Assignment – Week 2

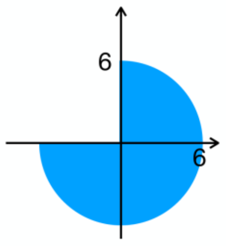
Azhar Harisandi

22320011

I. Problems

1. By using bisection method, find the root of the following functions:
   a. $f(x) = x^2 - 3x - 2$
   b. $f(x) = x^3 + x^2 - 3x - 2$

2. By using numerical integration, find the area under the following curve

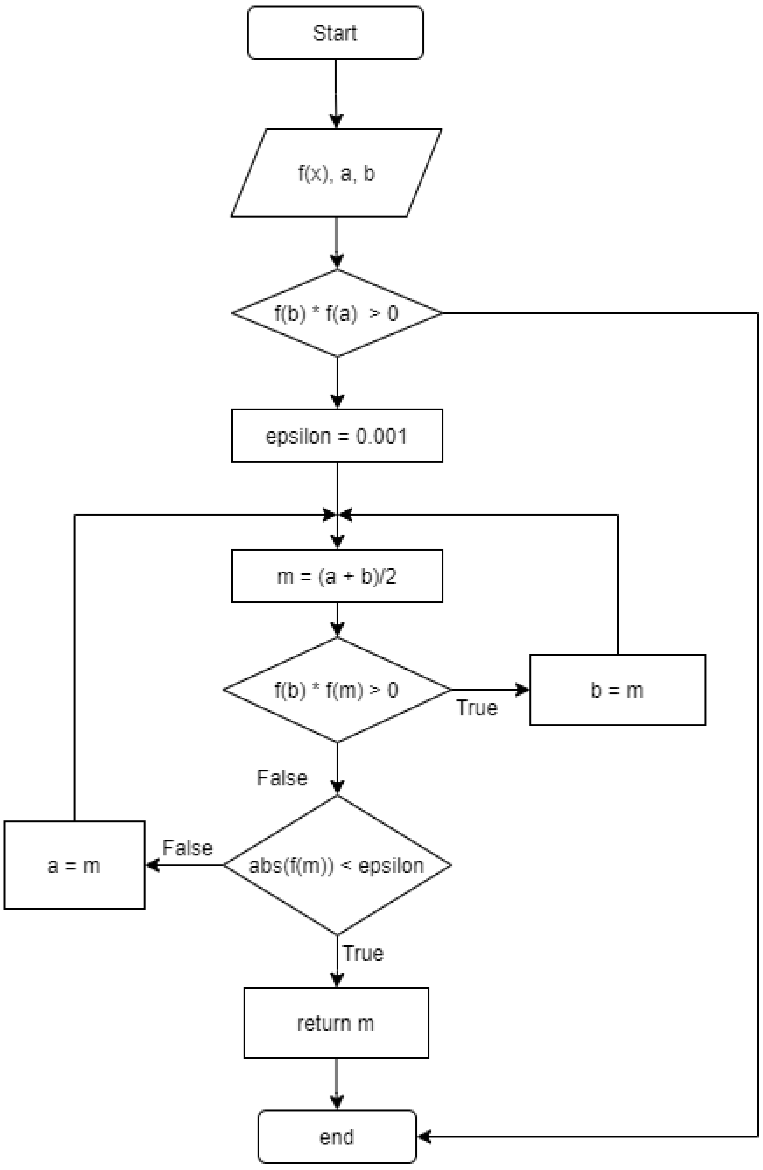


3. A ball at 1200K is allowed to cool down in air at an ambient temperature of 300K. Assuming heat is only due to radiation, the differential equation for the temperature of the ball is given by :

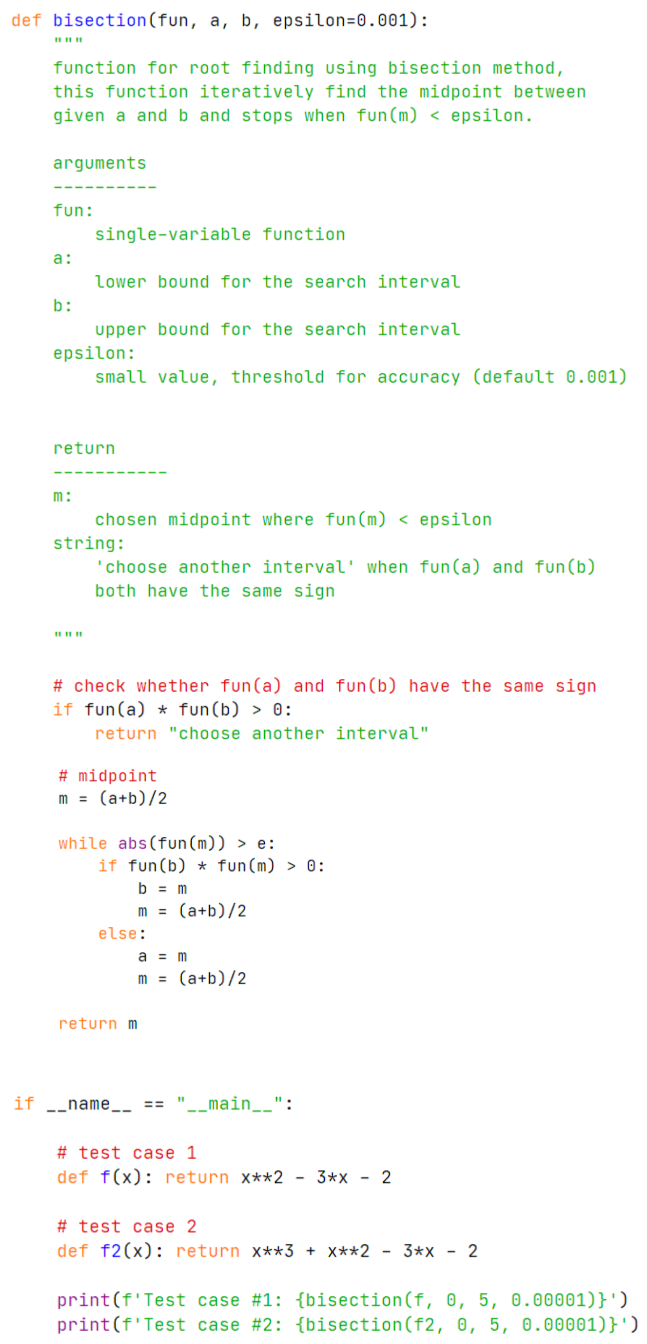
$$\frac{d\theta}{dt} = -2.2067 \times 10^{-12} \left(\theta^4 - 81 \times 10^9\right);\ \theta(0) = 1200K$$

II. Solutions

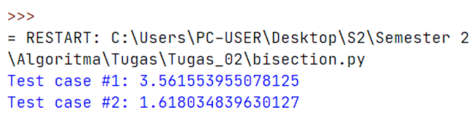
1. Flowchart for bisection method



Code snippets for bisection method

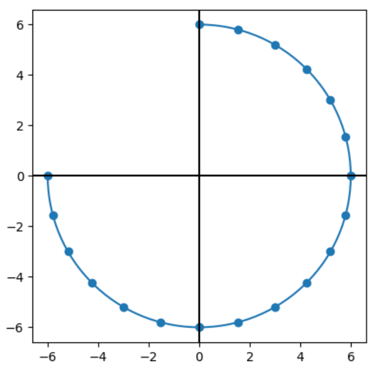


```python
def bisection(fun, a, b, epsilon=0.001):
    """
    function for root finding using bisection method,
    this function iteratively find the midpoint between
    given a and b and stops when fun(m) < epsilon.

    arguments
    ----------
    fun:
        single-variable function
    a:
        lower bound for the search interval
    b:
        upper bound for the search interval
    epsilon:
        small value, threshold for accuracy (default 0.001)


    return
    -----------
    m:
        chosen midpoint where fun(m) < epsilon
    string:
        'choose another interval' when fun(a) and fun(b)
        both have the same sign

    """

    # check whether fun(a) and fun(b) have the same sign
    if fun(a) * fun(b) > 0:
        return "choose another interval"

    # midpoint
    m = (a+b)/2

    while abs(fun(m)) > e:
        if fun(b) * fun(m) > 0:
            b = m
            m = (a+b)/2
        else:
            a = m
            m = (a+b)/2

    return m


if __name__ == "__main__":

    # test case 1
    def f(x): return x**2 - 3*x - 2

    # test case 2
    def f2(x): return x**3 + x**2 - 3*x - 2

    print(f'Test case #1: {bisection(f, 0, 5, 0.00001)}')
    print(f'Test case #2: {bisection(f2, 0, 5, 0.00001)}')
```



```
>>>
= RESTART: C:\Users\PC-USER\Desktop\S2\Semester 2
\Algoritma\Tugas\Tugas_02\bisection.py
Test case #1: 3.561553955078125
Test case #2: 1.618034839630127
```

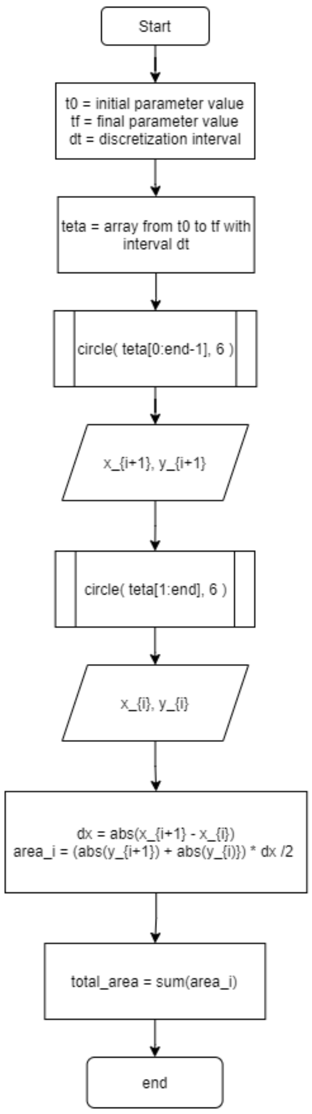
2. Parameterize the curve in terms of angle (from north)

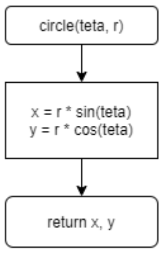


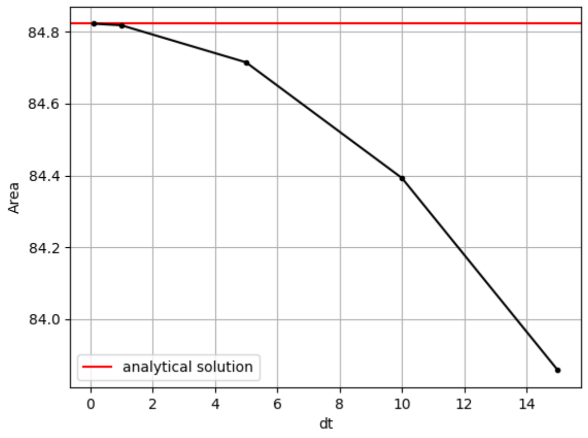
circle(t) = (x(t), y(t))

Where
x(t) = radius.sin(t)
y(t) = radius.cos(t)

Parametrized curve shown by continuous line, discretized interval (every 15 degree) shown in blue dots.


Start
t0 = initial parameter value
tf = final parameter value
dt = discretization interval
teta = array from t0 to tf with interval dt
circle(teta[0:end-1], 6)
x_{i+1}, y_{i+1}
circle(teta[1:end], 6)
x_{i}, y_{i}
dx = abs(x_{i+1} - x_{i})
area_i = (abs(y_{i+1}) + abs(y_{i})) * dx /2
total_area = sum(area_i)
end





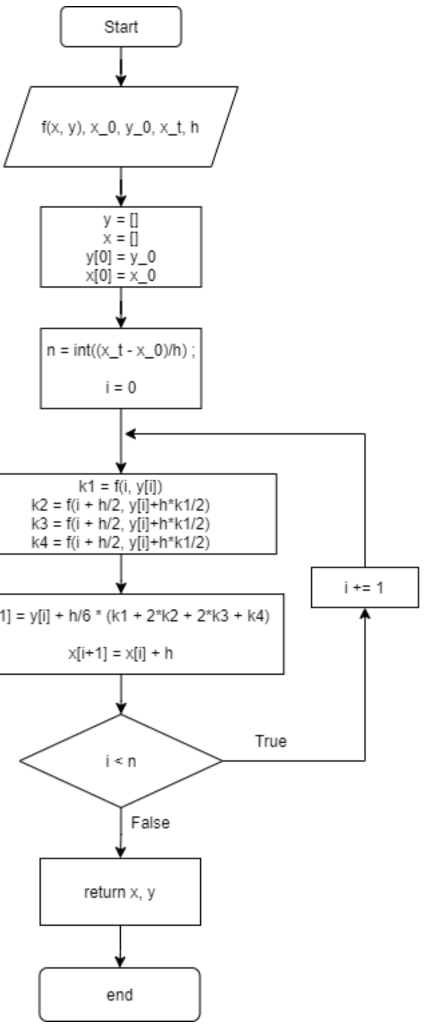
Comparison between analytical solution (red line) and various discretization interval (0.1, 1, 5, 10, 15)

Code snippets for numerical integration

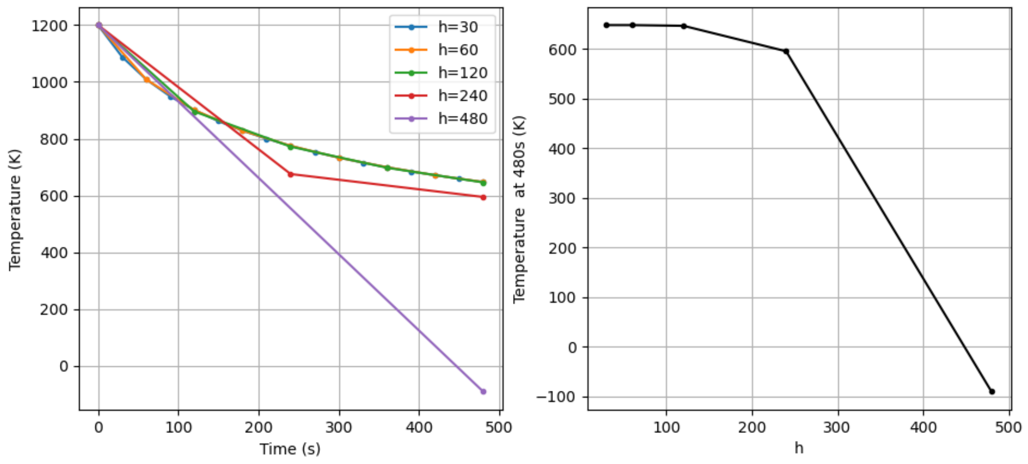
```
import numpy as np
import matplotlib.pyplot as plt

def circle(teta, r=6):
    """
    parameterized the curve (circle) based on angle
    angle convention : 0 from north (yaxis),
    and increasing clockwise

    c(t) = (x(t), y(t))
    x(t) = r*sin(t)
    y(t) = r*cos(t)

    params
    ---------
    teta:
     angle/azimuth from north (radian)
    r:
     radius of the circle (default=6)

    return
    ---------
     x, y : x and y coordinate of a point in a circle of radius r
    """
    x = r*np.sin(teta)
    y = r*np.cos(teta)
    return x, y
```

```
def numerical_integration(f, t0, tf, dt):
    """
    integrate numerically using trapezoidal method

    params
    --------
    f(t):
     parametric equation of a curve
     (must return x and y for every parameter t)

    t0, tf:
     initial and final value for the parameter

    dt:
     parameter's spacing (in degree for this particular case)


    return
    --------
    area under the curve
    """
    # discretize based on angle
    t = np.arange(t0, tf+np.deg2rad(dt), np.deg2rad(dt))

    # constraint
    if t[-1] > tf:
        t[-1] = tf

    # x_i, f_i
    x_before, y_before = f(t[:-1])

    # x_{i+1}, f_{i+1}
    x_after, y_after = f(t[1:])

    # delta x
    dx = abs(x_after - x_before)

    # trapezoid
    return np.sum((abs(y_before) + abs(y_after)) * dx / 2)
```

3. Runge-Kutta 4th order flowchart


Start
f(x, y), x_0, y_0, x_t, h
y = []
x = []
y[0] = y_0
x[0] = x_0
n = int((x_t - x_0)/h) ;
i = 0
k1 = f(i, y[i])
k2 = f(i + h/2, y[i]+h*k1/2)
k3 = f(i + h/2, y[i]+h*k1/2)
k4 = f(i + h/2, y[i]+h*k1/2)
y[i+1] = y[i] + h/6 * (k1 + 2*k2 + 2*k3 + k4)
x[i+1] = x[i] + h
i < n
True
i += 1
False
return x, y
end

Graph showing step size versus integration result



Code snippets

```python
def rk4(f, x0, y0, xt, h):

    # initial condition
    y = [y0]
    x = [x0]

    # loop until desired x
    for i in range(int((xt-x0)/h)):
        k1 = f(i, y[i])
        k2 = f(i + h/2, y[i] + h*k1/2)
        k3 = f(i + h/2, y[i] + h*k2/2)
        k4 = f(i + h, y[i] + h*k3)
        y.append(y[i] + h/6*(k1 + 2*k2 + 2*k3 + k4))
        x.append(x[i] + h)

    return x,y


def temp(t, teta):
    return -2.2067*1e-12 * (teta**4 - 8.1*1e9)
```