Math 128A, Summer 2019

PSET #1 (due Wednesday 7/3/2019)

Problem 1. Fix integer $n \geq 1$, n points x_i with $|x_i| \leq 1$, n points y_j with $|y_j| \leq 1$, n coefficients f_j, and n coefficients g_j.

(a) Fix integer $k \geq 0$. Design an algorithm for evaluating

$$f(x) = \sum_{j=1}^{n} f_j (x y_j)^k$$

at n points x_i, with $O(n)$ operations.

Solution. Notice that we can bring the x factor out of the summation, as it is independent from the index of summation and independent from the other factors. That is, $f(x) = x^k \cdot \sum_{j=1}^{n} f_j (x y_j)^k$. Our algorithm to evaluate $f(x)$ will first compute the value of the summation, then compute the total product for each x_i.

1. Initialize `xpowerk, ypowerk, f, ftimesy, fFinal` each as $1 \times n$ matrix of zeroes. This initialization has constant cost $O(1)$.
2. Compute x_i^k, y_j^k. For `i,j = 1:n`, store `xpowerk[1,j] <- x[1,i]**k` and `ypowerk[1,j] <- y[1,j]**k`. Each consists of n computations, so this step costs $2n$ at $O(n)$.
3. Compute $f_j y_j^k$. For each $j = 1 : n$, let `ftimesy[1,j] <- f[1,j] * ypowerk[1,j]`. This costs n multiplications for a runtime $O(n)$.
4. Compute $\sum_{j=1}^{n} f_j y_j^k$ by adding all elements of `ftimesy` and storing it as the number `sumfy`. This takes $(n-1)$ or n additions, depending how it is defined, with $O(n)$ cost.
5. Now compute $f(x) = [x_i^k] \left[\sum_{j=1}^{n} f_j y_j^k \right]$.
   For each `i = 1:n`, store `fFinal <- xpowerk[1,i] * sumfy`. This costs n multiplications with $O(n)$ cost.

□

(b) Find a polynomial $P(x)$ with complex coefficients such that

$$|P(x) - e^{ix}| \leq \varepsilon = 2^{-52}$$

on the interval $|x| \leq 1$.

Solution. We want to find some $P(x)$ where our absolute error is bounded by some given $\epsilon > 0$ (take this to be ε, the machine epsilon). We know the Taylor series (or sequence) converges, hence for some large enough order or number of terms, our Taylor expansion should satisfy this.

Consider the Taylor expansion of $e^{ix} = \sum_{k=0}^{\infty} \frac{(ix)^k}{k!}$ with $x \in \mathbb{R}$ and $|x| \leq 1$. Listing out these terms, we notice a pattern (which reminds us of a trigonometric separation of e^{ix}).

$$\begin{aligned} e^{ix} &= 1 + ix - \frac{x^2}{2} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \frac{x^6}{6!} - \frac{ix^7}{7!} + \frac{x^8}{8!} + \cdots \\ &= \left(1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots\right) + i\left(x - \frac{ix^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\right) \\ &= \cos x + i \sin x \end{aligned}$$

Hence let $P(x) := C_n(x) + iS_{n+1}(x)$ where $C_n(x), S_{n+1}(x)$ are $n, (n+1)$-degree Taylor polynomials of $\cos x, \sin x$, respectively. By definition of absolute value (pythagorean theorem for real and imaginary parts),

we have:

$$\begin{aligned}\left|P(x)-e^{ix}\right|^2 &= |C(x)-\cos x|^2 + |S(x)-\sin x|^2 \\ &\approx \left(\frac{x^n\cos\xi_1}{n!}\right)^2 + \left(\frac{x^{n+1}\sin\xi_2}{(n+1)!}\right) \quad \text{(cos even, sin odd)} \\ &\leq \left[\frac{1}{n!}\right]^2 + \left[\frac{1}{(n+1)!}\right]^2 \quad (|x|, |\cos x|, |\sin x| \leq 1)\end{aligned}$$

Hence we can choose n such that $C_n(x), S_{n+1}(x)$ satisfy the following, for the given ϵ:

$$\left|P(x)-e^{ix}\right|^2 \leq \left[\frac{1}{n!}\right]^2 + \left[\frac{1}{(n+1)!}\right]^2 < \epsilon$$

Now letting $\epsilon := \varepsilon$, we have :

$$\left[\frac{1}{n!}\right]^2 + \left[\frac{1}{(n+1)!}\right]^2 < \varepsilon$$

which by testing n, we have

```
( factorial(11)^ (-2) + factorial(12)^ (-2) ) = 6.3197e-16
( factorial(12)^ (-2) + factorial(13)^ (-2) ) = 4.3842e-18
eps = 2.2204e-16
```

So we reason we need a minimum degree $n = 12$ to cause underflow and get the maximum IEEE standard machine precision (ε).

□

(c) Design an algorithm for approximating

$$g(x) := \sum_{j=1}^{n} g_j e^{ixy_j}$$

at n points x_k in $O(n)$ operations, with absolute error bounded by $\epsilon \sum_{j=1}^{n} |g_j|$.

Solution. In part (b) above, we constructed $P_n(x) = C_n(x) + iS_n(x)$. By definition, our polynomial $P_n(x)$ can be expressed as a sum (linear combination) of $(n+1)$ monomials. That is, we let $P_m(x) = z_m x^n + z_{m-1}x^{m-1} + \cdots + z_1 + z_0$, where $z_i \in \mathbb{C}$. Consider:

$$\begin{aligned}g(x) := \sum_{j=1}^{n} g_j e^{ix_k y_j} &\approx \sum_{j=1}^{n} (g_j) P_m \\ &= \sum_{j=1}^{n} \left[(g_j) \left(\sum_{k=0}^{m} z_k (ix)^k \right) \right] \\ &= \sum_{k=0}^{m} \left[\sum_{j=1}^{n} (g_j z_k)(x_i y_j)^k \right] \\ &= \sum_{k=0}^{m} \left[z_k x_i \left(\sum_{j=1}^{n} (g_j)(y_j)^k \right) \right]\end{aligned}$$

This last expression is precisely that which we addressed in part (a), with $f_j := g_j$. Our algorithm proceeds as in (a), computing and storing the expression in the parenthesis first for `j = 1:n` in $O(n)$, multiplying and then adding terms `k = 0:m` in $O(n)$. □

(d) Define the $n \times n$ matrix F by

$$F_{j,k} := e^{ix_j y_k}.$$

Find a rank r independent of n and an $n \times n$ matrix B with elements

$$B_{j,k} = \sum_{i=1}^{r} c_{j,i} d_{i,k}$$

such that B has rank at most r and absolute error

$$|F_{j,k} - B_{j,k}| \leq \epsilon \quad \forall n.$$

Solution. Let r be the number of terms required as given by (b) above. As mentioned in class, consider $(Fg)_j = \sum_{k=1}^{n} \underbrace{e^{ix_j y_k}}_{F_{ij}} g_k$.

We need $O(n^2)$ cost to get the matrix elements of F_{jk} because each x_j and y_k has to talk to each other. To see this, recall that matrix multiplication is like dotting two vectors; n multiplications and n additions, per row. So in total for the matrix, we need $O(n^2)$.

The essence of the problem is that we can do this faster if we exploit the fact that we can bring out a term. Like in (a) where we let

$$\sum_{j=1}^{n} (x_i y_j)^k f_j = x_i^k.$$

So we have:

$$\begin{bmatrix} e^{ix_1 y_i} & \dots & e^{ix_1 y_n} \\ \vdots & \ddots & \vdots \\ e^{ix_n y_1} & \dots & e^{ix_n y_n} \end{bmatrix}$$

To quote Strain, this problem is about a "miracle" that rank $B \leq r$. Suppose our matrix C is $r \times n$, with $r = 15 << n$; thus $O(15n) = O(rn) = O(n) << O(n^2)$. 15 (or the actual number) is a lot, but it is surely less than n^2.

Hence r depends on our accuracy ε and not on n. So for the matrices $F, B := CD, E =$ error, we have:

$$\mathcal{M}(F) = \underbrace{\mathcal{M}(CD)}_{\mathcal{M}(B)} + \mathcal{M}(E),$$

where we have

$$\text{rank } [(n \times r)(r \times n)] = \min\{(n \times r), (r \times n)\}$$

So to ensure the error bound of each element, in this problem we use:

$$\begin{aligned} e^{ix_j y_k} &= \sum_{p=0}^{r-1} \frac{(ix_j y_k)^p}{p!} + \overbrace{O\left(\frac{1}{r!}\right)}^{E} \\ &= \sum_{p=0}^{r-1} \left(\frac{ix_j^p}{\sqrt{p!}}\right) \left[\frac{y_k^p}{\sqrt{p!}}\right] \quad \text{(or equivalently can split by } 1 \cdot p!) \\ &= \sum_{p=0}^{r-1} (C_{jp}) \cdot [D_{pk}] \end{aligned}$$

□

Problem 2. Show that the floating point arithmetic sums

$$S_n := \sum_{k=1}^{n} \frac{1}{k^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2}$$

with absolute error $E_n \leq (2n+1)\epsilon$ from left to right, while summing from right to left gives absolute error $\leq (3 + \ln n)\epsilon$. Estimate the maximum accuracy achievable and the number of terms required in each case.

Solution. Let $s_k := \frac{1}{k^2}$ and S_n be defined as above. We must account for precision and error due to floating point arithmetic, so let us define $\hat{S}_n$ as the floating point sum corresponding to the true S_n. Given by the problem, let us define the **absolute** error due to floating point arithmetic at the n-th term as $E_n := \hat{S}_n - S_n$ without absolute value.

(Summing $\rightarrow$)

When we sum left to right, we have the following algorithm:

$$\hat{S}_{n+1} = fl(\hat{S}_n + fl(s_{n+1}))$$

Then for $|\delta_j| \leq \frac{\varepsilon}{2}$ (whose indexes we drop), our summing algorithm gives us:

$$\begin{aligned}
\hat{S}_{n+1} &= \left(\hat{S}_n + s_{n+1}(1+\delta)\right)(1+\delta) \\
&= \hat{S}_n + s_{n+1} + s_{n+1}\cdot\delta + \hat{S}_n\delta + s_{n+1}\cdot\delta + (s_{n+1}\cdot\delta^2) \\
&= [\hat{S}_n] + s_{n+1} + s_{n+1}\cdot\delta + \hat{S}_n\delta + s_{n+1}\cdot\delta \quad \text{(throw away small terms)} \\
&= [E_n + S_n] + s_{n+1} + s_{n+1}\cdot\delta + \hat{S}_n\delta + s_{n+1}\cdot\delta \quad (\hat{S}_n - S_n = E_n \implies \hat{S}_n = E_n + S_n) \\
&= [S_n + s_{n+1}] + [E_n] + \delta(2s_{n+1} + [\hat{S}_n]) \quad \text{(grouping terms)} \\
&= S_n + s_{n+1} + E_n + \delta(s_{n+1} + s_{n+1} + [S_n + E_n]) \quad (\hat{S}_n = S_n + E_n) \\
&= S_n + s_{n+1} + E_n + \delta([s_{n+1} + S_{n+1} + E_n]) \quad (S_{n+1} = S_n + s_{n+1}) \\
&= S_{n+1} + E_n(1+\delta) + \delta(s_{n+1} + S_{n+1}) \\
&= S_{n+1} + E_n + \delta(s_{n+1} + S_{n+1})
\end{aligned}$$

Hence $|\hat{S}_{n+1} - S_{n+1}| =: |E_{n+1}| = |E_n| + \delta(s_{n+1} + S_{n+1}) \leq |E_n| + \varepsilon|s_{n+1} + S_{n+1}|$. Extending this logic gives an array of inequalities as such:

$$\begin{aligned}
|E_n| &\leq |E_{n-1}| + \varepsilon|s_n + S_n| \\
|E_{n-1}| &\leq |E_{n-2}| + \varepsilon|s_{n-1} + S_{n-1}| \\
|E_{n-2}| &\leq |E_{n-3}| + \varepsilon|s_{n-2} + S_{n-2}| \\
&\vdots \\
|E_1| &\leq |E_0| + \varepsilon|s_1 + S_1|
\end{aligned}$$

Adding all of these inequalities and subtracting the $|E_n| + |E_{n-1}| + \cdots + |E_1|$ across the inequality, we get:

$$|E_n| \leq \underbrace{|E_1|}_{0} + \varepsilon \underbrace{\sum_{k=2}^{n} s_k}_{<2} + \varepsilon \underbrace{\sum_{k=2}^{n} S_k}_{<2(n-2+1)} \leq 2n\varepsilon < (2n+1)\varepsilon, \quad \text{(geometric series)}$$

which is better than we were asked to show. (Alternatively, as suggested in lecture, we could have gotten a slightly even better bound by quoting the known result: $|fl(S_n) - S_n| \leq \frac{\varepsilon}{2} \cdot (|s_2| + |s_3| + \cdots + |s_n|) \leq (n-1)\varepsilon$.)
We use our upper bound of absolute error as an estimation for the maximum achievable accuracy.

As mentioned in class, the maximum accuracy achievable (in computing $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$):

$$fl\left(\sum_{k=1}^{n} \frac{1}{k^2}\right) = s_n\left(1 + 2n\delta\right), \qquad |\delta| \leq \varepsilon$$

There are two competing errors: truncation error on the left, and the floating point error in summing to infinity. Our question is to find the minimum error in between these two.
So the maximum accuracy at the bottom of this curve will be $O(n\varepsilon) + O\left(\frac{1}{n}\right)$, where $\sqrt{1/\varepsilon} = 2^{26}$. The best thing would be to take 2^{26}, and our value of accuracy will only be 2^{-26}.
(Summing $\leftarrow$)

Now we define a new algorithm (summing right to left) via inverted indexing $n \to 1$:

$$S_n := \sum_{k=n}^{1} s_k \quad , \quad s_k := \frac{1}{k^2} \quad , \quad E_n := \hat{S}_n - S_n$$

Then similarly as we have done in part (a), we have:

$$\begin{aligned}
\hat{S}_{k-1} &= fl[\hat{S}_k + fl(s_{k-1})] \quad \text{(by def of our summing algorithm)} \\
&= (\hat{S}_k + s_{k-1}(1+\delta))(1+\delta) \\
&= \left[\hat{S}_k + s_{k-1} + s_{k-1}\delta + \hat{S}_n\delta + s_{k-1}\delta\right] \quad \text{(throw away } \delta^2 \text{ term)} \\
&= (S_n + E_n) + s_{k-1} + \delta\left(2s_{k-1} + \hat{S}_k\right) \\
&= (S_n + s_{k-1}) + E_n + \delta\left(2s_{k-1} + \hat{S}_k\right) \\
&= S_{k-1} + E_n + \delta\left(2s_{k-1} + \hat{S}_n\right) \\
&= S_{k-1} + E_n + \delta\left(s_{k-1} + S_{k-1}\right) + \underbrace{\delta E_k}_{0}
\end{aligned}$$

Like in (a), this gives rise to another set of inequalities:

$$|E_{k-1}| \leq |E_k| + \delta s_{k-1} + \delta S_{k-1} \quad \text{(triangle ineq.)}$$

$$\text{Hence: } |E_k| \leq \underbrace{|E_n|}_{<2\delta} + \delta \underbrace{\sum_{k=1}^{n-2} s_k}_{<\int_1^k \frac{1}{k^2}\,dk} + \delta \underbrace{\sum_{1}^{n-2} S_k}_{<\int_1^n \frac{1}{k}\,dk}$$

$$< 2\delta + \delta \underbrace{\int_1^k \frac{1}{k^2}\,dk}_{<k\left(\frac{1}{k}\right)=1} + \delta \underbrace{\int_1^n \frac{1}{k}\,dk}_{=\ln n - \ln 1}$$

$$< 2 + 1 + \ln n = (3 + \ln n)\delta \leq (3 + \ln n)\varepsilon$$

□

Problem 3. Suppose a, b are floating point numbers with $0 < a < b < \infty$. Show that in IEEE standard floating point arithmetic, if no overflow occurs,

$$a \leq \text{fl}\left(\sqrt{ab}\right) \leq b.$$

Solution. As covered in class on Thursday, we know that because $0 < a < b < \infty$ where a, b are floating point numbers, we have:

$$\begin{aligned}
a < b \implies a^2 < ab \implies a &< \sqrt{ab} \\
&= \sqrt{fl(ab)} \\
&= \sqrt{ab(1+\delta)} \quad (|\delta| \leq \frac{\varepsilon}{2}) \\
&= \sqrt{ab} \cdot \sqrt{1+\delta} \\
&= \left[\sqrt{ab}\right] \cdot \left[1 + \frac{\delta}{2} - \frac{\delta^2}{4} + \cdots\right] \quad \text{(Taylor expansion)} \\
&< \left[\sqrt{ab}\right] \cdot \left[1 + \frac{\delta}{2}\right] \quad \text{(truncating alternating series)} \\
&\leq fl\left(\left[\sqrt{ab}\right] \cdot \left[1 + \frac{\varepsilon}{4}\right]\right) \\
&= fl(\sqrt{ab})
\end{aligned}$$

Analogously, we have: $a < b \implies ab < b^2 \implies \sqrt{ab} < b^2$. Thus:

$$\begin{aligned}
b^2 &> \sqrt{ab} \\
&= \sqrt{fl(ab)} \\
&= \sqrt{ab(1+\delta)} \\
&= \sqrt{ab}\left(1 + \frac{\delta}{2} + O(\varepsilon^2)\right) \quad \text{(as given in monday OH)} \\
&= \left[\sqrt{ab}\right] \cdot \left[1 + \frac{\delta}{2} - \frac{\delta^2}{4} + \cdots\right] \quad \text{(Taylor expansion)} \\
&> \left[\sqrt{ab}\right] \cdot \left[1 + \frac{\delta}{2} - \frac{\delta^2}{4}\right] \quad \text{(truncating alternating series)} \\
&\geq fl\left(\left[\sqrt{ab}\right] \cdot \left[1 + \frac{\varepsilon}{4} - \frac{\varepsilon^2}{16}\right]\right) \\
&= fl(\sqrt{ab}) + 0 + 0 \\
&= fl(\sqrt{ab})
\end{aligned}$$

We have shown $a < fl(\sqrt{ab}) < b$, as required. □

Problem 4. Design an algorithm to evaluate

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

in IEEE double precision arithmetic, to 12-digit accuracy for all machine numbers $|x| \leq 1$.

Solution. At a first glance, for $x \to 0$, the numerator $(e^x - 1 - x) \to 0$ and the denominator $(x^2) \to 0$, which is problematic. My first approach was to split up the terms to get $f(x) = e^x \left[\frac{1}{x^2}\right] - \frac{1}{x^2} - \frac{1}{x}$; however, the $-1 - x$ conveniently aligns with the first two terms in the expansion of e^x. We have:

$$\begin{aligned} f(x) := \frac{e^x - 1 - x}{x^2} &= \frac{\left[\sum_{k=0}^{\infty} \frac{x^k}{k!}\right] - 1 - x}{x^2} \\ &\approx \frac{\left[\sum_{k=0}^{n} \frac{x^k}{k!}\right] - 1 - x}{x^2} \quad \text{(for some large enough } n\text{)} \\ &= \frac{\left(1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots\right) - 1 - x}{x^2} = \sum_{k=2}^{n} \frac{x^k}{k! x^2} \end{aligned}$$

We want 12-digit accuracy, so our expression for relative error (where the denominator is strictly positive) is:

$$\begin{aligned} \left| \frac{e^x - 1 - x}{x^2} - \sum_{k=2}^{n} \frac{x^k}{k! x^2} \right| \Big/ \left(\frac{e^x - 1 - x}{x^2} \right) &= \left(\frac{1}{(e^x) - (1 + x)} \right) \cdot \left| \sum_{k=0}^{n} \frac{x^k}{k!} - e^x \right| \\ &= \frac{2}{x^2 e^{c_1}} \cdot \left| \frac{x^{n+1}}{(n+1)!} \right| e^{c_2} \quad \text{(error bounds by taylor remainders at } 2, (n+1)\text{)} \\ &= \frac{2 e^{c_2 - c_1} \cdot x^{n-1}}{(n_1)!} \end{aligned}$$

Because $x \leq 1 \implies |c_1|, |c_2| \leq 1$, in this last expression we can bound $e^{c_1} \leq 1$ and $e^{c_2} \leq 1$ to get 12-digit accuracy:

$$\text{relative error} \leq \frac{2e^2}{(n+1)!} \leq 10^{-12} \implies (n+1)! \geq (2 \cdot 10^{12}) e^2 \approx 1.4778 \times 10^{13}$$

Because this number is relatively small, we don't need Stirling's and just simply test to find the least n with $(n+1)!$ greater than this amount. `factorial(15)` in R or Matlab gives $1.3077 \times 10^{12} < 1.4778 \times 10^{13}$, but we see that `factorial(16)` gives $2.0923 \times 10^{13} > 1.4778 \times 10^{13}$.
Hence to guarantee 12-digit (base 10) accuracy, for our Taylor series expansion of e^x to the 16th power should suffice (we used inoptimal bounds so we overestimate the actual error). □

Problem 5. Figure out exactly what sequence of intervals is produced by bisection with the *arithmetic* mean for solving $x = 0$ with initial interval $[a_0, b_0] := [-1, 2]$. How many steps will it take to get maximum accuracy in IEEE standard floating point arithmetic?

Solution. A bisection algorithm solving $x = 0$ with initial bracket: $[-1, 2]$ will have steps producing the following intervals:

$$\text{step 0:} \quad [-1,2]\,; \qquad \text{step 1:} \quad \left[-1, \frac{1}{2}\right]; \qquad \text{step 2:} \quad \left[-\frac{1}{4}, \frac{1}{2}\right]; \qquad \text{step 3:} \quad \left[-\frac{1}{4}, \frac{1}{8}\right];$$

$$\text{step 4:} \quad \left[-\frac{1}{16}, \frac{1}{8}\right]; \qquad \text{step 5:} \quad \left[-\frac{1}{16}, \frac{1}{32}\right]; \qquad \text{step 6:} \quad \left[-\frac{1}{64}, \frac{1}{32}\right]; \qquad \dots$$

We deduce that for step $2k$ and $2k+1$, we have the intervals

$$I_{2k} = \left[-\frac{1}{2^{2k}}, \frac{1}{2^{2k-1}}\right], \qquad I_{2k+1} = \left[-\frac{1}{2^{2k}}, \frac{1}{2^{2k+1}}\right].$$

Because deduction does not suffice, we formalize with induction. Consider the subset $U \subset \mathbb{N}$ of k steps in our bisection that are correctly determined by our deduction. Trivially, step 0 is satisfied, and it's easy to see that step $1 = 2(0) + 1$ is correct with $S_1 = \left[-\frac{1}{2^0}, \frac{1}{2}\right] = \left[-1, \frac{1}{2}\right]$. So we have $0, 1 \in U$.
Assume $k \in U$ (we showed two things in our base case, so we can make this assumption) so that our expressions for I_{2k}, I_{2k+1} given above are correct. If we show this implies $(k+1) \in U$, then by the induction axiom, we have $U = \mathbb{N}$ and we are done. Consider:

$$\begin{aligned}
I_{2k} &= \left[-\frac{1}{2^{2k}}, \frac{1}{2^{2k-1}}\right] \quad \text{(inductive hypothesis)} \\
I_{2k+1} &= \left[-\frac{1}{2^{2k}}, \frac{1}{2^{2k+1}}\right] \quad \text{(inductive hypothesis)} \\
I_{2k+2} = I_{2(k+1)} &= \left[\frac{1}{2}\left(\frac{-1}{2^{2k}} + \frac{1}{2^{2k+1}}\right), \frac{1}{2^{2k+1}}\right] = \left[-\frac{1}{2^{2k+2}}, \frac{1}{2^{2k+1}}\right] \\
I_{2k+3} = I_{2(k+1)+1} &= \left[-\frac{1}{2^{2(k+1)}}, \frac{1}{2}\left(\frac{-1}{2^{2k+2}} + \frac{1}{2^{2k+1}}\right)\right] = \left[-\frac{1}{2^{2(k+1)}}, \frac{1}{2^{2(k+1)+1}}\right],
\end{aligned}$$

so we have shown $k \in U \implies (k+1) \in U$, and as stated above, by induction we have $U = \mathbb{N}$ as desired.
The greatest accuracy for IEEE standard floating point arithmetic can be achieved by using a subnormal number with "gradual underflow": $(-1)^S \cdot 2^{-1022} \cdot (f)$. As given by Strain in Lecture 2, here we lose the $1 + \cdots$ but get another 51 bits before our numbers go to zero, from $2^{-1023} \to 2^{-1022}$.
Hence we need the computation of the (arithmetic mean) midpoint to underflow to 0 at the highest precision, so we let $f := 2^{-52}$. Then we need a midpoint calculation to give $|\text{midpoint}| < 2^{-1022} \cdot 2^{-52} = 2^{-1074}$. Looking at our findings above, the new midpoint in each I_j is $\pm\frac{1}{2^j}$. Thus we need

$$\left|\frac{1}{2^j}\right| < \frac{1}{2^{1074}},$$

and this of course first happens at $j = 1075$ steps, our minimum number of steps to solve x to maximum IEEE standard floating point accuracy.

□

Problem 6. Implement a $\geq$ MATLAB function `bisection.m` of the form

```
function [r,h] = bisection(a,b,f,p,t)
% a: Beginning of interval [a,b]
% b: End of interval [a,b]
% f: function handle y = f(x,p)
% p: parameters to pass through to f
% t: User-provided tolerance for interval width
```

At each step $j = 1, \ldots, n$, carefully choose m as in bisection with the *geometric* mean (watch out for zeroes!). Replace $[a, b]$ by the smallest interval with endpoints chosen from a, m, b which keeps the root bracketed. Repeat until: (1) a f value exactly vanishes, (2) $(b-1 \leq t \min\{|a|, |b|\})$, or (3) b and a are adjacent floating point numbers, whichever comes first. Return the final approximation to the root r and a $3 \times n$ history matrix `h[1:3, 1:n]` with column `h[1:3, j]` $= (a, b, f(m))$ recorded at step j. Try to make your implementation as foolproof as possible.
QUESTIONS:

(a) (See BBF 2.1.7) Sketch the graphs of $y = x$ and $y = 2 \sin x$.
(b) Use `bisection.m` to find an approximation to within ϵ to the first value of x with $x = 2 \sin x$. Report the number of steps, the final result, and the absolute and relative errors.
Use `bisection.m` as many times as needed to find all solutions in a given interval for the following equations:
(c) $x > 0 : f(x) = \frac{1}{x} + \ln x - 2 = 0$.
(d) $[-1, 2] : f(x) = (x - \epsilon^3)^3 = 0$.
(e) $[-1, 2] : f(x) = \arctan(x - \epsilon^2) = 0$.

Solution. We use a mix of Matlab and R for this problem, with code appended at the end of the document.
(a)

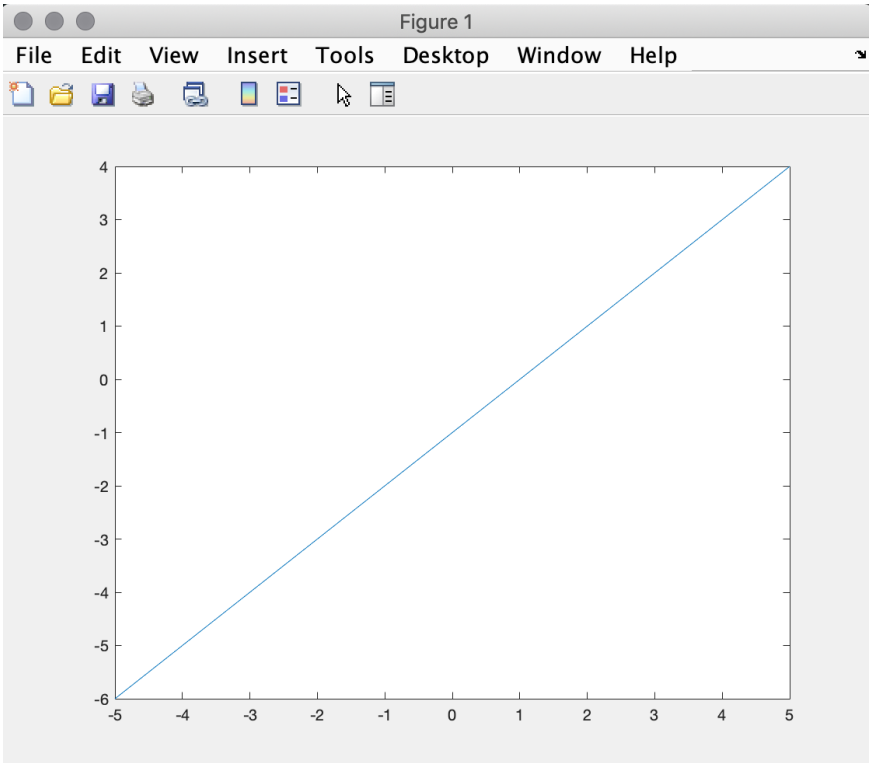

Figure 1: plot of y = x

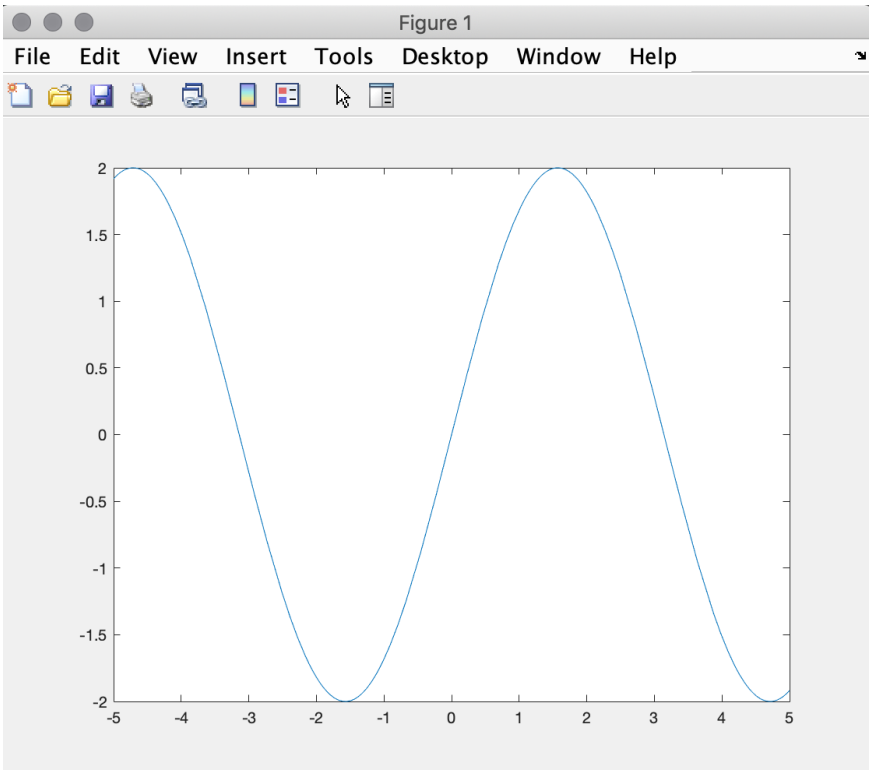

Figure 2: plot of y = 2 sin x

(b) For intuition, here's a plot of $y = x - 2sin(x)$:

(c)

Solution. We want to spam bisection to find all the roots in the given interval,

$$x > 0 : f(x) = \frac{1}{x} + \ln x - 2 = 0.$$

Solving, we get

$$x = 6.3054$$

□

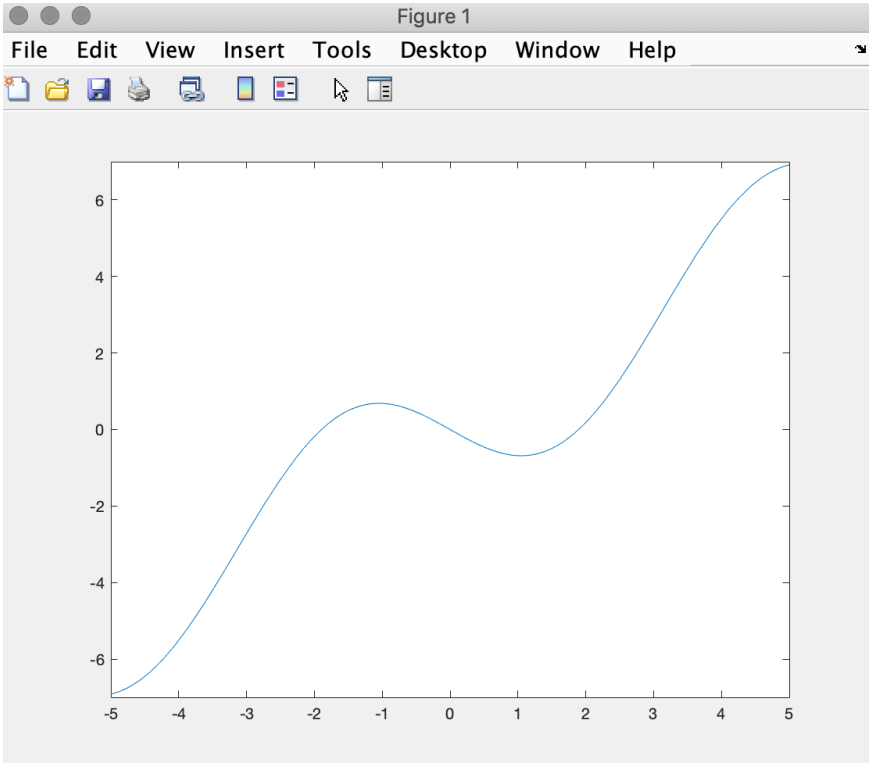

Figure 3: reference plot for y = x - 2 sin(x)

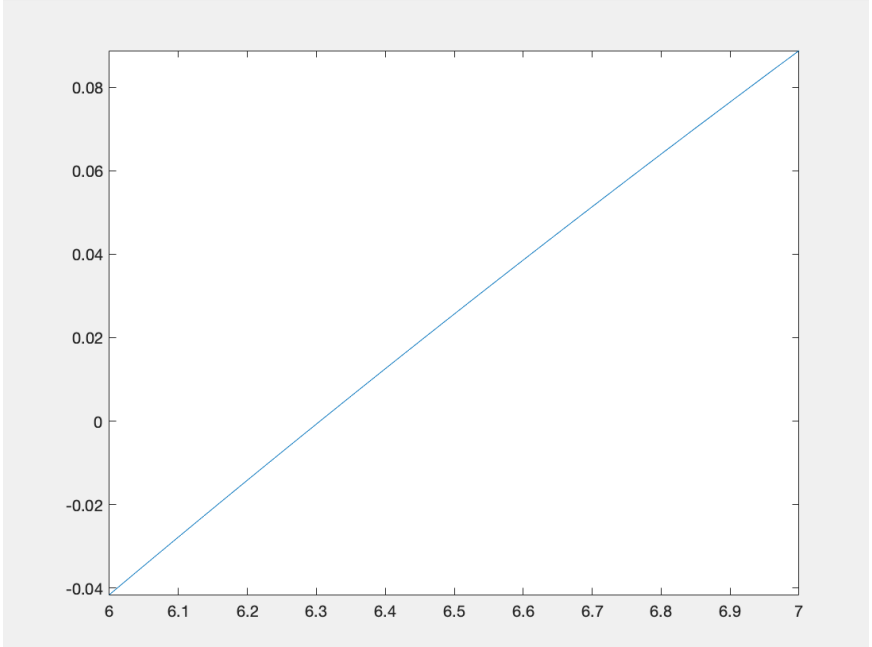

Figure 4: plot of 6c.

(d)

Solution. We want to spam bisection to find all the roots in the given interval,

$$[-1, 2] : f(x) = (x - \varepsilon^3)^3.$$

Solving, we get

$$x = -1.0015 \times 10^{-16} \approx -\frac{\varepsilon}{2}$$

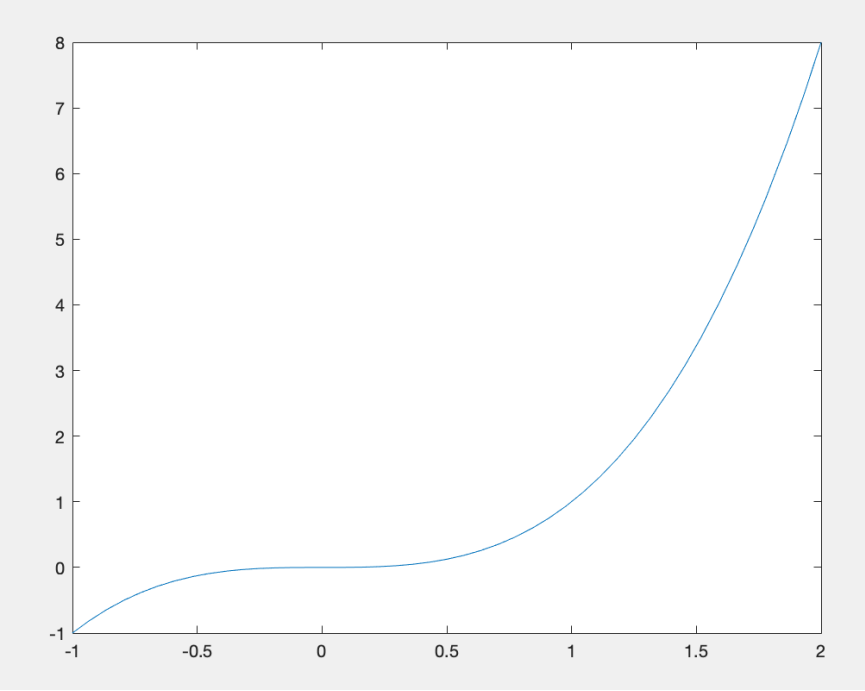

Figure 5: plot of 6d.

□

(e)

Solution. We want to spam bisection to find all the roots in the given interval,

$$[-1, 2] : f(x) = \arctan(x - \epsilon^2) = 0.$$

Solving, we get:

$$x = -8.9741 \times 10^{-27}.$$

□

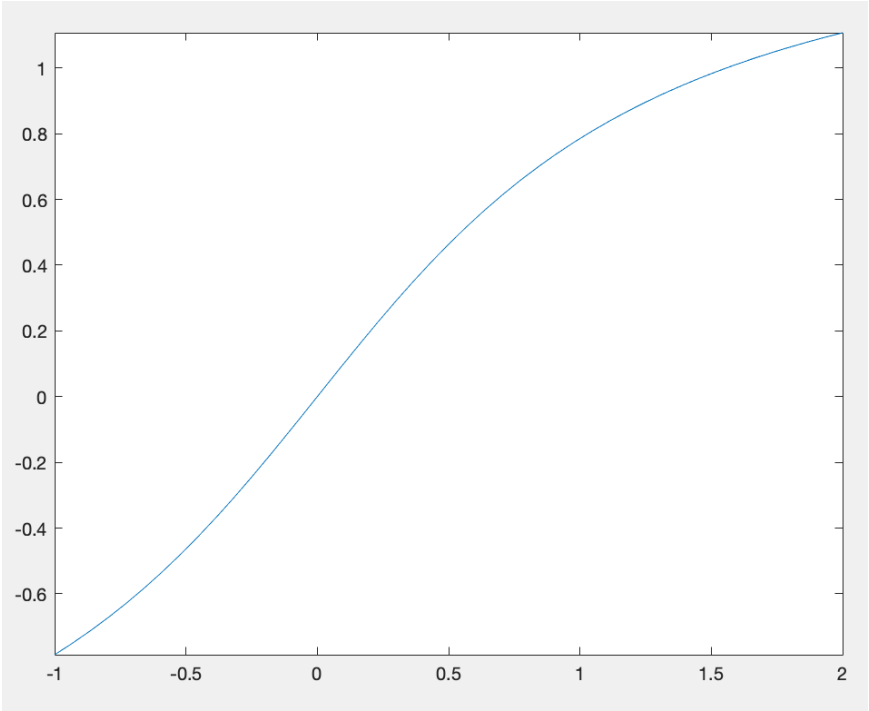

Figure 6: plot for 6e.

1 R and Matlab Code and Output

```
1
2  bisection <- function(a,b,f,p,t) {
3    # OUTPUT: [r,h] : [answer, history]
4    # a : Beginning of interval [a,b]
5    # b: End of interval [a,b]
6    # f: function handle y = f(x,p)
7    # p : parameters to pass through to f
8    # t : User-provided tolerance for interval width
9
10   # keep values in h (history) matrix
11   # h[1:3, 1:n], with column h[1:3, j] <- (a,b,f(m))
12   # recorded at step j
13   h <- list()
14
15   cntnue <- 1
16   while (cntnue) {
17     # new assignments
18     m <- geomean(a,b)
19     fa <- f(a,p)
20     fb <- f(b,p)
21     fm <- f(m,p)
22
23     if (fa == 0) {
24       r <- a
25       h <- c(h, list(a,b,fm))
26       rh <- c(list(r), h)
27       cntnue <- 0
28     } else {
29       if ( fb == 0 ) {
30         r  <- b
```

```
          h <- c(h, list(a,b,fm))
          rh <- c(list(r), h)
          cntnue <- 0
        } else {
          r <- m # return m for next iteration
          h <- c(h, list(a,b,fm))
          rh <- c(list(r), h)
          cntnue <- 1
        }
        }

      # case: END bc (b-a) \leq t \min (|a|, |b|)
      if ( (b-a) <= t * min( abs(a), abs(b) ) ) {
        # do nothing, just end
        cntnue <- 0
      }

      # case: a,b are adjacent floating point numbers
      if ( a == m || b == m ) {
        cntnue <- 0
      }

      # bisection
      if ( sign(fa) != sign(fm) ) {
        b <- m
      } else {
        a <- m
      }

    } # while


  }

```

And to compute the geometric mean of bisection, we have `geomean.r`:

```

geomean <- function(a,b=1) {
  # finds geometric mean of a,b
  # m <- \sqrt{ab}
  #
  # Catch edge cases
  if (a == 0 || b == 0){
    m <- .Machine$double.xmin # smallest positive normal number in double precision, approx. 2.225
     e^{-308}
  }
  else {
    if (sign(a) != sign(b)){
      m <- 0 # otherwise would give complex output
    }
    else {
      # both share same sign;
      # use absolute values then take sign
      m <- sign(a) * sqrt(abs(a)) * sqrt(abs(b))
    }
  }
}
```

Or equivalently, in Matlab:

```

function [r,h ] = bisection(a, b, f, p, t)
% a: Beginning of interval [a,b]
% b: End of interval [a,b]
% f: function handle y = f(x,p)
% p: parameters to pass through to f
```

```
% t: User-provided tolerance for interval width

    while (true)
        m = geomean(a, b);
        fa = f(a, p);
        fb = f(b, p);
        fm = f(m, p);

        % END CASE: if f vanishes
        if (fa == 0)
            r = a;
            h = [h, [a; b; fm]];
            break % = exit while
        elseif (fb == 0)
            r = b;
            h = [h, [a; b; fm]];
            break
        else
            r = m; % just output m and continue
            h = [h, [a; b; fm]];
        end

        % END CASE: a approx= b
        if ( (b - a) <= t * min( abs(a), abs(b) ) )
            break
        elseif ( m == a )
            break
        elseif ( m == b )
            break
        end

        % proceed with bisection
        if ( sign(fa) ~= sign(fm) )
            b = m;
        else
            a = m;
        end
    end
end
```

```

function m = geomean(a,b)

    % finds geometric mean of a,b
    % m = \sqrt{ab}
    % turns a,b to positives then brings back sign

    % catch edge cases
    % trying to compute zero
    if (a == 0)
        m = realmin; % smallest positive normal number in double
    elseif (b == 0)
        m = - realmin ; % negative of such
    % a,b opposite signs, 0 should be closer to the 'bisection'
    elseif ( sign(a) ~= sign(b) )
        m = 0;
    else
        m = ( sign(a) * sqrt( abs(a) * abs(b) ) );
    end
end
```

□