Project

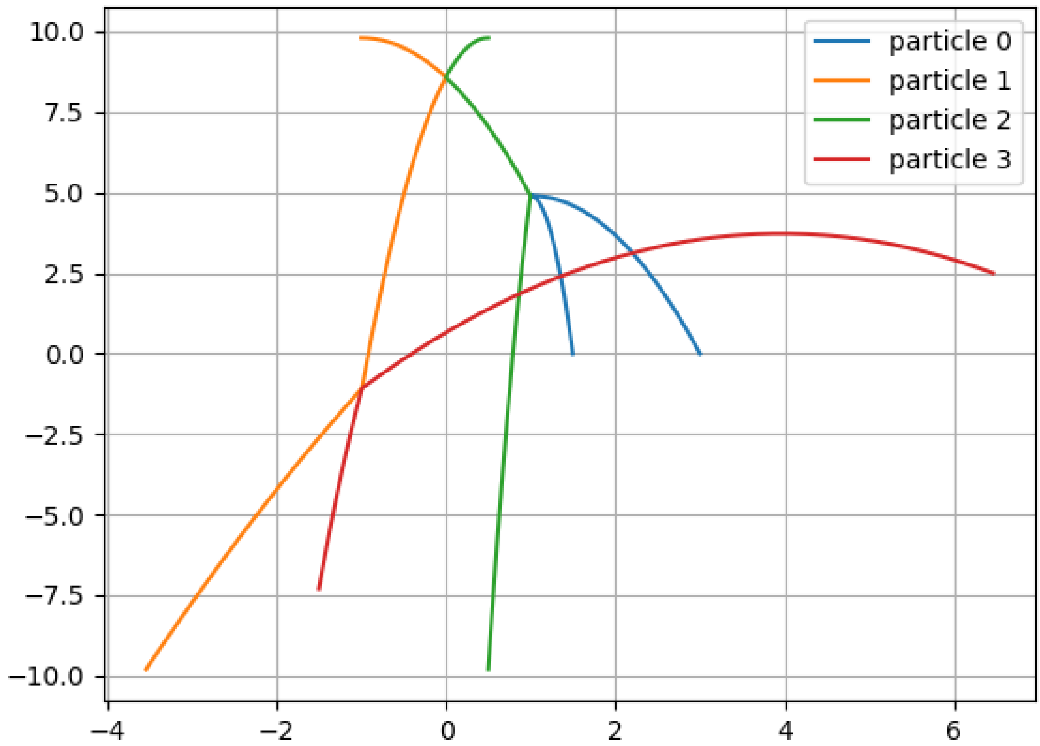
2017550029 최효규

3. Dynamics of multiple particles interacting

1-D collision

```
import numpy as np
import matplotlib.pyplot as plt

## distance of 2-D
## x = [ x, x', y, y']
## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5

## x      = [ x, x', y, y']
## F = dx/dt = [ v_x, a_x, v_y, a_y ]
def F(t,x):
        g = -9.8 # m/sec^2
        F = np.zeros(4)
        F[0] = x[1]
        F[1] = 0
        F[2] = x[3]
        F[3] = g
        return F

## using runge-kutta method
def integrate(F,tStart,x_init,tStop,h):
    def dx(F,t,x,h):
        k0 = h*F(t,x)
        k1 = h*F(t+h/2.0,x+k0/2.0)
        k2 = h*F(t+h/2.0,x+k1/2.0)
        k3 = h*F(t+h,x+k2)
        return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
    n = len(x_init)
    t = tStart
    x = x_init
    ts = []
    ts.append(t)
    xs = []
    for i in range(n):
        xs.append([])
        xs[i].append(x[i])
    while t < tStop:
        # collision test
        for i in range(n):
            for j in range(i+1,n):
                if distance(x[i],x[j]) < d:
                    temp = x[i][1]; x[i][1] = x[j][1]; x[j][1] = temp
        for i in range(n):
            x[i] = x[i] + dx(F,t,x[i],h)
            xs[i].append(x[i])
        t = t + h
        ts.append(t)
    for i in range(n):
```

```python
        xs[i] = np.array(xs[i])
    return np.array(ts),xs

d = 1.0e-03
h = 0.01

x0 = np.array([1.5,-0.5,0.0,9.8])
x1 = np.array([-1 ,2   ,9.8,0.0])
x2 = np.array([0.5,-1  ,9.8,0.0])
x3 = np.array([6.4599999999,-5  ,2.499,4.9])
x_init = [x0,x1,x2,x3]

ts, xs = integrate(F,0.0,x_init,2.0,h)

legend = []
for i in range(len(xs)):
    plt.plot(xs[i][:,0],xs[i][:,2],'-')
    legend.append('particle '+str(i))

plt.legend(legend)

plt.grid()
plt.show()
```



Integrate function.
if x1 collide with x2, v1_xf = v2_xi, v2_xf = v1_xi. (Momentum, Kinetic Energy conservation, e = 1.0)

```
## using runge-kutta method
def integrate(F,tStart,x_init,tStop,h):
    def dx(F,t,x,h):
        k0 = h*F(t,x)
        k1 = h*F(t+h/2.0,x+k0/2.0)
        k2 = h*F(t+h/2.0,x+k1/2.0)
        k3 = h*F(t+h,x+k2)
        return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
    n = len(x_init)
    t = tStart
    x = x_init
    ts = []
    ts.append(t)
    xs = []
    for i in range(n):
        xs.append([])
        xs[i].append(x[i])
    while t < tStop:
        # collision test
        for i in range(n):
            for j in range(i+1,n):
                if distance(x[i],x[j]) < d:
                    temp = x[i][1]; x[i][1] = x[j][1]; x[j][1] = temp
        for i in range(n):
            x[i] = x[i] + dx(F,t,x[i],h)
            xs[i].append(x[i])
        t = t + h
        ts.append(t)
    for i in range(n):
        xs[i] = np.array(xs[i])
    return np.array(ts),xs
```

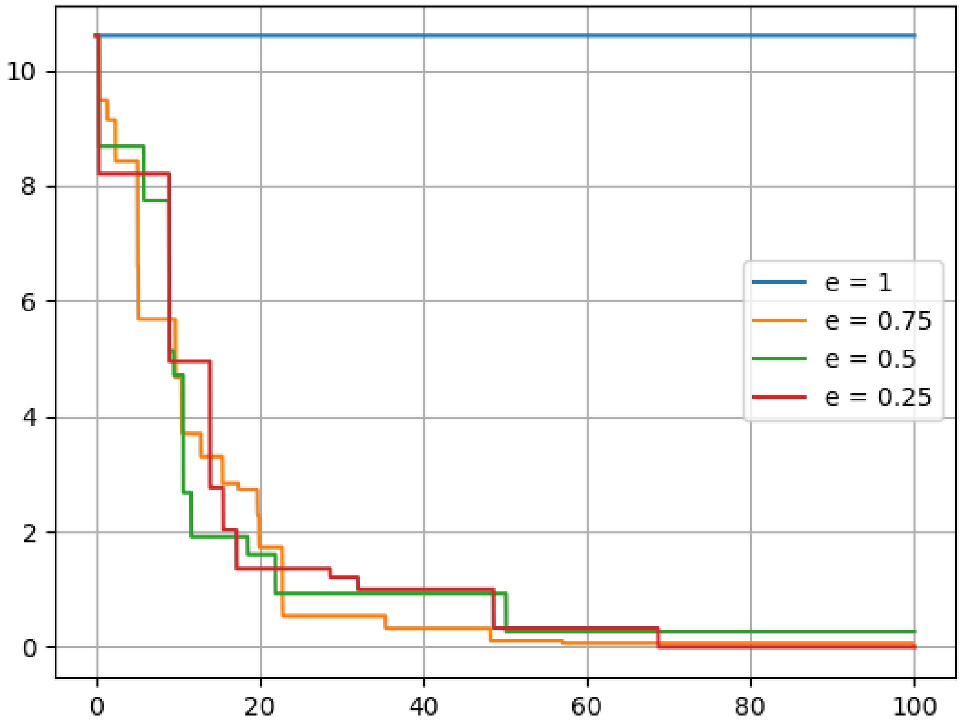
2-D collision

```
import numpy as np
import matplotlib.pyplot as plt

## distance of 2-D
## x = [ x, x', y, y']
## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5

## x      = [ x, x', y, y']
## F = dx/dt = [ v_x, a_x, v_y, a_y ]
def F(t,x):
        F = np.zeros(4)
        F[0] = x[1]
        F[1] = 0
        F[2] = x[3]
        F[3] = 0
        return F

## runge-kutta methon in many body problem
def integrate(F,tStart,x_init,tStop,h):
    def dx(F,t,x,h):
        k0 = h*F(t,x)
        k1 = h*F(t+h/2.0,x+k0/2.0)
        k2 = h*F(t+h/2.0,x+k1/2.0)
        k3 = h*F(t+h,x+k2)
        return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
    n = len(x_init)
    t = tStart
    x = x_init
    E = energy(x)
    Es = []
    Es.append(E)
    ts = []
    ts.append(t)
    xs = []
    for i in range(n):
        xs.append([])
        xs[i].append(x[i])
    while t < tStop:
        # collision test
        for i in range(n):
            if x[i][0] < -2 or x[i][0] > 2: x[i][1] = -x[i][1]
            if x[i][2] < -2 or x[i][2] > 2: x[i][3] = -x[i][3]
            for j in range(i+1,n):
                if distance(x[i],x[j]) < d:
                    collision(x[i],x[j],e)
        for i in range(n):
            x[i] = x[i] + dx(F,t,x[i],h)
            xs[i].append(x[i])
        E = energy(x)
        Es.append(E)
        t = t + h
        ts.append(t)
    for i in range(n):
        xs[i] = np.array(xs[i])
    return np.array(ts),xs,np.array(Es)
```

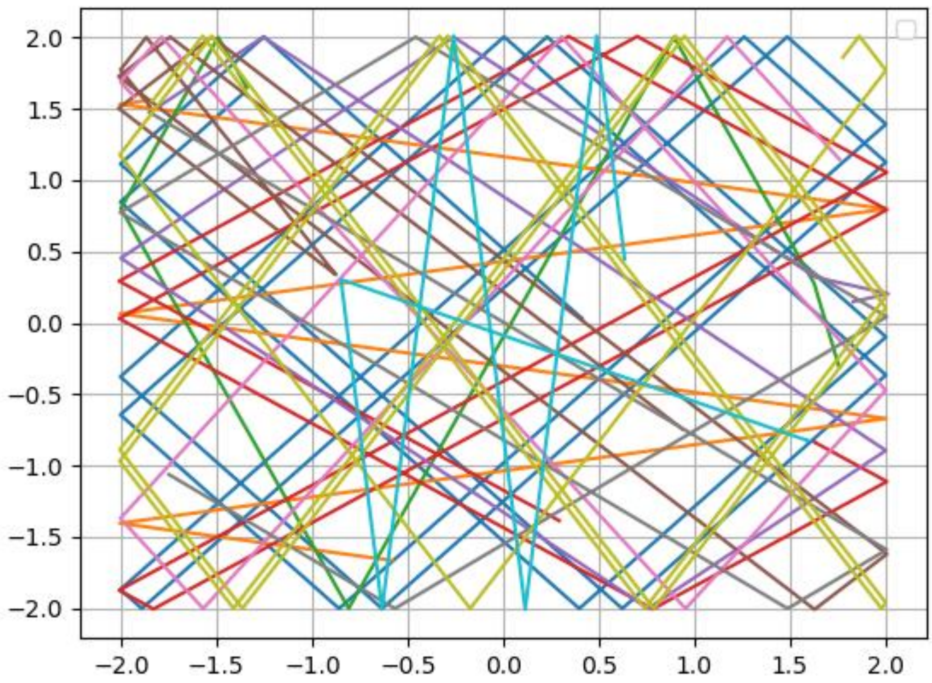
```

## 2-D collision
## e is coefficient of restitution
## p = [ x, x', y, y']
def collision(p1,p2,e):
    tan = (p2[2] - p1[2])/(p2[0] - p1[0])
    cos = 1/(tan**2+1)**0.5
    sin = (1-cos**2)**0.5

    u1 = p1.copy()
    u2 = p2.copy()

    p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin + u1[3]*sin*cos)
    p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos - u2[3]*sin*sin))

    p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin + u2[3]*sin*cos)
    p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos - u1[3]*sin*sin))
    return 0

e = 1.0 # elastic collision
d = 5.0e-02 # (radius * 2) of particle
h = 0.01    # time interval
n = 10  # count of particles

## Calculate total Kinetic Energy
def energy(x):
    E_tot = 0
    for i in range(len(x)):
        E_tot = E_tot + (x[i][1]**2 + x[i][3]**2)/2
    return E_tot

## Initial condition of particles
from random import random
x_init = []
for i in range(n):
    x_init.append(np.array([(random()-0.5)*4,(random()-0.5)*4,(random()-0.5)*4,(random()-0.5)*4]))

## Plot Energy graph
## e = 1.0, 0.75, 0.5, 0.25
legend = []
for i in [1,0.75,0.5,0.25]:
    e = i
    legend.append('e = '+str(e))
    x = x_init.copy()
    ts, xs, Es = integrate(F,0.0,x,100.0,h)
    plt.plot(ts,Es,'-')

plt.legend(legend)
plt.grid()
plt.savefig('n-body_energy.png')
plt.close()

## Plot orbitals of particles graph
```

```python
## Elastic collision, e = 1
legend = []
for i in range(len(xs)):
    plt.plot(xs[i][:,0],xs[i][:,2],'-')
    if range(len(xs)<6): legend.append('particle '+str(i))

plt.legend(legend)
plt.grid()
plt.savefig('n-body_orbit.png')
```

n-body_energy.png



n-body_orbit.png


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

2-D collision function in integrate

```
## 2-D collision
## e is coefficient of restitution
## p = [ x, x', y, y']
def collision(p1,p2,e):
    tan = (p2[2] - p1[2])/(p2[0] - p1[0])
    cos = 1/(tan**2+1)**0.5
    sin = (1-cos**2)**0.5

    u1 = p1.copy()
    u2 = p2.copy()

    p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin + u1[3]*sin*cos)
    p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos - u2[3]*sin*sin))

    p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin + u2[3]*sin*cos)
    p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos - u1[3]*sin*sin))
    return 0
```

use rotate matrix to calculate.

Visualizing 2-D collision. (OpenGL)

```
import numpy as np
from time import time
from sys import exit

## distance of 2-D
## x = [ x, x', y, y']
## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5

## x    = [ x, x', y, y']
## F = dx/dt = [ v_x, a_x, v_y, a_y ]
def F(t,x):
        g = -9.8 # m/sec^2
        F = np.zeros(4)
        F[0] = x[1]
        F[1] = 0
        F[2] = x[3]
        F[3] = 0
        return F

## runge-kutta methon in many body problem
def integrate(F,t,x,h):
    def dx(F,t,x,h):
        k0 = h*F(t,x)
        k1 = h*F(t+h/2.0,x+k0/2.0)
        k2 = h*F(t+h/2.0,x+k1/2.0)
        k3 = h*F(t+h,x+k2)
        return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
    n = len(x_init)
    # collision test
    for i in range(n):
        if x[i][0] < -2 or x[i][0] > 2:
            x[i][1] = -x[i][1]
            if x[i][0] < -2: x[i][0] = -2
            else: x[i][0] = 2
        if x[i][2] < -2 or x[i][2] > 2:
            x[i][3] = -x[i][3]
            if x[i][2] < -2: x[i][2] = -2
            else: x[i][2] = 2
        for j in range(i+1,n):
            if distance(x[i],x[j]) < d:
                collision(x[i],x[j],e)
    for i in range(n):
        x[i] = x[i] + dx(F,t,x[i],h)

## 2-D collision
## e is coefficient of restitution
## p = [ x, x', y, y']
def collision(p1,p2,e):
    tan = (p2[2] - p1[2])/(p2[0] - p1[0])
    cos = 1/(tan**2+1)**0.5
    sin = (1-cos**2)**0.5

    u1 = p1.copy()
    u2 = p2.copy()

```

```
    p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin + u1[3]*sin*cos)
    p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos - u2[3]*sin*sin))

    p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin + u2[3]*sin*cos)
    p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos - u1[3]*sin*sin))
    return 0

e = 1.0 # elastic collision
d = 5.0e-02
h = 0.01

def energy(x):
    E_tot = 0
    for i in range(len(x)):
        E_tot = E_tot + (x[i][1]**2 + x[i][3]**2)/2
    return E_tot

n = 100
from random import random
x_init = []
for i in range(n):
    x_init.append(np.array([(random()-0.5)*3.9,(random()-0.5)*2,(random()-0.5)*3.9,(random()-0.5)*2]))
x = x_init.copy()

ts = [0]
Es = [energy(x)]

def changeSize(w,h):
    if h==0: h=1
    ratio = w/h

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()

    glViewport(0,0,w,h)

    gluPerspective(45,ratio,0.1,1000)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0.0,0.0,10.0,0.0,0.0,-1.0,0.0,1.0,0.0)

def drawSphere(x,y,z):
    glTranslatef(x,y,z)
    glutSolidSphere(d/2,10,10)
    glTranslatef(-x,-y,-z)

time1 = time(); time2 = time(); t = 0; ite = 0
def renderScene():
    global time1, time2, t, ite, x
    ite = ite + 1
    glClearColor(0,0,0,0)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glPushMatrix()
```

```python

    for i in range(len(x)):
        glColor3f(0.1,0.1,0.1)
        drawSphere(x[i][0],x[i][2],0)

    glColor3f(0.9,0.9,0.9)
    glBegin(GL_QUADS)
    glVertex3f(2,2,0)
    glVertex3f(-2,2,0)
    glVertex3f(-2,-2,0)
    glVertex3f(2,-2,0)
    glEnd()

    glColor3f(0,1,0)
    drawSphere(2,2,0)

    glColor3f(0,0,0)
    drawSphere(-2,2,0)

    glColor3f(1,0,0)
    drawSphere(-2,-2,0)

    glColor3f(0,0,1)
    drawSphere(2,-2,0)

    glPopMatrix()
    dt = time2 - time1
    glutSwapBuffers()
    if ite > 1:
        integrate(F,t,x,dt)
        t = t + dt
        E = energy(x)
        Es.append(E)
        ts.append(t)
    time1 = time2
    time2 = time()

from OpenGL._bytes import as_8_bit
ESC = as_8_bit('\033')
def exitKey(key,x,y):
    if key == ESC:
        exit()
    if key == as_8_bit('m'):
        import matplotlib.pyplot as plt
        plt.plot(ts,Es,'-')
        plt.grid()
        plt.savefig('n-body_'+str(ite)+'.png')

## GLUT animation in main function
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

if __name__ == '__main__':
    glutInit()
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA)
    glutInitWindowPosition(0,30)
    glutInitWindowSize(1280,720)
```

```
    glutCreateWindow('n-body problem')
    glutDisplayFunc(renderScene)

    glutIdleFunc(renderScene)

    glutReshapeFunc(changeSize)

    glutKeyboardFunc(exitKey)

    glEnable(GL_DEPTH_TEST)
    glutMainLoop()
```