ME 2450 Assignment HW1b

Name: Christopher Wall

Submit this assignment as a single PDF file to **gradescope**.

I declare that the assignment here submitted is original except for source material explicitly acknowledged.

I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the University website.

Christopher Wall	9/1/2024
Name	Date
Chris Wall	u1467634
Signature	Student ID

Homework Formatting Tips
(These guidelines apply to this and all future HW assignments)

- Axes Labels and Legends on Plots:
  Please ensure that all your plots have clearly labeled axes and legends if applicable. This is crucial for understanding your visual representations. A plot without labeled axes does not mean anything!

- Avoid Printing and Scanning Code:
  Kindly refrain from printing out your code and then scanning it. Instead, export your code electronically in a readable format, such as a text document or pdf/html file. Codes that are printed then scanned are very hard to read and will result in reduced or zero credit.

- Proper Cropping of Screenshots:
  If you need to include screenshots in your submissions, please crop them to only contain the relevant regions. Including unnecessary parts of the screen makes it harder to assess your work efficiently. Also make sure that screenshots included are high quality and high resolution.

- Include Plots and Tables:
  It is imperative that you include the actual plots and tables with your submission. Merely providing the code used to generate them is not sufficient. Likewise, having the final plots or tables without the corresponding code is incomplete. Both components are necessary for a comprehensive evaluation of your work.

- Explain Your Process:
  It can be challenging for me to understand how you generated the plot or table. Please make sure to provide a brief explanation of your process in forms of comments, even if you encounter difficulties or errors in your code.

Q1 (2 pt + bonus 1 pt)

Express the *unsigned* binary number 11111111 in base-10. (Show your work)

1 bonus point: Write a piece of computer code that takes as input a base-2 integer and prints its base-10 representation. Prove that it's working by running 3 other (not 11111111) base-2 numbers as input and showing printed output. Include a screenshot of the outputs, AND your code, in your submission.

Q2 (2pt + bonus 1 pt)

Express the base-10 number 11111111 as a *signed* binary number. How many bits are required to do so? (Show your work)

1 bonus point: Write a fully commented computer code that takes as input a base-10 integer and prints its base-2 representation. Prove that it's working by running 3 other (not 11111111) base-10 numbers as input and showing printed output. Include a screenshot of the outputs, AND your code, in your submission.

Q3 (5pt)

Consider the function

$$f(x) = \frac{5x}{(1-2x^2)^2}$$

a) Evaluate the function at x = 0.423 using 3-digit arithmetic with chopping. Report the value obtained and the true relative error. (Turn in hand calculations only. No computer code is required.)

b) Repeat part (a) except using 4-digit arithmetic with chopping.

   HINT: Consider each arithmetic operation separately. First, calculate the numerator, i.e., $5x = (5) * (0.423)$and chop the answer to specified number of significant digits. Then, calculate $x^2 = (0.423) * (0.423)$and chop the answer to the specific number of significant digits, and so forth. After each operation, the resulting answer can only have 3 significant digits for part (a), and 4 significant digits for part (b).

Q4 (6 pts)

The quantity e^{-5} may be determined using the following two different formulas

$$\text{Formula 1 :} e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots$$
$$\text{Formula 2 :} e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots}$$

a) Write a *fully commented* computer code to evaluate formulas 1 and 2 using 1–20 terms. Copy and paste your code as text and include it in your pdf submission.

b) In your code, calculate the *true* relative error (ϵ_t) and *approximate* relative error (ϵ_a) of each calculation.

c) Report your results in a table that looks something similar to the one shown below. (Be sure to clearly label the columns of the table) *Hint: While it is acceptable for you to manually make the table, it will save you a lot of time if you can make the computer to generate the table.*

d) Comment on which formula has the least error.

	Formula 1			Formula 2		
terms	value	ϵ_t (%)	ϵ_a (%)	value	ϵ_t (%)	ϵ_a (%)
1						
.						
.						
.						
20						

Q5 (8 pts)

Recall the cylindrical storage tank from Assignment HW 1a:

Solve the cylindrical-tank IVP using your implementation of Euler's method from Assignment 1a using, $h = 1, 0.1, 0.01, 0.001$. Note, this will result in 4 different numerical approximations (one for each h) of the solution, $y(t)$.

a) At a 1-second interval (i.e. at $t = 0, 1, 2, \ldots, 10$ seconds) compute the approximate relative error between each refinement of h. This will result in three data sets: the approximate error between $h = 1 \rightarrow 0.1$, $h = 0.1 \rightarrow 0.01$, and $h = 0.01 \rightarrow 0.001$. Plot all three of these data sets together. Provide a statement regarding what information each data set provides.

b) Plot the approximate relative errors at $t = 2$ as a function of h. Provide a statement regarding what information this data set provides.

Q1: 11111111 in base 10

$(1)2^0 + (1)2^1 + (1)2^2 + (1)2^3 + (1)2^4 + (1)2^5 + (1)2^6 + (1)2^7$

or: $2^8 - 1$

$\boxed{11111111 \text{ in base } 10 = 255}$

Q2: 11111111 as a signed binary number

		remainder
11111111/2	= 5555555	1
5555555/2	= 2777777	1
2777777/2	= 1388888	1
1388888/2	= 694444	0
694444/2	= 347222	0
347222/2	= 173611	0
173611/2	= 86805	1
86805/2	= 43402	1
43402/2	= 21701	0
21701/2	= 10850	1
10850/2	= 5425	0
5425/2	= 2712	1
2712/2	= 1356	0
1356/2	= 678	0
678/2	= 339	0
339/2	= 169	1
169/2	= 84	1
84/2	= 42	0
42/2	= 21	0
21/2	= 10	1
10/2	= 5	0
5/2	= 2	1
2/2	= 1	0
1/2	= 0	1

$\boxed{11111111 \text{ in Base } 2 \text{ is: } 01010100110001\ 010\ 11000111}$

sign

Q3

a) $f(0.423) = \dfrac{5(.423)}{(1-2(.423)^2)^2}$

Chopping

$5(.423) = 2.11$

$.422^2 = 0.178$

$2(.178) = 0.356$

$1 - .356 = 0.644$

$(0.644)^2 = .414$

$\dfrac{2.11}{.414} = 5.09$

(No rounding)

$f(0.423) = 5.12918$
no chopping

a) difference: $\simeq 0.03256$

b) $5(0.423) = 2.115$

$0.423^2 = 0.1789$

$1 - 2(0.1789) = 0.6422$

$0.6422^2 = 0.4124$

$f(0.423) \simeq 5.128$

difference: 0.0006673

Q4

The second formula has the least error. I believe this is from avoiding the subtraction error from the first formula. Additionally the second formula only has one more operation than the first formula for any number of terms

Q5

a) the relative errors follow the magnitude of the step size - for a 10x reduction in step size there is ~10% the approximate error.

b) as step sizes become smaller, the final value converges.

- Also notable is how the graph follows a "stepping" pattern. This is likely from rounding or truncation errors.
- Additionally, the general trend is that as h values increase, the value of the y-level decreases. This means that for large h-values the simulation is likely to underestimate the final y-value.

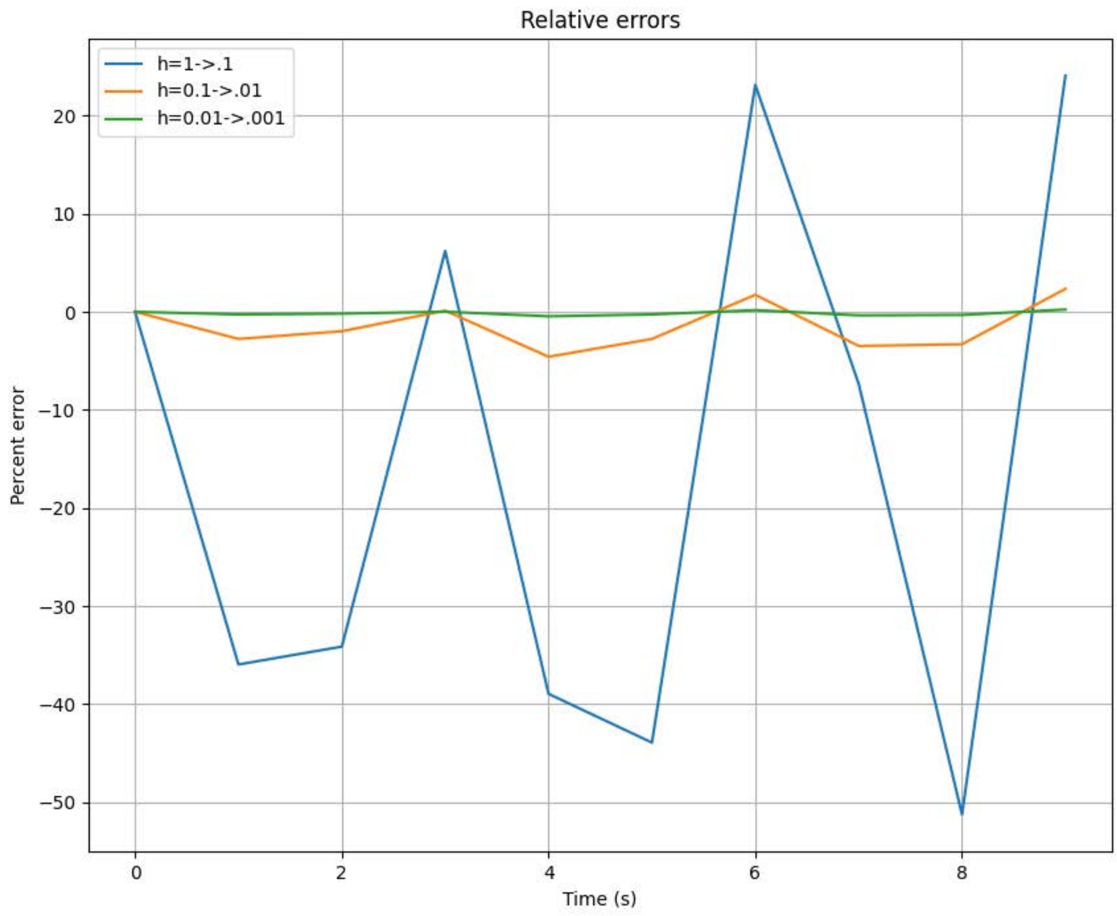
```
#Bonus problems

#Converting binary to decimal
number = 0b11111111
numberStr = "11111111"
origStr = numberStr
output = 0
while len(numberStr) > 0:
    output += int(numberStr[0]) * 2**(len(numberStr) - 1)
    numberStr = numberStr[1 : len(numberStr)]

print("Converting " + origStr + " to decimal yields %d" % output)

#Converting decimal to binary
#follows the remainder formula for converting decimal to binary
number = 11111111
sign = "0" if number > 0 else "1"
output = ""
while abs(number) > 0:
    output += str(number % 2)
    number = int(number/ 2)



print("Converting " + origStr + "in decimal to singed binary yields " + sign + output)
```

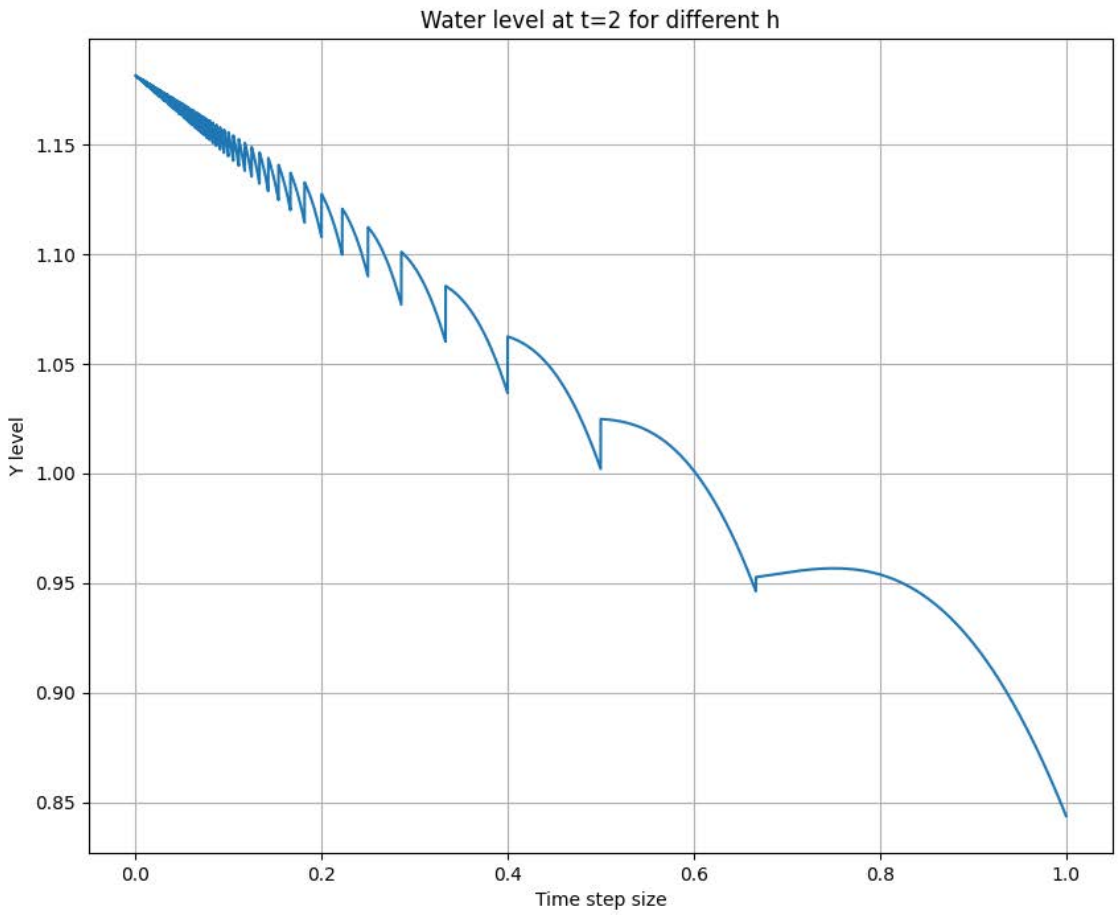
```
Converting 11111111 to decimal yields 255
Converting 11111111in decimal to singed binary yields 0111000110101000110010101
```

Expected value: 0.006737946999085469

Terms	Formula 1 Value	Formula 1 true error %	Formula 1 approx error %	Formula 2 Value	Formula 2 true error %	Formula 2 approx error %
1	1.000000	99.326205	100.000000	1.000000	99.326205	100.000000
2	-4.000000	100.168449	125.000000	0.166667	95.957232	500.000000
3	8.500000	99.920730	147.058824	0.054054	87.534798	208.333333
4	-12.333333	100.054632	168.918919	0.025424	73.497408	112.612613
5	13.708333	99.950848	189.969605	0.015296	55.950671	66.207627
6	-12.333333	100.054632	211.148649	0.010939	38.403935	39.834289
7	9.368056	99.928075	231.653076	0.008840	23.781654	23.738985
8	-6.132937	100.109865	252.749919	0.007775	13.337167	13.703376
9	3.555184	99.810475	272.506889	0.007230	6.809363	7.532415
10	-1.827105	100.368777	294.580103	0.006959	3.182806	3.891547
11	0.864039	99.220180	311.460966	0.006832	1.369527	1.872889
12	-0.359208	101.875777	340.539772	0.006775	0.545309	0.835662
13	0.150478	95.522306	338.711501	0.006752	0.201885	0.345307
14	-0.045555	114.790730	430.320215	0.006743	0.069799	0.132353
15	0.024457	72.449452	286.269025	0.006739	0.022625	0.047207
16	0.001119	501.935742	2084.841268	0.006738	0.006901	0.015728
17	0.008412	19.903471	86.693508	0.006738	0.001987	0.004914
18	0.006267	7.509364	34.224748	0.006738	0.000542	0.001445
19	0.006863	1.824096	8.681532	0.006738	0.000140	0.000401
20	0.006706	0.471284	2.338029	0.006738	0.000035	0.000106

```python
import numpy as np
import pandas as pd

formula1Data = []       #list of values from formula 1
formula1RelErrors = []  #relative errors from formula 1`
formula2Data = []       #list of values from formula 2
formula2RelErrors = []  #relative errors from formula 2

valueToFind = 5         #value in -exponent to find
trueValue = np.e**-5    #value to compare to

#factorial formula, could have use the gamma function in the numpy library, but
# this is probably more resource efficient
def factorial(value):
    result = 1
    while value > 0:
        result *= value
        value -= 1
    return result

#formula 1:
previousValue = 0       #Used to add up terms in series
sign = 1                #sign of current term, either 1 or 01
for i in range(1,21):
    #find next term in series
    term = sign * ((valueToFind**(i-1))/factorial(i - 1))
    #add term to previous term
    value = previousValue + term
    #store term
    formula1Data.append(value)
    formula1RelErrors.append(100 * np.abs((value - previousValue)/value))
    #update previous value and sign
    previousValue = value
    sign *= -1


#formula 2:
previousValue = 0       #set to zero to not preserve data from above
for i in range(1, 21):
    #find next term in demoninator
    term = (valueToFind**(i-1)/factorial(i-1))
    #add next term to previous denominator
    denominator = previousValue + term
    #store 1/denominator
    formula2Data.append(denominator**-1)
    #handles first iteration where I can't take 0**-1
    previousValueInverse = 0 if previousValue == 0 else previousValue**-1
    formula2RelErrors.append(100 * np.abs(denominator**-1 - previousValueInverse)/(denominator**-1))
    #update values
    previousValue = denominator

#plotting

data = {
    ('Formula 1', 'Value'): formula1Data,
    ('Formula 1', 'true error %'): [np.abs((x - trueValue)/x) * 100 for x in formula1Data],
    ('Formula 1', 'approx error %'): formula1RelErrors,
    ('',''): '',
    ('Formula 2', 'Value'): formula2Data,
    ('Formula 2', 'true error %'): [np.abs((x - trueValue)/x) * 100 for x in formula2Data],
    ('Formula 2', 'approx error %'): formula2RelErrors
}

dataFrame = pd.DataFrame(data)

dataFrame.index = range(1, len(formula1Data)+1)
dataFrame.index.name = 'Terms'

print("Expected value: " + str(np.e**-5))
print(dataFrame)

#Looking at the table, the second fomula has the least error. Also notable is how the
# error converges much much more quickly and consistently than the first formula
```


Relative errors
h=1->.1
h=0.1->.01
h=0.01->.001
Percent error
Time (s)


Water level at t=2 for different h
Y level
Time step size

```
import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

A = 850         # Area, m^2
Q = 325         # Flow rate m^3/s
alpha = 200     # Constant m^3/2 / s

# Stores time and water level values to be plotted
time_values = []
y_values = []

# Defines dy/dt
def forcingfunc(time, ylevel):
    return 2 * (Q/A) * math.sin(time)**2  - (alpha/A) * ((1 + ylevel)**(3/2))

def simulate(h, label):
    # Simulation parameters
    t = 0           # Time s
    y = 2           # Initial water level m
    maxTime = 10    # Simulation limit s

    times = []
    levels = []

    while t < maxTime:
        times.append(t)
        levels.append(y)
        y += forcingfunc(t, y) * h
        t += h

    time_values.append(times)
    y_values.append(levels)

    # Print the final time and level
    #print(f"Final Time: {t:.2f}, Final Y-level: {y:.2f}")

def simulate_no_store(h, maxTime):
    # Simulation parameters
    t = 0           # Time s
    y = 2           # Initial water level m

    while t < maxTime:
        y += forcingfunc(t, y) * h
        t += h

    return y

# Run simulations with different step sizes
simulate(1, 'h=1')
simulate(0.1, 'h=0.1')
simulate(0.01, 'h=0.01')
simulate(0.001, 'h=0.001')

# Plotting the water level over time
'''
plt.figure(figsize=(10, 8))
for i, label in enumerate(['h=1', 'h=0.1', 'h=0.01', 'h=0.001']):
    plt.plot(time_values[i], y_values[i], linestyle='-', label=label)
plt.title('Water Level Over Time Using Euler\'s Method')
plt.xlabel('Time (s)')
plt.ylabel('Water Level (m)')
plt.legend()
plt.grid(True)
plt.show()
'''

# Create a DataFrame for the results of the last simulation
data = {
    'Time': time_values[-1],
    'Water level': y_values[-1]
}

#Finding errors across different levels
def findErrors(level):
    errors1 = []
    #take 10 samples from data, second set will be 10 times as dense as the first
```

```
    for i in range(0, 10):
        value1 = y_values[level][i * 10**level]
        value2 = y_values[level + 1][i*10**(level + 1)]
        error = 100 * (value1 - value2)/value2
        errors1.append(error)
    return errors1


errors = {
    'h1/.1' : findErrors(0),
    'h.1/.01' : findErrors(1),
    'h.01/.001' : findErrors(2),
}

#Displaying chart of relative errors
'''
df = pd.DataFrame(data)
edf = pd.DataFrame(errors)
print(edf)
'''

#plotting relative errors

plt.figure(figsize=(10,8))
for i, label in enumerate(['h=1->.1', 'h=0.1->.01', 'h=0.01->.001']):
    plt.plot(range(0,10), findErrors(i),label=label)
plt.title('Relative errors')
plt.xlabel('Time (s)')
plt.ylabel('Percent error')
plt.legend()
plt.grid(True)
plt.show()

#displaying water level estimate in terms of h
print("Displaying y estimate in terms of h")
h_vals  = np.arange(.0001 , 1, .0001) #values from 0.001 to 1 in .05 increments
y_vals_at_2 = []

for h in h_vals:
    y_vals_at_2.append(simulate_no_store(h, 2))

plt.figure(figsize =(10,8))
plt.plot(h_vals, y_vals_at_2)
plt.xlabel('Time step size')
plt.ylabel('Y level')
plt.grid(True)
plt.title("Water level at t=2 for different h")
plt.show()
```