Proejct 2: Swimming Pool Modeling

Wafik Aboualim

February 4, 2022

Part I- Modeling :

1- Stage 1 Model:

We start by analyzing the problem:
At $t = 0$, we have a swimming pool of v_0 ounces of solution of a certain concentration c_0, we can easily, using these 2 quantities, calculate the starting amount of chlorine $Q_0 = c_0 v_0$.

Now we apply our knowledge that $Q' = Q_{in} - Q_{out}$:

$$Q' = Q_{in} - Q_{out} = c_{in} r_{in} - c_{out} \frac{Q}{v} \tag{1}$$

Looking at our problem closely, c_{in} = 1e-7 ounce/gallon, and $r_{in} = 5$ gallon/min. The starting volume of the pool is $84,000$ gallons and the starting concentration of chlorine is 1.3e-5, thus the starting amount of chlorine:

$$Q_0 = c_0 v_0 = 84,000 \text{ . } 1.3\text{e-}5 = 1.092 \tag{2}$$

Now we shift our attention to the change in volume, the pool is supplied at a rate of 5 gallons/min and has and the mixture drains at the same rate. Thus our knowledge that $v' = v_{in} - v_{out}$ and that $v(0) = 84,000$:

$$v'(t) = 5 - 5 = 0 \implies v = 84,000 \tag{3}$$

Now we are ready to deal with the O.D.E (1), substituting the values we computed:

$$Q'(t) = c_{in} r_{in} - c_{out} \frac{Q}{v} = 5\text{e-}7 - \frac{Q}{16800},\ Q(0) = Q_0 = 1.092 \tag{4}$$

Now we solve our O.D.E using the integrating factor technique, we can write the O.D.E in the form $Q' + p(t)Q = f(t)$, where $p(x) = \frac{1}{16800}$, $f(x) = 5\text{e-}7$:
let $I(t)$ be our integrating factor such that:

$$I(x) = \int e^{\frac{1}{16800}} dt \implies I(t) = e^{\frac{1}{16800} t} \tag{5}$$

Now we just use the formula $Q = \frac{1}{I(t)} \int I(t)p(t)dx$, we get:

$$Q = \frac{21}{2500} + c_1 e^{\frac{-t}{16800}} \tag{6}$$

Now we just apply our initial condition $Q(0) = 1.092$ to compute c_1, we get:

$$c_1 = 1.0836 \implies Q = \frac{21}{2500} + 1.0836\ e^{\frac{-t}{16800}}) \tag{7}$$

The solution has a negative exponential behavior which makes sense physically since the chlorine levels are decreasing, and if we plug $t = 0, Q = Q_0 = 1.092$, which models the problem correctly.

2- Stage 2 Model:

Now we try to model the second stage, We model the second stage by an O.D.E $Q_2'(t_2, Q_2)$ at the start of the second stage $t_2 = 0$. $t_2 = 0$ at $t = 24 * 60$ min, that means that $Q_2(0) = Q(1440)$. Referring to Q(t) from Stage 1:

$$Q(1440) = 1.002989237 \approx 1.003 \implies Q_2(0) = Q(1440) \approx 1.003, \text{which is our initial condition} \tag{8}$$

Now we have just have to solve for Q_2' just like we did in Stage 1, We have $c_{in} = 5e{-}5$ ounces/gallon, $r_{in} = 5$ gallon/min, using those info to solve for Q_2' just like we did in Stage 1, we get:

$$Q_2' = c_{in} r_{in} - c_{out} \frac{Q}{v} = 25\text{e-}5 - \frac{Q_2}{16800} = \frac{1}{4000} - \frac{Q_2}{16800} \tag{9}$$

Solving the I.V.P using the integrating factor method we used in Stage 1, we get:

$$Q_2 = 4.2 - 3.197\ e^{\frac{-t}{16800}} \tag{10}$$

The solution to the I.V.P makes sense physically because of the exponential growth pattern.

Now to compute the time needed for the concentration to reach a minimum of 1.3e-5 ounces/gallon, we just set $Q_{min} = c_{min}\ .\ v$ = 1.3e-5 . 84000 = 1.092 ounces. Next we just calculate t at $Q_2 = Q_{min} = 1.092$:

$$1.092 = 4.2 - 3.197\ e^{\frac{-t}{16800}} \implies t \approx 474.32 \text{ min} \approx 19.7 \text{ hours} \tag{11}$$

Part II- Numerical Methods :

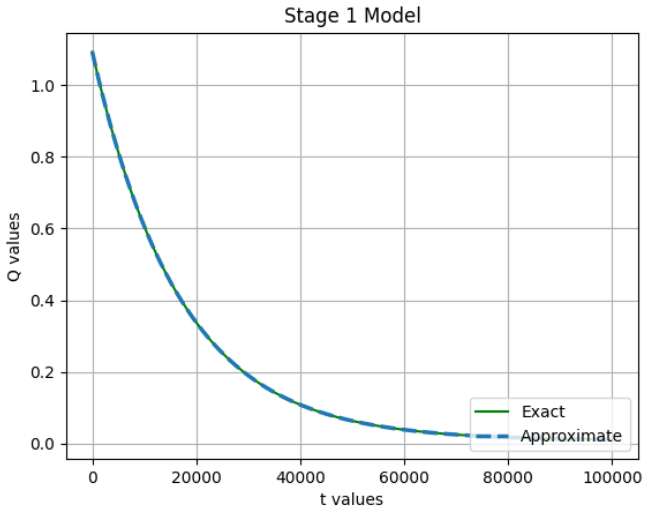

Figure 1: $h = 0.1,\ 0 \leq t \leq 100000$

```python
from re import I
from matplotlib import pyplot as plt
import numpy as np
import math
#author : Wafik Aboualim, Date : 2/2/2022
#first we define out function f(Q,t)
f1 = lambda Q,t: (5e-7) - (Q/16800)
#next we choose a stepsize 'h'
h = 0.1
#Now use np.arange to get t values, t_i+1 = t_i + h
#t starts at 0 --> 100000 , initialize t values array
t_vals = np.arange(0,100000,h)
#Now we choose our initial condition Q(0) = 1.092
Q0 = 1.092
#Now we initialize our Q values array
Q_vals = np.zeros(len(t_vals))
#first value in the Q values array is the initial condition
Q_vals[0] = Q0
#Now we use our recursive Euler method to calculate the rest of the Q values and store them in the Q_vals array
for i in range(0, len(t_vals)-1):
    Q_vals[i+1] = Q_vals[i] + (h*f1(Q_vals[i], t_vals[i]))
Q1_exact_values = np.zeros(len(t_vals))
g = lambda t: 0.0084 + (1.0836)*(pow(math.e,(-t/16800)))
for i in range(len(t_vals)):
    Q1_exact_values[i] = g(t_vals[i])
plt.plot(t_vals,Q1_exact_values,'g', label='Exact')
plt.plot(t_vals,Q_vals,'--', label='Approximate', linewidth = 2.5)
plt.title('Stage 1 Model')
plt.xlabel('t values')
plt.ylabel('Q values')
plt.grid()
plt.legend(loc='lower right')
plt.show()
```

Figure 2: Stage 1 Python script

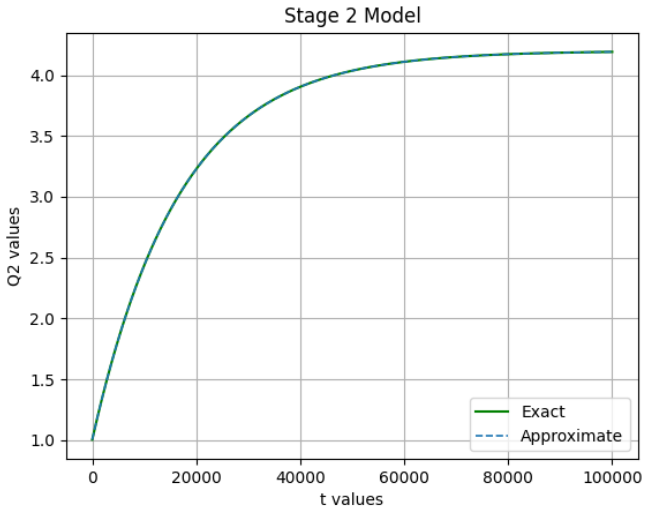

Figure 3: $h = 0.1,\ 0 \leq t \leq 100000$

```python
from re import I
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import math
#author : Wafik Aboualim, Date : 2/2/2022
#first we define out function f(Q,t)
f1 = lambda Q,t: 1/4000 - (Q/16800)
#(5e-7) - (Q/16800)
#next we choose a stepsize 'h'
h = 0.1
#Now use np.arange to get t values, t_i+1 = t_i + h
#t starts at 0 --> 100000 , initialize t values array
t_vals = np.arange(0,100000,h)
#Now we choose our initial condition Q(0) = 1.092
Q0 = 1.003
#Now we initialize our Q values array
Q_vals = np.zeros(len(t_vals))
#first value in the Q values array is the initial condition
Q_vals[0] = Q0
#Now we use our recursive Euler method to calculate the rest of the Q values and store them in the Q_vals array
for i in range(0, len(t_vals)-1):
    Q_vals[i+1] = Q_vals[i] + (h*f1(Q_vals[i], t_vals[i]))
Q1_exact_values = np.zeros(len(t_vals))
g = lambda t: 4.2 - (3.197*pow(math.e,-t/16800))
#0.0084 + (1.0836)*(pow(math.e,(-t/16800)))
for i in range(len(t_vals)):
    Q1_exact_values[i] = g(t_vals[i])
plt.plot(t_vals,Q1_exact_values,'g', label='Exact')
plt.plot(t_vals,Q_vals,'--', label='Approximate', linewidth = 1.2)
plt.title('Stage 2 Model')
plt.xlabel('t values')
plt.ylabel('Q2 values')
plt.grid()
plt.legend(loc='lower right')
plt.show()

```

Figure 4: Stage 2 Python script

Now we make a table with values $0.01 \leq h \leq 0.1$ to find the max error over the period $1 \leq t \leq 2880$ in both stages 1,2 :

[52]:
```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import math
#author : Wafik Aboualim, Date : 2/2/2022
#first we define out function f(Q,t)
h = 0.1
print("h  ||","   Q  ||","   Q_approx ||","   error " )
for i in range(10):
    f1 = lambda Q,t: (5e-7) - (Q/16800)
#
#next we choose a stepsize 'h'

#Now use np.arange to get t values, t_i+1 = t_i + h
#t starts at 0 --> 100000 , initialize t values array
    t_vals = np.arange(0,2880+h,h)
    n = 2880/h
#Now we choose our initial condition Q(0) = 1.092
    Q0 = 1.092
#Now we initialize our Q values array
    Q_vals = np.zeros(len(t_vals))
#first value in the Q values array is the initial condition
    Q_vals[0] = Q0
#Now we use our recursive Euler method to calculate the rest of the Q values and store them in the Q_vals array
    for i in range(0, len(t_vals)-1):
        Q_vals[i+1] = Q_vals[i] + (h*f1(Q_vals[i], t_vals[i]))
    Q_exact_vals = np.zeros(len(t_vals))
    g = lambda t: 0.0084 + (1.0836)*(pow(math.e,(-t/16800)))
#
    for i in range(len(t_vals)):
        Q_exact_vals[i] = g(t_vals[i])
    print(h,"||",Q_exact_vals[-1],"||",Q_vals[-1],"||",Q_exact_vals[-1]-Q_vals[-1])
    print("-----------------------------------------------------------------")
    h -= 0.01
#pd.DataFrame({"t":t_vals,"Q":Q_exact_vals,"Q_approx":Q_vals})
```

```
h  ||    Q  ||    Q_approx ||    error
0.1 || 0.9212901345359333 || 0.9212896687739441 || 4.6576198919900946e-07
-----------------------------------------------------------------
0.09000000000000001 || 0.9212901345359333 || 0.9212897153502977 || 4.1918563564724565e-07
-----------------------------------------------------------------
0.08000000000000002 || 0.9212901345359333 || 0.921289761926605 || 3.726093282807596 6e-07
-----------------------------------------------------------------
0.07000000000000002 || 0.9212895911491102 || 0.9212892651151339 || 3.260339763720886 6e-07
-----------------------------------------------------------------
0.06000000000000002 || 0.9212901345359333 || 0.92128985507916 || 2.7945677338888087e-07
-----------------------------------------------------------------
0.05000000000000002 || 0.9212901345359333 || 0.921289901655377 || 2.328805562834546e-07
-----------------------------------------------------------------
0.04000000000000015 || 0.9212901345359333 || 0.9212899482315506 || 1.8630438269884309e-07
-----------------------------------------------------------------
0.03000000000000013 || 0.9212901345359333 || 0.9212899948076847 || 1.3972824863817124e-07
-----------------------------------------------------------------
0.02000000000000001 || 0.9212901345359333 || 0.9212900413838013 || 9.315213200800088e-08
-----------------------------------------------------------------
0.01000000000000001 || 0.9212901345359332 || 0.9212900879598929 || 4.657604035784857e-08
-----------------------------------------------------------------
```

Figure 5: Stage 1 IPython script

Looking at the table, we can notice that we achieve the maximal error when h is at a maximum ($h = 0.1$)

[50]:
```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import math
#author : Wafik Aboualim, Date : 2/2/2022
#first we define out function f(Q,t)
h = 0.1
print("h  ||","   Q  ||","   Q_approx ||","   error " )
for i in range(10):
    f1 = lambda Q,t: 1/4000 - (Q/16800)
#(5e-7) - (Q/16800)
#next we choose a stepsize 'h'

#Now use np.arange to get t values, t_i+1 = t_i + h
#t starts at 0 --> 100000 , initialize t values array
    t_vals = np.arange(0,2880+h,h)
    n = 2880/h
#Now we choose our initial condition Q(0) = 1.092
    Q0 = 1.003
#Now we initialize our Q values array
    Q_vals = np.zeros(len(t_vals))
#first value in the Q values array is the initial condition
    Q_vals[0] = Q0
#Now we use our recursive Euler method to calculate the rest of the Q values and store them in the Q_vals array
    for i in range(0, len(t_vals)-1):
        Q_vals[i+1] = Q_vals[i] + (h*f1(Q_vals[i], t_vals[i]))
    Q_exact_vals = np.zeros(len(t_vals))
    g = lambda t: 4.2 - (3.197*pow(math.e,-t/16800))
#0.0084 + (1.0836)*(pow(math.e,(-t/16800)))
    for i in range(len(t_vals)):
        Q_exact_vals[i] = g(t_vals[i])
    print(h,"||",Q_exact_vals[-1],"||",Q_vals[-1],"||",Q_vals[-1]-Q_exact_vals[-1])
    print("-----------------------------------------------------------------------")
    h -= 0.01
#pd.DataFrame({"t":t_vals,"Q":Q_exact_vals,"Q_approx":Q_vals})
```

```
h  ||    Q  ||    Q_approx ||    error
0.1 || 1.506653968151182 || 1.5066553423124185 || 1.3741612365780753e-06
-----------------------------------------------------------------------
0.09000000000000001 || 1.506653968151182 || 1.5066552048958488 || 1.2367446668992699e-06
-----------------------------------------------------------------------
0.08000000000000002 || 1.506653968151182 || 1.5066550674793693 || 1.099328187370574e-06
-----------------------------------------------------------------------
0.07000000000000002 || 1.5066555713328666 || 1.5066565332474322 || 9.619145655559436e-07
-----------------------------------------------------------------------
0.06000000000000002 || 1.506653968151182 || 1.5066547926467109 || 8.244955289615774e-07
-----------------------------------------------------------------------
0.05000000000000002 || 1.506653968151182 || 1.5066546552305218 || 6.870793398672248e-07
-----------------------------------------------------------------------
0.04000000000000015 || 1.506653968151182 || 1.5066545178144548 || 5.496632728974049e-07
-----------------------------------------------------------------------
0.03000000000000013 || 1.506653968151182 || 1.5066543803984789 || 4.1224729696587303e-07
-----------------------------------------------------------------------
0.02000000000000001 || 1.506653968151182 || 1.5066542429825818 || 2.748313998601759e-07
-----------------------------------------------------------------------
0.01000000000000001 || 1.5066539681511824 || 1.506654105566846 || 1.3741566373681735e-07
-----------------------------------------------------------------------
```

Figure 6: Stage 2 IPython script

We can notice the same behavior around $h = 0.1$, which is trivial given that 0.1 is the maximum step-size

Most differential equations that govern real world problems are unsolvable analytically, that revolutionized the use of numerical methods to solve these types of problems. Using the numerical approach is the go-to to solve differential equations for scientists in the present day.

In this project, I split my work into 3 stages:
1- constructing the models 1,2 of the swimming pool.
2- using Euler method to solve the models numerically.
3- calculating the error in calculating the chlorine levels as the step size decreases.

1- Constructing models :

in this part I read the statement of the problem and used my perquisite knowledge of the mixing problem we took in class to express the problems of stages 1 as an I.V.P of an O.D.E $Q'(Q,t)$. Then I used the stage 1 model to compute the initial value of the chlorine levels at stage 2.
Hence, all what is left to do is to solve the 2 I.V.Ps of the stages 1,2. I solved them analytically using the integrating factor method we took in class and then used the initial values to compute the constants.

2- Euler's Method :

After constructing the models and solving them analytically, the next step is to develop a script that solves the I.V.Ps numerically, then contrasts both the numerical and the analytic solutions on a plot (Figures 2,3,4,5). For this project, I used a Python script using graphical and mathematical libraries (Matplotlib, Numpy).

3-Calculating the error :

This part was certainly the most challenging. In this part I used the data sets, that of Q exact and Q approximated values, that I calculated using the IPython script. I then subtracted the Q exact values from the Q approximated values to form a new data set which corresponds to the error. I used the last cell of the error data set and varied it with the step-size (h) to create the table in Figures 5,6.

Problems I faced :

· I faced some miscalculations that caused to redo some of the parts of the project. Most of the miscalculations were minor but some were caused me to redo some of the parts of the project.

· I had to take some time to write the python script to do the graphing part but it was worthwhile.

· Sense my choice for the step size was very small ($h = 0.1$) and I varied t from 1 to 100000, that caused the approximations to be really accurate. That caused some visual problems with the graph. To explain, at first sight, looking at the plots, we can see that the both the

approximated and the analytic solution overlap, which is certainly not the case, you just have to zoom enough to see the contrast between both solutions.