Homework No.1, 553.481/681, Due February 5, 2021.
Andrew Cornelio

Problem 1. (a) Show that to convert between binary and hexadecimal representations, each hecadecimal digit b is replaced, in order, by the four binary digits a, a', a'', a''' that satisfy

$$b = 2^3 a + 2^2 a' + 2a'' + a'''.$$

Solution: We can represent any hexadecimal number h by a sequence of hexdigits $b_i \in \{0, 1, 2, ..., 9, a, b, ...f\}$ such that:

$$h = \sum_{i=0}^{n} b_i \times (10_{16})^i$$

Note that 10_{16} is a hexadecimal number that is equal to 16 in decimal or 10000 in binary. Since a single hexdigit can represent the numbers from 0 to 15 inclusive in binary, we can also represent a hex digit with 4 binary number sequence since the range of binary numbers from 0000 to 1111 represents the range from 0 to 15. Combining these two facts, we can rewrite the sum:

$$h = \sum_{i=0}^{n} (a_i \times 1000_2 + a_i' \times 100_2 + a_i'' \times 10_2 + a_i''' \times 1_2) \times (10000_2)^i$$

Where $a_i, a_i', a_i'', a_i''' \in \{0, 1\}$. But with this sum, we can now see that at each subsequent iteration, the product shifts the least significant digit of the current iteration 4 places to the left of the least significant digit of the previous iteration. In contrast, the most significant digit of the previous iteration is only 3 places to the left compared to the least significant digit of that iteration. Hence, there is no overlap between the digits of different iterations. Therefore, we can replace every hexdigit of a hexadecimal number with its binary representation independently without looking at other digits.

(b) Use part (a) to write a Matlab script `hextobin.m` which converts any n-dimensional vector $[h_1, ..., h_n]$ of the values $h_i \in \{0, 1, 2, ..., 9, a, b, ...f\}$, with the definitions

```
a=10;b=11;c=12;d=13;e=14;f=15;
```

into a vector $[b_1, ..., b_{4n}]$ of the values $b_i \in \{0, 1\}$. *Hint:* Use Matlab's `mod` function.

Note: Function requires input argument to be a string of hex digits.

(c) Apply your script from part (b) to find the binary equivalent of the hexadecimal string `1fbc47a3`.

Solution: The output of the function was `00011111101111000100011110100011`.

Problem 2. In IEEE Standard Floating-Point representation for numbers, a 12-bit binary number is used first to encode the sign σ and exponent E, followed by a 52-bit binary to encode the fraction F. This can also be represented by a 3-digit hexadecimal number, followed by a 13-digit hexadecimal number, as in `MATLAB`. Consider the following IEEE Standard Floating-Point number in hexadecimal form:

```
x=3fbf9add37a88fe8
```

Citation: For hex/binary/decimal conversions, I used this online tool:
https://www.rapidtables.com/convert/number/hex-to-decimal.html

(a) Use the first 3 hexadecimal digits to determine the sign σ and exponent E of `x` in decimal format.

Solution: The first 3 hexdigits are $\texttt{3fb}_{16}$. This corresponds to 001111111011_2 in binary. Therefore we can see $\sigma = (-1)^0 = 1$. The next 11 bits technically correspond to $E+B$, the exponent plus the bias. Since we are working in double precision floating point, $B = 2^{11-1} - 1 = 1023$. So now we can convert binary to decimal and compute $E = 1019 - 1023 = -4$.

(b) Convert the remaining 13 hexadecimal digits to a decimal representation of the fraction F for `x`.

Solution: The last thirteen hexdigits can be written in binary as $1111100110101101110100110111101010001000111111101000_2$. To convert these to a decimal representation of the mantissa, we must use the formula:

$$F = \sum_{i=1}^{52} b_i \times 2^{-i}$$

Where b_j is the binary digit j positions to the right of the first (most significant) bit. We can use Matlab to obtain the results: $F = 0.975308625617794$

(c) Calculate the decimal representation of the double-precision number `x`. You can use the Matlab intrinsic function `hex2num` to check your answer in (c), but you must explain how the result follows from parts (a) and (b).

Solution: Finally, we need to combine the results. We use the formula presented in lecture:

$$\begin{aligned} x &= \sigma M \times 2^E = \sigma(1+F) \times 2^E \\ &= 1 \times 1.975308625617794 \times 0.0625 = 0.12345678910111213 \end{aligned}$$

Problem 3. This problem concerns a floating-point arithmetic with base β, precision p, and exponents E in the range $L \leq E \leq U$.

(a) Show that the underflow level for normalized numbers is $UFL = \beta^L$.

Solution: Normalized numbers can be written as:

$$x = \pm(x_0.x_1x_2x_3 \ldots x_p)_\beta \times \beta^E$$

Where x_0 must be a non zero number. The lowest unsigned number that satisfies this requirement is:

$$x = (1.000 \ldots 0)_\beta \times \beta^L = \beta^L$$

Any number lower would require $E < L$, which is impossible, or $x_0 = 0$, in which case the number would not be normalized.

(b) Show that overflow level is $OFL = (\beta - \beta^{-p})\beta^U$.

Solution: Based on the formula presented in the last part, the highest number that can be created with normalized numbers is:

$$x = (\alpha.\alpha\alpha\alpha \ldots \alpha)_\beta \times \beta^U$$

Where the digit $\alpha = \beta - 1$. We can represent this as:

$$\begin{aligned} x &= (\alpha(\beta)^0 + \alpha(\beta)^{-1} + \alpha(\beta)^{-2} + \alpha(\beta)^{-3} + \ldots + \alpha(\beta)^{-p})_\beta \times \beta^U \\ &= ((\beta-1)(\beta)^0 + (\beta-1)(\beta)^{-1} + (\beta-1)(\beta)^{-2} + (\beta-1)(\beta)^{-3} + \ldots + (\beta-1)(\beta)^{-p})_\beta \times \beta^U \\ &= (\beta - \beta^{-p})\beta^U \end{aligned}$$

Note that writing any larger number would require $E > U$ or x_0 to consist of more than one digit, both of which are impossible.

(c) Show that the underflow level for denormalized numbers is $UFL_* = \beta^{L-p}$. **Solution:** The smallest non zero number that can be represented by a denormalized number is:

$$x = (0.000\ldots1)_\beta \times \beta^L$$

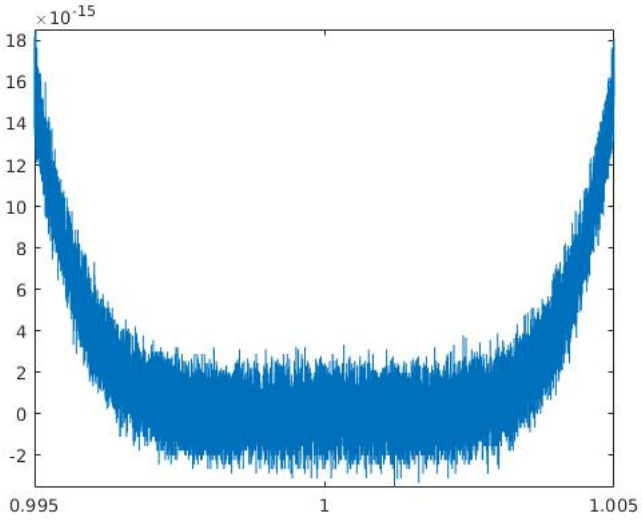
Note that while signed zero is a smaller number, it does not have any precision, and we are only concerned with numbers with finite precision that can be used in calculations. We can rewrite this number simply as:

$$x = (1(\beta^{-p}))_\beta \times \beta^L = \beta^{L-p}$$

Problem 4.
(a) Use Matlab's function `fplot` to plot the function

$$f(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$$

over the x-interval $[0.995, 1.005]$. Does the result look like the plot of a smooth polynomial?



Solution: No it does not look smooth.

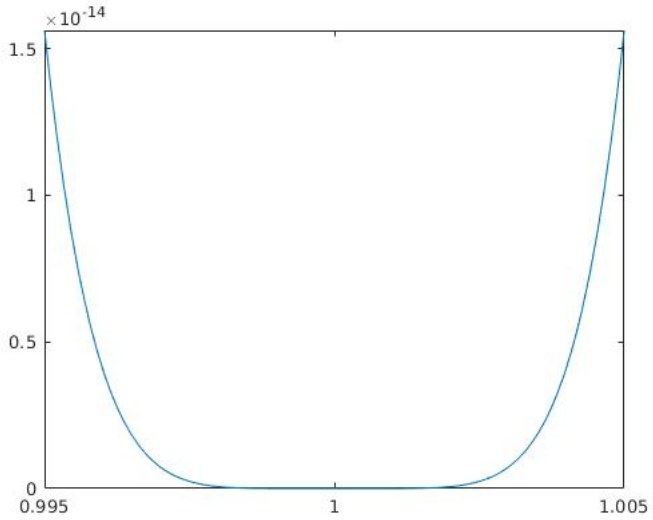
(b) Matlab's function `roots` approximates the n roots of a general nth-degree polynomial $p(x) = c_n x^c + c_{n-1}x^{n-1} + \cdots c_1 x + c_0$ by numerically computing the eigenvalues of a so-called "companion matrix". Calculate the 6 roots of the polynomial in part (a) using the function `roots` and compare them with the exact roots = 1 obtained by factorizing the polynomial. Do the approximate roots agree with the exact roots to double precision accuracy? Explain why this should have been expected based upon the results in part (a).

Solution: The roots to 4 decimal places are: 1.0030 + 0.0017i, 1.0030 - 0.0017i, 1.0000 + 0.0034i, 1.0000 - 0.0034i, 0.9971 + 0.0017i, 0.9971 - 0.0017i. They do not agree with the expected value of r=1 to double precision accuracy. This however is expected, since the plot is extremely noisy around the $x = 1$.

(c) Write the function in part (a) in factorized form

$$f(x) = (x-1)^6$$

and then use `fplot` to plot the function in this form over the same interval $[0.995, 1.005]$. How does the plot compare with that in part (a)? Explain why the plots differ so drastically.



Solution: The plot is much smoother compared the the plot generated in part (a). The plots differ because the we reduce the amount of additions and subtractions in the computation. Cancellations in a chain on additions and subtractions introduce more machine arithmetic error than a chain of multiplications.

(d) Use the Matlab function `roots` to find the zeros of the polynomial $g(y) = f(1+y) = y^6$ and those of $f(x)$ by setting $x = y + 1$. Are the roots now correct to double precision?

Solution: Matlab reports the roots of g(y) as all being exactly $y = 0$ in double precision. If we substitute $x = y + 1$, then we see that $x = 1$ in double precision.

Problem 5. Consider the following linear system

$$\begin{aligned} x + \alpha y &= 1 \\ \alpha x + y &= 0 \end{aligned}$$

for the real parameter α.

(a) Find the exact solutions $x(\alpha)$, $y(\alpha)$ as functions of α. Calculate the infinitesimal condition numbers $K_x(\alpha)$, $K_y(\alpha)$ of these functions. How do these behave for $\alpha \to 1$?

Solution: We can solve this system through substitution and obtain:

$$x(\alpha) = \frac{1}{1-\alpha^2} \qquad y(\alpha) = -\frac{\alpha}{1-\alpha^2}$$

The condition numbers of a function can be found through the formula:

$$K_f(x) = \left| \frac{x}{f(x)} \frac{df}{dx} \right|$$

This is straightforward calculation:

$$K_x(\alpha) = \left| \frac{\alpha}{\frac{1}{1-\alpha^2}} \frac{2\alpha}{(\alpha^2-1)^2} \right| \qquad K_y(\alpha) = \left| \frac{\alpha}{-\frac{\alpha}{1-\alpha^2}} \frac{-(\alpha^2+1)}{(\alpha^2-1)^2)} \right|$$

$$= \left| \frac{2\alpha^2}{1-\alpha^2} \right| \qquad = \left| \frac{1+\alpha^2}{1-\alpha^2} \right|$$

We can easily see that both $K_x(\alpha)$ and $K_y(\alpha)$ increase unbounded as α approaches 1.

(b) Find the function $z(\alpha) = x(\alpha) + y(\alpha)$ and simplify its form as much as possible. Calculate the infinitesimal condition number $K_z(\alpha)$ and determine its behavior for $\alpha \to 1$.

Solution: First we combine our results:

$$z(\alpha) = \frac{1}{1-\alpha^2} - \frac{\alpha}{1-\alpha^2} = \frac{1-\alpha}{1-\alpha^2} = \frac{1}{1+\alpha}$$

Again, we just apply the formula:

$$K_z(\alpha) = \left| \frac{\alpha}{\frac{1}{1+\alpha}} \frac{-1}{(\alpha+1)^2} \right|$$
$$= \left| \frac{\alpha}{1+\alpha} \right|$$

We can easily see that both $K_z(\alpha)$ approaches 0.5 as α approaches 1.

(c) Evaluate the function $z(\alpha)$ for $\alpha = 1 - 10^{-8}$ by solving the given linear system using the Matlab function `mldivide(A,b)` or, equivalently, `A\b` and then evaluate the function $z(\alpha)$ by summing the two components of the solution vector. Find a second approximation to $z(\alpha)$ for the same value of α by directly evaluating the expression in (b). What is the relative difference between the two approximations? Which value do believe is more accurate and why?

Solution: By the first method, I obtained $z =$ `0.500000000000000`. By the second method, I obtained $z2 =$ `0.500000002500000`. The difference is `2.5e-9`. I trust the second value since the magnitude of the condition number is smaller for that method of computation.

Problem 6*. (a) Show that the following infinite series can be exactly evaluated as

$$S := \sum_{n=1}^{\infty} \frac{2n+1}{n^2(n+1)^2} = 1.$$

Show furthermore that the kth partial sum S_k and its remainder $R_k = 1 - S_k$ are given by

$$S_k = \sum_{n=1}^{k} \frac{2n+1}{n^2(n+1)^2} = 1 - \frac{1}{(k+1)^2}, \quad R_k = \frac{1}{(k+1)^2}.$$

Solution: We can use partial fraction decomposition to see that for all $n \in \mathbb{N}$,

$$\frac{2n+1}{n^2(n+1)^2} = \frac{1}{n^2} - \frac{1}{(n+1)^2}$$

First, we will show that the kth partial sum S_k is as described above. We write the sum:

$$S_k = \sum_{n=1}^{k} \frac{2n+1}{n^2(n+1)^2} = \sum_{n=1}^{k} \left[\frac{1}{n^2} - \frac{1}{(n+1)^2} \right]$$

However, we can simply see that this is a telescoping series, and after all the cancellations, the only terms that survive are the first and last:

$$S_k = 1 - \frac{1}{(k+1)^2}$$

Now, we want to check if the infinite series has a sum. In order to do this, we must check that the partial sums converge in the infinite limit:

$$\begin{aligned} S = \lim_{k\to\infty} S_k &= \lim_{k\to\infty} \left[1 - \frac{1}{(k+1)^2} \right] \\ &= 1 - \lim_{k\to\infty} \frac{1}{(k+1)^2} \\ &= 1 \end{aligned}$$

Since the limit exists, the series converges and we can say $S_k := 1$. We can find the remainder of each partial sum by simply subtracting the partial sum from 1.

(b) Consider the following Matlab script to evaluate approximately the infinite series:

```
nn=1;
dS=(2*nn+1)/nn^2/(nn+1)^2;
Sold=0;
SS=dS;
while (abs(SS-SSold)>0)&(nn<1e6)
    nn=nn+1;
    dS=(2*nn+1)/nn^2/(nn+1)^2;
    Sold=SS;
    SS=SS+dS;
end
```

Run the code and state the approximate value of S obtained, the error, and the largest value of n employed. Is the error consistent with the exact result for the remainder?

Solution: The the algorithm terminates when $S = 0.999999999992635$, $n = 330281$, and the error is $1 - 0.999999999992635 = 7.365e-12$. However, by using the formula from b with the last value of n, I found that the remainder should be $9.167e-12$.

(c) Was double precision accuracy obtained in part (b)? If so, explain why. Otherwise, explain why the algorithm failed to achieve a double precision result and, if you can, devise a more accurate numerical approach.

Solution: Double precision was not obtained for this result. The chain of additions introduced some error. A more straightforward approach would be to use the partial sum formula derived in part b.