1239                         self.plot_xlabel = xlabel
1240                     if user_ylabel:

```

```

1241     def reset_plot_labels(self):
1242         value = self.change_plot_label_current
1243         newvalues = { 'shown_title': self.shown_title_old
1244 , 'xlabel': None, 'ylabel': None}
1244         self.plot_rdy[value][1]['title'].update(
1245             newvalues)
1245         self.ptitle.delete(0, END)
1246         self.ptitle.insert(END, newvalues['shown_title'])
1247
1247         self.pxtitle.delete(0, END)
1248         self.pytitle.delete(0, END)
1249         self.plot_title = None
1250         self.plot_xlabel = None
1251         self.plot_ylabel = None
1252
1253     def get_plot_titles(self, event):
1254         w = event.widget
1255         index = w.curselection()
1256         parent = self.pageone_listbox_plot
1257         if len(index) == 1:
1258             value = parent.get(index)
1259             self.change_plot_label_current = value
1260             element = self.plot_rdy[value]
1261             title_elements = element[1]['title']
1262             shown_title = title_elements['shown_title']
1263             self.shown_title_old = shown_title
1264
1265             self.ptitle.delete(0, END)
1266             self.pxtitle.delete(0, END)
1267             self.pytitle.delete(0, END)
1268             self.ptitle.insert(END, shown_title)
1269
1270     def get_fontsize(self, event):
1271         w = event.widget
1272         self.fontsize = int(w.get())
1273         self.label_font['text'] = 'Font size: ' + str(
1274             self.fontsize)
1274
1275     def set_gelmod(self):
1276         na_start = int(self.compl.get('1.0', END))
1277         na_mid = int(self.complmid.get('1.0', END))
1278         na_end = int(self.complend.get('1.0', END))
1279         ca_start = int(self.comp2.get('1.0', END))
1280         ca_mid = int(self.comp2mid.get('1.0', END))
1281         ca_end = int(self.comp2end.get('1.0', END))

```

```

1282         temp_start = int(self.comp3.get('1.0', END))
1283         temp_mid = int(self.comp3mid.get('1.0', END))
1284         temp_end = int(self.comp3end.get('1.0', END))
1285
1286         self.comp_na_start = na_start
1287         self.comp_na_mid = na_mid
1288         self.comp_na_end = na_end
1289         self.comp_ca_start = ca_start
1290         self.comp_ca_mid = ca_mid
1291         self.comp_ca_end = ca_end
1292         self.comp_temp_start = temp_start
1293         self.comp_temp_mid = temp_mid
1294         self.comp_temp_end = temp_end
1295
1296         self.comp_na = list(range(na_start, na_end+1,
1297                                   na_mid))
1297         self.comp_ca = list(range(ca_start, ca_end+1,
1298                                   ca_mid))
1298         self.comp_temp = list(range(temp_start, temp_end
1299                               +1, temp_mid))
1300
1300     def reset_gelmod(self):
1301         self.comp1.delete('1.0', END)
1302         self.comp1mid.delete('1.0', END)
1303         self.complend.delete('1.0', END)
1304         self.comp2.delete('1.0', END)
1305         self.comp2mid.delete('1.0', END)
1306         self.comp2end.delete('1.0', END)
1307         self.comp3.delete('1.0', END)
1308         self.comp3mid.delete('1.0', END)
1309         self.comp3end.delete('1.0', END)
1310
1311         self.comp1.insert(END, self.comp_na_start)
1312         self.comp1mid.insert(END, self.comp_na_mid)
1313         self.complend.insert(END, self.comp_na_end)
1314         self.comp2.insert(END, self.comp_ca_start)
1315         self.comp2mid.insert(END, self.comp_ca_mid)
1316         self.comp2end.insert(END, self.comp_ca_end)
1317         self.comp3.insert(END, self.comp_temp_start)
1318         self.comp3mid.insert(END, self.comp_temp_mid)
1319         self.comp3end.insert(END, self.comp_temp_end)
1320
1321     def plot_gelmod(self):
1322         fontsize = self.fontsize
1323         matplotlib.rcParams.update({'font.size':

```

```

1323 fontsize))
1324         self.delete_figures(2)
1325         share_axis, filename, aspect_wanted, aspect_auto
= (False, '', 1, True)
1326         chosen_rows, chosen_cols = (1, 1)
1327
1328         sharex_local, sharey_local = (False, False)
1329         fig, axes = (None, None)
1330         change_plot = self.plottype.get() # Allow user
to change this
1331         plot_version = None
1332         if change_plot == 1:
1333             plot_version = 1
1334             fig, axes = plt.subplots(
1335                 nrows=chosen_rows, ncols=chosen_cols,
sharex=sharex_local, sharey=sharey_local, figsize=(10,
10))
1336         elif change_plot == 0:
1337             plot_version = 0
1338             fig = Figure(figsize=(10, 10))
1339
1340         alpha_values = [2.000, 0.001, 0.017]
1341         beta_values = [1.0, 0.9]
1342         yield_values = [1.0, 0.0, 0.0]
1343         rg, eag, tref = [math.pow(10,-4), 77, 20]
1344         crit, surface_area = [0.20, 200]
1345         rvalue = 0.008314 # kj / K mol
1346         tref_kelvin = float(tref+273.15)
1347         conc_si = 10
1348         inner_factor_tref = float(eag/(rvalue*
tref_kelvin))
1349         effect_of_si = math.pow(conc_si, alpha_values[0])
)
1350         effect_of_tref = math.exp(inner_factor_tref)
1351
1352         xvalues_gelmod = []
1353         yvalues_gelmod = self.comp_temp
1354         zvalues_gelmod = []
1355         ivalue, jvalue, kvalue = [1]*3
1356         kvalues_check = []
1357         combined_check = []
1358         combined_check_values = []
1359         for i_na in self.comp_na:
1360             for j_ca in self.comp_ca:
inner_factor_na = math.pow(i_na,

```

```

1361 beta_values[0])
1362 na_exponent = alpha_values[1] *
    inner_factor_na
1363 effect_of_na = math.exp(na_exponent)
1364 inner_factor_ca = math.pow(j_ca,
    beta_values[1])
1365 ca_exponent = alpha_values[2] *
    inner_factor_ca
1366 effect_of_ca = math.exp(ca_exponent)
1367 # xvalue = float(effect_of_na/
    effect_of_na+effect_of_ca))
1368 xvalue = float(effect_of_na/effect_of_ca
    )
1369 # inner_xvalue = (1/(i_na+1)) + (1/(j_ca
    +1)) + (1/(1+(i_na*j_ca)))
1370 # inner_xvalue = (1 / (i_na + 1)) + (1
    / (j_ca + 1))
1371 # xvalue = math.pow(inner_xvalue,-1)
1372 xvalues_gelmod.append(xvalue)
1373 for k_temp in self.comp_temp:
    temp_kelvin = float(k_temp+273.15)
    inner_factor_temp = -(eag/(rvalue*
        temp_kelvin))
1376 effect_of_temp = math.exp(
    inner_factor_temp)
1377 zvalue = float(rg*effect_of_si*
    effect_of_na*effect_of_ca*effect_of_tref*effect_of_temp)
1378 zvalues_gelmod.append(zvalue)
1379 kvalues_check.append(kvalue)
1380 combined_check.append([ivalue,jvalue
    ,kvalue])
1381 combined_check_values.append([int(
    i_na),int(j_ca),xvalue])
1382 kvalue += 1
1383 print('ivalue: ' + str(ivalue) +
    'jvalue: ' + str(jvalue))
1384 jvalue += 1
1385 ivalue += 1
1386 print(np.array(combined_check))
1387 print(np.array(combined_check_values))
1388
1389 xi, yj = (xvalues_gelmod, yvalues_gelmod)
1390 dxj, dyj = ([1.0]*len(xi), [self.comp_temp_mid]*
    len(yj))
1391

```

```

1392         xbound = self.get_block_boundaries(cellvalues=xi
1393             , cellwidths=dxi)
1393         ybound = self.get_block_boundaries(cellvalues=yj
1394             , cellwidths=dyj)
1394
1395         x_id_v2, y_id_v2, z_id_v2 = (xi, yj,
1396             zvalues_gelmod)
1396         # x_id_v2, y_id_v2, z_id_v2 = (xi, yj,
1397             zvalues_gelmod)
1397         xlength, ylength = (len(x_id_v2), len(y_id_v2))
1398
1399         transpose_choice, rowshape, colshape = (None,
1400             None, None)
1400         rowshape, colshape = (xlength, ylength)
1401         transpose_choice = 0
1402
1403         x_grid, y_grid = np.meshgrid(x_id_v2, y_id_v2)
1404         z_grid = np.reshape(np.array(z_id_v2), (rowshape
1405             , colshape))
1405
1406         ax, prev_ax, im = (None, None, None)
1407         if plot_version == 0:
1408             ax = fig.add_subplot(chosen_rows,
1409                 chosen_cols, 1)
1409             elif plot_version == 1:
1410                 ax = axes
1411
1412         xlabel, ylabel, title = ('x', 'Temperature (°C)'
1413             , 'Gelation rate')
1413         if self.plot_title:
1414             title = self.plot_title
1415         if self.plot_xlabel:
1416             xlabel = self.plot_xlabel
1417         if self.plot_ylabel:
1418             ylabel = self.plot_ylabel
1419         ax.set_title(title)
1420         ax.set_xlabel(xlabel)
1421         ax.set_ylabel(ylabel)
1422         ax.xaxis.set_tick_params(which='both',
1423             labelbottom=True)
1423         ax.yaxis.set_tick_params(which='both',
1424             labelbottom=True)
1424         ax.set_xscale('log')
1425         # ax.set_yscale('log')
1426         # norm = clr.Normalize()

```

```

1427      # cmap = cm.get_cmap('gist_rainbow')
1428      # cmap = 'PuBu_r'
1429      # im = ax.pcolormesh(x_grid, y_grid, z_grid,
1430      norm=matplotlib.colors.LogNorm(vmin=z_grid.min(), vmax=
1431      z_grid.max()), cmap='PuBu_r')
1430      im = ax.pcolormesh(x_grid, y_grid, z_grid.T,
1431      norm=matplotlib.colors.LogNorm(), cmap='PuBu_r')
1431      # im = ax.pcolor(x_grid, y_grid, z_grid, norm=
1432      matplotlib.colors.LogNorm(vmin=z_grid.min(), vmax=z_grid
1433      .max()), cmap='gist_rainbow')
1432
1433      # if transpose_choice == 0:
1434      #     im = ax.pcolormesh(x_grid, y_grid, z_grid
1435      , cmap=cmap, norm=norm)
1436      # elif transpose_choice == 1:
1437      #     im = ax.pcolormesh(x_grid, y_grid, z_grid.
1438      T, cmap=cmap, norm=norm)
1439
1440      print('... ')
1441      print(': transposed?? ' + str(transpose_choice))
1442      print('xlength: ' + str(len(x_id_v2)) + '
1443      ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(len(
1444      z_id_v2)))
1445      # print('rowshape, colshape: (' + str(rowshape
1446      ) + '[' + str(rowtype) + '], ' + str(colshape) + '[' +
1447      str(coltype) + '])')
1448      print('xshape: ' + str(x_grid.shape) + ' yshape
1449      : ' + str(y_grid.shape) + ' zshape: ' + str(z_grid.shape
1450      ))
1451      print('... ')
1452
1453
1454      if aspect_auto is False:
1455          aspect_ratio_wanted = aspect_wanted
1456          aspect_ratio_correct = abs((x_max - x_min) /
1457          (y_max - y_min)) / aspect_ratio_wanted
1458          ax.set_aspect(aspect_ratio_correct)
1459
1460          fig.colorbar(im, ax=ax)
1461
1462          plt.tight_layout()
1463
1464          self.canvas = FigureCanvasTkAgg(fig, self.
1465          f2_plot)
1466          self.canvas.draw()
1467          self.canvas.get_tk_widget().pack(side=tk.TOP,

```

```

1456 fill=tk.BOTH, expand=True)
1457         self.canvas._tkcanvas.pack(side=tk.BOTTOM, fill=
1458             tk.BOTH, expand=True)
1459         self.toolbar = NavigationToolbar2Tk(self.canvas,
1460             self.f2_toolkit) # Toolbar is added to canvas
1461         self.toolbar.update()
1462
1463     def set_xyz(self):
1464         child2 = self.prep_sim_parameters
1465         child2.selection_clear(0, END)
1466         if self.freeze_x1 and self.freeze_y1 and self.
1467             freeze_z1 and self.freeze_time1:
1468             data = self.simcase_ijkt_count
1469             imin, imax, jmin, jmax, kmin, kmax, tmin,
1470             tmax = (data['imin'], data['imax'], data['jmin'], data['
1471                 jmax'], data['kmin'], data['kmax'], data['tmin'],
1472                 data['tmax'])
1473             current_xyz = self.xyz.get()
1474             iimin, iimax = (int(self.slidex_left.get()),
1475                 int(self.slidex_right.get()))
1476             jjmin, jjmax = (int(self.slidey_left.get()),
1477                 int(self.slidey_right.get()))
1478             kkmin, kkmax = (int(self.slidez_left.get()),
1479                 int(self.slidez_right.get()))
1480             ttmin, ttmax = (int(self.slidetime_left.get(
1481                 )), int(self.slidetime_right.get()))
1482             if current_xyz == 9:
1483                 self.xyz.set(0)
1484                 self.slidex_left.set(imin)
1485                 self.slidex_right.set(imax)
1486                 self.slidey_left.set(jmin)
1487                 self.slidey_right.set(jmax)
1488                 self.slidez_left.set(kmin)
1489                 self.slidez_right.set(kmax)
1490                 self.slidetime_left.set(tmin)
1491                 self.slidetime_right.set(tmax)
1492                 self.value_x1.set(0)
1493                 self.value_y1.set(0)
1494                 self.value_z1.set(0)
1495                 self.value_time1.set(0)
1496             elif current_xyz in [7,8]:
1497                 simcase = current_selection
1498                 core_path = global_sim_data[simcase][0]
1499                 chldx = self.x_listbox
1500                 chldy = self.y_listbox

```

```

1491                     allsx = list(childx.get(0,END))
1492                     allsy = list(childy.get(0,END))
1493
1494                     sliders = {'X': [self.slidex_left, self.
1495                         slidex_right], 'Y': [self.slidey_left, self.slidey_right
1496                         ], 'Z': [self.slidez_left, self.slidez_right], 'time': [
1497                         self.slidetime_left, self.slidetime_right]}
1498
1499                     freezes = {'X': [self.freezex1, self.
1500                         valuem1, self.slidex_label1, self.slidex_label2, sliders
1501                         ['X'][1]], 'Y': [self.freezeyle1, self.valuey1, self.
1502                         slidey_label1, self.slidey_label2, sliders['Y'][1]],
1503                         'Z': [self.freezeze1, self.
1504                         valuez1, self.slidez_label1, self.slidez_label2, sliders
1505                         ['Z'][1]], 'time': [self.freezetimel, self.valuetimel,
1506                         self.slidetime_label1, self.slidetime_label2, sliders['
1507                         time'][1]]}
1508
1509                     ijkvalues = {'X': [iimin, iimax, imin,
1510                         imax], 'Y': [jjmin, jjmax, jmin, jmax], 'Z': [kkmin,
1511                         kkmax, kmin, kmax], 'time': [ttmin, ttmax, tmin, tmax]}
1512
1513
1514                     if ttmin == ttmax:
1515                         self.slidetime_left.set(tmin)
1516                         self.slidetime_right.set(tmax)
1517                         self.valuetimel.set(0)
1518
1519
1520                     openlist, currentlist = (None, None)
1521                     self.plot_id_old = self.plot_id
1522                     self.plot_id = {'simcase': simcase, '
1523                         simcase_path': core_path, 'simcase_child': ['DATA',
1524                             ['DATA']], 'entries': ['Days'], 'cells': None, 'time':
1525                             None, 'X': None, 'Y': None}
1526
1527                     choices = ['X', 'Y']
1528                     choice = None
1529                     if current_xyz == 7 and allsx: # Replace
1530                         X with Days | Open fully Y
1531                         openlist = self.plot_y[allsy[0]]['
1532                             entries']
1533
1534                         currentlist = self.plot_x[allsx[0]][
1535                             'entries']
1536
1537                         choice = choices[0]
1538                     elif current_xyz == 8 and allsy: #
1539                         Replace Y with Days | Open fully X
1540                         openlist = self.plot_x[allsx[0]]['
1541                             entries']
1542
1543                         currentlist = self.plot_y[allsy[0]][

```

```

1515 'entries' ]
1516 choice = choices[1]
1517
1518 if openlist:
1519     if openlist[0] in ['X', 'Y', 'Z']
1520 and currentlist[0] in ['X', 'Y', 'Z']:
1521         self.add_to_xy(typedata=choice)
1522         closelist = ['X', 'Y', 'Z']
1523         closelist.pop(closelist.index(
1524             openlist[0]))
1525
1526         for item in openlist:
1527             min00, max00 = (ijkvalues[
1528                 item][2], ijkvalues[item][3])
1529             min01, max01 = (ijkvalues[
1530                 item][0], ijkvalues[item][1])
1531             slide_l, slide_r,
1532             checkbutton = (sliders[item][0], sliders[item][1],
1533             freezes[item][1])
1534             if min01 == max01:
1535                 slide_l.set(min00)
1536                 slide_r.set(max00)
1537                 checkbutton.set(0)
1538
1539             for item in closelist:
1540                 min00, max00 = (ijkvalues[
1541                     item][2], ijkvalues[item][3])
1542                 min01, max01 = (ijkvalues[
1543                     item][0], ijkvalues[item][1])
1544                 if min01 != max01:
1545                     freezeitem, checkbutton,
1546                     slide_l_label, slide_r_label, slide_l, slide_r =
1547                     (freezes[item][0], freezes[item][1], freezes[item][2],
1548                     freezes[item][3], sliders[item][0], freezes[item][4])
1549                     freezeitem.select()
1550                     self.freeze_val(
1551                         checkbutton, slide_l_label, slide_r_label, slide_r)
1552                     rnd = int(random.uniform(
1553                         (min00, max00 + 1)))
1554                     slide_l.set(rnd)
1555             elif currentlist[0] == 'Days':
1556                 self.add_to_xy(typedata=choice)
1557                 # New settings should be added automatically
1558                 self.plot_id = self.plot_id_old
1559             else:

```

```

1546             child = self.prep_sim_parameters
1547             alls = self.prep_sim_parameters.get(0,
1548             END)
1549             indexdata = alls.index('DATA')
1550             access = (indexdata,)
1551             self.prep_sim_parameters.selection_set(
1552             access)
1553             self.prep_sim_parameters.event_generate(
1554             '<<ListboxSelect>>')
1555
1556             simcase = current_selection
1557             core_path = global_sim_data[simcase][0]
1558
1559             self.valuex1.set(0)
1560             self.valuey1.set(0)
1561             self.valuez1.set(0)
1562             self.valuetime1.set(0)
1563             if current_xyz == 1 or current_xyz == 2:
1564                 # XY and YX
1565                 self.freeze_time1.select()
1566                 self.freeze_val_time(self.valuetime1
1567                 , self.slidetime_label1, self.slidetime_label2, self.
1568                 slidetime_right)
1569                 self.freeze_z1.select()
1570                 self.freeze_val(self.valuez1, self.
1571                 slidez_label1, self.slidez_label2, self.slidez_right)
1572
1573                 self.slidex_left.set(imin)
1574                 self.slidex_right.set(imax)
1575                 self.slidey_left.set(jmin)
1576                 self.slidey_right.set(jmax)
1577                 if kkmin != kkmax:
1578                     rnd_k = int(random.uniform(kmin,
1579                     kmax+1))
1580                     self.slidez_left.set(rnd_k)
1581                     if ttmin != ttmax:
1582                         rnd_t = int(random.uniform(tmin,
1583                         tmax+1))
1584                         self.slidetime_left.set(rnd_t)
1585
1586                     if current_xyz == 1:
1587                         self.plot_id_old = self.plot_id
1588                         self.plot_id = {'simcase':
1589                           simcase, 'simcase_path': core_path, 'simcase_child': [
1590                             'INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, ''

```

```

1579     'time': None, 'X': None, 'Y': None}
1580                     self.add_to_xy(typedata='X')
1581                     self.plot_id = {'simcase':
1582                         simcase, 'simcase_path': core_path, 'simcase_child': [
1583                             'INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None,
1584                             'time': None, 'X': None, 'Y': None}
1585                     self.add_to_xy(typedata='Y')
1586                     self.plot_id = self.plot_id_old
1587                     elif current_xyz == 2:
1588                         self.plot_id_old = self.plot_id
1589                         self.plot_id = {'simcase':
1590                             simcase, 'simcase_path': core_path, 'simcase_child': [
1591                                 'INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None,
1592                                 'time': None, 'X': None, 'Y': None}
1593                     self.add_to_xy(typedata='X')
1594                     self.plot_id = self.plot_id_old
1595                     elif current_xyz == 3 or current_xyz ==
1596                         4: # XZ and ZX
1597                         self.freezeTime1.select()
1598                         self.freeze_val_time(self.valuetime1
1599 , self.slidetime_label1, self.slidetime_label2, self.
1600 slidetime_right)
1601                         self.freezeY1.select()
1602                         self.freeze_val(self.valuey1, self.
1603 slidey_label1, self.slidey_label2, self.slidey_right)
1604
1605                         self.slidex_left.set(imin)
1606                         self.slidex_right.set(imax)
1607                         self.slidez_left.set(kmin)
1608                         self.slidez_right.set(kmax)
1609                         if jjmin != jjmax:
1610                             rnd_j = int(random.uniform(jmin,
1611 jmax + 1))
1612                             self.slidey_left.set(rnd_j)
1613                         if ttmin != ttmax:
1614                             rnd_t = int(random.uniform(tmin,
1615 tmax + 1))
1616                             self.slidetime_left.set(rnd_t)
1617
1618                         if current_xyz == 3:

```

```

1609                         self.plot_id_old = self.plot_id
1610                         self.plot_id = {'simcase':
1611                             simcase, 'simcase_path': core_path, 'simcase_child': [
1612                                 'INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, 'time':
1613                                 None, 'X': None, 'Y': None}
1614                         self.add_to_xy(typedata='X')
1615                         self.plot_id = {'simcase':
1616                             simcase, 'simcase_path': core_path, 'simcase_child': [
1617                                 'INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, 'time':
1618                                 None, 'X': None, 'Y': None}
1619                         self.add_to_xy(typedata='Y')
1620                         self.plot_id = self.plot_id_old
1621                         elif current_xyz == 4:
1622                             self.plot_id_old = self.plot_id
1623                             self.plot_id = {'simcase':
1624                                 simcase, 'simcase_path': core_path, 'simcase_child': [
1625                                     'INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, 'time':
1626                                     None, 'X': None, 'Y': None}
1627                             self.add_to_xy(typedata='X')
1628                             self.plot_id = self.plot_id_old
1629                             elif current_xyz == 5 or current_xyz ==
1630                                 6: # ZY and YZ
1631                                 self.freezeTime1.select()
1632                                 self.freeze_val_time(self.valuetime1,
1633                                     self.slidetime_label1, self.slidetime_label2, self.
1634                                     slidetime_right)
1635                                 self.freezeX1.select()
1636                                 self.freeze_val(self.valueX1, self.
1637                                     slidex_label1, self.slidex_label2, self.slidex_right)
1638                                 self.slidez_left.set(imin)
1639                                 self.slidez_right.set(imax)
1640                                 self.slidey_left.set(jmin)
1641                                 self.slidey_right.set(jmax)
1642                                 if iimin != iimax:
1643                                     rnd_x = int(random.uniform(imin,
1644                                         imax + 1))
1645                                     self.slidex_left.set(rnd_x)
1646                                 if ttmin != ttmax:
1647                                     rnd_t = int(random.uniform(tmin,

```

```

1636     tmax + 1))
1637                     self.slidetime_left.set(rnd_t)
1638
1639             if current_xyz == 5:
1640                 self.plot_id_old = self.plot_id
1641                 self.plot_id = {'simcase':
1642                     simcase, 'simcase_path': core_path, 'simcase_child': [
1643                         'INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None,
1644                         'time': None, 'X': None, 'Y': None}
1645                     self.add_to_xy(typedata='X')
1646                     self.plot_id = {'simcase':
1647                         simcase, 'simcase_path': core_path, 'simcase_child': [
1648                             'INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None,
1649                             'time': None, 'X': None, 'Y': None}
1650                     self.add_to_xy(typedata='Y')
1651                     self.plot_id = self.plot_id_old
1652
1653             elif current_xyz == 6:
1654                 self.plot_id_old = self.plot_id
1655                 self.plot_id = {'simcase':
1656                     simcase, 'simcase_path': core_path, 'simcase_child': [
1657                         'INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None,
1658                         'time': None, 'X': None, 'Y': None}
1659                     self.add_to_xy(typedata='X')
1660                     self.plot_id = self.plot_id_old
1661
1662             else:
1663                 self.xyz.set(0)
1664
1665         def fetch_data(self, path, value_returned,
1666             property_chosen, cells, time_element):
1667             sorted_dataframe, sorted_dataframe_small,
1668             sorted_dataframe_selection = (None, None, None)
1669             path_ar_input = os.path.join(path, 'INPUT' + '.'
1670                                         'parquet')
1671             x_input = pd.read_parquet(path_ar_input)
1672             ilist, jlist, klist, tlist = (time_element[2],
1673                 time_element[3], time_element[4], time_element[5])
1674             newcells = x_input.loc[(x_input['i'].isin(ilist)
1675                 ) & (x_input['j'].isin(jlist)) & (x_input['k'].isin(
1676                     klist)), 'Cell'].tolist()
1677             allcells = x_input['Cell'].tolist()

```

```

1663         # cells, times = (np.unique(cells), np.unique(
1664             times))
1664         filename = value_returned[0]
1665         if filename == 'COMP':
1666             path_ar_comp = os.path.join(path, 'COMP' +
1667                 '.parquet')
1667             x_comp = pd.read_parquet(path_ar_comp)
1668             pressure_chosen = value_returned[1] # 240.0
1669             component_chosen = value_returned[2] # Ca(
1670                 mg)
1670             sorted_dataframe = x_comp.loc[(x_comp['
1671                 Component'] == component_chosen) & (x_comp['Pressure']
1672                     == pressure_chosen), :].loc[times]
1671             sorted_dataframe_small = sorted_dataframe[
1672                 property_chosen]
1672             sorted_dataframe_selection =
1673                 sorted_dataframe_small.tolist()
1673             elif filename == 'REGION':
1674                 path_ar_region = os.path.join(path, 'REGION'
1675                     + '.parquet')
1675                 x_region = pd.read_parquet(path_ar_region)
1676                 region_chosen = value_returned[1]
1677                 component_chosen = value_returned[2]
1678                 sorted_dataframe = x_region.loc[(x_region['
1679                     Component'] == component_chosen) & (x_region['Region']
1680                         == region_chosen), :].loc[times]
1680                 sorted_dataframe_small = sorted_dataframe[
1681                     property_chosen]
1681                 sorted_dataframe_selection =
1682                     sorted_dataframe_small.tolist()
1681             elif filename == 'WELLS':
1682                 path_ar_wells = os.path.join(path, 'WELLS' +
1683                     '.parquet')
1683                 x_wells = pd.read_parquet(path_ar_wells)
1684                 well_chosen = value_returned[1]
1685                 sorted_dataframe = x_wells.loc[x_wells['
1686                     nWell'] == well_chosen, :].loc[times]
1686                 sorted_dataframe_small = sorted_dataframe[
1687                     property_chosen]
1687                 sorted_dataframe_selection =
1688                     sorted_dataframe_small.tolist()
1688             elif filename == 'TIME':
1689                 path_ar_time = os.path.join(path, 'TIME' +
1690                     '.parquet')

```

```

1690             x_time = pd.read_parquet(path_ar_time)
1691             df_prop = x_time[property_chosen]
1692             fetch_something = np.array(tlist) - 1
1693             t_to_plot = df_prop.iloc(axis=0) [
1694                 fetch_something].tolist()
1695             return t_to_plot
1696         elif filename == 'DATA':
1697             path_ar_data = os.path.join(path, 'DATA' +
1698               '.parquet')
1699             x_data = pd.read_parquet(path_ar_data)
1700             prop_final = []
1701             df_prop = x_data[property_chosen]
1702             maxcells = max(allcells)
1703             cells_unique = np.array(newcells) - 1
1704             timesteps = int(len(df_prop)/maxcells)
1705             for timestep in list(range(1,timesteps+1)):
1706                 top_pos = (timestep - 1)*maxcells
1707                 bot_pos = timestep*maxcells
1708                 current = df_prop[top_pos:bot_pos]
1709                 if timestep in tlist:
1710                     fetch_cells = cells_unique + (
1711                       timestep-1)*maxcells
1712                     prop_to_plot = df_prop.iloc(axis=0) [
1713                         fetch_cells].tolist()
1714                     prop_final = prop_final +
1715                     prop_to_plot
1716                     return prop_final
1717             elif filename == 'INPUT':
1718                 path_ar_input = os.path.join(path, 'INPUT' +
1719               '.parquet')
1720                 x_input = pd.read_parquet(path_ar_input)
1721                 ilist, jlist, klist, tlist = (time_element[2],
1722                   time_element[3], time_element[4], time_element[5])
1723                 newcells = x_input.loc[(x_input['i'].isin(
1724                   ilist)) & (x_input['j'].isin(jlist)) & (x_input['k'].isin(
1725                     klist)), 'Cell'].tolist()
1726                 if property_chosen == 'Cell':
1727                     return newcells
1728                 else:
1729                     df_prop = x_input[property_chosen]
1730                     fetch_cells = np.array(newcells) - 1
1731                     prop_to_plot = df_prop.iloc(axis=0) [
1732                         fetch_cells].tolist()
1733                     unique_values = np.unique(prop_to_plot)
1734                     return unique_values

```

```

1725         return sorted_dataframe, sorted_dataframe_small,
1726             sorted_dataframe_selection
1726
1727     def get_block_boundaries(self, cellvalues,
1728         cellwidths):
1728         cellvalues, cellwidths = (np.array(cellvalues),
1729             np.array(cellwidths))
1729         left_boundaries = (cellvalues - 0.5 * cellwidths
1730             ).tolist()
1730         right_boundaries = [cellvalues[-1] + 0.5 *
1731             cellwidths[-1]]
1731         boundaries = left_boundaries + right_boundaries
1732         return boundaries
1733
1734     def get_varying_block_boundaries(self, cellvalues):
1735         boundaries = []
1736         for item in cellvalues:
1737             left_boundary = item - item*0.5
1738             boundaries.append(left_boundary)
1739             if item == cellvalues[-1]:
1740                 right_boundary = item + item*0.5
1741                 boundaries.append(right_boundary)
1742         return boundaries
1743
1744     def get_block_centers(self, cellvalues, cellwidths):
1745         cellvalues, cellwidths = (np.array(cellvalues),
1746             np.array(cellwidths))
1746         left_boundaries = (cellvalues - 0.5 * cellwidths
1747             ).tolist()
1747         right_boundaries = [cellvalues[-1] + 0.5 *
1748             cellwidths[-1]]
1748         boundaries = left_boundaries + right_boundaries
1749         return boundaries
1750
1751     def get_time(self, core_path, property_chosen, tlist
1752     ):
1752         path_ar_time = os.path.join(core_path, 'TIME' +
1753             '.parquet')
1753         x_time = pd.read_parquet(path_ar_time)
1754         if property_chosen == 'Days':
1755             property_chosen = 'nDays'
1756             df = x_time[property_chosen]
1757             fetch_days = np.array(tlist) - 1
1758             values = df.iloc(axis=0)[fetch_days]
1759             return values

```

```

1760
1761     def get_input(self, core_path, property_chosen,
1762         ilist, jlist, klist):
1763         path_ar_input = os.path.join(core_path, 'INPUT'
1764             + '.parquet')
1765         x_input = pd.read_parquet(path_ar_input)
1766         allcells = x_input['Cell'].tolist()
1767         ijkcells = x_input.loc[(x_input['i'].isin(ilist)
1768             ) &
1769             (x_input['j'].isin(jlist)
1770             ) & (x_input['k'].isin(klist)), 'Cell'].tolist()
1771
1772         fetch_cells = np.array(ijkcells) - 1
1773         df = x_input[property_chosen]
1774         values = df.iloc(axis=0)[fetch_cells]
1775         return values
1776
1777
1778     def get_data(self, core_path, property_chosen,
1779         allcells, newcells, ilist, jlist, klist, tlist):
1780         values = []
1781         if property_chosen in ['Days', 'Timestep']:
1782             df = self.get_time(core_path,
1783                 property_chosen, tlist)
1784             values = df.values.tolist()
1785             return values
1786         elif property_chosen == 'Cell':
1787             df = self.get_input(core_path=core_path,
1788                 property_chosen=property_chosen, ilist=ilist, jlist=
1789                 jlist, klist=klist)
1790             values = df.values.tolist()
1791             return values
1792         elif property_chosen in ['X', 'Y', 'Z']:
1793             df = self.get_input(core_path=core_path,
1794                 property_chosen=property_chosen, ilist=ilist, jlist=
1795                 jlist, klist=klist)
1796             values = df.values.tolist()
1797             return values
1798         else:
1799             path_ar_data = os.path.join(core_path, 'DATA'
2000                 + '.parquet')
2001             x_data = pd.read_parquet(path_ar_data)
2002             z_final = []
2003             df_z = x_data[property_chosen]
2004             z_id = df_z.tolist()
2005             maxcells = max(allcells)

```

```

1794             cells_unique = np.array(newcells) - 1
1795             timesteps = int(len(z_id) / maxcells)
1796             for timestep in list(range(1, timesteps + 1)):
1797                 top_pos = (timestep - 1) * maxcells
1798                 bot_pos = timestep * maxcells
1799                 current = z_id[top_pos:bot_pos]
1800                 if timestep in tlist:
1801                     fetch_cells = cells_unique + (
1802                         timestep - 1) * maxcells
1803                     z_to_plot = df_z.iloc(axis=0)[
1804                         fetch_cells].tolist()
1805                     z_final = z_final + z_to_plot
1806                     values = z_final
1807             return values
1808
1809     def plot_graphv2(self):
1810         self.delete_figures(2)
1811         child, prev_simcase, local_dict, store_dict,
1812         chosen_parameters_rawtest = (self.pageone_listbox_plot,
1813             {}, {}, {}, [])
1814         alls = list(child.get(0, END))
1815         if self.hold.get() == 0 and self.xyz.get() != 0:
1816             simcase = current_selection
1817             corepath = global_sim_data[simcase][0]
1818             current_xyz = self.xyz.get() # 1,2,3,etc
1819             xnow, ynow, simcase_child = (None, None,
1820                 None)
1821             self.plot_id_old = self.plot_id
1822             xcoord = [[ 'X'], [ 'Y'], [ 'X'], [ 'Z'], [ 'Z'],
1823                 [ 'Y']]
1824             ycoord = [[ 'Y'], [ 'X'], [ 'Z'], [ 'X'], [ 'Y'],
1825                 [ 'Z']]
1826             if current_xyz in [7, 8]:
1827                 childx = self.x_listbox
1828                 childy = self.y_listbox
1829                 allsx = childx.get(0, END)
1830                 allsy = childy.get(0, END)
1831                 xnow = self.plot_x[allsx[0]]['entries']
1832                 ynow = self.plot_y[allsy[0]]['entries']
1833                 simcase_child_01, simcase_child_02 = (
1834                     None, None)
1835                 if current_xyz == 7: # Means X is days
1836                     simcase_child_01 = [ 'DATA', [ 'DATA' ]
1837                 ]

```

```

1829                     simcase_child_02 = ['INPUT', ['INPUT
1830                         ']]
1830                     elif current_xyz == 8: # Means Y is
1831                         days
1831                     simcase_child_01 = ['INPUT', ['INPUT
1832                         ']]
1832                     simcase_child_02 = ['DATA', ['DATA']]
1832                 ]
1833                     self.plot_id = {'simcase': simcase, '
1833             'simcase_path': corepath, 'simcase_child':
1833             simcase_child_01, 'entries': xnow, 'cells': None, 'time'
1833             : None, 'X': None, 'Y': None}
1834                     self.add_to_xy(typedata='X')
1835                     self.plot_id = {'simcase': simcase, '
1835             'simcase_path': corepath, 'simcase_child':
1835             simcase_child_02, 'entries': ynow, 'cells': None, 'time'
1835             : None, 'X': None, 'Y': None}
1836                     self.add_to_xy(typedata='Y')
1837                 else:
1838                     simcase_child = ['INPUT', ['INPUT']]
1839                     xnow, ynow = (xcoord[current_xyz - 1],
1839                         ycoord[current_xyz - 1])
1840                     self.plot_id = {'simcase': simcase, '
1840             'simcase_path': corepath, 'simcase_child': simcase_child,
1840             'entries': xnow, 'cells': None, 'time': None, 'X': None
1840             , 'Y': None}
1841                     self.add_to_xy(typedata='X')
1842                     self.plot_id = {'simcase': simcase, '
1842             'simcase_path': corepath, 'simcase_child': simcase_child,
1842             'entries': ynow, 'cells': None, 'time': None, 'X': None
1842             , 'Y': None}
1843                     self.add_to_xy(typedata='Y')
1844                     self.plot_id = self.plot_id_old
1845
1846                 if not alls:
1847                     plottings = list(self.plot_rdy.keys())
1848                     if plottings:
1849                         for prev_plotted_item in plottings:
1850                             oldcontent = self.plot_rdy[
1850                             prev_plotted_item][1]
1851                             current_path = oldcontent['
1851             'simcase_path']
1852                             cells, times = self.
1852             get_cell_time(corepath=current_path)
1853                             newvalues = {'cells': cells, '

```

```

1853     'time': times, 'X': self.plot_x, 'Y': self.plot_y}
1854                                         self.plot_rdy[prev_plotted_item]
1855                                         [1].update(newvalues)
1856                                         child.insert(END,
1857                                         prev_plotted_item)
1858                                         alls = list(child.get(0, END))
1859                                         self.figs = len(alls)
1860                                         self.grid_size_figures()
1861                                         self.label_figs['text'] = 'Figures
1862                                         : ' + str(self.figs)
1863                                         elif alls:
1864                                             for prev_plotted_item in alls:
1865                                                 # print('prev_plotted_item: ' + str(
1866                                                 prev_plotted_item))
1867                                                 oldcontent = self.plot_rdy[
1868                                                 prev_plotted_item][1]
1869                                                 core_path2 = oldcontent['
1870                                                 simcase_path']
1871                                                 # print('oldcontent: ' + str(
1872                                                 oldcontent))
1873                                                 marker = self.plot_rdy[
1874                                                 prev_plotted_item][2]
1875                                                 # print('marker: ' + str(marker))
1876                                                 if marker == 1:
1877                                                     current_path = oldcontent['
1878                                                     simcase_path']
1879                                                     cells2, time_element2 = self.
1880                                                     get_cell_time(corepath=current_path)
1881                                                     ilist2, jlist2, klist2, tlist2 =
1882                                                     (time_element2[2], time_element2[3], time_element2[4],
1883                                                     time_element2[5])
1884                                                     timedays2 = self.get_time(
1885                                                     core_path2, 'Days', tlist2).values.tolist()
1886                                                     data2 = self.plot_rdy[
1887                                                     prev_plotted_item][1]['title']
1888                                                     old_basetitle = data2['
1889                                                     shown_title'].split('=')[0]
1890                                                     new_basetitle = old_basetitle +
1891                                                     '=' + str(timedays2[0]) + ' days'
1892                                                     newvalues = {'shown_title':
1893                                                     new_basetitle, 'timedays': timedays2}
1894                                                     self.plot_rdy[prev_plotted_item]
1895                                                     [1]['title'].update(newvalues)
1896                                                     newvalues = {'cells': cells2, '
1897                                                     time': time_element2, 'X': self.plot_x, 'Y': self.plot_y

```

```

1878 }
1879             self.plot_rdy[prev_plotted_item]
1880             [1].update(newvalues)
1881         elif marker == 0:
1882             current_path = oldcontent['simcase_path']
1883             cells2, time_element2 = self.
1884             get_cell_time(corepath=current_path)
1885             newvalues = {'cellsX':
1886             ': self.plot_x, 'Y': self.plot_y}
1887             self.plot_rdy[prev_plotted_item]
1888             [1].update(newvalues)
1889
1890             share_axis, filename, aspect_wanted, aspect_auto
1891             = (False, '', 1, True)
1892             chosen_rows, chosen_cols = ([], [])
1893             global chosen_rows_alt, chosen_cols_alt
1894             if chosen_rows_alt is not None and
1895             chosen_cols_alt is not None:
1896                 chosen_rows, chosen_cols = (int(
1897                 chosen_rows_alt), int(chosen_cols_alt))
1898                 self.grid_button.config(text='Rows: ' +
1899                 chosen_rows_alt + ' Cols: ' + chosen_cols_alt)
1900                 chosen_rows_alt, chosen_cols_alt = (None,
1901 None)
1902             else:
1903                 dimensions = self.fig_grid_size[self.
1904                 grid_dropdown.get()]
1905                 chosen_rows, chosen_cols = dimensions
1906
1907                 tight_plot = True
1908                 sharex_local, sharey_local = (False, False)
1909                 fig, axes = (None, None)
1910                 change_plot = self.plottype.get() # Allow user
1911                 to change this
1912                 plot_version = None
1913                 if change_plot == 1:
1914                     plot_version = 1
1915                     if self.sharex.get() == 1:
1916                         sharex_local = True
1917                         if self.sharey.get() == 1:
1918                             sharey_local = True
1919                             fig, axes = plt.subplots(
1920                                 nrows=chosen_rows, ncols=chosen_cols,
1921                                 sharex=sharex_local, sharey=sharey_local, figsize=(10,

```

```

1909 10))
1910     elif change_plot == 0:
1911         plot_version = 0
1912         if self.sharex.get() == 1:
1913             sharex_local = 'all'
1914         if self.sharey.get() == 1:
1915             sharey_local = 'all'
1916         fig = Figure(figsize=(10, 10))
1917         fig_plotted = 0
1918         prev_ax = None
1919         for item in alls:
1920             xvalues, yvalues, zvalues = (None, None,
1921                                         None)
1922             raw = self.plot_rdy[item]
1923             data, identifier = (raw[1], raw[0])
1924             simcase_path = data['simcase_path']
1925             titledata = data['title']
1926             title = titledata['shown_title']
1927             new_xlabel = titledata['xlabel']
1928             new_ylabel = titledata['ylabel']
1929             fontsize = int(data['fontsize'])
1930             if self.hold3.get() == 0:
1931                 fontsize = self.fontsize
1932             matplotlib.rcParams.update({'font.size':
1933                                         fontsize})
1934             cells, time_element = (data['cells'], data['
1935             time'])
1936             if self.hold == 0:
1937                 cells, time_element = self.get_cell_time(
1938                     corepath=simcase_path)
1939                 times, days = (time_element[0].tolist(),
1940                               time_element[1].tolist())
1941                 xvalues, yvalues = (None, None)
1942                 x_property_chosen, y_property_chosen = (None
1943                                         , None)
1944                 x_path, y_path = (None, None)
1945                 test = None
1946                 xvalues_unique, yvalues_unique,
1947                 zvalues_unique = (None, None, None)
1948                 if data['X']:
1949                     x_key = list(data['X'].keys())[0]
1950                     x_key_prop = x_key.split(' ')[2]
1951                     x_coord_data = data['X'][x_key]
1952                     x_simcase_child = x_coord_data['
1953                     simcase_child'][1]

```

```

1946             x_property_chosen = x_coord_data['
1947             entries'][0]
1948             x_path = x_coord_data['simcase_path']
1949             x_simcase = x_coord_data['simcase']
1950             if x_simcase_child == 'DATA' and
1951                 x_key_prop == 'Cell':
1952                     x_simcase_child = ['INPUT']
1953                     x_filename = x_simcase_child[0]
1954                     xvalues_unique = self.fetch_data(path=
1955                         x_path, value_returned=x_simcase_child,
1956                         property_chosen=x_property_chosen,
1957                         cells=cells, time_element=
1958                         time_element)
1959                     if data['Y']:
1960                         y_key = list(data['Y'].keys())[0]
1961                         y_key_prop = y_key.split(' ')[2]
1962                         y_coord_data = data['Y'][y_key]
1963                         y_simcase_child = y_coord_data['
1964                         simcase_child'][1]
1965                         if y_simcase_child == 'DATA' and
1966                             y_key_prop == 'Cell':
1967                             y_simcase_child = ['TIME']
1968                             y_property_chosen = y_coord_data['
1969                             entries'][0]
1970                             y_path = y_coord_data['simcase_path']
1971                             y_simcase = y_coord_data['simcase']
1972                             y_filename = y_simcase_child[0]
1973                             yvalues_unique = self.fetch_data(path=
1974                                 y_path, value_returned=y_simcase_child,
1975                                 property_chosen=y_property_chosen,
1976                                 cells=cells, time_element=
1977                                 time_element)
1978                             keys = list(data.keys())
1979                             simcase_child = data['simcase_child']
1980                             filename = simcase_child[1][0]
1981                             z_property_chosen = data['entries']
1982                             simcase = data['simcase']
1983                             xlabel = 'Cell Numbering (unique)'
1984                             if new_xlabel:
1985                                 xlabel = new_xlabel
1986                             ylabel = 'Simulation runtime (days)'
1987                             if new_ylabel:
1988                                 ylabel = new_ylabel
1989
1990                             if filename == 'DATA':
1991                                 z_property_chosen = prep_pageone[simcase
1992                                     ][z_property_chosen]

```

```

1979             zvalues_unique = self.fetch_data(path=
    simcase_path, value_returned=simcase_child,
    property_chosen=z_property_chosen,
1980
    cells=cells, time_element=time_element)
1981
1982         ilist, jlist, klist, tlist = (time_element[2
    ], time_element[3], time_element[4], time_element[5])
1983         if self.hold == 0:
1984             imin, imax = (int(self.slidex_left.get()
    ), int(self.slidex_right.get()))
1985             jmin, jmax = (int(self.slidey_left.get()
    ), int(self.slidey_right.get()))
1986             kmin, kmax = (int(self.slidez_left.get()
    ), int(self.slidez_right.get()))
1987             tmin, tmax = (int(self.slidetime_left.
    get()), int(self.slidetime_right.get()))
1988             ilist = list(range(imin, imax + 1))
1989             jlist = list(range(jmin, jmax + 1))
1990             klist = list(range(kmin, kmax + 1))
1991             tlist = list(range(tmin, tmax + 1))
1992
1993         path_ar_input = os.path.join(simcase_path, 'INPUT' + '.parquet')
1994         x_input = pd.read_parquet(path_ar_input)
1995         newcells = x_input.loc[(x_input['i'].isin(
    ilist)) & (x_input['j'].isin(jlist)) & (x_input['k'].isin(
    klist)), 'Cell'].tolist()
1996         allcells = x_input['Cell']
1997         dims = x_input.loc[x_input['Cell'].isin(
    newcells), ['DX', 'DY', 'DZ']]
1998
1999         x_data = None
2000         x_id, y_id, z_id = (None, None, None)
2001         y_id_v2, y_id_v2_dates = ([], [])
2002         y_id_v3_days, y_id_v3_times = ([], [])
2003         for i in tlist:
2004             y_id_v2.append(i - 1)
2005
2006         y_id_final = []
2007         z_id_v2 = []
2008         z_final = []
2009         y_id_v2 = []
2010         x_id_v2 = []
2011         im = None

```

```

2012             dimi, dimj, dimk = (None, None, None)
2013             xdim, ydim, zdim = (None, None, None)
2014             dim_dx, dim_dy, dim_dz = ([], [], [])
2015             grid_xticks, grid_yticks, grid_zticks = (
2016                 None, None, None)
2017             rowshape, colshape = (None, None)
2018             rowtype, coltype = (None, None)
2019             x_grid, y_grid, z_grid = (None, None, None)
2020
2021             plot_type = None
2022             ilen, jlen, klen, tlen = (len(ilist), len(
2023                 jlist), len(klist), len(tlist))
2024             if ilen == 1 and jlen == 1 and klen == 1 and
2025                 tlen >= 1:
2026                 plot_type = 'time'
2027                 # elif tlen==1 and ((ilen!=1 and jlen==1 and
2028                     klen==1) or (ilen==1 and jlen!=1 and klen==1) or (ilen
2029                     ==1 and jlen==1 and klen!=1)):
2030                 #         plot_type = 'position'
2031             else:
2032                 plot_type = '2d'
2033             ax = None
2034             if filename == 'DATA' and plot_type == 'time
2035                 ':
2036                 core_path = simcase_path
2037                 y_id_v2 = self.get_data(core_path=
2038                     core_path, property_chosen='Days', allcells=allcells,
2039                     newcells=newcells, ilist=ilist, jlist=jlist, klist=klist
2040                     , tlist=tlist)
2041                 z_id_v2 = self.get_data(core_path=
2042                     core_path, property_chosen=z_property_chosen, allcells=
2043                     allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2044                     klist=klist, tlist=tlist)
2045
2046                 row_id, col_id = self.find_row_col(
2047                     identifier, chosen_rows, chosen_cols)
2048                 ax = None
2049                 if plot_version == 0:
2050                     if not prev_ax:
2051                         ax = fig.add_subplot(chosen_rows
2052                             , chosen_cols, identifier)
2053                     else:
2054                         if sharex_local == 'all' and
2055                             sharey_local != 'all':
2056                             ax = fig.add_subplot(

```

```

2041 chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2042                               elif sharex_local != 'all' and
2043                               sharey_local == 'all':
2044                               ax = fig.add_subplot(
2045                               chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2046                               elif sharex_local == 'all' and
2047                               sharey_local == 'all':
2048                               ax = fig.add_subplot(
2049                               chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2050                               sharey=prev_ax)
2051                               else:
2052                               ax = fig.add_subplot(
2053                               chosen_rows, chosen_cols, identifier)
2054                               elif plot_version == 1:
2055                               if (row_id, col_id) == (None, None):
2056                                   break
2057                               if chosen_rows == 1 and chosen_cols
2058                               == 1: # Only plot one figure
2059                               ax = axes
2060                               elif chosen_rows == 1 and
2061                               chosen_cols != 1: # Only plot against col_id
2062                               ax = axes[col_id]
2063                               tight_plot = True
2064                               elif chosen_rows != 1 and
2065                               chosen_cols == 1: # Only plot against row_id
2066                               ax = axes[row_id]
2067                               elif chosen_rows != 1 and
2068                               chosen_cols != 1: # Use both row_id and col_id
2069                               ax = axes[row_id, col_id]
2070                               title = z_property_chosen
2071                               ax.set_title(title)
2072                               xlabel, ylabel = (y_property_chosen,
2073                               z_property_chosen)
2074                               if new_xlabel:
2075                                   xlabel = new_xlabel
2076                               if new_ylabel:
2077                                   ylabel = new_ylabel
2078                               ax.set_xlabel(xlabel)
2079                               ax.set_ylabel(ylabel)
2080                               ax.plot(y_id_v2, z_id_v2)
2081                               elif filename == 'DATA' and plot_type == 'position':
2082                                   core_path = simcase_path
2083                                   if ilen > 1:

```

```

2074                     x_property_chosen = 'X'
2075             elif jlen > 1:
2076                 x_property_chosen = 'Y'
2077             elif klen > 1:
2078                 x_property_chosen = 'Z'
2079             dims = self.get_input(core_path=
2080                             core_path, property_chosen=x_property_chosen,
2081                                         ilist=ilist, jlist=jlist,
2082                                         klist=klist)
2081             x_id_v2 = dims.tolist()
2082             z_id_v2 = self.get_data(core_path=
2083                             core_path, property_chosen=z_property_chosen, allcells=
2084                             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2085                             klist=klist, tlist=tlist)
2083
2084             row_id, col_id = self.find_row_col(
2085                 identifier, chosen_rows, chosen_cols)
2085             ax = None
2086             if plot_version == 0:
2087                 if not prev_ax:
2088                     ax = fig.add_subplot(chosen_rows
2089                         , chosen_cols, identifier)
2089             else:
2090                 if sharex_local == 'all' and
2091                     sharey_local != 'all':
2091                     ax = fig.add_subplot(
2092                         chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2092             elif sharex_local != 'all' and
2093                     sharey_local == 'all':
2093                     ax = fig.add_subplot(
2094                         chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2094             elif sharex_local == 'all' and
2095                     sharey_local == 'all':
2095                     ax = fig.add_subplot(
2096                         chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2097                                         sharey=prev_ax)
2096             else:
2097                 ax = fig.add_subplot(
2098                     chosen_rows, chosen_cols, identifier)
2098             elif plot_version == 1:
2099                 if (row_id, col_id) == (None, None):
2100                     break
2101                 if chosen_rows == 1 and chosen_cols
2102                     == 1: # Only plot one figure
2102                     ax = axes

```

```

2103                     elif chosen_rows == 1 and
2104             chosen_cols != 1: # Only plot against col_id
2105                 ax = axes[col_id]
2106                 tight_plot = True
2107             elif chosen_rows != 1 and
2108                 chosen_cols == 1: # Only plot against row_id
2109                     ax = axes[row_id]
2110             elif chosen_rows != 1 and
2111                 chosen_cols != 1: # Use both row_id and col_id
2112                     ax = axes[row_id, col_id]
2113                     ax.set_title(title)
2114                     xlabel, ylabel = (x_property_chosen,
2115                                         z_property_chosen)
2116                     if new_xlabel:
2117                         xlabel = new_xlabel
2118                     if new_ylabel:
2119                         ylabel = new_ylabel
2120                     ax.set_xlabel(xlabel)
2121                     ax.set_ylabel(ylabel)
2122                     ax.plot(x_id_v2, z_id_v2)
2123             elif filename == 'DATA' and plot_type == '2d'
2124     ' and ((x_property_chosen == 'Cell' and
2125         y_property_chosen == 'Days') or (x_property_chosen == 'Days' and
2126         y_property_chosen == 'Cell')):
2127         core_path = simcase_path
2128
2129         x_id_v2_alt = self.get_data(core_path=
2130             core_path, property_chosen=x_property_chosen, allcells=
2131             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2132             klist=klist, tlist=tlist)
2133         y_id_v2_alt = self.get_data(core_path=
2134             core_path, property_chosen=y_property_chosen, allcells=
2135             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2136             klist=klist, tlist=tlist)
2137         z_id_v2_alt = self.get_data(core_path=
2138             core_path, property_chosen=z_property_chosen, allcells=
2139             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2140             klist=klist, tlist=tlist)
2141
2142         xi, yj = (x_id_v2_alt, y_id_v2_alt)
2143         dxi, dyj = ([1.0]*len(xi), [1.0]*len(yj))
2144     )
2145
2146         xnew = self.get_block_boundaries(
2147             cellvalues=xi, cellwidths=dxi)

```

```

2130                 ynew = self.get_block_boundaries(
2131                     cellvalues=yj, cellwidths=dyj)
2132
2133                 z_id_v2 = self.get_data(core_path=
2134                     core_path, property_chosen=z_property_chosen, allcells=
2135                     allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2136                     klist=klist, tlist=tlist)
2137
2138                 x_centers = xi
2139                 x_bound = xnew
2140                 y_centers = yj
2141                 y_bound = ynew
2142
2143                 x_id_v2, y_id_v2 = (x_id_v2_alt,
2144                     y_id_v2_alt)
2145                     xlenth, ylength = (len(x_id_v2), len(
2146                         y_id_v2))
2147
2148                     transpose_choice = None
2149                     if xlenth >= ylength:
2150                         rowshape, colshape = (ylength,
2151                             xlenth)
2152
2153                     transpose_choice = 0
2154                     elif xlenth < ylength:
2155                         rowshape, colshape = (xlenth,
2156                             ylength)
2157
2158                     transpose_choice = 1
2159
2160
2161                 x_grid, y_grid = np.meshgrid(x_id_v2,
2162                     y_id_v2)
2163
2164                 z_grid = np.reshape(np.array(z_id_v2), (
2165                     rowshape, colshape))
2166
2167
2168                 fig_plotted, ax = (fig_plotted + 1, None
2169 )
2170
2171                 row_id, col_id = self.find_row_col(
2172                     identifier, chosen_rows, chosen_cols)
2173
2174                 ax = None
2175                 if plot_version == 0:
2176                     if not prev_ax:
2177                         ax = fig.add_subplot(chosen_rows
2178                             , chosen_cols, identifier)
2179                     else:
2180                         if sharex_local == 'all' and
2181                             sharey_local != 'all':
2182                             ax = fig.add_subplot(
2183                                 chosen_rows, chosen_cols, identifier, sharex=prev_ax)

```

```

2160                               elif sharex_local != 'all' and
2161                                 sharey_local == 'all':
2162                                   ax = fig.add_subplot(
2163                                     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2162                               elif sharex_local == 'all' and
2163                                 sharey_local == 'all':
2164                                   ax = fig.add_subplot(
2165                                     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2166                                     sharey=prev_ax)
2164                           else:
2165                             ax = fig.add_subplot(
2166                               chosen_rows, chosen_cols, identifier)
2166                           elif plot_version == 1:
2167                             if (row_id, col_id) == (None, None):
2168                               break
2169                             if chosen_rows == 1 and chosen_cols
2170                               == 1: # Only plot one figure
2171                               ax = axes
2171                           elif chosen_rows == 1 and
2172                             chosen_cols != 1: # Only plot against col_id
2173                               ax = axes[col_id]
2173                               tight_plot = True
2174                           elif chosen_rows != 1 and
2175                             chosen_cols == 1: # Only plot against row_id
2176                               ax = axes[row_id]
2176                           elif chosen_rows != 1 and
2177                             chosen_cols != 1: # Use both row_id and col_id
2178                               ax = axes[row_id, col_id]
2179                               ax.set_title(title)
2180                               xlabel, ylabel = (x_property_chosen,
2181                                 y_property_chosen)
2181                               if new_xlabel:
2182                                 xlabel = new_xlabel
2183                               if new_ylabel:
2184                                 ylabel = new_ylabel
2185                               ax.set_xlabel(xlabel)
2186                               ax.set_ylabel(ylabel)
2187                               ax.xaxis.set_tick_params(which='both',
2188                                 labelbottom=True)
2188                               ax.yaxis.set_tick_params(which='both',
2189                                 labelbottom=True)
2189                               norm = clr.Normalize()
2190                               cmap = cm.get_cmap('gist_rainbow')
2191

```

```

2192                 if transpose_choice == 0:
2193                     im = ax.pcolormesh(x_grid, y_grid,
2194                                         z_grid, cmap=cmap, norm=norm)
2194                 elif transpose_choice == 1:
2195                     im = ax.pcolormesh(x_grid, y_grid,
2196                                         z_grid.T, cmap=cmap, norm=norm)
2196
2197                 print('...')
2198                 print(str(x_property_chosen) + str(
2199                     y_property_chosen) + ': transposed?? ' + str(
2200                     transpose_choice))
2200                 print('xlength: ' + str(len(x_id_v2)) +
2201                     ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
2202                     len(z_id_v2)))
2200                 print(' (rowshape, colshape): (' + str(
2201                     rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
2202                     '[' + str(coltype) + '])')
2201                 print('xshape: ' + str(x_grid.shape) +
2202                     ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
2203                     z_grid.shape))
2202                 print('...')
2203
2204                 if grid_xticks:
2205                     ax.set_xticks(grid_xticks)
2206                 if grid_yticks:
2207                     ax.set_yticks(grid_yticks)
2208
2209                 if aspect_auto is False:
2210                     aspect_ratio_wanted = aspect_wanted
2211                     aspect_ratio_correct = abs((x_max -
2212                         x_min) / (y_max - y_min)) / aspect_ratio_wanted
2212                     ax.set_aspect(aspect_ratio_correct)
2213
2214                 fig.colorbar(im, ax=ax)
2215
2216                 if tight_plot is True and aspect_auto is
2217                     True and plot_version == 1:
2218                         plt.tight_layout()
2218                 elif filename == 'DATA' and plot_type == '2d'
2219                     and ((x_property_chosen in ['X', 'Y', 'Z'] and
2220                         y_property_chosen in ['Days']) or (x_property_chosen in
2221                             ['Days'] and y_property_chosen in ['X', 'Y', 'Z'])):
2219                     core_path = simcase_path
2220                     datatypes = {'X': 'DX', 'Y': 'DY', 'Z':
2221                         'DZ'}

```

```

2221             dimi, dimj, dims, xi, yj, dx, dy = (
2222                 None, None, None, None, None, None)
2223
2223         x_id_v2_alt = self.get_data(core_path=
2224             core_path, property_chosen=x_property_chosen, allcells=
2225             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2226             klist=klist, tlist=tlist)
2227         y_id_v2_alt = self.get_data(core_path=
2228             core_path, property_chosen=y_property_chosen, allcells=
2229             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2230             klist=klist, tlist=tlist)
2231         z_id_v2_alt = self.get_data(core_path=
2232             core_path, property_chosen=z_property_chosen, allcells=
2233             allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2234             klist=klist, tlist=tlist)
2235
2236         xi, yj = (x_id_v2_alt, y_id_v2_alt)
2237
2238         if x_property_chosen in ['Days']:
2239             dimj = datatypes[y_property_chosen]
2240             dims = self.get_input(core_path=
2241                 core_path, property_chosen=[dimj], ilist=ilist, jlist=jlist,
2242                 klist=klist)
2243             dx, dy = ([1.0] * len(xi), list(
2244                 dims[dimj]))
2245
2246             elif y_property_chosen in ['Days']:
2247                 dimi = datatypes[x_property_chosen]
2248                 dims = self.get_input(core_path=
2249                     core_path, property_chosen=[dimi], ilist=ilist, jlist=jlist,
2250                     klist=klist)
2251                 dx, dy = (list(dims[dimi]), [1.0]
2252                     * len(yj))
2253
2254
2255         xnew = self.get_block_boundaries(
2256             cellvalues=xi, cellwidths=dx)
2257         ynew = self.get_block_boundaries(
2258             cellvalues=yj, cellwidths=dy)
2259
2260
2261         x_centers, x_bound, y_centers, y_bound =
2262             (None, None, None, None)
2263
2263         if x_property_chosen in ['Days']:
2264             x_centers = xi
2265             x_bound = xnew
2266             y_centers = (np.unique(yj)).tolist()
2267             y_bound = (np.unique(ynew)).tolist()

```

```

2247             elif y_property_chosen in ['Days']:
2248                 x_centers = (np.unique(xi)).tolist()
2249                 x_bound = (np.unique(xnew)).tolist()
2250                 y_centers = yj
2251                 y_bound = ynew
2252
2253                 print('x_centers len: ' + str(len(
2254                     x_centers)) + ' x_centers: ' + str(x_centers))
2255                 print('y_centers len: ' + str(len(
2256                     y_centers)) + ' y_centers: ' + str(y_centers))
2257                 print('x_bound len: ' + str(len(x_bound))
2258                     ) + ' x_bound: ' + str(x_bound))
2259                 print('y_bound len: ' + str(len(y_bound))
2260                     ) + ' y_bound: ' + str(y_bound))
2261
2262                 x_id_v2, y_id_v2, z_id_v2 = (x_bound,
2263                     y_bound, z_id_v2_alt)
2264                 xlenth, ylenth = (len(x_id_v2), len(
2265                     y_id_v2))
2266                 transpose_choice = None
2267                 if xlenth > ylenth:
2268                     rowshape, colshape = (ylenth - 1,
2269                         xlenth - 1)
2270                     transpose_choice = 0
2271                 elif xlenth < ylenth:
2272                     rowshape, colshape = (xlenth - 1,
2273                         ylenth - 1)
2274                     transpose_choice = 1
2275
2276                 elif xlenth == ylenth:
2277                     if (x_property_chosen == 'X' and
2278                         y_property_chosen == 'Days') or (x_property_chosen == 'Y'
2279                             and y_property_chosen == 'Days') or (x_property_chosen
2280                             == 'Z' and y_property_chosen == 'Days'):
2281                         rowshape, colshape = (ylenth -
2282                             1, xlenth - 1)
2283                         transpose_choice = 0
2284                     elif (x_property_chosen == 'Days'
2285                         and y_property_chosen == 'X') or (x_property_chosen == 'Days'
2286                             and y_property_chosen == 'Y') or (
2287                             x_property_chosen == 'Days' and y_property_chosen == 'Z'
2288                         ):
2289                         rowshape, colshape = (xlenth -
2290                             1, ylenth - 1)
2291                         transpose_choice = 1
2292
2293
2294

```

```

2275                 x_grid, y_grid = np.meshgrid(x_id_v2,
2276                                         y_id_v2)
2276                 z_grid = np.reshape(np.array(z_id_v2), (
2277                                             rowshape, colshape))
2277
2278                 fig_plotted, ax = (fig_plotted + 1, None)
2279
2280                 row_id, col_id = self.find_row_col(
2281                     identifier, chosen_rows, chosen_cols)
2282                 title = z_property_chosen
2283                 if plot_version == 0:
2284                     subplot_id = str(chosen_rows) + str(
2285                         chosen_cols) + str(identifier)
2286                     if not prev_ax:
2287                         ax = fig.add_subplot(chosen_rows
2288 , chosen_cols, identifier)
2289                     else:
2290                         if sharex_local == 'all' and
2291                             sharey_local != 'all':
2292                             ax = fig.add_subplot(
2293                                 chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2293                             elif sharex_local != 'all' and
2294                                 sharey_local == 'all':
2295                             ax = fig.add_subplot(
2296                                 chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2296                             elif sharex_local == 'all' and
2297                                 sharey_local == 'all':
2298                             ax = fig.add_subplot(
2299                                 chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2300                                 sharey=prev_ax)
2300                             else:
2301                             ax = fig.add_subplot(
2302                                 chosen_rows, chosen_cols, identifier)
2302                             elif plot_version == 1:
2303                                 if (row_id, col_id) == (None, None):
2304                                     break
2305                                 if chosen_rows == 1 and chosen_cols
2306 == 1: # Only plot one figure
2307                                     ax = axes
2308                                     elif chosen_rows == 1 and
2309                                         chosen_cols != 1: # Only plot against col_id
2310                                         ax = axes[col_id]
2311                                         tight_plot = True
2312                                         elif chosen_rows != 1 and
2313                                         chosen_cols == 1: # Only plot against row_id

```

```

2303                     ax = axes[row_id]
2304             elif chosen_rows != 1 and
2305                 chosen_cols != 1: # Use both row_id and col_id
2306                     ax = axes[row_id, col_id]
2307
2308                     ax.set_title(title)
2309                     xlabel, ylabel = (x_property_chosen,
2310                                         y_property_chosen)
2311                     if new_xlabel:
2312                         xlabel = new_xlabel
2313                     if new_ylabel:
2314                         ylabel = new_ylabel
2315                     ax.set_xlabel(xlabel)
2316                     ax.set_ylabel(ylabel)
2317                     ax.xaxis.set_tick_params(which='both',
2318                                         labelbottom=True)
2319                     ax.yaxis.set_tick_params(which='both',
2320                                         labelbottom=True)
2321                     norm = clr.Normalize()
2322                     cmap = cm.get_cmap('gist_rainbow')
2323
2324                     if transpose_choice == 0:
2325                         im = ax.pcolormesh(x_grid, y_grid,
2326                                         z_grid, cmap=cmap, norm=norm)
2327                     elif transpose_choice == 1:
2328                         im = ax.pcolormesh(x_grid, y_grid,
2329                                         z_grid.T, cmap=cmap, norm=norm)
2330
2331                     print('...')
2332                     print(str(x_property_chosen) + str(
2333                         y_property_chosen) + ': transposed?? ' + str(
2334                         transpose_choice))
2335                     print('xlength: ' + str(len(x_id_v2)) +
2336                         ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
2337                             len(z_id_v2)))
2338                     print('(rowshape, colshape): (' + str(
2339                         rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
2340                         '[' + str(coltype) + '])')
2341                     print('xshape: ' + str(x_grid.shape) +
2342                         ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
2343                             z_grid.shape))
2344                     print('...')
2345
2346                     if grid_xticks:
2347                         ax.set_xticks(grid_xticks)

```

```

2334             if grid_yticks:
2335                 ax.set_yticks(grid_yticks)
2336
2337             if aspect_auto is False:
2338                 aspect_ratio_wanted = aspect_wanted
2339                 aspect_ratio_correct = abs((x_max -
2340                     x_min) / (y_max - y_min)) / aspect_ratio_wanted
2341                 ax.set_aspect(aspect_ratio_correct)
2342
2343             fig.colorbar(im, ax=ax)
2344
2345
2346             #if tight_plot is True and aspect_auto
2347             #is True and plot_version == 1:
2348             #    plt.tight_layout()
2349
2350         elif filename == 'DATA' and plot_type == '2d
2351             ' and x_property_chosen in ['X','Y','Z'] and
2352             y_property_chosen in ['X','Y','Z']:
2353             core_path = simcase_path
2354             datatypes = {'X': 'DX', 'Y': 'DY', 'Z':
2355             'DZ'}
2356             dimi, dimj = (datatypes[
2357                 x_property_chosen], datatypes[y_property_chosen])
2358             dims = self.get_input(core_path=
2359                 core_path, property_chosen=[x_property_chosen,
2360                 y_property_chosen, dimi, dimj],
2361                 ilist=ilist, jlist=jlist
2362                 =jlist, klist=klist)
2363             xi, yj, dxi, dyj = (list(dims[
2364                 x_property_chosen]), list(dims[y_property_chosen]), list(
2365                 dims[dimi]), list(dims[dimj]))
2366
2367             xnew = self.get_block_boundaries(
2368                 cellvalues=xi, cellwidths=dxi)
2369             ynew = self.get_block_boundaries(
2370                 cellvalues=yj, cellwidths=dyj)
2371             z_id_v2 = self.get_data(core_path=
2372                 core_path, property_chosen=z_property_chosen, allcells=
2373                 allcells, newcells=newcells, ilist=ilist, jlist=jlist,
2374                 klist=klist, tlist=tlist)
2375
2376             x_centers = (np.unique(xi)).tolist()
2377             x_bound = (np.unique(xnew)).tolist()
2378             y_centers = (np.unique(yj)).tolist()

```

```

2363             y_bound = (np.unique(ynew)).tolist()
2364
2365             x_id_v2, y_id_v2 = (x_bound, y_bound)
2366             xlenth, ylenth = (len(x_id_v2), len(
2367                 y_id_v2))
2368             transpose_choice = None
2369             if xlenth > ylenth:
2370                 rowshape, colshape = (ylenth - 1,
2371                     xlenth - 1)
2372                 transpose_choice = 0
2373             elif xlenth < ylenth:
2374                 rowshape, colshape = (xlenth - 1,
2375                     ylenth - 1)
2376                 transpose_choice = 1
2377             elif xlenth == ylenth:
2378                 if (x_property_chosen == 'X' and
2379                     y_property_chosen == 'Y') or (x_property_chosen == 'X'
2380                     and y_property_chosen == 'Z') or (x_property_chosen == 'Y'
2381                     and y_property_chosen == 'Z'):
2382                     rowshape, colshape = (ylenth -
2383                         1, xlenth - 1)
2384                     transpose_choice = 0
2385                 elif (x_property_chosen == 'Y' and
2386                     y_property_chosen == 'X') or (x_property_chosen == 'Z'
2387                     and y_property_chosen == 'X') or (x_property_chosen == 'Z'
2388                     and y_property_chosen == 'Y'):
2389                     rowshape, colshape = (xlenth -
2390                         1, ylenth - 1)
2391                     transpose_choice = 1
2392
2393             x_grid, y_grid = np.meshgrid(x_id_v2,
2394                 y_id_v2)
2395             z_grid = np.reshape(np.array(z_id_v2), (
2396                 rowshape, colshape))
2397
2398             fig_plotted, ax = (fig_plotted + 1, None
2399             )
2400             row_id, col_id = self.find_row_col(
2401                 identifier, chosen_rows, chosen_cols)
2402
2403             if plot_version == 0:
2404                 subplot_id = str(chosen_rows) + str(
2405                     chosen_cols) + str(identifier)
2406                 if not prev_ax:
2407                     ax = fig.add_subplot(chosen_rows

```

```

2391 , chosen_cols, identifier)
2392             else:
2393                 if sharex_local == 'all' and
2394                     sharey_local != 'all':
2395                         ax = fig.add_subplot(
2396                             chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2395                 elif sharex_local != 'all' and
2396                     sharey_local == 'all':
2397                         ax = fig.add_subplot(
2398                             chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2397                 elif sharex_local == 'all' and
2398                     sharey_local == 'all':
2399                         ax = fig.add_subplot(
2400                             chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2400                             sharey=prev_ax)
2400             else:
2401                 ax = fig.add_subplot(
2402                     chosen_rows, chosen_cols, identifier)
2401             elif plot_version == 1:
2402                 if (row_id, col_id) == (None, None):
2403                     break
2404                 if chosen_rows == 1 and chosen_cols
2404 == 1: # Only plot one figure
2405                     ax = axes
2406                 elif chosen_rows == 1 and
2407                     chosen_cols != 1: # Only plot against col_id
2407                     ax = axes[col_id]
2408                     tight_plot = True
2409                 elif chosen_rows != 1 and
2410                     chosen_cols == 1: # Only plot against row_id
2410                     ax = axes[row_id]
2411                 elif chosen_rows != 1 and
2412                     chosen_cols != 1: # Use both row_id and col_id
2412                     ax = axes[row_id, col_id]
2413
2414                 ax.set_title(title)
2415                 xlabel, ylabel = (x_property_chosen,
2415 y_property_chosen)
2416                 if new_xlabel:
2417                     xlabel = new_xlabel
2418                 if new_ylabel:
2419                     ylabel = new_ylabel
2420                 ax.set_xlabel(xlabel)
2421                 ax.set_ylabel(ylabel)
2422                 ax.xaxis.set_tick_params(which='both',

```

```

2422 labelbottom=True)
2423         ax.xaxis.set_tick_params(which='both',
2424             labelbottom=True)
2425             norm = clr.Normalize()
2426             cmap = cm.get_cmap('gist_rainbow')
2427
2428             if transpose_choice == 0:
2429                 im = ax.pcolormesh(x_grid, y_grid,
2430                     z_grid, cmap=cmap, norm=norm)
2431             elif transpose_choice == 1:
2432                 im = ax.pcolormesh(x_grid, y_grid,
2433                     z_grid.T, cmap=cmap, norm=norm)
2434
2435             print('...')
2436             print(str(x_property_chosen) + str(
2437                 y_property_chosen) + ': transposed?? ' + str(
2438                 transpose_choice))
2439             print('xlength: ' + str(len(x_id_v2)) +
2440                 ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
2441                 len(z_id_v2)))
2442             print('(rowshape, colshape): (' + str(
2443                 rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
2444                 '[' + str(coltype) + '])')
2445             print('xshape: ' + str(x_grid.shape) +
2446                 ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
2447                 z_grid.shape))
2448             print('...')
2449
2450             if grid_xticks:
2451                 ax.set_xticks(grid_xticks)
2452             if grid_yticks:
2453                 ax.set_yticks(grid_yticks)
2454
2455             if aspect_auto is False:
2456                 aspect_ratio_wanted = aspect_wanted
2457                 aspect_ratio_correct = abs((x_max -
2458                     x_min) / (y_max - y_min)) / aspect_ratio_wanted
2459                 ax.set_aspect(aspect_ratio_correct)
2460
2461             fig.colorbar(im, ax=ax)
2462
2463             plt.tight_layout()
2464
2465             #if tight_plot is True and aspect_auto
2466             is True and plot_version == 1:

```

```

2454                 #     plt.tight_layout()
2455
2456         elif filename == 'INPUT' and plot_type == '2d' and x_property_chosen in ['X', 'Y', 'Z'] and
2457             y_property_chosen in ['X', 'Y', 'Z']:
2458                 core_path = simcase_path
2459                 datatypes = {'X': 'DX', 'Y': 'DY', 'Z': 'DZ'}
2460
2461                 dimi, dimj = (datatypes[
2462                     x_property_chosen], datatypes[y_property_chosen])
2463                 dims = self.get_input(core_path=
2464                     core_path, property_chosen=[x_property_chosen,
2465                     y_property_chosen, z_property_chosen, dimi, dimj],
2466                     ilist=ilist, jlist=jlist,
2467                     klist=klist)
2468
2469                 xi, yj, dxi, dyj = (list(dims[
2470                     x_property_chosen]), list(dims[y_property_chosen]), list(
2471                     dims[dimi]), list(dims[dimj]))
2472
2473                 listitems = [xi, yj]
2474                 for litem in listitems:
2475                     litem = [float(i) for i in litem]
2476
2477
2478                 xnew = self.get_block_boundaries(
2479                     cellvalues=xi, cellwidths=dxi)
2480
2481                 ynew = self.get_block_boundaries(
2482                     cellvalues=yj, cellwidths=dyj)
2483
2484                 z_id_v2 = [float(i) for i in list(dims[
2485                     z_property_chosen])]

2486
2487                 x_centers = list(np.unique(xi))
2488                 x_bound = list(np.unique(xnew))
2489                 y_centers = list(np.unique(yj))
2490                 y_bound = list(np.unique(ynew))

2491
2492                 x_id_v2, y_id_v2 = (x_bound, y_bound)
2493                 xlengt, ylength = (len(x_id_v2), len(
2494                     y_id_v2))
2495
2496                 transpose_choice = None
2497
2498                 if xlengt >= ylength:
2499                     rowshape, colshape = (ylength - 1,
2500                         xlengt - 1)
2501
2502                     transpose_choice = 0
2503
2504                 elif xlengt < ylength:
2505                     rowshape, colshape = (xlengt - 1,
2506                         ylength - 1)
2507
2508                     transpose_choice = 1

```

```

2484
2485             x_grid, y_grid = np.meshgrid(x_id_v2,
2486                                         y_id_v2)
2486             z_grid = np.reshape(np.array(z_id_v2), (
2487                                         rowshape, colshape))
2487
2488             fig_plotted, ax = (fig_plotted + 1, None
2489 )
2489             row_id, col_id = self.find_row_col(
2490                 identifier, chosen_rows, chosen_cols)
2490             ax = None
2491             if plot_version == 0:
2492                 if not prev_ax:
2493                     ax = fig.add_subplot(chosen_rows
2493 , chosen_cols, identifier)
2494             else:
2495                 if sharex_local == 'all' and
2495 sharey_local != 'all':
2496                     ax = fig.add_subplot(
2496 chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2497                 elif sharex_local != 'all' and
2497 sharey_local == 'all':
2498                     ax = fig.add_subplot(
2498 chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2499                 elif sharex_local == 'all' and
2499 sharey_local == 'all':
2500                     ax = fig.add_subplot(
2500 chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2500 sharey=prev_ax)
2501             else:
2502                 ax = fig.add_subplot(
2502 chosen_rows, chosen_cols, identifier)
2503             elif plot_version == 1:
2504                 if (row_id, col_id) == (None, None):
2505                     break
2506                 if chosen_rows == 1 and chosen_cols
2506 == 1: # Only plot one figure
2507                     ax = axes
2508                 elif chosen_rows == 1 and
2508 chosen_cols != 1: # Only plot against col_id
2509                     ax = axes[col_id]
2510                     tight_plot = True
2511                 elif chosen_rows != 1 and
2511 chosen_cols == 1: # Only plot against row_id
2512                     ax = axes[row_id]

```

```

2513                     elif chosen_rows != 1 and
2514             chosen_cols != 1: # Use both row_id and col_id
2515                 ax = axes[row_id, col_id]
2515
2516                 ax.set_title(title)
2517                 xlabel, ylabel = (x_property_chosen,
2518                                     y_property_chosen)
2518                     if new_xlabel:
2519                         xlabel = new_xlabel
2520                     if new_ylabel:
2521                         ylabel = new_ylabel
2522                         ax.set_xlabel(xlabel)
2523                         ax.set_ylabel(ylabel)
2524                         ax.xaxis.set_tick_params(which='both',
2525                                     labelbottom=True)
2525                         ax.yaxis.set_tick_params(which='both',
2526                                     labelbottom=True)
2526                         norm = clr.Normalize()
2527                         cmap = cm.get_cmap('gist_rainbow')
2528
2529                         print('...')
2530                         print(str(x_property_chosen) + str(
2531                             y_property_chosen) + ': transposed?? ' +
2532                             transpose_choice))
2531                         print('xlength: ' + str(len(x_id_v2)) +
2532 ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
2533                             len(z_id_v2)))
2532                         print('(rowshape, colshape): (' +
2533                             str(rowshape) + '[' + str(rowtype) + '], ' +
2534                             str(colshape) + '[' + str(coltype) + '])')
2533                         print('xshape: ' + str(x_grid.shape) +
2534 ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
2535                             z_grid.shape))
2534                         print('...')
2535
2536                     if transpose_choice == 0:
2537                         im = ax.pcolormesh(x_grid, y_grid,
2538                             z_grid, cmap=cmap, norm=norm)
2538                     elif transpose_choice == 1:
2539                         im = ax.pcolormesh(x_grid, y_grid,
2540                             z_grid.T, cmap=cmap, norm=norm)
2541                         print('...')
2542                         print(str(x_property_chosen) + str(
2543                             y_property_chosen) + ': transposed?? ' +
2544                             transpose_choice))

```

```

2542 transpose_choice))
2543             print('xlength: ' + str(len(x_id_v2)) +
2544         ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
2545             len(z_id_v2)))
2546             print(' (rowshape, colshape): (' + str(
2547                 rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
2548                 '[' + str(coltype) + '])')
2549             print('xshape: ' + str(x_grid.shape) +
2550         ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
2551             z_grid.shape))
2552             print('...')

2553     if grid_xticks:
2554         ax.set_xticks(grid_xticks)
2555     if grid_yticks:
2556         ax.set_yticks(grid_yticks)

2557     if aspect_auto is False:
2558         aspect_ratio_wanted = aspect_wanted
2559         aspect_ratio_correct = abs((x_max -
2560             x_min) / (y_max - y_min)) / aspect_ratio_wanted
2561         ax.set_aspect(aspect_ratio_correct)

2562         #if tight_plot is True and aspect_auto
2563         #is True and plot_version == 1:
2564         #    plt.tight_layout()
2565     elif filename == 'COMP':
2566         core_path = simcase_path
2567         path_ar_comp = os.path.join(core_path, 'COMP' + '.parquet')
2568         path_ar_time = os.path.join(core_path, 'TIME' + '.parquet')
2569         x_comp = pd.read_parquet(path_ar_comp)
2570         x_time = pd.read_parquet(path_ar_time)
2571         pressure = [simcase_child[1][1]]
2572         component = [simcase_child[1][2]]
2573         compbase = x_comp.loc[(x_comp.index.isin
2574             (time_element[0])) & (x_comp['Component'].isin(component)) & (x_comp['Pressure'].isin(pressure)),
2575             z_property_chosen]
2576         timebase = compbase.index.tolist()

```

```

2574             timesnew = []
2575             for timeitem in timebase:
2576                 currenttime = datetime.datetime.
2577                     strftime(str(timeitem), "%Y-%m-%d %H:%M:%S")
2578                     daysnow = currenttime.timetuple().
2579                         tm_yday
2580                     timesnew.append(daysnow)
2581
2582                     row_id, col_id = self.find_row_col(
2583                         identifier, chosen_rows, chosen_cols)
2584                         ax = None
2585                         if plot_version == 0:
2586                             if not prev_ax:
2587                                 ax = fig.add_subplot(chosen_rows,
2588                                     chosen_cols, identifier)
2589                             else:
2590                                 if sharex_local == 'all' and
2591                                     sharey_local != 'all':
2592                                         ax = fig.add_subplot(
2593                                             chosen_rows, chosen_cols, identifier,
2594                                             sharex=prev_ax)
2595                                         elif sharex_local != 'all' and
2596                                             sharey_local == 'all':
2597                                                 ax = fig.add_subplot(
2598                                                     chosen_rows, chosen_cols, identifier,
2599                                                     sharey=prev_ax)
2600                                         elif sharex_local == 'all' and
2601                                             sharey_local == 'all':
2602                                                 ax = fig.add_subplot(
2603                                                     chosen_rows, chosen_cols, identifier)
2604                                         elif plot_version == 1:
2605                                             if (row_id, col_id) == (None, None):
2606                                                 break
2607                                                 if chosen_rows == 1 and chosen_cols
2608 == 1: # Only plot one figure
2609                                                 ax = axes
2610                                                 elif chosen_rows == 1 and
2611                                                     chosen_cols != 1: # Only plot against col_id
2612                                                     ax = axes[col_id]
2613                                                     tight_plot = True
2614                                                     elif chosen_rows != 1 and
2615                                                         chosen_cols == 1: # Only plot against row_id
2616                                                         ax = axes[row_id]

```

```

2604                     elif chosen_rows != 1 and
2605                         chosen_cols != 1: # Use both row_id and col_id
2606                             ax = axes[row_id, col_id]
2607                             ax.set_title(title)
2608                             xlabel, ylabel = ('Days',
2609                               z_property_chosen)
2610                             if new_xlabel:
2611                                 xlabel = new_xlabel
2612                             if new_ylabel:
2613                                 ylabel = new_ylabel
2614                                 ax.set_xlabel(xlabel)
2615                                 ax.set_ylabel(ylabel)
2616                                 ax.plot(timesnew, compbase)
2617                             elif filename == 'REGION':
2618                                 core_path = simcase_path
2619                                 path_ar_reg = os.path.join(core_path, 'REGION' + '.parquet')
2620                                 path_ar_time = os.path.join(core_path, 'TIME' + '.parquet')
2621                                 x_reg = pd.read_parquet(path_ar_reg)
2622                                 x_time = pd.read_parquet(path_ar_time)
2623
2624                                 region = [simcase_child[1][1]]
2625                                 component = [simcase_child[1][2]]
2626                                 regbase = x_reg.loc[(x_reg.index.isin(
2627                                   time_element[0])) & (x_reg['Component'].isin(component)) & (x_reg['Region'].isin(region)), z_property_chosen]
2628                                 timebase = regbase.index.tolist()
2629                                 timesnew = []
2630                                 for timeitem in timebase:
2631                                     currenttime = datetime.datetime.strptime(str(timeitem), "%Y-%m-%d %H:%M:%S")
2632                                     daysnow = currenttime.timetuple().tm_yday
2633                                     timesnew.append(daysnow)
2634
2635                                     row_id, col_id = self.find_row_col(
2636                                       identifier, chosen_rows, chosen_cols)
2637                                     ax = None
2638                                     if plot_version == 0:
2639                                         if not prev_ax:
2640                                             ax = fig.add_subplot(chosen_rows,
2641                                               chosen_cols, identifier)
2642                                         else:
2643                                             if sharex_local == 'all' and

```

```

2638 sharey_local != 'all':
2639                         ax = fig.add_subplot(
2640                         chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2641                         elif sharex_local != 'all' and
2642                         sharey_local == 'all':
2643                         ax = fig.add_subplot(
2644                         chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2645                         elif sharex_local == 'all' and
2646                         sharey_local == 'all':
2647                         ax = fig.add_subplot(
2648                         chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2649                         sharey=prev_ax)
2650                         else:
2651                         ax = fig.add_subplot(
2652                         chosen_rows, chosen_cols, identifier)
2653                         elif plot_version == 1:
2654                             if (row_id, col_id) == (None, None):
2655                                 break
2656                             if chosen_rows == 1 and chosen_cols
2657                             == 1: # Only plot one figure
2658                                 ax = axes
2659                                 elif chosen_rows == 1 and
2660                         chosen_cols != 1: # Only plot against col_id
2661                                 ax = axes[col_id]
2662                                 tight_plot = True
2663                                 elif chosen_rows != 1 and
2664                         chosen_cols == 1: # Only plot against row_id
2665                                 ax = axes[row_id]
2666                                 elif chosen_rows != 1 and
2667                         chosen_cols != 1: # Use both row_id and col_id
2668                                 ax = axes[row_id, col_id]
2669                                 ax.set_title(title)
2670                                 xlabel, ylabel = ('Days',
2671                                     z_property_chosen)
2672                                 if new_xlabel:
2673                                     xlabel = new_xlabel
2674                                 if new_ylabel:
2675                                     ylabel = new_ylabel
2676                                 ax.set_xlabel(xlabel)
2677                                 ax.set_ylabel(ylabel)
2678                                 ax.plot(timesnew, regbase)
2679                                 elif filename == 'WELLS':
2680                                     core_path = simcase_path
2681                                     path_ar_wells = os.path.join(core_path,
2682                                         'WELLS' + '.parquet')

```

```

2670             path_ar_time = os.path.join(core_path, 'TIME' + '.parquet')
2671             x_wells = pd.read_parquet(path_ar_wells)
2672             x_time = pd.read_parquet(path_ar_time)
2673             dates = time_element[0]
2674             if len(tlist) == 1:
2675                 dates = x_time.index.tolist()
2676
2677             wellnumber = [simcase_child[1][1]]
2678             wellname = [simcase_child[1][2]]
2679             welltype = [simcase_child[1][3]]
2680             connections = x_wells['Connection'].tolist()
2681             newconnect = []
2682             for connection in connections:
2683                 if 'Total' in connection:
2684                     newconnect.append(connection)
2685             try:
2686                 new_z = dict_paramv2[simcase][z_property_chosen]
2687             except KeyError:
2688                 new_z = z_property_chosen
2689                 wellbase = x_wells.loc[(x_wells.index.isin(dates)) & (x_wells['nWell'].isin(wellnumber)) & (x_wells['nWellName'].isin(wellname) & (x_wells['nType'].isin(welltype)) & (x_wells['Connection'].isin(newconnect))), new_z]
2690                 timebase = wellbase.index.tolist()
2691                 timesnew = []
2692                 for timeitem in timebase:
2693                     currenttime = datetime.datetime.strptime(str(timeitem), "%Y-%m-%d %H:%M:%S")
2694                     daysnow = currenttime.timetuple().tm_yday
2695                     timesnew.append(daysnow)
2696
2697                     row_id, col_id = self.find_row_col(identifier, chosen_rows, chosen_cols)
2698                     ax = None
2699                     if plot_version == 0:
2700                         if not prev_ax:
2701                             ax = fig.add_subplot(chosen_rows, chosen_cols, identifier)
2702                         else:
2703                             if sharex_local == 'all' and

```

```

2703 sharey_local != 'all':
2704                                     ax = fig.add_subplot(
2705                                         chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2706                                     elif sharex_local != 'all' and
2707                                         sharey_local == 'all':
2708                                     ax = fig.add_subplot(
2709                                         chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2710                                     elif sharex_local == 'all' and
2711                                         sharey_local == 'all':
2712                                     ax = fig.add_subplot(
2713                                         chosen_rows, chosen_cols, identifier, sharex=prev_ax,
2714                                         sharey=prev_ax)
2715                                     else:
2716                                     ax = fig.add_subplot(
2717                                         chosen_rows, chosen_cols, identifier)
2718                                     elif plot_version == 1:
2719                                         if (row_id, col_id) == (None, None):
2720                                             break
2721                                         if chosen_rows == 1 and chosen_cols
2722                                         == 1: # Only plot one figure
2723                                             ax = axes
2724                                         elif chosen_rows == 1 and
2725                                         chosen_cols != 1: # Only plot against col_id
2726                                             ax = axes[col_id]
2727                                             tight_plot = True
2728                                         elif chosen_rows != 1 and
2729                                         chosen_cols == 1: # Only plot against row_id
2730                                             ax = axes[row_id]
2731                                         elif chosen_rows != 1 and
2732                                         chosen_cols != 1: # Use both row_id and col_id
2733                                             ax = axes[row_id, col_id]
2734                                             ax.set_title(title)
2735                                             xlabel, ylabel = ('Days',
2736                                         z_property_chosen)
2737                                             if new_xlabel:
2738                                                 xlabel = new_xlabel
2739                                             if new_ylabel:
2740                                                 ylabel = new_ylabel
2741                                             ax.set_xlabel(xlabel)
2742                                             ax.set_ylabel(ylabel)
2743                                             ax.plot(timesnew, wellbase)
2744                                             prev_ax = ax
2745
2746                                         if plot_version == 1:
2747                                             for delete_empty_fig in list(range(

```

```

2735 fig_plotted + 1, chosen_cols * chosen_rows + 1)):
2736             row_id_del, col_id_del = self.
2737             find_row_col(delete_empty_fig, chosen_rows, chosen_cols)
2738             if chosen_rows == 1 and chosen_cols != 1
2739             :
2740                 axes[col_id_del].remove()
2741                 axes[col_id_del] = None
2742             elif chosen_rows != 1 and chosen_cols ==
2743                 1:
2744                     axes[row_id_del].remove()
2745                     axes[row_id_del] = None
2746             elif chosen_rows != 1 and chosen_cols !=
2747                 1:
2748                 axes[row_id_del, col_id_del].remove(
2749 )
2750                 axes[row_id_del, col_id_del] = None
2751             elif chosen_rows == 1 and chosen_cols ==
2752                 1:
2753                 axes.remove()
2754                 axes = None
2755
2756         if alls:
2757             self.canvas = FigureCanvasTkAgg(fig, self.
2758 f2_plot)
2759             self.canvas.draw()
2760             self.canvas.get_tk_widget().pack(side=tk.TOP,
2761 , fill=tk.BOTH, expand=True)
2762             self.canvas._tkcanvas.pack(side=tk.BOTTOM,
2763 fill=tk.BOTH, expand=True)
2764             self.toolbar = NavigationToolbar2Tk(self.
2765 canvas, self.f2_toolkit) # Toolbar is added to canvas
2766             self.toolbar.update()
2767             if self.hold.get() == 1:
2768                 self.figs = 0
2769                 self.label_figs['text'] = 'Figures: ' +
str(self.figs)
2770                 self.grid_size_figures()
2771                 for i_del in reversed(alls):
2772                     x_del = i_del
2773                     child.delete(alls.index(x_del))
2774
2775             def clear_xy(self, typedata):
2776                 child = None
2777                 if typedata == 'X':
2778                     child = self.x_listbox

```

```

2769             self.plot_x = None
2770         elif typedata == 'Y':
2771             child = self.y_listbox
2772             self.plot_y = None
2773             alls = list(child.get(0, END))
2774             if alls:
2775                 child.delete(0, END)
2776
2777     def add_to_xy(self, typedata):
2778         if self.plot_id and len(self.plot_id['entries']) == 1:
2779             core_path = self.plot_id['simcase_path']
2780             cells, times = self.get_cell_time(corepath=core_path)
2781             newvalues = {'cells': cells, 'time': times}
2782             self.plot_id.update(newvalues)
2783             simcase = self.plot_id['simcase']
2784             simcase_child = self.plot_id['simcase_child'][0]
2785             entry = self.plot_id['entries']
2786             child = None
2787             if typedata == 'X':
2788                 self.plot_x = {}
2789                 child = self.x_listbox
2790                 alls = list(child.get(0, END))
2791                 if alls:
2792                     child.delete(0, END)
2793                     element_shown = str(simcase) + ' ' + str(simcase_child) + ' ' + str(entry[0])
2794                     element = self.plot_id
2795                     self.plot_x[element_shown] = element
2796                     child.insert(END, element_shown)
2797             elif typedata == 'Y':
2798                 self.plot_y = {}
2799                 child = self.y_listbox
2800                 alls = list(child.get(0, END))
2801                 if alls:
2802                     child.delete(0, END)
2803                     element_shown = str(simcase) + ' ' + str(simcase_child) + ' ' + str(entry[0])
2804                     element = self.plot_id
2805                     self.plot_y[element_shown] = element
2806                     child.insert(END, element_shown)
2807
2808     def add_to_plot_list(self):

```

```

2809         child_clear = self.local_sim_parameters
2810         child_clear.selection_clear(0, END)
2811         childx = self.x_listbox
2812         childdy = self.y_listbox
2813
2814         choicedays = [7, 8]
2815         currentdays = self.xyz.get()
2816         if currentdays in choicedays:
2817             self.xyz.set(currentdays) # Updates chosen
2818             cells and times manually
2819
2820         if self.plot_id:
2821             child = self.pageone_listbox_plot # Listbox
2822             where to insert
2823             data = self.plot_id # New data potentially
2824             incoming
2825             core_path = self.plot_id['simcase_path']
2826             cells, time_element = self.get_cell_time(
2827             corepath=core_path)
2828             ilist, jlist, klist, tlist = (time_element[2],
2829             time_element[3], time_element[4], time_element[5])
2830             timedays = self.get_time(core_path, 'Days',
2831             tlist).values.tolist()
2832             simcase = self.plot_id['simcase']
2833             simcase_child = self.plot_id['simcase_child']
2834             [0]
2835             newvalues = {'cells': cells, 'time':
2836             time_element, 'X': self.plot_x, 'Y': self.plot_y}
2837             self.plot_id.update(newvalues)
2838             entries = list(self.plot_id['entries'])
2839
2840             for item in entries:
2841                 newdict = {}
2842                 for i in list(self.plot_id.keys()):
2843                     if i == 'entries':
2844                         newdict[i] = item
2845                     else:
2846                         newdict[i] = self.plot_id[i]
2847                         alls = list(child.get(0, END))
2848                         count = len(alls) + 1
2849                         marker = self.hold2.get()
2850                         element_shown = str(count) + ': ' + str(
2851                         simcase) + ' ' + str(simcase_child) + ' ' + str(item)
2852                         shown_title = None
2853                         if self.showsimcase.get() == 0:

```

```

2845                         shown_title = str(simcase) + ' ' +
2846                         str(item)
2847                         elif self.showsimcase.get() == 1:
2848                             shown_title = str(item)
2849                             if self.showtime.get() == 0:
2850                                 shown_title = shown_title + ' t=' +
2851                                 str(timedays[0]) + ' days'
2852                                 dict_title = {'shown_title': shown_title
2853 , 'simcase': str(simcase) + ':', 'parameter': str(item),
2854 'timedays': timedays, ' xlabel': None, ' ylabel': None}
2855                                 newdict['title'] = dict_title
2856                                 newdict['fontsize'] = self.fontsize
2857                                 element = [count, newdict, marker]
2858                                 self.plot_rdy[element_shown] = element
2859                                 child.insert(END, element_shown)
2860                                 alls = list(child.get(0, END))
2861                                 self.figs = len(alls)
2862                                 self.label_figs['text'] = 'Figures: ' + str(
2863 self.figs)
2864                                 self.grid_size_figures()
2865
2866     def get_cell_time(self, corepath):
2867         core_path = corepath
2868         path_ar_input = os.path.join(core_path, 'INPUT'
2869 + '.parquet')
2870         path_ar_time = os.path.join(core_path, 'TIME' +
2871 '.parquet')
2872         x_input = pd.read_parquet(path_ar_input)
2873         x_time = pd.read_parquet(path_ar_time)
2874         imin, imax = (int(self.slidex_left.get()), int(
2875 self.slidex_right.get()))
2876         jmin, jmax = (int(self.slidey_left.get()), int(
2877 self.slidey_right.get()))
2878         kmin, kmax = (int(self.slidez_left.get()), int(
2879 self.slidez_right.get()))
2880         tmin, tmax = (int(self.slidetime_left.get()),
2881 int(self.slidetime_right.get()))
2882         ilist = list(range(imin, imax + 1))
2883         jlist = list(range(jmin, jmax + 1))
2884         klist = list(range(kmin, kmax + 1))
2885         tlist = list(range(tmin, tmax + 1))
2886         cells = x_input.loc[(x_input['i'].isin(ilist)) &
2887 (x_input['j'].isin(jlist)) & (x_input['k'].isin(klist))
2888 , 'Cell'].tolist()
2889         times = x_time.loc[x_time['nStep'].isin(tlist),

```

```

2876 :].index
2877     days = x_time.loc[x_time['nStep'].isin(tlist), 'nDays']
2878     time_element = [times, days, ilist, jlist, klist
2879 , tlist]
2880     return cells, time_element
2881
2882     def remove_from_plot_list(self):
2883         child = self.pageone_listbox_plot
2884         cursors = child.curselection()
2885         alls = None
2886         for item in reversed(cursors):
2887             x_del = child.get(item)
2888             child.delete(item)
2889             numbering = self.plot_rdy[x_del][0] #
2890             Position before deletion
2891             alls = list(child.get(0, END))
2892             self.plot_rdy.pop(x_del, None)
2893             for i in list(range(numbering, len(alls)+1)):
2894                 :
2895                 access = (i-1)
2896                 old_key = child.get(access)
2897                 old_key_data = self.plot_rdy[old_key]
2898                 self.plot_rdy.pop(old_key, None)
2899                 current_numbering = old_key_data[0]
2900                 current_marker = old_key_data[2]
2901                 new_numbering = current_numbering - 1
2902                 new_key_data = [new_numbering,
2903                 old_key_data[1], current_marker]
2904                 new_key = str(new_numbering) + ':' +
2905                 old_key.split(':')[1]
2906                 self.plot_rdy[new_key] = new_key_data
2907                 child.delete(access)
2908                 child.insert(access, new_key)
2909                 self.figs = len(alls)
2910                 self.label_figs['text'] = 'Figures: ' + str(self
2911 .figs)
2912                 self.grid_size_figures()
2913
2914     def get_multiple_items(self, event):
2915         '''Use current selection as 'parent', then tie
2916         that to how many are selected in the second listbox (
2917         with the properties)'''
2918         simcase = self.simcase
2919         simcase_path = self.simcase_path

```

```

2912         simcase_child = self.simcase_child
2913         element_list = {}
2914         w = event.widget
2915         index = 0
2916         child = self.pageone_listbox_plot
2917         alls = list(child.get(0, END))
2918         self.plot_id = {}
2919         for i in ['simcase', 'simcase_path', 'simcase_child', 'entries', 'cells', 'time', 'X', 'Y']:
2920             self.plot_id[i] = None
2921         try:
2922             index = w.curselection()[0]
2923             properties = [w.get(int(i)) for i in w.curselection()]
2924             simcase_child_element = [simcase_child] + [self.merged_listbox_items[simcase_child]]
2925             newvalues = {'simcase': simcase, 'simcase_path': simcase_path, 'simcase_child': simcase_child_element, 'entries': properties}
2926             self.plot_id.update(newvalues)
2927         except IndexError:
2928             pass
2929
2930     def get_folded_properties(self, event):
2931         if current_tab == 'Page One':
2932             simcase = self.simcase
2933             w = event.widget
2934             index = 0
2935             try:
2936                 index = w.curselection()[0]
2937                 value = w.get(index)
2938                 print('value: ' + str(value))
2939                 if value != self.simcase_child:
2940                     self.simcase_child = value
2941                     filename = self.merged_listbox_items[value][0]
2942                     core_path = self.simcase_path
2943                     columns = None
2944                     if filename == 'INPUT':
2945                         path_ar_input = os.path.join(core_path, 'INPUT' + '.parquet')
2946                         columns = pd.read_parquet(path_ar_input).columns.tolist()
2947                     elif filename == 'DATA':
2948                         path_ar_data = os.path.join(

```

```

2948 core_path, 'DATA' + '.parquet')
2949 path_ar_param = os.path.join(
    core_path, 'PARAMETERS' + '.parquet')
2950 x_param = pd.read_parquet(
    path_ar_param)
2951 unconverted = list(x_param.index)
2952 converted = list(x_param.iloc[:, 0])
2953 extra_columns = pd.read_parquet(
    path_ar_data).columns.tolist()[:-len(converted)]
2954 unconverted = extra_columns +
    unconverted
2955 converted = extra_columns +
    converted
2956 for pos in list(range(len(
    converted))):
2957     self.data_conversion[
        unconverted[pos]] = converted[pos]
2958     columns = unconverted
2959 elif filename == 'TIME':
2960     path_ar_time = os.path.join(
        core_path, 'TIME' + '.parquet')
2961     columns = pd.read_parquet(
        path_ar_time).columns.tolist()
2962 elif filename == 'COMP':
2963     path_ar_comp = os.path.join(
        core_path, 'COMP' + '.parquet')
2964     columns = pd.read_parquet(
        path_ar_comp).columns.tolist()[2:]
2965 elif filename == 'REGION':
2966     path_ar_region = os.path.join(
        core_path, 'REGION' + '.parquet')
2967     columns = pd.read_parquet(
        path_ar_region).columns.tolist()[2:]
2968 elif filename == 'WELLS':
2969     path_ar_wells = os.path.join(
        core_path, 'WELLS' + '.parquet')
2970     try:
2971         columns = list(dict_paramv2[
            simcase].keys())
2972     except KeyError:
2973         columns = pd.read_parquet(
            path_ar_wells).columns.tolist()[3:]
2974     values_to_be_inserted = columns

```

```

2975                     alls = list(self.
2976                         local_sim_parameters.get(0, END))
2977                         if alls:
2978                             self.local_sim_parameters.delete
2979                                 (0, END)
2980                                 for item in values_to_be_inserted:
2981                                     self.local_sim_parameters.insert
2982                                         (END, item)
2983                                         except IndexError:
2984                                             pass
2985                                         pass
2986
2987
2988
2989     def get_selected_item_prep(self, event):
2990         if current_tab == 'Page One':
2991             global current_selection
2992             w = event.widget
2993             index = 0
2994             try:
2995                 index = w.curselection()[0]
2996                 value = w.get(index)
2997
2998                 if value != current_selection:
2999                     self.merged_listbox_items = {}
3000                     self.simcase_child = None
3001                     self.clear_xy(typedata='X')
3002                     self.clear_xy(typedata='Y')
3003                     child = self.local_sim_parameters
3004                     child.selection_clear(0, END)
3005                     child.delete(0, END)
3006                     child2 = self.prep_sim_parameters
3007                     child2.selection_clear(0, END)
3008                     child2.delete(0, END)
3009
3010                     # child = self.prep_sim_parameters
3011
3012                     current_selection = value
3013                     self.simcase = value
3014                     values_to_be_inserted = []
3015                     core_path = global_sim_data[value][0]
3016                         ]
3017                         print('core_path: ' + str(core_path))
3018
3019                     self.simcase_path = core_path
3020                     ref_values = {'INPUT': 1, 'DATA': 2,

```

```

3014     'TIME': 3, 'COMP': 4, 'REGION': 5, 'WELLS': 6}
3015
3016             for filename in global_sim_data[
3017                 value][1:]:
3018
3019                     if filename not in ['PARAMETERS'
3020                         , 'WELLPARAM']:
3021
3022                         element = [filename,
3023                         ref_values[filename]]
3024
3025                         values_to_be_inserted.append(
3026                             (element))
3027
3028                     sorted_version = sorted(
3029                         values_to_be_inserted, key=lambda x1: x1[1])
3030
3031                     values_to_be_inserted = []
3032                     for i in sorted_version:
3033
3034                         values_to_be_inserted.append(i[0]
3035 )
3036
3037                     for filename in
3038                         values_to_be_inserted:
3039
3040                         if filename == 'COMP':
3041
3042                             path_ar_comp = os.path.join(
3043                                 core_path, 'COMP' + '.parquet')
3044
3045                             x_comp = pd.read_parquet(
3046                                 path_ar_comp)
3047
3048
3049                             components = np.unique(
3050                                 x_comp['Component'].tolist())
3051
3052                             pressures = np.unique(x_comp
3053                                 ['Pressure'])
3054
3055                             for pressure in pressures:
3056
3057                                 for component in
3058                                     components:
3059
3060                                     element = ['COMP',
3061                                     pressure, component]
3062
3063                                     element_shown = ' '
3064                                     element_shown += 'COMP ' + 'P=' + str(pressure) + ' '
3065                                     element_shown += str(component)
3066                                     element_shown += self.
3067
3068                                     merged_listbox_items[element_shown] = element
3069
3070                                     elif filename == 'DATA':
3071
3072                                         self.merged_listbox_items[' '
3073                                             'DATA'] = ['DATA']
3074
3075                                     elif filename == 'INPUT':
3076
3077                                         self.merged_listbox_items[' '
3078                                             'INPUT'] = ['INPUT']
3079
3080                                     elif filename == 'REGION':
3081
3082                                         self.merged_listbox_items[' '
3083                                             'REGION'] = ['REGION']
3084
3085                                         self.merged_listbox_items[' '
3086                                             'WELLS'] = ['WELLS']
3087
3088                                         self.merged_listbox_items[' '
3089                                             'TIME'] = ['TIME']
3090
3091                                         self.merged_listbox_items[' '
3092                                             'COMP'] = ['COMP']
3093
3094                                         self.merged_listbox_items[' '
3095                                             'REGION'] = ['REGION']
3096
3097                                         self.merged_listbox_items[' '
3098                                             'WELLPARAM'] = ['WELLPARAM']
3099
3100                                         self.merged_listbox_items[' '
3101                                             'PARAMETERS'] = ['PARAMETERS']
3102
3103                                         self.merged_listbox_items[' '
3104                                             'DATA'] = ['DATA']
3105
3106                                         self.merged_listbox_items[' '
3107                                             'INPUT'] = ['INPUT']
3108
3109                                         self.merged_listbox_items[' '
3110                                             'TIME'] = ['TIME']
3111
3112                                         self.merged_listbox_items[' '
3113                                             'COMP'] = ['COMP']
3114
3115                                         self.merged_listbox_items[' '
3116                                             'REGION'] = ['REGION']
3117
3118                                         self.merged_listbox_items[' '
3119                                             'WELLS'] = ['WELLS']
3120
3121                                         self.merged_listbox_items[' '
3122                                             'TIME'] = ['TIME']
3123
3124                                         self.merged_listbox_items[' '
3125                                             'COMP'] = ['COMP']
3126
3127                                         self.merged_listbox_items[' '
3128                                             'REGION'] = ['REGION']
3129
3130                                         self.merged_listbox_items[' '
3131                                             'WELLPARAM'] = ['WELLPARAM']
3132
3133                                         self.merged_listbox_items[' '
3134                                             'PARAMETERS'] = ['PARAMETERS']
3135
3136                                         self.merged_listbox_items[' '
3137                                             'DATA'] = ['DATA']
3138
3139                                         self.merged_listbox_items[' '
3140                                             'INPUT'] = ['INPUT']
3141
3142                                         self.merged_listbox_items[' '
3143                                             'TIME'] = ['TIME']
3144
3145                                         self.merged_listbox_items[' '
3146                                             'COMP'] = ['COMP']
3147
3148                                         self.merged_listbox_items[' '
3149                                             'REGION'] = ['REGION']
3150
3151                                         self.merged_listbox_items[' '
3152                                             'WELLS'] = ['WELLS']
3153
3154                                         self.merged_listbox_items[' '
3155                                             'TIME'] = ['TIME']
3156
3157                                         self.merged_listbox_items[' '
3158                                             'COMP'] = ['COMP']
3159
3160                                         self.merged_listbox_items[' '
3161                                             'REGION'] = ['REGION']
3162
3163                                         self.merged_listbox_items[' '
3164                                             'WELLPARAM'] = ['WELLPARAM']
3165
3166                                         self.merged_listbox_items[' '
3167                                             'PARAMETERS'] = ['PARAMETERS']
3168
3169                                         self.merged_listbox_items[' '
3170                                             'DATA'] = ['DATA']
3171
3172                                         self.merged_listbox_items[' '
3173                                             'INPUT'] = ['INPUT']
3174
3175                                         self.merged_listbox_items[' '
3176                                             'TIME'] = ['TIME']
3177
3178                                         self.merged_listbox_items[' '
3179                                             'COMP'] = ['COMP']
3180
3181                                         self.merged_listbox_items[' '
3182                                             'REGION'] = ['REGION']
3183
3184                                         self.merged_listbox_items[' '
3185                                             'WELLS'] = ['WELLS']
3186
3187                                         self.merged_listbox_items[' '
3188                                             'TIME'] = ['TIME']
3189
3190                                         self.merged_listbox_items[' '
3191                                             'COMP'] = ['COMP']
3192
3193                                         self.merged_listbox_items[' '
3194                                             'REGION'] = ['REGION']
3195
3196                                         self.merged_listbox_items[' '
3197                                             'WELLPARAM'] = ['WELLPARAM']
3198
3199                                         self.merged_listbox_items[' '
3200                                             'PARAMETERS'] = ['PARAMETERS']
3201
3202                                         self.merged_listbox_items[' '
3203                                             'DATA'] = ['DATA']
3204
3205                                         self.merged_listbox_items[' '
3206                                             'INPUT'] = ['INPUT']
3207
3208                                         self.merged_listbox_items[' '
3209                                             'TIME'] = ['TIME']
3210
3211                                         self.merged_listbox_items[' '
3212                                             'COMP'] = ['COMP']
3213
3214                                         self.merged_listbox_items[' '
3215                                             'REGION'] = ['REGION']
3216
3217                                         self.merged_listbox_items[' '
3218                                             'WELLS'] = ['WELLS']
3219
3220                                         self.merged_listbox_items[' '
3221                                             'TIME'] = ['TIME']
3222
3223                                         self.merged_listbox_items[' '
3224                                             'COMP'] = ['COMP']
3225
3226                                         self.merged_listbox_items[' '
3227                                             'REGION'] = ['REGION']
3228
3229                                         self.merged_listbox_items[' '
3230                                             'WELLPARAM'] = ['WELLPARAM']
3231
3232                                         self.merged_listbox_items[' '
3233                                             'PARAMETERS'] = ['PARAMETERS']
3234
3235                                         self.merged_listbox_items[' '
3236                                             'DATA'] = ['DATA']
3237
3238                                         self.merged_listbox_items[' '
3239                                             'INPUT'] = ['INPUT']
3240
3241                                         self.merged_listbox_items[' '
3242                                             'TIME'] = ['TIME']
3243
3244                                         self.merged_listbox_items[' '
3245                                             'COMP'] = ['COMP']
3246
3247                                         self.merged_listbox_items[' '
3248                                             'REGION'] = ['REGION']
3249
3250                                         self.merged_listbox_items[' '
3251                                             'WELLS'] = ['WELLS']
3252
3253                                         self.merged_listbox_items[' '
3254                                             'TIME'] = ['TIME']
3255
3256                                         self.merged_listbox_items[' '
3257                                             'COMP'] = ['COMP']
3258
3259                                         self.merged_listbox_items[' '
3260                                             'REGION'] = ['REGION']
3261
3262                                         self.merged_listbox_items[' '
3263                                             'WELLPARAM'] = ['WELLPARAM']
3264
3265                                         self.merged_listbox_items[' '
3266                                             'PARAMETERS'] = ['PARAMETERS']
3267
3268                                         self.merged_listbox_items[' '
3269                                             'DATA'] = ['DATA']
3270
3271                                         self.merged_listbox_items[' '
3272                                             'INPUT'] = ['INPUT']
3273
3274                                         self.merged_listbox_items[' '
3275                                             'TIME'] = ['TIME']
3276
3277                                         self.merged_listbox_items[' '
3278                                             'COMP'] = ['COMP']
3279
3280                                         self.merged_listbox_items[' '
3281                                             'REGION'] = ['REGION']
3282
3283                                         self.merged_listbox_items[' '
3284                                             'WELLS'] = ['WELLS']
3285
3286                                         self.merged_listbox_items[' '
3287                                             'TIME'] = ['TIME']
3288
3289                                         self.merged_listbox_items[' '
3290                                             'COMP'] = ['COMP']
3291
3292                                         self.merged_listbox_items[' '
3293                                             'REGION'] = ['REGION']
3294
3295                                         self.merged_listbox_items[' '
3296                                             'WELLPARAM'] = ['WELLPARAM']
3297
3298                                         self.merged_listbox_items[' '
3299                                             'PARAMETERS'] = ['PARAMETERS']
3300
3301                                         self.merged_listbox_items[' '
3302                                             'DATA'] = ['DATA']
3303
3304                                         self.merged_listbox_items[' '
3305                                             'INPUT'] = ['INPUT']
3306
3307                                         self.merged_listbox_items[' '
3308                                             'TIME'] = ['TIME']
3309
3310                                         self.merged_listbox_items[' '
3311                                             'COMP'] = ['COMP']
3312
3313                                         self.merged_listbox_items[' '
3314                                             'REGION'] = ['REGION']
3315
3316                                         self.merged_listbox_items[' '
3317                                             'WELLS'] = ['WELLS']
3318
3319                                         self.merged_listbox_items[' '
3320                                             'TIME'] = ['TIME']
3321
3322                                         self.merged_listbox_items[' '
3323                                             'COMP'] = ['COMP']
3324
3325                                         self.merged_listbox_items[' '
3326                                             'REGION'] = ['REGION']
3327
3328                                         self.merged_listbox_items[' '
3329                                             'WELLPARAM'] = ['WELLPARAM']
3330
3331                                         self.merged_listbox_items[' '
3332                                             'PARAMETERS'] = ['PARAMETERS']
3333
3334                                         self.merged_listbox_items[' '
3335                                             'DATA'] = ['DATA']
3336
3337                                         self.merged_listbox_items[' '
3338                                             'INPUT'] = ['INPUT']
3339
3340                                         self.merged_listbox_items[' '
3341                                             'TIME'] = ['TIME']
3342
3343                                         self.merged_listbox_items[' '
3344                                             'COMP'] = ['COMP']
3345
3346                                         self.merged_listbox_items[' '
3347                                             'REGION'] = ['REGION']
3348
3349                                         self.merged_listbox_items[' '
3350                                             'WELLS'] = ['WELLS']
3351
3352                                         self.merged_listbox_items[' '
3353                                             'TIME'] = ['TIME']
3354
3355                                         self.merged_listbox_items[' '
3356                                             'COMP'] = ['COMP']
3357
3358                                         self.merged_listbox_items[' '
3359                                             'REGION'] = ['REGION']
3360
3361                                         self.merged_listbox_items[' '
3362                                             'WELLPARAM'] = ['WELLPARAM']
3363
3364                                         self.merged_listbox_items[' '
3365                                             'PARAMETERS'] = ['PARAMETERS']
3366
3367                                         self.merged_listbox_items[' '
3368                                             'DATA'] = ['DATA']
3369
3370                                         self.merged_listbox_items[' '
3371                                             'INPUT'] = ['INPUT']
3372
3373                                         self.merged_listbox_items[' '
3374                                             'TIME'] = ['TIME']
3375
3376                                         self.merged_listbox_items[' '
3377                                             'COMP'] = ['COMP']
3378
3379                                         self.merged_listbox_items[' '
3380                                             'REGION'] = ['REGION']
3381
3382                                         self.merged_listbox_items[' '
3383                                             'WELLS'] = ['WELLS']
3384
3385                                         self.merged_listbox_items[' '
3386                                             'TIME'] = ['TIME']
3387
3388                                         self.merged_listbox_items[' '
3389                                             'COMP'] = ['COMP']
3390
3391                                         self.merged_listbox_items[' '
3392                                             'REGION'] = ['REGION']
3393
3394                                         self.merged_listbox_items[' '
3395                                             'WELLPARAM'] = ['WELLPARAM']
3396
3397                                         self.merged_listbox_items[' '
3398                                             'PARAMETERS'] = ['PARAMETERS']
3399
3400                                         self.merged_listbox_items[' '
3401                                             'DATA'] = ['DATA']
3402
3403                                         self.merged_listbox_items[' '
3404                                             'INPUT'] = ['INPUT']
3405
3406                                         self.merged_listbox_items[' '
3407                                             'TIME'] = ['TIME']
3408
3409                                         self.merged_listbox_items[' '
3410                                             'COMP'] = ['COMP']
3411
3412                                         self.merged_listbox_items[' '
3413                                             'REGION'] = ['REGION']
3414
3415                                         self.merged_listbox_items[' '
3416                                             'WELLS'] = ['WELLS']
3416
3417                                         self.merged_listbox_items[' '
3418                                             'TIME'] = ['TIME']
3419
3420                                         self.merged_listbox_items[' '
3421                                             'COMP'] = ['COMP']
3422
3423                                         self.merged_listbox_items[' '
3424                                             'REGION'] = ['REGION']
3425
3426                                         self.merged_listbox_items[' '
3427                                             'WELLPARAM'] = ['WELLPARAM']
3427
3428                                         self.merged_listbox_items[' '
3429                                             'PARAMETERS'] = ['PARAMETERS']
3429
3430                                         self.merged_listbox_items[' '
3431                                             'DATA'] = ['DATA']
3432
3433                                         self.merged_listbox_items[' '
3434                                             'INPUT'] = ['INPUT']
3435
3436                                         self.merged_listbox_items[' '
3437                                             'TIME'] = ['TIME']
3438
3439                                         self.merged_listbox_items[' '
3440                                             'COMP'] = ['COMP']
3440
3441                                         self.merged_listbox_items[' '
3442                                             'REGION'] = ['REGION']
3442
3443                                         self.merged_listbox_items[' '
3444                                             'WELLS'] = ['WELLS']
3444
3445                                         self.merged_listbox_items[' '
3446                                             'TIME'] = ['TIME']
3447
3448                                         self.merged_listbox_items[' '
3449                                             'COMP'] = ['COMP']
3449
3450                                         self.merged_listbox_items[' '
3451                                             'REGION'] = ['REGION']
3451
3452                                         self.merged_listbox_items[' '
3453                                             'WELLPARAM'] = ['WELLPARAM']
3452
3453                                         self.merged_listbox_items[' '
3454                                             'PARAMETERS'] = ['PARAMETERS']
3454
3455                                         self.merged_listbox_items[' '
3456                                             'DATA'] = ['DATA']
3457
3458                                         self.merged_listbox_items[' '
3459                                             'INPUT'] = ['INPUT']
3459
3460                                         self.merged_listbox_items[' '
3461                                             'TIME'] = ['TIME']
3462
3463                                         self.merged_listbox_items[' '
3464                                             'COMP'] = ['COMP']
3464
3465                                         self.merged_listbox_items[' '
3466                                             'REGION'] = ['REGION']
3466
3467                                         self.merged_listbox_items[' '
3468                                             'WELLS'] = ['WELLS']
3468
3469                                         self.merged_listbox_items[' '
3470                                             'TIME'] = ['TIME']
3470
3471                                         self.merged_listbox_items[' '
3472                                             'COMP'] = ['COMP']
3472
3473                                         self.merged_listbox_items[' '
3474                                             'REGION'] = ['REGION']
3474
3475                                         self.merged_listbox_items[' '
3476                                             'WELLPARAM'] = ['WELLPARAM']
3476
3477                                         self.merged_listbox_items[' '
3478                                             'PARAMETERS'] = ['PARAMETERS']
3478
3479                                         self.merged_listbox_items[' '
3480                                             'DATA'] = ['DATA']
3481
3482                                         self.merged_listbox_items[' '
3483                                             'INPUT'] = ['INPUT']
3483
3484                                         self.merged_listbox_items[' '
3485                                             'TIME'] = ['TIME']
3485
3486                                         self.merged_listbox_items[' '
3487                                             'COMP'] = ['COMP']
3487
3488                                         self.merged_listbox_items[' '
3489                                             'REGION'] = ['REGION']
3489
3490                                         self.merged_listbox_items[' '
3491                                             'WELLS'] = ['WELLS']
3491
3492                                         self.merged_listbox_items[' '
3493                                             'TIME'] = ['TIME']
3493
3494                                         self.merged_listbox_items[' '
3495                                             'COMP'] = ['COMP']
3495
3496                                         self.merged_listbox_items[' '
3497                                             'REGION'] = ['REGION']
3497
3498                                         self.merged_listbox_items[' '
3499                                             'WELLPARAM'] = ['WELLPARAM']
3499
3500                                         self.merged_listbox_items[' '
3501                                             'PARAMETERS'] = ['PARAMETERS']
3501
3502                                         self.merged_listbox_items[' '
3503                                             'DATA'] = ['DATA']
3504
3505                                         self.merged_listbox_items[' '
3506                                             'INPUT'] = ['INPUT']
3506
3507                                         self.merged_listbox_items[' '
3508                                             'TIME'] = ['TIME']
3508
3509                                         self.merged_listbox_items[' '
3510                                             'COMP'] = ['COMP']
3510
3511                                         self.merged_listbox_items[' '
3512                                             'REGION'] = ['REGION']
3512
3513                                         self.merged_listbox_items[' '
3514                                             'WELLS'] = ['WELLS']
3514
3515                                         self.merged_listbox_items[' '
3516                                             'TIME'] = ['TIME']
3516
3517                                         self.merged_listbox_items[' '
3518                                             'COMP'] = ['COMP']
3518
3519                                         self.merged_listbox_items[' '
3520                                             'REGION'] = ['REGION']
3520
3521                                         self.merged_listbox_items[' '
3522                                             'WELLPARAM'] = ['WELLPARAM']
3522
3523                                         self.merged_listbox_items[' '
3524                                             'PARAMETERS'] = ['PARAMETERS']
3524
3525                                         self.merged_listbox_items[' '
3526                                             'DATA'] = ['DATA']
3527
3528                                         self.merged_listbox_items[' '
3529                                             'INPUT'] = ['INPUT']
3529
3530                                         self.merged_listbox_items[' '
3531                                             'TIME'] = ['TIME']
3531
3532                                         self.merged_listbox_items[' '
3533                                             'COMP'] = ['COMP']
3533
3534                                         self.merged_listbox_items[' '
3535                                             'REGION'] = ['REGION']
3535
3536                                         self.merged_listbox_items[' '
3537                                             'WELLS'] = ['WELLS']
3537
3538                                         self.merged_listbox_items[' '
3539                                             'TIME'] = ['TIME']
3539
3540                                         self.merged_listbox_items[' '
3541                                             'COMP'] = ['COMP']
3541
3542                                         self.merged_listbox_items[' '
3543                                             'REGION'] = ['REGION']
3543
3544                                         self.merged_listbox_items[' '
3545                                             'WELLPARAM'] = ['WELLPARAM']
3545
3546                                         self.merged_listbox_items[' '
3547                                             'PARAMETERS'] = ['PARAMETERS']
3547
3548                                         self.merged_listbox_items[' '
3549                                             'DATA'] = ['DATA']
3550
3551                                         self.merged_listbox_items[' '
3552                                             'INPUT'] = ['INPUT']
3552
3553                                         self.merged_listbox_items[' '
3554                                             'TIME'] = ['TIME']
3554
3555                                         self.merged_listbox_items[' '
3556                                             'COMP'] = ['COMP']
3556
3557                                         self.merged_listbox_items[' '
3558                                             'REGION'] = ['REGION']
3558
3559                                         self.merged_listbox_items[' '
3560                                             'WELLS'] = ['WELLS']
3560
3561                                         self.merged_listbox_items[' '
3562                                             'TIME'] = ['TIME']
3562
3563                                         self.merged_listbox_items[' '
3564                                             'COMP'] = ['COMP']
3564
3565                                         self.merged_listbox_items[' '
3566                                             'REGION'] = ['REGION']
3566
3567                                         self.merged_listbox_items[' '
3568                                             'WELLPARAM'] = ['WELLPARAM']
3568
3569                                         self.merged_listbox_items[' '
3570                                             'PARAMETERS'] = ['PARAMETERS']
3570
3571                                         self.merged_listbox_items[' '
3572                                             'DATA'] = ['DATA']
3573
3574                                         self.merged_listbox_items[' '
3575                                             'INPUT'] = ['INPUT']
3575
3576                                         self.merged_listbox_items[' '
3577                                             'TIME'] = ['TIME']
3577
3578                                         self.merged_listbox_items[' '
3579                                             'COMP'] = ['COMP']
3579
3580                                         self.merged_listbox_items[' '
3581                                             'REGION'] = ['REGION']
3581
3582                                         self.merged_listbox_items[' '
3583                                             'WELLS'] = ['WELLS']
3583
3584                                         self.merged_listbox_items[' '
3585                                             'TIME'] = ['TIME']
3585
3586                                         self.merged_listbox_items[' '
3587                                             'COMP'] = ['COMP']
3587
3588                                         self.merged_listbox_items[' '
3589                                             'REGION'] = ['REGION']
3589
3590                                         self.merged_listbox_items[' '
3591                                             'WELLPARAM'] = ['WELLPARAM']
3591
3592                                         self.merged_listbox_items[' '
3593                                             'PARAMETERS'] = ['PARAMETERS']
3593
3594                                         self.merged_listbox_items[' '
3595                                             'DATA'] = ['DATA']
3596
3597                                         self.merged_listbox_items[' '
3598                                             'INPUT'] = ['INPUT']
3598
3599                                         self.merged_listbox_items[' '
3600                                             'TIME'] = ['TIME']
3600
3601                                         self.merged_listbox_items[' '
3602                                             'COMP'] = ['COMP']
3602
3603                                         self.merged_listbox_items[' '
3604                                             'REGION'] = ['REGION']
3604
3605                                         self.merged_listbox_items[' '
3606                                             'WELLS'] = ['WELLS']
3606
3607                                         self.merged_listbox_items[' '
3608                                             'TIME'] = ['TIME']
3608
3609                                         self.merged_listbox_items[' '
3610                                             'COMP'] = ['COMP']
3610
3611                                         self.merged_listbox_items[' '
3612                                             'REGION'] = ['REGION']
3612
3613                                         self.merged_listbox_items[' '
3614                                             'WELLPARAM'] = ['WELLPARAM']
3614
3615                                         self.merged_listbox_items[' '
3616                                             'PARAMETERS'] = ['PARAMETERS']
3616
3617                                         self.merged_listbox_items[' '
3618                                             'DATA'] = ['DATA']
3619
3620                                         self.merged_listbox_items[' '
3621                                             'INPUT'] = ['INPUT']
3621
3622                                         self.merged_listbox_items[' '
3623                                             'TIME'] = ['TIME']
3623
3624                                         self.merged_listbox_items[' '
3625                                             'COMP'] = ['COMP']
3625
3626                                         self.merged_listbox_items[' '
3627                                             'REGION'] = ['REGION']
3627
3628                                         self.merged_listbox_items[' '
3629                                             'WELLS'] = ['WELLS']
3629
3630                                         self.merged_listbox_items[' '
3631                                             'TIME'] = ['TIME']
3631
3632                                         self.merged_listbox_items[' '
3633                                             'COMP'] = ['COMP']
3633
3634                                         self.merged_listbox_items[' '
3635                                             'REGION'] = ['REGION']
3635
3636                                         self.merged_listbox_items[' '
3637                                             'WELLPARAM'] = ['WELLPARAM']
3637
3638                                         self.merged_listbox_items[' '
3639                                             'PARAMETERS'] = ['PARAMETERS']
3639
3640                                         self.merged_listbox_items[' '
3641                                             'DATA'] = ['DATA']
3642
3643                                         self.merged_listbox_items[' '
3644                                             'INPUT'] = ['INPUT']
3644
3645                                         self.merged_listbox_items[' '
3646                                             'TIME'] = ['TIME']
3646
3647                                         self.merged_listbox_items[' '
3648                                             'COMP'] = ['COMP']
3648
3649                                         self.merged_listbox_items[' '
3650                                             'REGION'] = ['REGION']
3650
3651                                         self.merged_listbox_items[' '
3652                                             'WELLS'] = ['WELLS']
3652
3653                                         self.merged_listbox_items[' '
3654                                             'TIME'] = ['TIME']
3654
3655                                         self.merged_listbox_items[' '
3656                                             'COMP'] = ['COMP']
3656
3657                                         self.merged_listbox_items[' '
3658                                             'REGION'] = ['REGION']
3658
3659                                         self.merged_listbox_items[' '
3660                                             'WELLPARAM'] = ['WELLPARAM']
3660
3661                                         self.merged_listbox_items[' '
3662                                             'PARAMETERS'] = ['PARAMETERS']
3662
3663                                         self.merged_listbox_items[' '
3664                                             'DATA'] = ['DATA']
3665
3666                                         self.merged_listbox_items[' '
3667                                             'INPUT'] = ['INPUT']
3667
3668                                         self.merged_listbox_items[' '
3669                                             'TIME'] = ['TIME']
3669
3670                                         self.merged_listbox_items[' '
3671                                             'COMP'] = ['COMP']
3671
3672                                         self.merged_listbox_items[' '
3673                                             'REGION'] = ['REGION']
3673
3674                                         self.merged_listbox_items[' '
3675                                             'WELLS'] = ['WELLS']
3675
3676                                         self.merged_listbox_items[' '
3677                                             'TIME'] = ['TIME']
3677
3678                                         self.merged_listbox_items[' '
3679                                             'COMP'] = ['COMP']
3679
3680                                         self.merged_listbox_items[' '
3681                                             'REGION'] = ['REGION']
3681
3682                                         self.merged_listbox_items[' '
3683                                             'WELLPARAM'] = ['WELLPARAM']
3683
3684                                         self.merged_listbox_items[' '
3685                                             'PARAMETERS'] = ['PARAMETERS']
3685
3686                                         self.merged_listbox_items[' '
3687                                             'DATA'] = ['DATA']
3688
3689                                         self.merged_listbox_items[' '
3690                                             'INPUT'] = ['INPUT']
3690
3691                                         self.merged_listbox_items[' '
3692                                             'TIME'] = ['TIME']
3692
3693                                         self.merged_listbox_items[' '
3694                                             'COMP'] = ['COMP']
3694
3695                                         self.merged_listbox_items[' '
3696                                             'REGION'] = ['REGION']
3696
3697                                         self.merged_listbox_items[' '
3698                                             'WELLS'] = ['WELLS']
3698
3699                                         self.merged_listbox_items[' '
3700                                             'TIME'] = ['TIME']
3700
3701                                         self.merged_listbox_items[' '
3702                                             'COMP'] = ['COMP']
3702
3703                                         self.merged_listbox_items[' '
3704                                             'REGION'] = ['REGION']
3704
3705                                         self.merged_listbox_items[' '
3706                                             'WELLPARAM'] = ['WELLPARAM']
3706
3707                                         self.merged_listbox_items[' '
3708                                             'PARAMETERS'] = ['PARAMETERS']
3708
3709                                         self.merged_listbox_items[' '
3710                                             'DATA'] = ['DATA']
3711
3712                                         self.merged_listbox_items[' '
3713                                             'INPUT'] = ['INPUT']
3713
3714                                         self.merged_listbox_items[' '
3715                                             'TIME'] = ['TIME']
3715
3716                                         self.merged_listbox_items[' '
3717                                             'COMP'] = ['COMP']
3717
3718                                         self.merged_listbox_items[' '
3719                                             'REGION'] = ['REGION']
3719
3720                                         self.merged_listbox_items[' '
3721                                             'WELLS'] = ['WELLS']
3721
3722                                         self.merged_listbox_items[' '
3723                                             'TIME'] = ['TIME']
3723
3724                                         self.merged_listbox_items[' '
3725                                             'COMP'] = ['COMP']
3725
3726                                         self.merged_listbox_items[' '
3727                                             'REGION'] = ['REGION']
3727
3728                                         self.merged_listbox_items[' '
3729                                             'WELLPARAM'] = ['WELLPARAM']
3729
3730                                         self.merged_listbox_items[' '
3731                                             'PARAMETERS'] = ['PARAMETERS']
3731
3732                                         self.merged_listbox_items[' '
3733                                             'DATA'] = ['DATA']
3734
3735                                         self.merged_listbox_items[' '
3736                                             'INPUT'] = ['INPUT']
3736
3737                                         self.merged_listbox_items[' '
3738                                             'TIME'] = ['TIME']
3738
3739                                         self.merged_listbox_items[' '
3740                                             'COMP'] = ['COMP']
3740
3741                                         self.merged_listbox_items[' '
3742                                             'REGION'] = ['REGION']
3742
3743                                         self.merged_listbox_items[' '
3744                                             'WELLS'] = ['WELLS']
3744
3745                                         self.merged_listbox_items[' '
3746                                             'TIME'] = ['TIME']
3746
3747                                         self.merged_listbox_items[' '
3748                                             'COMP'] = ['COMP']
3748
3749                                         self.merged_listbox_items[' '
3750                                             'REGION'] = ['REGION']
3750
3751                                         self.merged_listbox_items[' '
3752                                             'WELLPARAM'] = ['WELLPARAM']
3752
3753                                         self.merged_listbox_items[' '
3754                                             'PARAMETERS'] = ['PARAMETERS']
3754
3755                                         self.merged_listbox_items[' '
3756                                             'DATA'] = ['DATA']
3757
3758                                         self.merged_listbox_items[' '
3759                                             'INPUT'] = ['INPUT']
3759
3760                                         self.merged_listbox_items[' '
3761                                             'TIME'] = ['TIME']
3761
3762                                         self.merged_listbox_items[' '
3763                                             'COMP'] = ['COMP']
3763
3764                                         self.merged_listbox_items[' '
3765                                             'REGION'] = ['REGION']
3765
3766                                         self.merged_listbox_items[' '
3767                                             'WELLS'] = ['WELLS']
3767
3768                                         self.merged_listbox_items[' '
3769                                             'TIME'] = ['TIME']
3769
3770                                         self.merged_listbox_items[' '
3771                                             'COMP'] = ['COMP']
3771
3772                                         self.merged_listbox_items[' '
3773                                             'REGION'] = ['REGION']
3773
3774                                         self.merged_listbox_items[' '
3775                                             'WELLPARAM'] = ['WELLPARAM']
3775
3776                                         self.merged_listbox_items[' '
3777                                             'PARAMETERS'] = ['PARAMETERS']
3777
3778                                         self.merged_listbox_items[' '
3779                                             'DATA'] = ['DATA']
3780
3781                                         self.merged_listbox_items[' '
3782                                             'INPUT'] = ['INPUT']
3782
3783                                         self.merged_listbox_items[' '
3784                                             'TIME'] = ['TIME']
3784
3785                                         self.merged_listbox_items[' '
3786                                             'COMP'] = ['COMP']
3786
3787                                         self.merged_listbox_items[' '
3788                                             'REGION'] = ['REGION']
3788
3789                                         self.merged_listbox_items[' '
3790                                             'WELLS'] = ['WELLS']
3790
3791                                         self.merged_listbox_items[' '
3792                                             'TIME'] = ['TIME']
3792
3793                                         self.merged_listbox_items[' '
3794                                             'COMP'] = ['COMP']
3794
3795                                         self.merged_listbox_items[' '
3796                                             'REGION'] = ['REGION']
3796
3797                                         self.merged_listbox_items[' '
3798                                             'WELLPARAM'] = ['WELLPARAM']
3798
3799                                         self.merged_listbox_items[' '
3800                                             'PARAMETERS'] = ['PARAMETERS']
3800
3801                                         self.merged_listbox_items[' '
3802                                             'DATA'] = ['DATA']
3803
3804                                         self.merged_listbox_items[' '
3805                                             'INPUT'] = ['INPUT']
3805
3806                                         self.merged_listbox_items[' '
3807                                             'TIME'] = ['TIME']
3807
3808                                         self.merged_listbox_items[' '
3809                                             'COMP'] = ['COMP']
3809
3810                                         self.merged_listbox_items[' '
3811                                             'REGION'] = ['REGION']
3811
3812                                         self.merged_listbox_items[' '
3813                                             'WELLS'] = ['WELLS']
3813
3814                                         self.merged_listbox_items[' '
3815                                             'TIME'] = ['TIME']
3815
3816                                         self.merged_listbox_items[' '
3817                                             'COMP'] = ['COMP']
3817
3818                                         self.merged_listbox_items[' '
3819                                             'REGION'] = ['REGION']
3819
3820                                         self.merged_listbox_items[' '
3821                                             'WELLPARAM'] = ['WELLPARAM']
3821
3822                                         self.merged_listbox_items[' '
3823                                             'PARAMETERS'] = ['PARAMETERS']
3823
3824                                         self.merged_listbox_items[' '
3825                                             'DATA'] = ['DATA']
3826
3827                                         self.merged_listbox_items[' '
3828                                             'INPUT'] = ['INPUT']
3828
3829                                         self.merged_listbox_items[' '
3830                                             'TIME'] = ['TIME']
3830
3831                                         self.merged_listbox_items[' '
3832                                             'COMP'] = ['COMP']
3832
3833                                         self.merged_listbox_items[' '
3834                                             'REGION'] = ['REGION']
3834
3835                                         self.merged_listbox_items[' '
3836                                             'WELLS'] = ['WELLS']
3836
3837                                         self.merged_listbox_items[' '
3838                                             'TIME'] = ['TIME']
3838
3839                                         self.merged_listbox_items[' '
3840                                             'COMP'] = ['COMP']
3840
3841                                         self.merged_listbox_items[' '
3842                                             'REGION'] = ['REGION']
3842
3843                                         self.merged_listbox_items[' '
3844                                             'WELLPARAM'] = ['WELLPARAM']
3844
3845                                         self.merged_listbox_items[' '
3846                                             'PARAMETERS'] = ['PARAMETERS']
3846
3847                                         self.merged_listbox_items[' '
3848                                             'DATA'] = ['DATA']
3849
3850                                         self.merged_listbox_items[' '
3851                                             'INPUT'] = ['INPUT']
3851
3852                                         self.merged_listbox_items[' '
3853                                             'TIME'] = ['TIME']
3853
3854                                         self.merged_listbox_items[' '
3855                                             'COMP'] = ['COMP']
3855
3856                                         self.merged_listbox_items[' '
3857                                             'REGION'] = ['REGION']
3857
3858                                         self.merged_listbox_items[' '
3859                                             'WELLS'] = ['WELLS']
3859
3860                                         self.merged_listbox_items[' '
3861                                             'TIME'] = ['TIME']
3861
3862                                         self.merged_listbox_items[' '
3863                                             'COMP'] = ['COMP']
3863
3864                                         self.merged_listbox_items[' '
3865                                             'REGION'] = ['REGION']
3865
3866                                         self.merged_listbox_items[' '
3867                                             'WELLPARAM'] = ['WELLPARAM']
3867
3868                                         self.merged_listbox_items[' '
3869                                             'PARAMETERS'] = ['PARAMETERS']
3869
3870                                         self.merged_listbox_items[' '
3871                                             'DATA'] = ['DATA']
3872
3873                                         self.merged_listbox_items[' '
3874                                             'INPUT'] = ['INPUT']
3874
3875                                         self.merged_listbox_items[' '
3876                                             'TIME'] = ['TIME']
3876
3877                                         self.merged_listbox_items[' '
3878                                             'COMP'] = ['COMP']
3878
3879                                         self.merged_listbox_items[' '
3880                                             'REGION'] = ['REGION']
3880
3881                                         self.merged_listbox_items[' '
3882                                             'WELLS'] = ['WELLS']
3882
3883                                         self.merged_listbox_items[' '
3884                                             'TIME'] = ['TIME']
3884
3885                                         self.merged_listbox_items[' '
3886                                             'COMP'] = ['COMP']
3886
3887                                         self.merged_listbox_items[' '
3888                                             'REGION'] = ['REGION']
3888
3889                                         self.merged_listbox_items[' '
3890                                             'WELLPARAM'] = ['WELLPARAM']
3890
3891                                         self.merged_listbox_items[' '
3892                                             'PARAMETERS'] = ['PARAMETERS']
3892
3893                                         self.merged_listbox_items[' '
3894                                             'DATA'] = ['DATA']
3895
3896                                         self.merged_listbox
```

```

3042                               path_ar_region = os.path.
3043                               join(core_path, 'REGION' + '.parquet')
3043                               x_region = pd.read_parquet(
3044                               path_ar_region)
3045                               components = np.unique(
3046                               x_region['Component'].tolist())
3046                               regions = np.unique(x_region
3047                               ['Region'])
3047                               for region in regions:
3048                                   for component in
3049                                   components:
3050                                       element = ['REGION',
3050                                       region, component]
3050                                       element_shown = ' '
3051                                       element_shown += 'REGION ' + str(region) + ' ' + str(component)
3051                                       element_shown += self.
3052                                       merged_listbox_items[element_shown] = element
3052                                       elif filename == 'TIME':
3053                                           self.merged_listbox_items['
3053 TIME'] = ['TIME']
3054                                       elif filename == 'WELLS':
3055                                           path_ar_wells = os.path.join
3055                                           (core_path, 'WELLS' + '.parquet')
3056                                           x_wells = pd.read_parquet(
3056                                           path_ar_wells)
3057                                           wells_columns = x_wells.
3058                                           columns.tolist()
3059                                           properties_wells = []
3060                                           adapt_wellname = None
3061                                           try:
3062                                               well_names = np.unique(
3062                                               x_wells['nWellName'].tolist())
3063                                               adapt_wellname = ' '
3063                                               adapt_wellname += 'nWellName'
3064                                               except KeyError:
3065                                                   well_names = np.unique(
3065                                                   x_wells['n_well_name'].tolist())
3066                                                   adapt_wellname = ' '
3066                                                   adapt_wellname += 'n_well_name'
3067
3068                               for column_name in
3069                               wells_columns:
3069                               if column_name not in [

```

```

3069 nWell', adapt_wellname, 'nType', 'Connection']:  

3070                                         properties_wells.  

3071                                         append(column_name)  

3072                                         for well in well_names:  

3073                                             well_attributes =  

3074                                                 x_wells.loc[x_wells[adapt_wellname] == well].iloc[0, :]  

3075                                             well_number =  

3076                                                 well_attributes['nWell']  

3077                                             well_type =  

3078                                                 well_attributes['nType']  

3079                                         element = ['WELLS',  

3080                                             well_number, well, well_type]  

3081                                         element_shown = 'WELLS '  

3082                                             + str(well_number) + ' ' + str(well) + ' ' + str(  

3083                                             well_type)  

3084                                         self.  

3085                                         merged_listbox_items[element_shown] = element  

3086                                         values_to_be_inserted = list(self.  

3087                                         merged_listbox_items.keys())  

3088                                         alls = list(self.prep_sim_parameters  

3089                                         .get(0, END))  

3090                                         if alls:  

3091                                             self.prep_sim_parameters.delete(  

3092                                             0, END)  

3093                                         for item in values_to_be_inserted:  

3094                                             self.prep_sim_parameters.insert(  

3095                                             END, item)  

3096                                         if self.xyz.get() != 0:  

3097                                             alls = self.prep_sim_parameters.  

3098                                             get(0, END)  

3099                                         indexdata = alls.index('DATA')  

3100                                         access = (indexdata,)  

3101                                         self.prep_sim_parameters.  

3102                                         selection_set(access)  

3103                                         self.prep_sim_parameters.  

3104                                         event_generate('<<ListboxSelect>>')  

3105                                         core_path = self.simcase_path  

3106                                         path_ar_input = os.path.join(  

3107                                         core_path, 'INPUT' + '.parquet')  

3108                                         path_ar_time = os.path.join(  

3109                                         core_path, 'TIME' + '.parquet')  

3110                                         self.x_input = pd.read_parquet(

```

```

3096 path_ar_input)
3097                     self.x_time = pd.read_parquet(
3098                         path_ar_time)
3099                     ival, jval, kval = (self.x_input['i'],
3100                                         self.x_input['j'], self.x_input['k'])
3101                     imin, imax, jmin, jmax, kmin, kmax =
3102                         (ival.min(), ival.max(), jval.min(), jval.max(), kval.
3103                             min(), kval.max())
3104                     imin, imax, jmin, jmax, kmin, kmax =
3105                         (int(imin), int(imax), int(jmin), int(jmax), int(kmin),
3106                             int(kmax))
3107                     tmin, tmax = (self.x_time['nStep'].min(),
3108                                         self.x_time['nStep'].max())
3109                     tmin, tmax = (int(tmin), int(tmax))
3110                     self.tlimits = [tmin, tmax]
3111                     nsteps, ndays = (self.x_time['nStep']
3112                         .tolist(), self.x_time['nDays'].tolist())
3113
3114                     for i in range(len(nsteps)):
3115                         item = str(nsteps[i])
3116                         days = ndays[i]
3117                         self.browse_days[item] = days
3118                         self.create_slider_widgets(imin,
3119                                         imax, jmin, jmax, kmin, kmax, tmin, tmax)
3120
3121                     if self.xyz.get() != 0:
3122                         self.set_xyz()
3123                     if self.settings_stored:
3124                         self.restore_settings()
3125
3126             except IndexError:
3127                 pass
3128
3129         pass
3130
3131     def get_selected_item(self, event):
3132         '''Get current selected item in listbox.
3133         Prevents data from registering when the
3134         same selection is clicked (ie. same item still
3135         in focus)'''
3136
3137         if current_tab == 'Page One':
3138             global current_selection
3139             w = event.widget
3140             index = 0
3141
3142             try:
3143                 index = w.curselection()[0]
3144                 value = w.get(index)

```

```

3130             if value != current_selection:
3131                 current_selection = value
3132                 values_to_be_inserted = dict_param[
3133                     current_selection][1:]
3134                 if self.doitonce == 0:
3135                     for item in
3136                         values_to_be_inserted:
3137                             self.local_sim_parameters.
3138                             insert(END, item)
3139                             self.doitonce = 1
3140                         else:
3141                             self.properties_available = {}
3142                             # Deleted to refill with new properties
3143                             self.local_sim_parameters.delete
3144                             (0, END) # Can use a separate parameter for the ones
3145                             they still want plotted
3146                             for item in
3147                             values_to_be_inserted: # These are just 'potential
3148                             candidates' for plotting (to be ready
3149                             self.local_sim_parameters.
3150                             insert(END, item) # for the user when he/she needs to
3151                             plot them fast.
3152                             self.doitonce = 0
3153                             core_path = dict_param[value][0]
3154                             path_ar_input = os.path.join(
3155                                 core_path, 'INPUT' + '.parquet')
3156                             path_ar_time = os.path.join(
3157                                 core_path, 'TIME' + '.parquet')
3158                             self.x_input = pd.read_parquet(
3159                                 path_ar_input, columns=['Cell'] + ['i'] + ['j'] + ['k']
3160                                 + ['x'] + ['y'] + ['z'])
3161                             self.x_time = pd.read_parquet(
3162                                 path_ar_time)
3163                             ival, jval, kval = (self.x_input['i']
3164                                 , self.x_input['j'], self.x_input['k'])
3165                             imin, imax, jmin, jmax, kmin, kmax =
3166                             (ival.min(), ival.max(), jval.min(), jval.max(), kval.
3167                             min(), kval.max())
3168                             imin, imax, jmin, jmax, kmin, kmax =
3169                             (int(imin), int(imax), int(jmin), int(jmax), int(kmin),
3170                             int(kmax))
3171                             tmin, tmax = (self.x_time['nStep'].
3172                             min(), self.x_time['nStep'].max())
3173                             tmin, tmax = (int(tmin), int(tmax))
3174                             nsteps, ndays = (self.x_time['nStep']

```

```

3153 ].tolist(), self.x_time['nDays'].tolist())
3154             for i in range(len(nsteps)):
3155                 item = str(nsteps[i])
3156                 days = ndays[i]
3157                 self.browse_days[item] = days
3158                 self.create_slider_widgets(imin,
3159                     imax, jmin, jmax, kmin, kmax, tmin, tmax)
3160             except IndexError:
3161                 pass
3162
3162     def left_range_x(self, val):
3163         w1, w2, label1, label2 = (self.slidex_left, self
3164             .slidex_right, self.slidex_label1, self.slidex_label2)
3164         range_type, value, lower, upper = (['LEFT', 'X']
3164             , int(self.valuex1.get()), int(val), int(label2['text']))
3165
3165         self.range_calculation(ranger=range_type, w1=w1,
3166             w2=w2, label1=label1, label2=label2, single=value,
3166             lower=lower, upper=upper)
3166
3167     def right_range_x(self, val):
3168         w1, w2, label1, label2 = (self.slidex_left, self
3169             .slidex_right, self.slidex_label1, self.slidex_label2)
3169         range_type, value, lower, upper = (['RIGHT', 'X']
3169             , int(self.valuex1.get()), int(label1['text']), int(val))
3170
3170         self.range_calculation(ranger=range_type, w1=w1,
3170             w2=w2, label1=label1, label2=label2, single=value,
3170             lower=lower, upper=upper)
3171
3172     def left_range_y(self, val):
3173         w1, w2, label1, label2 = (self.slidey_left, self
3174             .slidey_right, self.slidey_label1, self.slidey_label2)
3174         range_type, value, lower, upper = (['LEFT', 'Y']
3174             , int(self.valuey1.get()), int(val), int(label2['text']))
3175
3175         self.range_calculation(ranger=range_type, w1=w1,
3175             w2=w2, label1=label1, label2=label2, single=value,
3175             lower=lower, upper=upper)
3176
3177     def right_range_y(self, val):
3178         w1, w2, label1, label2 = (self.slidey_left, self
3179             .slidey_right, self.slidey_label1, self.slidey_label2)
3179         range_type, value, lower, upper = (['RIGHT', 'Y']
3179             , int(self.valuey1.get()), int(label1['text']), int(val))

```

```
3179  ))
3180      self.range_calculation(ranger=range_type, w1=w1,
3181      w2=w2, label1=label1, label2=label2, single=value,
3182      lower=lower, upper=upper)
3183
3184      def left_range_z(self, val):
3185          w1, w2, label1, label2 = (self.slidez_left, self.
3186          .slidez_right, self.slidez_label1, self.slidez_label2)
3187          range_type, value, lower, upper = (['LEFT', 'Z']
3188          , int(self.valuez1.get()), int(val), int(label2['text']))
3189
3190          self.range_calculation(ranger=range_type, w1=w1,
3191          w2=w2, label1=label1, label2=label2, single=value,
3192          lower=lower, upper=upper)
3193
3194      def right_range_z(self, val):
3195          w1, w2, label1, label2 = (self.slidez_left, self.
3196          .slidez_right, self.slidez_label1, self.slidez_label2)
3197          range_type, value, lower, upper = (['RIGHT', 'Z']
3198          , int(self.valuez1.get()), int(label1['text']), int(val))
3199
3200      self.range_calculation(ranger=range_type,
3201      w1=w1, w2=w2, label1=label1, label2=label2, single=value
3202      , lower=lower, upper=upper)
3203
3204      def left_range_time(self, val):
3205          w1, w2, label1, label2 = (self.slidetime_left,
3206          self.slidetime_right, self.slidetime_label1, self.
3207          slidetime_label2)
3208          range_type, value, lower, upper = ('LEFT', int(
3209          self.valuetime1.get()), int(val), int(w2.get()))
3210
3211      self.range_calculation_time(ranger=range_type,
3212      w1=w1, w2=w2, label1=label1, label2=label2, single=value
3213      , lower=lower, upper=upper)
3214
3215      def right_range_time(self, val):
3216          w1, w2, label1, label2 = (self.slidetime_left,
3217          self.slidetime_right, self.slidetime_label1, self.
3218          slidetime_label2)
3219          range_type, value, lower, upper = ('RIGHT', int(
3220          self.valuetime1.get()), int(w1.get()), int(val))
3221
3222      self.range_calculation_time(ranger=range_type,
3223      w1=w1, w2=w2, label1=label1, label2=label2, single=value
3224      , lower=lower, upper=upper)
```

```

3202     def restore_settings(self):
3203         imin, imax = (int(self.last_settings['imin']),
3204                         int(self.last_settings['imax']))
3205         jmin, jmax = (int(self.last_settings['jmin']),
3206                         int(self.last_settings['jmax']))
3207         kmin, kmax = (int(self.last_settings['kmin']),
3208                         int(self.last_settings['kmax']))
3209         tmin, tmax = (int(self.last_settings['tmin']),
3210                         int(self.last_settings['tmax']))
3211         self.slidex_left.set(imin)
3212         self.slidex_right.set(imax)
3213         self.slidey_left.set(jmin)
3214         self.slidey_right.set(jmax)
3215         self.slidez_left.set(kmin)
3216         self.slidez_right.set(kmax)
3217         self.slidetime_left.set(tmin)
3218         self.slidetime_right.set(tmax)
3219
3220     def store_settings(self):
3221         imin, imax = (int(self.slidex_left.get()), int(
3222             self.slidex_right.get()))
3223         jmin, jmax = (int(self.slidey_left.get()), int(
3224             self.slidey_right.get()))
3225         kmin, kmax = (int(self.slidez_left.get()), int(
3226             self.slidez_right.get()))
3227         tmin, tmax = (int(self.slidetime_left.get()),
3228                         int(self.slidetime_right.get()))
3229         self.last_settings['imin'] = imin
3230         self.last_settings['imax'] = imax
3231         self.last_settings['jmin'] = jmin
3232         self.last_settings['jmax'] = jmax
3233         self.last_settings['kmin'] = kmin
3234         self.last_settings['kmax'] = kmax
3235         self.last_settings['tmin'] = tmin
3236         self.last_settings['tmax'] = tmax
3237         self.settings_stored = 1
3238
3239     def freeze_val(self, int_var, label1, label2, w2):
3240         single_value = int(int_var.get())
3241         lower_bound = label1['text']
3242         if single_value == 1:
3243             w2.set(lower_bound)
3244             label2['text'] = lower_bound
3245
3246     def freeze_val_time(self, int_var, label1, label2,

```

```

3238 w2):
3239     single_value, lower_bound, lower = (int(int_var.
3240         get()), label1['text'], None)
3241     for keys in list(self/browse_days.keys()):
3242         if self/browse_days[keys] == lower_bound:
3243             lower = int(keys)
3244             break
3245     if single_value == 1:
3246         w2.set(lower)
3247         label2['text'] = lower_bound
3248
3249     def range_calculation(self, ranger, w1, w2, label1,
3250         label2, single, lower, upper):
3251         if ranger[0] == 'LEFT':
3252             if single == 1:
3253                 w2.set(lower)
3254                 label1['text'] = lower
3255                 label2['text'] = lower
3256             elif lower > upper:
3257                 w1.set(upper)
3258             else:
3259                 label1['text'] = lower
3260         elif ranger[0] == 'RIGHT':
3261             if single == 1:
3262                 if label1['text'] == label2['text']:
3263                     w1.set(upper)
3264                     label1['text'] = upper
3265                     label2['text'] = upper
3266                 else:
3267                     w2.set(lower)
3268                     label2['text'] = lower
3269             elif upper < lower:
3270                 w2.set(lower)
3271             else:
3272                 label2['text'] = upper
3273
3274     def range_calculation_time(self, ranger, w1, w2,
3275         label1, label2, single, lower, upper):
3276         if ranger == 'LEFT':
3277             if single == 1:
3278                 w2.set(lower)
3279                 label1['text'] = self/browse_days[str(
3280                     lower)]
3281                 label2['text'] = self/browse_days[str(
3282                     lower)]

```

```

3278             elif lower > upper:
3279                 w1.set(upper)
3280             else:
3281                 label1['text'] = self/browse_days[str(
3282                     lower)]
3282             elif ranger == 'RIGHT':
3283                 if single == 1:
3284                     if label1['text'] == label2['text']:
3285                         w1.set(upper)
3286                         label1['text'] = self/browse_days[
3287                             str(upper)]
3287                         label2['text'] = self/browse_days[
3288                             str(upper)]
3288             else:
3289                 w2.set(lower)
3290                 label2['text'] = self/browse_days[
3291                     str(lower)]
3291             elif upper < lower:
3292                 w2.set(lower)
3293             else:
3294                 label2['text'] = self/browse_days[str(
3295                     upper)]
3295
3296     def popup(self):
3297         self.w = PopupWindow(self.master)
3298         self.grid_button['state'] = 'disabled'
3299         self.master.wait_window(self.w.top)
3300         self.grid_button['state'] = 'normal'
3301
3302     def grid_size_figures(self): # x = lambda lx: x+i+1
3303         if x % 2 == 0 else x+i
3304         figs = self.figs
3305         self.fig_grid_size = {}
3306         self.grid_dropdown.delete(0, END)
3307         if figs == 1:
3308             chosen_cols, chosen_rows = (1, 1)
3309             self.fig_grid_size['1x1'] = [1, 1]
3310             self.grid_dropdown['values'] = ['1x1']
3311             self.grid_dropdown.current(0)
3312         elif figs:
3313             if figs % 2 == 0:
3314                 even = 1
3315                 factors01, factors02, even = ([], [], 0)
3316                 for i in range(-1, 11, 2):
3317                     if i == -1:

```

```

3317             i = 0
3318             factors01.append([figs, 1])
3319             x = figs + i + even
3320             x_step = x
3321             depth = 1
3322             while x_step % 2 == 0:
3323                 x_step = int(x_step / 2)
3324                 if x_step != 1:
3325                     factors01.append([x_step, 2 ** depth])
3326                     depth += 1
3327
3328         for item in factors01:
3329             new_item = [item[1], item[0]]
3330             if new_item not in factors01:
3331                 factors02.append(new_item)
3332         factors01 = factors01 + factors02
3333         sortlist = []
3334         for item in factors01:
3335             combobox_item = str(item[0]) + 'x' + str(item[1])
3336             x = item[0] + item[1]
3337             x2 = random.uniform(0.10, 0.20)
3338             xnew = round(x + x2, 2)
3339             sortlist.append([xnew, combobox_item])
3340             self.fig_grid_size[combobox_item] = item
3341         newsortlist = sorted(sortlist, key=lambda xl : xl[0])
3342
3343         sorted_combobox_list = []
3344         for element in newsortlist:
3345             sorted_combobox_list.append(element[1])
3346             self.grid_dropdown['values'] =
            sorted_combobox_list
3347             self.grid_dropdown.current(0)
3348
3349     def delete_figures(self, choice):
3350         if self.canvas:
3351             plt.clf()
3352             self.f2_toolkit.destroy()
3353             self.f2_toolkit = Frame(self)
3354             self.f2_toolkit.pack(side=TOP, fill='both',
            expand=False)
3355             if choice == 2:
3356                 self.f2_plot.destroy()

```

```

3357                     self.f2_plot = Frame(self)
3358                     self.f2_plot.pack(side=TOP, padx=10,
3359                                   pady=10, expand=1, fill='both')
3360                     gc.collect()
3361                     self.fig = 0
3362
3362     def create_slider_widgets(self, imin, imax, jmin,
3363                               jmax, kmin, kmax, tmin, tmax):
3363         newvalues = {'imin': imin, 'imax': imax, 'jmin':
3364                      jmin, 'jmax': jmax, 'kmin': kmin, 'kmax': kmax, 'tmin':
3365                      tmin, 'tmax': tmax}
3364         self.simcase_ijkt_count.update(newvalues)
3365
3366         # X-DIRECTION SLIDER
3367         self.slidex_label1 = Label(self.sliders, width=5
3368                                   , text=str(imin), bg='white', relief=SUNKEN)
3368         self.slidex_label1.grid(column=0, row=0, sticky=
3369                                 'nw', padx=3, pady=3, ipady=2)
3369         self.slidex_left = Scale(self.sliders, from_=
3370                                  imin, to=max, orient=HORIZONTAL, showvalue=0, relief=
3371                                  SUNKEN, width=17, command=self.left_range_x)
3370         self.slidex_left.grid(column=1, row=0, sticky='
3371                                 nw', padx=3, pady=3)
3371         self.slidex_right = Scale(self.sliders, from_=
3372                                  imin, to=max, orient=HORIZONTAL, showvalue=0, relief=
3373                                  SUNKEN, width=17, command=self.right_range_x)
3372         self.slidex_right.grid(column=2, row=0, sticky='
3373                                 nw', padx=3, pady=3)
3373         self.slidex_label2 = Label(self.sliders, width=5
3374                                   , text=str(max), bg='white', relief=SUNKEN)
3374         self.slidex_label2.grid(column=3, row=0, sticky=
3375                                 'nw', padx=3, pady=3, ipady=2)
3375         self.valuex1 = IntVar()
3376         self.freeze_x1 = Checkbutton(self.sliders,
3377                                     variable=self.valuex1,
3378                                     command=lambda: self
3379                                         .freeze_val(self.valuex1, self.slidex_label1, self.
3380                                         slidex_label2, self.slidex_right))
3380         self.freeze_x1.grid(column=4, row=0, sticky='nw',
3381                             padx=3, pady=3)
3382         self.slidex_left.set(imin)
3383         self.slidex_right.set(max)
3384         # X-DIRECTION SLIDER
3385
3386
3387
3388
3389
3390
3391
3392
3393

```

```

3384
3385          # Y-DIRECTION SLIDER
3386          self.slidey_label1 = Label(self.sliders, width=5
3387          , text=str(jmin), bg='white', relief=SUNKEN)
3388          self.slidey_label1.grid(column=0, row=1, sticky=
3389          'nw', padx=3, pady=3, ipady=2)
3388          self.slidey_left = Scale(self.sliders, from_=
3389          jmin, to=jmax, orient=HORIZONTAL, showvalue=0, relief=
3389          SUNKEN, width=17, command=self.left_range_y)
3389          self.slidey_left.grid(column=1, row=1, sticky='
3389          nw', padx=3, pady=3)
3390          self.slidey_right = Scale(self.sliders, from_=
3390          jmin, to=jmax, orient=HORIZONTAL, showvalue=0, relief=
3390          SUNKEN, width=17, command=self.right_range_y)
3391          self.slidey_right.grid(column=2, row=1, sticky='
3391          nw', padx=3, pady=3)
3392          self.slidey_label2 = Label(self.sliders, width=5
3392          , text=str(jmax), bg='white', relief=SUNKEN)
3393          self.slidey_label2.grid(column=3, row=1, sticky=
3393          'nw', padx=3, pady=3, ipady=2)
3394          self.valuey1 = IntVar()
3395          self.freezeylevel = Checkbutton(self.sliders,
3395          variable=self.valuey1,
3396                                     command=lambda: self
3396          .freeze_val(self.valuey1, self.slidey_label1, self.
3396          slidey_label2, self.slidey_right))
3397          self.freezeylevel.grid(column=4, row=1, sticky='nw',
3397          padx=3, pady=3)
3398          self.slidey_left.set(jmin)
3399          self.slidey_right.set(jmax)
3400          # Y-DIRECTION SLIDER
3401
3402          # Z-DIRECTION SLIDER
3403          self.slidez_label1 = Label(self.sliders, width=5
3403          , text=str(kmin), bg='white', relief=SUNKEN)
3404          self.slidez_label1.grid(column=0, row=2, sticky=
3404          'nw', padx=3, pady=3, ipady=2)
3405          self.slidez_left = Scale(self.sliders, from_=
3405          kmin, to=kmax, orient=HORIZONTAL, showvalue=0, relief=
3405          SUNKEN, width=17, command=self.left_range_z)
3406          self.slidez_left.grid(column=1, row=2, sticky='
3406          nw', padx=3, pady=3)
3407          self.slidez_right = Scale(self.sliders, from_=
3407          kmin, to=kmax, orient=HORIZONTAL, showvalue=0, relief=
3407          SUNKEN, width=17, command=self.right_range_z)

```

```

3408         self.slidez_right.grid(column=2, row=2, sticky='nw',
3409             padx=3, pady=3)
3410         self.slidez_label2 = Label(self.sliders, width=5
3411             , text=str(kmax), bg='white', relief=SUNKEN)
3412         self.slidez_label2.grid(column=3, row=2, sticky='nw',
3413             padx=3, pady=3, ipady=2)
3414         self.valuez1 = IntVar()
3415         self.freezez1 = Checkbutton(self.sliders,
3416             variable=self.valuez1,
3417                 command=lambda: self
3418                 .freeze_val(self.valuez1, self.slidez_label1, self.
3419                 slidez_label2, self.slidez_right))
3420         self.freezez1.grid(column=4, row=2, sticky='nw',
3421             padx=3, pady=3)
3422         self.slidez_left.set(kmin)
3423         self.slidez_right.set(kmax)
3424         # Z-DIRECTION SLIDER
3425
3426         # TIME SLIDER time
3427         tmin_time, tmax_time = (self.browse_days[str(
3428             tmin)], self.browse_days[str(tmax)])
3429         self.tmin_stored, self.tmax_stored = (tmin, tmax
3430 )
3431         self.slidetime_label1 = Label(self.sliders,
3432             width=5, text=tmin_time, bg='white', relief=SUNKEN)
3433         self.slidetime_label1.grid(column=0, row=3,
3434             sticky='nw', padx=3, pady=3, ipady=2)
3435         self.slidetime_left = Scale(self.sliders, from_=
3436             tmin, to=tmax, orient=HORIZONTAL, showvalue=0, relief=
3437             SUNKEN, width=17, command=self.left_range_time)
3438         self.slidetime_left.grid(column=1, row=3, sticky=
3439             'nw', padx=3, pady=3)
3440         self.slidetime_right = Scale(self.sliders, from_=
3441             tmin, to=tmax, orient=HORIZONTAL, showvalue=0, relief=
3442             SUNKEN, width=17, command=self.right_range_time)
3443         self.slidetime_right.grid(column=2, row=3,
3444             sticky='nw', padx=3, pady=3)
3445         self.slidetime_label2 = Label(self.sliders,
3446             width=5, text=tmax_time, bg='white', relief=SUNKEN)
3447         self.slidetime_label2.grid(column=3, row=3,
3448             sticky='nw', padx=3, pady=3, ipady=2)
3449         self.valuetime1 = IntVar()
3450         self.freeze timel = Checkbutton(self.sliders,
3451             variable=self.valuetime1,
3452                 command=lambda:

```

```

3432 self.freeze_val_time(self.valuetime1, self.
3433     slidetime_label1, self.slidetime_label2, self.
3434     slidetime_right))
3435         self.freeze_time1.grid(column=4, row=3, sticky='
3436             nw', padx=3, pady=3)
3437             self.slidetime_left.set(tmin)
3438             self.slidetime_right.set(tmax)
3439             # TIME SLIDER
3440
3441     def find_row_col(self, identifier, user_rows,
3442         user_cols):
3443         ni, chosen_cols, chosen_rows = (identifier,
3444             user_cols, user_rows)
3445         row, col = (0, 0)
3446         if ni > chosen_cols * chosen_rows:
3447             return None, None
3448         else:
3449             if ni <= chosen_cols:
3450                 row = 0
3451                 col = ni - 1
3452             elif ni > chosen_cols:
3453                 row = 0
3454                 nb = ni
3455                 while nb not in list(range(1,
3456                     chosen_cols + 1)):
3457                     nb = nb - chosen_cols
3458                     row += 1
3459                     col = nb - 1
3460             return row, col
3461
3462 class PageTwo(tk.Frame):
3463     def __init__(self, parent, controller):
3464         self.controller = controller
3465         self.parent = parent
3466         tk.Frame.__init__(self, parent)
3467         label = tk.Label(self, text='Page Two', font=
3468             LARGE_FONT)
3469         label.pack(padx=10, pady=10)
3470
3471 class PageThree(tk.Frame):
3472     def __init__(self, parent, controller):
3473         self.controller = controller
3474         self.parent = parent
3475         tk.Frame.__init__(self, parent)

```

```
3470         label = ttk.Label(self, text='Page One...', font=
3471                             LARGE_FONT)
3472
3473 if __name__ == '__main__':
3474     app = SimPlotJIN()
3475     app.protocol('WM_DELETE_WINDOW', app.on_closing)
3476     app.mainloop()
3477
3478
3479
```