Numerisk simulering av vekselvirkende partikler i sentralt gravitasjonsfelt.

Håkon Taskén, Paul Thrane

Sammendrag

Verlet integrasjon ble benyttet for å beregne partikkelbevegelser numerisk. Systemene som ble undersøkt var henholdsvis en og to partikler i et sentralt gravitasjonsfelt, i topartikkel-systemet ble det tatt hensyn til gravitasjonskraft mellom partiklene.

1. Teori og metode

Bevegelsen til en partikkel i posisjon $\vec{r}_i$ med masse m_i, utsatt for en kraft $\vec{F}_i$, vil være bestemt av Newtons bevegelsesligning

$$m_i \frac{\mathrm{d}^2\vec{r}_i}{\mathrm{d}t^2} = \vec{F}_i. \tag{1}$$

Ligning (1) kan skrives om til to koblede differensialligninger [1]

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = \vec{f}_i, \tag{2}$$

$$\frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} = \vec{v}_i, \tag{3}$$

hvor $\vec{f}_i$ er kraft per masse på partikkelen og $\vec{v}_i$ er hastigheten til partikkelen. Dersom initialbetingelser for $\vec{r}_i$ of $\vec{v}_i$ er kjent kan videre tidsforløp bestemmes ved numerisk løsning av (2) og (3) hvis $\vec{f}_i$ kan beregnes på grunnlag av posisjon og hastighet til partikkelen. En metode som kan benyttes til dette er Verlet integrasjon, hvor $\vec{r}_i(t+\Delta t)$ og $\vec{v}_i(t+\Delta t)$ beregnes etter kjennskap til $\vec{r}_i(t)$ ved følgende ligninger[1]

$$\vec{v}_i(t+\frac{\Delta t}{2}) = \vec{v}_i(t) + \vec{f}_i(t)\frac{\Delta t}{2}, \tag{4}$$

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \vec{v}_i(t+\frac{\Delta t}{2})\Delta t, \tag{5}$$

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t+\frac{\Delta t}{2}) + \vec{f}_i(t+\Delta t)\frac{\Delta t}{2}. \tag{6}$$

Dersom potensialet til partikkelen, V_i, er kjent, kan kraften på partikkelen beregnes ut fra sammenhengen $\vec{F}_i = -\nabla V_i$. I dette numeriske forsøket ble Verlet integrasjon benyttet for å beregne bevegelsene til én og to massive partikler i bane rundt et sentralt legeme med masse M og med gravitasjon som eneste vekselvirkende kraft. Det ble antatt at partiklenes masse, m_1 og m_2, var mye mindre enn M slik at den sentrale massen står fast i origo. For én enkelt partikkel blir potensialet

$$V_1 = \frac{GMm_1}{r_1}, \tag{7}$$

med G en konstant som bestemmer styrken på gravitasjonskraften. For to partikler blir potensialet på partikkel i også påvirket av posisjonen til partikkel j

$$V_i = \frac{GMm_i}{r_i} + \frac{Gm_im_j}{|\vec{r}_i - \vec{r}_j|}. \tag{8}$$

For å undersøke nøyaktignheten av den numeriske metoden ble bevaring av energi og dreieimpuls undersøkt. I tillegg ble de beregnede orbitalene til en enkelt partikkel sammenlignet med det eksakte uttrykket [2]

$$r = \frac{a(1-e^2)}{1+e\cos(\theta-\theta')}, \tag{9}$$

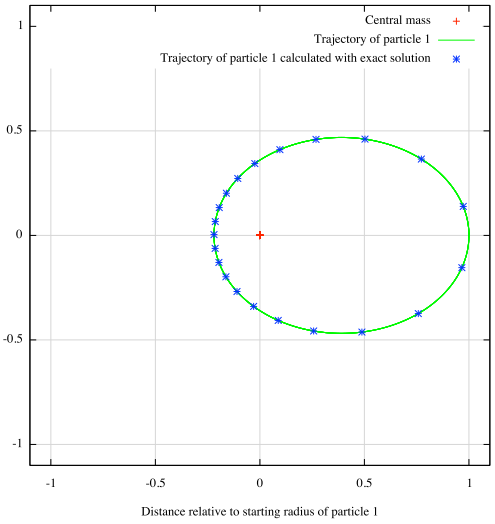
hvor (r,θ) er posisjonen til partikkelen i polarkoordinater. Konstantene $a = -GM/2E$, og $e = (1+2El^2/m_1G^2M^2)^{0.5}$, der E er energien til partikkelen og l er dreieimpulsen om origo. Konstanten θ' spesifiseres av initialbetingelsene og kan settes lik null om en kun er interessert i hele orbitalen. Videre ble også virialteoremet for å undersøke nøyaktigheten av metoden da den gir en sammenheng mellom gjennomsnittlig kinetisk energi $\bar{T}$ og gjennomsnittlig potensiell energi $\bar{V}$ for en partikkel i et sentralt gravitasjonsfelt gitt ved[2]

$$\frac{\bar{T}}{\bar{V}} = -\frac{1}{2}. \tag{10}$$

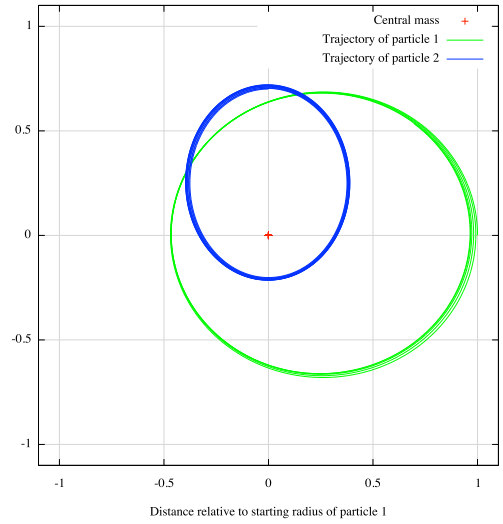
Den numeriske metoden skal i utgangspunktet kunne benyttes på alle initialbetingelser, men i dette tilfellet er det valgt å kun undersøke bundne tilstander, som har $E = T + V < 0$ [2]. Koden som er blitt benyttet for å løse ligning (2) og (3) er i appendiks.

2. Resultat

For kun en partikkel gir metoden en godt definert ellipseformet orbital, ettersom kun tilfeller med $E < 0$ undersøkes. Figur 1 viser resultatet av en simulasjon sammen med tilhørende eksakte løsning, beregnet med ligning (9). Ved å teste for ulike initialbetingelser ble det funnet at et tidsteg på under en promille av beregnet omløpstid som regel resulterte i en endring av total energi i størrelsesorden



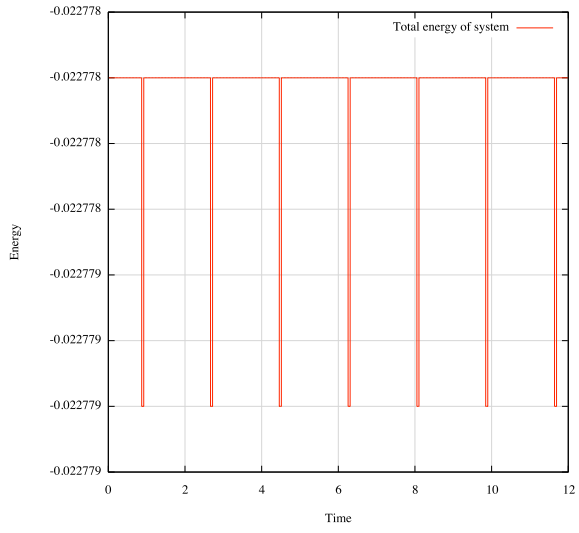
Figur 1: Numerisk beregnet bevegelse for en partikkel i et sentralt gravitasjonsfelt vist med grønn linje. Partikkelen beveger seg mot klokken med omløpstid 1,8. Eksakt bane for partikkelen er beregnet med ligning (9) og er vist med blå stjerner. Partikkelen starter i posisjon (1;0)



Figur 3: Bevegelse for to vekselvirkende partikler i et sentralt gravitasjonsfelt. Partikkel 1 går mot klokken med omløpstid 3,0, partikkel 2 går mot klokken med omløpstid 1,4. Partikkel 1 starter i posisjon (1;0), og partikkel 2 starter i (0;0,7)

0.01%, mens et tidssteg på omtrent 1% av beregnet omløpstid resulterte i en endring av energi i størrelsesorden 1%. Figur 2 viser tilhørende total energi som funksjon av tid for systemet vist i figur 1. Størst endring i total energi ble funnet å skje når partikkelen befant seg nærmest origo. For initialbetingelser som resulterte i god bevaring av total energi ble det ikke registrert noen endring av total dreieimpuls.

$\bar{T}$ og $\bar{V}$ ble beregnet numerisk og sammenlignet med den teoretiske sammenhengen gitt av ligning (10). Generelt for initialbetingelser som resulterte i endring av total energi på mindre enn 1% ble det funnet at forholdet $\bar{T}/\bar{V}$ hadde et avvik fra teoretisk verdi på mindre enn 0.1%.

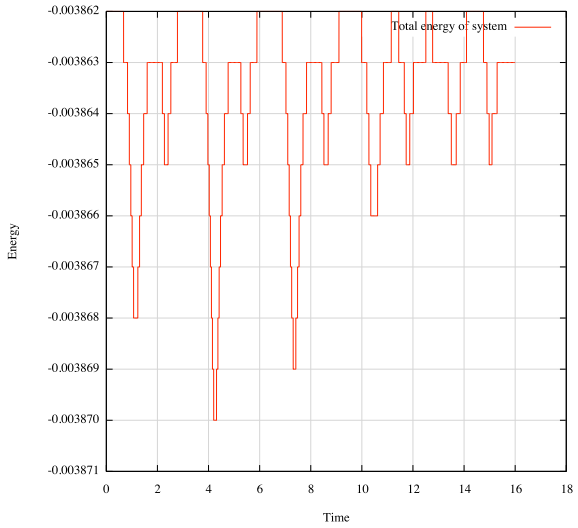


Figur 2: Total energi for en partikkel i et sentralt gravitasjonsfelt. Viser tidsforløp for system vist i figur 1.

For to vekselvirkende partikler i et sentralt gravitasjonsfelt ble nøyaktigheten dåligere. Dersom initialbetingelsene førte partiklene i nærheten av hverandre ble det ofte store avvik i total energi og ustabile orbitaler. Dersom initialbetingelsene ble bestemt slik at partiklene ikke kom for nærme hverandre ble imidlertid banene ganske stabile, men med en regelmessig forskyvning av orbitalen iløpet av simuleringen, figur 3 viser et slikt tilfelle. Figur 4 viser total energi som funksjon av tid for samme simulering, som på det meste avviker 0.20% fra startenergi. Endring av total dreieimpuls for samme simulering var for liten til å kunne leses av, noe som også gjaldt andre simuleringer uten store avvik av total energi. Som for simulering av en partikkel ble det størst endring av energi for de posisjonene der partiklene var i nærheten av origo eller hverandre.

3. Diskusjon og konklusjon

For både simulering av en og to partikler er det tydelig at unøyaktigheten i den numeriske metoden øker med kortere avstand mellom massene. Ved å undersøke figur 1 og 2 kan en se at endring av energi opptrer en gang per runde, og da i det punktet hvor partikkelen befinner seg nærmest origo. Tilsvarende kan en se i figur 3 og 4 at energien til topartikkel-systemet endres mest i intervallet der orbitalene er i nærheten av hverandre. Denne effekten kan forklares ved å se på ligning (4) og (6); feilen vil være avhengig av hvor raskt $\vec{f}_i$ endrer seg. Videre vil $\vec{f}_i$ endre seg raskest når partikkelen beveger seg fort og avstanden til



Figur 4: Total energi for system av to vekselvirkende partikler i et sentralt gravitasjonsfelt. Viser tidsforløp for system vist i figur 3

andre masser er liten, som begge opptrer samtidig i nærheten av andre masser.

Generelt virket metoden godt. For en enkelt partikkel samsvarte beregningene overens med ligning (9) og ligning (10), samtidig som total energi og dreieimpuls var bevart uten å måtte gjøre Δt veldig liten. For to vekselvirkende partikler ble metoden noe mer ustabil, men så lenge initialbetingelser ble valgt slik at partiklene ikke beveget seg for nærme hverandre ga beregningene svakt forkjøvede orbitaler. For to partikler måtte imidlertid Δt i noen tilfeller gjøres et par størrelsesordner mindre for å oppnå samme bevaring av energi som for en enkelt partikkel.

Referanser

[1] Prosjektbeskrivelse, Classical Mechanics TFY4345 - Computational Physics Project. NTNU, Høsten 2015.

[2] Goldstein, Safko, Poole. Classical Mechanics. Pearson Education Limited, Pearson new international edition, 3. utgave, 2014.

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
```