

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon May 14 16:32:56 2018
5
6 @author: hm1234
7 """
8
9 import os
10 import pickle
11 import matplotlib.pyplot as plt
12 import numpy as np
13 import scipy.interpolate as interpolate
14 from scipy.signal import argrelextrema
15 from mpl_toolkits.mplot3d import Axes3D
16 from scipy.optimize import curve_fit
17
18 from sklearn.preprocessing import PolynomialFeatures
19 from sklearn import linear_model
20
21 import pandas as pd
22
23 from sklearn.gaussian_process import GaussianProcessRegressor
24 from sklearn.gaussian_process.kernels import RBF
25 import scipy.stats as st
26
27 from more_itertools import flatten
28
29 #####
30 ##### load the pickles
31
32 os.chdir('./pickles')
33
34 with open('psi_r-28819.pickle', 'rb') as handle:
35     psi_r-28819 = pickle.load(handle)
36 with open('psi_r-30417.pickle', 'rb') as handle:
37     psi_r-30417 = pickle.load(handle)
38 with open('psi_rz-28819.pickle', 'rb') as handle:
39     psi_rz-28819 = pickle.load(handle)
40 with open('psi_rz-30417.pickle', 'rb') as handle:
41     psi_rz-30417 = pickle.load(handle)
42
43 with open('ayc_r-28819.pickle', 'rb') as handle:
44     ayc_r-28819 = pickle.load(handle)
45 with open('ayc_ne-28819.pickle', 'rb') as handle:
46     ayc_ne-28819 = pickle.load(handle)
47 with open('ayc_te-28819.pickle', 'rb') as handle:
48     ayc_te-28819 = pickle.load(handle)
49 with open('ayc_r-30417.pickle', 'rb') as handle:
50     ayc_r-30417 = pickle.load(handle)
51 with open('ayc_ne-30417.pickle', 'rb') as handle:
52     ayc_ne-30417 = pickle.load(handle)
53 with open('ayc_te-30417.pickle', 'rb') as handle:
54     ayc_te-30417 = pickle.load(handle)
55
56 with open('efm_grid_r-28819.pickle', 'rb') as handle:
57     efm_grid_r-28819 = pickle.load(handle)
58 with open('efm_grid_z-28819.pickle', 'rb') as handle:
59     efm_grid_z-28819 = pickle.load(handle)
60 with open('efm_grid_r-30417.pickle', 'rb') as handle:

```

```

61     efm_grid_r_30417 = pickle.load(handle)
62 with open('efm_grid_z_30417.pickle', 'rb') as handle:
63     efm_grid_z_30417 = pickle.load(handle)
64
65 with open('efm_psi_axis_28819.pickle', 'rb') as handle:
66     efm_psi_axis_28819 = pickle.load(handle)
67 with open('efm_psi_boundary_28819.pickle', 'rb') as handle:
68     efm_psi_boundary_28819 = pickle.load(handle)
69 with open('efm_psi_axis_30417.pickle', 'rb') as handle:
70     efm_psi_axis_30417 = pickle.load(handle)
71 with open('efm_psi_boundary_30417.pickle', 'rb') as handle:
72     efm_psi_boundary_30417 = pickle.load(handle)
73
74 os.chdir('../')
75
76 #####
77 ##### let's declare some arrays
78
79 # 2-dimensional (r,z) at different times
80 # psi as function of radius in m
81 psi_x = psi_rz_28819['x']
82 # psi as function of time in s
83 psi_t = psi_rz_28819['time']
84 # value of psi at specific radius, time, and z
85 psi_dat = psi_rz_28819['data']
86
87 # psi grid values
88 # same as psi_x
89 efm_grid_r = efm_grid_r_28819['data'].squeeze()
90 # would be psi_z i.e. z location of flux
91 efm_grid_z = efm_grid_z_28819['data'].squeeze()
92
93 # channel number vs radius
94 # x is the channel number
95 ayc_r_x = ayc_r_28819['x']
96 # t is the time
97 ayc_r_t = ayc_r_28819['time']
98 # dat is the radius corresponding to specific channel number at some time t
99 ayc_r_dat = ayc_r_28819['data']
100
101 # electron temperature data given as a function of time and channel number
102 # channel number
103 ayc_te_x = ayc_te_28819['x']
104 # time
105 ayc_te_t = ayc_te_28819['time']
106 # T_e at channel number and time
107 ayc_te_dat = ayc_te_28819['data']
108
109 # magnetic axis and psi boundary
110 # magnetic axis
111 mag_axis = efm_psi_axis_28819['data']
112 mag_axis_tme = efm_psi_axis_28819['time']
113 # psi boundary
114 psi_bound = efm_psi_boundary_28819['data']
115 psi_bound_tme = efm_psi_boundary_28819['time']
116
117 # arbitrary value to check slices of data
118 chk_t = 44
119 chk_x = 44
120 # marker size for plotting

```

```

121 mrk = 2
122
123 tme = ayc_te_t
124 psi_ch = np.linspace(1, len(ayc_te_x), len(ayc_te_x))
125
126 # there is wobble in the ayc_r_dat that means the channel number as a fnction
127 # of the radial positon changes with time by a very small amount
128 # defining psi_rng basically ignores these tiny perturbations
129 psi_rng = np.linspace(np.amin(ayc_r_dat), np.amax(ayc_r_dat),
130                        ayc_r_dat.shape[1])
131
132 psi_N_rng = np.linspace(-1, 1, 200)
133
134 #####
135 #####    useful functions
136
137 # finds array index corresponding to array value closest to some value
138 def find_closest(data, v):
139     return (np.abs(data-v)).argmin()
140
141 def nan_finder(y):
142     return np.isnan(y), lambda z: z.nonzero()[0]
143
144 def nan_interp(arr):
145     '''
146     insert a 2d array with length len(tme)
147     '''
148     y = np.copy(arr)
149     for i in range(len(tme)):
150         nans, x = nan_finder(y[i])
151         y[i][nans] = np.interp(x(nans), x(~nans), y[i][~nans])
152     return y
153
154 #####
155 #####    do some stuff
156
157 # find the z=0 coordinate
158 z0_axis = np.where(efm_grid_z == 0)[0][0]
159 # define psi only along the z0 axis
160 psi_dat_z0 = psi_dat[:, z0_axis, :]
161
162 # used to see how good the interpolation is in the time axis
163 interp_test = interpolate.interp1d(psi_t, psi_dat_z0[:, chk_x],
164                                    kind='cubic', fill_value='extrapolate')
165 test = interp_test(ayc_r_t)
166
167 # used to see how good the interpolation is in the radial direction
168 interp_test = interpolate.interp1d(psi_x, psi_dat_z0[chk_t],
169                                    kind='cubic', fill_value='extrapolate')
170 test2 = interp_test(psi_rng)
171
172 #####
173 #####    interpolation
174
175 # interpolation of psi data so that it corresponds to the same channel number
176 # and time as the electron temperature data
177 # perform 2 seperate 1d interpolations. Not ideal but was struggling with 2d
178 # interpolation. Have some fun trying scikit learn and knn approach :D
179
180 '''

```

```

181 for j in range(len(ayc_r['data'][0,:])):
182     R_channel_av = ayc_r['data'][:,j].mean()
183     for i in range(len(psi['time'])):
184         psi_channel_t[j,i] = InterpolatedUnivariateSpline(psi['x'],
185                                                             psi['data'][i,32,:])(R_channel_av)
186     '',
187
188 # time axis for psi_boundary data needs to be interpolated to coincide with tme
189 bound_interp = interpolate.interp1d(psi_bound_tme, psi_bound, kind='cubic',
190                                     fill_value='extrapolate')
191 psi_boundary = bound_interp(tme)
192
193 # mag axis data interpolated
194 mag_axis_interp_tmp = interpolate.interp1d(mag_axis_tme, mag_axis,
195                                             kind='cubic',
196                                             fill_value='extrapolate')
197 mag_axis_interp = mag_axis_interp_tmp(tme)
198
199 def interp_1d():
200     psi_t_interp = []
201     for i in range(0, len(psi_x)):
202         interp_test = interpolate.interp1d(psi_t, psi_dat_z0[:,i], kind='cubic',
203                                             fill_value='extrapolate')
204         psi_t_interp.append(interp_test(ayc_r_t))
205     psi_t_interp = np.array(psi_t_interp).T
206     # psi_t_interp is psi but with same time values as T_e data
207
208     psi_x_interp = []
209     for i in range(0, psi_t_interp.shape[0]):
210         interp_test = interpolate.interp1d(psi_x, psi_t_interp[i], kind='cubic',
211                                             fill_value='extrapolate')
212         psi_x_interp.append(interp_test(ayc_r_dat[i]))
213         #psi_x_interp.append(interp_test(psi_rng))
214     psi_x_interp = np.array(psi_x_interp)
215     return psi_x_interp
216
217 # psi_t_interp is psi but with same channel number values as T_e data
218 # since the time data is also the same the outputted array should be the
219 # correct shape
220 psi_dat_z0_new = interp_1d()
221
222 def interp_2d():
223     f = interpolate.interp2d(psi_x, psi_t, psi_dat_z0, kind='cubic',
224                             fill_value='extrapolate')
225     f_interp = f(psi_rng, tme)
226     return f_interp
227
228 psi_dat_z0_new2 = interp_2d()
229
230
231 #####
232 ##### Normalisation
233
234 mag_ax_psi = []
235 mag_ax = []
236 norm_ind = []
237 for i in range(len(tme)):
238     a = np.where(psi_dat_z0_new2[i] == np.amax(psi_dat_z0_new2[i]))[0][0]
239     mag_ax_psi.append(psi_dat_z0_new2[i][a])
240     mag_ax.append(ayc_r_dat[i][a])

```

```

241     norm_ind.append(a)
242 mag_ax_psi = np.array(mag_ax_psi)
243 mag_ax_r = np.array(mag_ax)
244 norm_ind = np.array(norm_ind)
245
246 psi_N = []
247 for i in range(len(tme)):
248     psi_N_temp = ( (psi_dat_z0_new2[i] - mag_ax_psi[i]) /
249                   (psi_boundary[i] - mag_ax_psi[i]) )
250     psi_N_temp[norm_ind[i]:,] = -psi_N_temp[norm_ind[i]:,]
251     psi_N.append(-psi_N_temp)
252 psi_N = np.array(psi_N)
253
254
255 #####
256 #####    get rid of NaNs
257
258 Te = []
259 psi_fin = []
260
261 for i in range(len(tme)):
262     nan_drop = np.where(np.isnan(ayc_te_dat[i]) == False)[0]
263     Te.append(ayc_te_dat[i][nan_drop])
264     psi_fin.append(psi_N[i][nan_drop])
265
266 Te = np.array(Te)
267 psi_fin = np.array(psi_fin)
268
269 #psi_fin2 = []
270 #Te2 = []
271 #for i in range(len(tme)):
272 #    norm_rng = np.where(np.logical_and(psi_N[i] >= -1, psi_N[i] <= 1))
273 #    #print(norm_rng)
274 #    psi_fin2.append(psi_N[i][norm_rng])
275 #    Te2.append(ayc_te_dat[i][norm_rng])
276
277 #####
278 #####    get the same psi values for all Te
279
280
281 def same_psi():
282     y2 = []
283     y = nan_interp(ayc_te_dat)
284     for i in range(len(tme)):
285         f = interpolate.interp1d(psi_N[i], y[i], kind='nearest',
286                                 fill_value='extrapolate')
287         y2.append(f(psi_N_rng))
288         #print(y2[i])
289     return y2
290
291 def same_psi2():
292     y2 = []
293     y = nan_interp(ayc_te_dat)
294     for i in range(len(tme)):
295         f = interpolate.interp1d(psi_N[i], y[i], kind='linear',
296                                 fill_value='extrapolate')
297         fl = np.floor(psi_N[i][0])
298         if fl > -1:
299             fl = -1
300         cl = np.ceil(psi_N[i][-1])

```

```

301         if cl < 1:
302             cl = 1
303             rng = np.linspace(fl, cl, int(200*(cl-fl)) + 1)
304             norm_rng = np.where(np.logical_and(rng >= -1, rng <= 1))
305             y2.append(f(rng[norm_rng]))
306             #print(y2[i])
307             psi_rng2 = rng[norm_rng]
308             return y2, psi_rng2
309
310 Te_interp = same_psi()
311 Te_interp2, psi_rng2 = same_psi2()
312
313 #####
314 #####      append range of psi to -1 and 1
315
316 psi_fin2 = []
317 Te2 = []
318 for i in range(len(tme)):
319     cp_psi = np.copy(psi_N[i])
320     cp_te = np.copy(ayc_te_dat[i])
321     norm_rng = np.where(np.logical_or(cp_psi <= -1, cp_psi >= 1))
322     cp_psi[norm_rng] = np.NaN
323     cp_te[norm_rng] = np.NaN
324     psi_fin2.append(cp_psi)
325     Te2.append(cp_te)
326
327 #####
328 #####      get rid of anomalies
329
330 Te_peaks = []
331 psi_peaks = []
332
333 for i in range(len(tme)):
334     peaks = argrextrema(Te[i], np.greater, order=4)
335     Te_peaks.append(Te[i][peaks])
336     psi_peaks.append(psi_fin[i][peaks])
337
338 Te_peaks = np.array(Te_peaks)
339 psi_peaks = np.array(psi_peaks)
340
341
342
343 # smooths by looking at windows of data with length n_slice and calculating
344 # the average and standard deviation. If any point is too far from the mean in
345 # in the window then its value is set to the average value in the window
346 def smooth(arr=Te2, n_slice=6, smooth_rep=2, w=1.5):
347     for hi in range(0, smooth_rep):
348         for k in range(len(tme)):
349             tmp = np.array_split(arr[k], len(arr[k])/n_slice)
350             for j in range(len(tmp)):
351                 m = np.mean(tmp[j])
352                 s = np.std(tmp[j])
353                 for n, i in enumerate(tmp[j]):
354                     if int(i) > m + w*s or int(i) < m - w*s:
355                         not_ind = np.where(tmp[j]) != int(i)
356                         m2 = np.mean(tmp[j][not_ind][0])
357                         tmp[j][n] = m2
358
359
360 def nan_smooth(arr=Te2, n_slice=4, smooth_rep=2, w=1.5):

```

```

361     for hi in range(0,smooth_rep):
362         for k in range(len(tme)):
363             tmp = np.array_split(arr[k], len(arr[k])/n_slice)
364             for j in range(len(tmp)):
365                 if np.all(np.isnan(tmp[j])) == False:
366                     m = np.nanmean(tmp[j])
367                     s = np.nanstd(tmp[j])
368                     for n, i in enumerate(tmp[j]):
369                         if np.isnan(i) == False:
370                             if int(i) > m + w*s or int(i) < m - w*s:
371                                 not_ind = np.where(tmp[j] != int(i))
372                                 m2 = np.nanmean(tmp[j][not_ind][0])
373                                 tmp[j][n] = m2
374
375 #smooth(arr=Te_interp2, n_slice=10, smooth_rep=2, w=1)
376 #nan_smooth(n_slice=4, smooth_rep=2, w=1.5)
377 #nan_smooth(arr=Te_interp, n_slice=7, smooth_rep=2, w=1.1)
378
379 #####
380 #####      suface fitting
381
382
383 #Te_interp = Te_interp/np.amax(Te_interp)
384
385 #x = psi_rng2
386 x = psi_N_rng
387 y = tme
388 X1, Y1 = np.meshgrid(x, y, copy=False)
389 Z1 = np.array(Te_interp)
390
391 X = X1.flatten()
392 Y = Y1.flatten()
393
394 #A = np.array([X*0+1, X, Y, X**2, X**2*Y, X**2*Y**2, Y**2, X*Y**2, X*Y]).T
395 #A = np.array([X**2, Y**2, X*Y, X, Y, X*0+1]).T
396 A2 = np.array([X**3, Y**3, (X**2)*Y, (Y**2)*X, X**2, Y**2, X*Y, X, Y, X*0+1]).T
397 #A3= np.array([X**3, (X**2)*Y, X**2, X*Y, X, Y, X*0+1]).T
398 B = Z1.flatten()
399
400 #c1, r1, rank1, s1 = np.linalg.lstsq(A, B)
401 c2, r2, rank2, s2 = np.linalg.lstsq(A2, B)
402 #c3, r3, rank3, s3 = np.linalg.lstsq(A3, B)
403
404 #tst_z = c1[0]*(X1**2) + c1[1]*(Y1**2) + c1[2]*X1*Y1
405 #+ c1[3]*X1 + c1[4]*Y1 + c1[5]*X1*0+1
406
407 tst_z2 = c2[0]*(X1**3) + c2[1]*(Y1**3) + c2[2]*((X1**2)*Y1)
408 + c2[3]*((Y1**2)*X1) + c2[4]*(X1**2) + c2[5]*(Y1**2) + c2[6]*(X1*Y1)
409 + c2[7]*X1 + c2[8]*Y1 + c2[9]*X1*0+1
410
411 #tst_z3 = c3[0]*(X1**3) + c3[1]*((X1**2)*Y1) + c3[2]*(X1**2) - c3[3]*(X1*Y1)
412 #+ c3[4]*X1 - c3[5]*Y1 + c3[6]*X1*0+1
413
414 #####
415 #####      polynomial features
416
417 def bivar_polyfit(deg=5):
418     x = psi_N_rng
419     y = tme
420     X1, Y1 = np.meshgrid(x, y, copy=False)

```

```

421 B = np.array(Te_interp).flatten()
422
423 X = X1.flatten()
424 Y = Y1.flatten()
425
426 aa = np.array(list(zip(X, Y)))
427
428 # rand = np.random.choice(len(B), len(B)/10)
429 # aa = aa[rand]
430 # B = B[rand]
431
432 poly = PolynomialFeatures(deg)
433 X_fit = poly.fit_transform(aa)
434 clf = linear_model.LinearRegression()
435 clf.fit(X_fit, B)
436
437 predict_x = np.concatenate((X1.reshape(-1,1), Y1.reshape(-1,1)), axis=1)
438 pred_x = poly.fit_transform(predict_x)
439 pred_y = clf.predict(pred_x)
440
441 return pred_y.reshape(X1.shape)
442
443 te_fit = abs(bivar_polyfit(6))
444
445 plt.figure()
446 plt.contourf(X1, Y1, te_fit, 66)
447 plt.colorbar()
448
449
450 #fig5 = plt.figure()
451 #ax5 = fig5.add_subplot(111, projection="3d")
452 ##ax5.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
453 ##ax5.plot_surface(X1, Y1, Z2, cmap="autumn", lw=0.5, rstride=1, cstride=1)
454 #ax5.plot_surface(X1, Y1, pred_y.reshape(X1.shape), cmap="jet", lw=0.5,
455 #                 rstride=1, cstride=1, alpha=0.7)
456 ##ax5.plot_surface(X1, Y1, Z4, cmap="summer", lw=0.5, rstride=1, cstride=1)
457 #ax5.grid(False)
458 #plt.show()
459
460 def fit_compare():
461     fig = plt.figure(figsize=(11, 7))
462     ax1 = fig.add_subplot(121, projection='3d')
463     surf1 = ax1.plot_surface(X1, Y1, te_fit, cmap="jet", lw=0.5,
464                             rstride=1, cstride=1, alpha=0.7)
465     # ax1.set_xlim((Length, 0))
466     # ax1.set_ylim((0, Length))
467     fig.colorbar(surf1, ax=ax1)
468     ax2 = fig.add_subplot(122, projection='3d')
469     cs = ax2.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
470     fig.colorbar(cs, ax=ax2)
471     plt.show()
472
473 def fit_compare2():
474     fig = plt.figure(figsize=(11, 7))
475     ax1 = fig.add_subplot(111, projection='3d')
476     ax1.plot_surface(X1, Y1, te_fit, cmap="winter", lw=0.5,
477                     rstride=1, cstride=1, alpha=0.8)
478     ax1.plot_surface(X1, Y1, Te_interp, cmap="autumn", lw=0.5,
479                     rstride=1, cstride=1, alpha=0.4)
480     ax1.set_xlabel('$\Psi_{N}$')

```



```

481 ax1.set_ylabel('time (s)')
482 ax1.set_zlabel('$T_{e}$ (eV)')
483 ax1.grid(False)
484 plt.show()
485
486 def plotly_test():
487     import plotly.plotly as py
488     import plotly.graph_objs as go
489
490     z1 = te_fit
491     z2 = Te_interp
492
493     data = [
494         go.Surface(z=z1, x=psi_N_rng, y=tme, colorscale='Jet'),
495         go.Surface(z=z2, x=psi_N_rng, y=tme, showscale=False,
496                     opacity=0.7, colorscale='autumn'),
497     ]
498
499
500     layout = go.Layout(
501         width=800,
502         height=700,
503         autosize=False,
504         title='T_e fitted with 6th order bivariate polynomial',
505         scene=dict(
506             xaxis=dict(
507                 title = 'Psi-N',
508                 gridcolor='rgb(255, 255, 255)',
509                 zerolinecolor='rgb(255, 255, 255)',
510                 showbackground=True,
511                 backgroundcolor='rgb(230, 230,230)'
512             ),
513             yaxis=dict(
514                 title = 'time(s)',
515                 gridcolor='rgb(255, 255, 255)',
516                 zerolinecolor='rgb(255, 255, 255)',
517                 showbackground=True,
518                 backgroundcolor='rgb(230, 230,230)'
519             ),
520             zaxis=dict(
521                 title = 'T_e (eV)',
522                 gridcolor='rgb(255, 255, 255)',
523                 zerolinecolor='rgb(255, 255, 255)',
524                 showbackground=True,
525                 backgroundcolor='rgb(230, 230,230)'
526             ),
527             aspectratio = dict( x=1, y=1, z=0.7 ),
528             aspectmode = 'manual'
529         )
530     )
531     #py.plot(data)
532     fig = go.Figure(data=data, layout=layout)
533     py.plot(fig)
534
535
536 #data = pd.DataFrame.from_dict({
537 #    'x': psi_N_rng,
538 #    'y': np.random.randint(low=-1, high=1, size=5),
539 #    # 'z': np.random.randint(low=-2, high=5, size=5),
540 #})

```

```

541 #
542 #p = PolynomialFeatures(degree=2).fit(data)
543 #print(p.get_feature_names(data.columns))
544 #
545 #features = pd.DataFrame(p.transform(data),
546 #                          columns=p.get_feature_names(data.columns))
547 #print(features)
548
549
550 #####
551 #####      curve-fit ??
552
553 #coords = (psi_N_rng, tme)
554 #
555 #def func(arr=coords, *p):
556 #    X1, Y1 = arr
557 #    result = p[0]*(X1**3) + p[1]*(Y1**3) + p[2]*((X1**2)*Y1)
558 #    + p[3]*((Y1**2)*X1) + p[4]*(X1**2) + p[5]*(Y1**2) + p[6]*(X1*Y1)
559 #    + p[7]*X1 + p[8]*Y1 + p[9]*X1*0+1
560 #    return result.ravel()
561
562 #xdim = 101
563 #ydim = 90
564 #
565 #x = np.linspace(0.1,1.1,xdim)
566 #y = np.linspace(1.,2., ydim)
567 #X=np.meshgrid(x,y)
568 #a, b, c = 10., 4., 6.
569 #z = func(X, a, b, c) * 1 + np.random.random(xdim*ydim) / 100
570 #
571 #p0 = 8., 2., 7.
572 #print(curve_fit(func, X, z, p0))
573
574
575
576
577 #####
578 #####      2d Gaussian processes – Not smooth enough
579
580 #Te_flat = np.array(Te_interp).flatten()
581 #
582 #xx = np.vstack((X1.flatten(), Y1.flatten())).T
583 #
584 #rand = np.random.choice(range(len(Te_flat)), 2222)
585 #
586 #Te_rand = Te_flat[rand]
587 #xx_rand = xx[rand]
588 #
589 #kernel = RBF()
590 #gp = GaussianProcessRegressor(kernel=kernel,
591 #                               n_restarts_optimizer=10)
592 #gp.fit(xx_rand, Te_rand)
593 #print("Learned kernel", gp.kernel_)
594 #
595 #Te_gpmodel = gp.predict(xx).reshape(-1, len(psi_N_rng))
596 #
597 #plt.contourf(psi_N_rng, tme, Te_gpmodel, 33)
598 #plt.colorbar()
599 #plt.xlabel('$\Psi_{N}$')
600 #plt.ylabel('t (s)')

```

```

601 plt.title('$T_{e}$ (eV)$')
602
603
604
605 #posterior_nums = 3
606 #posteriors = st.multivariate_normal.rvs(mean=y_mean,
607 #                                         size=posterior_nums)
608 #
609 #fig, axs = plt.subplots(posterior_nums+1)
610 #
611 #ax = axs[0]
612 #ax.contourf(psi_N_rng, tme, Te_interp, 33)
613 #ax.plot(X[:, 0], X[:, 1], "r.", ms=12)
614 #
615 #for i, post in enumerate(posteriors, 1):
616 #    axs[i].contourf(psi_N_rng, tme, post.reshape(-1, len(psi_N_rng)))
617 #
618 #plt.show()
619
620
621
622 #####
623 ##### plotting
624
625 def poly2dfit_plot():
626     plt.figure()
627     plt.contourf(x, y, tst_z, 33)
628     plt.colorbar()
629
630     plt.figure()
631     plt.contourf(x, y, tst_z2, 33)
632     plt.colorbar()
633
634     plt.figure()
635     plt.contourf(x, y, tst_z3, 33)
636     plt.colorbar()
637
638     plt.show()
639
640 def psi_N_interps(x=chk_t):
641     plt.figure()
642     plt.plot(psi_fin2[x], Te2[x], label='no smoothing')
643     plt.plot(psi_N_rng, Te_interp[x], label='same $\Psi_{1}$')
644     plt.plot(psi_rng2, Te_interp2[x], label='same $\Psi_{2}$')
645     plt.xlabel('$\Psi_{N}$')
646     plt.ylabel('$T_{e}$ (eV)$')
647     plt.legend()
648     plt.show()
649
650 def Te_vs_psiN():
651     plt.figure()
652     plt.contourf(psi_N_rng, tme, Te_interp, 33)
653     plt.colorbar()
654     plt.xlabel('$\Psi_{N}$')
655     plt.ylabel('t (s)')
656     plt.title('$T_{e}$ (eV)$')
657     plt.show()
658
659 def Te_vs_psiN_3d():
660     # fig = plt.figure()

```

```

661 # ax = fig.add_subplot(111, projection="3d")
662 # ax.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
663 #
664 # fig = plt.figure()
665 # ax = fig.add_subplot(111, projection="3d")
666 # Z2 = tst_z
667 # ax.plot_surface(X1, Y1, Z2, cmap="autumn_r", lw=0.5, rstride=1, cstride=1)
668 #
669 # fig = plt.figure()
670 # ax = fig.add_subplot(111, projection="3d")
671 # Z3 = tst_z2
672 # ax.plot_surface(X1, Y1, Z3, cmap="autumn_r", lw=0.5, rstride=1, cstride=1)
673
674 fig = plt.figure()
675 ax = fig.add_subplot(111, projection="3d")
676 ax.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
677 #ax.plot_surface(X1, Y1, Z2, cmap="autumn", lw=0.5, rstride=1, cstride=1)
678 ax.plot_surface(X1, Y1, tst_z2, cmap="autumn_r", lw=0.5, rstride=1,
679                cstride=1, alpha=0.7)
680 #ax.plot_surface(X1, Y1, Z4, cmap="summer", lw=0.5, rstride=1, cstride=1)
681 ax.grid(False)
682
683 plt.show()
684
685
686 def psi_plot(x=chk_x):
687     plt.figure()
688     plt.plot(tme, psi_dat_z0_new2[:,x], label='$\Psi$ at pos_index {}'.
689             .format(x))
689     plt.plot(tme, mag_axis_interp, label='mag-axis from freia')
690     plt.plot(tme, mag_ax_psi, label='mag-axis from code')
691     plt.plot(tme, psi_boundary, label='boundary from freia')
692     plt.fill_between(tme, mag_ax_psi, psi_boundary, alpha=0.3)
693     plt.xlabel('time (s)')
694     plt.ylabel('$\Psi$', rotation=0)
695     plt.legend()
696     plt.show()
697
698
699 def check_same_psi():
700     for i in range(len(tme)):
701         plt.figure()
702         plt.plot(psi_N_rng, Te[i], 'bx-', ms=2)
703         plt.plot(psi_N[i], ayc_te_dat[i], 'ro-', ms=2)
704         plt.title(i)
705
706 def R_ch():
707     plt.figure()
708     #plt.plot(ayc_r_t, ayc_r_dat[:,20])
709     plt.contourf(ayc_r_dat.T, 11)
710     plt.colorbar()
711     plt.ylabel('channel number')
712     plt.xlabel('time (index value)')
713     plt.title('Radius (m)')
714     plt.show()
715
716 def te_channel():
717     plt.figure()
718     plt.contourf(tme, psi_ch, ayc_te_dat.T, 33)
719     plt.colorbar()
720     plt.xlabel('time (s)')

```

```

721     plt.ylabel('channel number')
722     plt.title('$T_{e}$')
723     plt.show()
724
725 def psi_channel(chk_t=chk_t):
726     plt.figure()
727     plt.contourf(tme, psi_ch, psi_dat_z0_new2.T, 33)
728     plt.colorbar()
729     plt.xlabel('time (s)')
730     plt.ylabel('channel number')
731     plt.title('psi_new at z=0')
732     #
733     #plt.figure()
734     #plt.contourf(psi_t, psi_x, psi_dat_z0.T, 33)
735     #plt.colorbar()
736     #plt.xlabel('time (s)')
737     #plt.ylabel('radial position (m)')
738     #plt.title('psi at z=0')
739
740     plt.figure(chk_t)
741     plt.plot(psi_ch, psi_dat_z0_new2[chk_t])
742     plt.xlabel('channel number')
743     plt.ylabel('$\Psi$', rotation=0)
744     plt.title('for time index {}'.format(chk_t))
745     plt.show()
746
747 def te_x(chk_t=chk_t):
748     # plt.figure()
749     # plt.plot(psi_ch, ayc_te_dat[chk_t])
750     # plt.xlabel('channel number')
751     # plt.ylabel('$T_{e}$', rotation=0)
752     # plt.title('for time index {}'.format(chk_t))
753
754     plt.figure()
755     plt.plot(ayc_r_dat[chk_t], ayc_te_dat[chk_t])
756     plt.xlabel('radial position (m)')
757     plt.ylabel('$T_{e}$', rotation=0)
758     plt.title('for time index {}'.format(chk_t))
759
760     # plt.figure()
761     # plt.plot(ayc_r_x, ayc_r_dat[chk_t])
762     # plt.xlabel('radial index (channel number)')
763     # plt.ylabel('radial position (m)')
764     plt.show()
765
766 def te_psi():
767     plt.figure()
768     #plt.plot(psi_sort, T_e_sort)
769     #plt.plot(psi_fin[chk], Te_fin[chk], 'r')
770     plt.plot(psi_dat_z0_new2[chk_t,:], ayc_te_dat[chk_t,:], 'g')
771     plt.xlabel('$\Psi$')
772     plt.ylabel('$T_{e}$', rotation=0)
773     plt.title('for time index {}'.format(chk_t))
774     plt.show()
775
776 def te_multi_psi(strt=0, stp=len(tme)-1):
777     plt.figure()
778     plt.xlabel('$\Psi$')
779     plt.ylabel('$T_{e}$ (eV)')
780     tmp1 = strt

```

```

781 tmp2 = stp
782 plt.title('time: {} to {}'.format(round(tme[tmp1], 3),
783                                     round(tme[tmp2], 3)))
784 for i in range(tmp1, tmp2):
785     plt.plot(psi_fin2[i], Te2[i])
786     #plt.plot(psi_N[i], ayc_te_dat[i])
787     #plt.plot(psi_peaks[i], Te_peaks[i], 'go')
788
789 plt.figure()
790 plt.xlabel('$\Psi$')
791 plt.ylabel('$T_{\text{e}}$', rotation=0)
792 tmp1 = strt
793 tmp2 = stp
794 plt.title('time: {} to {}'.format(round(tme[tmp1], 3),
795                                     round(tme[tmp2], 3)))
796 for i in range(tmp1, tmp2):
797     plt.plot(psi_N_rng, Te_interp[i])
798     #plt.plot(psi_N[i], ayc_te_dat[i])
799     #plt.plot(psi_peaks[i], Te_peaks[i], 'go')
800 plt.show()
801
802 def psi_rz(cont_type='contour', chk_t=chk_t, res=33):
803     plt.figure()
804     if cont_type == 'contour':
805         plt.contour(efm_grid_r, efm_grid_z, psi_dat[chk_t, :, :], res)
806     elif cont_type == 'contourf':
807         plt.contourf(efm_grid_r, efm_grid_z, psi_dat[chk_t, :, :], res)
808     else:
809         plt.contour(efm_grid_r, efm_grid_z, psi_dat[chk_t, :, :], res)
810     #plt.axis('equal')
811     plt.colorbar()
812     plt.ylabel('z (m)')
813     plt.xlabel('radial position (m)')
814     plt.title('$\Psi$ (r,z) at t = {}'.format(chk_t))
815     plt.show()
816
817 def psi_interp_multi():
818     plt.figure()
819     plt.contourf(psi_dat_z0_new2, 33)
820     plt.colorbar()
821     plt.xlabel('channel number')
822     plt.ylabel('time index')
823     plt.title('$\Psi$ (z=0) interpolated (2D interp)')
824
825     plt.figure()
826     plt.contourf(psi_dat_z0, 33)
827     plt.colorbar()
828     plt.xlabel('channel number')
829     plt.ylabel('time index')
830     plt.title('$\Psi$ (z=0)')
831
832     plt.figure()
833     plt.contourf(psi_dat_z0_new, 33)
834     plt.colorbar()
835     plt.xlabel('channel number')
836     plt.ylabel('time index')
837     plt.title('$\Psi$ (z=0) interpolated (2 x 1D interp)')
838     plt.show()
839
840 def psi_interp_test(chk_x=chk_x, chk_t=chk_t):

```

```

841 plt.figure()
842 plt.plot(psi_t, psi_dat_z0[:,chk_x], 'bx', ms=mrk)
843 plt.plot(ayc_r_t, test, 'ro', ms=mrk)
844 plt.xlabel('time (s)')
845 plt.ylabel('$\Psi$', rotation=0)
846 plt.title('for pos index {}'.format(chk_x))
847 #
848 plt.figure()
849 plt.plot(psi_x, psi_dat_z0[chk_t], 'bx', ms=mrk)
850 plt.plot(ayc_r_dat[chk_t], test2, 'ro', ms=mrk)
851 plt.xlabel('radial position (m)')
852 plt.ylabel('$\Psi$', rotation=0)
853 plt.title('for time index {}'.format(chk_t))
854 plt.show()
855
856 Te_vs_psiN()
857 #psi_N_interps(44)
858 #psi_plot()
859 #te_multi_psi(40,45)
860 #poly2dfit_plot()
861 #Te_vs_psiN_3d()
862 #psi_rz()
863
864 #####
865 ##### Fancy animations
866
867 #for i in range(0, psi_rz_28819['time'].shape[0]):
868 #    fig = plt.figure()
869 #    plt.contour(efm_grid_r.squeeze(), efm_grid_z.squeeze(),
870 #                psi_dat[i,:,:], 50)
871 #    plt.colorbar()
872 #    plt.ylabel('z (m)')
873 #    plt.xlabel('radial position (m)')
874 #    plt.title('$\Psi(r,z)$')
875 #    print(i)
876 #    plt.savefig(str(i).zfill(4) + '.png')
877 #    plt.close(fig)
878
879 #for i in range(0, len(tme)):
880 #    fig = plt.figure()
881 #    plt.plot(psi_dat_z0_new2[i], ayc_te_dat[i])
882 #    plt.xlabel('$\Psi$')
883 #    plt.ylabel('$T_{e}$', rotation=0)
884 #    plt.title('$T_{e}$ vs $\Psi$')
885 #    print(i)
886 #    plt.savefig(str(i).zfill(4) + '.png')
887 #    plt.close(fig)
888
889 #os.chdir('./pics')
890 #for i in range(0, len(tme)):
891 #    fig = plt.figure()
892 #    te_multi_psi(i, i+1)
893 #    print(i)
894 #    plt.savefig(str(i).zfill(4) + '.png')
895 #    plt.close(fig)
896 #os.chdir('../')
897 #plt.show()

```