Assignment 1

1. **A condition number example.** Consider the function

$$f(x) = e^x - 1 \, .$$

   (a) Show that the problem of computing $f(x)$ is well conditioned uniformly for x near zero. This means, for example, that $\kappa(x) \leq C$ whenever $|x| \leq 1$, and that C is a reasonable number (maybe 2 or 10, but not 10^6). It also means that $\kappa(x)$ is bounded as $x \to 0$. If you're not a pure math nerd, it just means that it should be possible to compute $f(x)$ without losing more than one digit of accuracy when x is close to zero.

   (b) Consider the algorithm

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

   Show that this algorithm is unstable for small x in the sense that the relative error in computing $f(x)$ "blows up" (the answer is 100% wrong) if x is close enough to zero.

2. **A quirk of floating point.** One member of the *logistic map* family is

$$u(x) = 4(x - x^2)$$

The logistic map recurrence is the iteration $x_{n+1} = u(x_n)$ It maps the unit interval to itself, which means that if $0 \leq x_n \leq 1$ then $0 \leq x_{n+1} \leq 1$. If is a model of "chaos" in that if x_0 and y_0 are very close, then x_n quickly separates from y_n as n increases. It is a 2 to 1 map in that the interval $[0, \frac{1}{2}]$ is mapped to $[0, 1]$ and $[\frac{1}{2}, 1]$ is mapped to the same interval, but backwards. Another map with these properties is the *baker's map*, which is

$$v(x) = \begin{cases} 2x & \text{if } 0 \leq x \leq \frac{1}{2} \\ 2 - 2x & \text{if } \frac{1}{2} \leq x \leq 1 \end{cases}$$

The name comes from kneading bread, in which it is first stretched and then folded over. The two maps have similar graphs and properties, except that the graph of u is smooth at $x = \frac{1}{2}$ while $v(x)$ is pointed there. Explain the following behavior: If you code the baker's map in simple Python floating point and iterate ($x_{n+1} = v(x_n)$) then no matter what x_0 you take, eventually (after less than 100 steps), you get $x_n = 0$ for all subsequent n. The mathematical iteration does not do this, nor does the logistic map.

3. **Coding and analysis.**

This exercise explores a simple but unstable algorithm for solving a *two point boundary value problem*. The problem is to find numbers $x_1, \cdots, x_{n-1}$, assuming *boundary conditions* $x_0 = 0$ and $x_n = 0$. The indices $k = 0$ and $k = n$ are the two *boundary points*, and the corresponding conditions $x_0 = 0$ and $x_n = 0$ are the boundary values or *boundary conditions*. The recurrence relation is

$$x_{k+1} - 3x_k + x_{k-1} = f_k \ . \tag{1}$$

The numbers f_k are the *right hand side* or *inhomogeneous term* in the recurrence. The corresponding *homogeneous* recurrence relation is (1) with $f_k = 0$ for all k. The relation (1) is supposed to apply for $k = 1, 2, \cdots, n-1$. The boundary values are implicitly used in the $k = 1$ and $k = n-1$ equations. For example, since $x_0 = 0$, the $k = 1$ equation is

$$x_2 - 3x_1 = f_1 \ .$$

There are fast and accurate methods for solving this two point boundary value problem, which we will see in a few weeks. The purpose of this exercise is to illustrate a algorithm that is equally fast, but unstable and inaccurate except for very small n.

This bad algorithms is called *shooting*. The name is based on the image that you start with the left boundary value $x_0 = 0$ and "shoot" to "hit" the other boundary condition $x_n = 0$. The method uses two sequences u_k and v_k, constructed as follows. The u_k sequence takes $u_0 = 0$ and $u_1 = 0$, and then uses the recurrence

$$u_{k+1} - 3u_k + u_{k-1} = f_k$$

to compute u_2, u_3, etc. This sequence is $u_2 = f_1$, then $u_3 = 3u_2 + f_2$, and so on. It is unlikely that $u_n = 0$, so this sequence does not satisfy the right ($k = n$) boundary condition. The v_k sequence starts with $v_0 = 0$, $v_1 = 1$, then (using the same recurrence relation), $v_2 = f_1 + 3$, $v_3 = f_2 + 3v_2 - v_1$, etc. The "shooting" trick is to look for our solution sequence as a linear combination of these two computed ones:

$$x_k = u_k + av_k \ . \tag{2}$$

We then see the correct value of the parameter a.

(a) Show that the x sequence defined by (2) satisfies the recurrence relation formulas (1) for all $k > 0$.

(b) Find a formula for the parameter a in terms of u_n and v_n.

(c) Show that the homogeneous recurrence relation (the one with $f_k = 0$ for all k) has only one solution, which is $x_k = 0$ for all k. *Warning*: It is clear that $x_k = 0$ for all k is *a* solution. The point is to show

it's the *only* solution (which mathematicians call *uniqueness*). *Hint*: You only need to show that $x_1 = 0$ (why), which can be done by contradiction. If $x_1 \neq 0$, then $x_2 \neq 0$ too (why?), and so on. *Warning*: showing x_2 is probably not zero is not the same thing as showing it's definitely not zero.

Write a Python function in a module to implement this shooting algorithm. The function should take as inputs the number n and a one index `numpy` array of (double precision) floats $f = [f_1, \cdots, f_{n-1}]$. Be careful with the coding because Python arrays usually start with $k = 0$ rather than $k = 1$. A simple way around this is to declare `f` to have length n but never assess f_0. The function should return the solution as a `numpy` array $x = [x_1, \cdots, x_{n-1}]$.

Test that your routine works by giving it a problem you know the answer to. For that, take

$$x_k = \sin\left(\frac{\pi k}{n}\right) .$$

You can use the recurrence (1) to find the corresponding f_k (hint: do this numerically using the simple formula (1) rather than some trigonometric identies). Your function will take this f and return a computed array $\widehat{x}$. If this were done in *exact arithmetic* (i.e., mathematically), you would get $\widehat{x} = x$. The result in floating point will have a mean error (average error per value)

$$E_n = \frac{1}{n} \left\| \widehat{x} - x \right\|_1 = \frac{1}{n} \sum_{k=1}^{n-1} |\widehat{x}_k - x_k| .$$

Note that the important distinction between relative and absolute error is not so important here because the exact answer is on the order of one. This error should be on the order of (double precision) roundoff for small n and grow as n increases. Print E_n for a few chosen n values (some small, some medium, etc.) to check this.

Finally, the "picture worth a thousand words" is a semi-log plot of E_n as a function of n for n in a range ranging from small (where the error is order roundoff) to larger (but not much larger) so that the computed $\widehat{x}$ is completely wrong. That is, create a loop that finds the numbers E_n for $n = N_{\min}$ to $n = N_{\max}$ and makes the semi-log plot. You should experiment with the code to get a good $n = N_{\max}$. Hand in your one best picture.

(d) What is the shape of this error curve in a semi-log plot? If the curve were exactly that shape, what would it say about E_n as a function of n?

4. **Recurrence relations, not to hand in** Write a Python function that takes as inputs x_0, x_1, a, and b, and n and returns x_n. It should define

x_n using the recurrence relation

$$x_{k+1} = ax_k + bx_{k-1} \ .$$

The *characteristic polynomial* of this is

$$p(z) = z^2 - az - b \ .$$

Choose values of a and b so that p has two real roots $z_+ > z_1$. Write another function that takes as inputs x_0, x_1, z_+ and z_- and returns c_+ and c_- so that

$$\begin{aligned} x_0 &= c_+ + c_- \\ x_1 &= c_+ z_+ + c_- z_- \end{aligned}$$

Check that this software products numbers that satisfy

$$x_n = c_+ z_+^n + c_- z_-^n \ .$$

Of course, this relation will be only approximately true for the computed numbers because of roundoff. Print the error $e = x_n - \left(c_+ z_+^n + c_- z_-^n \right)$ to see that it's on the order of roundoff. Try a few coefficient sets, possibly including Fibonacci ($a = b = 1$) and $a = 3$, $b = -1$ from above. Don't take n too big.

Choose parameters with $|z_+| > |z_-|$. Consider the approximation that includes only the faster growing mode:

$$x_n \approx c_+ z_+^n \ .$$

See in computed results that the relative error of this approximation goes to zero as $n \to \infty$. For Fibonacci, this happens pretty fast. Use this to explain what was said in class, that

$$\frac{x_{n+1}}{x_n} \to z_+ \quad \text{as} \quad n \to \infty \ .$$

Again, you don't have to take n very large for this behavior to be clear. It should be true for other a and b too.