```
 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,
    n_title, n_temp, n_sim, n_ver, n_file, n_file_type,
    pre_line, n_steps, keys, \
 92          check_grid01, pre_pre_line, df_grid_data,
    blacklist, x_check, y_check, z_check, k_check, ln,
    cells_grid, \
 93          cells_col, pre_pre_pre_line = ([] for ti in range
    (31))
 94
 95          add_folder, name_of_file, core_path, new_path,
    select_folder, print_now, keys2, comp_temp, reg_temp,
    well_temp, \
 96          n_pressure, ndaysnow, ndates, complist,
    well_list_w, path_comp, path_reg, path_well, comp_list,
    temp_list, templistr, \
 97          temp_list_w, print_now_r, reg_list, n_reg,
    current, wellreads, read_now, current_f, nwells,
    n_well_name, ntypes, n_pv_well, \
 98          n_well_temp, well_temp_itemsize, final_line,
    x_current, sum_qst, path_data, cell_index = ([] for ti in
     range(40))
 99
100
101          normal_length = 0
102
103          n_summary, getcomps, store_once_one,
    store_data_once, firstvalues, start_main, x, check_title,
     check_grid02, check_grid03, \
104          key_search, indexing_go, = [0] * 12
105
106          start = "01-Jan-2010 00:00:00"
107          i_day = datetime.datetime.strptime(start, "%d-%b
    -%Y %H:%M:%S")
108
109          temp_dict, cells_dict, grid_dim_dict,
    temp_storage, dict_comp, keys2_dict, summary_info,
    store_once, col_width, \
110          current_dict, store_col, col_width_data,
    cells_data = ({} for tj in range(13))
111
112          well_param = 0
113
114          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
    ', 'DY', 'DZ', 'X', 'Y', 'Z']
132             cell_list_dict = {}
133             for nok in cells_col_names:
134                 cell_list_dict[nok] = []
135 
136             if option == "single":
137                 ncell = (cellj - 1) * imax + (cellk - 1)
    * jmax * kmax + celli
138                 return ncell
139             elif option == "full":
140                 for nk in range(1, kmax + 1):
141                     for nj in range(1, jmax + 1):
142                         for ni in range(1, imax + 1):
143                             ncell = (nj - 1) * imax + (nk
     - 1) * jmax * imax + ni
144                             cell_list_dict['Cell'].append
    (int(ncell))
145                             cell_list_dict['i'].append(
    int(ni))
146                             cell_list_dict['j'].append(
    int(nj))
147                             cell_list_dict['k'].append(
    int(nk))
148                 cell_list_dict['NX'] = imax
149                 cell_list_dict['NY'] = jmax
150                 cell_list_dict['NZ'] = kmax
151                 return cell_list_dict
152 
```

```
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] *
     3
169 
170             ivalue = list(np.unique(all_cells['i']))
171             jvalue = list(np.unique(all_cells['j']))
172             kvalue = list(np.unique(all_cells['k']))
173 
174             length_z, dzk_prev = [0]*2
175             for kl in kvalue:
176                 dzk = float(dim_dz[kl-1])
177                 length_z = length_z + float(0.5 * (
    dzk_prev + dzk))
178                 length_y, dyj_prev = [0] * 2
179                 for jl in jvalue:
180                     dyj = float(dim_dy[jl-1])
181                     length_y = length_y + float(0.5 * (
    dyj_prev + dyj))
182                     length_x, dxi_prev = [0]*2
183                     for il in ivalue:
184                         dxi = float(dim_dx[il-1])
185                         length_x = length_x + float(0.5 *
     (dxi_prev + dxi))
186                         all_cells['DX'].append(dxi)
187                         all_cells['DY'].append(dyj)
188                         all_cells['DZ'].append(dzk)
189                         all_cells['X'].append(length_x)
190                         all_cells['Y'].append(length_y)
191                         all_cells['Z'].append(length_z)
192                         dxi_prev = dxi
193                     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
    of XKEYS you want removed |"""
200
201             length = len(args)
202             if length != 2:
203                 return None
204
205             keys_original = args[0]
206             keys_remove = args[1]
207             keys_remaining = []
208
209             for m in keys_original:
210                 if any(m in s for s in keys_remove):
211                     pass
212                 else:
213                     keys_remaining.append(m)
214
215             return keys_remaining
216
217         def create_directory(*args):
218             """You must specify: | Core Path | New Path
    (+1 increment) |"""
219
220             length = len(args)
221             if length != 2:
222                 return None
223
224             core = args[0]
225             new = args[1]
226
227             if os.path.exists(core):
228                 if os.path.exists(new):
229                     pass
230                 else:
231                     os.mkdir(new)
232             else:
233                 os.mkdir(core)
234                 os.mkdir(new)
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
    , name + '.parquet')
258                 table = pa.Table.from_pandas(dataframe)
259                 write_id = id_unique
260                 if write_id is None:
261                     write_id = pq.ParquetWriter(path,
    table.schema)
262                 write_id.write_table(table=table)
263                 id_unique = write_id
264                 return id_unique
265 
266             elif store_method == 'hdf5':
267                 path = os.path.join(path_original, folder
    , name + '.h5')
268                 if no_col_restraint == 1:
269                     dataframe.to_hdf(path, key=name,
    format='table', append=True)
270                 elif no_col_restraint == 0:
271                     dataframe.to_hdf(path, key=name,
    format='table', append=True, data_columns=True, complevel
    =9, complib='blosc',
272                                      min_itemsize=
    column_width)
273                 return None
274 
```

```
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
    , 2)
308                 # print('line: ' + str(lineNumber
    ) + ' ' + str(line))
309                 if skip_line == 0:
310                     global global_x_label
311                     global_x_label.config(text=str(x) +
    ' %')
312                     n_count2 = 5
313                 else:
314                     skip_line -= 1
315                 if not data_input:
316                     if not close01 and 'Input file' in
```

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

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

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

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

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

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

```
496 '.', '')
497                             try:
498                                 T_loc = current.index('
    temperature')
499                                 try:
500                                     n_well_temp = float(
    current[T_loc + 2])
501                                 except ValueError:
502                                     n_well_temp = float(
    current[T_loc + 1].replace('=', ''))
503                             except ValueError:
504                                 n_well_temp = 999  #
    Indicates well has been closed
505                             wellreads = 0
506                         elif "Connection" in pre_pre_line and
     wellreads == 0:
507                             current = pre_pre_pre_line.
    replace('block', '')
508                             current = current.split()
509                             current_n = pre_pre_line.split()[
    1:]
510                             current_f = []
511                             well_param_names = []
512                             for u in range(len(current)):
513                                 if well_param == 0:
514                                     storethis = str(current[u
    ]) + ' ' + str(current_n[u])
515                                     well_param_names.append(
    storethis)
516                                 current_f0 = current[u].
    replace('/', 'per') + current_n[u].replace('/', 'per')
517                                 current_f0 = current_f0.
    translate({ord(i): None for i in ['(', ')', '³']})
518                                 current_f.append(current_f0)
519                             if well_param == 0:
520                                 well_param_names = ['nPVwell'
    , 'Temperature', 'Connection'] + well_param_names
521                                 well_col = pd.DataFrame(pd.
    Series(well_param_names))
522                                 well_col.set_index(well_col.
    columns[0])
523                                 name_of_file = 'WELLPARAM'
524
525                                 write_id_parquet =
    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 =
    write_id_parquet
528                                 well_param = 1
529                             read_now = 0
530                         while wellreads == 0 and read_now ==
    0:  # and line !='\n':
531                             current = []
532                             temp_list_w = []
533                             if "---" in line:
534                                 current = pre_line.split()
535                                 for z in current[1:]:
536                                     # print('current: ' + str
    (z))
537                                     if normal_length == 0:
538                                         normal_length = len(
    well_param_names) - 2
539                                     current_length = len(
    current)
540                                     if normal_length !=
    current_length:
541                                         if '-' in z and not '
    e-' in z:
542                                             pass
543                                             # print('yup,
    there is extra -, and no e-') # could be just negative
    number
544                                         elif '-' in z and 'e
    -' in z:
545                                             split_it = z.
    split('e-')
546                                             base_number =
    split_it[0]
547                                             if '-' in
    base_number:
548                                                 new_split =
    base_number.split('-')
549                                                 if len(
    new_split) > 1:
550
    first_number = float(new_split[0])
551
    temp_list_w.append(first_number)
```

```
552
    second_number = float('-' + new_split[1] + 'e-' +
    split_it[1])
553
    temp_list_w.append(second_number)
554                                     else:
555                                         z = float(z)
556                                         temp_list_w.append(z)
557                                 string01 = current[0]
558                                 current = [pd.to_datetime(
    ndates)] + [nwells] + [n_well_name] + [ntypes] + [
    n_pv_well] + [n_well_temp] + [string01] + temp_list_w
559                             elif 'Total' in pre_pre_line:
560                                 name_of_file = 'WELLS'
561                                 if store_once_one == 0:
562                                     well_list_w = ['Time'] +
    ['nWell'] + ['nWellName'] + ['nType'] + ['nPVwell'] + ['
    nWellTemp'] + ['Connection'] + current_f
563                                     for e in well_list_w[1:]:
564                                         col_width[str(e)] =
    50
565                                     store_once_one = 1
566                                 well_temp = pd.DataFrame(
    well_temp, columns=well_list_w)
567                                 well_temp = well_temp.
    set_index(well_temp.columns[0])
568
569                                 write_id_parquet =
    write_to_file(store_method, well_temp, core_path,
    select_folder, name_of_file, write_id_wells, col_width)
570                                 if write_id_parquet is not
    None:
571                                     write_id_wells =
    write_id_parquet
572                                 well_temp = []
573                                 wellreads, read_now, sum_qst
    = (1, 1, 1)
574                                 break
575                             if current:
576                                 well_temp.append(current)
577                             break
578                         if wellreads == 1 and read_now == 1:
579                             if ("Summary" in line and "
    Timestep" in line) or 'CPU summary report' in line:
580                                 name_of_file, cell_number = (
```

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

```
607                                     if len(store_col) !=
    len(keys):
608                                         x_current =
    remove.translate({ord(i): None for i in ['_', '[', ']',
    ' ', '(', ')', '-', '/', '%', '°']})
609                                         store_col[remove]
     = x_current
610                                     else:
611                                         if
    store_parameters == 0:
612                                             test_col = pd
    .DataFrame(pd.Series(store_col))
613                                             test_col.
    set_index(test_col.columns[0])
614                                             name_of_file
    = 'PARAMETERS'
615
616
    write_id_parquet = write_to_file(store_method, test_col,
    core_path, select_folder, name_of_file, write_id_param)
617                                             if
    write_id_parquet is not None:
618
    write_id_param = write_id_parquet
619
    store_parameters = 1
620                                         x_current =
    store_col[remove]
621                                 elif xi in line and
    start_main == 1:
622                                     start_main = 0
623                                     break
624
625                                 while start_main == 1:
626                                     current = line.split()
627                                     if x_current not in
    current_dict:
628                                         current_dict[
    x_current] = list(map(float, current))
629                                     elif current:
630                                         current_dict[
    x_current] = current_dict[x_current] + list(map(float,
    current))
631                                     break
632                 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() -
    start_time
653         # print('Time elapsed (hh:mm:ss.ms) {}'.format(
    time_elapsed))
654 
655 
656 # END OF IMPORT / CONVERT FUNCTION
657 #          -------------
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,
    ipady=1)
674         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)
      # Calls the class
715             self.frames[page] = frame
716             self.nb.add(frame, text=labels[i])
717 
718         def prep_local_param(event):
719             selection = event.widget.select()
720             tab = event.widget.tab(selection, 'text')
721             global current_tab
722             current_tab = tab
723             current = global_sim_data
724             if tab == 'Page One' and current:
725                 alls = list(global_sim_data_listbox.get(0
    , END))
726                 for num in reversed(range(len(alls))):
727                     keys = alls[num]
728                     if 'DATA' not in current[keys] or '
    PARAMETERS' not in current[keys]:
729                         global_sim_data_listbox.delete(
    num)
730                     else:
731                         store_path = current[keys][0]
732                         path_param = os.path.join(
    store_path, 'PARAMETERS' + '.parquet')
733                         avail_param = pd.read_parquet(
    path_param)
734                         if 'WELLPARAM' in current[keys]:
735                             path_wellparam = os.path.join
    (store_path, 'WELLPARAM' + '.parquet')
736                             avail_wellparam = pd.
    read_parquet(path_wellparam)
737                             store_wellparam =
    avail_wellparam.iloc[:,0].tolist()
738                             path_wells = os.path.join(
    store_path, 'WELLS' + '.parquet')
739                             wells_col = pd.read_parquet(
    path_wells).columns.tolist()
740                             temp_dict = {}
741                             shown = store_wellparam
742                             hidden = wells_col[3:]
743                             for ik in list(range(len(
    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,
    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,
    pady=3)
795         f2.pack(side=TOP, expand=0, fill='both', padx=3,
    pady=3)
796         f3.pack(side=TOP, expand=1, fill='both', padx=3,
    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..',
    command=lambda: self.load_file(f1))
803         button_import.grid(column=0, row=0, sticky='nw',
    padx=3, pady=3)
804         button_save = ttk.Button(f1, text='Save to..',
    command=lambda: self.save_file(f1))
805         button_save.grid(column=0, row=1, sticky='nw',
    padx=3, pady=3)
806         self.button_convert = ttk.Button(f1, text='
    Convert', command=lambda: self.convert_file(f1))
807         self.button_convert.grid(column=2, row=0)
808         button_read = ttk.Button(f1, text='Read from..',
    command=lambda: self.read_folder(f1, f2a.list_parent))
809         button_read.grid(column=0, row=2, sticky='nw',
    padx=3, pady=3)
810         button_add = ttk.Button(f2b, text='Add', command=
    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',
    command=lambda: self.remove_name(child=f2c.list_child))
813         button_del.pack()
814 
815         f1.label = Label(f1, text=self.filename, width=1,
     relief=SUNKEN)
816         f1.label.grid(column=1, row=0, padx=3, pady=3,
    ipadx=250, ipady=2)
817         f1.label4 = Label(f1, text=self.home_location,
    width=1, relief=SUNKEN)
818         f1.label4.grid(column=1, row=1, padx=3, pady=3,
    ipadx=250, ipady=2)
819         f1.label5 = Label(f1, text=self.read_location,
    width=1, relief=SUNKEN)
820         f1.label5.grid(column=1, row=2, padx=3, pady=3,
    ipadx=250, ipady=2)
821         global global_x_label
822         global_x_label = Label(f1, text='0.00 %', width=5
    )
823         global_x_label.grid(column=3, row=0, padx=3, pady
    =3, ipadx=15, ipady=2)
824 
825         f2a.label6 = Label(f2a, text='  Available
    Simulation Cases  ')
826         f2a.label6.grid(column=0, row=0, padx=3, pady=3)
827         f2a.list_parent = Listbox(f2a, height=10,
    selectmode=EXTENDED, relief=SUNKEN)
828         f2a.list_parent.grid(column=0, row=1, padx=3,
    pady=3)
829         f2c.label7 = Label(f2c, text='  Cases available
    for plotting  ')
830         f2c.label7.grid(column=0, row=0, padx=3, pady=3)
831         f2c.list_child = Listbox(f2c, height=10,
    selectmode=EXTENDED, relief=SUNKEN)
832         f2c.list_child.grid(column=0, row=1, padx=3, pady
    =3)
833 
834         self.local_simulations(path_to_check=self.
    read_location, f2a_listbox=f2a.list_parent)
835 
836     def local_simulations(self, path_to_check,
    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', 'PARAMETERS', 'REGION', 'TIME', 'WELLS', '
    WELLPARAM']:
843                     break
844                 elif current[1] == 'parquet':
845                     store_list.append(current[0])
846             if store_list:
847                 sim_folder = os.path.basename(path)
848                 combined = [path] + store_list
849                 f2a_listbox.insert(END, sim_folder)
850                 self.simulation_data_found[sim_folder] =
    combined
851 
852     def add_name(self, parent, child):
853         cursors = parent.curselection()
854         alls = list(child.get(0, END))
855         global global_sim_data_listbox
856         for item in list(cursors):
857             x_add = parent.get(item)
858             self.simulation_data_to_plot[x_add] = self.
    simulation_data_found[x_add]
859             if x_add not in alls:
860                 child.insert(END, x_add)
861                 global_sim_data_listbox.insert(END, x_add
    )
862         global global_sim_data
863         global_sim_data = self.simulation_data_to_plot
864 
865     def remove_name(self, child):
866         cursors = child.curselection()
867         global global_sim_data_listbox
868         for item in reversed(cursors):
869             x_del = child.get(item)
870             self.simulation_data_to_plot.pop(x_del, None)
871             child.delete(item)
872             global_sim_data_listbox.delete(item)
873 
874     def load_file(self, cont):
875         self.filename = askopenfilename(title='Select .
    out file', filetypes=(('OUT File', '*.out'),))
876         if self.filename:
```

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

```
912 pady=2, fill='y')
913         global_sim_data_listbox.bind('<<ListboxSelect>>',
     self.get_selected_item_prep)
914         self.prep_sim_parameters = Listbox(self.f1_input,
     height=5, selectmode=SINGLE, relief=SUNKEN, width=24,
    exportselection=False)
915         self.prep_sim_parameters.pack(side=LEFT, padx=2,
    pady=2, fill='y')
916         self.prep_sim_parameters.bind('<<ListboxSelect>>'
    , self.get_folded_properties)
917         self.local_sim_parameters = Listbox(self.f1_input
    , height=5, selectmode=EXTENDED, relief=SUNKEN)
918         self.local_sim_parameters.pack(side=LEFT, padx=2,
     pady=2, fill='y')
919         self.local_sim_parameters.bind('<<ListboxSelect
    >>', 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='
    Hold3', variable=self.hold3, onvalue=0, offvalue=1)
931         self.hold_choice3.pack()
932         self.grid_dropdown_font = ttk.Combobox(self.f3b,
    height=1, width=7, state='readonly')
933         self.grid_dropdown_font.pack()
934         self.label_font = Label(self.f3b, text='Font size
    : ' + str(self.fontsize), height=1, width=10)
935         self.label_font.pack()
936         self.grid_dropdown_font['values'] = list(range(1,
    100+1,1))
937         self.grid_dropdown_font.current(self.fontsize-1)
938         self.grid_dropdown_font.bind('<<ComboboxSelected
    >>', self.get_fontsize)
939         self.button_add_plots = ttk.Button(self.f3b, text
    ='Add', width=10, command=self.add_to_plot_list)
940         self.button_add_plots.pack()
941         self.button_remove_plots = ttk.Button(self.f3b,
    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.
    listboxes_frame, bg='blue')
969         self.z_listbox_frame.pack(side=TOP, padx=2, pady
    =2, fill='both', expand=1)
970         self.pageone_listbox_plot = Listbox(self.
    z_listbox_frame, height=5, selectmode=EXTENDED, relief=
    SUNKEN)
971         self.pageone_listbox_plot.pack(side=LEFT, padx=2
    , pady=2, fill='both', expand=1)
972         self.pageone_listbox_plot.bind('<<ListboxSelect
    >>', self.get_plot_titles)
973 
974         self.checkmarks = Frame(self.f1_input, bg='black
    ')
975         self.checkmarks.pack(side=LEFT, expand=0, fill='
    both')
976         self.checkmarks_a = Frame(self.checkmarks, bg='
    green')
977         self.checkmarks_a.pack(side=TOP, expand=0, fill=
    'both')
978         self.checkmarks_b = Frame(self.checkmarks, bg='
    white')
979         self.checkmarks_b.pack(side=TOP, expand=0, fill=
    'both')
980         self.checkmarks_c = Frame(self.checkmarks, bg='
    orange')
981         self.checkmarks_c.pack(side=TOP, expand=1, fill=
    'both')
982         self.checkmarks_d = Frame(self.checkmarks, bg='
    orange')
983         self.checkmarks_d.pack(side=TOP, expand=1, fill=
    'both')
984         self.figs = 0
985         self.label_figs = Label(self.checkmarks_a, text=
    'Figures: ' + str(self.figs), height=1, relief=SUNKEN,
    width=9)
986         self.label_figs.grid(column=0, row=0, sticky='nw
    ', padx=3, pady=3, ipady=2)
987         self.grid_dropdown = ttk.Combobox(self.
    checkmarks_a, height=1, width=4)
988         self.grid_dropdown.grid(column=1, row=0, sticky=
    '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='
     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',
     padx=3, pady=3)
1033 
1034         self.slide_and_gelmod = Frame(self.f1_input, bg=
     'black')
1035         self.slide_and_gelmod.pack(side=LEFT, expand=0,
     fill='both')
1036         self.sliders = Frame(self.slide_and_gelmod, bg='
     red')
1037         self.sliders.pack(side=TOP, expand=0, fill='both
     ')
1038         self.prep_gelmods = Frame(self.slide_and_gelmod,
      bg='grey')
1039         self.prep_gelmods.pack(side=TOP, expand=1, fill=
     'both')
1040         self.gelmods = Frame(self.prep_gelmods, bg='grey
     ')
1041         self.gelmods.pack(side=LEFT, expand=1, fill='
     both')
1042         self.buttons_gelmods = Frame(self.prep_gelmods,
     bg='grey')
1043         self.buttons_gelmods.pack(side=LEFT, expand=1,
     fill='both')
1044 
1045         # Component (1) - Na (ppm)
1046         self.comp1 = Text(self.gelmods, height=1, width=
     6)
1047         self.comp1mid = Text(self.gelmods, height=1,
     width=5)
1048         self.comp1end = Text(self.gelmods, height=1,
     width=6)
1049         self.comp1_label = Label(self.gelmods, text='Na
      (ppm):', height=1, relief=SUNKEN, width=9)
1050         self.comp1_label2 = Label(self.gelmods, text=':'
     , height=1)
1051         self.comp1_label3 = Label(self.gelmods, text=':'
     , height=1)
1052 
1053         self.comp1_label.grid(column=0, row=0, sticky='
     nw', padx=3, pady=3)
1054         self.comp1.grid(column=1, row=0, sticky='nw',
     padx=3, pady=3)
1055         self.comp1_label2.grid(column=2, row=0, sticky='
     nw', padx=3, pady=3)
```

```
1056            self.comp1mid.grid(column=3, row=0, sticky='nw',
      padx=3, pady=3)
1057            self.comp1_label3.grid(column=4, row=0, sticky='
     nw', padx=3, pady=3)
1058            self.comp1end.grid(column=5, row=0, sticky='nw',
      padx=3, pady=3)
1059
1060            # Component (2) - Ca (ppm)
1061            self.comp2 = Text(self.gelmods, height=1, width=
     6)
1062            self.comp2mid = Text(self.gelmods, height=1,
     width=5)
1063            self.comp2end = Text(self.gelmods, height=1,
     width=6)
1064            self.comp2_label = Label(self.gelmods, text='Ca
      (ppm):', height=1, relief=SUNKEN, width=9)
1065            self.comp2_label2 = Label(self.gelmods, text=':'
     , height=1)
1066            self.comp2_label3 = Label(self.gelmods, text=':'
     , height=1)
1067
1068            self.comp2_label.grid(column=0, row=1, sticky='
     nw', padx=3, pady=3)
1069            self.comp2.grid(column=1, row=1, sticky='nw',
     padx=3, pady=3)
1070            self.comp2_label2.grid(column=2, row=1, sticky='
     nw', padx=3, pady=3)
1071            self.comp2mid.grid(column=3, row=1, sticky='nw',
      padx=3, pady=3)
1072            self.comp2_label3.grid(column=4, row=1, sticky='
     nw', padx=3, pady=3)
1073            self.comp2end.grid(column=5, row=1, sticky='nw',
      padx=3, pady=3)
1074
1075            # Component (3) - T (°C)
1076            self.comp3 = Text(self.gelmods, height=1, width=
     6)
1077            self.comp3mid = Text(self.gelmods, height=1,
     width=5)
1078            self.comp3end = Text(self.gelmods, height=1,
     width=6)
1079            self.comp3_label = Label(self.gelmods, text='
     Temp (°C):', height=1, relief=SUNKEN, width=9)
1080            self.comp3_label2 = Label(self.gelmods, text=':'
     , height=1)
```

```
1081            self.comp3_label3 = Label(self.gelmods, text=':'
    , height=1)
1082
1083            self.comp3_label.grid(column=0, row=2, sticky='
    nw', padx=3, pady=3)
1084            self.comp3.grid(column=1, row=2, sticky='nw',
    padx=3, pady=3)
1085            self.comp3_label2.grid(column=2, row=2, sticky='
    nw', padx=3, pady=3)
1086            self.comp3mid.grid(column=3, row=2, sticky='nw',
     padx=3, pady=3)
1087            self.comp3_label3.grid(column=4, row=2, sticky='
    nw', padx=3, pady=3)
1088            self.comp3end.grid(column=5, row=2, sticky='nw',
     padx=3, pady=3)
1089
1090            # self.comp1button.grid(column=0, row=0, sticky
    ='nw', padx=3, pady=3)
1091            self.comp1button = ttk.Button(self.
    buttons_gelmods, text='Set', command=lambda: self.
    set_gelmod())
1092            self.comp1button.pack(side=TOP, expand=1, fill='
    both', padx=3, pady=1)
1093            self.comp1button2 = ttk.Button(self.
    buttons_gelmods, text='Reset', command=lambda: self.
    reset_gelmod())
1094            self.comp1button2.pack(side=TOP, expand=1, fill=
    'both', padx=3, pady=1)
1095            self.comp1button3 = ttk.Button(self.
    buttons_gelmods, text='Plot gelmod', command=lambda:
    self.plot_gelmod())
1096            self.comp1button3.pack(side=TOP, expand=1, fill=
    'both', padx=3, pady=1)
1097
1098            # SLIDERS
1099            self.slidex_label1, self.slidex_left, self.
    slidex_right, self.slidex_label2, self.valuex1, self.
    freezex1 = [None] * 6
1100            self.slidey_label1, self.slidey_left, self.
    slidey_right, self.slidey_label2, self.valuey1, self.
    freezey1 = [None] * 6
1101            self.slidez_label1, self.slidez_left, self.
    slidez_right, self.slidez_label2, self.valuez1, self.
    freezez1 = [None] * 6
1102            self.slidetime_label1, self.slidetime_left, self
```

```
1102 .slidetime_right, self.slidetime_label2, self.valuetime1
     , self.freezetime1 = [None] * 6
1103         self.last_settings = {}
1104         self.last_settings_old = {}
1105         for dim in ['imin', 'imax', 'jmin', 'jmax', '
     kmin', 'kmax', 'tmin', 'tmax']:
1106             self.last_settings[dim] = 1
1107         self.x_input, self.x_time = (None, None)
1108         # SLIDERS
1109 
1110         self.f6 = Frame(self.f1_input, bg='white')
1111         self.f6.pack(side=LEFT, expand=1, fill='both')
1112         self.plotlabels = Frame(self.f6)
1113         self.plotlabels.pack(side=TOP, expand=1, fill='
     both')
1114         self.plotlabels_toppart = Frame(self.plotlabels,
      bg='black')
1115         self.plotlabels_toppart.pack(side=TOP, expand=0,
      fill='x')
1116         self.plotlabels_labels = Frame(self.
     plotlabels_toppart, bg='red')
1117         self.plotlabels_labels.pack(side=LEFT, expand=0,
      fill='both')
1118         self.plotlabels_buttons = Frame(self.
     plotlabels_toppart)
1119         self.plotlabels_buttons.pack(side=LEFT, expand=1
     , fill='both')
1120         self.ptitle_label = Label(self.plotlabels_labels
     , text='Plot title: ', height=1, relief=SUNKEN, width=9)
1121         self.pxtitle_label = Label(self.
     plotlabels_labels, text='X label: ', height=1, relief=
     SUNKEN, width=9)
1122         self.pytitle_label = Label(self.
     plotlabels_labels, text='Y label: ', height=1, relief=
     SUNKEN, width=9)
1123         self.ptitle = Entry(self.plotlabels_labels,
     width=40)
1124         self.pxtitle = Entry(self.plotlabels_labels,
     width=40)
1125         self.pytitle = Entry(self.plotlabels_labels,
     width=40)
1126         self.ptitle_label.grid(column=0, row=0, sticky='
     nw', padx=3, pady=3)
1127         self.ptitle.grid(column=1, row=0, sticky='nw',
     padx=3, pady=3)
```

```
1128            self.title_set = ttk.Button(self.
    plotlabels_buttons, text='Set', command=lambda: self.
    set_plot_labels())
1129            self.title_reset = ttk.Button(self.
    plotlabels_buttons, text='Reset', command=lambda: self.
    reset_plot_labels())
1130            self.title_set.pack(anchor='nw', expand=1, fill=
    'y', padx=3, pady=3)
1131            self.title_reset.pack(anchor='nw', expand=1,
    fill='y', padx=3, pady=3)
1132            self.pxtitle_label.grid(column=0, row=1, sticky=
    'nw', padx=3, pady=3)
1133            self.pxtitle.grid(column=1, row=1, sticky='nw',
    padx=3, pady=3)
1134            self.pytitle_label.grid(column=0, row=2, sticky=
    'nw', padx=3, pady=3)
1135            self.pytitle.grid(column=1, row=2, sticky='nw',
    padx=3, pady=3)
1136
1137            button1 = ttk.Button(self.f1_input, text='Plot
    it', command=self.plot_graphv2)
1138            button1.pack(padx=3, pady=3)
1139            button2 = ttk.Button(self.f1_input, text='Delete
     all', command=lambda: self.delete_figures(2))
1140            button2.pack(padx=3, pady=3)
1141            button3 = ttk.Button(self.f1_input, text='Save
    setup', command=lambda: self.store_settings())
1142            button3.pack(padx=3, pady=3)
1143            button4 = ttk.Button(self.f1_input, text='
    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          self.current_selection_v2 = None
1161          self.merged_listbox_items = {}
1162          self.simcase_child = None
1163          self.simcase = None
1164          self.simcase_path = None
1165          self.data_conversion = {}
1166          self.plot_id = None
1167          self.plot_rdy = {}
1168          self.plot_x = {}
1169          self.plot_y = {}
1170          self.simcase_ijkt_count = {}
1171          self.plot_id_old = {}
1172          self.plot_id_hist = []
1173          self.settings_stored = 0
1174          self.tlimits = []
1175
1176          self.comp_na_start = 0
1177          self.comp_na_mid = 2000
1178          self.comp_na_end = 16000
1179          self.comp1.insert(END,self.comp_na_start)
1180          self.comp1mid.insert(END,self.comp_na_mid)
1181          self.comp1end.insert(END,self.comp_na_end)
1182
1183          self.comp_ca_start = 0
1184          self.comp_ca_mid = 50
1185          self.comp_ca_end = 500
1186          self.comp2.insert(END,self.comp_ca_start)
1187          self.comp2mid.insert(END,self.comp_ca_mid)
1188          self.comp2end.insert(END,self.comp_ca_end)
1189
1190          self.comp_temp_start = 10
1191          self.comp_temp_mid = 10
1192          self.comp_temp_end = 140
1193          self.comp3.insert(END,self.comp_temp_start)
1194          self.comp3mid.insert(END,self.comp_temp_mid)
1195          self.comp3end.insert(END,self.comp_temp_end)
1196
1197          self.comp_na = list(range(self.comp_na_start,
      self.comp_na_end+1, self.comp_na_mid))
1198          self.comp_ca = list(range(self.comp_ca_start,
      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
    (), 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, '
    xlabel': xlabel, 'ylabel': ylabel}
1224                self.plot_rdy[value][1]['title'].update(
    newvalues)
1225            else:
1226                title_input = self.ptitle.get()
1227                if title_input:
1228                    user_title = str(title_input)
1229                    split_title = user_title.split(' ')
1230                    new_title = ' '.join(split_title)
1231                    self.plot_title = new_title
1232                xlabel, ylabel = (None, None)
1233                user_xlabel, user_ylabel = (self.pxtitle.get
    (), self.pytitle.get())
1234                if user_xlabel:
1235                    xlabel = str(user_xlabel)
1236                    self.plot_xlabel = xlabel
1237                if user_ylabel:
1238                    ylabel = str(user_ylabel)
1239                    self.plot_ylabel = ylabel
1240
```

```
1241     def reset_plot_labels(self):
1242         value = self.change_plot_label_current
1243         newvalues = {'shown_title': self.shown_title_old
     , 'xlabel': None, 'ylabel': None}
1244         self.plot_rdy[value][1]['title'].update(
     newvalues)
1245         self.ptitle.delete(0, END)
1246         self.ptitle.insert(END, newvalues['shown_title']
     )
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(
     self.fontsize)
1274 
1275     def set_gelmod(self):
1276         na_start = int(self.comp1.get('1.0',END))
1277         na_mid = int(self.comp1mid.get('1.0',END))
1278         na_end = int(self.comp1end.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,
     na_mid))
1297          self.comp_ca = list(range(ca_start, ca_end+1,
     ca_mid))
1298          self.comp_temp = list(range(temp_start, temp_end
     +1, temp_mid))
1299
1300      def reset_gelmod(self):
1301          self.comp1.delete('1.0',END)
1302          self.comp1mid.delete('1.0',END)
1303          self.comp1end.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.comp1end.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':
```

```
fontsize})
        self.delete_figures(2)
        share_axis, filename, aspect_wanted, aspect_auto = (False, '', 1, True)
        chosen_rows, chosen_cols = (1, 1)

        sharex_local, sharey_local = (False, False)
        fig, axes = (None, None)
        change_plot = self.plottype.get()  # Allow user to change this
        plot_version = None
        if change_plot == 1:
            plot_version = 1
            fig, axes = plt.subplots(
                nrows=chosen_rows, ncols=chosen_cols, sharex=sharex_local, sharey=sharey_local, figsize=(10, 10))
        elif change_plot == 0:
            plot_version = 0
            fig = Figure(figsize=(10, 10))

        alpha_values = [2.000, 0.001, 0.017]
        beta_values = [1.0, 0.9]
        yield_values = [1.0, 0.0, 0.0]
        rg, eag, tref = [math.pow(10,-4), 77, 20]
        crit, surface_area = [0.20, 200]
        rvalue = 0.008314 # kj / K mol
        tref_kelvin = float(tref+273.15)
        conc_si = 10
        inner_factor_tref = float(eag/(rvalue* tref_kelvin))
        effect_of_si = math.pow(conc_si, alpha_values[0])
        effect_of_tref = math.exp(inner_factor_tref)

        xvalues_gelmod = []
        yvalues_gelmod = self.comp_temp
        zvalues_gelmod = []
        ivalue, jvalue, kvalue = [1]*3
        kvalues_check = []
        combined_check = []
        combined_check_values = []
        for i_na in self.comp_na:
            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:
1374                     temp_kelvin = float(k_temp+273.15)
1375                     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         dxi, dyj = ([1.0]*len(xi), [self.comp_temp_mid]*
     len(yj))
1391
```

```
1392            xbound = self.get_block_boundaries(cellvalues=xi
    , cellwidths=dxi)
1393            ybound = self.get_block_boundaries(cellvalues=yj
    , cellwidths=dyj)
1394
1395            x_id_v2, y_id_v2, z_id_v2 = (xi, yj,
    zvalues_gelmod)
1396            # x_id_v2, y_id_v2, z_id_v2 = (xi, yj,
    zvalues_gelmod)
1397            xlength, ylength = (len(x_id_v2), len(y_id_v2))
1398
1399            transpose_choice, rowshape, colshape = (None,
    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
    , colshape))
1405
1406            ax, prev_ax, im = (None, None, None)
1407            if plot_version == 0:
1408                ax = fig.add_subplot(chosen_rows,
    chosen_cols, 1)
1409            elif plot_version == 1:
1410                ax = axes
1411
1412            xlabel, ylabel, title = ('x', 'Temperature (°C)'
    , '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',
    labelbottom=True)
1423            ax.yaxis.set_tick_params(which='both',
    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,
    norm=matplotlib.colors.LogNorm(vmin=z_grid.min(), vmax=
    z_grid.max()), cmap='PuBu_r')
1430            im = ax.pcolormesh(x_grid, y_grid, z_grid.T,
    norm=matplotlib.colors.LogNorm(), cmap='PuBu_r')
1431            # im = ax.pcolor(x_grid, y_grid, z_grid, norm=
    matplotlib.colors.LogNorm(vmin=z_grid.min(), vmax=z_grid
    .max()), cmap='gist_rainbow')
1432
1433            # if transpose_choice == 0:
1434            #     im = ax.pcolormesh(x_grid, y_grid, z_grid
    , cmap=cmap, norm=norm)
1435            # elif transpose_choice == 1:
1436            #     im = ax.pcolormesh(x_grid, y_grid, z_grid.
    T, cmap=cmap, norm=norm)
1437
1438            print('...')
1439            print(': transposed?? ' + str(transpose_choice))
1440            print('xlength: ' + str(len(x_id_v2)) + '
    ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(len(
    z_id_v2)))
1441            # print('(rowshape, colshape): (' + str(rowshape
    ) + '[' + str(rowtype) + '], ' + str(colshape) + '[' +
    str(coltype) + '])')
1442            print('xshape: ' + str(x_grid.shape) + ' yshape
    : ' + str(y_grid.shape) + ' zshape: ' + str(z_grid.shape
    ))
1443            print('...')
1444
1445            if aspect_auto is False:
1446                aspect_ratio_wanted = aspect_wanted
1447                aspect_ratio_correct = abs((x_max - x_min) /
     (y_max - y_min)) / aspect_ratio_wanted
1448                ax.set_aspect(aspect_ratio_correct)
1449
1450            fig.colorbar(im, ax=ax)
1451
1452            plt.tight_layout()
1453
1454            self.canvas = FigureCanvasTkAgg(fig, self.
    f2_plot)
1455            self.canvas.draw()
1456            self.canvas.get_tk_widget().pack(side=tk.TOP,
```

```
1456 fill=tk.BOTH, expand=True)
1457         self.canvas._tkcanvas.pack(side=tk.BOTTOM, fill=
     tk.BOTH, expand=True)
1458         self.toolbar = NavigationToolbar2Tk(self.canvas,
      self.f2_toolkit)  # Toolbar is added to canvas
1459         self.toolbar.update()
1460 
1461     def set_xyz(self):
1462         child2 = self.prep_sim_parameters
1463         child2.selection_clear(0, END)
1464         if self.freezex1 and self.freezey1 and self.
     freezez1 and self.freezetime1:
1465             data = self.simcase_ijkt_count
1466             imin, imax, jmin, jmax, kmin, kmax, tmin,
     tmax = (data['imin'], data['imax'], data['jmin'], data['
     jmax'], data['kmin'], data['kmax'], data['tmin'], data['
     tmax'])
1467             current_xyz = self.xyz.get()
1468             iimin, iimax = (int(self.slidex_left.get()),
      int(self.slidex_right.get()))
1469             jjmin, jjmax = (int(self.slidey_left.get()),
      int(self.slidey_right.get()))
1470             kkmin, kkmax = (int(self.slidez_left.get()),
      int(self.slidez_right.get()))
1471             ttmin, ttmax = (int(self.slidetime_left.get(
     )), int(self.slidetime_right.get()))
1472             if current_xyz == 9:
1473                 self.xyz.set(0)
1474                 self.slidex_left.set(imin)
1475                 self.slidex_right.set(imax)
1476                 self.slidey_left.set(jmin)
1477                 self.slidey_right.set(jmax)
1478                 self.slidez_left.set(kmin)
1479                 self.slidez_right.set(kmax)
1480                 self.slidetime_left.set(tmin)
1481                 self.slidetime_right.set(tmax)
1482                 self.valuex1.set(0)
1483                 self.valuey1.set(0)
1484                 self.valuez1.set(0)
1485                 self.valuetime1.set(0)
1486             elif current_xyz in [7,8]:
1487                 simcase = current_selection
1488                 core_path = global_sim_data[simcase][0]
1489                 childx = self.x_listbox
1490                 childy = 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.
     slidex_right], 'Y': [self.slidey_left, self.slidey_right
     ], 'Z': [self.slidez_left, self.slidez_right], 'time': [
     self.slidetime_left, self.slidetime_right]}
1495                    freezes = {'X': [self.freezex1, self.
     valuex1, self.slidex_label1, self.slidex_label2, sliders
     ['X'][1]], 'Y': [self.freezey1, self.valuey1, self.
     slidey_label1, self.slidey_label2, sliders['Y'][1]],
1496                                'Z': [self.freezez1, self.
     valuez1, self.slidez_label1, self.slidez_label2, sliders
     ['Z'][1]], 'time': [self.freezetime1, self.valuetime1,
     self.slidetime_label1, self.slidetime_label2, sliders['
     time'][1]]}
1497                    ijkvalues = {'X': [iimin, iimax, imin,
     imax], 'Y': [jjmin, jjmax, jmin, jmax], 'Z': [kkmin,
     kkmax, kmin, kmax], 'time': [ttmin, ttmax, tmin, tmax]}
1498
1499                    if ttmin == ttmax:
1500                        self.slidetime_left.set(tmin)
1501                        self.slidetime_right.set(tmax)
1502                        self.valuetime1.set(0)
1503
1504                    openlist, currentlist = (None, None)
1505                    self.plot_id_old = self.plot_id
1506                    self.plot_id = {'simcase': simcase, '
     simcase_path': core_path, 'simcase_child': ['DATA', ['
     DATA']], 'entries': ['Days'], 'cells': None, 'time':
     None, 'X': None, 'Y': None}
1507                    choices = ['X', 'Y']
1508                    choice = None
1509                    if current_xyz == 7 and allsx: # Replace
      X with Days | Open fully Y
1510                        openlist = self.plot_y[allsy[0]]['
     entries']
1511                        currentlist = self.plot_x[allsx[0]][
     'entries']
1512                        choice = choices[0]
1513                    elif current_xyz == 8 and allsy: #
     Replace Y with Days | Open fully X
1514                        openlist = self.plot_x[allsx[0]]['
     entries']
1515                        currentlist = self.plot_y[allsy[0]][
```

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

```
1546                    child = self.prep_sim_parameters
1547                    alls = self.prep_sim_parameters.get(0,
     END)
1548                    indexdata = alls.index('DATA')
1549                    access = (indexdata,)
1550                    self.prep_sim_parameters.selection_set(
     access)
1551                    self.prep_sim_parameters.event_generate(
     '<<ListboxSelect>>')
1552
1553                    simcase = current_selection
1554                    core_path = global_sim_data[simcase][0]
1555
1556                    self.valuex1.set(0)
1557                    self.valuey1.set(0)
1558                    self.valuez1.set(0)
1559                    self.valuetime1.set(0)
1560                    if current_xyz == 1 or current_xyz == 2:
      # XY and YX
1561                        self.freezetime1.select()
1562                        self.freeze_val_time(self.valuetime1
     , self.slidetime_label1, self.slidetime_label2, self.
     slidetime_right)
1563                        self.freezez1.select()
1564                        self.freeze_val(self.valuez1, self.
     slidez_label1, self.slidez_label2, self.slidez_right)
1565
1566                        self.slidex_left.set(imin)
1567                        self.slidex_right.set(imax)
1568                        self.slidey_left.set(jmin)
1569                        self.slidey_right.set(jmax)
1570                        if kkmin != kkmax:
1571                            rnd_k = int(random.uniform(kmin,
     kmax+1))
1572                            self.slidez_left.set(rnd_k)
1573                        if ttmin != ttmax:
1574                            rnd_t = int(random.uniform(tmin,
     tmax+1))
1575                            self.slidetime_left.set(rnd_t)
1576
1577                        if current_xyz == 1:
1578                            self.plot_id_old = self.plot_id
1579                            self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, '
```

```
time': None, 'X': None, 'Y': None}
                        self.add_to_xy(typedata='X')
                        self.plot_id = {'simcase':
    simcase, 'simcase_path': core_path, 'simcase_child': ['
    INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None, '
    time': None, 'X': None, 'Y': None}
                        self.add_to_xy(typedata='Y')
                        self.plot_id = self.plot_id_old
                    elif current_xyz == 2:
                        self.plot_id_old = self.plot_id
                        self.plot_id = {'simcase':
    simcase, 'simcase_path': core_path, 'simcase_child': ['
    INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None, '
    time': None, 'X': None, 'Y': None}
                        self.add_to_xy(typedata='X')
                        self.plot_id = {'simcase':
    simcase, 'simcase_path': core_path, 'simcase_child': ['
    INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, '
    time': None, 'X': None, 'Y': None}
                        self.add_to_xy(typedata='Y')
                        self.plot_id = self.plot_id_old
                elif current_xyz == 3 or current_xyz ==
    4: # XZ and ZX
                    self.freezetime1.select()
                    self.freeze_val_time(self.valuetime1
    , self.slidetime_label1, self.slidetime_label2, self.
    slidetime_right)
                    self.freezey1.select()
                    self.freeze_val(self.valuey1, self.
    slidey_label1, self.slidey_label2, self.slidey_right)

                    self.slidex_left.set(imin)
                    self.slidex_right.set(imax)
                    self.slidez_left.set(kmin)
                    self.slidez_right.set(kmax)
                    if jjmin != jjmax:
                        rnd_j = int(random.uniform(jmin,
     jmax + 1))
                        self.slidey_left.set(rnd_j)
                    if ttmin != ttmax:
                        rnd_t = int(random.uniform(tmin,
     tmax + 1))
                        self.slidetime_left.set(rnd_t)

                    if current_xyz == 3:
```

```
1609                            self.plot_id_old = self.plot_id
1610                            self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1611                            self.add_to_xy(typedata='X')
1612                            self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1613                            self.add_to_xy(typedata='Y')
1614                            self.plot_id = self.plot_id_old
1615                        elif current_xyz == 4:
1616                            self.plot_id_old = self.plot_id
1617                            self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1618                            self.add_to_xy(typedata='X')
1619                            self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['X'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1620                            self.add_to_xy(typedata='Y')
1621                            self.plot_id = self.plot_id_old
1622                    elif current_xyz == 5 or current_xyz ==
     6: # ZY and YZ
1623                        self.freezetime1.select()
1624                        self.freeze_val_time(self.valuetime1
     , self.slidetime_label1, self.slidetime_label2, self.
     slidetime_right)
1625                        self.freezex1.select()
1626                        self.freeze_val(self.valuex1, self.
     slidex_label1, self.slidex_label2, self.slidex_right)
1627
1628                        self.slidez_left.set(imin)
1629                        self.slidez_right.set(imax)
1630                        self.slidey_left.set(jmin)
1631                        self.slidey_right.set(jmax)
1632                        if iimin != iimax:
1633                            rnd_x = int(random.uniform(imin,
      imax + 1))
1634                            self.slidex_left.set(rnd_x)
1635                        if ttmin != ttmax:
1636                            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':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1642                          self.add_to_xy(typedata='X')
1643                          self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1644                          self.add_to_xy(typedata='Y')
1645                          self.plot_id = self.plot_id_old
1646                      elif current_xyz == 6:
1647                          self.plot_id_old = self.plot_id
1648                          self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Y'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1649                          self.add_to_xy(typedata='X')
1650                          self.plot_id = {'simcase':
     simcase, 'simcase_path': core_path, 'simcase_child': ['
     INPUT', ['INPUT']], 'entries': ['Z'], 'cells': None, '
     time': None, 'X': None, 'Y': None}
1651                          self.add_to_xy(typedata='Y')
1652                          self.plot_id = self.plot_id_old
1653          else:
1654              self.xyz.set(0)
1655
1656      def fetch_data(self, path, value_returned,
     property_chosen, cells, time_element):
1657          sorted_dataframe, sorted_dataframe_small,
     sorted_dataframe_selection = (None, None, None)
1658          path_ar_input = os.path.join(path, 'INPUT' + '.
     parquet')
1659          x_input = pd.read_parquet(path_ar_input)
1660          ilist, jlist, klist, tlist = (time_element[2],
     time_element[3], time_element[4], time_element[5])
1661          newcells = x_input.loc[(x_input['i'].isin(ilist)
     ) & (x_input['j'].isin(jlist)) & (x_input['k'].isin(
     klist)), 'Cell'].tolist()
1662          allcells = x_input['Cell'].tolist()
```

```
1663            # cells, times = (np.unique(cells), np.unique(
    times))
1664            filename = value_returned[0]
1665            if filename == 'COMP':
1666                path_ar_comp = os.path.join(path, 'COMP' +
    '.parquet')
1667                x_comp = pd.read_parquet(path_ar_comp)
1668                pressure_chosen = value_returned[1]  # 240.0
                            Input
1669                component_chosen = value_returned[2]  # Ca(
    mg)
1670                sorted_dataframe = x_comp.loc[(x_comp['
    Component'] == component_chosen) & (x_comp['Pressure']
    == pressure_chosen), :].loc[times]
1671                sorted_dataframe_small = sorted_dataframe[
    property_chosen]
1672                sorted_dataframe_selection =
    sorted_dataframe_small.tolist()
1673            elif filename == 'REGION':
1674                path_ar_region = os.path.join(path, 'REGION'
     + '.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['
    Component'] == component_chosen) & (x_region['Region']
    == region_chosen), :].loc[times]
1679                sorted_dataframe_small = sorted_dataframe[
    property_chosen]
1680                sorted_dataframe_selection =
    sorted_dataframe_small.tolist()
1681            elif filename == 'WELLS':
1682                path_ar_wells = os.path.join(path, 'WELLS' +
     '.parquet')
1683                x_wells = pd.read_parquet(path_ar_wells)
1684                well_chosen = value_returned[1]
1685                sorted_dataframe = x_wells.loc[x_wells['
    nWell'] == well_chosen, :].loc[times]
1686                sorted_dataframe_small = sorted_dataframe[
    property_chosen]
1687                sorted_dataframe_selection =
    sorted_dataframe_small.tolist()
1688            elif filename == 'TIME':
1689                path_ar_time = os.path.join(path, 'TIME' +
    '.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)[
    fetch_something].tolist()
1694                return t_to_plot
1695            elif filename == 'DATA':
1696                path_ar_data = os.path.join(path, 'DATA' +
    '.parquet')
1697                x_data = pd.read_parquet(path_ar_data)
1698                prop_final = []
1699                df_prop = x_data[property_chosen]
1700                maxcells = max(allcells)
1701                cells_unique = np.array(newcells) - 1
1702                timesteps = int(len(df_prop)/maxcells)
1703                for timestep in list(range(1,timesteps+1)):
1704                    top_pos = (timestep - 1)*maxcells
1705                    bot_pos = timestep*maxcells
1706                    current = df_prop[top_pos:bot_pos]
1707                    if timestep in tlist:
1708                        fetch_cells = cells_unique + (
    timestep-1)*maxcells
1709                        prop_to_plot = df_prop.iloc(axis=0)[
    fetch_cells].tolist()
1710                        prop_final = prop_final +
    prop_to_plot
1711                return prop_final
1712            elif filename == 'INPUT':
1713                path_ar_input = os.path.join(path, 'INPUT' +
     '.parquet')
1714                x_input = pd.read_parquet(path_ar_input)
1715                ilist, jlist, klist, tlist = (time_element[2
    ], time_element[3], time_element[4], time_element[5])
1716                newcells = x_input.loc[(x_input['i'].isin(
    ilist)) & (x_input['j'].isin(jlist)) & (x_input['k'].
    isin(klist)), 'Cell'].tolist()
1717                if property_chosen == 'Cell':
1718                    return newcells
1719                else:
1720                    df_prop = x_input[property_chosen]
1721                    fetch_cells = np.array(newcells) - 1
1722                    prop_to_plot = df_prop.iloc(axis=0)[
    fetch_cells].tolist()
1723                    unique_values = np.unique(prop_to_plot)
1724                    return unique_values
```

```
1725            return sorted_dataframe, sorted_dataframe_small,
      sorted_dataframe_selection
1726
1727        def get_block_boundaries(self, cellvalues,
     cellwidths):
1728            cellvalues, cellwidths = (np.array(cellvalues),
     np.array(cellwidths))
1729            left_boundaries = (cellvalues - 0.5 * cellwidths
     ).tolist()
1730            right_boundaries = [cellvalues[-1] + 0.5 *
     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),
     np.array(cellwidths))
1746            left_boundaries = (cellvalues - 0.5 * cellwidths
     ).tolist()
1747            right_boundaries = [cellvalues[-1] + 0.5 *
     cellwidths[-1]]
1748            boundaries = left_boundaries + right_boundaries
1749            return boundaries
1750
1751        def get_time(self, core_path, property_chosen, tlist
     ):
1752            path_ar_time = os.path.join(core_path, 'TIME' +
     '.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,
     ilist, jlist, klist):
1762         path_ar_input = os.path.join(core_path, 'INPUT'
     + '.parquet')
1763         x_input = pd.read_parquet(path_ar_input)
1764         allcells = x_input['Cell'].tolist()
1765         ijkcells = x_input.loc[(x_input['i'].isin(ilist)
     ) &
1766                                (x_input['j'].isin(jlist)
     ) & (x_input['k'].isin(klist)), 'Cell'].tolist()
1767
1768         fetch_cells = np.array(ijkcells) - 1
1769         df = x_input[property_chosen]
1770         values = df.iloc(axis=0)[fetch_cells]
1771         return values
1772
1773     def get_data(self, core_path, property_chosen,
     allcells, newcells, ilist, jlist, klist, tlist):
1774         values = []
1775         if property_chosen in ['Days', 'Timestep']:
1776             df = self.get_time(core_path,
     property_chosen, tlist)
1777             values = df.values.tolist()
1778             return values
1779         elif property_chosen == 'Cell':
1780             df = self.get_input(core_path=core_path,
     property_chosen=property_chosen, ilist=ilist, jlist=
     jlist, klist=klist)
1781             values = df.values.tolist()
1782             return values
1783         elif property_chosen in ['X', 'Y', 'Z']:
1784             df = self.get_input(core_path=core_path,
     property_chosen=property_chosen, ilist=ilist, jlist=
     jlist, klist=klist)
1785             values = df.values.tolist()
1786             return values
1787         else:
1788             path_ar_data = os.path.join(core_path, 'DATA
     ' + '.parquet')
1789             x_data = pd.read_parquet(path_ar_data)
1790             z_final = []
1791             df_z = x_data[property_chosen]
1792             z_id = df_z.tolist()
1793             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 + (
    timestep - 1) * maxcells
1802                        z_to_plot = df_z.iloc(axis=0)[
    fetch_cells].tolist()
1803                        z_final = z_final + z_to_plot
1804                values = z_final
1805            return values
1806
1807        def plot_graphv2(self):
1808            self.delete_figures(2)
1809            child, prev_simcase, local_dict, store_dict,
    chosen_parameters_rawtest = (self.pageone_listbox_plot,
    {}, {}, {}, [])
1810            alls = list(child.get(0, END))
1811            if self.hold.get() == 0 and self.xyz.get() != 0:
1812                simcase = current_selection
1813                corepath = global_sim_data[simcase][0]
1814                current_xyz = self.xyz.get()  # 1,2,3,etc
1815                xnow, ynow, simcase_child = (None, None,
    None)
1816                self.plot_id_old = self.plot_id
1817                xcoord = [['X'], ['Y'], ['X'], ['Z'], ['Z'],
     ['Y']]
1818                ycoord = [['Y'], ['X'], ['Z'], ['X'], ['Y'],
     ['Z']]
1819                if current_xyz in [7, 8]:
1820                    childx = self.x_listbox
1821                    childy = self.y_listbox
1822                    allsx = childx.get(0, END)
1823                    allsy = childy.get(0, END)
1824                    xnow = self.plot_x[allsx[0]]['entries']
1825                    ynow = self.plot_y[allsy[0]]['entries']
1826                    simcase_child_01, simcase_child_02 = (
    None, None)
1827                    if current_xyz == 7:  # Means X is days
1828                        simcase_child_01 = ['DATA', ['DATA']
    ]
```

```
1829                        simcase_child_02 = ['INPUT', ['INPUT
    ']]
1830                    elif current_xyz == 8:  # Means Y is
    days
1831                        simcase_child_01 = ['INPUT', ['INPUT
    ']]
1832                        simcase_child_02 = ['DATA', ['DATA']
    ]
1833                    self.plot_id = {'simcase': simcase, '
    simcase_path': corepath, 'simcase_child':
    simcase_child_01, 'entries': xnow, 'cells': None, 'time'
    : None, 'X': None, 'Y': None}
1834                    self.add_to_xy(typedata='X')
1835                    self.plot_id = {'simcase': simcase, '
    simcase_path': corepath, 'simcase_child':
    simcase_child_02, 'entries': ynow, 'cells': None, 'time'
    : 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],
    ycoord[current_xyz - 1])
1840                    self.plot_id = {'simcase': simcase, '
    simcase_path': corepath, 'simcase_child': simcase_child,
     'entries': xnow, 'cells': None, 'time': None, 'X': None
    , 'Y': None}
1841                    self.add_to_xy(typedata='X')
1842                    self.plot_id = {'simcase': simcase, '
    simcase_path': corepath, 'simcase_child': simcase_child,
     'entries': ynow, 'cells': None, 'time': None, 'X': None
    , '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[
    prev_plotted_item][1]
1851                            current_path = oldcontent['
    simcase_path']
1852                            cells, times = self.
    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]
     [1].update(newvalues)
1855                         child.insert(END,
     prev_plotted_item)
1856                     alls = list(child.get(0, END))
1857                     self.figs = len(alls)
1858                     self.grid_size_figures()
1859                     self.label_figs['text'] = 'Figures
     : ' + str(self.figs)
1860             elif alls:
1861                 for prev_plotted_item in alls:
1862                     # print('prev_plotted_item: ' + str(
     prev_plotted_item))
1863                     oldcontent = self.plot_rdy[
     prev_plotted_item][1]
1864                     core_path2 = oldcontent['
     simcase_path']
1865                     # print('oldcontent: ' + str(
     oldcontent))
1866                     marker = self.plot_rdy[
     prev_plotted_item][2]
1867                     # print('marker: ' + str(marker))
1868                     if marker == 1:
1869                         current_path = oldcontent['
     simcase_path']
1870                         cells2, time_element2 = self.
     get_cell_time(corepath=current_path)
1871                         ilist2, jlist2, klist2, tlist2 =
      (time_element2[2], time_element2[3], time_element2[4],
     time_element2[5])
1872                         timedays2 = self.get_time(
     core_path2, 'Days', tlist2).values.tolist()
1873                         data2 = self.plot_rdy[
     prev_plotted_item][1]['title']
1874                         old_basetitle = data2['
     shown_title'].split('=')[0]
1875                         new_basetitle = old_basetitle +
     '=' + str(timedays2[0]) + ' days'
1876                         newvalues = {'shown_title':
     new_basetitle, 'timedays': timedays2}
1877                         self.plot_rdy[prev_plotted_item]
     [1]['title'].update(newvalues)
1878                         newvalues = {'cells': cells2, '
     time': time_element2, 'X': self.plot_x, 'Y': self.plot_y
```

```
1878 }
1879                         self.plot_rdy[prev_plotted_item]
     [1].update(newvalues)
1880                     elif marker == 0:
1881                         current_path = oldcontent['
     simcase_path']
1882                         cells2, time_element2 = self.
     get_cell_time(corepath=current_path)
1883                         newvalues = {'cells': cells2, 'X
     ': self.plot_x, 'Y': self.plot_y}
1884                         self.plot_rdy[prev_plotted_item]
     [1].update(newvalues)
1885
1886         share_axis, filename, aspect_wanted, aspect_auto
      = (False, '', 1, True)
1887         chosen_rows, chosen_cols = ([], [])
1888         global chosen_rows_alt, chosen_cols_alt
1889         if chosen_rows_alt is not None and
     chosen_cols_alt is not None:
1890             chosen_rows, chosen_cols = (int(
     chosen_rows_alt), int(chosen_cols_alt))
1891             self.grid_button.config(text='Rows: ' +
     chosen_rows_alt + ' Cols: ' + chosen_cols_alt)
1892             chosen_rows_alt, chosen_cols_alt = (None,
     None)
1893         else:
1894             dimensions = self.fig_grid_size[self.
     grid_dropdown.get()]
1895             chosen_rows, chosen_cols = dimensions
1896
1897         tight_plot = True
1898         sharex_local, sharey_local = (False, False)
1899         fig, axes = (None, None)
1900         change_plot = self.plottype.get()  # Allow user
     to change this
1901         plot_version = None
1902         if change_plot == 1:
1903             plot_version = 1
1904             if self.sharex.get() == 1:
1905                 sharex_local = True
1906             if self.sharey.get() == 1:
1907                 sharey_local = True
1908             fig, axes = plt.subplots(
1909                 nrows=chosen_rows, ncols=chosen_cols,
     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,
     None)
1921             raw = self.plot_rdy[item]
1922             data, identifier = (raw[1], raw[0])
1923             simcase_path = data['simcase_path']
1924             titledata = data['title']
1925             title = titledata['shown_title']
1926             new_xlabel = titledata['xlabel']
1927             new_ylabel = titledata['ylabel']
1928             fontsize = int(data['fontsize'])
1929             if self.hold3.get() == 0:
1930                 fontsize = self.fontsize
1931             matplotlib.rcParams.update({'font.size':
     fontsize})
1932             cells, time_element = (data['cells'], data['
     time'])
1933             if self.hold == 0:
1934                 cells, time_element = self.get_cell_time
     (corepath=simcase_path)
1935             times, days = (time_element[0].tolist(),
     time_element[1].tolist())
1936             xvalues, yvalues = (None, None)
1937             x_property_chosen, y_property_chosen = (None
     , None)
1938             x_path, y_path = (None, None)
1939             test = None
1940             xvalues_unique, yvalues_unique,
     zvalues_unique = (None, None, None)
1941             if data['X']:
1942                 x_key = list(data['X'].keys())[0]
1943                 x_key_prop = x_key.split(' ')[2]
1944                 x_coord_data = data['X'][x_key]
1945                 x_simcase_child = x_coord_data['
     simcase_child'][1]
```

```
1946                 x_property_chosen = x_coord_data['
    entries'][0]
1947                 x_path = x_coord_data['simcase_path']
1948                 x_simcase = x_coord_data['simcase']
1949                 if x_simcase_child == 'DATA' and
    x_key_prop == 'Cell':
1950                     x_simcase_child = ['INPUT']
1951                 x_filename = x_simcase_child[0]
1952                 xvalues_unique = self.fetch_data(path=
    x_path, value_returned=x_simcase_child, property_chosen=
    x_property_chosen, cells=cells, time_element=
    time_element)
1953             if data['Y']:
1954                 y_key = list(data['Y'].keys())[0]
1955                 y_key_prop = y_key.split(' ')[2]
1956                 y_coord_data = data['Y'][y_key]
1957                 y_simcase_child = y_coord_data['
    simcase_child'][1]
1958                 if y_simcase_child == 'DATA' and
    y_key_prop == 'Cell':
1959                     y_simcase_child = ['TIME']
1960                 y_property_chosen = y_coord_data['
    entries'][0]
1961                 y_path = y_coord_data['simcase_path']
1962                 y_simcase = y_coord_data['simcase']
1963                 y_filename = y_simcase_child[0]
1964                 yvalues_unique = self.fetch_data(path=
    y_path, value_returned=y_simcase_child, property_chosen=
    y_property_chosen, cells=cells, time_element=
    time_element)
1965             keys = list(data.keys())
1966             simcase_child = data['simcase_child']
1967             filename = simcase_child[1][0]
1968             z_property_chosen = data['entries']
1969             simcase = data['simcase']
1970             xlabel = 'Cell Numbering (unique)'
1971             if new_xlabel:
1972                 xlabel = new_xlabel
1973             ylabel = 'Simulation runtime (days)'
1974             if new_ylabel:
1975                 ylabel = new_ylabel
1976
1977             if filename == 'DATA':
1978                 z_property_chosen = prep_pageone[simcase
    ][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 = (
    None, None, None)
2016                rowshape, colshape = (None, None)
2017                rowtype, coltype = (None, None)
2018                x_grid, y_grid, z_grid = (None, None, None)
2019
2020                plot_type = None
2021                ilen, jlen, klen, tlen = (len(ilist), len(
    jlist), len(klist), len(tlist))
2022                if ilen == 1 and jlen == 1 and klen == 1 and
     tlen >= 1:
2023                    plot_type = 'time'
2024                # elif tlen==1 and ((ilen!=1 and jlen==1 and
     klen==1) or (ilen==1 and jlen!=1 and klen==1) or (ilen
    ==1 and jlen==1 and klen!=1)):
2025                #     plot_type = 'position'
2026                else:
2027                    plot_type = '2d'
2028                ax = None
2029                if filename == 'DATA' and plot_type == 'time
    ':
2030                    core_path = simcase_path
2031                    y_id_v2 = self.get_data(core_path=
    core_path, property_chosen='Days', allcells=allcells,
    newcells=newcells, ilist=ilist, jlist=jlist, klist=klist
    , tlist=tlist)
2032                    z_id_v2 = self.get_data(core_path=
    core_path, property_chosen=z_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2033
2034                    row_id, col_id = self.find_row_col(
    identifier, chosen_rows, chosen_cols)
2035                    ax = None
2036                    if plot_version == 0:
2037                        if not prev_ax:
2038                            ax = fig.add_subplot(chosen_rows
    , chosen_cols, identifier)
2039                        else:
2040                            if sharex_local == 'all' and
    sharey_local != 'all':
2041                                ax = fig.add_subplot(
```

```
2041 chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2042                         elif sharex_local != 'all' and
     sharey_local == 'all':
2043                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2044                         elif sharex_local == 'all' and
     sharey_local == 'all':
2045                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2046                         else:
2047                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier)
2048                 elif plot_version == 1:
2049                     if (row_id, col_id) == (None, None):
2050                         break
2051                     if chosen_rows == 1 and chosen_cols
     == 1:  # Only plot one figure
2052                         ax = axes
2053                     elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2054                         ax = axes[col_id]
2055                         tight_plot = True
2056                     elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2057                         ax = axes[row_id]
2058                     elif chosen_rows != 1 and
     chosen_cols != 1:  # Use both row_id and col_id
2059                         ax = axes[row_id, col_id]
2060                 title = z_property_chosen
2061                 ax.set_title(title)
2062                 xlabel, ylabel = (y_property_chosen,
     z_property_chosen)
2063                 if new_xlabel:
2064                     xlabel = new_xlabel
2065                 if new_ylabel:
2066                     ylabel = new_ylabel
2067                 ax.set_xlabel(xlabel)
2068                 ax.set_ylabel(ylabel)
2069                 ax.plot(y_id_v2, z_id_v2)
2070             elif filename == 'DATA' and plot_type == '
     position':
2071                 core_path = simcase_path
2072 
2073                 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=
    core_path, property_chosen=x_property_chosen,
2080                                          ilist=ilist, jlist
    =jlist, klist=klist)
2081                    x_id_v2 = dims.tolist()
2082                    z_id_v2 = self.get_data(core_path=
    core_path, property_chosen=z_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2083
2084                    row_id, col_id = self.find_row_col(
    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
    , chosen_cols, identifier)
2089                        else:
2090                            if sharex_local == 'all' and
    sharey_local != 'all':
2091                                ax = fig.add_subplot(
    chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2092                            elif sharex_local != 'all' and
    sharey_local == 'all':
2093                                ax = fig.add_subplot(
    chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2094                            elif sharex_local == 'all' and
    sharey_local == 'all':
2095                                ax = fig.add_subplot(
    chosen_rows, chosen_cols, identifier, sharex=prev_ax,
    sharey=prev_ax)
2096                            else:
2097                                ax = fig.add_subplot(
    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
    == 1:  # Only plot one figure
2102                            ax = axes
```

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

```
2130                 ynew = self.get_block_boundaries(
    cellvalues=yj, cellwidths=dyj)
2131                 z_id_v2 = self.get_data(core_path=
    core_path, property_chosen=z_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2132
2133                 x_centers = xi
2134                 x_bound = xnew
2135                 y_centers = yj
2136                 y_bound = ynew
2137
2138                 x_id_v2, y_id_v2 = (x_id_v2_alt,
    y_id_v2_alt)
2139                 xlength, ylength = (len(x_id_v2), len(
    y_id_v2))
2140                 transpose_choice = None
2141                 if xlength >= ylength:
2142                     rowshape, colshape = (ylength,
    xlength)
2143                     transpose_choice = 0
2144                 elif xlength < ylength:
2145                     rowshape, colshape = (xlength,
    ylength)
2146                     transpose_choice = 1
2147
2148                 x_grid, y_grid = np.meshgrid(x_id_v2,
    y_id_v2)
2149                 z_grid = np.reshape(np.array(z_id_v2), (
    rowshape, colshape))
2150
2151                 fig_plotted, ax = (fig_plotted + 1, None
    )
2152                 row_id, col_id = self.find_row_col(
    identifier, chosen_rows, chosen_cols)
2153                 ax = None
2154                 if plot_version == 0:
2155                     if not prev_ax:
2156                         ax = fig.add_subplot(chosen_rows
    , chosen_cols, identifier)
2157                     else:
2158                         if sharex_local == 'all' and
    sharey_local != 'all':
2159                             ax = fig.add_subplot(
    chosen_rows, chosen_cols, identifier, sharex=prev_ax)
```

```
2160                            elif sharex_local != 'all' and
     sharey_local == 'all':
2161                                ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2162                            elif sharex_local == 'all' and
     sharey_local == 'all':
2163                                ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2164                            else:
2165                                ax = fig.add_subplot(
     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
     == 1:  # Only plot one figure
2170                            ax = axes
2171                        elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2172                            ax = axes[col_id]
2173                            tight_plot = True
2174                        elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2175                            ax = axes[row_id]
2176                        elif chosen_rows != 1 and
     chosen_cols != 1:  # Use both row_id and col_id
2177                            ax = axes[row_id, col_id]
2178
2179                    ax.set_title(title)
2180                    xlabel, ylabel = (x_property_chosen,
     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',
     labelbottom=True)
2188                    ax.yaxis.set_tick_params(which='both',
     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,
     z_grid, cmap=cmap, norm=norm)
2194                 elif transpose_choice == 1:
2195                     im = ax.pcolormesh(x_grid, y_grid,
     z_grid.T, cmap=cmap, norm=norm)
2196 
2197                 print('...')
2198                 print(str(x_property_chosen) + str(
     y_property_chosen) + ': transposed?? ' + str(
     transpose_choice))
2199                 print('xlength: ' + str(len(x_id_v2)) +
     ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
     len(z_id_v2)))
2200                 print('(rowshape, colshape): (' + str(
     rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
      '[' + str(coltype) + '])')
2201                 print('xshape: ' + str(x_grid.shape) +
     ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
     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 -
     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
      True and plot_version == 1:
2217                     plt.tight_layout()
2218             elif filename == 'DATA' and plot_type == '2d
     ' and ((x_property_chosen in ['X', 'Y', 'Z'] and
     y_property_chosen in ['Days']) or (x_property_chosen in
     ['Days'] and y_property_chosen in ['X', 'Y', 'Z'])):
2219                 core_path = simcase_path
2220                 datatypes = {'X': 'DX', 'Y': 'DY', 'Z':
     'DZ'}
```

```
2221                dimi, dimj, dims, xi, yj, dxi, dyj = (
    None, None, None, None, None, None, None)
2222
2223                x_id_v2_alt = self.get_data(core_path=
    core_path, property_chosen=x_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2224                y_id_v2_alt = self.get_data(core_path=
    core_path, property_chosen=y_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2225                z_id_v2_alt = self.get_data(core_path=
    core_path, property_chosen=z_property_chosen, allcells=
    allcells, newcells=newcells, ilist=ilist, jlist=jlist,
    klist=klist, tlist=tlist)
2226
2227                xi, yj = (x_id_v2_alt, y_id_v2_alt)
2228
2229                if x_property_chosen in ['Days']:
2230                    dimj = datatypes[y_property_chosen]
2231                    dims = self.get_input(core_path=
    core_path, property_chosen=[dimj], ilist=ilist, jlist=
    jlist, klist=klist)
2232                    dxi, dyj = ([1.0] * len(xi), list(
    dims[dimj]))
2233                elif y_property_chosen in ['Days']:
2234                    dimi = datatypes[x_property_chosen]
2235                    dims = self.get_input(core_path=
    core_path, property_chosen=[dimi], ilist=ilist, jlist=
    jlist, klist=klist)
2236                    dxi, dyj = (list(dims[dimi]), [1.0]
    * len(yj))
2237
2238                xnew = self.get_block_boundaries(
    cellvalues=xi, cellwidths=dxi)
2239                ynew = self.get_block_boundaries(
    cellvalues=yj, cellwidths=dyj)
2240
2241                x_centers, x_bound, y_centers, y_bound =
     (None, None, None, None)
2242                if x_property_chosen in ['Days']:
2243                    x_centers = xi
2244                    x_bound = xnew
2245                    y_centers = (np.unique(yj)).tolist()
2246                    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(
    x_centers)) + ' x_centers: ' + str(x_centers))
2254                 print('y_centers len: ' + str(len(
    y_centers)) + ' y_centers: ' + str(y_centers))
2255                 print('x_bound len: ' + str(len(x_bound)
    ) + ' x_bound: ' + str(x_bound))
2256                 print('y_bound len: ' + str(len(y_bound)
    ) + ' y_bound: ' + str(y_bound))
2257 
2258                 x_id_v2, y_id_v2, z_id_v2 = (x_bound,
    y_bound, z_id_v2_alt)
2259                 xlength, ylength = (len(x_id_v2), len(
    y_id_v2))
2260                 transpose_choice = None
2261                 if xlength > ylength:
2262                     rowshape, colshape = (ylength - 1,
    xlength - 1)
2263                     transpose_choice = 0
2264                 elif xlength < ylength:
2265                     rowshape, colshape = (xlength - 1,
    ylength - 1)
2266                     transpose_choice = 1
2267                 elif xlength == ylength:
2268                     if (x_property_chosen == 'X' and
    y_property_chosen == 'Days') or (x_property_chosen == 'Y
    ' and y_property_chosen == 'Days') or (x_property_chosen
     == 'Z' and y_property_chosen == 'Days'):
2269                         rowshape, colshape = (ylength -
    1, xlength - 1)
2270                         transpose_choice = 0
2271                     elif (x_property_chosen == 'Days'
    and y_property_chosen == 'X') or (x_property_chosen == '
    Days' and y_property_chosen == 'Y') or (
    x_property_chosen == 'Days' and y_property_chosen == 'Z'
    ):
2272                         rowshape, colshape = (xlength -
    1, ylength - 1)
2273                         transpose_choice = 1
2274 
```

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

```
2303                            ax = axes[row_id]
2304                        elif chosen_rows != 1 and
     chosen_cols != 1:   # Use both row_id and col_id
2305                            ax = axes[row_id, col_id]
2306
2307                    ax.set_title(title)
2308                    xlabel, ylabel = (x_property_chosen,
     y_property_chosen)
2309                    if new_xlabel:
2310                        xlabel = new_xlabel
2311                    if new_ylabel:
2312                        ylabel = new_ylabel
2313                    ax.set_xlabel(xlabel)
2314                    ax.set_ylabel(ylabel)
2315                    ax.xaxis.set_tick_params(which='both',
     labelbottom=True)
2316                    ax.yaxis.set_tick_params(which='both',
     labelbottom=True)
2317                    norm = clr.Normalize()
2318                    cmap = cm.get_cmap('gist_rainbow')
2319
2320                    if transpose_choice == 0:
2321                        im = ax.pcolormesh(x_grid, y_grid,
     z_grid, cmap=cmap, norm=norm)
2322                    elif transpose_choice == 1:
2323                        im = ax.pcolormesh(x_grid, y_grid,
     z_grid.T, cmap=cmap, norm=norm)
2324
2325                    print('...')
2326                    print(str(x_property_chosen) + str(
     y_property_chosen) + ': transposed?? ' + str(
     transpose_choice))
2327                    print('xlength: ' + str(len(x_id_v2)) +
     ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
     len(z_id_v2)))
2328                    print('(rowshape, colshape): (' + str(
     rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
      '[' + str(coltype) + '])')
2329                    print('xshape: ' + str(x_grid.shape) +
     ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
     z_grid.shape))
2330                    print('...')
2331
2332                    if grid_xticks:
2333                        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 -
     x_min) / (y_max - y_min)) / aspect_ratio_wanted
2340                     ax.set_aspect(aspect_ratio_correct)
2341 
2342                 fig.colorbar(im, ax=ax)
2343 
2344                 plt.tight_layout()
2345 
2346                 #if tight_plot is True and aspect_auto
     is True and plot_version == 1:
2347                 #    plt.tight_layout()
2348             elif filename == 'DATA' and plot_type == '2d
     ' and x_property_chosen in ['X','Y','Z'] and
     y_property_chosen in ['X','Y','Z']:
2349                 core_path = simcase_path
2350                 datatypes = {'X': 'DX', 'Y': 'DY', 'Z':
     'DZ'}
2351                 dimi, dimj = (datatypes[
     x_property_chosen], datatypes[y_property_chosen])
2352                 dims = self.get_input(core_path=
     core_path, property_chosen=[x_property_chosen,
     y_property_chosen, dimi, dimj],
2353                                       ilist=ilist, jlist
     =jlist, klist=klist)
2354                 xi, yj, dxi, dyj = (list(dims[
     x_property_chosen]), list(dims[y_property_chosen]), list
     (dims[dimi]), list(dims[dimj]))
2355 
2356                 xnew = self.get_block_boundaries(
     cellvalues=xi, cellwidths=dxi)
2357                 ynew = self.get_block_boundaries(
     cellvalues=yj, cellwidths=dyj)
2358                 z_id_v2 = self.get_data(core_path=
     core_path, property_chosen=z_property_chosen, allcells=
     allcells, newcells=newcells, ilist=ilist, jlist=jlist,
     klist=klist, tlist=tlist)
2359 
2360                 x_centers = (np.unique(xi)).tolist()
2361                 x_bound = (np.unique(xnew)).tolist()
2362                 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                    xlength, ylength = (len(x_id_v2), len(
     y_id_v2))
2367                    transpose_choice = None
2368                    if xlength > ylength:
2369                        rowshape, colshape = (ylength - 1,
     xlength - 1)
2370                        transpose_choice = 0
2371                    elif xlength < ylength:
2372                        rowshape, colshape = (xlength - 1,
     ylength - 1)
2373                        transpose_choice = 1
2374                    elif xlength == ylength:
2375                        if (x_property_chosen == 'X' and
     y_property_chosen == 'Y') or (x_property_chosen == 'X'
     and y_property_chosen == 'Z') or (x_property_chosen == '
     Y' and y_property_chosen == 'Z'):
2376                            rowshape, colshape = (ylength -
     1, xlength - 1)
2377                            transpose_choice = 0
2378                        elif (x_property_chosen == 'Y' and
     y_property_chosen == 'X') or (x_property_chosen == 'Z'
     and y_property_chosen == 'X') or (x_property_chosen == '
     Z' and y_property_chosen == 'Y'):
2379                            rowshape, colshape = (xlength -
     1, ylength - 1)
2380                            transpose_choice = 1
2381
2382                    x_grid, y_grid = np.meshgrid(x_id_v2,
     y_id_v2)
2383                    z_grid = np.reshape(np.array(z_id_v2), (
     rowshape, colshape))
2384
2385                    fig_plotted, ax = (fig_plotted + 1, None
     )
2386                    row_id, col_id = self.find_row_col(
     identifier, chosen_rows, chosen_cols)
2387
2388                    if plot_version == 0:
2389                        subplot_id = str(chosen_rows) + str(
     chosen_cols) + str(identifier)
2390                        if not prev_ax:
2391                            ax = fig.add_subplot(chosen_rows
```

```
2391 , chosen_cols, identifier)
2392                     else:
2393                         if sharex_local == 'all' and
     sharey_local != 'all':
2394                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2395                         elif sharex_local != 'all' and
     sharey_local == 'all':
2396                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2397                         elif sharex_local == 'all' and
     sharey_local == 'all':
2398                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2399                         else:
2400                             ax = fig.add_subplot(
     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
     == 1:  # Only plot one figure
2405                         ax = axes
2406                     elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2407                         ax = axes[col_id]
2408                         tight_plot = True
2409                     elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2410                         ax = axes[row_id]
2411                     elif chosen_rows != 1 and
     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,
     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.yaxis.set_tick_params(which='both',
     labelbottom=True)
2424                 norm = clr.Normalize()
2425                 cmap = cm.get_cmap('gist_rainbow')
2426 
2427                 if transpose_choice == 0:
2428                     im = ax.pcolormesh(x_grid, y_grid,
     z_grid, cmap=cmap, norm=norm)
2429                 elif transpose_choice == 1:
2430                     im = ax.pcolormesh(x_grid, y_grid,
     z_grid.T, cmap=cmap, norm=norm)
2431 
2432                 print('...')
2433                 print(str(x_property_chosen) + str(
     y_property_chosen) + ': transposed?? ' + str(
     transpose_choice))
2434                 print('xlength: ' + str(len(x_id_v2)) +
     ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
     len(z_id_v2)))
2435                 print('(rowshape, colshape): (' + str(
     rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
      '[' + str(coltype) + '])')
2436                 print('xshape: ' + str(x_grid.shape) +
     ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
     z_grid.shape))
2437                 print('...')
2438 
2439                 if grid_xticks:
2440                     ax.set_xticks(grid_xticks)
2441                 if grid_yticks:
2442                     ax.set_yticks(grid_yticks)
2443 
2444                 if aspect_auto is False:
2445                     aspect_ratio_wanted = aspect_wanted
2446                     aspect_ratio_correct = abs((x_max -
     x_min) / (y_max - y_min)) / aspect_ratio_wanted
2447                     ax.set_aspect(aspect_ratio_correct)
2448 
2449                 fig.colorbar(im, ax=ax)
2450 
2451                 plt.tight_layout()
2452 
2453                 #if tight_plot is True and aspect_auto
     is True and plot_version == 1:
```

```
2454                 #    plt.tight_layout()
2455             elif filename == 'INPUT' and plot_type == '
     2d' and x_property_chosen in ['X','Y','Z'] and
     y_property_chosen in ['X','Y','Z']:
2456                 core_path = simcase_path
2457                 datatypes = {'X': 'DX', 'Y': 'DY', 'Z':
     'DZ'}
2458                 dimi, dimj = (datatypes[
     x_property_chosen], datatypes[y_property_chosen])
2459                 dims = self.get_input(core_path=
     core_path, property_chosen=[x_property_chosen,
     y_property_chosen, z_property_chosen, dimi, dimj],
2460                                       ilist=ilist, jlist
     =jlist, klist=klist)
2461                 xi, yj, dxi, dyj = (list(dims[
     x_property_chosen]), list(dims[y_property_chosen]), list
     (dims[dimi]), list(dims[dimj]))
2462                 listitems = [xi, yj]
2463                 for litem in listitems:
2464                     litem = [float(i) for i in litem]
2465
2466                 xnew = self.get_block_boundaries(
     cellvalues=xi, cellwidths=dxi)
2467                 ynew = self.get_block_boundaries(
     cellvalues=yj, cellwidths=dyj)
2468                 z_id_v2 = [float(i) for i in list(dims[
     z_property_chosen])]
2469
2470                 x_centers = list(np.unique(xi))
2471                 x_bound = list(np.unique(xnew))
2472                 y_centers = list(np.unique(yj))
2473                 y_bound = list(np.unique(ynew))
2474
2475                 x_id_v2, y_id_v2 = (x_bound, y_bound)
2476                 xlength, ylength = (len(x_id_v2), len(
     y_id_v2))
2477                 transpose_choice = None
2478                 if xlength >= ylength:
2479                     rowshape, colshape = (ylength - 1,
     xlength - 1)
2480                     transpose_choice = 0
2481                 elif xlength < ylength:
2482                     rowshape, colshape = (xlength - 1,
     ylength - 1)
2483                     transpose_choice = 1
```

```
2484
2485                 x_grid, y_grid = np.meshgrid(x_id_v2,
     y_id_v2)
2486                 z_grid = np.reshape(np.array(z_id_v2), (
     rowshape, colshape))
2487
2488                 fig_plotted, ax = (fig_plotted + 1, None
     )
2489                 row_id, col_id = self.find_row_col(
     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
     , chosen_cols, identifier)
2494                     else:
2495                         if sharex_local == 'all' and
     sharey_local != 'all':
2496                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2497                         elif sharex_local != 'all' and
     sharey_local == 'all':
2498                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2499                         elif sharex_local == 'all' and
     sharey_local == 'all':
2500                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2501                         else:
2502                             ax = fig.add_subplot(
     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
     == 1:  # Only plot one figure
2507                         ax = axes
2508                     elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2509                         ax = axes[col_id]
2510                         tight_plot = True
2511                     elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2512                         ax = axes[row_id]
```

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

```
2542 transpose_choice))
2543                 print('xlength: ' + str(len(x_id_v2)) +
     ' ylength: ' + str(len(y_id_v2)) + ' zlength: ' + str(
     len(z_id_v2)))
2544                 print('(rowshape, colshape): (' + str(
     rowshape) + '[' + str(rowtype) + '], ' + str(colshape) +
      '[' + str(coltype) + '])')
2545                 print('xshape: ' + str(x_grid.shape) +
     ' yshape: ' + str(y_grid.shape) + ' zshape: ' + str(
     z_grid.shape))
2546                 print('...')
2547
2548                 if grid_xticks:
2549                     ax.set_xticks(grid_xticks)
2550                 if grid_yticks:
2551                     ax.set_yticks(grid_yticks)
2552
2553                 if aspect_auto is False:
2554                     aspect_ratio_wanted = aspect_wanted
2555                     aspect_ratio_correct = abs((x_max -
     x_min) / (y_max - y_min)) / aspect_ratio_wanted
2556                     ax.set_aspect(aspect_ratio_correct)
2557
2558                 fig.colorbar(im, ax=ax)
2559
2560                 plt.tight_layout()
2561
2562                 #if tight_plot is True and aspect_auto
     is True and plot_version == 1:
2563                 #    plt.tight_layout()
2564             elif filename == 'COMP':
2565                 core_path = simcase_path
2566                 path_ar_comp = os.path.join(core_path, '
     COMP' + '.parquet')
2567                 path_ar_time = os.path.join(core_path, '
     TIME' + '.parquet')
2568                 x_comp = pd.read_parquet(path_ar_comp)
2569                 x_time = pd.read_parquet(path_ar_time)
2570                 pressure = [simcase_child[1][1]]
2571                 component = [simcase_child[1][2]]
2572                 compbase = x_comp.loc[(x_comp.index.isin
     (time_element[0])) & (x_comp['Component'].isin(component
     )) & (x_comp['Pressure'].isin(pressure)),
     z_property_chosen]
2573                 timebase = compbase.index.tolist()
```

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

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

```
2638 sharey_local != 'all':
2639                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2640                         elif sharex_local != 'all' and
     sharey_local == 'all':
2641                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2642                         elif sharex_local == 'all' and
     sharey_local == 'all':
2643                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2644                         else:
2645                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier)
2646                 elif plot_version == 1:
2647                     if (row_id, col_id) == (None, None):
2648                         break
2649                     if chosen_rows == 1 and chosen_cols
     == 1:  # Only plot one figure
2650                         ax = axes
2651                     elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2652                         ax = axes[col_id]
2653                         tight_plot = True
2654                     elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2655                         ax = axes[row_id]
2656                     elif chosen_rows != 1 and
     chosen_cols != 1:  # Use both row_id and col_id
2657                         ax = axes[row_id, col_id]
2658                 ax.set_title(title)
2659                 xlabel, ylabel = ('Days',
     z_property_chosen)
2660                 if new_xlabel:
2661                     xlabel = new_xlabel
2662                 if new_ylabel:
2663                     ylabel = new_ylabel
2664                 ax.set_xlabel(xlabel)
2665                 ax.set_ylabel(ylabel)
2666                 ax.plot(timesnew, regbase)
2667             elif filename == 'WELLS':
2668                 core_path = simcase_path
2669                 path_ar_wells = os.path.join(core_path,
     '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(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax)
2705                         elif sharex_local != 'all' and
     sharey_local == 'all':
2706                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharey=prev_ax)
2707                         elif sharex_local == 'all' and
     sharey_local == 'all':
2708                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier, sharex=prev_ax,
     sharey=prev_ax)
2709                         else:
2710                             ax = fig.add_subplot(
     chosen_rows, chosen_cols, identifier)
2711                 elif plot_version == 1:
2712                     if (row_id, col_id) == (None, None):
2713                         break
2714                     if chosen_rows == 1 and chosen_cols
     == 1:  # Only plot one figure
2715                         ax = axes
2716                     elif chosen_rows == 1 and
     chosen_cols != 1:  # Only plot against col_id
2717                         ax = axes[col_id]
2718                         tight_plot = True
2719                     elif chosen_rows != 1 and
     chosen_cols == 1:  # Only plot against row_id
2720                         ax = axes[row_id]
2721                     elif chosen_rows != 1 and
     chosen_cols != 1:  # Use both row_id and col_id
2722                         ax = axes[row_id, col_id]
2723                 ax.set_title(title)
2724                 xlabel, ylabel = ('Days',
     z_property_chosen)
2725                 if new_xlabel:
2726                     xlabel = new_xlabel
2727                 if new_ylabel:
2728                     ylabel = new_ylabel
2729                 ax.set_xlabel(xlabel)
2730                 ax.set_ylabel(ylabel)
2731                 ax.plot(timesnew, wellbase)
2732             prev_ax = ax
2733
2734         if plot_version == 1:
2735             for delete_empty_fig in list(range(
```

```
2735 fig_plotted + 1, chosen_cols * chosen_rows + 1)):
2736                 row_id_del, col_id_del = self.
     find_row_col(delete_empty_fig, chosen_rows, chosen_cols)
2737                 if chosen_rows == 1 and chosen_cols != 1
     :
2738                     axes[col_id_del].remove()
2739                     axes[col_id_del] = None
2740                 elif chosen_rows != 1 and chosen_cols ==
      1:
2741                     axes[row_id_del].remove()
2742                     axes[row_id_del] = None
2743                 elif chosen_rows != 1 and chosen_cols !=
      1:
2744                     axes[row_id_del, col_id_del].remove(
     )
2745                     axes[row_id_del, col_id_del] = None
2746                 elif chosen_rows == 1 and chosen_cols ==
      1:
2747                     axes.remove()
2748                     axes = None
2749
2750         if alls:
2751             self.canvas = FigureCanvasTkAgg(fig, self.
     f2_plot)
2752             self.canvas.draw()
2753             self.canvas.get_tk_widget().pack(side=tk.TOP
     , fill=tk.BOTH, expand=True)
2754             self.canvas._tkcanvas.pack(side=tk.BOTTOM,
     fill=tk.BOTH, expand=True)
2755             self.toolbar = NavigationToolbar2Tk(self.
     canvas, self.f2_toolkit)  # Toolbar is added to canvas
2756             self.toolbar.update()
2757             if self.hold.get() == 1:
2758                 self.figs = 0
2759                 self.label_figs['text'] = 'Figures: ' +
     str(self.figs)
2760                 self.grid_size_figures()
2761                 for i_del in reversed(alls):
2762                     x_del = i_del
2763                     child.delete(alls.index(x_del))
2764
2765     def clear_xy(self, typedata):
2766         child = None
2767         if typedata == 'X':
2768             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            childy = 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
    cells and times manually
2818
2819            if self.plot_id:
2820                child = self.pageone_listbox_plot  # Listbox
     where to insert
2821                data = self.plot_id  # New data potentially
    incoming
2822                core_path = self.plot_id['simcase_path']
2823                cells, time_element = self.get_cell_time(
    corepath=core_path)
2824                ilist, jlist, klist, tlist = (time_element[2
    ], time_element[3], time_element[4], time_element[5])
2825                timedays = self.get_time(core_path, 'Days',
    tlist).values.tolist()
2826                simcase = self.plot_id['simcase']
2827                simcase_child = self.plot_id['simcase_child'
    ][0]
2828                newvalues = {'cells': cells, 'time':
    time_element, 'X': self.plot_x, 'Y': self.plot_y}
2829                self.plot_id.update(newvalues)
2830                entries = list(self.plot_id['entries'])
2831
2832                for item in entries:
2833                    newdict = {}
2834                    for i in list(self.plot_id.keys()):
2835                        if i == 'entries':
2836                            newdict[i] = item
2837                        else:
2838                            newdict[i] = self.plot_id[i]
2839                    alls = list(child.get(0, END))
2840                    count = len(alls) + 1
2841                    marker = self.hold2.get()
2842                    element_shown = str(count) + ': ' + str(
    simcase) + ' ' + str(simcase_child) + ' ' + str(item)
2843                    shown_title = None
2844                    if self.showsimcase.get() == 0:
```

```
2845                    shown_title = str(simcase) + ' ' +
     str(item)
2846                elif self.showsimcase.get() == 1:
2847                    shown_title = str(item)
2848                if self.showtime.get() == 0:
2849                    shown_title = shown_title + ' t=' +
     str(timedays[0]) + ' days'
2850                dict_title = {'shown_title': shown_title
     , 'simcase': str(simcase) + ':', 'parameter': str(item),
      'timedays': timedays, 'xlabel': None, 'ylabel': None}
2851                newdict['title'] = dict_title
2852                newdict['fontsize'] = self.fontsize
2853                element = [count, newdict, marker]
2854                self.plot_rdy[element_shown] = element
2855                child.insert(END, element_shown)
2856            alls = list(child.get(0, END))
2857            self.figs = len(alls)
2858            self.label_figs['text'] = 'Figures: ' + str(
     self.figs)
2859            self.grid_size_figures()
2860
2861    def get_cell_time(self, corepath):
2862        core_path = corepath
2863        path_ar_input = os.path.join(core_path, 'INPUT'
     + '.parquet')
2864        path_ar_time = os.path.join(core_path, 'TIME' +
     '.parquet')
2865        x_input = pd.read_parquet(path_ar_input)
2866        x_time = pd.read_parquet(path_ar_time)
2867        imin, imax = (int(self.slidex_left.get()), int(
     self.slidex_right.get()))
2868        jmin, jmax = (int(self.slidey_left.get()), int(
     self.slidey_right.get()))
2869        kmin, kmax = (int(self.slidez_left.get()), int(
     self.slidez_right.get()))
2870        tmin, tmax = (int(self.slidetime_left.get()),
     int(self.slidetime_right.get()))
2871        ilist = list(range(imin, imax + 1))
2872        jlist = list(range(jmin, jmax + 1))
2873        klist = list(range(kmin, kmax + 1))
2874        tlist = list(range(tmin, tmax + 1))
2875        cells = x_input.loc[(x_input['i'].isin(ilist)) &
      (x_input['j'].isin(jlist)) & (x_input['k'].isin(klist))
     , 'Cell'].tolist()
2876        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
     , tlist]
2879         return cells, time_element
2880 
2881     def remove_from_plot_list(self):
2882         child = self.pageone_listbox_plot
2883         cursors = child.curselection()
2884         alls = None
2885         for item in reversed(cursors):
2886             x_del = child.get(item)
2887             child.delete(item)
2888             numbering = self.plot_rdy[x_del][0] #
     Position before deletion
2889             alls = list(child.get(0, END))
2890             self.plot_rdy.pop(x_del, None)
2891             for i in list(range(numbering, len(alls)+1))
     :
2892                 access = (i-1,)
2893                 old_key = child.get(access)
2894                 old_key_data = self.plot_rdy[old_key]
2895                 self.plot_rdy.pop(old_key, None)
2896                 current_numbering = old_key_data[0]
2897                 current_marker = old_key_data[2]
2898                 new_numbering = current_numbering - 1
2899                 new_key_data = [new_numbering,
     old_key_data[1], current_marker]
2900                 new_key = str(new_numbering) + ':' +
     old_key.split(':')[1]
2901                 self.plot_rdy[new_key] = new_key_data
2902                 child.delete(access)
2903                 child.insert(access, new_key)
2904         self.figs = len(alls)
2905         self.label_figs['text'] = 'Figures: ' + str(self
     .figs)
2906         self.grid_size_figures()
2907 
2908     def get_multiple_items(self, event):
2909         '''Use current selection as 'parent', then tie
     that to how many are selected in the second listbox (
     with the properties)'''
2910         simcase = self.simcase
2911         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.
     local_sim_parameters.get(0, END))
2976                        if alls:
2977                            self.local_sim_parameters.delete
     (0, END)
2978                        for item in values_to_be_inserted:
2979                            self.local_sim_parameters.insert
     (END, item)
2980                except IndexError:
2981                    pass
2982            pass
2983
2984        def get_selected_item_prep(self, event):
2985            if current_tab == 'Page One':
2986                global current_selection
2987                w = event.widget
2988                index = 0
2989                try:
2990                    index = w.curselection()[0]
2991                    value = w.get(index)
2992
2993                    if value != current_selection:
2994                        self.merged_listbox_items = {}
2995                        self.simcase_child = None
2996                        self.clear_xy(typedata='X')
2997                        self.clear_xy(typedata='Y')
2998                        child = self.local_sim_parameters
2999                        child.selection_clear(0, END)
3000                        child.delete(0,END)
3001                        child2 = self.prep_sim_parameters
3002                        child2.selection_clear(0, END)
3003                        child2.delete(0, END)
3004
3005                        # child = self.prep_sim_parameters
3006
3007
3008                        current_selection = value
3009                        self.simcase = value
3010                        values_to_be_inserted = []
3011                        core_path = global_sim_data[value][0
     ]
3012                        print('core_path: ' + str(core_path)
     )
3013                        self.simcase_path = core_path
3014                        ref_values = {'INPUT': 1, 'DATA': 2,
```

```
3014  'TIME': 3, 'COMP': 4, 'REGION': 5, 'WELLS': 6}
3015                      for filename in global_sim_data[
     value][1:]:
3016                          if filename not in ['PARAMETERS'
     , 'WELLPARAM']:
3017                              element = [filename,
     ref_values[filename]]
3018                              values_to_be_inserted.append
     (element)
3019
3020                      sorted_version = sorted(
     values_to_be_inserted, key=lambda x1: x1[1])
3021                      values_to_be_inserted = []
3022                      for i in sorted_version:
3023                          values_to_be_inserted.append(i[0
     ])
3024
3025                      for filename in
     values_to_be_inserted:
3026                          if filename == 'COMP':
3027                              path_ar_comp = os.path.join(
     core_path, 'COMP' + '.parquet')
3028                              x_comp = pd.read_parquet(
     path_ar_comp)
3029
3030                              components = np.unique(
     x_comp['Component'].tolist())
3031                              pressures = np.unique(x_comp
     ['Pressure'])
3032                              for pressure in pressures:
3033                                  for component in
     components:
3034                                      element = ['COMP',
     pressure, component]
3035                                      element_shown = '
     COMP ' + 'P=' + str(pressure) + '  ' + str(component)
3036                                      self.
     merged_listbox_items[element_shown] = element
3037                          elif filename == 'DATA':
3038                              self.merged_listbox_items['
     DATA'] = ['DATA']
3039                          elif filename == 'INPUT':
3040                              self.merged_listbox_items['
     INPUT'] = ['INPUT']
3041                          elif filename == 'REGION':
```

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

```
3069 nWell', adapt_wellname, 'nType', 'Connection']:
3070                                         properties_wells.
     append(column_name)
3071 
3072                                 for well in well_names:
3073                                     well_attributes =
     x_wells.loc[x_wells[adapt_wellname] == well].iloc[0, :]
3074                                     well_number =
     well_attributes['nWell']
3075                                     well_type =
     well_attributes['nType']
3076                                     element = ['WELLS',
     well_number, well, well_type]
3077                                     element_shown = 'WELLS '
      + str(well_number) + ' ' + str(well) + ' ' + str(
     well_type)
3078                                     self.
     merged_listbox_items[element_shown] = element
3079                         values_to_be_inserted = list(self.
     merged_listbox_items.keys())
3080                         alls = list(self.prep_sim_parameters
     .get(0, END))
3081                         if alls:
3082                             self.prep_sim_parameters.delete(
     0, END)
3083                         for item in values_to_be_inserted:
3084                             self.prep_sim_parameters.insert(
     END, item)
3085 
3086                         if self.xyz.get() != 0:
3087                             alls = self.prep_sim_parameters.
     get(0, END)
3088                             indexdata = alls.index('DATA')
3089                             access = (indexdata,)
3090                             self.prep_sim_parameters.
     selection_set(access)
3091                             self.prep_sim_parameters.
     event_generate('<<ListboxSelect>>')
3092 
3093                         core_path = self.simcase_path
3094                         path_ar_input = os.path.join(
     core_path, 'INPUT' + '.parquet')
3095                         path_ar_time = os.path.join(
     core_path, 'TIME' + '.parquet')
3096                         self.x_input = pd.read_parquet(
```

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

```
3130                    if value != current_selection:
3131                        current_selection = value
3132                        values_to_be_inserted = dict_param[
    current_selection][1:]
3133                        if self.doitonce == 0:
3134                            for item in
    values_to_be_inserted:
3135                                self.local_sim_parameters.
    insert(END, item)
3136                            self.doitonce = 1
3137                        else:
3138                            self.properties_available = {}
    # Deleted to refill with new properties
3139                            self.local_sim_parameters.delete
    (0, END)  # Can use a separate parameter for the ones
    they still want plotted
3140                            for item in
    values_to_be_inserted:  # These are just 'potential
    candidates' for plotting (to be ready
3141                                self.local_sim_parameters.
    insert(END, item)  # for the user when he/she needs to
    plot them fast.
3142                            self.doitonce = 0
3143                        core_path = dict_param[value][0]
3144                        path_ar_input = os.path.join(
    core_path, 'INPUT' + '.parquet')
3145                        path_ar_time = os.path.join(
    core_path, 'TIME' + '.parquet')
3146                        self.x_input = pd.read_parquet(
    path_ar_input, columns=['Cell'] + ['i'] + ['j'] + ['k']
    + ['X'] + ['Y'] + ['Z'])
3147                        self.x_time = pd.read_parquet(
    path_ar_time)
3148                        ival, jval, kval = (self.x_input['i'
    ], self.x_input['j'], self.x_input['k'])
3149                        imin, imax, jmin, jmax, kmin, kmax =
     (ival.min(), ival.max(), jval.min(), jval.max(), kval.
    min(), kval.max())
3150                        imin, imax, jmin, jmax, kmin, kmax =
     (int(imin), int(imax), int(jmin), int(jmax), int(kmin),
     int(kmax))
3151                        tmin, tmax = (self.x_time['nStep'].
    min(), self.x_time['nStep'].max())
3152                        tmin, tmax = (int(tmin), int(tmax))
3153                        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,
     imax, jmin, jmax, kmin, kmax, tmin, tmax)
3159             except IndexError:
3160                 pass
3161 
3162     def left_range_x(self, val):
3163         w1, w2, label1, label2 = (self.slidex_left, self
     .slidex_right, self.slidex_label1, self.slidex_label2)
3164         range_type, value, lower, upper = (['LEFT', 'X']
     , int(self.valuex1.get()), int(val), int(label2['text'])
     )
3165         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3166 
3167     def right_range_x(self, val):
3168         w1, w2, label1, label2 = (self.slidex_left, self
     .slidex_right, self.slidex_label1, self.slidex_label2)
3169         range_type, value, lower, upper = (['RIGHT', 'X'
     ], int(self.valuex1.get()), int(label1['text']), int(val
     ))
3170         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3171 
3172     def left_range_y(self, val):
3173         w1, w2, label1, label2 = (self.slidey_left, self
     .slidey_right, self.slidey_label1, self.slidey_label2)
3174         range_type, value, lower, upper = (['LEFT', 'Y']
     , int(self.valuey1.get()), int(val), int(label2['text'])
     )
3175         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3176 
3177     def right_range_y(self, val):
3178         w1, w2, label1, label2 = (self.slidey_left, self
     .slidey_right, self.slidey_label1, self.slidey_label2)
3179         range_type, value, lower, upper = (['RIGHT', 'Y'
     ], int(self.valuey1.get()), int(label1['text']), int(val
```

```
3179 ))
3180         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3181
3182     def left_range_z(self, val):
3183         w1, w2, label1, label2 = (self.slidez_left, self
     .slidez_right, self.slidez_label1, self.slidez_label2)
3184         range_type, value, lower, upper = (['LEFT', 'Z']
     , int(self.valuez1.get()), int(val), int(label2['text'])
     )
3185         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3186
3187     def right_range_z(self, val):
3188         w1, w2, label1, label2 = (self.slidez_left, self
     .slidez_right, self.slidez_label1, self.slidez_label2)
3189         range_type, value, lower, upper = (['RIGHT', 'Z'
     ], int(self.valuez1.get()), int(label1['text']), int(val
     ))
3190         self.range_calculation(ranger=range_type, w1=w1,
      w2=w2, label1=label1, label2=label2, single=value,
     lower=lower, upper=upper)
3191
3192     def left_range_time(self, val):
3193         w1, w2, label1, label2 = (self.slidetime_left,
     self.slidetime_right, self.slidetime_label1, self.
     slidetime_label2)
3194         range_type, value, lower, upper = ('LEFT', int(
     self.valuetime1.get()), int(val), int(w2.get()))
3195         self.range_calculation_time(ranger=range_type,
     w1=w1, w2=w2, label1=label1, label2=label2, single=value
     , lower=lower, upper=upper)
3196
3197     def right_range_time(self, val):
3198         w1, w2, label1, label2 = (self.slidetime_left,
     self.slidetime_right, self.slidetime_label1, self.
     slidetime_label2)
3199         range_type, value, lower, upper = ('RIGHT', int(
     self.valuetime1.get()), int(w1.get()), int(val))
3200         self.range_calculation_time(ranger=range_type,
     w1=w1, w2=w2, label1=label1, label2=label2, single=value
     , lower=lower, upper=upper)
3201
```

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

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

```
3278                elif lower > upper:
3279                    w1.set(upper)
3280                else:
3281                    label1['text'] = self.browse_days[str(
     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[
     str(upper)]
3287                        label2['text'] = self.browse_days[
     str(upper)]
3288                    else:
3289                        w2.set(lower)
3290                        label2['text'] = self.browse_days[
     str(lower)]
3291                elif upper < lower:
3292                    w2.set(lower)
3293                else:
3294                    label2['text'] = self.browse_days[str(
     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
      if x % 2 == 0 else x+i
3303            figs = self.figs
3304            self.fig_grid_size = {}
3305            self.grid_dropdown.delete(0, END)
3306            if figs == 1:
3307                chosen_cols, chosen_rows = (1, 1)
3308                self.fig_grid_size['1x1'] = [1, 1]
3309                self.grid_dropdown['values'] = ['1x1']
3310                self.grid_dropdown.current(0)
3311            elif figs:
3312                if figs % 2 == 0:
3313                    even = 1
3314                factors01, factors02, even = ([], [], 0)
3315                for i in range(-1, 11, 2):
3316                    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,
     pady=10, expand=1, fill='both')
3359                 gc.collect()
3360             self.figs = 0
3361 
3362     def create_slider_widgets(self, imin, imax, jmin,
     jmax, kmin, kmax, tmin, tmax):
3363         newvalues = {'imin': imin, 'imax': imax, 'jmin':
      jmin, 'jmax': jmax, 'kmin': kmin, 'kmax': kmax, 'tmin':
      tmin, 'tmax': tmax}
3364         self.simcase_ijkt_count.update(newvalues)
3365 
3366         # X-DIRECTION SLIDER
3367         self.slidex_label1 = Label(self.sliders, width=5
     , text=str(imin), bg='white', relief=SUNKEN)
3368         self.slidex_label1.grid(column=0, row=0, sticky=
     'nw', padx=3, pady=3, ipady=2)
3369         self.slidex_left = Scale(self.sliders, from_=
     imin, to=imax, orient=HORIZONTAL, showvalue=0, relief=
     SUNKEN, width=17, command=self.left_range_x)
3370         self.slidex_left.grid(column=1, row=0, sticky='
     nw', padx=3, pady=3)
3371         self.slidex_right = Scale(self.sliders, from_=
     imin, to=imax, orient=HORIZONTAL, showvalue=0, relief=
     SUNKEN, width=17, command=self.right_range_x)
3372         self.slidex_right.grid(column=2, row=0, sticky='
     nw', padx=3, pady=3)
3373         self.slidex_label2 = Label(self.sliders, width=5
     , text=str(imax), bg='white', relief=SUNKEN)
3374         self.slidex_label2.grid(column=3, row=0, sticky=
     'nw', padx=3, pady=3, ipady=2)
3375         self.valuex1 = IntVar()
3376         self.freezex1 = Checkbutton(self.sliders,
     variable=self.valuex1,
3377                                     command=lambda: self
     .freeze_val(self.valuex1, self.slidex_label1, self.
     slidex_label2, self.slidex_right))
3378         self.freezex1.grid(column=4, row=0, sticky='nw',
      padx=3, pady=3)
3379         self.slidex_left.set(imin)
3380         self.slidex_right.set(imax)
3381         # X-DIRECTION SLIDER
3382 
3383 
```

```
3384
3385            # Y-DIRECTION SLIDER
3386            self.slidey_label1 = Label(self.sliders, width=5
    , text=str(jmin), bg='white', relief=SUNKEN)
3387            self.slidey_label1.grid(column=0, row=1, sticky=
    'nw', padx=3, pady=3, ipady=2)
3388            self.slidey_left = Scale(self.sliders, from_=
    jmin, to=jmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.left_range_y)
3389            self.slidey_left.grid(column=1, row=1, sticky='
    nw', padx=3, pady=3)
3390            self.slidey_right = Scale(self.sliders, from_=
    jmin, to=jmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.right_range_y)
3391            self.slidey_right.grid(column=2, row=1, sticky='
    nw', padx=3, pady=3)
3392            self.slidey_label2 = Label(self.sliders, width=5
    , text=str(jmax), bg='white', relief=SUNKEN)
3393            self.slidey_label2.grid(column=3, row=1, sticky=
    'nw', padx=3, pady=3, ipady=2)
3394            self.valuey1 = IntVar()
3395            self.freezey1 = Checkbutton(self.sliders,
    variable=self.valuey1,
3396                                        command=lambda: self
    .freeze_val(self.valuey1, self.slidey_label1, self.
    slidey_label2, self.slidey_right))
3397            self.freezey1.grid(column=4, row=1, sticky='nw',
     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
    , text=str(kmin), bg='white', relief=SUNKEN)
3404            self.slidez_label1.grid(column=0, row=2, sticky=
    'nw', padx=3, pady=3, ipady=2)
3405            self.slidez_left = Scale(self.sliders, from_=
    kmin, to=kmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.left_range_z)
3406            self.slidez_left.grid(column=1, row=2, sticky='
    nw', padx=3, pady=3)
3407            self.slidez_right = Scale(self.sliders, from_=
    kmin, to=kmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.right_range_z)
```

```
3408            self.slidez_right.grid(column=2, row=2, sticky='
    nw', padx=3, pady=3)
3409            self.slidez_label2 = Label(self.sliders, width=5
    , text=str(kmax), bg='white', relief=SUNKEN)
3410            self.slidez_label2.grid(column=3, row=2, sticky=
    'nw', padx=3, pady=3, ipady=2)
3411            self.valuez1 = IntVar()
3412            self.freezez1 = Checkbutton(self.sliders,
    variable=self.valuez1,
3413                                        command=lambda: self
    .freeze_val(self.valuez1, self.slidez_label1, self.
    slidez_label2, self.slidez_right))
3414            self.freezez1.grid(column=4, row=2, sticky='nw',
     padx=3, pady=3)
3415            self.slidez_left.set(kmin)
3416            self.slidez_right.set(kmax)
3417            # Z-DIRECTION SLIDER
3418
3419            # TIME SLIDER time
3420            tmin_time, tmax_time = (self.browse_days[str(
    tmin)], self.browse_days[str(tmax)])
3421            self.tmin_stored, self.tmax_stored = (tmin, tmax
    )
3422            self.slidetime_label1 = Label(self.sliders,
    width=5, text=tmin_time, bg='white', relief=SUNKEN)
3423            self.slidetime_label1.grid(column=0, row=3,
    sticky='nw', padx=3, pady=3, ipady=2)
3424            self.slidetime_left = Scale(self.sliders, from_=
    tmin, to=tmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.left_range_time)
3425            self.slidetime_left.grid(column=1, row=3, sticky
    ='nw', padx=3, pady=3)
3426            self.slidetime_right = Scale(self.sliders, from_
    =tmin, to=tmax, orient=HORIZONTAL, showvalue=0, relief=
    SUNKEN, width=17, command=self.right_range_time)
3427            self.slidetime_right.grid(column=2, row=3,
    sticky='nw', padx=3, pady=3)
3428            self.slidetime_label2 = Label(self.sliders,
    width=5, text=tmax_time, bg='white', relief=SUNKEN)
3429            self.slidetime_label2.grid(column=3, row=3,
    sticky='nw', padx=3, pady=3, ipady=2)
3430            self.valuetime1 = IntVar()
3431            self.freezetime1 = Checkbutton(self.sliders,
    variable=self.valuetime1,
3432                                           command=lambda:
```

```
3432 self.freeze_val_time(self.valuetime1, self.
     slidetime_label1, self.slidetime_label2, self.
     slidetime_right))
3433         self.freezetime1.grid(column=4, row=3, sticky='
     nw', padx=3, pady=3)
3434         self.slidetime_left.set(tmin)
3435         self.slidetime_right.set(tmax)
3436         # TIME SLIDER
3437 
3438     def find_row_col(self, identifier, user_rows,
     user_cols):
3439         ni, chosen_cols, chosen_rows = (identifier,
     user_cols, user_rows)
3440         row, col = (0, 0)
3441         if ni > chosen_cols * chosen_rows:
3442             return None, None
3443         else:
3444             if ni <= chosen_cols:
3445                 row = 0
3446                 col = ni - 1
3447             elif ni > chosen_cols:
3448                 row = 0
3449                 nb = ni
3450                 while nb not in list(range(1,
     chosen_cols + 1)):
3451                     nb = nb - chosen_cols
3452                     row += 1
3453                     col = nb - 1
3454             return row, col
3455 
3456 
3457 class PageTwo(tk.Frame):
3458     def __init__(self, parent, controller):
3459         self.controller = controller
3460         self.parent = parent
3461         tk.Frame.__init__(self, parent)
3462         label = tk.Label(self, text='Page Two', font=
     LARGE_FONT)
3463         label.pack(padx=10, pady=10)
3464 
3465 class PageThree(tk.Frame):
3466     def __init__(self, parent, controller):
3467         self.controller = controller
3468         self.parent = parent
3469         tk.Frame.__init__(self, parent)
```

```
3470         label = ttk.Label(self, text='Page One..', font=
     LARGE_FONT)
3471         label.pack(padx=10, pady=10)
3472
3473 if __name__ == '__main__':
3474     app = SimPlotJIN()
3475     app.protocol('WM_DELETE_WINDOW', app.on_closing)
3476     app.mainloop()
3477
3478
3479
```