Computational Physics Lab

Homework 1

108000204
Yuan-Yen Peng
Dept. of Physics, NTHU
Hsinchu, Taiwan

October 15, 2022

1 Programming Assignments

1.1 π calculation

In this section, we redo the π 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., $N = 1, 2 \ldots N$ with Algorithm1.

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

```
# cal_meth1 means use method 1
t1 = time.time()
X = np.linspace(1, N, N)
Y = np.array([])
for i in range (0, N):
  Y = np.append(Y, cal_meth1(int(X[i]))
t2 = time.time()

print("time different = ", [t2 - t1])
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 π calculation

Algorithm 2: Method 1, using default for loop.

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

Area = area * 4
```

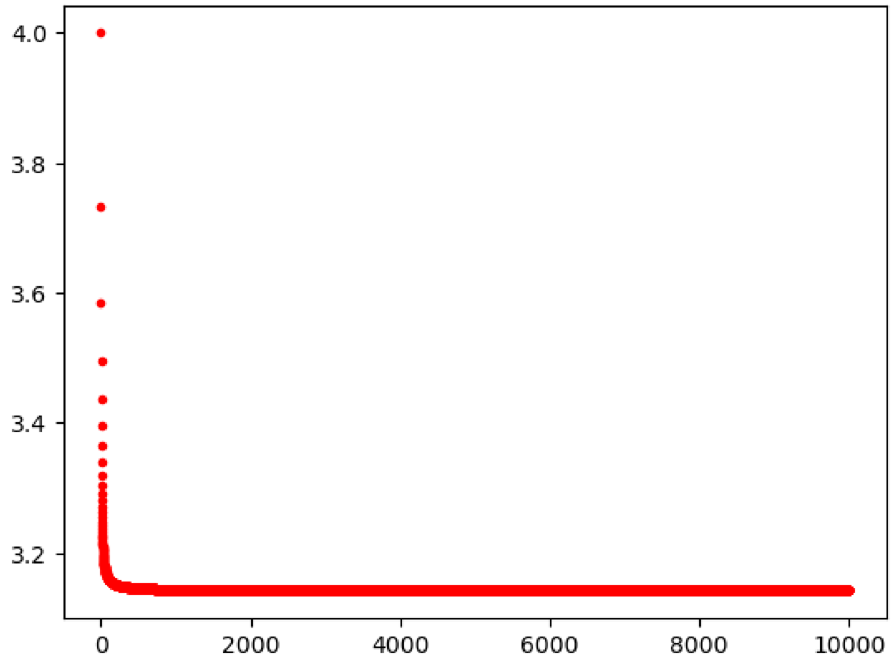

Figure 1: Method 1, the performance is approximately 24.22 seconds; the π is approaching to the stable value: 3.141791477611317.

Algorithm 3: Method 2, using default sum.

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

Area = area * 4
```

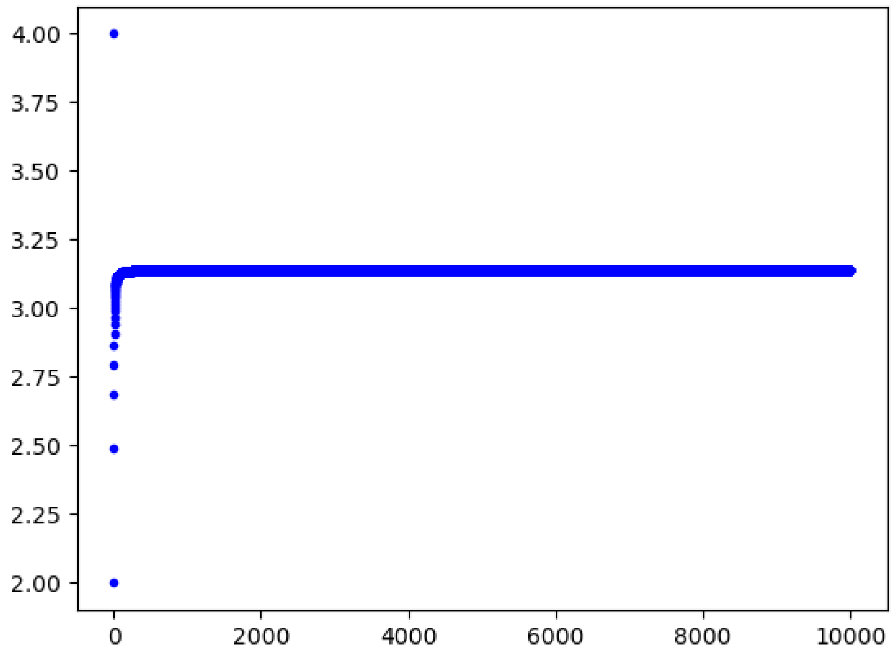

Figure 2: Method 2, the performance is approximately 0.31 seconds; the π is approaching to the stable value: 3.1414773182871603.

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

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

Area = area * 4
```

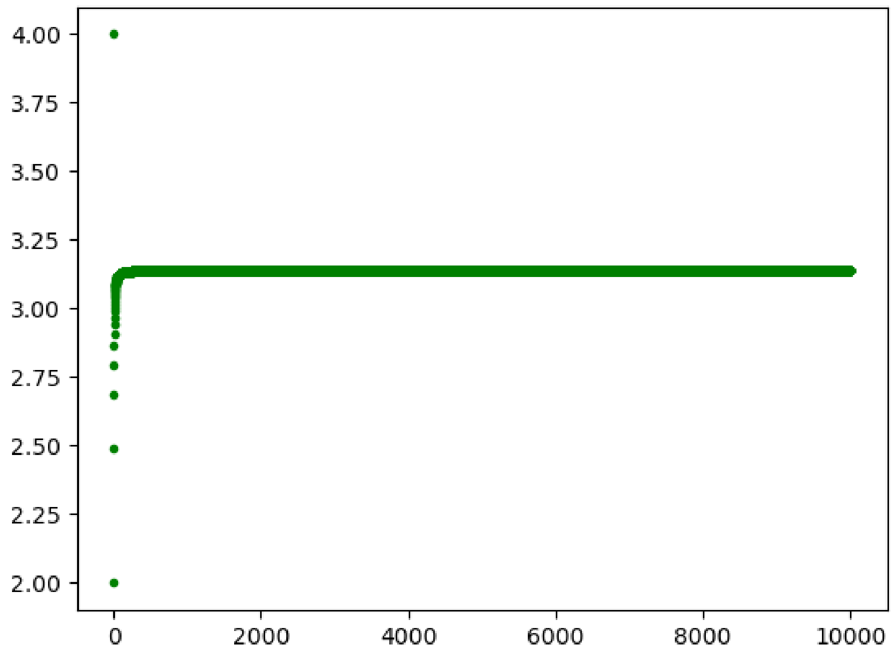

Figure 3: Method 3, the performance is approximately 2.40 seconds; the π is approaching to the stable value: 3.1414773182871647.

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 σ_B

The method is similar to the π 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 σ_B calculation, performance = $15.6ms \pm 312\mu s$.

```
    def sigma(N, upper):
        '''
        :param N: divided number.
        :upper: upper bound for infinity, so set it as large as possible.
        '''
        tor_denu = 10E-10
        # lower tolerance math error.
        dNu = upper/N
        Nu = np.linspace(0, upper, N)
        B_nu = 2 * h * Nu**3
        B_denu = c**2 * (np.exp(h * Nu / (k * T)) - 1) + tor_denu
        # lower tolerance (+ 10E-10) to avoid 1/0 (math error).
        sig = np.sum(B_nu * dNu / B_denu) * pi/T**4

        return sig

    N = int(10E5)
    upper = int(10E15)
    # as large as we can, because it need to be inf.
    print(sigma(N = N, upper = upper))
    %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.

```
def tr(PosX, PosY, Vel, theta, eta):
    VelX = Vel * np.sin(theta)
    VelY = Vel * np.cos(theta)
    K = 6 * np.pi * R
    x, y, vx, vy = PosX, PosY, VelX, VelY # temp
    ax = -K * eta * vx # temp
    ay = -K * eta * vy # temp
    X = np.array([x])
    Y = np.array([y])
    VX = np.array([vx])
    VY = np.array([vy])
    AX = np.array([ax])
    AY = np.array([ay])
    while (x <= D_x and y >= D_y):
            vy += (-g + ay) * dt # temp
            VY = np.append(VY, vy)
            vx += ax * dt # temp
            VX = np.append(VX, vx)
            ax = -K * eta * vx # temp
            AX = np.append(AX, ax)
            ay = -K * eta * vy # temp
            AY = np.append(AY, ay)
            x += vx * dt # temp
            X = np.append(X, x)
            y += vy * dt # temp
            Y = np.append(Y, y)

            if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <= PosSec):
                    print('Booom!')
                    break

    return [X, Y, x, y]
```

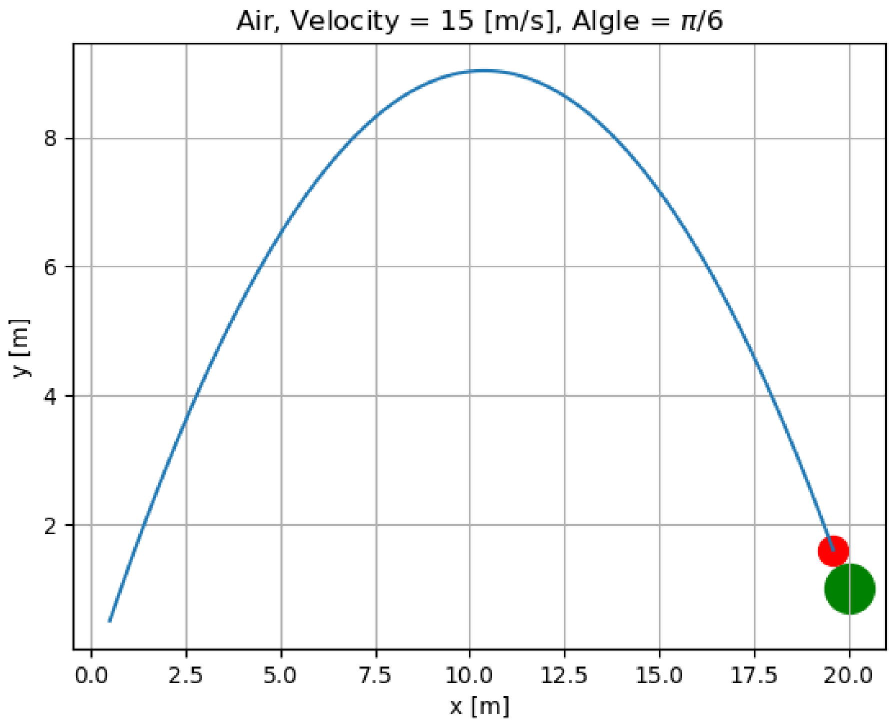

Figure 4: If the bird is in the air with $\eta = 2E - 4[mks\ unit]$, we can get the successful event with initial speed and angle $(15, \pi/6)[mks\ unit]$.

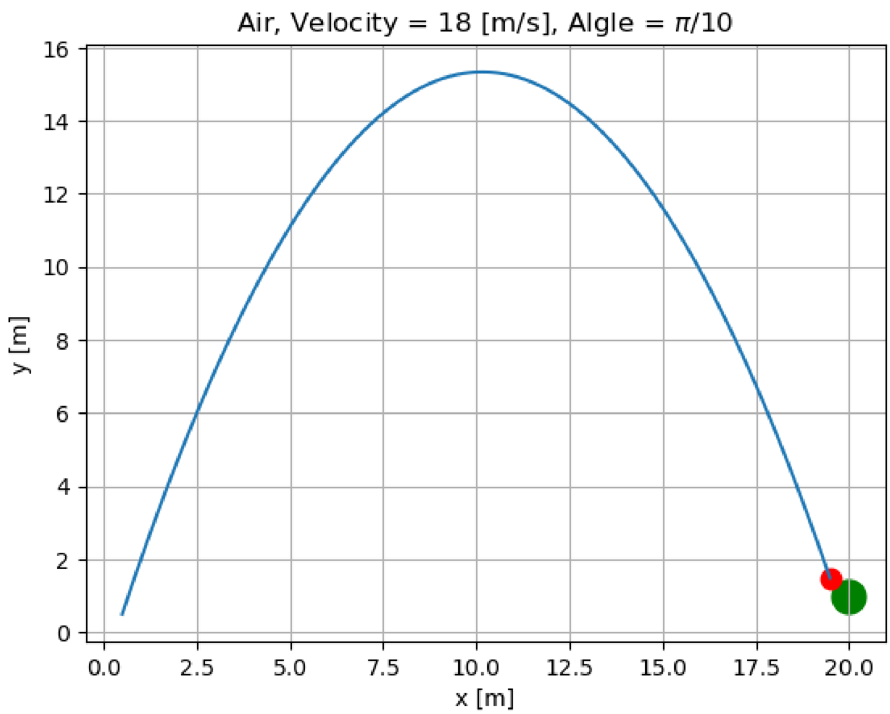

Figure 5: If the bird is in the air with $\eta = 2E - 4[mks\ unit]$, we can get the successful event with initial speed and angle $(18, \pi/10)[mks\ unit]$.

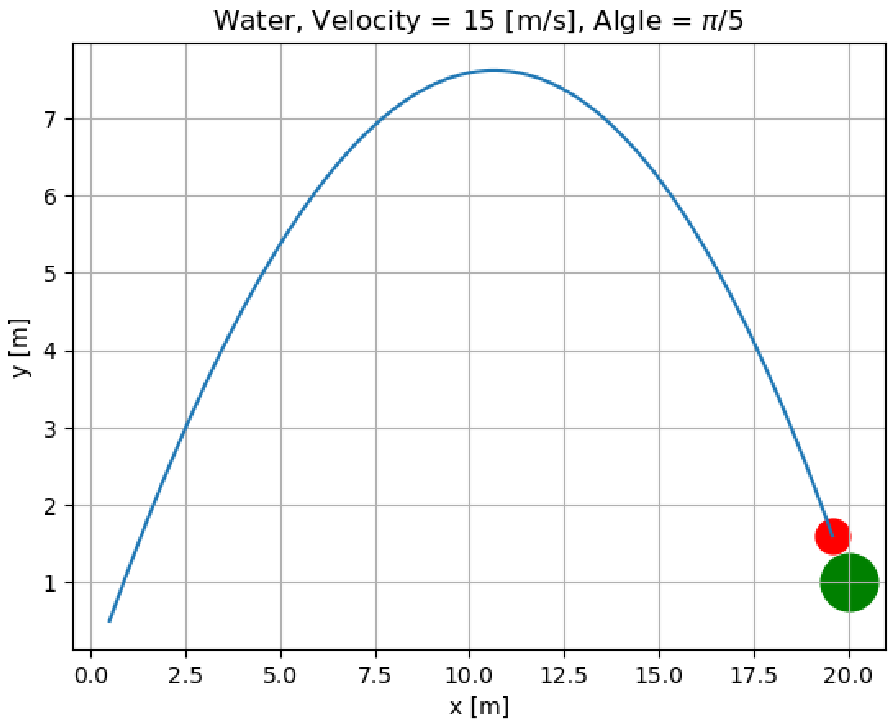

Figure 6: If the bird is in the water with $\eta = 0.01[mks\ unit]$, we can get the successful event with initial speed and angle $(15, \pi/5)[mks\ unit]$.

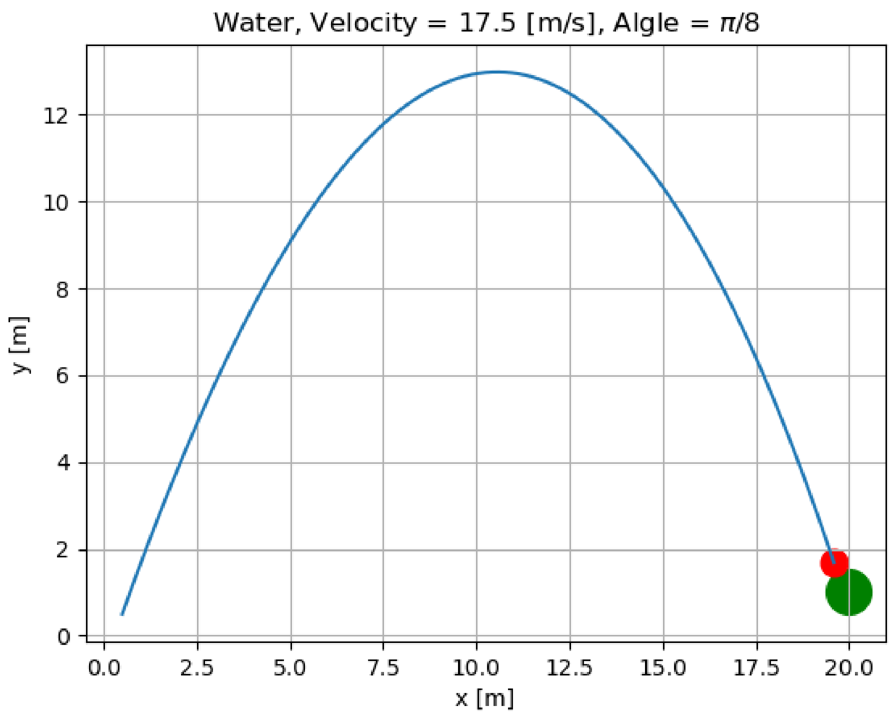

Figure 7: If the bird is in the water with $\eta = 0.01[mks\ unit]$, we can get the successful event with initial speed and angle $(17.5, \pi/8)[mks\ unit]$.

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 π calculation

```
# %% [markdown]
# ## Computational Physics Lab
# ### Homework 1.1
# Yuan-Yen Peng
# Dept of Physics, NTHU, Taiwan
# October 17, 2022

# %% [markdown]
#  calculation

# %%
import numpy as np

N1 = int(10E4)
N2 = int(10E5)
N3 = int(10E6)
# Set how many small rectangles. (divided number)

# %%
# method 1, use hand writing for-loop.
def cal_meth1(N: int):
  dx = 1/N
  area = 0

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

  return Area

print("pi of N = 10000: ", cal_meth1(N1))
%timeit cal_meth1(N1)
print("pi of N = 100000: ", cal_meth1(N2))
%timeit cal_meth1(N2)
print("pi of N = 1000000: ", cal_meth1(N3))
%timeit cal_meth1(N3)

# %%
# method 2, use default sum.
def cal_meth2(N: int):
  dx = 1/N
  area = 0

```

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

  return Area

print("pi of N = 10000: ", cal_meth2(N1))
%timeit cal_meth1(N1)
print("pi of N = 100000: ", cal_meth2(N2))
%timeit cal_meth1(N2)
print("pi of N = 1000000: ", cal_meth2(N3))
%timeit cal_meth1(N3)

# %%
# method 3, use numpy.sum.
def cal_meth3(N: int):
  dx = 1/N
  area = 0

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

  return Area

print("pi of N = 10000: ", cal_meth3(N1))
%timeit cal_meth1(N1)
print("pi of N = 100000: ", cal_meth3(N2))
%timeit cal_meth1(N2)
print("pi of N = 1000000: ", cal_meth3(N3))
%timeit cal_meth1(N3)
```

The performance code is in the below.

```
# %% [markdown]
# ## Computational Physics Lab
# Yuan-Yen Peng
# Dept of Physics, NTHU, Taiwan

# %%
import numpy as np
import matplotlib.pyplot as plt
import time

N = int(10E3)

# %%
# method 1, use hand writing for-loop.
def cal_meth1(N: int):
  dx = 1/N
  area = 0

```

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

    return Area


t1 = time.time()
X = np.linspace(1, N, N)
Y = np.array([])
for i in range (0, N):
  Y = np.append(Y, cal_meth1(int(X[i])))
t2 = time.time()

print("time different = ", [t2 - t1])
print(f"The value is {Y[-1]}")
plt.plot(X, Y, ".", color = "r", label = "N = 10E3")

# %%
# method 2, use default sum.
def cal_meth2(N: int):
  dx = 1/N
  area = 0

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

  return Area

t1 = time.time()
X = np.linspace(1, N, N)
Y = np.array([])
for i in range (0, N):
  Y = np.append(Y, cal_meth2(int(X[i])))
t2 = time.time()

print("time different = ", [t2 - t1])
print(f"The value is {Y[-1]}")
plt.plot(X, Y, ".", color = "g", label = "N = 10E3")

# %%
# method 3, use numpy.sum.
def cal_meth3(N: int):
  dx = 1/N
  area = 0

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

```
        return Area

    t1 = time.time()
    X = np.linspace(1, N, N)
    Y = np.array([])
    for i in range (0, N):
        Y = np.append(Y, cal_meth3(int(X[i])))
    t2 = time.time()

    print("time different = ", [t2 - t1])
    print(f"The value is {Y[-1]}")
    plt.plot(X, Y, ".", color = "b", label = "N = 10E3")
```

2.2 Stefan-Boltzmann constant σ_B

```
# %% [markdown]
# ## Computational Physics Lab
# ### Homework 1.2
# Yuan-Yen Peng
# Dept of Physics, NTHU, Taiwan
# October 17, 2022

# %% [markdown]
# The Stefan-Boltzmann constant $\sigma_{B}$ calculation

# %%
%reset -f
# clear previous variables

import numpy as np
import scipy.constants as const

pi = const.pi
h = const.h
c = const.c
k = const.k
T = 6000
# set constants

# %%
def sigma(N, upper):
  '''
  :param N: divided number.
  :upper: upper bound for infinity, so set it as large as possible.
  '''
  tor_denu = 10E-10
  # lower tolerance math error.
  dNu = upper/N
  Nu = np.linspace(0, upper, N)
  B_nu = 2 * h * Nu**3
  B_denu = c**2 * (np.exp(h * Nu / (k * T)) - 1) + tor_denu
```

```
    # lower tolerance (+ 10E-10) to avoid 1/0 (math error).
    sig = np.sum(B_nu * dNu / B_denu) * pi/T**4

    return sig

N = int(10E5)
upper = int(10E15)
# as large as we can, because it need to be inf.
print(sigma(N = N, upper = upper))
%timeit sigma(N, upper)
```

2.3 Angry Bird

```
# %% [markdown]
# ## Computational Physics Lab
# ### Homework 1.3
# Yuan-Yen Peng
# Dept of Physics, NTHU, Taiwan
# October 17, 2022

# %% [markdown]
# Angry Bird

# %%
import numpy as np
import scipy as sp
import scipy.constants as const
import matplotlib.pyplot as plt

# %%
dt = 0.01
g = const.g
tor = 0.03 # (+ 0.05 --> tolerance)
mass = 5 # kg
R = 0.3 # bird radius [meter]
R_T = 0.5 # target radius [meter]
# bird's position
PosX = 0.5
PosY = 0.5
# The size of the playground
D_x = 100
D_y = 0 + R + tor
# Target position [meter, meter]
T = [20, 1]
# collide parameter
PosSec = R + R_T + tor
# specify the point size
s_bird = np.square(40 * R) # the point area (cm^2)
s_pig = np.square(40 * R_T) # the point area (cm^2)

# %% [markdown]
# Normal condition
```

```
# %%
def tr(PosX, PosY, Vel, theta):
      VelX = Vel * np.sin(theta)
      VelY = Vel * np.cos(theta)
      x, y, vx, vy = PosX, PosY, VelX, VelY # temp
      X = np.array([x])
      Y = np.array([y])
      VX = np.array([vx])
      VY = np.array([vy])
      while (x <= D_x and y >= D_y):
            vy += -g * dt # temp
            VY = np.append(VY, vy)
            vx = vx # temp
            VX = np.append(VX, vx)
            x += vx * dt # temp
            X = np.append(X, x)
            y += vy * dt # temp
            Y = np.append(Y, y)
            if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <=
               PosSec):
                  print('Booom!')
                  break

      return [X, Y, x, y]


# %%
# bird's velocity and angle
Vel = 15
theta = np.pi/6

plt.scatter(T[0], T[1], s_pig, c = 'g')
plt.scatter(tr(PosX, PosY, Vel, theta)[2], tr(PosX, PosY, Vel, theta)[3],
   s_bird, c = 'r')
plt.grid()
plt.plot(tr(PosX, PosY, Vel, theta)[0],
       tr(PosX, PosY, Vel, theta)[1])

# %% [markdown]
# Add the drag force, air and water respectively.

# %%
def tr(PosX, PosY, Vel, theta, eta):
      VelX = Vel * np.sin(theta)
      VelY = Vel * np.cos(theta)
      K = 6 * np.pi * R
      x, y, vx, vy = PosX, PosY, VelX, VelY # temp
      ax = -K * eta * vx # temp
      ay = -K * eta * vy # temp
      X = np.array([x])
      Y = np.array([y])
      VX = np.array([vx])
```

```
    VY = np.array([vy])
    AX = np.array([ax])
    AY = np.array([ay])
    while (x <= D_x and y >= D_y):
        vy += (-g + ay) * dt # temp
        VY = np.append(VY, vy)
        vx += ax * dt # temp
        VX = np.append(VX, vx)
        ax = -K * eta * vx # temp
        AX = np.append(AX, ax)
        ay = -K * eta * vy # temp
        AY = np.append(AY, ay)
        x += vx * dt # temp
        X = np.append(X, x)
        y += vy * dt # temp
        Y = np.append(Y, y)

        if (np.sqrt(np.square(T[0] - x) + np.square(T[1] - y)) <= PosSec):
            print('Booom!')
            break

    return [X, Y, x, y]

# %%
# in air

# bird's velocity and angle
Vel = 18
n = 10
theta = np.pi/n

eta_air = 2E-4
tr_air = tr(PosX, PosY, Vel, theta, eta_air)
plt.scatter(T[0], T[1], s_pig, c = 'g')
plt.scatter(tr_air[2], tr_air[3], s_bird, c = 'r')
plt.grid()
plt.plot(tr_air[0], tr_air[1])
plt.title(f'Air, Velocity = {Vel} [m/s], Algle = $\pi$/{n}')
plt.xlabel('x [m]')
plt.ylabel('y [m]')

# %%
# in water

# bird's velocity and angle
Vel = 17.5
n = 8
theta = np.pi/n

eta_water = 1E-2
tr_water = tr(PosX, PosY, Vel, theta, eta_water)
plt.scatter(T[0], T[1], s_pig, c = 'g')
```

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