BIRZEIT UNIVERSITY
Physics Department
PHYS338: Computational Physics
Final Exam

Name: Rashad Hamidi Fall Semester 2020/2021

Problem 1

(20%) Consider the differential equation

$$\frac{dy}{dx} = \sqrt{y + x^2}$$

If $y(0) = 1$,

a) Plot y for x between 0 and 3.
b) Find the value of $y(3)$ to at least 8 decimal places? How did you estimate the accuracy?

Solution 1

Computational Methods

Euler Method.

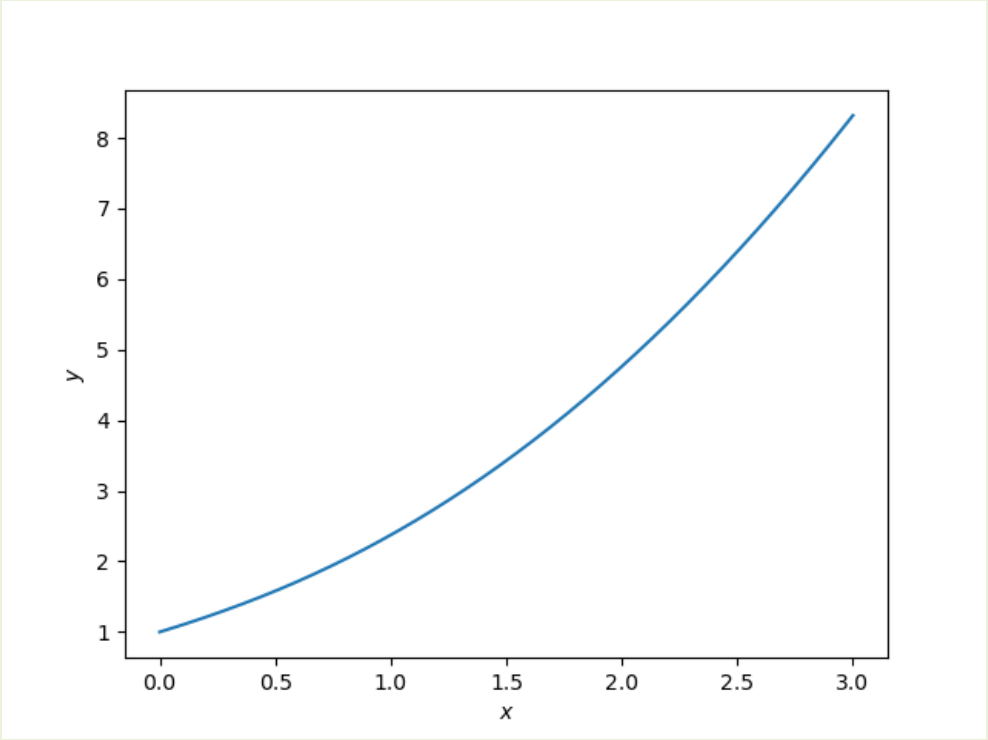
Python Code

```
'''
  Equation : dy/dx = sqrt(y + x^2)
  Computational Method : Euler Method
'''

# libraries
import numpy as np
import matplotlib.pyplot as plt

# number of iterations
N = 10000

x = np.zeros(N)
y = np.zeros(N)

# initial condition
x[0] = 0
y[0] = 1

# the greatest value of x
x[N-1] = 3

# step length
dx = (x[N-1] - x[0])/N

# fill x array
for i in range(N-1):
  x[i+1] = x[i] + dx

# fill y array using Euler method
for i in range(N-1):
```

```
    y[i+1] = y[i] + (x[i+1] - x[i]) * np.sqrt(y[i] + x[i]**2)

# plot the results
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.plot(x,y)
plt.savefig('E:/Birzeit_University/PHYS338/Exams/Q1.png')

# print y(3)
print(y[N-1])
```

Results

a) Plot:



b) The value of $y(3)$ is:

Number of Iterations	Value
10000	8.317758979934174
100000	8.31943675529212
1000000	8.31960454658485

The code itself estimates the value for high number of decimals (16 digits).

Problem 2

(15%) Evaluate to at least 12 decimal places

$$I = \int_{-\pi}^{\pi} e^{-\sin(x)} dx$$

What method did you use? and how did estimate accuracy?

Solution 2

Computational Methods

Simpson Rule.

Python Code

```
'''
  Equation : I = integrate[exp(-sin(x)),(x,-pi,pi)]
  Computational Method : Simpson Rule
'''

# libraries
import matplotlib.pyplot as plt
import numpy as np

# the integrand f(x)
def f(x):
  return np.exp(-np.sin(x))

# Simpson Rule
def Simpson(a, b, N):
  S = 0
  dz = (b - a)/(2 * N)
  for i in range(0, N):
    z0 = a + dz * 2 * i
    z1 = a + dz + dz * 2 * i
    z2 = a + 2 * dz + dz * 2 * i
    S = S + dz/3 * (f(z0) + 4*f(z1) + f(z2))
  return S

x_initial = -np.pi  # x initial
x_final = np.pi    # x final
N = 100        # number of partitions of integration

# integrate
I = Simpson(x_initial, x_final, N)

# print the value of the integral I
print(I)
```

Results

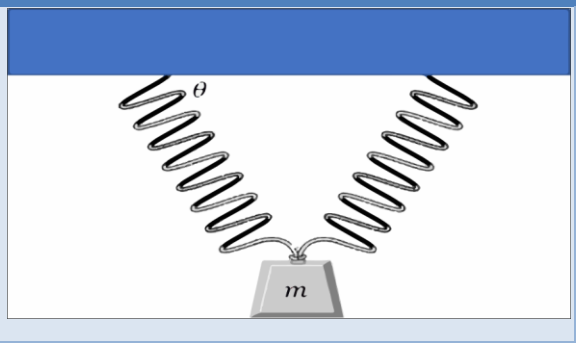
The value of I is:

Number of Iterations	Value
10	7.954926522165998
100	7.954926521012852
1000	7.95492652101284
10000	7.954926521012873

The code itself estimates the value for high number of decimals (16 digits).

Problem 3

(20%) Consider two springs of equilibrium length $L_0 = 10.0\ cm$ and spring constant $k = 640\ N/m$ that are fixed to the ceiling at two points $2L_0$ apart. If an object of mass m is linked to the other sides of the two springs as shown in the figure. Find the mass m that makes the angle $\theta = 45°$.



Solution 3

Computational Methods

Newton-Raphson Method.

Python Code

```
'''
  Equation : mg = 2 k L0 (sec(theta) - 1) sin(theta)
  Computational Method : Newton-Raphson Method
'''

# libraries
import numpy as np
import matplotlib.pyplot as plt

# define the function f(theta)
def f(theta):
  return 2 * k * L0 * (1/np.cos(theta) - 1) * np.sin(theta) - m * g

# define the first derivative of the function f(theta)
def diff(theta):
  return 2 * k * L0 * (np.sin(theta)**2/np.cos(theta)**2 + (1/np.cos(theta) - 1) * np.cos(theta))

# Newton Raphson Method
def NewtonRaphson(m):
  theta = x0       # set theta as the initial guess
  # the iteration equation
  for i in range (0, N):
    Theta = theta - f(theta)/diff(theta)
    theta = Theta
  return theta

# the constants of the equation
m = 5            # the mass range (kg)
g = 9.8          # the gravity acceleration (m/s^2)
k = 640          # the spring constant (N/m)
L0 = 0.1         # the rlaxation length of the spring (m)

# the initial parameters
x0 = 1           # the initial guess
N = 100          # the number of iterations
```

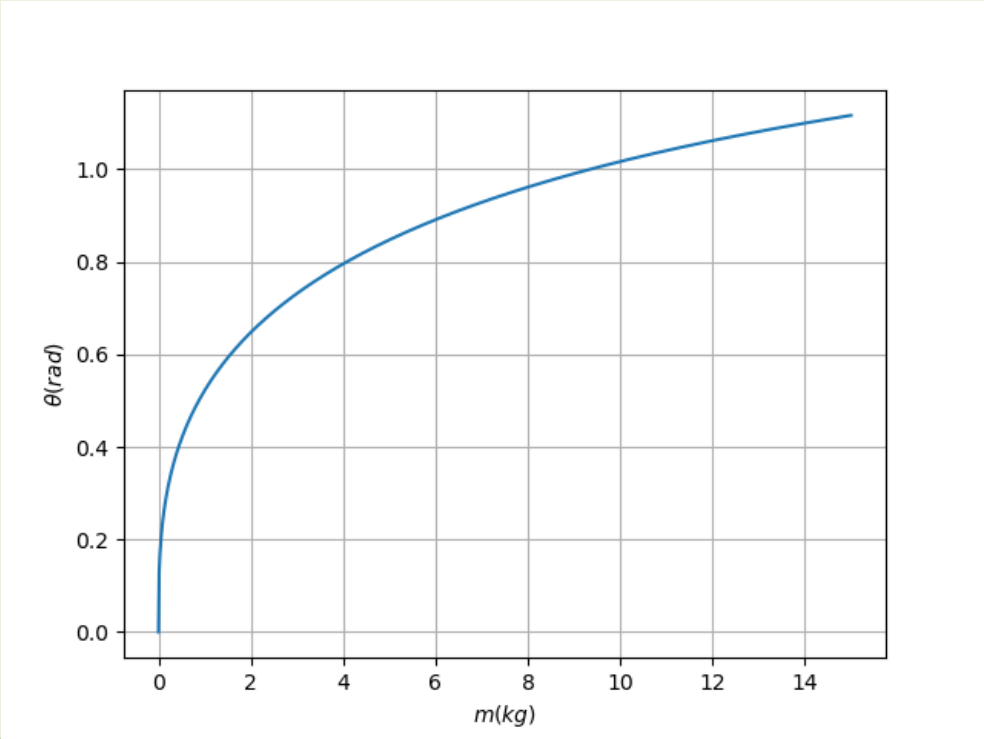
```
# find theta for m = 5 kg using Newton Raphson method
theta = NewtonRaphson(m)

# print theta (radian, degree) for m = 5 kg
print(theta, theta*180/np.pi)
```

Results

The corresponding θ for $m = 5\ kg$ is

$$\big(\theta(rad) \leftrightarrow\ \theta(degree)\big) = (0.847395791567932 \leftrightarrow 48.5522024339901)$$

Here a plot illustrates the relation between θ and m.



Problem 4

(15%) Consider a very long rectangular capillary tube of edge length $10nm$. This tube is filled with electrolyte solution. If three sides of the tube where held at a potential of $1V$, while the fourth side is held at zero potential. Find the electrostatic potential of the system by solving

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} - k^2 V = 0$$

With the boundary conditions

$$V(x,0) = 1.0\,, V(x,10) = 1.0\,, V(0,y) = 1.0 \text{ and } V(10,y) = 0$$

Solve the equation above for different values of k to plot $V(5{,}5)$ as a function of k for k between 0 and 10.

Solution 4

Computational Methods

2D Finite Difference Method.

Python Code

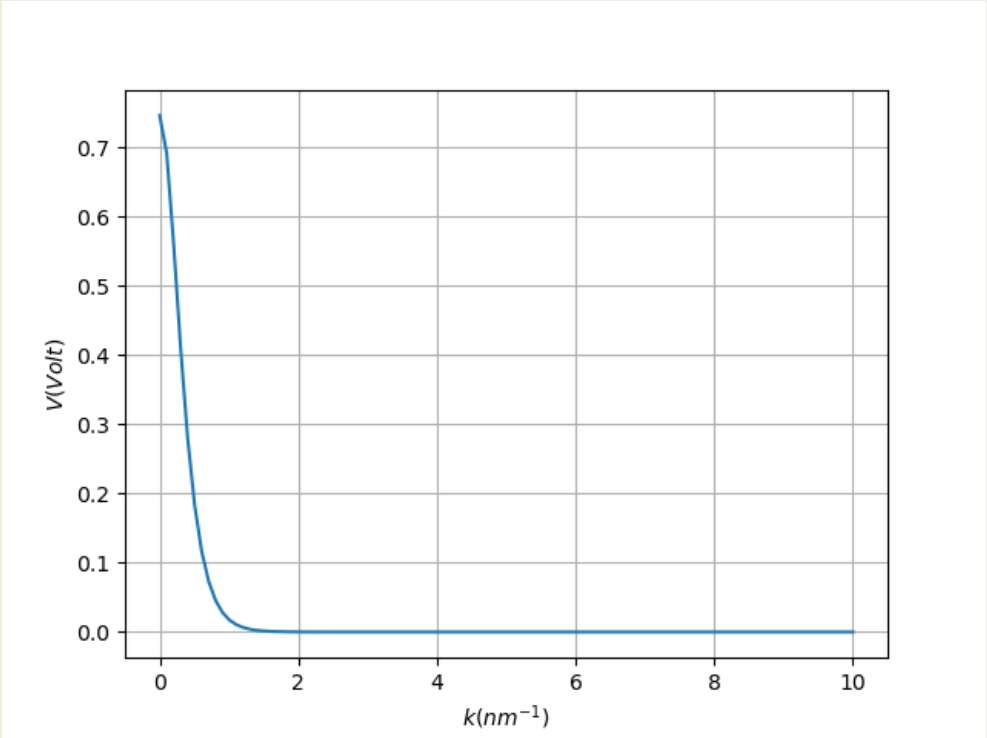
```
'''
  Equation : (D^2 - k^2) V(x,y) = 0
  Computational Method : 2D Finite Difference Method
'''

# libraries
import numpy as np
import matplotlib.pyplot as plt

N = 100   # number of points in each column/row

# boundaries (Volt)
V_bottom = 1
V_top = 1
V_left = 1
V_right = 0

L = 10            # m, square plate length
k = np.linspace(0, 10, 100) # nm^-1, values of k
h = L/N           # nm/point, step length

# the system AV = b
A = np.zeros([N*N, N*N])
V = np.zeros([N*N, 1])
b = np.zeros([N*N, 1])

R = np.zeros(len(k))     # array to save all values of V(5, 5)

# building the array R
for j in range(len(k)):

  # build A matrix and b for Finite Difference Method
  for i in range(0, N*N):

    # left-bottom corner
    if i == 0:
      A[i, i] = -4 -k[j]**2 * h**2
```

```
      A[i, i+1] = 1
      A[i, i+N] = 1
      b[i] = -V_left -V_bottom

    # left-top corner
    elif i == N-1:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i-1] = 1
      A[i, i+N] = 1
      b[i] = -V_left -V_top

    # right-bottom corner
    elif i == N*N-N:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i+1] = 1
      A[i, i-N] = 1
      b[i] = -V_right -V_bottom

    # right-top corner
    elif i == N*N-1:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i-1] = 1
      A[i, i-N] = 1
      b[i] = -V_right -V_top

    # left side
    elif i > 0 and i < N - 1:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i+1] = 1
      A[i, i-1] = 1
      A[i, i+N] = 1
      b[i] = -V_left

    # right side
    elif i > N*N-N-1 and i < N*N-1:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i+1] = 1
      A[i, i-1] = 1
      A[i, i-N] = 1
      b[i] = -V_right

    # top side
    elif (i+1)%N == 0:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i-1] = 1
      A[i, i+N] = 1
      A[i, i-N] = 1
      b[i] = -V_top

    # bottom side
    elif i%N == 0:
      A[i, i] = -4 -k[j]**2 * h**2
      A[i, i+1] = 1
      A[i, i+N] = 1
      A[i, i-N] = 1
      b[i] = -V_bottom

    # inner block
    else:
```

```
            A[i, i] = -4 -k[j]**2 * h**2
            A[i, i+1] = 1
            A[i, i-1] = 1
            A[i, i+N] = 1
            A[i, i-N] = 1

        # y = H_inverse * b
        V = np.linalg.solve(A, b)

        # reshape V from vector to matrix
        V = np.reshape(V, (N, N)).T

        # filling the array R
        R[j] = V[int(5/h), int(5/h)]

        # progress
        print(j)

# plot V(5, 5) vs k
plt.xlabel('$k (nm^{-1})$')
plt.ylabel('$V (Volt)$')
plt.plot(k, R)
plt.grid(True)
plt.savefig('E:/Birzeit_University/PHYS338/Exams/Q4.png')
```

Results

Plot $[k, V(5,5)]$ for $k \in [0,10]$:



Problem 5

(15%) In 2012 Felix Baumgartner jumped from an altitude of $h_0 = 39{,}045\ m$ towards the earth. We would like to model his fall trajectory, velocity and duration. To achieve that we consider having two forces affecting him during the fall. The first force is the gravitational force $F_g = G\frac{mM_{earth}}{(R_{earth}+h)^2} = \frac{mg}{\left(1+\frac{h}{R_{earth}}\right)^2}$. Where $m = 70\ kg$ is the mass of Felix and h is his distance from the surface of the earth. The second force is the drag force which we will model with $F_D = \frac{1}{2}C\rho A v^2$. Where C is the drag coefficient; ρ is the density of air; $A = 0.120\ m^2$ and v are the effective cross section area and velocity of the falling object, respectively. We need to further consider that the air density over earth surface is not constant. It decreases with height. As an approximation, consider that $\rho(h) = \rho_0 \exp\left(-\frac{h}{H}\right) kg/m^3$. Where $\rho_0 = 1.225\ kg/m^3$ and $H = 10.4\ km$. In addition, the drag coefficient usually depends on the shape and velocity of the moving object relative to the speed of sound ($v_s = 343\ m/s$). Assume for our case, it is given by

$$C(v) = \begin{cases} 0.65 & , \quad \frac{v}{v_s} < 0.6 \\ 0.65 + 0.55\left(\frac{v}{v_s} - 0.6\right)^2 & , \quad 0.6 < \frac{v}{v_s} < 1.1 \\ 0.7875 + 0.32\left(\frac{v}{v_s} - 1.1\right)^2 & , \quad \frac{v}{v_s} > 1.1 \end{cases}$$

Therefore, Felix trajectory can be modeled using

$$\frac{dv}{dt} = \frac{g}{\left(1+\frac{h}{R_{earth}}\right)^2} - \frac{1}{2}\frac{A}{m}C(v)\rho_0 \exp\left(-\frac{h}{H}\right)v^2$$

$$\frac{dh}{dt} = -v$$

Write your own code or modify the codes provided to you to solve these equations and subsequently

a) Plot Felix velocity as a function of distance from earth surface of the case of skydiving head down ($A = 0.120\ m^2$) and skydiving with belly down ($A = 0.450\ m^2$).
b) Compare the flight time of both cases mentioned in (a).

Solution 5

Computational Methods

Fourth-Order Runge-Kutta Method.

Python Code

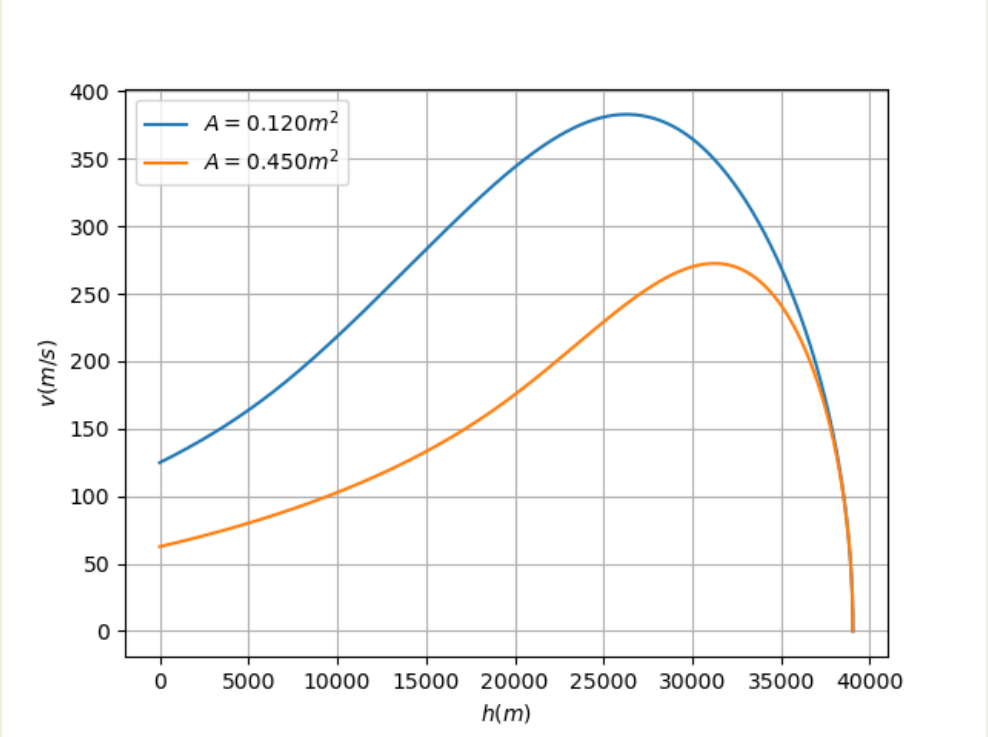
```
'''
  Equation : Felix's Jump
  Computational Method : Fourth-Order Runge-Kutta Method
'''

# libraries
import numpy as np
import matplotlib.pyplot as plt


# The gravitational force over Felix's mass.
def fg(h):
```

```
  return g / (1 + h / R_earth)**2

# The drag force of the air over Felix's mass.
def fD(v, h, A):

  # The drag coefcient.
  r = v / v_sound
  C = 0
  if r < 0.6:
    C = 0.65
  elif r >= 0.6 and r <= 1.1:
    C = 0.65 + 0.55 * (r - 0.6)**2
  elif r > 1.1:
    C = 0.7875 + 0.32 * (r - 1.1)

  # The density of air.
  p = p0 * np.exp(-h / H)

  return A / (2 * m) * C * p * v**2

# dv/dt
def dv(v, h, A):
  return fg(h) - fD(v, h, A)

# dh/dt
def dh(v):
  return -v

# Runge-Kutta Method
def RK4(v0, h0, dt, A):

  # Arrays
  t = [0]    # Time array with initial value 0.
  h = [h0]   # Latitude array with initial value h0.
  v = [v0]   # Velocity array with initial value v0.

  # Parameters for the while loop.
  k = 1     # Closed loop parameter
  i = 0     # Increment parameter

  # RK4 algorithm
  while k == 1:

    ch1 = dt * dh(v[i])
    ch2 = dt * dh(v[i] + 0.5 * ch1)
    ch3 = dt * dh(v[i] + 0.5 * ch2)
    ch4 = dt * dh(v[i] + ch3)

    # If Felix arrive the surface shut the loop off
    if h[i] < 0:
      break

    h.append(h[i] + 1 / 6 * (ch1 + 2 * ch2 + 2 * ch3 + ch4))

    cv1 = dt * dv(v[i], h[i], A)
    cv2 = dt * dv(v[i] + 0.5 * cv1, h[i] + 0.5 * ch1, A)
    cv3 = dt * dv(v[i] + 0.5 * cv2, h[i] + 0.5 * ch2, A)
    cv4 = dt * dv(v[i] + cv3, h[i] + ch3, A)
```

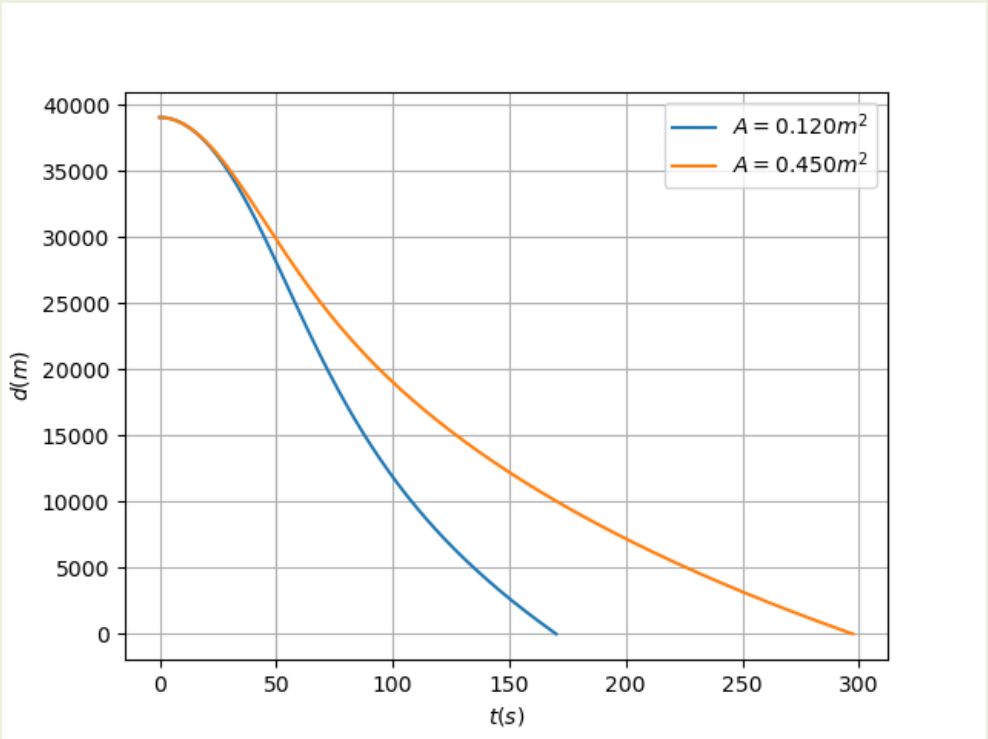
```
    v.append(v[i] + 1 / 6 * (cv1 + 2 * cv2 + 2 * cv3 + cv4))

    t.append(t[i] + dt)

    i += 1

  return t, h, v

# Constants
v_sound = 343 # m/s, the speed of sound
G = 6.67408 * 10**-11 # m^3/kg.s^2, the gravitational constant
M_earth = 5.972 * 10**24 # kg, the mass of the earth
R_earth = 6371.4 * 10**3 # m, the radius of the earth, from New Mixeco where Felix jumped
p0 = 1.225 # kg/m^2, the density of air at the surface
H = 10.4 * 10**3 # m, scalling hight constant
A = [0.120, 0.450] # m^2, the effective cross sectional area skydiving [head, belly] down
m = 70 # kg, the mass of Felix
g = G * M_earth / R_earth**2 # m/s^2, gravitational acceleration from the surface

v0 = 0       # m/s, initial velocity
h0 = 39045    # m, initial height
dt = 0.01     # s, time step

# Solution
Sol1 = RK4(v0, h0, dt, A[0])
Sol2 = RK4(v0, h0, dt, A[1])

# the velocity as a function of distance
fig = plt.figure()
plt.xlabel('$h (m)$')
plt.ylabel('$v (m/s)$')
plt.plot(Sol1[1], Sol1[2], label='$A = 0.120 m^{2}$')
plt.plot(Sol2[1], Sol2[2], label='$A = 0.450 m^{2}$')
plt.grid(True)
plt.legend()
plt.savefig('E:/Birzeit_University/PHYS338/Exams/Q5.png')

# the distance as a function of time
fig = plt.figure()
plt.xlabel('$t (s)$')
plt.ylabel('$d (m)$')
plt.plot(Sol1[0], Sol1[1], label='$A = 0.120 m^{2}$')
plt.plot(Sol2[0], Sol2[1], label='$A = 0.450 m^{2}$')
plt.grid(True)
plt.legend()
plt.savefig('E:/Birzeit_University/PHYS338/Exams/Q52.png')

# print the total time for (A = 0.120 m^2, A = 0.450 m^2)
print(Sol1[0][-1], Sol2[0][-1])
```

Results

a) Felix's velocity as a function of distance from earth surface:



b) Felix's distance from earth surface as a function of time:



The total time for $(A = 0.120\ m^2 \leftrightarrow A = 0.450\ m^2)$ in seconds $(170.15999999999022 \leftrightarrow 297.4399999998745)$