Obligatorisk oppgave 2, MAT-INF1100, Høst 2020

Cory Alexander Balaton

November 2020

Task 1

a

```python
import matplotlib.pyplot as plt

class Accel:
    def __init__(self, x_arr, y_arr):
        self.x_arr = x_arr
        self.y_arr = y_arr
        self.a_arr = []

    def plot(self, img_name):
        for idx in range(len(self.x_arr)-1):
            self.a_arr.append((self.y_arr[idx + 1] - self.y_arr[idx])/(self.x_arr[idx + 1] - self.x_arr[idx]))

        fig = plt.figure()
        fig.patch.set_facecolor('#cccccc')
        plt.plot(self.x_arr[:-1], self.a_arr)
        plt.xlabel("time(s)")
        plt.ylabel("acceleration(m/s^2)")
        plt.title("Acceleration")
        plt.savefig(img_name, facecolor=fig.get_facecolor(), edgecolor='none')
        #plt.show()
```

This is a class that takes the difference of some y values and divides it by the corresponding x values to get an average acceleration between to points, and plots an acceleration graph.

b

```python
import matplotlib.pyplot as plt

class Distance:
    def __init__(self, x_arr, y_arr):
        self.x_arr = x_arr
        self.y_arr = y_arr
        self.d_arr = []

    def plot_trapezoid(self, img_name, s_0=0):
        s = s_0
        for i in range(len(self.x_arr)-1):
            s += (self.x_arr[i+1] - self.x_arr[i]) * ((self.y_arr[i+1] + self.y_arr[i])/2)
            self.d_arr.append(s)

        fig = plt.figure()
        fig.patch.set_facecolor('#cccccc')
        plt.plot(self.x_arr[:-1], self.d_arr)
        plt.xlabel("time(s)")
        plt.ylabel("distance(m)")
        plt.title("Distance")
        plt.savefig(img_name, facecolor=fig.get_facecolor(), edgecolor='none')
        #plt.show()
```

This is a class that approximates the area of a curve by using the trapezoidal rule.

c

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

class TextToList:
    def __init__(self, textfile):
        self.textfile = textfile
        self.x = []
        self.y = []

    def ret_list(self):
        with open(self.textfile, "r") as f:
            a = f.readlines()
            for i in a:
                s = i.replace("\n", "").split(",")
                self.x.append(float(s[0]))
                self.y.append(float(s[1]))

        return [self.x, self.y]
```

This class takes a file that contains 2 parameters divided by a comma, converts it to 2 lists, then returns a list containing the 2 lists.

```
from TextToList import TextToList
from Accel import Accel
from Distance import Distance

f = TextToList("Oppg_1/running.txt")

l = f.ret_list()

a = Accel(l[0], l[1])
a.plot("images/Accel.png")

d = Distance(l[0], l[1])
d.plot_trapezoid("images/Distance.png")
```

This is the main program that creates instances of all the classes to create 2 plots.

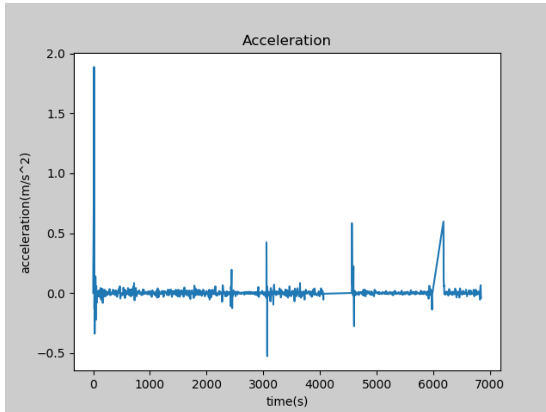

Figure 1: Acceleration plot

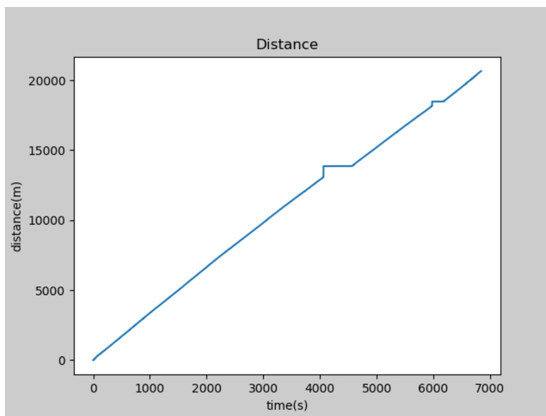

Figure 2: Distance plot

Looking at the Distance plot that was made by the program, it looks like the running session went for a little bit over 20km.

Task 2

a

Under is the original equation:

$$x' = x(\frac{1}{2} - x), \text{ where } x(0) = 1 \tag{1}$$

Solving the separable differential equation

Start by separating and then integrating:

$$\begin{aligned}
x' &= x(\frac{1}{2} - x) \\
\frac{dx}{dt} &= x(\frac{1}{2} - x) \\
\frac{1}{x(\frac{1}{2} - x)} dx &= 1dt \\
\frac{2}{x(1 - 2x)} dx &= 1dt \\
\int \frac{2}{x(1 - 2x)} dx &= \int 1dt \\
\int \frac{2}{x} + \frac{4}{1 - 2x} dx &= \int 1dt \\
2\int \frac{1}{x} dx - 2\int \frac{-2}{1 - 2x} dx &= \int 1dt \\
u = 1 - 2x \text{ and } du = -2dx \\
2\int \frac{1}{x} dx - 2\int \frac{1}{u} du &= \int 1dt \\
2\ln|x| - 2\ln|u| &= t + C
\end{aligned} \tag{2}$$

Then solve the general equation:

$$\begin{aligned}
2\ln|x| - 2\ln|u| &= t + C \\
\ln|x| - \ln|1-2x| &= \frac{t+C}{2} \\
\ln|\frac{x}{1-2x}| &= \frac{t+C}{2} \\
|\frac{x}{1-2x}| &= e^{\frac{t+C}{2}} \\
|\frac{x}{1-2x}| &= e^{\frac{t}{2}} \cdot e^{\frac{C}{2}} \\
\frac{x}{1-2x} &= \pm Ce^{\frac{t}{2}} \\
x &= Ce^{\frac{t}{2}} \cdot (1-2x) \\
x &= Ce^{\frac{t}{2}} - 2x \cdot Ce^{\frac{t}{2}} \\
x + 2x \cdot Ce^{\frac{t}{2}} &= Ce^{\frac{t}{2}} \\
x \cdot (1 + 2 \cdot Ce^{\frac{t}{2}}) &= Ce^{\frac{t}{2}} \\
x &= \frac{Ce^{\frac{t}{2}}}{1 + 2 \cdot Ce^{\frac{t}{2}}} \\
x &= \frac{1}{Ce^{-\frac{t}{2}} + 2}
\end{aligned} \tag{3}$$

Plug in x(0) = 1 to the equation, and solve for C

$$\begin{aligned}
1 &= \frac{1}{Ce^{-\frac{0}{2}} + 2} \\
1 &= \frac{1}{C+2} \\
C + 2 &= 1 \\
C &= -1
\end{aligned} \tag{4}$$

Plug in C in equation 3:

$$\begin{aligned}
x &= \frac{1}{-1e^{-\frac{t}{2}} + 2} \\
x &= -\frac{1}{e^{-\frac{t}{2}} - 2}
\end{aligned} \tag{5}$$

b & c

```python
import matplotlib.pyplot as plt

class Euler:
    def __init__(self, x, y, y_prime):
        self.x = x
        self.y = y
        self.y_prime = y_prime

    def plot(self, method, interval, steps, return_arr=False):
        if method == "forward euler":
            arr = self.forward_euler(interval, steps)
        elif method == "midpoint euler":
            arr = self.midpoint_euler(interval, steps)
        else:
            raise Exception(f"{method} is not a valid method")
        plt.plot(arr[0], arr[1], label=method)
        if return_arr:
            return arr


    def forward_euler(self, interval, steps):
        plot_arr = [
            [self.x],
            [self.y]
        ]
        h = (interval[1]-interval[0])/steps
        for i in range(0, steps):
            plot_arr[0].append((i+1)*h)
            plot_arr[1].append(plot_arr[1][i] + self.y_prime(plot_arr[0][i], plot_arr[1][i])*h)

        return plot_arr

    def midpoint_euler(self, interval, steps):
        plot_arr = [
            [self.x],
            [self.y]
        ]
        h = (interval[1]-interval[0])/steps
        for i in range(0, steps):
            k_1 = self.y_prime(plot_arr[0][i], plot_arr[1][i])
            k_2 = self.y_prime(plot_arr[0][i] + (h/2), plot_arr[1][i] + (h/2)*k_1)
            plot_arr[1].append(plot_arr[1][i] + k_2*h)
            plot_arr[0].append((i+1)*h)

        return plot_arr
```

For task b and c, I created a class Euler that implements the forward euler method, the midpoint euler method, and a plot method.

```
from Euler import Euler

import matplotlib.pyplot as plt
import numpy as np
from numpy import cos

eq = Euler(0, 1, lambda x,y: y * (0.5 - y))

eq.plot("forward euler", [0,3], 6)

analytic = lambda x: -1/(np.exp(-x/2) - 2)

x = np.linspace(0, 3, 101)
y = np.vectorize(analytic)

plt.plot(x, y(x), label="analytic")
plt.legend()
plt.savefig("images/task_2b.png")

eq.plot("midpoint euler", [0,3], 6)

plt.legend()
plt.savefig("images/task_2c.png")
plt.show()
```

This is the main program that uses the Euler class to approximate the curve of the differential equation using the two method that are available.

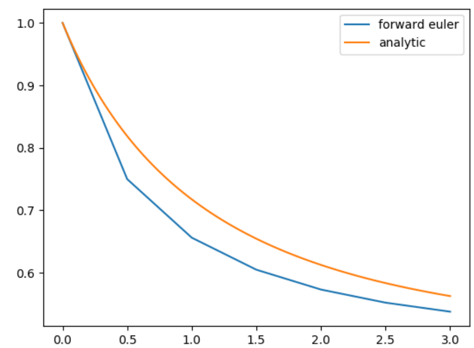

Figure 3: analytical and forward euler plot

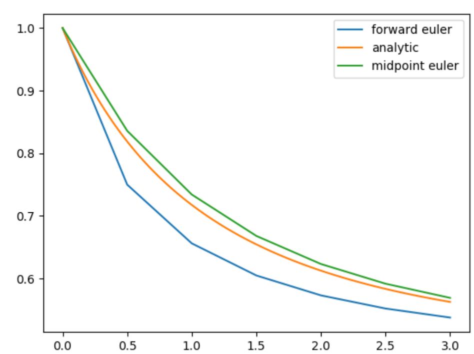

Figure 4: analytical, forward euler, and midpoint euler plot

Those are the plots that are created when I run my program, and we can clearly see that the midpoint euler method is far more precise than the forward euler method, but at a cost of computing power.

d

We know that $x(0) = 1$ is the upper limit for the function if we can prove that $x(t) = \frac{1}{2-e^{-\frac{t}{2}}}$ is monotonically decreasing (Which looks like is the case in the plots made earlier).
The only term that changes in $x(t)$ is $e^{-\frac{t}{2}}$. If we define a function $y(t) = e^{-\frac{t}{2}} = \frac{1}{e^{\frac{t}{2}}}$ and a number $h > 0$ we can see that $y(t+h) \leq y(t)$ where $t \geq 0$.
We replace $e^{-\frac{t}{2}}$ in $x(t)$ with $y(t)$ so that $x(t) = \frac{1}{2-y(t)}$. Since we know that $y(t)$ decreases as t gets smaller, the denominator of $x(t)$ will get bigger, which means that $x(t)$ decreases as t gets larger. This means that $x(t)$ is monotonically decreasing when $t \geq 0$.

To prove that $x(t) \geq \frac{1}{2}$ for $t \geq 0$, we have to look at what happens when $t \to \infty$.

$$\begin{aligned}
&\lim_{t\to\infty} x(t) \\
&= \lim_{t\to\infty} \frac{1}{2-e^{-\frac{t}{2}}} \\
&= \lim_{t\to\infty} \frac{1}{2-e^{-\frac{1}{2}t}} \\
&= \lim_{t\to\infty} \frac{1}{2-e^{-\frac{1}{2}\infty}} \\
&= \lim_{t\to\infty} \frac{1}{2-0} \\
&= \frac{1}{2}
\end{aligned} \tag{6}$$

This means that no matter how big t is, $x(t) \geq \frac{1}{2}$.

This limit also applies for the forward euler method for this equation. this is because $x(t+h) = y(t) + y(t)(\frac{1}{2} - y(t))$ where $h > 0$, will decrease less and less as $y(x) \rightarrow \frac{1}{2}$. In other words:

$$\begin{aligned}
&\lim_{x \to \frac{1}{2}} y(t) + y(t)(\frac{1}{2} - y(t)) \\
=&\lim_{x \to \frac{1}{2}} \frac{1}{2} + \frac{1}{2}(\frac{1}{2} - \frac{1}{2}) \\
=&\frac{1}{2} + \frac{1}{2}(0) \\
=&\frac{1}{2} + 0 \\
=&\frac{1}{2}
\end{aligned} \tag{7}$$

Which means that $x(t) \geq \frac{1}{2}$