

Project

2017550029 최효규

3. Dynamics of multiple particles interacting

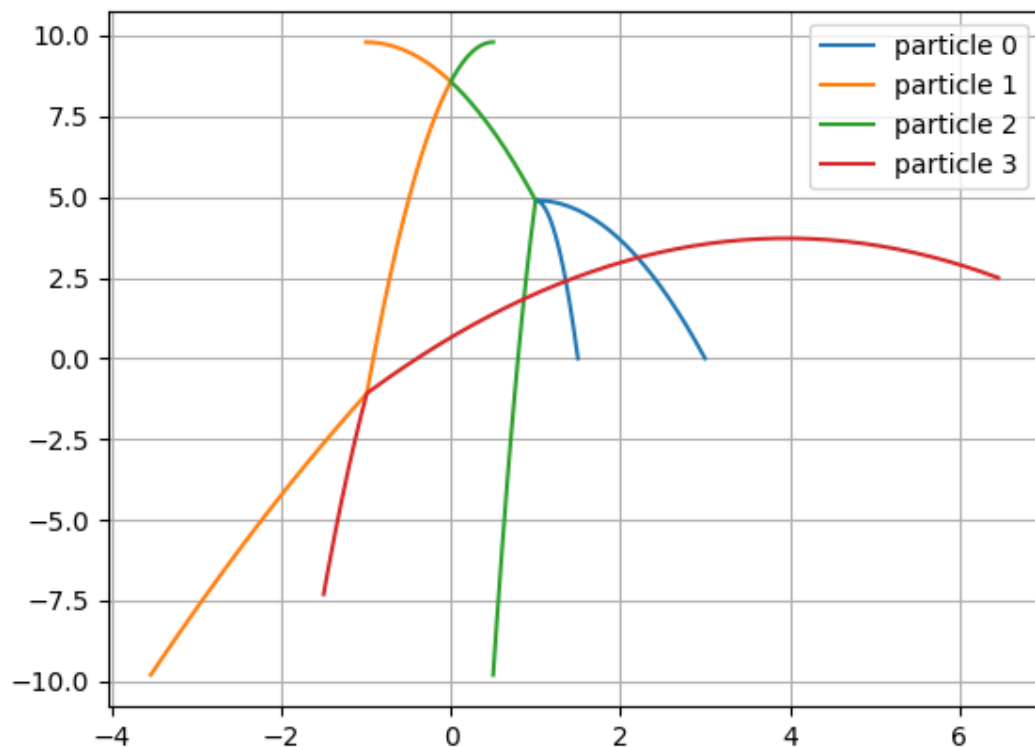
1-D collision

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. ## distance of 2-D
5. ## x = [ x, x', y, y']
6. ## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
7. distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5
8.
9. ## x      = [ x, x', y, y']
10. ## F = dx/dt = [ v_x, a_x, v_y, a_y ]
11. def F(t,x):
12.     g = -9.8 # m/sec^2
13.     F = np.zeros(4)
14.     F[0] = x[1]
15.     F[1] = 0
16.     F[2] = x[3]
17.     F[3] = g
18.     return F
19.
20. ## using runge-kutta method
21. def integrate(F,tStart,x_init,tStop,h):
22.     def dx(F,t,x,h):
23.         k0 = h*F(t,x)
24.         k1 = h*F(t+h/2.0,x+k0/2.0)
25.         k2 = h*F(t+h/2.0,x+k1/2.0)
26.         k3 = h*F(t+h,x+k2)
27.         return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
28.     n = len(x_init)
29.     t = tStart
30.     x = x_init
31.     ts = []
32.     ts.append(t)
33.     xs = []
34.     for i in range(n):
35.         xs.append([])
36.         xs[i].append(x[i])
37.     while t < tStop:
38.         # collision test
39.         for i in range(n):
40.             for j in range(i+1,n):
41.                 if distance(x[i],x[j]) < d:
42.                     temp = x[i][1]; x[i][1] = x[j][1]; x[j][1] = temp
43.         for i in range(n):
44.             x[i] = x[i] + dx(F,t,x[i],h)
45.             xs[i].append(x[i])
46.         t = t + h
47.         ts.append(t)
48.     for i in range(n):
```

```

49.     xs[i] = np.array(xs[i])
50.     return np.array(ts),xs
51.
52. d = 1.0e-03
53. h = 0.01
54.
55. x0 = np.array([1.5,-0.5,0.0,9.8])
56. x1 = np.array([-1,2,9.8,0.0])
57. x2 = np.array([0.5,-1,9.8,0.0])
58. x3 = np.array([6.459999999,-5,2.499,4.9])
59. x_init = [x0,x1,x2,x3]
60.
61. ts, xs = integrate(F,0.0,x_init,2.0,h)
62.
63. legend = []
64. for i in range(len(xs)):
65.     plt.plot(xs[i][:,0],xs[i][:,2],'-')
66.     legend.append('particle '+str(i))
67.
68. plt.legend(legend)
69.
70. plt.grid()
71. plt.show()

```



Integrate function.

if x1 collide with x2, $v1_{xf} = v2_{xi}$, $v2_{xf} = v1_{xi}$. (Momentum, Kinetic Energy conservation, $e = 1.0$)

```
1. ## using runge-kutta method
2. def integrate(F,tStart,x_init,tStop,h):
3.     def dx(F,t,x,h):
4.         k0 = h*F(t,x)
5.         k1 = h*F(t+h/2.0,x+k0/2.0)
6.         k2 = h*F(t+h/2.0,x+k1/2.0)
7.         k3 = h*F(t+h,x+k2)
8.         return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
9.     n = len(x_init)
10.    t = tStart
11.    x = x_init
12.    ts = []
13.    ts.append(t)
14.    xs = []
15.    for i in range(n):
16.        xs.append([])
17.        xs[i].append(x[i])
18.    while t < tStop:
19.        # collision test
20.        for i in range(n):
21.            for j in range(i+1,n):
22.                if distance(x[i],x[j]) < d:
23.                    temp = x[i][1]; x[i][1] = x[j][1]; x[j][1] = temp
24.        for i in range(n):
25.            x[i] = x[i] + dx(F,t,x[i],h)
26.            xs[i].append(x[i])
27.        t = t + h
28.        ts.append(t)
29.    for i in range(n):
30.        xs[i] = np.array(xs[i])
31.    return np.array(ts),xs
```

2-D collision

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. ## distance of 2-D
5. ## x = [ x, x', y, y']
6. ## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
7. distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5
8.
9. ## x      = [ x, x', y, y']
10. ## F = dx/dt = [ v_x, a_x, v_y, a_y ]
11. def F(t,x):
12.     F = np.zeros(4)
13.     F[0] = x[1]
14.     F[1] = 0
15.     F[2] = x[3]
16.     F[3] = 0
17.     return F
18.
19. ## runge-kutta methon in many body problem
20. def integrate(F,tStart,x_init,tStop,h):
21.     def dx(F,t,x,h):
22.         k0 = h*F(t,x)
23.         k1 = h*F(t+h/2.0,x+k0/2.0)
24.         k2 = h*F(t+h/2.0,x+k1/2.0)
25.         k3 = h*F(t+h,x+k2)
26.         return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
27.     n = len(x_init)
28.     t = tStart
29.     x = x_init
30.     E = energy(x)
31.     Es = []
32.     Es.append(E)
33.     ts = []
34.     ts.append(t)
35.     xs = []
36.     for i in range(n):
37.         xs.append([])
38.         xs[i].append(x[i])
39.     while t < tStop:
40.         # collision test
41.         for i in range(n):
42.             if x[i][0] < -2 or x[i][0] > 2: x[i][1] = -x[i][1]
43.             if x[i][2] < -2 or x[i][2] > 2: x[i][3] = -x[i][3]
44.             for j in range(i+1,n):
45.                 if distance(x[i],x[j]) < d:
46.                     collision(x[i],x[j],e)
47.         for i in range(n):
48.             x[i] = x[i] + dx(F,t,x[i],h)
49.             xs[i].append(x[i])
50.         E = energy(x)
51.         Es.append(E)
52.         t = t + h
53.         ts.append(t)
54.     for i in range(n):
55.         xs[i] = np.array(xs[i])
56.     return np.array(ts),xs,np.array(Es)
```

```

57.
58. ## 2-D collision
59. ## e is coefficient of restitution
60. ## p = [ x, x', y, y']
61. def collision(p1,p2,e):
62.     tan = (p2[2] - p1[2])/(p2[0] - p1[0])
63.     cos = 1/(tan**2+1)**0.5
64.     sin = (1-cos**2)**0.5
65.
66.     u1 = p1.copy()
67.     u2 = p2.copy()
68.
69.     p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin +
70.     u1[3]*sin*cos)
71.     p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos -
72.     u2[3]*sin*sin))
73.     p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin +
74.     u2[3]*sin*cos)
75.     p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos -
76.     u1[3]*sin*sin))
77.     return 0
78.
79. e = 1.0 # elastic collision
80. d = 5.0e-02 # (radius * 2) of particle
81. h = 0.01 # time interval
82. n = 10 # count of particles
83.
84. ## Calculate total Kinetic Energy
85. def energy(x):
86.     E_tot = 0
87.     for i in range(len(x)):
88.         E_tot = E_tot + (x[i][1]**2 + x[i][3]**2)/2
89.     return E_tot
90.
91. ## Initial condition of particles
92. from random import random
93. x_init = []
94. for i in range(n):
95.     x_init.append(np.array([(random()-0.5)*4,(random()-
96.     0.5)*4,(random()-0.5)*4,(random()-0.5)*4]))
97.
98. ## Plot Energy graph
99. ## e = 1.0, 0.75, 0.5, 0.25
100. legend = []
101. for i in [1,0.75,0.5,0.25]:
102.     e = i
103.     legend.append('e = '+str(e))
104.     x = x_init.copy()
105.     ts, xs, Es = integrate(F,0.0,x,100.0,h)
106.     plt.plot(ts,Es,'-')
107.
108. plt.legend(legend)
109. plt.grid()
110. plt.savefig('n-body_energy.png')
111. plt.close()
112.
113. ## Plot orbitals of particles graph

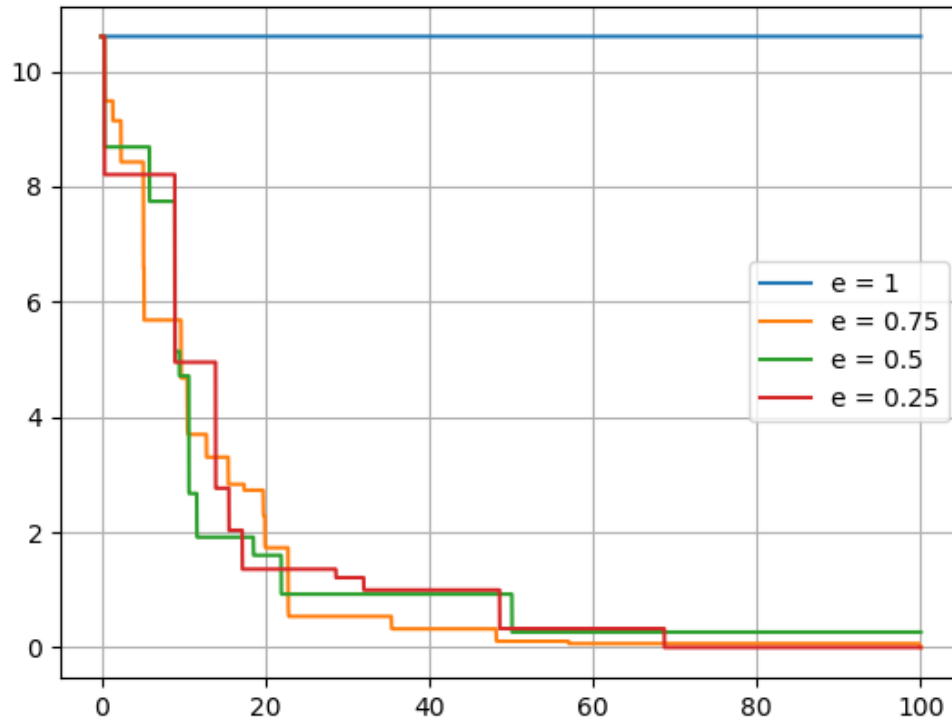
```

```

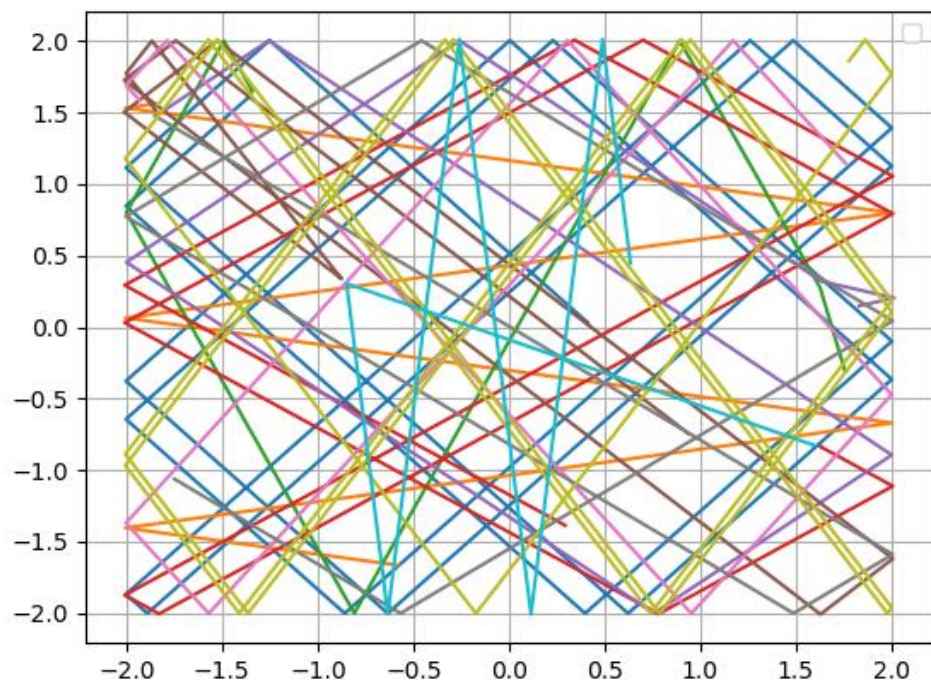
110. ## Elastic collision, e = 1
111. legend = []
112. for i in range(len(xs)):
113.     plt.plot(xs[i][:,0],xs[i][:,2],'-')
114.     if range(len(xs)<6): legend.append('particle '+str(i))
115.
116. plt.legend(legend)
117. plt.grid()
118. plt.savefig('n-body_orbit.png')

```

n-body_energy.png



n-body_orbit.png



2-D collision function in integrate

```
1. ## 2-D collision
2. ## e is coefficient of restitution
3. ## p = [ x, x', y, y']
4. def collision(p1,p2,e):
5.     tan = (p2[2] - p1[2])/(p2[0] - p1[0])
6.     cos = 1/(tan**2+1)**0.5
7.     sin = (1-cos**2)**0.5
8.
9.     u1 = p1.copy()
10.    u2 = p2.copy()
11.
12.    p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin +
13.               u1[3]*sin*cos)
14.    p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos -
15.               u2[3]*sin*sin))
16.    p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin +
17.               u2[3]*sin*cos)
18.    p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos -
19.               u1[3]*sin*sin))
20.    return 0
```

use rotate matrix to calculate.

Visualizing 2-D collision. (OpenGL)

```
1. import numpy as np
2. from time import time
3. from sys import exit
4.
5. ## distance of 2-D
6. ## x = [ x, x', y, y']
7. ## distance(particle 1,particle 2) = ( (x1-x2)^2 + (y1-y2)^2 )^0.5
8. distance = lambda x1,x2:((x1[0]-x2[0])**2+(x1[2]-x2[2])**2)**0.5
9.
10. ## x = [ x, x', y, y']
11. ## F = dx/dt = [ v_x, a_x, v_y, a_y ]
12. def F(t,x):
13.     g = -9.8 # m/sec^2
14.     F = np.zeros(4)
15.     F[0] = x[1]
16.     F[1] = 0
17.     F[2] = x[3]
18.     F[3] = 0
19.     return F
20.
21. ## runge-kutta methon in many body problem
22. def integrate(F,t,x,h):
23.     def dx(F,t,x,h):
24.         k0 = h*F(t,x)
25.         k1 = h*F(t+h/2.0,x+k0/2.0)
26.         k2 = h*F(t+h/2.0,x+k1/2.0)
27.         k3 = h*F(t+h,x+k2)
28.         return (k0 + 2.0*k1 + 2.0*k2 + k3)/6.0
29.     n = len(x_init)
30.     # collision test
31.     for i in range(n):
32.         if x[i][0] < -2 or x[i][0] > 2:
33.             x[i][1] = -x[i][1]
34.             if x[i][0] < -2: x[i][0] = -2
35.             else: x[i][0] = 2
36.         if x[i][2] < -2 or x[i][2] > 2:
37.             x[i][3] = -x[i][3]
38.             if x[i][2] < -2: x[i][2] = -2
39.             else: x[i][2] = 2
40.         for j in range(i+1,n):
41.             if distance(x[i],x[j]) < d:
42.                 collision(x[i],x[j],e)
43.     for i in range(n):
44.         x[i] = x[i] + dx(F,t,x[i],h)
45.
46. ## 2-D collision
47. ## e is coefficient of restitution
48. ## p = [ x, x', y, y']
49. def collision(p1,p2,e):
50.     tan = (p2[2] - p1[2])/(p2[0] - p1[0])
51.     cos = 1/(tan**2+1)**0.5
52.     sin = (1-cos**2)**0.5
53.
54.     u1 = p1.copy()
55.     u2 = p2.copy()
56.
```

```

57.     p1[1] = e*((u2[1]*cos*cos - u2[3]*sin*cos) + u1[1]*sin*sin +
    u1[3]*sin*cos)
58.     p1[3] = e*(u1[1]*sin*cos + u1[3]*cos*cos - (u2[1]*sin*cos -
    u2[3]*sin*sin))
59.
60.     p2[1] = e*((u1[1]*cos*cos - u1[3]*sin*cos) + u2[1]*sin*sin +
    u2[3]*sin*cos)
61.     p2[3] = e*(u2[1]*sin*cos + u2[3]*cos*cos - (u1[1]*sin*cos -
    u1[3]*sin*sin))
62.     return 0
63.
64. e = 1.0 # elastic collision
65. d = 5.0e-02
66. h = 0.01
67.
68. def energy(x):
69.     E_tot = 0
70.     for i in range(len(x)):
71.         E_tot = E_tot + (x[i][1]**2 + x[i][3]**2)/2
72.     return E_tot
73.
74. n = 100
75. from random import random
76. x_init = []
77. for i in range(n):
78.     x_init.append(np.array([(random()-0.5)*3.9,(random()-
    0.5)*2,(random()-0.5)*3.9,(random()-0.5)*2]))
79. x = x_init.copy()
80.
81. ts = [0]
82. Es = [energy(x)]
83.
84. def changeSize(w,h):
85.     if h==0: h=1
86.     ratio = w/h
87.
88.     glMatrixMode(GL_PROJECTION)
89.     glLoadIdentity()
90.
91.     glViewport(0,0,w,h)
92.
93.     gluPerspective(45,ratio,0.1,1000)
94.     glMatrixMode(GL_MODELVIEW)
95.     glLoadIdentity()
96.     gluLookAt(0.0,0.0,10.0,0.0,0.0,-1.0,0.0,1.0,0.0)
97.
98. def drawSphere(x,y,z):
99.     glTranslatef(x,y,z)
100.    glutSolidSphere(d/2,10,10)
101.    glTranslatef(-x,-y,-z)
102.
103. time1 = time(); time2 = time(); t = 0; ite = 0
104. def renderScene():
105.     global time1, time2, t, ite, x
106.     ite = ite + 1
107.     glClearColor(0,0,0,0)
108.     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
109.     glPushMatrix()

```

```

110.
111.     for i in range(len(x)):
112.         glColor3f(0.1,0.1,0.1)
113.         drawSphere(x[i][0],x[i][2],0)
114.
115.     glColor3f(0.9,0.9,0.9)
116.     glBegin(GL_QUADS)
117.     glVertex3f(2,2,0)
118.     glVertex3f(-2,2,0)
119.     glVertex3f(-2,-2,0)
120.     glVertex3f(2,-2,0)
121.     glEnd()
122.
123.     glColor3f(0,1,0)
124.     drawSphere(2,2,0)
125.
126.     glColor3f(0,0,0)
127.     drawSphere(-2,2,0)
128.
129.     glColor3f(1,0,0)
130.     drawSphere(-2,-2,0)
131.
132.     glColor3f(0,0,1)
133.     drawSphere(2,-2,0)
134.
135.     glPopMatrix()
136.     dt = time2 - time1
137.     glutSwapBuffers()
138.     if ite > 1:
139.         integrate(F,t,x,dt)
140.         t = t + dt
141.         E = energy(x)
142.         Es.append(E)
143.         ts.append(t)
144.     time1 = time2
145.     time2 = time()
146.
147. from OpenGL._bytes import as_8_bit
148. ESC = as_8_bit('\033')
149. def exitKey(key,x,y):
150.     if key == ESC:
151.         exit()
152.     if key == as_8_bit('m'):
153.         import matplotlib.pyplot as plt
154.         plt.plot(ts,Es,'-')
155.         plt.grid()
156.         plt.savefig('n-body_'+str(ite)+'.png')
157.
158. ## GLUT animation in main function
159. from OpenGL.GL import *
160. from OpenGL.GLU import *
161. from OpenGL.GLUT import *
162.
163. if __name__ == '__main__':
164.     glutInit()
165.     glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA)
166.     glutInitWindowPosition(0,30)
167.     glutInitWindowSize(1280,720)

```

```
168.    glutCreateWindow( 'n-body problem' )
169.    glutDisplayFunc( renderScene )
170.
171.    glutIdleFunc( renderScene )
172.
173.    glutReshapeFunc( changeSize )
174.
175.    glutKeyboardFunc( exitKey )
176.
177.    glEnable( GL_DEPTH_TEST )
178.    glutMainLoop( )
```