Problem 1. This is K&C 2.2.24, reproduced for convenience. In computing the infinite sum ∑∞n=1 xn, suppose that we want the answer with an absolute error at most some value ϵ (arbitrary, not machine epsilon). Is it safe to stop the addition of terms when the magnitude falls below ϵ? Illustrate by setting xn = 0.99n.

Problem 2. Consider the problem of finding the roots of

\[ ax^2 + bx + c = 0 \]

where a = c = 1 and b = 10^3. Suppose we are using floating point numbers in base ten with rounding and 4 digits past the decimal (so 1.00005 rounds up to 1.0001).

i) Compute the two roots (using floating point arithmetic) with the quadratic formula

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \]

Verify that one of the computed roots is zero. Why does this not make sense?

ii) Derive a new formula by multiplying the numerator/denominator by \(-b \mp \sqrt{b^2 - 4ac}\). Use it to compute the roots again. Does this work any better?

\[ ^1\text{Problem adapted from Alan J. Laub, } Computational Matrix Analysis. \]
Problem 3. Let $x$ be a real number given by
\[ x = (1 + f) \times 2^e, \quad f = (0.d_1d_2\cdots)_2. \]
Suppose we have a binary floating point system with $N$ digits. Two schemes for $\text{fl}(x)$ are

\textbf{truncation}: $\text{fl}(x) = (1 + \tilde{f}) \times 2^e$, $\tilde{f} = (0.d_1d_2\cdots d_N)_2$,

\textbf{rounding}: $\text{fl}(x) = (1 + \tilde{f}) \times 2^e$, $\tilde{f} = \begin{cases} (0.d_1d_2\cdots d_N)_2 \\ (0.d_1d_2\cdots d_N)_2 + 2^{-N} \end{cases}$ $d_{N+1} = \begin{cases} 0 \\ 1 \end{cases}$.

(Here ties are broken by rounding up).

a) Show that if truncation is used then
\[ \frac{|\text{fl}(x) - x|}{|x|} \leq 2^{-N}. \]

b) Show that if rounding is used then the bound can be improved to
\[ \frac{|\text{fl}(x) - x|}{|x|} \leq 2^{-(N+1)}. \]

Problem 4. Consider the problem of evaluating an $n$-th degree polynomial
\[ P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \]
at a point $x$.

a) The algorithm below uses the formula (1) to compute $P_n(x)$. How many operations (multiplications and additions) are required?

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Algorithm 1 Naïve polynomial evaluation}
\State \textbf{Input:} $n \geq 0, x \in \mathbb{R}$ and $A = [a_n \ a_{n-1} \cdots \ a_0]$
\State \textbf{Output:} $y = P_n(x)$
\State $y \leftarrow a_0$
\State $z \leftarrow x$ \Comment{stores $x^i$}
\For{$i = 1, \cdots, n - 1, n$}
\State $y \leftarrow y + za_i$
\State $z \leftarrow xz$
\EndFor
\State \textbf{return} $y$
\end{algorithmic}
\end{algorithm}

b) A better approach is Horner’s method, which proceeds by writing
\[ P_n(x) = a_0 + x(a_1 + a_2(x + \cdots + x(a_{n-1} + xa_n))\cdots). \]
Write an algorithm (in pseudocode) for calculating $P_n(x)$ using Horner’s method. How many operations are required?
c) In MATLAB, the convention is to represent the polynomial (1) using a list $A$ of length $n + 1$:

$$A = [a_n \ a_{n-1} \cdots \ a_0]$$

Write a function `horner(A,X)` that takes a list of $m$ points $X = [x_1 \cdots x_m]$ and coefficient list $A$ and outputs the polynomial evaluated at those points, i.e. the array $Y = [P(x_1) \cdots P(x_m)]$. Turn in this code.\(^2\)

\(^2\)MATLAB’s command is to do this is `polyval(A,x)`, which uses Horner’s method.

d) Consider the polynomial $P(x) = (x - 1)^7$. Written out, this is

$$P(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1.$$ 

Compare the results of Horner’s method and the (nearly) ‘exact’ calculation $(x - 1)^7$ in the intervals $[0.998, 1.002]$ and $[-1.002, -0.998]$ (suggestion: make a plot). Comment on the accuracy in each case (if there is a notable error, offer a plausible explanation).

**Code considerations:** Use element-wise operations on vectors, e.g. $X.*Y$ to compute $[x_1y_1 \cdots x_ny_n]$. If using `numpy`, the same can be done using the numpy array type ($*$ acts element-wise by default).

To initialize the output with the right shape, you may have to use `zeros` or `ones` to make an array of all zeros/ones.