```
from numpy import *
import math
from Conjugate_Gradient import position as pos
from project_update import begin, getEarthRadius, getSpeedOfLight, \
     getGPS1Data   , walkPerson1, turnPerson1,  \
     getGPS2Data   , walkPerson2, turnPerson2,  \
     getGPSBothData, walkPersons,             \
     noiseOn, noiseOff, end

def getUsTogether():

    noiseOn()
    begin('physics')


    #guess to start conjudgate gradient method, North Pole location.
    start = array([0,0,1.,.001])

    #initial gps call
    gps1data,gps2data = getGPSBothData()

    #initial position
    p1i = pos(gps1data,start)
    p2i = pos(gps2data,start)

    #store this position for reference
    p1l = p1i
    p2l = p2i

    #define line between initial positons to reference when orienting person
    m = (p2l[1]-p1l[1])/(p2l[0]-p1l[0])
    b = p1l[1]-m*p1l[0]



    # Using a line that defines thier separation, which is defined every 3 calls
          of the satellite,
    # we compare our current position mapped onto the line with a previous
          mapping onto the line.
    # We use this projection because a person could be turning appropriately, but
          the the increased distance between
    # then will distrupt turning rules

    def xline(pos):
        return (-(-pos[0]-m*pos[1])-m*b)/(m**2+1)

    def yline(pos):
        return (m*(pos[0]+m*pos[1])+b)/(m**2+1)



    def turn(posi,posf,pl):
        vPf=[posf[0] - posi[0],posf[1]-posi[1]]     # Person's Displacement
               Vector
        vPl=[pl[0] - posi[0],pl[1]-posi[1]]         # Vector from person's
               initial location to other person's
```

```
                                                    # defined position point
                                                       (updated every 3 calls)
        dot=vPf[0]*vPl[0]+vPf[1]*vPl[1]             # use cosine-dot product
               relation for estimate of turn angle
        lenf=math.sqrt(vPf[0]*vPf[0]+vPf[1]*vPf[1])
        lenl=math.sqrt(vPl[0]*vPl[0]+vPl[1]*vPl[1])
        return abs(arccos(dot/(lenf*lenl)))         # Using A.B = ||A|| ||B||
               cos(theta) find approx turn. angle



    #intialization of turn sign
    turnSign1 =1
    turnSign2 =-1

    #Conditions to insure correct loops will be entered in beginning

    #At a certain distance, we want one person to remain stationary.
    #When they near each other, it becomes more difficult to orient one's self.
    stay = False
    #We need stay==True to certain trigger loops, but only trigger once for some,
          so we keep a count with stayTime
    stayTime=0
    distance = math.sqrt((p2l[0]-p1l[0])**2+(p2l[1]-p1l[1])**2) # Stored to be
          printed at end
    dis = distance
    count = 0
    #static correction, which will be checked for use.
    angCorr1=0.2
    extraAngCorr1=0.0135
    angCorr2=0.2
    extraAngCorr2=0.0135


    while dis > 5: # keep searching until they find each other.


        if count > 0: # start the search after an initial move is made to
               establish orientation
            count+=1 # Satellite call count

             ##############################################################
             #                         Person II                          #
             ##############################################################

            if dis2f < dis2Prev: # If person 2 gets closer to a point established
                    by person 1.
                # Turn calculation with a static correction.
                # Correct person II's direction by 0.2 radian to compensate for
                         tendency to stray off straight path.
                # Turn sign defines right or left turn
                turnPerson2(turnSign2*(turn(p2is,p2f,p1l)+angCorr2))

                if dis > 1000: # Do this if ||Distance Vector|| is more then 1000
                         meters
                    #Increase step length as confidence in orientation is gained.
```

```
            #Must account for need to correct the turn angle with
                     additional steps.
            turnPerson2(turnSign2*(count*extraAngCorr2))
            step2=100.0+5*count # Sets steps Person II will be walking

        elif dis > 300: # Condition for if 1000 > ||Distance Vector|| >
                  300

            step2=100.0   # reduce steps Person II will be walking, dont
                     want to pass each other



        elif dis > 150:  # Condition if 300 > ||Distance Vector|| > 150
            step2=75.0   # redeuce stepsPerson II will be walking dont
                     want to pass each other

            #make other person start walking again if they begin to
                     seperate
            stay = False
            stayTime=0

        elif dis > 50: # Condition if 150 > ||Distance Vector|| > 50

            step2=50.0 # Sets length Person II will be walking

        else:
            step2=dis  # Person II will determine steps from the distance
                     that seperates them



    else:  # If person 2 gets closer to a point established by person 1.



        #turn opposite direction, this is how orientation is established
                  and corrected
        turnSign2 *=-1

        #Correct person II's direction, 0.2 to compinsate for randomness
        turnPerson2(turnSign2*(turn(p2is,p2f,p1l)+angCorr2))

        if dis > 150:    # Condition for if ||Distance Vector|| > 150

            step2=100.0  # Sets how far Person II will walk

        elif dis > 50:   # Condition for if 100 > ||Distance Vector|| >
                  50

            step2=50.0#25.0   # Sets how far Person II will walk

        else:            # If ||Distance Vector|| < 50

            step2=dis    # Sets how far Person II will walk
```

```
        ##############################################################
        #                         Person I                           #
        ##############################################################


        #repeat same process for person I, with differences noted
        if dis1f < dis1Prev:

            turnPerson1(turnSign1*(turn(p1is,p1f,p2l)+angCorr2))
            if dis > 1000:
                turnPerson1(turnSign1*(count*extraAngCorr1))
                step1=100.0+5*count


            elif dis > 300:

                step1=100.0


            elif dis > 150 and stay==False:
                step1=75.0



            else:
                step1=0.0    # Keep person I stationary
                stay = True



        else:

            turnSign1 *=-1
            turnPerson1(turnSign1*(turn(p1is,p1f,p2l)+angCorr1))

            if dis > 150 and stay == False:

                step1=100.0


            else:
                step1=0.0                  # Keep person I stationary



    else:           # Moves to be executed on first iteration as we need to
           get our bearings
        step1=50.0  # 50+50 steps to make sure we have moved far enough to
                get an accurate idea of change
        step2=50.0  # in distance from other person. Must be far enough that
                uncertainty is negligible.

        # here we verify that we need the static angle correciton.
        # a total of 100 steps are taken, with a mid point satellite call
```

```
                recorded.
        walkPersons(step1,step2)

        gps1data,gps2data = getGPSBothData()
        p1mid = pos(gps1data,start)
        p2mid = pos(gps2data,start)

        count=1     # Only need this loop once, so we never use it again.

    calls = 3       # Number of satellite calls before correcting distance
           vector.
                    # Updating too often cause disorientation




    if dis > 50: # Walk People as long as ||Distance Vector|| > 50 meters
        walkPersons(step1,step2)

    else:        # Because Dr. Ringland's code only checks our distance once
           we stop walking
                 # we take many small calls here to ensure we do not get
                            close without noticing.
                 # steps = 5/6 dis, because the average stride is less than a
                            meter, and
                 # we only want to get within 10 meters
        walkPersons(step1,step2/2)
        walkPersons(step1,step2/6)
        walkPersons(step1,step2/6)



    p1is = p1i # Hold location prior to the previous satellite call person I
    p2is = p2i # Hold location prior to the previous satellite call person II

    # Distance person I was from Distance Vector before
    dis1Prev = math.sqrt((p2l[0]-xline(p1i))**2+(p2l[1]-yline(p1i))**2)
    # Distance person II was from Distance Vector before
    dis2Prev = math.sqrt((xline(p2i)-p1l[0])**2+(yline(p2i)-p1l[1])**2)

    gps1data,gps2data = getGPSBothData() #gps call


    p1f = pos(gps1data,start) # use gps to find position, via conjugate
           gradient method
    p2f = pos(gps2data,start)
    dis =math.sqrt((p2f[0]-p1f[0])**2+(p2f[1]-p1f[1])**2) # How far apart are
           the people?
    # How far is the projection of person I's location from static point set
           by person II?
    dis1f = math.sqrt((p2l[0]-xline(p1f))**2+(p2l[1]-yline(p1f))**2)
    # How far is the projection of person II's location from static point set
           by person I?
    dis2f = math.sqrt((xline(p2f)-p1l[0])**2+(yline(p2f)-p1l[1])**2)

    if count%calls==0 and stay==False: # update line connecting the travelers
```

```
                every 3 satellite calls as
                                        # long as they are not less than 150
                                                    meters away
            p1l = p1f
            p2l = p2f
            m = (p2l[1]-p1l[1])/(p2l[0]-p1l[0])
            b = p1l[1]-m*p1l[0]

        elif stay == True and stayTime==0: # Person I is staying now, so we
                update
            stayTime+=1                   # we only update once, unless person
                    II stays farther than 150 meters
            p1l = p1f
            p2l = p2f
            m = (p2l[1]-p1l[1])/(p2l[0]-p1l[0])
            b = p1l[1]-m*p1l[0]



        p1i = p1f # set the initial point for next for next walk
        p2i = p2f


        # Here is our way to determine if
        if count ==1:
            r1 = turn(p1mid,p1l,p1f)

            r2 = turn(p2mid,p2l,p2f)
            print r1,r2
            if r1>0.99*math.pi:
                angCorr1=0
                extraAngCorr1=0
            if r2>0.99*math.pi:
                angCorr2=0
                extraAngCorr2=0







    return end() # how sad :(
    print count
    print 'Initial Distance: ',distance # So we can see how far we have come





#                    *
#
#                    ^
#
#                  ^ ^ ^
#
```

```
#                ^ ^ ^ ^ ^
#
#              ^ ^ ^ ^ ^ ^ ^
#                   | |
#         Merry Christmas Professor,
#         It has been a great semester thank you for all the knowledge.
#         - Sean & Mike




from numpy import loadtxt
from pylab import plot,axes,show,xlabel,ylabel,title
data = loadtxt('testu4.dat')
plot(data[:,1],data[:,2],'b-')
plot(data[:,5],data[:,6],'m-')
axes().set_aspect('equal', 'datalim')
xlabel('x'),ylabel('y')
title('x,y projection of trajectories')
show()
```