Appendiks

Koden som ble benyttet i simuleringen.

```
import numpy as np
from math import *    #this command gives you acces to math
    functions, such as sin(), pow() etc

# If you want to only analyze orbit of a single particle,
    set m2 = 0, and disregard files giving position, energy
    and angular momentum for particle 2.
##########################################
G = 1                # Gravitational constant
M = 1                # Central Mass
m1 = 0.01            # mass of particle 1
m2 = 0.00          # mass of particle 2

x1 = 1               #initial position in x-direction,
    particle 1
y1 =0                #initial position in y-direction,
    particle 1
vx1 = 0              #initial velocity in x-direction,
    particle 1
vy1 = 0.6            #initial velocity in y-direction,
    particle 1

x2 = 0               #initial position in x-direction,
    particle 2
y2 = 0.7               #initial position in y-direction,
    particle 2
vx2 = -0.8            #initial velocity in x-direction,
    particle 2
vy2 = 0.0             #initial velocity in y-direction,
    particle 2

dt = 0.00001           # integration timestep
endtime=20         #total simulation time

positionUncertainty = 0.00001      # Used to determine if
    the particle has returned to it's initial position, may
    be changed for accuracy, notice that it is uncertainty in
     position after change to dimensionless variables, where
    particle 1 will have distance 1 to the origin
plotspacing = 1     # How often variables are written to
    file (in terms of how often they are calculated)

##########################################
# Changing to dimensionless variables, using starting
    position and speed of particle 1 as referance
```

```
R01 = (x1**2 + y1**2)**0.5
x1 = x1 / R01
y1 = y1 / R01
x2 = x2 / R01
y2 = y2 / R01
V01 = (vx1**2 + vy1**2)**0.5
vx1 = vx1 / V01
vy1 = vy1 / V01
vx2 = vx2 / V01
vy2 = vy2 / V01

dt = dt * V01 / R01
endtime = endtime * V01 / R01

G = G / (R01 * V01**2)

time=0.0                                # this is the start
    time
maxEnergyDeviation = 0.0                # maximum deviation of
     energy for a particle, relative to start
maxAngularMomentumDeviation = 0.0       # maximum deviation of
     angular momentum for a particle, relative to start
i = 1                                   # variable used to
    save only each <plotspacing>'th value to file...

x01 = x1            # Saving starting positions to be able
    to calculate average kinetic and potential energy
y01 = y1
x02 = x2
y02 = y2

averageT1 = 0       # Variables for calculating average
    kinetic and potential energy of particles ov er one
    orbital period
averageT2 = 0
averageV1 = 0
averageV2 = 0

inStartingPosition1 = 1              # these are to keep
    track of whether the particle has left the starting
    position, and if it has compleated it's first revolution
inStartingPosition2 = 1
compleatedFirstRevolution1 = 0
compleatedFirstRevolution2 = 0

# create files to save simulation data in:
f = open('tertiarypos1.txt','w') # notice: the write option
    'w' erases previous data in the file
f2 = open('tertiarypos2.txt','w')
f3 = open('tertiaryenergy.txt','w')
```

```
f4 = open('tertiaryangularmomentum.txt','w')

def r1():    # distance between origin and particle 1
    return (x1**2 + y1**2)**0.5
def r2():    # distance between origin and particle 2
    return (x2**2 + y2**2)**0.5
def r1r2():  # distance between particle 1 and 2
    return ( (x1 - x2)**2 + (y1 - y2)**2)**0.5

def f1x():  # x component of f1 = F1/m1
    return -G*M/r1()**3*x1 - G*m2/r1r2()**3*(x1-x2)
def f1y():  # y component of f1 = F1/m1
    return -G*M/r1()**3*y1 - G*m2/r1r2()**3*(y1-y2)

def f2x(): # x component of f2 = F2/m2
    return -G*M/r2()**3*x2 - G*m1/r1r2()**3*(x2-x1)
def f2y():  # y component of f2 = F2/m2
    return -G*M/r2()**3*y2 - G*m1/r1r2()**3*(y2-y1)

E01 = 0.5*m1*(vx1**2 + vy1**2) - G*M*m1/r1() - G*m1*m2/r1r2
    () # Total starting energy particle 1
E02 = 0.5*m2*(vx2**2 + vy2**2) - G*M*m2/r2() - G*m1*m2/r1r2
    () # Total starting energy particle 2
E0 = E01 + E02  # Total startin energy of system

L01 = m1*(x1*vy1 - y1*vx1)   # total angular momentum in
    start of particle 1, in z-direction
L02 = m2*(x2*vy2 - y2*vx2)   # total angular momentum in
    start of particle 2, in z-direction
L0 = L01 + L02

print("The energy of particle 1 (dimensionless) is: %f" %
    E01)
print("The energy of particle 2 (dimensionless) is: %f" %
    E02)

while (time < endtime):

        time = time + dt
        i = i+1

        fx1 = f1x()     # calculating force/mass of particle
             1 at time t
        fy1 = f1y()
        vxm1 = vx1 + 0.5*dt*fx1 # first part og verlet
            algorthm for particle 1
        vym1 = vy1 + 0.5*dt*fy1

        fx2 = f2x()     # calculating force/mass of particle
             2 at time t
```

```
fy2 = f2y()
vxm2 = vx2 + 0.5*dt*fx2 # first part og verlet
    algorthm for particle 2
vym2 = vy2 + 0.5*dt*fy2

x1 = x1 + vxm1*dt       # updating new position,
    particle 1
y1 = y1 + vym1*dt
x2 = x2 + vxm2*dt       # updating new position,
    particle 2
y2 = y2 + vym2*dt

fx1 = f1x()     # calculating new force/mass to
    update new speed, particle 1
fy1 = f1y()
vx1 = vxm1 + 0.5*dt*fx1 # calculating new speed,
    particle 1
vy1 = vym1 + 0.5*dt*fy1

fx2 = f2x()     # calculating new force/mass to
    update new speed, particle 2
fy2 = f2y()
vx2 = vxm2 + 0.5*dt*fx2 # calculating new speed,
    particle 2
vy2 = vym2 + 0.5*dt*fy2

T1 = 0.5*m1*(vx1**2 + vy1**2)       # Kinetic energy
     of particles
T2 = 0.5*m2*(vx2**2 + vy2**2)

V1 = - G*M*m1/r1() - G*m1*m2/r1r2() # Potential
    energy of particles
V2 = - G*M*m2/r2() - G*m1*m2/r1r2()

E1 = T1 + V1                        # total energy
    of particles
E2 = T2 + V2
E= E1 + E2

L1 = m1*(x1*vy1 - y1*vx1)   # total angular momentum
     of particle 1, in z-direction
L2 = m2*(x2*vy2 - y2*vx2)   # total angular momentum
     of particle 2, in z-direction
L = L1 + L2

if abs(E0) > 0 and (abs(1-E/E0) > abs(
    maxEnergyDeviation)):  # Checking if deviation of
     energy or ang.momentum has grown.
     maxEnergyDeviation = 1-E/E0
```

```
if abs(L0) > 0 and (abs(1-L/L0) > abs(
    maxAngularMomentumDeviation)):
     maxAngularMomentumDeviation = L/L0 - 1

averageT1 = averageT1 + T1      # adding energy of
    current state, will divide by number of states
    later
averageT2 = averageT2 + T2
averageV1 = averageV1 + V1
averageV2 = averageV2 + V2

# Checking if particles have left their strting
    positions
if abs(x1-x01) > positionUncertainty and abs(y1 -
    y01) > positionUncertainty and
    inStartingPosition1:
     inStartingPosition1 = 0
if abs(x2-x02) > positionUncertainty and abs(y2 -
    y02) > positionUncertainty and
    inStartingPosition2:
     inStartingPosition2 = 0

# Checking if particle 1 has returned to it's
    original position, displaying info if it has
if compleatedFirstRevolution1 ==0 and
    inStartingPosition1 ==0 and (abs(x1-x01) <
    positionUncertainty) and (abs(y1 - y01) <
    positionUncertainty):
     print("\nParticle 1:")
     outPut = time*R01/V01   # Changing back to right
         dimension
     print("Orbital period is %f" % outPut)
     outPut = averageT1*dt/time  # averaging T1
     print("Average kintetic energy is %f" % outPut)
     outPut = averageV1*dt/time  # averaging V1
     print("Average potential energy is %f\n" %
         outPut)
     compleatedFirstRevolution1 = 1

# Checking if particle 2 has returned to it's
    original position, displaying info if it has
if compleatedFirstRevolution2==0 and
    inStartingPosition2==0 and (abs(x2-x02) <
    positionUncertainty) and (abs(y2 - y02) <
    positionUncertainty):
     print("\nParticle 2:")
     outPut = time*R01/V01   # Changing back to right
         dimension
     print("Orbital period is %f" % outPut)
     outPut = averageT2*dt/time   # averaging T2
```

```
            print("Average kintetic energy is %f" % outPut)
            outPut = averageV2*dt/time   # averaging V2
            print("Average potential energy is %f\n" %
                outPut)
            compleatedFirstRevolution2 = 1


        if (i % plotspacing == 0):  # writing to files only
            each <plotspacing>'th iteration

            f.write("%f %f\n" % (x1,y1))    # writing to
                files
            f2.write("%f %f\n" % (x2,y2))
            outPut = E1 + E2
            f3.write("%f %f\n" % (time,outPut))
            outPut = L1 + L2
            f4.write("%f %f\n" % (time, outPut))


print("Maximum deviation of energy for a particle relative
    to start energy is %f" % maxEnergyDeviation)
print("Maximum deviation of angular momentum for a particle
    relative to start angular momentum is %f" %
    maxAngularMomentumDeviation)

#closing files
f.close()
f2.close()
f3.close()
f4.close()

################
# Making a file containing the orbit of particle 1
    calculated by the exact solution if m2 = 0 (or
    approximate if m2 << m1), otherwise disregard this file

# create file to save simulation data in:
f = open('tertiarypos1EXACT.txt','w') # notice: the write
    option 'w' erases previous data in the file

a = -G*M*m1/(2.0*E01)
e = ( 1.0 + 2.0*E01*L01**2.0/(m1**3*G**2.0*M**2.0) )**0.5

def r1EXACT(theta):
    # ignoring the constant inside cos() as we are
        interested in plotting a complete orbit
    return a*(1.0-e**2.0)/( 1.0 + e*cos(theta) )

#for Theta in np.nditer(np.arange(0.0,2.0*pi,0.01)):
Theta = 0
```

```
while Theta < 2*pi:
    Theta = Theta + 0.3
    x1 = -r1EXACT(Theta)*cos(Theta)
    y1 = r1EXACT(Theta)*sin(Theta)
    f.write("%f %f\n" % (x1,y1))    # writing to file


# closing file
f.close
```