```
#-----------------------------------------------------------------
#
# Python library for evaluation of principal components of unpolar
# using both procedures of Sambridge et al. (2008) and Jackson et
#
# M. Sambridge and A. Jackson, 2018.
#
#-----------------------------------------------------------------
# Import required python libraries
import numpy as np
import mpmath as mp
import matplotlib.pyplot as plt
import pandas as pd
import time
from scipy.optimize import minimize
#
# Absorbance library routines
#
#################################################################
def pdf(Q,a1,a2,a3): # Probability density function, PDF, eqns (3

    if(not a1>=a2>a3): return print('Function pdf: Input error in
    if(Q<=a3):
        return 0.0
    elif(Q>=a1):
        return 0.0
    elif(Q>a2):
        return 1.0/np.pi/np.sqrt((a1-a2)*(Q-a3))*float(mp.ellipk((
    else:
        return 1.0/np.pi/np.sqrt((a2-a3)*(a1-Q))*float(mp.ellipk((
#################################################################
def G(c,b,p): # G function, eqns (28)-(29) of Jackson et al. (201

    return 2*(c-b)*np.sqrt(1.0/(p+1.0)/(b+c))*float(mp.ellippi(2.0
#################################################################
def cdf(Q,a1,a2,a3): # Cummulative density function, CDF, eqns (2

    if(not a1>=a2>=a3): return print('Function cdf: Input error ir
    if(Q<=a3):
        return 0.0
    elif(Q>=a1):
        return 1.0
    elif(Q==a2):
        return 1.0/np.pi*2.0*np.arctan((a2-a3)/(a1-a2))
    elif(Q>a2):
        return 1.0/np.pi/np.sqrt(2*a1-a2-a3)*G(2*Q-a2-a3,a3-a2,(a2
```

```
    else:
        return 1.0 - 1.0/np.pi/np.sqrt(a1+a2-2*a3)*G(a1+a2-2*Q,a2-
##########################################################################
def pdfsmooth(Q,a1,a2,a3,delta): # evaluate smoothed PDF for a si

    return (cdf(Q+delta,a1,a2,a3)-cdf(Q-delta,a1,a2,a3))/(2.0*delt

##########################################################################
def opt_func(x,data,delta): # evaluate -ve log(likelihood)

    if((x[0] - x[1] <= x[2]) & (x[2] <= x[1]) & (x[1] <= x[0])):
        out = np.zeros(len(data))
        for i in range(len(data)):
            out[i] = pdfsmooth(data[i],x[0],x[1],x[2],delta)
            if(out[i]==0.0):
                #print (' i = ',i)
                return np.finfo(np.float32).max # return barrier f
        return -np.sum(np.log(out))
    else:
        #print (' point infeasible')
        return np.finfo(np.float32).max # return barrier function
##########################################################################
def checkPDF(x,data,delta): # check if PDF is zero because of data

    if((x[0] - x[1] <= x[2]) & (x[2] <= x[1]) & (x[1] <= x[0])):
        for i in range(len(data)):
            if(pdfsmooth(data[i],x[0],x[1],x[2],delta)==0.0):
                return True # return barrier function
        return False
    else:
        return True
##########################################################################
def abs_ParamSearchSize(x,y,z): # Calculate number of feasible com

    xv,yv,zv = np.meshgrid(x,y,z)
    bl = (xv - yv <= z) & (zv <= yv) & (yv <= xv) # locations of g
    return len(xv[bl])
##########################################################################
def abs_ParamSearch(x,y,z,data,delta): # perform grid search to ma

    xv,yv,zv = np.meshgrid(x,y,z)
    bl = (xv - yv <= z) & (zv <= yv) & (yv <= xv) # locations of g
    #a1b,a2b,a3b,best = xv[bl][0],yv[bl][0],zv[bl][0],np.sum(np.lo
    a1b,a2b,a3b,best = 0.0,0.0,0.,-np.finfo(np.float32).max
    for i in range(len(xv[bl])):
        loglike = -opt_func([xv[bl][i],yv[bl][i],zv[bl][i]],data,d
```

```
        #print(' i ',i,'like ',loglike,' a1 ',xv[bl][i],' a2 ',yv
        if(loglike>best):
            best, a1b, a2b, a3b = loglike,xv[bl][i],yv[bl][i],zv[b
            print(' i ',i,'loglike ',loglike,' a1 ',xv[bl][i],' a2
    return best,a2b+a3b-a1b,a1b+a3b-a2b,a1b+a2b-a3b,a1b+a2b+a3b,a1
##########################################################################
def gridsearchRect(data,delta,n): # Perform grid search for princ

    # set up Cartesian search grid for parameters (a1,a2,a3)
    a2_start = 0.9*np.min(data)  # set limits of a2
    a2_stop = 1.1*np.max(data)   # set limits of a2
    a3_start = 0.0               # set lower limit of a3 using eq
    a3_stop = a2_stop            # set upper limit of a3 using eq
    a1_start = a2_start          # set lower limit of a1 using eq
    a1_stop = 2.0*a2_stop        # set upper limit of a1 using eq
    x = np.linspace(a1_start,a1_stop,n[0]) # a1 discretized value
    y = np.linspace(a2_start,a2_stop,n[1]) # a2 discretized value
    z = np.linspace(a3_start,a3_stop,n[2]) # a3 discretized value

    print('\nPerforming grid search on grid ',n[0],'x',n[1],'x',n[
    print('Discretization intervals delta a1 =',(a1_stop-a1_start)
                                    'delta a2 =',(a2_stop-a2_start)
                                    'delta a3 =',(a3_stop-a3_start)
    print('with',abs_ParamSearchSize(x,y,z),'feasible points...\n
    start_time = time.time()
    sol = abs_ParamSearch(x,y,z,data,delta)
    print("--- %s seconds compute time ---" % (time.time() - start
    return sol
##########################################################################
def gridsearch(data,delta,n): # Perform grid search for principal

    y = np.linspace(0.9*np.min(data),1.1*np.max(data),n[1]+1) # a2
    a1b,a2b,a3b,best = 0.0,0.0,0.,-np.finfo(np.float32).max

    print('\nPerforming grid search on grid ',n[0],'x',n[1],'x',n[
    print('Discretization of a2 axis: a2 start ',0.9*np.min(data),
                                    ' a2_stop ',1.1*np.max(data),
                                    ' delta a2',y[1]-y[0])
    start_time = time.time()
    for j in range(n[1]+1):
        z = np.linspace(0.0,y[j],n[2]+1)
        for k in range(n[2]+1):
            x = np.linspace(y[j],y[j]+z[k],n[0]+1)
            for i in range(n[0]+1):
                loglike = -opt_func([x[i],y[j],z[k]],data,delta)
                #print(i,j,k,loglike,' a1 ',x[i],' a2 ',y[j],' a3
```

```
                if(loglike>best):
                    best, a1b, a2b, a3b = loglike,x[i],y[j],z[k]
                    print(i,j,k,'  New best loglike ',loglike,' a
    print("--- %s seconds compute time ---" % (time.time() - start
    return best,a2b+a3b-a1b,a1b+a3b-a2b,a1b+a2b-a3b,a1b+a2b+a3b,a
##########################################################################
def randomfeasible(data,delta): # Randomly generate feasible princ

    x = np.random.rand(3)
    d1,d2 = 0.9*np.min(data),1.1*np.max(data)
    a2 =((1.-x[0])*(d1**3)+x[0]*(d2**3))**(1/3.)
    a3 = a2*np.sqrt(x[1])
    a1 = (a2+a3)*x[2] + (1.-x[2])*a2

    #d3 = np.min(data)-delta
    #d4 = np.max(data)+delta
    #a3 = np.min([a2,d3])*np.sqrt(x[1])
    #a1 = (a2+a3)*x[2] + (1.-x[2])*np.max([a2,d4])

    return [a1,a2,a3]
##########################################################################
def randomfeasibleloop(data,delta): # Randomly generate set of pr
                                    # feasible w.r.t model and da

    a = randomfeasible(data,delta)
    count = 0
    while((checkPDF(a,data,delta)) and (count < 1000)):
        a = randomfeasible(data,delta)
        count += 1
    return a
##########################################################################
def checkfeasible(a): # check model feasibility of principal abso

    return (a[0] - a[1] <= a[2]) & (a[2] <= a[1]) & (a[1] <= a[0])
##########################################################################
def initialsimplex(data,delta): # Randomly generate feasible init

    np.random.seed(61254557)
    out = np.zeros([4,3])
    for i in range(4):
        out[i,:]=randomfeasibleloop(data,delta)
    return out
##########################################################################
def localsearch(data,delta): # Perform local optimisation for pri

    mthd = 'Powell'
```

```
    mthd = 'Nelder-Mead'
    if(mthd=='Nelder-Mead'):
        print('\nOptimisation by Nelder-Mead simplex algorithm')
        np.random.seed(61254557)
        #xsimp = initialsimplex(data,delta) # initialize starting
        #print(' initial simplex:',xsimp)
        x0 = [np.max(data),3.0*np.mean(data)-np.min(data)-np.max(d
        x0 = randomfeasibleloop(data,delta) # initialize starting
        start_time = time.time()
        res = minimize(opt_func, x0, args=(data,delta),method='Nel
                    options={'xtol': 0.01, 'disp': True})
        # options={'xtol': 0.01, 'disp': True, 'initial_simplex':
        end_time = time.time()
    elif(mthd=='Powell'):
        print('\nOptimisation by Powell algorithm')
        #x0 = [np.max(data),3.0*np.mean(data)-np.min(data)-np.max(
        np.random.seed(61254557)
        x0 = randomfeasibleloop(data,delta) # initialize starting
        #x0 = randomfeasible(data) # initialize starting point to
        #print(x0)
        start_time = time.time()
        res = minimize(opt_func, x0, args=(data,delta),method='Pow
                    options={'xtol': 0.01, 'disp': True})
        end_time = time.time()
    print("--- %s seconds compute time ---" % (end_time - start_ti
    out = [res.fun,res.x[1]+res.x[2]-res.x[0],res.x[0]+res.x[2]-re
          res.x[0]+res.x[1]-res.x[2],res.x[0]+res.x[1]+res.x[2],re
    return out
##########################################################################
def plotCond(data,delta,sol): # plot conditional PDFs for principa

    plt.rcParams["figure.figsize"]=(8.0,3.0)
    # These plot limits can be adjusted for increased resolution
    a2_start = 0.9*np.min(data)  # set limits of a2 for plot
    a2_stop = 1.1*np.max(data)   # set limits of a2 for plot
    a3_start = 0.0               # set lower limit of a3 using equ
    a3_stop = a2_stop            # set upper limit of a3 using equ
    a1_start = a2_start          # set lower limit of a1 using equ
    a1_stop = 2.0*a2_stop        # set upper limit of a1 using equ
    a1 = np.linspace(a1_start,a1_stop,100) # a1 discretized values
    a2 = np.linspace(a2_start,a2_stop,100) # a2 discretized values
    a3 = np.linspace(a3_start,a3_stop,100) # a3 discretized values

    fig, (ax1,ax2,ax3) = plt.subplots(1,3)
    data1 = list(map(lambda x: np.exp(-opt_func([x,sol[1],sol[2]],
    ax1.plot(a1,data1) # plot similar to 9a of Jackson et al. (20
```

```
    ax1.set_title('Conditional PDF for $a_1$')
    ax1.set_xlabel('$a_1$')
    ax1.set_ylabel('P($a_1|Q,a_2,a_3$)')
    #
    data1 = list(map(lambda x: np.exp(-opt_func([sol[0],x,sol[2]],
    ax2.plot(a2,data1) # plot similar to 9b of Jackson et al. (20
    ax2.set_title('Conditional PDF for $a_2$')
    ax2.set_xlabel('$a_2$')
    ax2.set_ylabel('P($a_2|Q,a_1,a_3$)')
    #
    data1 = list(map(lambda x: np.exp(-opt_func([sol[0],sol[1],x],
    ax3.plot(a3,data1) # plot similar to 9c of Jackson et al. (20
    ax3.set_title('Conditional PDF for $a_3$')
    ax3.set_xlabel('$a_3$')
    ax3.set_ylabel('P($a_3|Q,a_1,a_2$)')
    plt.tight_layout()
    plt.show()
    plt.savefig("Figure2.pdf")
    #
    return
##########################################################################
def Atoa(A): # utility routine to convert big A absorbance to lit
    return 0.5*(A[2]+A[1]), 0.5*(A[2]+A[0]), 0.5*(A[0]+A[1]), np.s
##########################################################################
def atoA(a): # utility routine to convert big A absorbance to lit
    return a[1]+a[2]-a[0], a[0]+a[2]-a[1], a[1]+a[0]-a[2], np.sum(
##########################################################################
#
# Main body of script. (Only runs if this script is executed as a
#
if __name__ == "__main__":
    plt.rcParams["figure.figsize"]=(8.0,3.0)

    # Parameters for user to change
    datatype = 'Excel'                              # choose source o
    optmeth = 'Grid'                                # Type of optimis
    optmeth = 'Simplex'                             # Type of optimis
                                                    # NB: Simplex is
    Ndsets  =  [10,20,60,300,450,600,750,1000]      # List of data se
    Ndsets  =  [10,20]                              # List of data se
    delta   = 1.0                                   # data error size
    na1,na2,na3 = 5,5,5                             # Set discretizat
                                                    # These values sh
    PlotPQ = True                                   # Plot P(Q) figur
    PlotConditionals = True                         # Plot conditiona
                                                    # similar to Figu
```

```
#
# calculate P(Q) using formulae for three cases of observatio
#
# Here is an example of the pdf for a=(3.0,2.5,1.0) and delta=
if(PlotPQ):  # set to True to see plot similar to Figure 5.2 
    plt.subplot(1, 3, 1)
    data1 = list(map(lambda x: pdfsmooth(x,3.0,2.5,1.0,0.01),
    plt.plot(np.arange(1-0.01,3+0.01,0.01),data1) # plot 5.1 
    plt.title('$\Delta=0.01$')
    plt.xlabel('Q')
    plt.ylabel('P(Q)')
    #
    data2 = list(map(lambda x: pdfsmooth(x,3.0,2.5,1.0,0.05),
    plt.subplot(1, 3, 2)
    plt.plot(np.arange(1-0.01,3+0.01,0.01),data2) # plot 5.2 
    plt.title('$\Delta=0.05$')
    plt.xlabel('Q')
    plt.ylabel('P(Q)')
    #
    data3 = list(map(lambda x: pdfsmooth(x,3.0,2.5,1.0,1.0), r
    plt.subplot(1, 3, 3)
    plt.plot(np.arange(0.01,4+0.01,0.02),data3) # plot 5.3 of
    plt.title('$\Delta=1.0$')
    plt.xlabel('Q')
    plt.ylabel('P(Q)')

    plt.tight_layout()
    plt.show()
    plt.savefig("Figure1.pdf")
#
# Set up test data corresponding to that used in Jackson et a
if(datatype=='Testset'): # small demonstration dataset
    data_all = np.array([44.15967, 49.92802, 43.72898, 34.9610
    39.17191, 39.30958, 41.48838, 60.79360, 27.77383, 62.4397
    25.06300, 33.80495, 42.13757, 36.95285, 29.79391, 26.6633
    Ndsets = [len(data_all)]
    Atruth = [1.89, 45.6, 80.94]
    atruth = Atoa(Atruth)
elif(datatype=='Excel'):
    # read in absorbance data from excel file here (Default is
    exceldata = pd.read_excel('Absorbance_testdata.xlsx',sheet
    data_all = exceldata.loc[:, 'Delta=1.0'] # read excel colu

# Find principal absorbances which maximize the likelihood de

for nv in Ndsets:                                          #
```

```
    data0=np.array(data_all[:nv])                                    #
    data = data0[np.argsort(-np.abs(data0-np.mean(data0)))] #
    print('\nTest data: Number of absorbance values = ',len(da
    print('\nResults using Sambridge et al. (2008)')
    print('a1 = ',np.max(data),'a2 = ',3.0*np.mean(data)-np.mi
    print('A_a = ',3.0*np.mean(data)-2.0*np.max(data),'A_b =

    if(optmeth=='Simplex'): # optimsation using Nealder-Mead s
        sol = localsearch(data,delta)  # minimize -ve log like
        print('\nResults of local optimisation for maximum Lik
    elif(optmeth=='Grid'): # optimsation using grid search ove
        sol = gridsearch(data,delta,[na1,na2,na3])   # maximiz
        print('\nResults of grid search for maximum Likelihood
    print('A_a =',sol[1],' A_b =',sol[2],' A_c =',sol[3],' A_t

# Plot conditional PDF of each of (a1,a2,a3) through maximum .
# Similar plot to Figure 9 of Jackson et al. (2018) from whic

if(PlotConditionals):
    print(' Building conditional plot of Likelihood through so
    plotCond(data,delta,[sol[5],sol[6],sol[7]]) # plot conditi
```