PROGRAMMING ASSIGNMENT 2

This programming assignment has two parts and is to be written in Python. As an applied mathematician, one should be familiar with more than one programming language, or at least be able to navigate Matlab and Python. So either sign up for a free account on CoCalc, or install Anaconda, or just use the SageMathCell at `https://sagecell.sagemath.org/`.

Problem 1. First we will write a short program to demonstrate the difference between machine epsilon and underflow. Recall the following basic facts:

- Underflow is related to the possible values the exponent can contain. IEEE double precision allocates 53 bits to the exponent, which can be used to store any (binary) exponent between -1022 and $+1023$. Numbers smaller in magnitude than 2^{-1022} cannot be stored exactly because it is impossible to represent their exponent.
- Machine epsilon is related to the number of bits used in the fraction. You can think of it roughly as the maximum relative representation error. Another way to think about it is as the smallest positive number that is both exactly represented and for which $1 + \epsilon \neq 1$ (as far as the computer is concerned). In other words, when adding 1 to numbers smaller than machine epsilon the result returned is 1 (Can you explain why?).

  As a base ten example, imagine you have six digits of precision. Then you can store the number 1.00000×10^0 and the number 1.00000×10^{-50} (assuming you have enough bits to represent the exponent -50). However, if you add those two numbers, the result will be rounded to 1. You can find machine epsilon in this case by writing down the number "after" 1 and then subtract 1:

$$1.00001 \times 10^0 - 1.00000 \times 10^0 = 0.00001 \times 10^0 = 10^{-5}$$

  You can find the machine epsilon $\epsilon = 2^{-53}$ in IEEE double precision in a similar way.

For this program, initialize a variable to 1.0, then run a `for` loop for, say, 1200 iterations, and have it print the following three numbers: the iteration number, x, and $(x + 1) - 1$. Finish the loop by updating x to $x/2$.

Can you describe what happens?

Problem 2. The second part is based on Question 1.9 in the text. Begin by reading the problem. The goal is to replicate the pictures at the bottom of page 25.

The biggest difference between the two approaches is in the sum $1 + x$. What's happening is that a relative small magnitude number, x, is being added to 1, and some of its digits are being discarded. This is like the base ten analog:

$$1 + 3.18812 \times 10^{-4} = 1.000318812 \approx 1.00032$$

In this rounding, several significant digits of x are lost. Instead of $1 + x$, the computer is finding $(1 + x)(1 + \delta)$ with $|\delta| \leq \epsilon$. Even if every other operation is computed with perfect accuracy, we still get

$$\frac{\log((1+x)(1+\delta))}{x} = \frac{\log(1+x)}{x} + \frac{\log(1+\delta)}{x} \approx 1 + \frac{\delta}{x}$$

Note that δ jumps around between the values $-\epsilon$ and $+\epsilon$. When x is large enough in magnitude, δ/x is still small, and you don't see this, but when x is on the order of ϵ, then δ/x can be around ± 1, which is the zigzag graph you see in the figure.

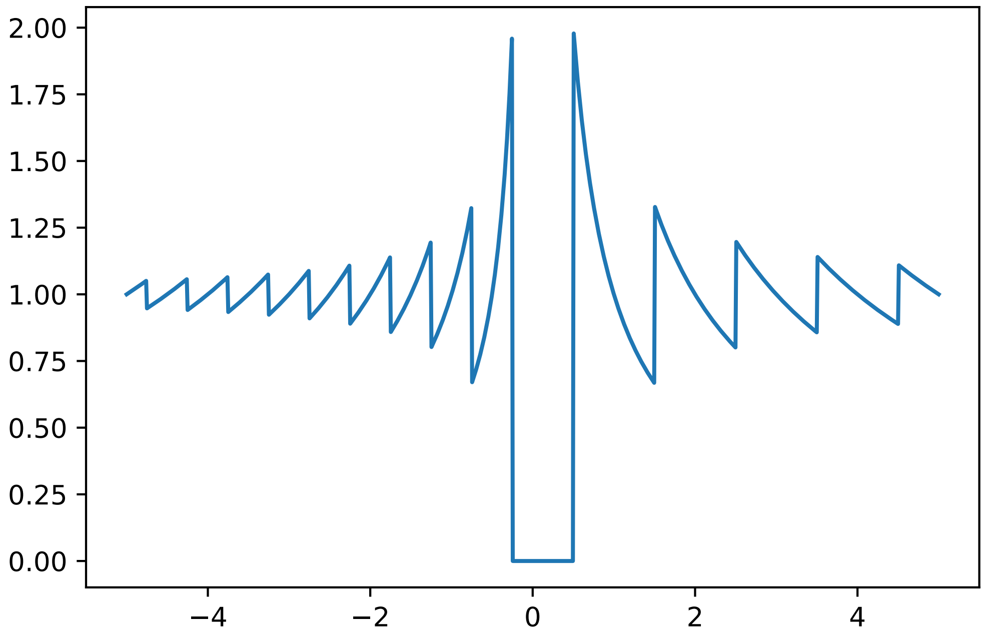
Another key point is that when $|x| < \epsilon$, d will evaluate to exactly 1 and the `if` statement in the alternative algorithm catches that and returns 1, thus avoiding the weird 0 behavior near $x = 0$.

In your program, implement both methods and plot them. First, by having x do 1000 equally spaced steps between -0.99999 and 1. And secondly, by doing the same amount of steps between $-5 *$ `eps` and $5 *$ `eps`, where `eps` is machine epsilon.

Note that in Python, you can get machine epsilon by using the command:

```
import numpy as np
eps = np.finfo(float).eps
```

You should get a picture like this:


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−4
−2
0
2
4