Assignment 1

Corrections. Exercise 1 has been edited. It used to say %, but now it says 2% for the relative accuracy. It used to talk about the distance from 1 to the smallest normalized number, which is "obviously" wrong (obvious if you understand everything perfectly). Now it talks about the distance between 1 and the next larger number.

1. The task is to design and understand an LPA (low precision arithmetic) floating point arithmetic standard that uses the fewest bits that can approximate any x with $a = .01 \leq x < b = 1000$ to 2% relative accuracy. Design the LPA system so that a and b are in the range of normalized numbers. Use an IEEE like system, with fraction bits, exponent bits etc., but as few as possible.

   (a) How many fraction bits are needed?

   (b) How many exponent bits are needed?

   (c) What is the exponent offset?

   (d) What is ϵ_{mach} in this LPA system? This is the same as asking what is distance between 1 and the smallest floating point number larger than 1, though the answers may differ by a factor of 2.

   (e) What is the smallest non-zero normalized number?

   (f) What is the smallest non-zero denormalized number?

2. Consider the problem of finding x so that $\sin(x) - x = a$ when a is close to zero but not equal to zero. How does the condition number of this problem behave as a goes to zero? Warning: the notation is backwards from the notation is the notes and lecture. Here, a is the data and x is the "answer" to be calculated.

3. We want to evaluate $f(x) = e^x - 1$ to high relative accuracy when x is close to zero. The goal is that if $|x| < 1$ and x is within the range of normalized double precision floating point, then the relative accuracy satisfies

$$\frac{\left|\widehat{f} - f\right|}{|f|} \leq \text{tol} = 10^{-6} \ .$$

   (a) Show that the condition number of the problem allows this accuracy in double precision arithmetic.

   (b) Show that direct evaluation using

```
f = np.exp(x)-1.
```

will not achieve this accuracy for normalized x values close to zero.

(c) Design a hybrid algorithm that uses direct evaluation if $|x|$ is larger than some threshold. For smaller x, it should use the Taylor series

$$e^x - 1 \approx x + \frac{1}{2}x^2 + \cdots + \frac{1}{n!}x^n \ .$$

Use the error approximation that replaces the tail of the Taylor series with the first neglected term:

$$\left| e^x - 1 - \left(x + \cdots + \frac{1}{n!}x^n \right) \right| \approx \frac{1}{(n+1)!} \, |x|^{n+1} \ .$$

Determine the smallest number of terms needed. Find a threshold to minimize the number of terms needed.

4. **Coding and analysis.** Please read the **coding standards** material below and make sure the code you upload meets the standards before you upload. Please upload one Python module that does parts (a) to (e). You should not need to upload plots because running the code will produce them.

The Fibonacci recurrence relation is

$$f_{n+1} = f_n + f_{n-1}$$

Write a Python module with the following components:

- A function `FibForward(f0,f1,M)` that takes arguments f_0, f_1, and M and returns the tuple $[f_M, f_{M+1}]$.
- A function `FibBackward(fM,FMp1,M)` that takes f_M and f_{M+1} and returns the tuple $[f_0, f_1]$.
- A function `UpDown(f0,f1,M)` that uses `FibForward` to compute (approximately) f_M and f_{M+1}, then uses `FibBackward` to re-compute f_0 and f_1, and returns these as a tuple.
- A main program that does the tasks described below.

(a) Use the routines `FibForward` and `FibBackward` for some sensible f inputs and moderate M, print the results from each to check that they agree with the analytic (mathematical) values. You should always do checks like this, but this is the last time you need to hand it in.

(b) Compute `[f0h,f1h] = UpDown( 1, 1, M)` and print the errors $\widehat{f_0} = 1$ for a reasonable (not too small and not too large) collection of M values. Note the type of the result and that results for moderate and large M are different. Explain this behavior using the properties of integer arithmetic. Warning: `f0=1` makes `f0` an integer, while `f0=1.` makes `f0` the double precision floating point 1. This part explores the consequence of using integer rather than floating point arithmetic. The variable name `f0h` is for "f zero hat", which is $\widehat{f_0}$.

(c) Repeat part (b), but using 64 bit floating point starting values `[f0h,f1h] = UpDown( 1., 1., M)`. Explain the different range of M values for which $\widehat{f}_0$ is close to 1. Print the results in "exponential" format such as `11.3e` rather than "floating point" format, such as `11.3f`. You may miss the contrast between the result here and in part (d) if you use `f` format.

(d) Repeat part (c) but with `[f0h,f1h] = UpDown( 1., 1.003, M)`. Changing $f_1 = 1$ to $f_1 = 1.003$ is a small mathematical change, but 1.003 is not represented exactly in floating point. Describe the loss of accuracy as M increases and contrast this to the result of part (c). Why is the loss sudden in one case and gradual in another.

(e) Plot (and store plot(s) in `pdf` format) the error $\widehat{f}_0 - f_0$ as a function of M for a range of M using the f_0 and f_1 values from part (c) and part (d). For this, you may use all M values in your chosen range rather than selecting a small sample as in parts (c) and (d). Put the two plots in the same graph for easy comparison. Comment on the differences and similarities of the two curves. Choose the plotting style (linear scale or log scale on the x and y axes) to illustrate the (loss of accuracy) clearly.

(f) Some analysis: suppose the all the arithmetic in the forward and backward calculations is done is done exactly except that f_M is replaced by $f_M(1 + \epsilon_{\rm mach})$ (double precision floating point). Find (approximately) what M gives 100% relative error in f_0, which means

$$\left| \frac{\widehat{f}_0 - f_0}{f_0} \right| \approx 1 \; .$$

You need to play with recurrence relations to do that and to think about the mode decomposition of f_M and f_{M+1}, and the growth/decay rates corresponding to the two modes. Does this estimate agree approximately with the M you get from part (d) and part (c)?

coding standards The standards for this exercise are illustrated in the file *BinomialCoefficients.py* posted with the assignment. This program calculates binomial coefficients $\binom{n}{k}$ for $k = 0, 1, \cdots, n$ using $\binom{n}{0} = 1$ and the recurrence

$$\binom{n}{k} = \frac{k}{n-k+1} \binom{n}{k-1} \; .$$

The coefficients grow from 1 to

$$\binom{n}{\frac{n}{2}} \approx \frac{2^n}{\sqrt{2\pi n}} \; .$$

For $n = 1000$ this number is barely in the range of double precision floating point. For $n = 2000$, the number is too large. After the maximum, the binomial

coefficients get smaller again, until the last one, like the first, is

$$\binom{n}{n} = 1 \ .$$

The surprising thing is that all these binomial coefficients are computed to high relative accuracy, even when the numbers themselves are extremely large, as long as they are in the range of floating point arithmetic. The multiplications and divisions do not induce cancellations. It is cancellation, not big numbers, that leads to floating point calculations being inaccurate.

Please download that file and open it in a code editor that shows line numbers. Here are standards your code should follow:

- (lines 1-9) Comments at the top saying what the code is for, the name of the file, who wrote it, with contact information, and when.
- (lines 9-11) Use white space (blanks and blank lines) to make code easier to read. Make things line up vertically.
- (line 13) Separate major code segments with comment lines and explanations of what the sections do.
- (lines 16 - 20) Make a docstring for every function defined. The docstring should say what the function does, what the arguments are (with type) and what the output is.
- (line s 22-26) Comments explaining variable names (if necessary). Use white space in comments to make them easier to read.
- (line 22) Use a `numpy` array rather than a list for mathematical arrays. The `numpy` method `zeros` gives you an array of a given shape.
- (line 38) You can use a Python list (not a numpy array) for things that are not used much, especially if you want to change the items and the number of items, easily while "playing with" the code.
- (lines 46-48) Printed numbers should be formatted so that similar numbers in different lines of output line up so the overall output is easy to read. In Python language, `format` is an *attribute* of the character string class. Choose the output format for numbers from the choices `d`, or `e`, or `f`. Look at the formatted output documentation for details. Output formatting in Python is clumsy, but with copy/paste and patience you can do it.
- (lines 46-50) Python style guides all say that lines of Python code should not be long. Some say no more than 72 or maybe 80 characters per line. These four lines (and a blank line) could have been made into one line, but that line would have been too long.

- (line 54 and 63) Saving plots in files makes it easy to look at many plots related to a single computation. Make sure the plotfile names are helpful and the plots in the plotfiles have enough information for you to tell what the plot is about. It is important to put formatted numbers into plot titles and legends if they are necessary, for example, to tell you what the run parameters are.

- (lines 66-70) every plot should have a title, labelled axes, a legend (if more than one curve), and grid lines.

- (lines 73-90) Choose axes to make it clear what the function being plotted is. Here, you learn a lot from the semi-log plot that you don't see in the linear scale plot. Compare the two output plots to see this point.

- (output) Here is what I see when I run my Python file at the command line

```
>>> python3 BinomialCoefficients.py
    Experiment computing binomial coefficients in floating point
    n is    5, max is  1.000e+01, error is    0.0000e+00
    n is   40, max is  1.378e+11, error is    0.0000e+00
    n is   80, max is  1.075e+23, error is    4.4409e-16
    n is  300, max is  9.376e+88, error is    6.6613e-16
    n is 1000, max is 2.703e+299, error is    1.1102e-15
    n is 2000, max is        inf, error is           inf
```

  Notice that the numbers line up so you can, for example, easily see the n values and the corresponding maximum and error values. The `inf` in the output means that the floating point number being printed has the value `inf` (larger than any number than can be represented in floating point).

- (plots) The plots show behavior of the binomial coefficients in different "regimes" (ranges of k). The linear scale plot shows the shape of the binomial "curve" (thinking of k as a continuous variable), but doesn't say what happens when k is far from $\frac{n}{2}$. The semi-log plot (log on the y axis, linear on the x axis) shows that $\binom{n}{k}$ continues to decrease "exponentially" as k moves away from $\frac{1}{2}$ in either direction. The scale on the y axis shows how small the binomial coefficients become, relative to their maximum value. The semi-log plots suggest that $\log(\binom{n}{k})$ is a relatively simple function of $x = \frac{k}{n}$, maybe just a quadratic? An analysis using Stirling's formula shows this function is not quadratic, but has the form $\phi(x) = a + b[x\log(x) + (1-x)\log(1-x)]$.