

# Homework 1

108000204

Yuan-Yen Peng

Dept. of Physics, NTHU

Hsinchu, Taiwan

October 15, 2022

## 1 Programming Assignments

### 1.1 $\pi$ calculation

In this section, we redo the  $\pi$  calculation in the class and run the code with  $N = 10000$ ,  $100000$ , and  $1000000$ . Then, we use the `%timeit` command in a jupyter notebook to evaluate the performance. Table1 is our result with 3 different methods. The first is to use hand write for-loop; the second is to use the default sum with numpy arrays; the last is to use the numpy sum with numpy arrays. On the other hand, we use  $N = 10E3$  to plot the tendency of each methods; also calculate the performance of plotting set value points, i.e.,  $N = 1, 2 \dots N$  with Algorithm1.

Algorithm 1: Performance plotting, Three of mehod use the same plotting approach.

---

```

1  # cal_meth1 means use method 1
2  t1 = time.time()
3  X = np.linspace(1, N, N)
4  Y = np.array([])
5  for i in range (0, N):
6      Y = np.append(Y, cal_meth1(int(X[i])))
7  t2 = time.time()
8
9  print("time different = ", [t2 - t1])
10 print(f"The value is {Y[-1]}")

```

---

N	method 1	method 2	method 3
10E4	49.8ms $\pm$ 805 $\mu$ s	50.8ms $\pm$ 1.06 $\mu$ s	49.7ms $\pm$ 721 $\mu$ s
10E5	493ms $\pm$ 6.05ms	502ms $\pm$ 3.69ms	485ms $\pm$ 6.9ms
10E6	5.01s $\pm$ 62.4ms	5.02s $\pm$ 99.9ms	4.9s $\pm$ 85ms

---

Table 1: Three different methods for the  $\pi$  calculation

---

Algorithm 2: Method 1, using default for loop.

---

```
1  for i in range (0, N + 1, 1):
2      h = np.sqrt(1 - (i/N)**2)
3      area += dx * h
4
5  Area = area * 4
```

---

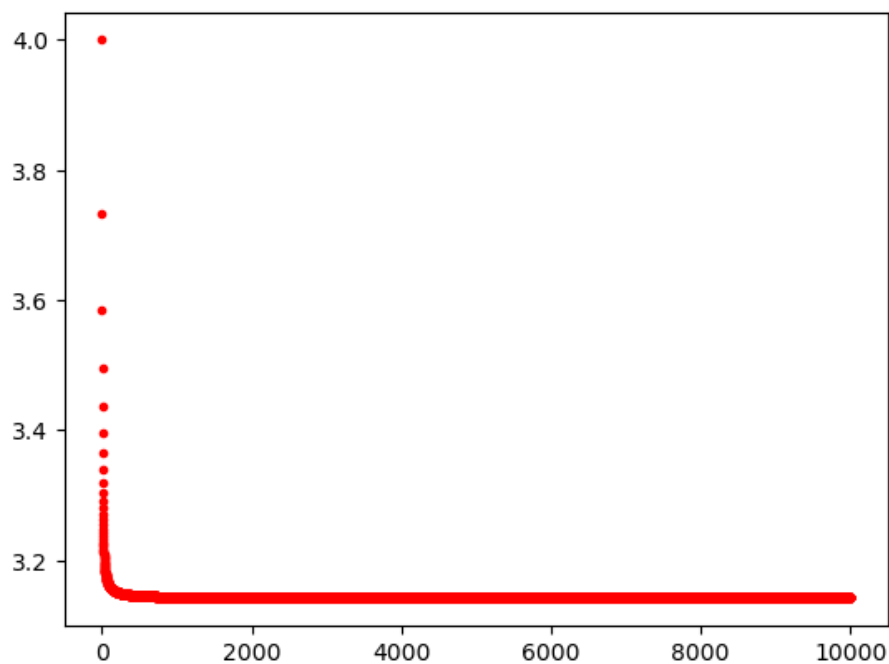


Figure 1: Method 1, the performance is approximately 24.22 seconds; the  $\pi$  is approaching to the stable value: 3.141791477611317.

---

Algorithm 3: Method 2, using default sum.

---

```
1  x = np.linspace(0, 1, N)
2  y = np.sqrt(1 - x**2)
3  area = sum(dx*y)
4
5  Area = area * 4
```

---

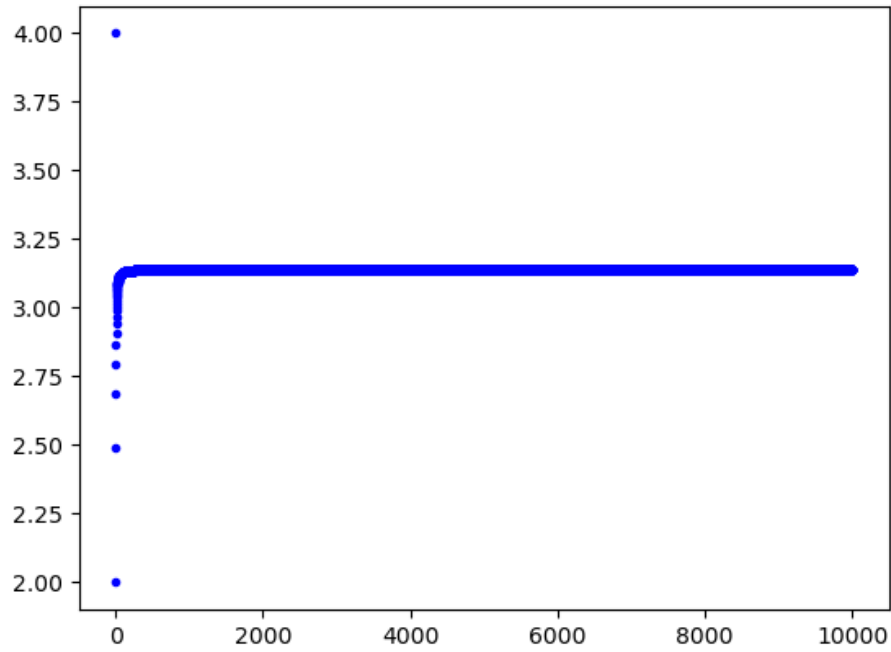


Figure 2: Method 2, the performance is approximately 0.31 seconds; the  $\pi$  is approaching to the stable value: 3.1414773182871603.

Algorithm 4: Method 1, using `np.sum`.

---

```

1  x = np.linspace(0, 1, N)
2  y = np.sqrt(1 - x**2)
3  area = np.sum(dx*y)
4
5  Area = area * 4

```

---

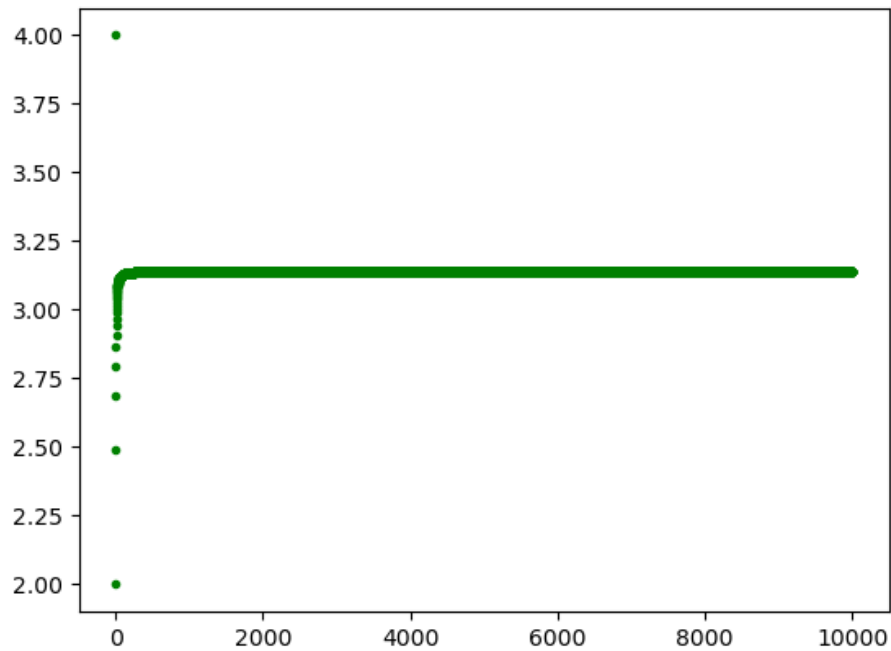


Figure 3: Method 3, the performance is approximately 2.40 seconds; the  $\pi$  is approaching to the stable value: 3.1414773182871647.

For results in Table1, with  $10E4$ , method 2 is the slowest, and method 3 is faster than the others, so we can find that a for-loop for python is slow while it is still faster than the default sum in python; moreover, if using the sum with numpy, the performance of the code will be accelerated. This phenomenon is much more obvious when the  $N$  increases, meeting our expectations. In the last class, we have also learned the performance plotting; thus, I use  $N = 10E3$  to calculate the performance and also plot it in method 1: figure1, method 2: figure2, method 3: figure3. The performance strongly depends on what methods we use to plot. If we use `numpy.sum` it will be the fastest among the three of them, and if using the for-loop, it will be the slowest, which is not the same as the outcomes of using `%timeit`. I think that it might depend on how many for-loops we used; that is, when we plot the performance utilizing two for-loops which will degrade the speed much more manifest than taking the default sum for only one for-loop. All in all, to summarize the consequences, using a high-speed computing kit, numpy, will significantly speed up our computational performances.

## 1.2 Stefan-Boltzmann constant $\sigma_B$

The method is similar to the  $\pi$  calculation. We use the implementation of `numpy.sum` so as to upgrade or speed up computation. In order to avoid the denominator going to zeros leading to the result bursting into infinity, we put in the additional "tolerance" such that the code can evade this erroneous. According to the hint from the question, we set the upper bound to the  $10E15$ , as large as we can; lastly get the reasonable result:  $\sigma_B = 5.6703687488100114E - 08$ , comparing to the theoretical result:  $\sigma_B = 5.67037442E - 08$ .

---

Algorithm 5: The Stefan-Boltzmann constant  $\sigma_B$  calculation, performance =  $15.6ms \pm 312\mu s$ .

---

```

1  def sigma(N, upper):
2      '''
3      :param N: divided number.
4      :upper: upper bound for infinity, so set it as large as possible.
5      '''
6      tor_denu = 10E-10
7      # lower tolerance math error.
8      dNu = upper/N
9      Nu = np.linspace(0, upper, N)
10     B_nu = 2 * h * Nu**3
11     B_denu = c**2 * (np.exp(h * Nu / (k * T)) - 1) + tor_denu
12     # lower tolerance (+ 10E-10) to avoid 1/0 (math error).
13     sig = np.sum(B_nu * dNu / B_denu) * pi/T**4
14
15     return sig
16
17     N = int(10E5)
18     upper = int(10E15)
19     # as large as we can, because it need to be inf.
20     print(sigma(N = N, upper = upper))
21     %timeit sigma(N, upper)

```

---

## 1.3 Angry Brid

We set the parameters with playground length  $100[m]$ , initial bird's position at  $(0.5[m], 0.5[m])$ , the pig's position at  $(20[m], 1[m])$ , and regarding the bird as a circle(2D) and also the pig with radius 0.3 and 0.5 respectively. The method we use is that using another variable, temp, to store the information from the last step. We, afterward, utilize temp to calculate the information of the current step and

store them in the specified array. The definition of "success" is within the tolerance of the distance between the target and the bird. We, eventually, get the results in the different mediators showing in the Figure4, 5, 6, 7. Although the outcome is physically correct, the visualizations of the bird(red) and pig(green) are not rendered as the "true" size. Due to the sophisticated settings of `plt.scatter`, I cannot plot the real visualization's results.

---

Algorithm 6: This is the algorithm of the simulations of Angry Bird with different circumstances.

---

```

1  def tr(PosX, PosY, Vel, theta, eta):
2      VelX = Vel * np.sin(theta)
3      VelY = Vel * np.cos(theta)
4      K = 6 * np.pi * R
5      x, y, vx, vy = PosX, PosY, VelX, VelY # temp
6      ax = -K * eta * vx # temp
7      ay = -K * eta * vy # temp
8      X = np.array([x])
9      Y = np.array([y])
10     VX = np.array([vx])
11     VY = np.array([vy])
12     AX = np.array([ax])
13     AY = np.array([ay])
14     while (x <= D_x and y >= D_y):
15         vy += (-g + ay) * dt # temp
16         VY = np.append(VY, vy)
17         vx += ax * dt # temp
18         VX = np.append(VX, vx)
19         ax = -K * eta * vx # temp
20         AX = np.append(AX, ax)
21         ay = -K * eta * vy # temp
22         AY = np.append(AY, ay)
23         x += vx * dt # temp
24         X = np.append(X, x)
25         y += vy * dt # temp
26         Y = np.append(Y, y)
27
28         if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <= PosSec):
29             print('Booom!')
30             break
31
32     return [X, Y, x, y]
```

---

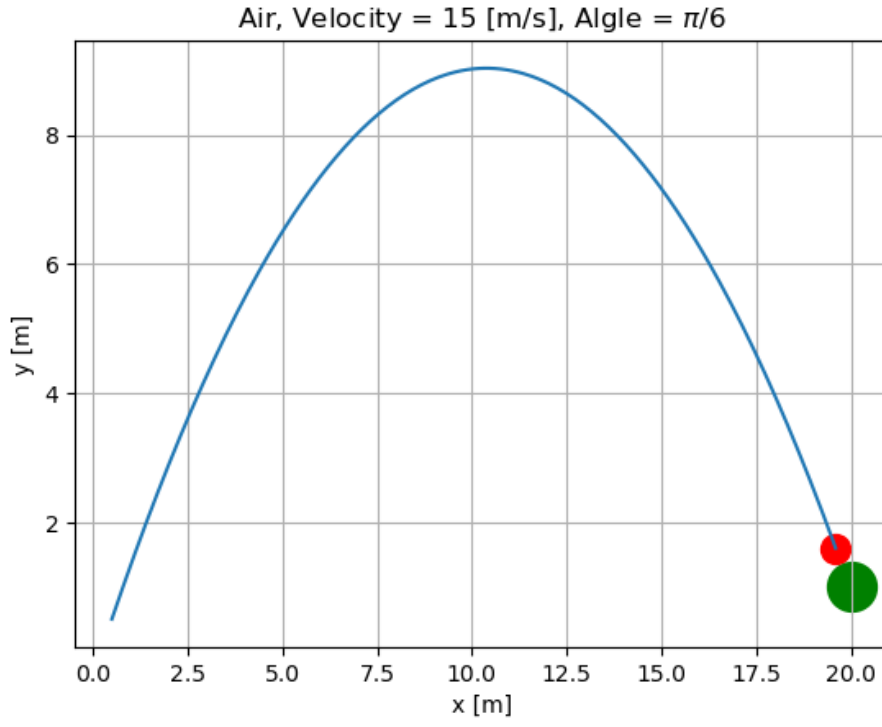


Figure 4: If the bird is in the air with  $\eta = 2E - 4[mks unit]$ , we can get the successful event with initial speed and angle  $(15, \pi/6)[mks unit]$ .

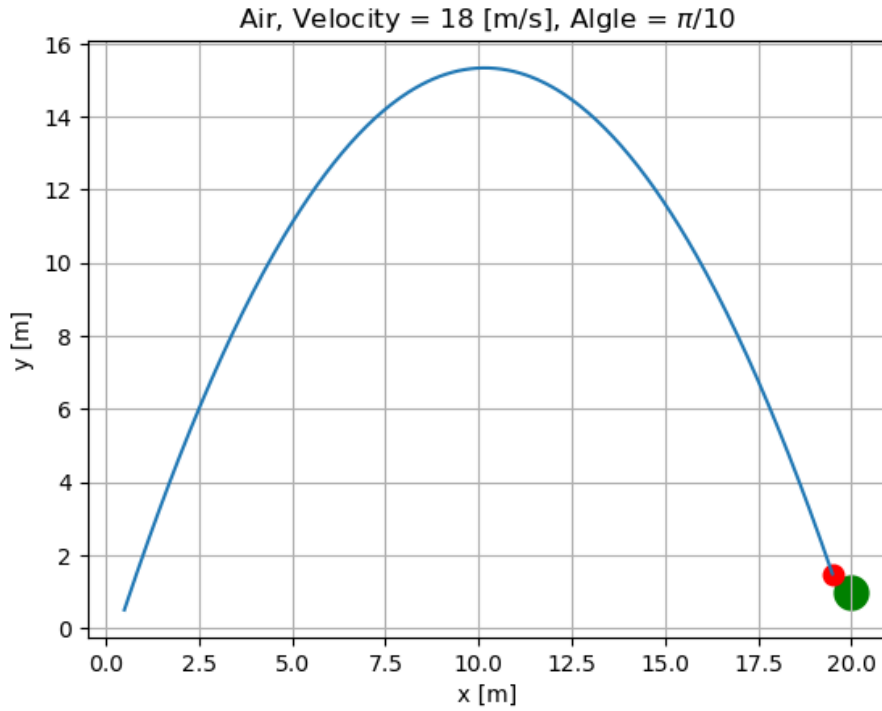


Figure 5: If the bird is in the air with  $\eta = 2E - 4[mks unit]$ , we can get the successful event with initial speed and angle  $(18, \pi/10)[mks unit]$ .

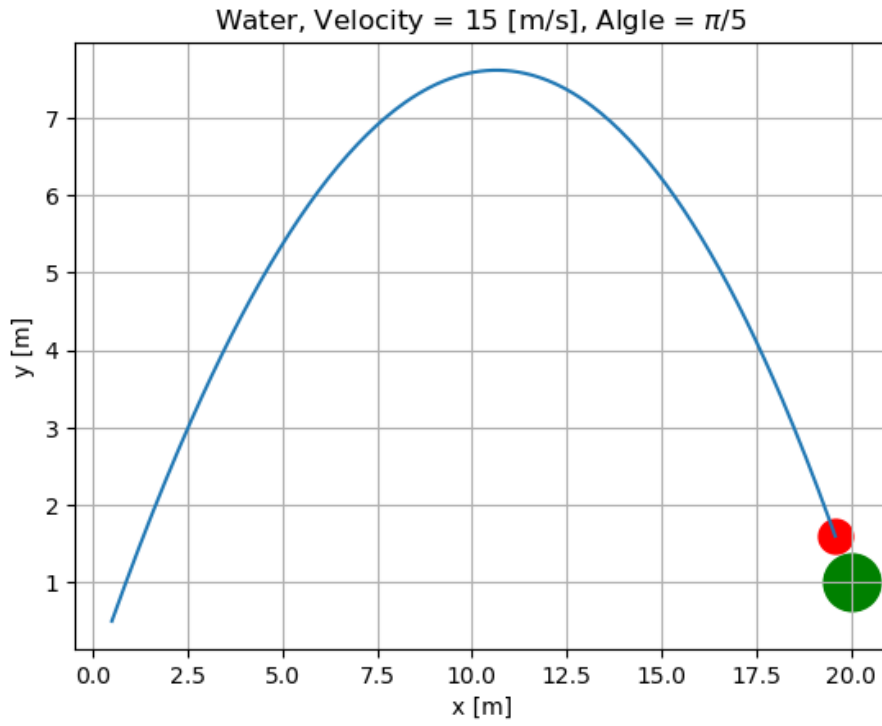


Figure 6: If the bird is in the water with  $\eta = 0.01[mks\ unit]$ , we can get the successful event with initial speed and angle  $(15, \pi/5)[mks\ unit]$ .

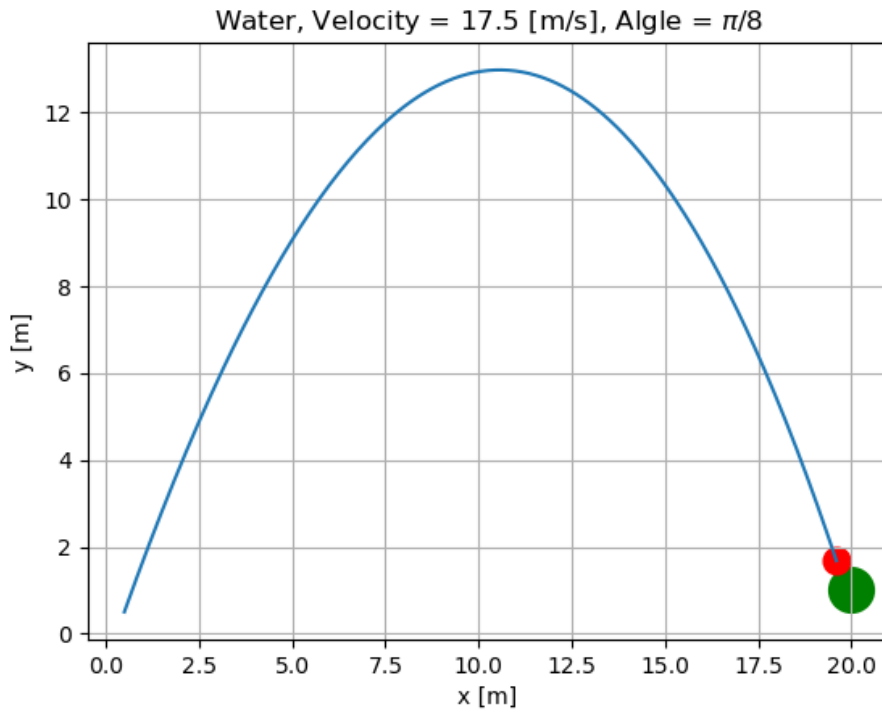


Figure 7: If the bird is in the water with  $\eta = 0.01[mks\ unit]$ , we can get the successful event with initial speed and angle  $(17.5, \pi/8)[mks\ unit]$ .

## 2 Codes

All the codes are transferred from jupyterlab; hence, if you want to re-run them, please see the source code in the attached files or my GitHub repository: <<https://github.com/gary20000915/Comphyslab-HW1.git>>.

### 2.1 $\pi$ calculation

---

```
1      # %% [markdown]
2      # ## Computational Physics Lab
3      # ### Homework 1.1
4      # Yuan-Yen Peng
5      # Dept of Physics, NTHU, Taiwan
6      # October 17, 2022
7
8      # %% [markdown]
9      # calculation
10
11     # %%
12     import numpy as np
13
14     N1 = int(10E4)
15     N2 = int(10E5)
16     N3 = int(10E6)
17     # Set how many small rectangles. (divided number)
18
19     # %%
20     # method 1, use hand writing for-loop.
21     def cal_meth1(N: int):
22         dx = 1/N
23         area = 0
24
25         for i in range (0, N + 1, 1):
26             h = np.sqrt(1 - (i/N)**2)
27             area += dx * h
28         Area = area * 4
29
30         return Area
31
32     print("pi of N = 10000: ", cal_meth1(N1))
33     %timeit cal_meth1(N1)
34     print("pi of N = 100000: ", cal_meth1(N2))
35     %timeit cal_meth1(N2)
36     print("pi of N = 1000000: ", cal_meth1(N3))
37     %timeit cal_meth1(N3)
38
39     # %%
40     # method 2, use default sum.
41     def cal_meth2(N: int):
42         dx = 1/N
43         area = 0
44
```



```

45     x = np.linspace(0, 1, N)
46     y = np.sqrt(1 - x**2)
47     area = sum(dx*y)
48     Area = area * 4
49
50     return Area
51
52     print("pi of N = 10000: ", cal_meth2(N1))
53     %timeit cal_meth1(N1)
54     print("pi of N = 100000: ", cal_meth2(N2))
55     %timeit cal_meth1(N2)
56     print("pi of N = 1000000: ", cal_meth2(N3))
57     %timeit cal_meth1(N3)
58
59     # %%
60     # method 3, use numpy.sum.
61     def cal_meth3(N: int):
62         dx = 1/N
63         area = 0
64
65         x = np.linspace(0, 1, N)
66         y = np.sqrt(1 - x**2)
67         area = np.sum(dx*y)
68         Area = area * 4
69
70         return Area
71
72     print("pi of N = 10000: ", cal_meth3(N1))
73     %timeit cal_meth1(N1)
74     print("pi of N = 100000: ", cal_meth3(N2))
75     %timeit cal_meth1(N2)
76     print("pi of N = 1000000: ", cal_meth3(N3))
77     %timeit cal_meth1(N3)

```

---

The performance code is in the below.

---

```

1     # %% [markdown]
2     # ## Computational Physics Lab
3     # Yuan-Yen Peng
4     # Dept of Physics, NTHU, Taiwan
5
6     # %%
7     import numpy as np
8     import matplotlib.pyplot as plt
9     import time
10
11     N = int(10E3)
12
13     # %%
14     # method 1, use hand writing for-loop.
15     def cal_meth1(N: int):
16         dx = 1/N
17         area = 0
18

```

```

19     for i in range (0, N + 1, 1):
20         h = np.sqrt(1 - (i/N)**2)
21         area += dx * h
22     Area = area * 4
23
24     return Area
25
26
27 t1 = time.time()
28 X = np.linspace(1, N, N)
29 Y = np.array([])
30 for i in range (0, N):
31     Y = np.append(Y, cal_meth1(int(X[i])))
32 t2 = time.time()
33
34 print("time different = ", [t2 - t1])
35 print(f"The value is {Y[-1]}")
36 plt.plot(X, Y, ".", color = "r", label = "N = 10E3")
37
38 # %%
39 # method 2, use default sum.
40 def cal_meth2(N: int):
41     dx = 1/N
42     area = 0
43
44     x = np.linspace(0, 1, N)
45     y = np.sqrt(1 - x**2)
46     area = sum(dx*y)
47     Area = area * 4
48
49     return Area
50
51 t1 = time.time()
52 X = np.linspace(1, N, N)
53 Y = np.array([])
54 for i in range (0, N):
55     Y = np.append(Y, cal_meth2(int(X[i])))
56 t2 = time.time()
57
58 print("time different = ", [t2 - t1])
59 print(f"The value is {Y[-1]}")
60 plt.plot(X, Y, ".", color = "g", label = "N = 10E3")
61
62 # %%
63 # method 3, use numpy.sum.
64 def cal_meth3(N: int):
65     dx = 1/N
66     area = 0
67
68     x = np.linspace(0, 1, N)
69     y = np.sqrt(1 - x**2)
70     area = np.sum(dx*y)
71     Area = area * 4

```

```

72
73     return Area
74
75 t1 = time.time()
76 X = np.linspace(1, N, N)
77 Y = np.array([])
78 for i in range (0, N):
79     Y = np.append(Y, cal_meth3(int(X[i])))
80 t2 = time.time()
81
82 print("time different = ", [t2 - t1])
83 print(f"The value is {Y[-1]}")
84 plt.plot(X, Y, ".", color = "b", label = "N = 10E3")

```

---

## 2.2 Stefan-Boltzmann constant $\sigma_B$

---

```

1     # %% [markdown]
2     # ## Computational Physics Lab
3     # ### Homework 1.2
4     # Yuan-Yen Peng
5     # Dept of Physics, NTHU, Taiwan
6     # October 17, 2022
7
8     # %% [markdown]
9     # The Stefan-Boltzmann constant  $\sigma_B$  calculation
10
11     # %%
12     %reset -f
13     # clear previous variables
14
15     import numpy as np
16     import scipy.constants as const
17
18     pi = const.pi
19     h = const.h
20     c = const.c
21     k = const.k
22     T = 6000
23     # set constants
24
25     # %%
26     def sigma(N, upper):
27         '''
28         :param N: divided number.
29         :upper: upper bound for infinity, so set it as large as possible.
30         '''
31         tor_denu = 10E-10
32         # lower tolerance math error.
33         dNu = upper/N
34         Nu = np.linspace(0, upper, N)
35         B_nu = 2 * h * Nu**3
36         B_denu = c**2 * (np.exp(h * Nu / (k * T)) - 1) + tor_denu

```

```

37     # lower tolerance (+ 10E-10) to avoid 1/0 (math error).
38     sig = np.sum(B_nu * dNu / B_denu) * pi/T**4
39
40     return sig
41
42 N = int(10E5)
43 upper = int(10E15)
44 # as large as we can, because it need to be inf.
45 print(sigma(N = N, upper = upper))
46 %timeit sigma(N, upper)

```

---

## 2.3 Angry Bird

---

```

1     # %% [markdown]
2     # ## Computational Physics Lab
3     # ### Homework 1.3
4     # Yuan-Yen Peng
5     # Dept of Physics, NTHU, Taiwan
6     # October 17, 2022
7
8     # %% [markdown]
9     # Angry Bird
10
11     # %%
12     import numpy as np
13     import scipy as sp
14     import scipy.constants as const
15     import matplotlib.pyplot as plt
16
17     # %%
18     dt = 0.01
19     g = const.g
20     tor = 0.03 # (+ 0.05 --> tolerance)
21     mass = 5 # kg
22     R = 0.3 # bird radius [meter]
23     R_T = 0.5 # target radius [meter]
24     # bird's position
25     PosX = 0.5
26     PosY = 0.5
27     # The size of the playground
28     D_x = 100
29     D_y = 0 + R + tor
30     # Target position [meter, meter]
31     T = [20, 1]
32     # collide parameter
33     PosSec = R + R_T + tor
34     # specify the point size
35     s_bird = np.square(40 * R) # the point area (cm^2)
36     s_pig = np.square(40 * R_T) # the point area (cm^2)
37
38     # %% [markdown]
39     # Normal condition

```

```

40
41 # %%
42 def tr(PosX, PosY, Vel, theta):
43     VelX = Vel * np.sin(theta)
44     VelY = Vel * np.cos(theta)
45     x, y, vx, vy = PosX, PosY, VelX, VelY # temp
46     X = np.array([x])
47     Y = np.array([y])
48     VX = np.array([vx])
49     VY = np.array([vy])
50     while (x <= D_x and y >= D_y):
51         vy += -g * dt # temp
52         VY = np.append(VY, vy)
53         vx = vx # temp
54         VX = np.append(VX, vx)
55         x += vx * dt # temp
56         X = np.append(X, x)
57         y += vy * dt # temp
58         Y = np.append(Y, y)
59         if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <=
60             PosSec):
61             print('Booom!')
62             break
63
64     return [X, Y, x, y]
65
66 # %%
67 # bird's velocity and angle
68 Vel = 15
69 theta = np.pi/6
70
71 plt.scatter(T[0], T[1], s_pig, c = 'g')
72 plt.scatter(tr(PosX, PosY, Vel, theta)[2], tr(PosX, PosY, Vel, theta)[3],
73             s_bird, c = 'r')
74 plt.grid()
75 plt.plot(tr(PosX, PosY, Vel, theta)[0],
76          tr(PosX, PosY, Vel, theta)[1])
77
78 # %% [markdown]
79 # Add the drag force, air and water respectively.
80
81 # %%
82 def tr(PosX, PosY, Vel, theta, eta):
83     VelX = Vel * np.sin(theta)
84     VelY = Vel * np.cos(theta)
85     K = 6 * np.pi * R
86     x, y, vx, vy = PosX, PosY, VelX, VelY # temp
87     ax = -K * eta * vx # temp
88     ay = -K * eta * vy # temp
89     X = np.array([x])
90     Y = np.array([y])
91     VX = np.array([vx])

```

```

91     VY = np.array([vy])
92     AX = np.array([ax])
93     AY = np.array([ay])
94     while (x <= D_x and y >= D_y):
95         vy += (-g + ay) * dt # temp
96         VY = np.append(VY, vy)
97         vx += ax * dt # temp
98         VX = np.append(VX, vx)
99         ax = -K * eta * vx # temp
100        AX = np.append(AX, ax)
101        ay = -K * eta * vy # temp
102        AY = np.append(AY, ay)
103        x += vx * dt # temp
104        X = np.append(X, x)
105        y += vy * dt # temp
106        Y = np.append(Y, y)
107
108        if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <=
109            PosSec):
110            print('Booom!')
111            break
112
113    return [X, Y, x, y]
114
115    # %%
116    # in air
117
118    # bird's velocity and angle
119    Vel = 18
120    n = 10
121    theta = np.pi/n
122
123    eta_air = 2E-4
124    tr_air = tr(PosX, PosY, Vel, theta, eta_air)
125    plt.scatter(T[0], T[1], s_pig, c = 'g')
126    plt.scatter(tr_air[2], tr_air[3], s_bird, c = 'r')
127    plt.grid()
128    plt.plot(tr_air[0], tr_air[1])
129    plt.title(f'Air, Velocity = {Vel} [m/s], Angle = $\pi$/n')
130    plt.xlabel('x [m]')
131    plt.ylabel('y [m]')
132
133    # %%
134    # in water
135
136    # bird's velocity and angle
137    Vel = 17.5
138    n = 8
139    theta = np.pi/n
140
141    eta_water = 1E-2
142    tr_water = tr(PosX, PosY, Vel, theta, eta_water)
143    plt.scatter(T[0], T[1], s_pig, c = 'g')

```

```
143     plt.scatter(tr_water[2], tr_water[3], s_bird, c = 'r')
144     plt.grid()
145     plt.plot(tr_water[0], tr_water[1])
146     plt.title(f'Water, Velocity = {Vel} [m/s], Algle =  $\pi/{n}$ ')
147     plt.xlabel('x [m]')
148     plt.ylabel('y [m]')
```

---