Assignment 1

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. This exercise asks you to go through some aspects of linear recurrence relations that was covered in class.

   (a) Consider the four term recurrence relation

   $$x_{n+1} = -2x_n + x_{n-1} + x_{n-2} \; . \tag{1}$$

   Let $\mathcal{S}$ be the set of all sequences $x_0, x_1, x_2, \cdots$ (defined for all $n \geq 0$) that satisfy the recurrence relation (1) for all $n \geq 2$. Show that $\mathcal{S}$ is a vector space of dimension 3. *Remarks.* There are two parts to this. First show $\mathcal{S}$ is a vector space by showing that if $x \in \mathcal{S}$ and $y \in \mathcal{S}$ are two such semi-infinite (i.e., defined for all $n \geq 0$ sequences in $\mathcal{S}$ and if a is a number, then the sequence $ax = ax_0, ax_1, \cdots$ has $ax \in \mathcal{S}$, and $x + y \in \mathcal{S}$. Then show $\mathcal{S}$ is three dimensional by showing that there are three sequences $u \in \mathcal{S}$, $v \in \mathcal{S}$, and $w \in \mathcal{S}$ that are linearly independent and that any $x \in \mathcal{S}$ may be represented as $x = au + bv + cw$ for some numbers a, b, c. The purpose of this exercise it to have you go through the definition of vector space and basis in an abstract way, which will be useful in the rest of the course. For a review of linear algebra, consider the book by Peter Lax.

   (b) Consider the recurrence relation

   $$x_{n+1} = x_n - x_{n-1} \; . \tag{2}$$

Let $\mathcal{S}$ be the set of all doubly infinite sequences (i.e., x_n defined for all integers n, both positive and negative) that satisfy (2). Show that every $x \in \mathcal{S}$ is bounded. *Remarks.* A sequence is *bounded* if there is a C (an *upper bound*) so that $|x_n| \le C$ for all n. For example, the sequence $x_n = \sin(n) - 3\cos(n^2)$ (warning, this sequence is not in $\mathcal{S}$) is bounded because the sine or cosine of anything is not more than 1, therefore $C = 4$ works. Some tricks for boundedness are

- If x and y are bounded, then $x + y$ is bounded. To see this, suppose C_x is an upper bound for x and C_y is an upper bound for y, then $C = C_x + C_y$ is an upper bound for $x + y$.
- If x is bounded and a is a number, then ax is bounded. If C_x is an upper bound for x, then $|a|\, C_x$ is an upper bound for ax (the absolute value is because a might be negative).
- The linear algebra of solutions of recurrence relations and the properties of bounded sequences also apply if the numbers x_n are allowed to be complex numbers that are not real. For example, if $z^2 - z + 1 = 0$ then z is not real but $x_n = z^n$ is in $\mathcal{S}$. The *multiplier*, a, above does not have to be real.
- If $z = \xi + i\eta$ is a complex number (ξ and η being real), then $|z|^2 = \xi^2 + \eta^2$ defines the *norm* $|z|$. This satisfies $|zw| = |z| \cdot |w|$ (w being any other complex number). [If you haven't seen this before, do the algebra to check it for yourself. It's easy and very important.] In particular, $|z^n| = |z|^n$. [Why is it true both for $n > 0$ and $n < 0$ if $z \neq 0$?] Therefore, if $|z| = 1$, then the sequence z^n is bounded.

(c) This asks you to find a formula for the solution of an *inhomogeneous* Fibonacci sequence

$$y_{n+1} = y_n + y_{n-1} + r_n \ . \tag{3}$$

The numbers r_n are called *forcing* or *inhomogeneous terms*. First find a formula for the sequence $T_{k,n}$ that has forcing only when $n = k$. Specifically,

$$\begin{aligned} T_{k,n} &= 0 \ \text{ for } n < k \ , \\ r_n &= 0 \ \text{ for } n \neq k \ , \\ r_k &= 1 \ . \end{aligned}$$

For each $k \ge 2$ there is a sequence $T_{k,n}$ defined for $n \ge 0$. This collection of sequences is sometimes called the *fundamental solution* of the recurrence. Second show that if $T_{k,n}$ is a fundamental solution, then the solution of (3) is

$$y_n = x_n + \sum_{k=2}^{\infty} r_k T_{k,n} \ . \tag{4}$$

Here, x is a solution of the homogeneous ($r_n = 0$ for all n) Fibonacci relation. You might worry that the infinite sum does not converge, but you don't have to worry because, for each n there are only finitely many k with $T_{k,n} \neq 0$. The sums in (4) are finite sums.

(d) Show that if $y_0 = y_1 = 1$ and $|r_n| \leq r$ for all n, then

$$|y_n| \leq \gamma^n \frac{r}{\gamma - 1} \ . \tag{5}$$

Here $\gamma = \frac{1+\sqrt{5}}{2}$ is the *golden mean*. *Hint.* The geometric sum formula is

$$\sum_{k=0}^{n-1} \gamma^k = \frac{\gamma^n - 1}{\gamma - 1} \leq \gamma^n \frac{1}{\gamma - 1} \ .$$

Pay attention to the mathematicians trick in the last inequality. Dropping the -1 term in the numerator gives a valid inequality and what you get is simpler. You lose almost nothing because for large n, $\gamma^n - 1$ and γ^n are almost the same. *Remark.* If you think of r_n as coming from rounding errors, then

3. **Coding and analysis.**

This exercise explores the effect of roundoff in computing with three term recurrence relations of the form

$$x_{n+1} = ax_n + bx_{n-1} \ . \tag{6}$$

We will go "up and back down" using (6) in the forward direction and then in the reverse direction, and then see how close the result is to the starting x_0. Specifically, we will program

$$\begin{aligned}
&\text{going up} \\
x_2 &= ax_1 + bx_0 \\
x_3 &= ax_2 + bx_1 \\
&\vdots \\
x_{M+1} &= ax_M + bx_{M-1} \\
&\text{going back down} \\
\widehat{x}_{M-1} &= \frac{1}{b}\left(x_{M+1} - ax_M \right) \\
\widehat{x}_{M-2} &= \frac{1}{b}\left(x_M - a\widehat{x}_{M-1} \right) \\
\widehat{x}_{M-3} &= \frac{1}{b}\left(\widehat{x}_{M-1} - a\widehat{x}_{M-2} \right) \\
&\vdots \\
\widehat{x}_0 &= \frac{1}{b}\left(\widehat{x}_2 - a\widehat{x}_1 \right)
\end{aligned}$$

Suppose all the operations here are done in double precision floating point in the order given. Then $\widehat{x}_0 \neq x_0$. The difference $\widehat{x}_0 = x_0$ depends on M, which is how far up you go before coming back down.

Code this experiment using four methods (at least) in one Python module:

Write a Python module with the following components:

- A function `Forward(x0,x1,a,b,M)` that takes arguments x_0, x_1, and M and returns the tuple (x_M, x_{M+1}).
- A function `Backward(xM,xMp1,a,b,M)` that takes x_M and x_{M+1} and returns the tuple (x_0, x_1).
- A function `UpDown(x0,x1,a,b,M)` that uses `Forward` to compute (approximately) x_M and x_{M+1}, then uses `Backward` to re-compute x_0 and x_1. It should also return $x_{\max} = \max(|x_M|, |x_{M+1}|)$. It should return the tuple x_0, x_1 and $x_{\max}$.
- A main program that does the tasks described below.

(a) Verify the correctness of `Forward`, `Backward` and `UpDown`for some simple input arguments where you can hand-compute the results. Print the output and correct results in a way that makes it easy to compare. Make sure your test problem would uncover errors such as mixing up x_0 with x_1 or a with b. For example, do not use $a = b = 1$ and $x_0 = x_1 = 1$, which are the Fibonacci values. You should always do checks like this, but this is the last time you need to hand it in.

(b) Compute `(x0h,x1h) = UpDown( 1., 1., 1, 1, M)` and print the errors `x0h-1.` and the max value for a thoughtful (not too small and not too large) collection of M values. These should include M for which there is no roundoff error and M where the recomputed $\widehat{x}_0$ is completely wrong (relative error on the order of 100% or `Inf` or `NaN`).

(c) Repeat part (3b), using non-integer x_0, x_1, a, and b, but with values within, say, 1% of the integer Fibonacci values. Explain the two observations:

- Now there is roundoff in (almost) every computed value.
- The breakdown (computed $\widehat{x}_0$ completely wrong) happens for an M not so different from before.

(d) Repeat part (3b) but with all integer values using the `int` datatype in Python. You achieve that using `1` instead of `1.` (no decimal point). Explain that the behavior indicates that arithmetic with the `int` datatype is not 32 or 64 bit integer arithmetic.

(e) Plot (on one plot) $|\widehat{x_0} - x_0|$ and $x_{\max}$ as a function of M. Do this using the true Fibonacci values of part (3b) and the slightly different values of part (3c). Make plots where the y axis is a linear scale and a log scale and note which is easier to interpret. Explain why the log scale results are mostly straight lines. Be careful with coding when

you take the log of zero. Save the plots to files and hand them in separately.

(f) Repeat part (3c) in single precision if you can (it's hard to force Python to use single precision) and note the differences in the results.

(g) (*Coding hint*). You should have at most one program for each part and maybe one code that does several parts. It should be very easy to redo experiments with small changes if everything is automated, including creating and labelling plots.