```
# -*- coding: utf-8 -*-
"""
Created on Fri May 29 21:33:16 2020

@author: zhongwei
"""
import numpy as np
from pylab import *


################Function##############

# caculating δR
def Get_dR(R):
    R_loop = int((R - d)/ndR)
    d_R = dR_time + R_loop*ndR

    return d_R

# caculationg angele of arrive (AOA) for a piont in  X1Y1Z1
def AOA(Rt):

    #θ
    cos_theta = Rt.T*n_z/len_Rt
    sin_theta = np.sqrt(1-cos_theta**2)

    #ϕ
    if sin_theta == 0:
        cos_phi = np.array([[1/len_Rt]])
        sin_phi = np.array([[1/len_Rt]])
    else:
        # exception for 0 zenith
        cos_phi = Rt.T*n_x/len_Rt/sin_theta
        sin_phi = Rt.T*n_y/len_Rt/sin_theta

    return [cos_theta[0,0],sin_theta[0,0],cos_phi[0,0],sin_phi[0,0]]


# input 3 rotation angle in degree
# output rotation matrix
def CTM_Rotate(phi_x,phi_y,phi_z):
    x = phi_x*np.pi/180
    y = phi_y*np.pi/180
    z = phi_z*np.pi/180


    a_11 = np.cos(y)*np.cos(z)
    a_12 = np.sin(x)*np.sin(y)*np.cos(z) - np.cos(x)*np.sin(z)
    a_13 = np.cos(x)*np.sin(y)*np.cos(z) + np.sin(x)*np.sin(z)

    a_21 = np.cos(y)*np.sin(z)
    a_22 = np.sin(x)*np.sin(y)*np.sin(z) + np.cos(x)*np.cos(z)
    a_23 = np.cos(x)*np.sin(y)*np.sin(z) - np.sin(x)*np.cos(z)
```

```
    a_31 = -np.sin(y)
    a_32 = np.sin(x)*np.cos(y)
    a_33 = np.cos(x)*np.cos(y)

    A = np.mat( [[a_11,a_12,a_13]
                ,[a_21,a_22,a_23]
                ,[a_31,a_32,a_33]])

    return A

##error propagation function
# input sin/cos value of AOAs

# output [δθ,/,/]
def f_1(cos_theta,sin_theta,cos_phi,sin_phi):

    w1 = cos_phi/cos_theta/(D_1/length)/(2*np.pi)
    w2 = sin_phi/cos_theta/(D_2/length)/(2*np.pi)

    f = np.sqrt(w1**2*d_D_phi_1**2 + w2**2*d_D_phi_2**2)

    return [f,w1,w2]

# output [δϕ,/,/]
def f_2(cos_theta,sin_theta,cos_phi,sin_phi):

    if sin_theta == 0:
        w1 = -1/(D_1/length)/(2*np.pi)
        w2 = 1/(D_2/length)/(2*np.pi)
    else:
        w1 = -sin_phi/sin_theta/(D_1/length)/(2*np.pi)
        w2 = cos_phi/sin_theta/(D_2/length)/(2*np.pi)

    f = np.sqrt((w1*d_D_phi_1)**2 + (w2*d_D_phi_2)**2)

    return [f,w1,w2]

# output [fR, f_θ,f_ϕ]
def F_Rt(cos_theta,sin_theta,cos_phi,sin_phi):

    P = A_11*sin_theta*cos_phi + A_12*sin_theta*sin_phi + A_13*cos_theta
    dP_theta = A_11*cos_theta*cos_phi + A_12*cos_theta*sin_phi - A_13*sin_theta
    dP_phi = -A_11*sin_theta*sin_phi + A_12*sin_theta*cos_phi

    N = (R - d*P)**2

    M_1 = d**2 + R**2 -2*R*d*P
    f_R = M_1/N/2

    M_2 = d*(R**2-d**2)*dP_theta
    f_theta = M_2/N/2

    M_3 = d*(R**2-d**2)*dP_phi
    f_phi = M_3/N/2
```

```
    return [f_R,f_theta,f_phi]


def mat_W(w_dR):
      f_R = w_dR[0]
      f_theta = w_dR[1]
      f_phi = w_dR[2]
      Rt_sin = len_Rt*sin_theta

      weight_Rt = np.mat([[f_R,f_theta,f_phi],[0,len_Rt,0],[0,0,Rt_sin]])


      Aoa_Rt = np.mat([[sin_theta*cos_phi,-cos_theta*cos_phi,-sin_phi],
                       [sin_theta*sin_phi,-cos_theta*sin_phi,cos_phi],
                       [cos_theta,sin_theta,0]])

      Ot_matrix = np.mat([[1,0,0],
                          [0,w_d_theta[1],w_d_theta[2]],
                          [0,w_d_phi[1],w_d_phi[2]]])

      W = A0*Aoa_Rt*weight_Rt* Ot_matrix

      return W


########## 3D spatial distribution caculation###############
###################### main ########################

###### parameters settings #####

#wave length
length = 1.0

# baseline length / wave length
D_1 = 4.5
D_2 = 4.5

#premitted phase measurment errors (in degree)
d_D_phi_1 = 35/180*np.pi
d_D_phi_2 = 35/180*np.pi

# space grid length
dx = 5.0
dy = 5.0
dz = 1.0

# space range to caculate
x0_length = 1000
y0_length = 1000
z0_low = 60
z0_hig = 120

# point in x'0 y'0 z'0
```

```
x0 = np.arange(-x0_length/2,x0_length/2+dx,dx)
y0 = np.arange(-y0_length/2,y0_length/2+dy,dy)
z0 = np.arange(z0_low,z0_hig+dz,dz)
x0_inverse = x0[::-1]

# 6 rotation angle
psi_x_1,psi_y_1,psi_z_1 = 0, 0, 0
px_10,py_10,pz_10 = 0, 0, 30

# baseline length
d = 150

#pulse lengthc*Delta_Tp
ct = 4
S = ct/2
#PRF
f = 625
ndR = 300000/f

#GPS error(δR)
dR_time = 6.3

#A_total
A_x_x1 = CTM_Rotate(psi_x_1,psi_y_1,psi_z_1)
A_11 = A_x_x1[0,0]
A_12 = A_x_x1[0,1]
A_13 = A_x_x1[0,2]
#point in xyz reprsent in x1y1z1
n_x = np.mat([[A_11],[A_x_x1[1,0]],[A_x_x1[2,0]]])
n_y = np.mat([[A_12],[A_x_x1[1,1]],[A_x_x1[2,1]]])
n_z = np.mat([[A_13],[A_x_x1[1,2]],[A_x_x1[2,2]]])

#A'_total
A_x1_x0 = CTM_Rotate(px_10,py_10,pz_10)
A_x0_x1 = A_x1_x0.I

#A'*A
A0 = A_x1_x0*A_x_x1


##### save caculation results
d2_e1_x0 = []
d2_e1_y0 = []
d2_e1_z0 = []

d2_e2_x0 = []
d2_e2_y0 = []
d2_e2_z0 = []


#loop to ergodic every point in 3D space
for k in z0:
    tempY_x_1 = []
    tempY_y_1 = []
```

```
tempY_z_1 = []

tempY_x_2 = []
tempY_y_2 = []
tempY_z_2 = []

for j in y0:

    tempX_x_1 = []
    tempX_y_1 = []
    tempX_z_1 = []

    tempX_x_2 = []
    tempX_y_2 = []
    tempX_z_2 = []

    for i in x0:

            ############### error 1###############

            # x0y0z0 - x1y1z1
            R0 = np.mat([[i],[j],[k]])
            Rt = A_x0_x1*R0
            len_Rt = np.sqrt(Rt.T*Rt)[0,0]

            #sin/cos(theta) sin/cos(phi)
            aoa = AOA(Rt)
            cos_theta,sin_theta,cos_phi,sin_phi = aoa[0],aoa[1],aoa[2],aoa[3]

            w_d_theta = f_1(cos_theta,sin_theta,cos_phi,sin_phi)
            w_d_phi = f_2(cos_theta,sin_theta,cos_phi,sin_phi)

            ##F_Rt
            #R
            R1 = Rt - np.mat([[d],[0],[0]])
            len_R1 = np.sqrt(R1.T*R1)[0,0]
            R = len_R1 + len_Rt
            #d_R
            d_R = Get_dR(R)
            #error indepdent
            v_erro = np.mat([[d_R],[d_D_phi_1],[d_D_phi_2]])
            Square_v_erro = np.multiply(v_erro,v_erro)
            ###error weighting
            w_dR = F_Rt(cos_theta,sin_theta,cos_phi,sin_phi)
            W_erro = mat_W(w_dR)
            Square_W_erro = np.multiply(W_erro,W_erro)

            #MSE
            Square_M_erro =  Square_W_erro*Square_v_erro

            #error 1 evaluation
            tempX_x_1.append(Square_M_erro[0,0])
            tempX_y_1.append(Square_M_erro[1,0])
            tempX_z_1.append(Square_M_erro[2,0])
```

```
                ############### error 2 ###################

                ##X'Y'Z'
                #gemetry
                a1 = (len_Rt - S)/len_Rt
                a2 = (len_R1 - S)/len_R1
                W_Rt = (2-a1-a2)/2
                b_Rt = np.mat([[d*(a1-1)/2],[0],[0]])
                ##error vector
                DA_erro = W_Rt*Rt + b_Rt

                #transform to X0Y0Z0 true error
                RealErro_DA = A_x1_x0*DA_erro
                #MSE
                SquareErro_DA =  np.multiply(RealErro_DA,RealErro_DA)

                #error 2 evaluation
                tempX_x_2.append(SquareErro_DA[0,0])
                tempX_y_2.append(SquareErro_DA[1,0])
                tempX_z_2.append(SquareErro_DA[2,0])


            #a hight Z evaluation
            ##error 1
            tempY_x_1.append(tempX_x_1)
            tempY_y_1.append(tempX_y_1)
            tempY_z_1.append(tempX_z_1)
            ##error 2
            tempY_x_2.append(tempX_x_2)
            tempY_y_2.append(tempX_y_2)
            tempY_z_2.append(tempX_z_2)

        # evaluation [Z, Y, X]
        ##error 1
        d2_e1_x0.append(tempY_x_1)
        d2_e1_y0.append(tempY_y_1)
        d2_e1_z0.append(tempY_z_1)
        ##error 2
        d2_e2_x0.append(tempY_x_2)
        d2_e2_y0.append(tempY_y_2)
        d2_e2_z0.append(tempY_z_2)

#list to array

## 9 output
## error 1
X0_SquareE1 = np.array(d2_e1_x0)
Y0_SquareE1 = np.array(d2_e1_y0)
Z0_SquareE1 = np.array(d2_e1_z0)
## error 2
X0_SquareE2 = np.array(d2_e2_x0)
Y0_SquareE2 = np.array(d2_e2_y0)
```

```
Z0_SquareE2 = np.array(d2_e2_z0)
## total error
X0_SquareE_total = X0_SquareE1 + X0_SquareE2
Y0_SquareE_total = Y0_SquareE1 + Y0_SquareE2
Z0_SquareE_total = Z0_SquareE1 + Z0_SquareE2
```