```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon May 14 16:32:56 2018

@author: hm1234
"""

import os
import pickle
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate as interpolate
from scipy.signal import argrelextrema
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import curve_fit

from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model

import pandas as pd

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
import scipy.stats as st

from more_itertools import flatten

###############################################################################
#####    load the pickles

os.chdir('./pickles')

with open('psi_r_28819.pickle', 'rb') as handle:
    psi_r_28819 = pickle.load(handle)
with open('psi_r_30417.pickle', 'rb') as handle:
    psi_r_30417 = pickle.load(handle)
with open('psi_rz_28819.pickle', 'rb') as handle:
    psi_rz_28819 = pickle.load(handle)
with open('psi_rz_30417.pickle', 'rb') as handle:
    psi_rz_30417 = pickle.load(handle)

with open('ayc_r_28819.pickle', 'rb') as handle:
    ayc_r_28819 = pickle.load(handle)
with open('ayc_ne_28819.pickle', 'rb') as handle:
    ayc_ne_28819 = pickle.load(handle)
with open('ayc_te_28819.pickle', 'rb') as handle:
    ayc_te_28819 = pickle.load(handle)
with open('ayc_r_30417.pickle', 'rb') as handle:
    ayc_r_30417 = pickle.load(handle)
with open('ayc_ne_30417.pickle', 'rb') as handle:
    ayc_ne_30417 = pickle.load(handle)
with open('ayc_te_30417.pickle', 'rb') as handle:
    ayc_te_30417 = pickle.load(handle)

with open('efm_grid_r_28819.pickle', 'rb') as handle:
    efm_grid_r_28819 = pickle.load(handle)
with open('efm_grid_z_28819.pickle', 'rb') as handle:
    efm_grid_z_28819 = pickle.load(handle)
with open('efm_grid_r_30417.pickle', 'rb') as handle:
```

```
    efm_grid_r_30417 = pickle.load(handle)
with open('efm_grid_z_30417.pickle', 'rb') as handle:
    efm_grid_z_30417 = pickle.load(handle)

with open('efm_psi_axis_28819.pickle', 'rb') as handle:
    efm_psi_axis_28819 = pickle.load(handle)
with open('efm_psi_boundary_28819.pickle', 'rb') as handle:
    efm_psi_boundary_28819 = pickle.load(handle)
with open('efm_psi_axis_30417.pickle', 'rb') as handle:
    efm_psi_axis_30417 = pickle.load(handle)
with open('efm_psi_boundary_30417.pickle', 'rb') as handle:
    efm_psi_boundary_30417 = pickle.load(handle)

os.chdir('../')

###############################################################################
#####    let's declare some arrays

# 2-dimensional (r,z) at different times
# psi as function of radius in m
psi_x = psi_rz_28819['x']
# psi as function of time in s
psi_t = psi_rz_28819['time']
# value of psi at specific radius, time, and z
psi_dat = psi_rz_28819['data']

# psi grid values
# same as psi_x
efm_grid_r = efm_grid_r_28819['data'].squeeze()
# would be psi_z i.e. z location of flux
efm_grid_z = efm_grid_z_28819['data'].squeeze()

# channel number vs radius
# x is the channel number
ayc_r_x = ayc_r_28819['x']
# t is the time
ayc_r_t = ayc_r_28819['time']
# dat is the radius correesponding to specific channel number at some time t
ayc_r_dat = ayc_r_28819['data']

# electron temperature data given as a function of time and channel number
# channel number
ayc_te_x = ayc_te_28819['x']
# time
ayc_te_t = ayc_te_28819['time']
# T_e at channel number and time
ayc_te_dat = ayc_te_28819['data']

# magnetic axis and psi boundary
# magnetic axis
mag_axis = efm_psi_axis_28819['data']
mag_axis_tme = efm_psi_axis_28819['time']
# psi boundary
psi_bound = efm_psi_boundary_28819['data']
psi_bound_tme = efm_psi_boundary_28819['time']

# arbitrary value to check slices of data
chk_t = 44
chk_x = 44
# marker size for plotting
```

```
mrk = 2

tme = ayc_te_t
psi_ch = np.linspace(1, len(ayc_te_x), len(ayc_te_x))

# there is wobble in the ayc_r_dat that means the channel number as a fnction
# of the radial positon changes with time by a very small amount
# defining psi_rng basically ignores these tiny perturbations
psi_rng = np.linspace(np.amin(ayc_r_dat), np.amax(ayc_r_dat),
                        ayc_r_dat.shape[1])

psi_N_rng = np.linspace(-1, 1, 200)

##################################################################################
#####    useful functions

# finds array index corresponding to array value closest to some value
def find_closest(data, v):
  return (np.abs(data-v)).argmin()

def nan_finder(y):
    return np.isnan(y), lambda z: z.nonzero()[0]

def nan_interp(arr):
    '''
    insert a 2d array with length len(tme)
    '''
    y = np.copy(arr)
    for i in range(len(tme)):
       nans, x = nan_finder(y[i])
       y[i][nans]= np.interp(x(nans), x(~nans), y[i][~nans])
    return y

##################################################################################
#####    do some stuff

# find the z=0 coordinate
z0_axis = np.where(efm_grid_z == 0)[0][0]
# define psi only along the z0 axis
psi_dat_z0 = psi_dat[:,z0_axis,:]

# used to see how good the interpolation is in the time axis
interp_test = interpolate.interp1d(psi_t, psi_dat_z0[:,chk_x],
                                    kind='cubic', fill_value='extrapolate')
test = interp_test(ayc_r_t)

# used to see how good the interpolation is in the radial direction
interp_test = interpolate.interp1d(psi_x, psi_dat_z0[chk_t],
                                    kind='cubic', fill_value='extrapolate')
test2 = interp_test(psi_rng)

##################################################################################
#####    interpolation

# interpolation of psi data so that it corresponds to the same channel number
# and time as the electron temperature data
# perform 2 seperate 1d interpolations. Not ideal but was struggling with 2d
# interpolation. Have some fun trying scikit learn and knn approach :D

'''
```

```
for j in range(len(ayc_r['data'][0,:])):
  R_channel_av = ayc_r['data'][:,j].mean()
  for i in range(len(psi['time'])):
    psi_channel_t[j,i] = InterpolatedUnivariateSpline(psi['x'],
                   psi['data'][i,32,:])(R_channel_av)
'''

# time axis for psi_boundary data needs to be interpolated to coincide with tme
bound_interp = interpolate.interp1d(psi_bound_tme, psi_bound, kind='cubic',
                                    fill_value='extrapolate')
psi_boundary = bound_interp(tme)

# mag axis data interpolated
mag_axis_interp_tmp = interpolate.interp1d(mag_axis_tme, mag_axis,
                                           kind='cubic',
                                           fill_value='extrapolate')
mag_axis_interp = mag_axis_interp_tmp(tme)

def interp_1d():
    psi_t_interp = []
    for i in range(0, len(psi_x)):
        interp_test = interpolate.interp1d(psi_t, psi_dat_z0[:,i], kind='cubic',
                                           fill_value='extrapolate')
        psi_t_interp.append(interp_test(ayc_r_t))
    psi_t_interp = np.array(psi_t_interp).T
    # psi_t_interp is psi but with same time values as T_e data

    psi_x_interp = []
    for i in range(0, psi_t_interp.shape[0]):
        interp_test = interpolate.interp1d(psi_x, psi_t_interp[i], kind='cubic',
                                           fill_value='extrapolate')
        psi_x_interp.append(interp_test(ayc_r_dat[i]))
        #psi_x_interp.append(interp_test(psi_rng))
    psi_x_interp = np.array(psi_x_interp)
    return psi_x_interp

# psi_t_interp is psi but with same channel number values as T_e data
# since the time data is also the same the outputted array should be the
# correct shape
psi_dat_z0_new = interp_1d()

def interp_2d():
    f = interpolate.interp2d(psi_x, psi_t, psi_dat_z0, kind='cubic',
                             fill_value='extrapolate')
    f_interp = f(psi_rng, tme)
    return f_interp

psi_dat_z0_new2 = interp_2d()


###############################################################################
#####    Normalisation

mag_ax_psi = []
mag_ax = []
norm_ind = []
for i in range(len(tme)):
    a = np.where(psi_dat_z0_new2[i] == np.amax(psi_dat_z0_new2[i]))[0][0]
    mag_ax_psi.append(psi_dat_z0_new2[i][a])
    mag_ax.append(ayc_r_dat[i][a])
```

```
    norm_ind.append(a)
mag_ax_psi = np.array(mag_ax_psi)
mag_ax_r = np.array(mag_ax)
norm_ind = np.array(norm_ind)

psi_N = []
for i in range(len(tme)):
    psi_N_temp = ( (psi_dat_z0_new2[i] - mag_ax_psi[i]) /
                   (psi_boundary[i] - mag_ax_psi[i]) )
    psi_N_temp[norm_ind[i]:,] = -psi_N_temp[norm_ind[i]:,]
    psi_N.append(-psi_N_temp)
psi_N = np.array(psi_N)


##########################################################################################
#####     get rid of NaNs

Te = []
psi_fin = []

for i in range(len(tme)):
    nan_drop = np.where(np.isnan(ayc_te_dat[i]) == False)[0]
    Te.append(ayc_te_dat[i][nan_drop])
    psi_fin.append(psi_N[i][nan_drop])

Te = np.array(Te)
psi_fin = np.array(psi_fin)

#psi_fin2 = []
#Te2 = []
#for i in range(len(tme)):
#    norm_rng = np.where(np.logical_and(psi_N[i] >= -1, psi_N[i] <= 1))
#    #print(norm_rng)
#    psi_fin2.append(psi_N[i][norm_rng])
#    Te2.append(ayc_te_dat[i][norm_rng])

##########################################################################################
#####     get the same psi values for all Te


def same_psi():
    y2 = []
    y = nan_interp(ayc_te_dat)
    for i in range(len(tme)):
        f = interpolate.interp1d(psi_N[i], y[i], kind='nearest',
                                 fill_value='extrapolate')
        y2.append(f(psi_N_rng))
        #print(y2[i])
    return y2

def same_psi2():
    y2 = []
    y = nan_interp(ayc_te_dat)
    for i in range(len(tme)):
        f = interpolate.interp1d(psi_N[i], y[i], kind='linear',
                                 fill_value='extrapolate')
        fl = np.floor(psi_N[i][0])
        if fl > -1:
            fl = -1
        cl = np.ceil(psi_N[i][-1])
```

```
        if cl < 1:
            cl = 1
        rng = np.linspace(fl , cl , int(200*(cl-fl)) + 1)
        norm_rng = np.where(np.logical_and(rng >= -1, rng <= 1))
        y2.append(f(rng[norm_rng]))
        #print(y2[i])
    psi_rng2 = rng[norm_rng]
    return y2, psi_rng2

Te_interp = same_psi()
Te_interp2, psi_rng2 = same_psi2()

################################################################################
#####    append range of psi to -1 and 1

psi_fin2 = []
Te2 = []
for i in range(len(tme)):
    cp_psi = np.copy(psi_N[i])
    cp_te = np.copy(ayc_te_dat[i])
    norm_rng = np.where(np.logical_or(cp_psi <= -1, cp_psi >= 1))
    cp_psi[norm_rng] = np.NaN
    cp_te[norm_rng] = np.NaN
    psi_fin2.append(cp_psi)
    Te2.append(cp_te)


################################################################################
#####    get rid of anomolies

Te_peaks = []
psi_peaks = []

for i in range(len(tme)):
    peaks = argrelextrema(Te[i], np.greater, order=4)
    Te_peaks.append(Te[i][peaks])
    psi_peaks.append(psi_fin[i][peaks])

Te_peaks = np.array(Te_peaks)
psi_peaks = np.array(psi_peaks)


# smooths by looking at windows of data with length n_slice and calculating
# the average and standard deviation. If any point is too far from the mean in
# in the window then its value is set to the average value in the window
def smooth(arr=Te2 ,n_slice=6, smooth_rep=2, w=1.5):
    for hi in range(0,smooth_rep):
        for k in range(len(tme)):
            tmp = np.array_split(arr[k], len(arr[k])/n_slice)
            for j in range(len(tmp)):
                m = np.mean(tmp[j])
                s = np.std(tmp[j])
                for n, i in enumerate(tmp[j]):
                    if int(i) > m + w*s or int(i) < m - w*s:
                        not_ind = np.where(tmp[j]) != int(i)
                        m2 = np.mean(tmp[j][not_ind][0])
                        tmp[j][n] = m2


def nan_smooth(arr=Te2 ,n_slice=4, smooth_rep=2, w=1.5):
```

```
    for hi in range(0,smooth_rep):
        for k in range(len(tme)):
            tmp = np.array_split(arr[k], len(arr[k])/n_slice)
            for j in range(len(tmp)):
                if np.all(np.isnan(tmp[j])) == False:
                    m = np.nanmean(tmp[j])
                    s = np.nanstd(tmp[j])
                    for n, i in enumerate(tmp[j]):
                        if np.isnan(i) == False:
                            if int(i) > m + w*s or int(i) < m - w*s:
                                not_ind = np.where(tmp[j]) != int(i)
                                m2 = np.nanmean(tmp[j][not_ind][0])
                                tmp[j][n] = m2

#smooth(arr=Te_interp2, n_slice=10, smooth_rep=2, w=1)
#nan_smooth(n_slice=4, smooth_rep=2, w=1.5)
#nan_smooth(arr=Te_interp, n_slice=7, smooth_rep=2, w=1.1)


##################################################################################
#####    suface fitting

#Te_interp = Te_interp/np.amax(Te_interp)

#x = psi_rng2
x = psi_N_rng
y = tme
X1, Y1 = np.meshgrid(x, y, copy=False)
Z1 = np.array(Te_interp)

X = X1.flatten()
Y = Y1.flatten()

#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
#A = np.array([X**2, Y**2, X*Y, X, Y, X*0+1]).T
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
#A3= np.array([X**3, (X**2)*Y, X**2, X*Y, X, Y, X*0+1]).T
B = Z1.flatten()

#c1, r1, rank1, s1 = np.linalg.lstsq(A, B)
c2, r2, rank2, s2 = np.linalg.lstsq(A2, B)
#c3, r3, rank3, s3 = np.linalg.lstsq(A3, B)

#tst_z = c1[0]*(X1**2) + c1[1]*(Y1**2) + c1[2]*X1*Y1
#+ c1[3]*X1 + c1[4]*Y1 + c1[5]*X1*0+1

tst_z2 = c2[0]*(X1**3) + c2[1]*(Y1**3) + c2[2]*((X1**2)*Y1)
+ c2[3]*((Y1**2)*X1) + c2[4]*(X1**2) + c2[5]*(Y1**2) + c2[6]*(X1*Y1)
+ c2[7]*X1 + c2[8]*Y1 + c2[9]*X1*0+1

#tst_z3 = c3[0]*(X1**3) + c3[1]*((X1**2)*Y1) + c3[2]*(X1**2) - c3[3]*(X1*Y1)
#+ c3[4]*X1 - c3[5]*Y1 + c3[6]*X1*0+1

##################################################################################
#####    polynomial features

def bivar_polyfit(deg=5):
    x = psi_N_rng
    y = tme
    X1, Y1 = np.meshgrid(x, y, copy=False)
```

```
    B = np.array(Te_interp).flatten()

    X = X1.flatten()
    Y = Y1.flatten()

    aa = np.array(list(zip(X, Y)))

#     rand = np.random.choice(len(B), len(B)/10)
#     aa = aa[rand]
#     B = B[rand]

    poly = PolynomialFeatures(deg)
    X_fit = poly.fit_transform(aa)
    clf = linear_model.LinearRegression()
    clf.fit(X_fit, B)

    predict_x = np.concatenate((X1.reshape(-1,1), Y1.reshape(-1,1)), axis=1)
    pred_x = poly.fit_transform(predict_x)
    pred_y = clf.predict(pred_x)

    return pred_y.reshape(X1.shape)

te_fit = abs(bivar_polyfit(6))

plt.figure()
plt.contourf(X1, Y1, te_fit, 66)
plt.colorbar()


#fig5 = plt.figure()
#ax5 = fig5.add_subplot(111, projection="3d")
##ax5.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
##ax.plot_surface(X1, Y1, Z2, cmap="autumn", lw=0.5, rstride=1, cstride=1)
#ax5.plot_surface(X1, Y1, pred_y.reshape(X1.shape), cmap="jet", lw=0.5,
#                 rstride=1, cstride=1, alpha=0.7)
##ax.plot_surface(X1, Y1, Z4, cmap="summer", lw=0.5, rstride=1, cstride=1)
#ax5.grid(False)
#plt.show()

def fit_compare():
    fig = plt.figure(figsize=(11, 7))
    ax1 = fig.add_subplot(121, projection='3d')
    surf1 = ax1.plot_surface(X1, Y1, te_fit, cmap="jet", lw=0.5,
                              rstride=1, cstride=1, alpha=0.7)
#     ax1.set_xlim((Length, 0))
#     ax1.set_ylim((0, Length))
    fig.colorbar(surf1, ax=ax1)
    ax2 = fig.add_subplot(122, projection='3d')
    cs = ax2.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
    fig.colorbar(cs, ax=ax2)
    plt.show()

def fit_compare2():
    fig = plt.figure(figsize=(11, 7))
    ax1 = fig.add_subplot(111, projection='3d')
    ax1.plot_surface(X1, Y1, te_fit, cmap="winter", lw=0.5,
                     rstride=1, cstride=1, alpha=0.8)
    ax1.plot_surface(X1, Y1, Te_interp, cmap="autumn", lw=0.5,
                     rstride=1, cstride=1, alpha=0.4)
    ax1.set_xlabel('$\Psi_{N}$')
```

```
    ax1.set_ylabel('time (s)')
    ax1.set_zlabel('$T_{e}$ (eV)')
    ax1.grid(False)
    plt.show()

def plotly_test():
    import plotly.plotly as py
    import plotly.graph_objs as go

    z1 = te_fit
    z2 = Te_interp

    data = [
        go.Surface(z=z1, x=psi_N_rng, y=tme, colorscale='Jet'),
        go.Surface(z=z2, x=psi_N_rng, y=tme, showscale=False,
                   opacity=0.7, colorscale='autumn'),
    ]


    layout = go.Layout(
        width=800,
        height=700,
        autosize=False,
        title='T_e fitted with 6th order bivariate polynomial',
        scene=dict(
            xaxis=dict(
                title = 'Psi_N',
                gridcolor='rgb(255, 255, 255)',
                zerolinecolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            ),
            yaxis=dict(
                title = 'time(s)',
                gridcolor='rgb(255, 255, 255)',
                zerolinecolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            ),
            zaxis=dict(
                title = 'T_e (eV)',
                gridcolor='rgb(255, 255, 255)',
                zerolinecolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            ),
            aspectratio = dict( x=1, y=1, z=0.7 ),
            aspectmode = 'manual'
        )
    )
    #py.plot(data)
    fig = go.Figure(data=data, layout=layout)
    py.plot(fig)


#data = pd.DataFrame.from_dict({
#    'x': psi_N_rng,
#    'y': np.random.randint(low=-1, high=1, size=5),
#    #'z': np.random.randint(low=-2, high=5, size=5),
#})
```

```
#
#p = PolynomialFeatures(degree=2).fit(data)
#print(p.get_feature_names(data.columns))
#
#features = pd.DataFrame(p.transform(data),
#                             columns=p.get_feature_names(data.columns))
#print(features)


########################################################################################
#####     curve-fit ??

#coords = (psi_N_rng, tme)
#
#def func(arr=coords, *p):
#     X1, Y1 = arr
#     result = p[0]*(X1**3) + p[1]*(Y1**3) + p[2]*((X1**2)*Y1)
#     + p[3]*((Y1**2)*X1) + p[4]*(X1**2) + p[5]*(Y1**2) + p[6]*(X1*Y1)
#     + p[7]*X1 + p[8]*Y1 + p[9]*X1*0+1
#     return result.ravel()

#xdim = 101
#ydim = 90
#
#x = np.linspace(0.1,1.1,xdim)
#y = np.linspace(1.,2., ydim)
#X=np.meshgrid(x,y)
#a, b, c = 10., 4., 6.
#z = func(X, a, b, c) * 1 + np.random.random(xdim*ydim) / 100
#
#p0 = 8., 2., 7.
#print(curve_fit(func, X, z, p0))




########################################################################################
#####     2d Gaussian processes - Not smooth enough

#Te_flat = np.array(Te_interp).flatten()
#
#xx = np.vstack((X1.flatten(), Y1.flatten())).T
#
#rand = np.random.choice(range(len(Te_flat)), 2222)
#
#Te_rand = Te_flat[rand]
#xx_rand = xx[rand]
#
#kernel = RBF()
#gp = GaussianProcessRegressor(kernel=kernel,
#                                    n_restarts_optimizer=10)
#gp.fit(xx_rand, Te_rand)
#print("Learned kernel", gp.kernel_)
#
#Te_gpmodel = gp.predict(xx).reshape(-1, len(psi_N_rng))
#
#plt.contourf(psi_N_rng, tme, Te_gpmodel, 33)
#plt.colorbar()
#plt.xlabel('$\Psi_{N}$')
#plt.ylabel('t (s)')
```

```
#plt.title('$T_{e} (eV)$')



#posterior_nums = 3
#posteriors = st.multivariate_normal.rvs(mean=y_mean,
#                                         size=posterior_nums)
#
#fig, axs = plt.subplots(posterior_nums+1)
#
#ax = axs[0]
#ax.contourf(psi_N_rng, tme, Te_interp, 33)
#ax.plot(X[:, 0], X[:, 1], "r.", ms=12)
#
#for i, post in enumerate(posteriors, 1):
#    axs[i].contourf(psi_N_rng, tme, post.reshape(-1, len(psi_N_rng)))
#
#plt.show()



#####################################################################################
#####    plotting

def poly2dfit_plot():
    plt.figure()
    plt.contourf(x, y, tst_z, 33)
    plt.colorbar()

    plt.figure()
    plt.contourf(x, y, tst_z2, 33)
    plt.colorbar()

    plt.figure()
    plt.contourf(x, y, tst_z3, 33)
    plt.colorbar()

    plt.show()

def psi_N_interps(x=chk_t):
    plt.figure()
    plt.plot(psi_fin2[x], Te2[x], label='no smoothing')
    plt.plot(psi_N_rng, Te_interp[x], label='same $\Psi$_1')
    plt.plot(psi_rng2, Te_interp2[x], label='same $\Psi$_2')
    plt.xlabel('$\Psi_{N}$')
    plt.ylabel('$T_{e} (eV)$')
    plt.legend()
    plt.show()

def Te_vs_psiN():
    plt.figure()
    plt.contourf(psi_N_rng, tme, Te_interp, 33)
    plt.colorbar()
    plt.xlabel('$\Psi_{N}$')
    plt.ylabel('t (s)')
    plt.title('$T_{e} (eV)$')
    plt.show()

def Te_vs_psiN_3d():
#    fig = plt.figure()
```

```python
#     ax = fig.add_subplot(111, projection="3d")
#     ax.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
#
#     fig = plt.figure()
#     ax = fig.add_subplot(111, projection="3d")
#     Z2 = tst_z
#     ax.plot_surface(X1, Y1, Z2, cmap="autumn_r", lw=0.5, rstride=1, cstride=1)
#
#     fig = plt.figure()
#     ax = fig.add_subplot(111, projection="3d")
#     Z3 = tst_z2
#     ax.plot_surface(X1, Y1, Z3, cmap="autumn_r", lw=0.5, rstride=1, cstride=1)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.plot_surface(X1, Y1, Z1, cmap="jet", lw=0.5, rstride=1, cstride=1)
    #ax.plot_surface(X1, Y1, Z2, cmap="autumn", lw=0.5, rstride=1, cstride=1)
    ax.plot_surface(X1, Y1, tst_z2, cmap="autumn_r", lw=0.5, rstride=1,
                    cstride=1, alpha=0.7)
    #ax.plot_surface(X1, Y1, Z4, cmap="summer", lw=0.5, rstride=1, cstride=1)
    ax.grid(False)

    plt.show()


def psi_plot(x=chk_x):
    plt.figure()
    plt.plot(tme, psi_dat_z0_new2[:,x], label='$\Psi$ at pos_index {}'
             .format(x))
    plt.plot(tme, mag_axis_interp, label='mag-axis from freia')
    plt.plot(tme, mag_ax_psi, label='mag-axis from code')
    plt.plot(tme, psi_boundary, label='boundary from freia')
    plt.fill_between(tme, mag_ax_psi, psi_boundary, alpha=0.3)
    plt.xlabel('time (s)')
    plt.ylabel('$\Psi$', rotation=0)
    plt.legend()
    plt.show()

def check_same_psi():
    for i in range(len(tme)):
        plt.figure()
        plt.plot(psi_N_rng, Te[i], 'bx-', ms=2)
        plt.plot(psi_N[i], ayc_te_dat[i], 'ro-', ms=2)
        plt.title(i)

def R_ch():
    plt.figure()
    #plt.plot(ayc_r_t, ayc_r_dat[:,20])
    plt.contourf(ayc_r_dat.T, 11)
    plt.colorbar()
    plt.ylabel('channel number')
    plt.xlabel('time (index value)')
    plt.title('Radius (m)')
    plt.show()

def te_channel():
    plt.figure()
    plt.contourf(tme, psi_ch, ayc_te_dat.T, 33)
    plt.colorbar()
    plt.xlabel('time (s)')
```

```
    plt.ylabel('channel number')
    plt.title('$T_{e}$')
    plt.show()

def psi_channel(chk_t=chk_t):
    plt.figure()
    plt.contourf(tme, psi_ch, psi_dat_z0_new2.T, 33)
    plt.colorbar()
    plt.xlabel('time (s)')
    plt.ylabel('channel number')
    plt.title('psi_new at z=0')
    #
    #plt.figure()
    #plt.contourf(psi_t, psi_x, psi_dat_z0.T, 33)
    #plt.colorbar()
    #plt.xlabel('time (s)')
    #plt.ylabel('radial position (m)')
    #plt.title('psi at z=0')

    plt.figure(chk_t)
    plt.plot(psi_ch, psi_dat_z0_new2[chk_t])
    plt.xlabel('channel number')
    plt.ylabel('$\Psi$', rotation=0)
    plt.title('for time index {}'.format(chk_t))
    plt.show()

def te_x(chk_t=chk_t):
#    plt.figure()
#    plt.plot(psi_ch, ayc_te_dat[chk_t])
#    plt.xlabel('channel number')
#    plt.ylabel('$T_{e}$', rotation=0)
#    plt.title('for time index {}'.format(chk_t))

    plt.figure()
    plt.plot(ayc_r_dat[chk_t], ayc_te_dat[chk_t])
    plt.xlabel('radial position (m)')
    plt.ylabel('$T_{e}$', rotation=0)
    plt.title('for time index {}'.format(chk_t))

#    plt.figure()
#    plt.plot(ayc_r_x, ayc_r_dat[chk_t])
#    plt.xlabel('radial index (channel number)')
#    plt.ylabel('radial position (m)')
    plt.show()

def te_psi():
    plt.figure()
    #plt.plot(psi_sort, T_e_sort)
    #plt.plot(psi_fin[chk], Te_fin[chk], 'r')
    plt.plot(psi_dat_z0_new2[chk_t,:], ayc_te_dat[chk_t,:], 'g')
    plt.xlabel('$\Psi$')
    plt.ylabel('$T_{e}$', rotation=0)
    plt.title('for time index {}'.format(chk_t))
    plt.show()

def te_multi_psi(strt=0, stp=len(tme)-1):
    plt.figure()
    plt.xlabel('$\Psi$')
    plt.ylabel('$T_{e}$ (eV)')
    tmp1 = strt
```

```python
    tmp2 = stp
    plt.title('time: {} to {}'.format(round(tme[tmp1], 3),
              round(tme[tmp2], 3)))
    for i in range(tmp1, tmp2):
        plt.plot(psi_fin2[i], Te2[i])
        #plt.plot(psi_N[i], ayc_te_dat[i])
        #plt.plot(psi_peaks[i], Te_peaks[i], 'go')

    plt.figure()
    plt.xlabel('$\Psi$')
    plt.ylabel('$T_{e}$', rotation=0)
    tmp1 = strt
    tmp2 = stp
    plt.title('time: {} to {}'.format(round(tme[tmp1], 3),
              round(tme[tmp2], 3)))
    for i in range(tmp1, tmp2):
        plt.plot(psi_N_rng, Te_interp[i])
        #plt.plot(psi_N[i], ayc_te_dat[i])
        #plt.plot(psi_peaks[i], Te_peaks[i], 'go')
    plt.show()

def psi_rz(cont_type='contour', chk_t=chk_t, res=33):
    plt.figure()
    if cont_type == 'contour':
        plt.contour(efm_grid_r, efm_grid_z, psi_dat[chk_t,:,:], res)
    elif cont_type == 'contourf':
        plt.contourf(efm_grid_r, efm_grid_z, psi_dat[chk_t,:,:], res)
    else:
        plt.contour(efm_grid_r, efm_grid_z, psi_dat[chk_t,:,:], res)
    #plt.axis('equal')
    plt.colorbar()
    plt.ylabel('z (m)')
    plt.xlabel('radial position (m)')
    plt.title('$\Psi (r,z)$ at t = {}'.format(chk_t))
    plt.show()

def psi_interp_multi():
    plt.figure()
    plt.contourf(psi_dat_z0_new2, 33)
    plt.colorbar()
    plt.xlabel('channel number')
    plt.ylabel('time index')
    plt.title('$\Psi (z=0)$ interpolated (2D interp)')

    plt.figure()
    plt.contourf(psi_dat_z0, 33)
    plt.colorbar()
    plt.xlabel('channel number')
    plt.ylabel('time index')
    plt.title('$\Psi (z=0)$')

    plt.figure()
    plt.contourf(psi_dat_z0_new, 33)
    plt.colorbar()
    plt.xlabel('channel number')
    plt.ylabel('time index')
    plt.title('$\Psi (z=0)$ interpolated (2 x 1D interp)')
    plt.show()

def psi_interp_test(chk_x=chk_x, chk_t=chk_t):
```

```
    plt.figure()
    plt.plot(psi_t, psi_dat_z0[:,chk_x], 'bx', ms=mrk)
    plt.plot(ayc_r_t, test, 'ro', ms=mrk)
    plt.xlabel('time (s)')
    plt.ylabel('$\Psi$', rotation=0)
    plt.title('for pos index {}'.format(chk_x))
#
    plt.figure()
    plt.plot(psi_x, psi_dat_z0[chk_t], 'bx', ms=mrk)
    plt.plot(ayc_r_dat[chk_t], test2, 'ro', ms=mrk)
    plt.xlabel('radial position (m)')
    plt.ylabel('$\Psi$', rotation=0)
    plt.title('for time index {}'.format(chk_t))
    plt.show()

Te_vs_psiN()
#psi_N_interps(44)
#psi_plot()
#te_multi_psi(40,45)
#poly2dfit_plot()
#Te_vs_psiN_3d()
#psi_rz()

################################################################################
#####    Fancy animations

#for i in range(0, psi_rz_28819['time'].shape[0]):
#    fig = plt.figure()
#    plt.contour(efm_grid_r.squeeze(), efm_grid_z.squeeze(),
#                 psi_dat[i,:,:], 50)
#    plt.colorbar()
#    plt.ylabel('z (m)')
#    plt.xlabel('radial position (m)')
#    plt.title('$\Psi (r,z)$')
#    print(i)
#    plt.savefig(str(i).zfill(4) +'.png')
#    plt.close(fig)

#for i in range(0, len(tme)):
#    fig = plt.figure()
#    plt.plot(psi_dat_z0_new2[i], ayc_te_dat[i])
#    plt.xlabel('$\Psi$')
#    plt.ylabel('$T_{e}$', rotation=0)
#    plt.title('$T_{e}$ vs $\Psi$')
#    print(i)
#    plt.savefig(str(i).zfill(4) +'.png')
#    plt.close(fig)

#os.chdir('./pics')
#for i in range(0, len(tme)):
#    fig = plt.figure()
#    te_multi_psi(i, i+1)
#    print(i)
#    plt.savefig(str(i).zfill(4) +'.png')
#    plt.close(fig)
#os.chdir('../')
#plt.show()
```