**Math 125** **Problem Set 1** **Due: Monday 9/19 at 11:59 p.m. EST**

**Instruction**: Read the assignment policy. For problem 5(c), include a printout your code with your homework submission. You should submit your assignment on Gradescope.

**1**. For each of the following expressions, indicate for what values catastrophic cancellations could occur. Find an alternative expression that avoids the cancellation.

(a) $f(x) = \dfrac{\sqrt{25+x}-5}{x}$. Show that the alternative expression correctly computes $\lim_{x\to 0} f(x)$.

(b) $f(x) = \dfrac{1-\cos(x)}{x^2}$.

(c) $f(x) = \dfrac{1-\sec(x)}{\tan^2(x)}$.

**Solution:**

(a) Catastrophic cancellations occur when $x \approx 0$. In this range, the term $\dfrac{\sqrt{25+x}-5}{x}$ is prone to catastrophic cancellation. An alternative expression that avoids the cancellation is

$$\frac{\sqrt{25+x}-5}{x} \cdot \frac{\sqrt{25+x}+5}{\sqrt{25+x}+5} = \frac{1}{\sqrt{25+x}+5}.$$

We can also see that $\lim_{x\to 0} \dfrac{1}{\sqrt{25+x}+5} = \dfrac{1}{10}$. Note that this is the same limit obtained if we apply l'Hospital's rule on $f(x)$.

(b) Catastrophic cancellations occur when $x \approx 0$. We consider the trigonometric identity $\cos(x) = 1 - 2\sin^2(x/2)$. An alternative expression that avoids the cancellation is

$$\frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)$$

We note that $\lim_{x\to 0} \dfrac{1}{2}\left(\dfrac{\sin(x/2)}{x/2}\right) = \dfrac{1}{2}$.

(c) Catastrophic cancellations occur when $x \approx 2\pi n$ where $n$ is an integer. An alternative expression that avoids the cancellation is

$$\frac{1-\sec(x)}{\tan^2(x)} \cdot \frac{1+\sec(x)}{1+\sec(x)} = \frac{1-\sec^2(x)}{\tan^2(x)(1+\sec(x)} = -\frac{1}{1+\sec(x)},$$

where the last equality uses the trigonometric identity that $1+\tan^2(x) = \sec^2(x)$.

**2**. For each of the following expressions, indicate whether underflow, overflow or both could occur. Find an alternative expression that avoids these issues.

(a) Computing the determinant of a matrix $\mathbf{A} \in \mathcal{R}^{n \times n}$.
**Remark**: Any matrix $\mathbf{A}$ has the following decomposition $\mathbf{A} = \mathbf{LU}$ where $\mathbf{L}$ is lower-triangular and $\mathbf{U}$ is upper-triangular. This is known as the LU decomposition which we cover later in the course. With that, $\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{U})$. Since $\mathbf{L}$ and $\mathbf{U}$ are triangular matrices, the determinants are simply the product of the diagonal entries. Given this, without loss of generality, you can consider computing the determinant of an upper triangular matrix $\mathbf{A}$ and determine when underflow/overflow occurs.

(b) Computing the length of a vector $\mathbf{x} \in \mathcal{R}^n$ using $||\mathbf{x}|| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$.

**Solution:**

(a) Based on the remark, it suffices to consider underflow/overflow when taking the product of $n$ numbers. To lessen the effects of overflow and underflow, we compute the logarithm of the determinant itself. To be precise, we have

$$\log(|\det(\mathbf{A})|) = \log\left(|\lambda_1| \cdot |\lambda_2| \ldots |\lambda_n|\right) = \sum_{i=1}^{n} \log(|\lambda_i|).$$

Two remarks are in order. First, we compute $|\det(\mathbf{A})|$ since the sign can be computed separately. Second, taking the logarithm helps with both underflow and overflow.

(b) Let $c = \max(|x_1|, |x_2|, ..., |x_n|)$. To avoid overflow, we can consider the following alternative expression

$$c\sqrt{\left(\frac{x_1}{c}\right)^2 + \left(\frac{x_2}{c}\right)^2 + ... + \left(\frac{x_n}{c}\right)^2}.$$

Note that each of the term in the summand i.e $\frac{x_i}{c}$ is bounded above by 1.

**Remark**: It is important to note that overflow and underflow errors can not be eliminated for all problems for all ranges of values. The idea of finding an alternative expression is to facilitate computation (by lessening the effects of overflow or underflow) in a desired interval of interest.

**3**. This question concerns floating point arithmetic.

(a) The binary representation of a certain number $x$ is 101110. What is $x$ in base 10?

(b) Find the binary representations of 0.375 and 1.25. Consider the sum of the two numbers in their binary representations and show that it agrees with the sum in the standard base 10 representation.

(c) Recall the representation of a number in single precision: $(-1)^s \cdot 2^{e-127} \cdot (1 + f)$ where $s$ denotes the 1-bit sign, $e$ denotes the 8-bit exponent and $f$ denotes the 23-bit fraction. Assume that an exponent of all zeros and an exponent of all ones are special and reserved. What is the smallest number that can be represented in single precision? What is the largest number that can be represented in single precision?

(d) Give an example for a number that does not have an exact representation either in single or double precision arithmetic.

(e) Prove that the floating numbers in the range $[2^k, 2^{k+1}]$ are just the numbers in $[1, 2]$ multiplied by $2^k$. What does this inform you about the gaps between small numbers as opposed to gaps between large numbers?

**Solution:**

(a) The number $x$ is $x = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 = 2 + 4 + 8 + 32 = 46$.

(b) To find the representation of 0.375, we consider the following equation.

$$0.375 = x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + ... + x_n \cdot 2^{-n},$$

where $x_1, x_2, ..., x_n$ are the unknowns. Multiplying the above expression by 2 on the sides, we obtain

$$0.75 = x_1 + x_2 \cdot 2^{-1} + ... + x_n \cdot 2^{1-n}$$

Note that $x_2 \cdot 2^{-1} + ... + x_n \cdot 2^{1-n} \leq \sum_{i=1}^{i=n-1} 2^{-i} = 1 - \left(\frac{1}{2}\right)^{n-1} < 1$. Therefore, $x_1 = 0$. Multiplying both sides of the equation above by 2 on both sides, we obtain

$$1.5 = x_2 + x_3 \cdot 2^{-2} + ... + x_n \cdot 2^{2-n}$$

In a similar way as before, $x_3 \cdot 2^{-2} + ... + x_n \cdot 2^{2-n} < 1$. Therefore, $x_2 = 1$. Multiplying both sides of the equation above by 2 on both sides, we obtain

$$1 = x_3 + x_4 \cdot 2^{-1} + ... + x_n \cdot 2^{3-n}$$

In a similar way as before, $x_4 \cdot 2^{-1} + ... + x_n \cdot 2^{3-n} < 1$. Therefore, $x_3 = 1$ and $x_4 = x_5 \ldots x_n = 0$. Therefore, the binary representation of 0.375 is 0.011. To find the binary representation of 1.25, note that $1.25 = 2^0 + 2^{-2}$. Therefore, its binary representation is 1.010. The sum of the two numbers in binary is $0.011 + 1.010 = 1.101$. The base 10 representation is $2^0 + 2^{-1} + 2^{-3} = 1.625$ which agrees to the standard sum.

(c) For the smallest number, all the fraction bits are zero. Since an exponent of all zeros is reserved, we set the exponent 1. The smallest positive number that can be represented is $2^{-126}$. To find the largest number, we set all the fraction bits to one. Since an exponent of all ones is reserved, we set the exponent to 254. The largest positive number that can be represented is

$$\left(1 + (2^{-1} + 2^{-2} + .... + 2^{-23})\right) \cdot 2^{254-127} = 2^{127} \cdot \left(1 + \frac{2^{-1}(1 - 2^{-23})}{\frac{1}{2}}\right) = 2^{127}(2 - 2^{-23})$$

(d) Any irrational number. Another example is 0.1.

(e) Without loss of generality, let's assume double precision with normalized representation. Any number between 1 and 2 can be represented as

$$1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + ...x_{52}2^{-52},$$

where $x_i \in \{0, 1\}$ for any $1 \leq i \leq 52$. The numbers are as follows

$$[1\ 1 + 2^{-52}\ 1 + 2 \cdot 2^{-52}\ 1 + 3 \cdot 2^{-52}\ 1 + 4 \cdot 2^{-52} \ldots 2)$$

Any number between 2 and 4 can be represented as

$$(1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + ...x_{52}2^{-52}) \cdot 2^1,$$

where $x_i \in \{0, 1\}$ for any $1 \leq i \leq 52$. The numbers are as follows

$$[2 \quad 2 + 2^{-51} \quad 2 + 2 \cdot 2^{-51} \quad 2 + 3 \cdot 2^{-51} \quad 2 + 4 \cdot 2^{-51} \ldots 4)$$

Similarly, the numbers between $2^k$ and $2^{k+1}$ are

$$[2^k \quad 2^k + 2^{k-52} \quad 2^k + 2 \cdot 2^{k-52} \quad 2^k + 3 \cdot 2^{k-52} \quad 2^k + 4 \cdot 2^{k-52} \ldots 2^{k+1})$$

We note that the $i$-th number in the interval is $2^k + (i-1)2^{k-52}$ which is exactly $2^k$ times the $i$-th number in the $[1, 2)$ interval which is $1 + (i-1)2^{-52}$. Hence, all intervals contain equal numbers but the gaps are getting larger and larger. As discussed in class, this indicates that the floating numbers are dense close to zero and get sparse as we get further on the number line.

**4**. Consider the normalized representation of a number $x$ in some base $B$ as follows

$$x = \pm B^e \sum_{k=m}^{k=0} d_k B^{-k} \quad d_0 = 1, d_k \in \{0, 1, 2, ..., B-1\} \text{ for } k > 0$$

In the above representation, $m$ denotes the precision. We now consider the case where a number $y$ may not have a finite representation.

$$y = \pm B^e \sum_{k=\infty}^{k=0} d_k B^{-k} \quad d_0 = 1, d_k \in \{0, 1, 2, ..., B-1\} \text{ for } k > 0$$

To obtain a finite representation, we truncate all digits $d_{m+1}$ forward. The rounded representation of $y$ is as follows

$$\text{fl}(y) = \pm B^e \sum_{k=m}^{k=0} d_k B^{-k} \quad d_0 = 1, d_k \in \{0, 1, 2, ..., B-1\} \text{ for } k > 0$$

Prove that $\dfrac{|y - \text{fl}(y)|}{|y|} \leq B^{-m}$.

**Solution:** We note the following equations

$$\begin{aligned}
\frac{|y - \text{fl}(y)|}{|y|} &= \frac{B^e \sum_{k=m+1}^{k=\infty} d_k B^{-k}}{B^e \sum_{k=\infty}^{k=0} d_k B^{-k}} \\
&\leq \frac{B^e \sum_{k=m+1}^{k=\infty} (B-1) B^{-k}}{B^e} \\
&= (B-1) \sum_{k=m+1}^{k=\infty} B^{-k} = (B-1) \frac{B^{-(m+1)}}{1 - \frac{1}{B}} = B^{-m}.
\end{aligned}$$

**5**. This question concerns polynomial evaluation.

(a) Find an efficient method for evaluating the polynomial $P(x) = 3x^5 + 6x^8 - 2x^{11} + 9x^{14}$. [*Hint*: First do some re-arranging before applying Horner's scheme].

(b) Consider the polynomial $p(x) \equiv a_0 + a_1 x + a_2 x^2 + ... + a_k x^k$ . For fixed $x_0 \in \mathcal{R}$, define $c_0, ..., c_k \in \mathcal{R}$ recursively as follows:

$$\begin{aligned} c_k &\equiv a_k \\ c_i &\equiv a_i + c_{i+1} x_0 \quad \forall i < k. \end{aligned}$$

Show $c_0 = p(x_0)$. How many operations are needed to compute $p(x_0)$?

(c) Implement Horner's method and test your function on the following polynomial: $p(x) = 7x^3 - 11x^2 + 12x + 5$ evaluated at $x = 100$.

(d) Consider the polynomial $p(x) = (x+1)^8$. What is the most efficient way to compute $p(x)$? Does this contradict the optimality of Horner's scheme? Explain.

**Solution:**

(a) We evaluate the polynomial as follows

$$\begin{aligned} 3x^5 + 6x^8 - 2x^{11} + 9x^{14} &= x^5(3 + 6x^3 - 2x^6 + x^9) \\ &= x^5(3 + 6x^3 - x^6(2 - x^3)) \\ &= x^5(3 + x^3(6 - x^3(2 + x^3))) \end{aligned}$$

We need to compute $x^2$, $x^3$ and $x^5$. This costs three multiplications. Next, we evaluate $a = 2 - x^3$ using 1 addition. The next computation is $b = 6 - ax^3$ which requires one addition and one multiplication. After computing $b$, we compute $c = 3 + bx^3$ which requires one addition and one multiplication. Finally, we multiply $c$ by $x^5$ which requires 1 multiplication. In total, evaluating $p(x)$ requires three additions and six multiplications.

(b)

$$\begin{aligned} p(x_0) = \sum_{j=0}^{k} a_j x_0^j &= \sum_{j=0}^{k-2} a_j x_0^j + a_{k-1} x_0^{k-1} + c_k x_0^k = \sum_{j=0}^{k-2} a_j x_0^j + x_0^{k-1}(a^{k-1} + c_k x_0) \\ &= \sum_{j=0}^{k-2} a_j x_0^j + x_0^{k-1} c_{k-1} \\ &= \sum_{j=0}^{k-3} a_j x_0^j + a_{k-2} x_0^{k-2} + x_0^{k-1} c_{k-1} = \sum_{j=0}^{k-3} a_j x_0^j + x_0^{k-2}(a_{k-2} + x_0 c_{k-1}) \\ &= \sum_{j=0}^{k-3} a_j x_0^j + x_0^{k-2} c_{k-2} \\ &= \vdots \\ &= a_0 + c_1 x_0 = p(x_0) \end{aligned}$$

We need to evaluate $c_i$ for $i < k$. Each term requires one addition and one multiplication. therefore, the total number of operations is $2k$.

(c) You can find an implementation of Horner's method in the `HW1_Soln` folder on Canvas. A copy of the code is also included at the end of this document.

(d) The most efficient way is to first compute $x+1$ first and then compute $(x+1)^2$, $(x+1)^4$ and $(x+1)^8$. This requires 1 addition and 3 multiplication. This does not contradict optimality of Horner as its optimality is for a general polynomial. In this case, we do have a special structure that can be used for efficient computation.

**6.** For each of the following problems, state whether or not the computation is backward stable. Assume that the computation is done with floating point addition $\oplus$, floating point subtraction $\ominus$ and floating point multiplication $\otimes$. Justify your answer. [*Hint*: Use the definition of backward stability and the fundamental axiom of floating point arithmetic].

(a) Subtraction of two numbers: $\tilde{f}(x, y) = \mathrm{fl}(x) \ominus \mathrm{fl}(y)$.

(b) Outer product of two vectors: $\tilde{f}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^T$.

**Solution:**

(a) Using properties of floating point arithmetic, $\tilde{f}(x, y)$ can be written as follows:

$$\begin{aligned}\tilde{f}(x, y) &= \mathrm{fl}(x) \ominus fl(y) \\ &= x(1+\epsilon_1) \ominus y(1+\epsilon_2) \\ &= [x(1+\epsilon_1) - y(1+\epsilon_2)](1+\epsilon_3) \\ &= x(1+\epsilon_1)(1+\epsilon_3) - y(1+\epsilon_2)(1+\epsilon_3).\end{aligned}$$

where $|\epsilon_1| \leq |\epsilon_{\text{mach}}, |\epsilon_2| \leq \epsilon_{\text{mach}}$ and $|\epsilon_3| \leq |\epsilon_{\text{mach}}$. To determine backward stability of this operation, define $\tilde{x} = x(1+\epsilon_1)(1+\epsilon_3)$ and $\tilde{y} = y(1+\epsilon_2)(1+\epsilon_3)$. Then, to check the backward stability, we need to bound the following expression:

$$\begin{aligned}\frac{|\tilde{x} - x|}{|x|} &= \frac{|x + x\epsilon_3 + x\epsilon_1 + x\epsilon_3\epsilon_1 - x|}{|x|} \\ &= \frac{|x\epsilon_3 + x\epsilon_1 + x\epsilon_3\epsilon_1|}{|x|} \\ &\leq |\epsilon_3| + |\epsilon_1| + |\epsilon_3\epsilon_1| \\ &= O(\epsilon_{\text{mach}})\end{aligned}$$

We can do the same verification with the $\tilde{y}$ and $y$. It follows that the algorithm is backward stable.

(b) The result of the outer product is a matrix. Any entry of this outer product can be written as $(\mathbf{x}\mathbf{y}^T)_{ij} = \mathbf{x}_i\mathbf{y}_j$. First, we will show that the entry wise product is backward stable. Following a similar procedure as in $1a$, we see that

$$\mathbf{x}_i\mathbf{y}_j = \mathbf{x}_i(1+\epsilon_1) \otimes \mathbf{y}_j(1+\epsilon_2) = \mathbf{x}_i(1+\epsilon_1)\mathbf{y}_j(1+\epsilon_2)(1+\epsilon_3)$$

with where $|\epsilon_1| \leq |\epsilon_{\text{mach}}$ and $|\epsilon_2| \leq \epsilon_{\text{mach}}$. Define $\tilde{\mathbf{x}}_i = \mathbf{x}_i(1+\epsilon_1)$ and $\tilde{\mathbf{y}}_i = \mathbf{y}_j(1+\epsilon_2)(1+\epsilon_3)$. Then, to check the backward stability, we need to bound the following expressions:

$$\frac{|\tilde{\mathbf{x}}_i - \mathbf{x}_i|}{|\mathbf{x}_i|} \leq |\epsilon_1| \leq \epsilon_{\text{mach}} \frac{|\tilde{\mathbf{y}}_i - \mathbf{y}_i|}{|\mathbf{y}_i|} \leq |\epsilon_1| \leq |\epsilon_2| + |\epsilon_3| + |\epsilon_2\epsilon_3| \leq 3\epsilon_{\text{mach}} = O(\epsilon_{\text{mach}})$$

It follows that the computation of entries for outer product is backwards stable. However, note that there is no $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ such that $\tilde{f}(x, y) = xy^T = f(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y}^T$. We can see this by

considering the following scenario. Consider $\mathbf{x}_1\mathbf{y}_1$, $\mathbf{x}_1\mathbf{y}_2$ and $\mathbf{x}_2\mathbf{y}_1$ which are three different entries of the matrix in consideration. The key observation is that the floating point representation will have different errors. In particular, we have the following

$$\begin{aligned}\mathbf{x}_1\mathbf{y}_1 &= \mathbf{x}_1(1+\epsilon_1)\mathbf{y}_1(1+\epsilon_2)(1+\epsilon_3)\\ \mathbf{x}_1\mathbf{y}_2 &= \mathbf{x}_1(1+\epsilon_1)\mathbf{y}_2(1+\epsilon_2)(1+\epsilon_4)\\ \mathbf{x}_2\mathbf{y}_1 &= \mathbf{x}_2(1+\epsilon_5)\mathbf{y}_1(1+\epsilon_2)(1+\epsilon_6)\end{aligned}$$

where all the floating point rounding errors are within machine epsilon. For example, we can assign $\tilde{\mathbf{x}}_1 = \mathbf{x}_1(1+\epsilon_1)$ and $\tilde{\mathbf{x}}_2 = \mathbf{x}_1(1+\epsilon_5)$. However, it is not clear how to define $\tilde{\mathbf{y}}_1$ as it can be defined in two different ways. This shows that the outer product is not backward stable.

**7. Extra Credit**: During the Gulf War, an American Patriot Missile battery in Saudi Arabia, failed to intercept an incoming Iraqi Scud missile. The report of the software problem that led to the system failure can be found here `https://www.gao.gov/assets/imtec-92-26.pdf`. The main problem was attributed to an internal clock which recorded time in tenths of a second which is then stored as an integer i.e. if the clock reads 10, it means $10 \times (1/10\text{sec}) = 1\text{sec}$ has elapsed. The calculations were done using a 24-bit arithmetic.

(a) The 24-bit binary approximation of 1/10 is 0.00011001100110011001100. Convert this binary number to a base 10 representation. Let $y$ denote the result. What is $|y - \frac{1}{10}|$ ?

(b) What is the time error, in units of seconds, after 100 hours of operation?

(c) The Scud missile traveled at approximately 3750 miles per hour. Find the distance that a Scud missile would travel during the time error.

**Solution:**

(a) The standard representation is

$$2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21} = 0.099999904632568$$

(b) The time error is

$$(0.1 - 0.099999904632568) \cdot 100 \cdot 60 \cdot 60 \cdot 10 \approx 0.34$$

(c) The distance the Scud missile would travel during the time error is

$$(3750/(3600)) * 0.34 = 0.354\text{miles} \approx 0.566\text{kilometers}$$

This is a far enough distance the Scud missile traveled which led to the failure.

## Code

```
% ------------------------------------------------------------------------
% This code implements Horner's method for evaluating a polynomial of
% degree n: p(x) = a_0 + a_1*x + a_2*x^2 + ... + a_n*x^n
% Inputs
% x0
% array of coefficients: a = [a0 a1 a2 ...an]
% degree of polynomial: n
% Output
% p(x0)
% ------------------------------------------------------------------------
function y = horner_scheme(n,x0,a)
y = a(n+1);
for i = n:-1:1
    y =   y*x0+a(i);
end
end
```