ASSIGNMENT 1

MATH 387

QUESTION 1

Part (a): Take $x > 0$ to be a large number, and consider the truncated Taylor series

$$e^{-x} = 1 - x + \frac{x^2}{2!} - ... + \frac{x^n}{n!}$$

where $|\frac{x^n}{n!}| < \varepsilon$ for some error bound $\varepsilon > 0$. One can make the reasonable assumption that $n > x$. Thus, take $k \in [0, n]$ such that $k - x$ is minimized. When x is an integer, we can guarantee that $k = x$. Otherwise, we can choose $k : |x - k| < {}^1\!/_2$. Thus, consider the adjacent terms at k:

Even for x as small as 10, it takes 25 terms to yield a precision of 1.

$$\star \quad \left| \frac{x^{k-1}}{k!} - \frac{x^k}{k!} \right| = \frac{x^k}{k!} \left| \frac{k}{x} - 1 \right|$$

We have that $\frac{k}{x} \approxeq 1$, so there is a catastrophic cancellation of digits here.

As a concrete example, take $x := 40.23$ and $\varepsilon = .01$. The following script will return n such that the last term in our truncated series is $< \varepsilon$:

```
def errorindex(x, e):
  n=0
  error = x
  while error > e:
    n += 1
    error = (x**n)/factorial(n)
  return n
>>>errorindex(40.23, .01)
111
```

In this example, cancellation occurs at $k = 40$, which minimizes $|k - x|$: one calculates $\frac{k}{n} = \frac{40}{40.23} \approx 1.005 \approx 1$. This will yield cancellation in the $\star$ term.

To visualize where exactly things blow up, consider the following script:

```
def func(x, e):
  series = 0
  for i in range(errorindex(x, e)):
    series = series + ((-1**n)*(x**i))/factorial(i)
  return series

  e = 0.01
  plot(x, func(x, e))
```

When we consider the first 30 or-so elements, our estimator looks well-behaved:

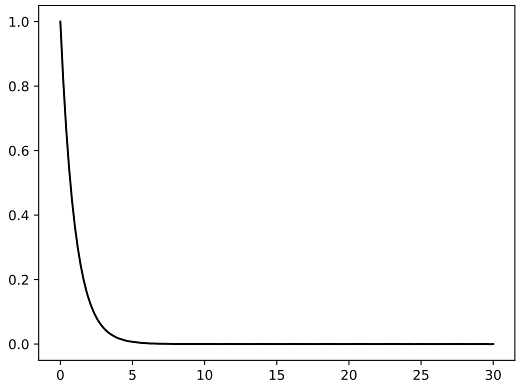
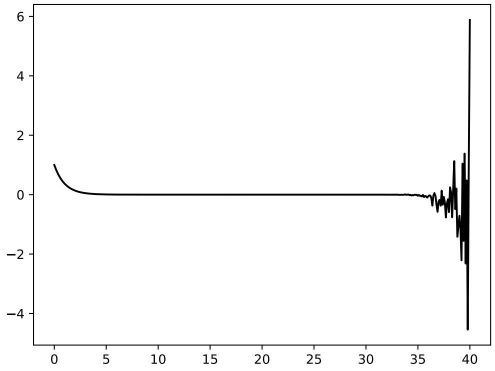
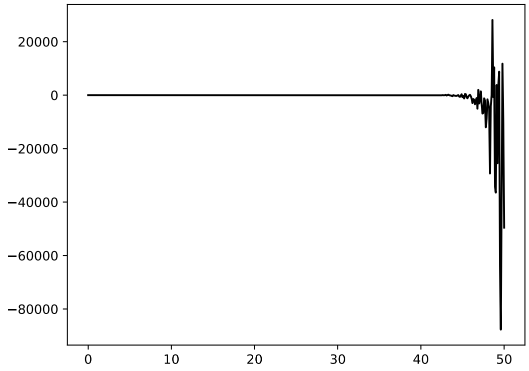
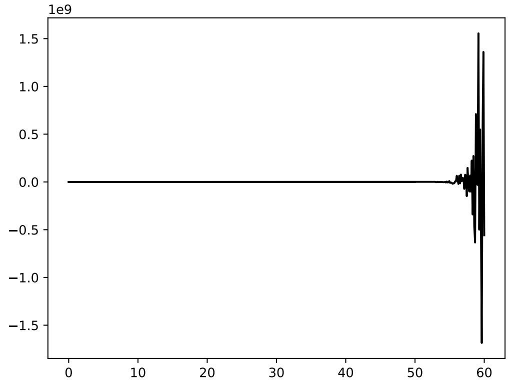


Figure 1: x ∈[0, 30]

But this quickly devolves as we consider larger $x \leqslant 40, 50$ and 60.







Part (b): We know already that $\sum_{i=0}^{\infty} \frac{x^i}{i!}$ converges rapidly to e^x. It makes sense, then, to invert a truncated Taylor series for e^x to calculate e^{-x}. This series only sums positive values, so we can be sure that we won't run into cancellation of digits. However, large inputs (i.e. 40) are unreasonable to calculate directly this way, as overflow becomes an issue. We can remedy this with recursion.

For some large input x, the following code will shrink our inputs according to

$$e^{-x} = (e^{\frac{-x}{2^n}})^{2^n}$$

where `lagrangeIndex(x)` calculates how many terms i are necessary to reach a certain error, and `euler(x)` calculates a truncated series for e^x with i terms:

```
def recursiveEstimator(x):
  if x <= 30:
    index = lagrangeIndex(x)
    return 1/eulerSeries(x, index)

  n = 0
  while x > 30:
    x = x/2
    n += 1
  index = lagrangeIndex(x)

  return (1/eulerSeries(x, index))**(2**n)
```

Here, our function shrinks x by half recursively, stopping when $x \leqslant 30$, and then passes the halving index and our new x to both `lagrangeIndex` and `eulerSeries`, as described above. The following is a simple function to calculate the e^x series:

```
def eulerSeries(x, index):
  series = 0
  for i in range(index):
    series = series + (x**i)/math.factorial(i)
  return sum
```

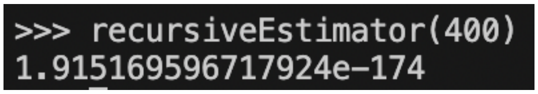
We still require an appropriate index to sum over. For e^x, a rough but sufficient bound is $\frac{e^x x^{n+1}}{n!}$, by Lagrange. Suppose, for any x, we want an error of $e^x/100$. This will yield a relative error of $\frac{|e^x - e^{\tilde{x}}|}{|e^x|} < 0.01$, which is desirable:

In the computation of a summation index, we can rough our calculation of e^x to $1 + x + \frac{x^2}{2}$. Lagrange bounds are quite rough themselves, so this should still be sufficient.

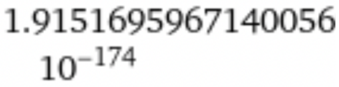
```
  def lagrangeIndex(x):
    bound = (1+x+(x**2)/2)/100
    currentError = (1+x+(x**2)/2)
    #This way, we always enter the while loop.
```

```
      n = 0
    while currentError > bound:
      n += 1
      currentError = ((1+x+(x**2)/2)*(x**(n+1)))/math.factorial(n+1)
    return n
```

As a concrete example, compare the first 17 digits of `recursiveEstimator(400)` to Wolfram's for e^{-x}.



```
>>> recursiveEstimator(400)
1.915169596717924e-174
```

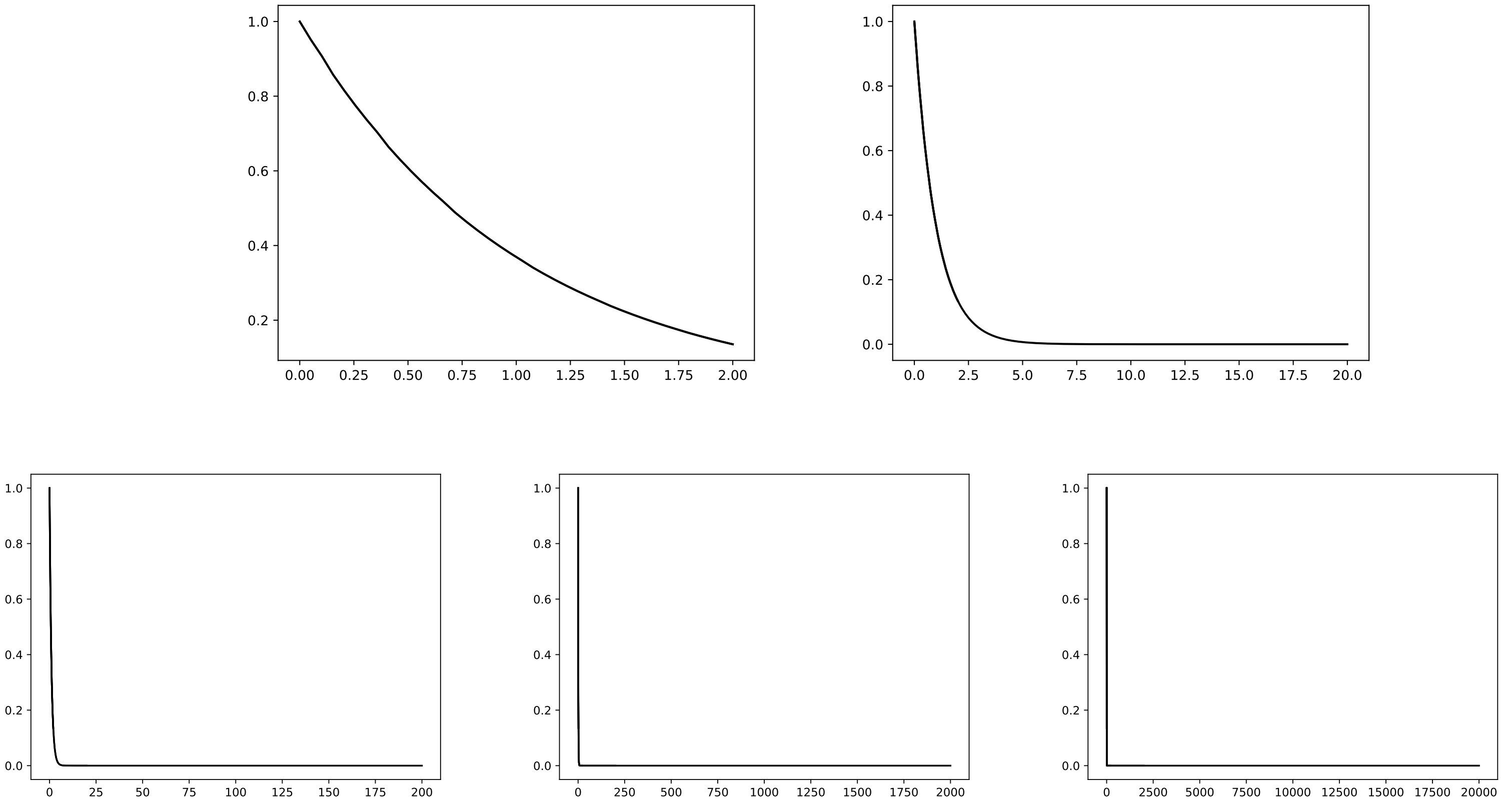


1.9151695967140056
10^{-174}

We get a relative error of $\frac{x-\tilde{x}}{x} = \frac{1.915...-\widetilde{1.915...}}{1.915...} = 5.772 \times 10^{-12}$.

If Wolfram is "God" for the purposes of true x.

So: this algorithm is OK! Here is the graph for a domain of $2 * 10^i, i \in [0, 4]$, with an absolute resolution of $\Delta x = 0.2$:



QUESTION 2

The following algorithm will compute $\sqrt[3]{x}$ digit-by-digit, and requires knowledge of some elementary operations ($+, -, \times, \div$, and powers) and inequalities.

1. We want to find a number a such that $a^3 = x$. For the first digit of a, find the largest number of the form $a_1 = r_1 \times 10^n$, where $r_1 \in [1, 9]$, such that $a_1^3 \leqslant x$. n may be positive or negative. We have that r_1 is our first digit. Set $A = a_1$.
2. We conclude that $(A + a_2)^3 = x$ for some unknown a_2. Expanding, we get

   $$A^3 + 3Aa_2^2 + 3A^2a_2 + a_2^3 = x$$

   Subtracting off the A^3 term, we set $x_1 = x - A^3$:

   $$3Aa_2^2 + 3A^2a_2 + a_2^3 = x_1$$

3. Now, try to find the largest a_2 of the form $r_2 \times 10^{n-1}$ such that $3Aa_2^2 + 3A^2a_2 + a_2^3 \leqslant x_1$. If $n \leqslant -1$, it may be useful to only plug in for the $3A^2a_2$ term, and consider the other terms if you get close.
4. r_2 is our next digit, and update A to be $A + a_2$.
5. We have $(A + a_3)^3 = x$ for some unknown a_3. Expanding:

   $$A^3 + 3Aa_3^2 + 3A^2a_3 + a_3^3 = x$$

   Subtracting off the A^3 term, we set $x_2 = x - A^3$:

   $$3Aa_3^2 + 3A^2a_3 + a_3^3 = x_2$$

6. Find the largest a_3 of the form $r_3 \times 10^{n-2}$ such that $3Aa_3^2 + 3A^2a_3 + a_3^3 \leqslant x_2$. Our third digit is r_3. To prepare for the next round, update $A = A + a_3$.
7. This process can be done indeterminately, but if one finds a_i such that $3Aa_i^2 + 3A^3a_i + a_i^3 = x_{i-1}$, then we have found an exact answer.

Why does this work? If $a^3 = x$, we know the solution will be of the form

$$\star \quad (r_1 \times 10^n + ... + r_{n+1} + r_{n+2} \times 10^{-1} + ...)^3 = x$$

One finds $a_1 = r_1 \times 10^n$ the same way one does in division: if $(r_1 \times 10^n)^3$ is larger than our output, then clearly $\star$ doesn't hold. If we find a sufficiently small r_1, but $r_1 + 1$ is sufficient too, then we will never make up the difference in the $10^{n-1}, 10^{n-2}$, etc. terms. Thus, r_1 must be the *largest* digit such that $(r_1 \times 10^n)^3 \leqslant x$. We have a remainder to deal with, so one uses $\star$ to write $(r_1 \times 10^n + a_2)^3 = x$, and we continue the process.

To keep using our "largest that is $\leqslant x$" qualifier, a_2 must be one order of magnitude less than than a_1, but one order greater than any future terms. Thus, one is instructed to find the largest a_2 of the form $r_2 \times 10^{n-1}$, where n is the order of magnitude of the first term.

Lastly, note that $r_1 \times 10^n + ... + r_{n+1} + r_{n+2} \times 10^{-1} + ...$ in "digit" form is just

$$r_1 ... r_{n+1} (.) r_{n+2} ... \qquad \text{as desired}$$

As an example, we'll calculate the cube root of 6,028,568.

1. The largest number such that $(r_1 \times 10^n)^3 \leqslant 6,028,568$ is 1×10^2, so $r_1 = 1$. Note that $(2 \times 10^2)^3$ is 8,000,000.

2. Thus, $(1 \times 10^2 + a_2)^3 = 6,028,568$. We can expand to find

$$10^6 + 3a_2^2 \times 10^2 + 3a_2 \times 10^4 + a_2^3 = 6,028,568$$

$$300a_2^2 + 30,000a_2 + a_2^3 = 5,028,568$$

   We are now looking for numbers of one magnitude less than 10^2, so 10, 20, 30, etc.

   (a) $r_2 = 5$, we get $300(50^2) + 30,000(50) + (50^3) = 2,375,000$, which is too small.

   (b) Trying $r_2 = 9$, we get $300(90^2) + 30,000(90) + (90^3) = 5,859,000$, which is *a hair* too big.

   (c) $r_2 = 8$ should work: $300(80^2) + 30,000(80) + (80^3) = 4,832,000$, so this is the right answer.

3. Now let $A = 1 \times 10^2 + 8 \times 10^1$. We are now looking for a single digit number $a_3 = r_3$ such that $(A + a_3)^3 = x$. We expand the cubic as before:

$$(180^3) + 3(180)a_3^2 + 3(180^2)a_3 + a_3^3 = 6,028,568$$

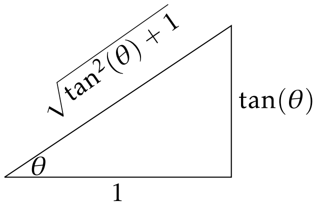
$$540a_3^2 + 97,200a_3 + a_3^3 = 196,568$$

   One can see that 2 would be a pretty close guess, just by the 97,200 term. We check to find 540(4) + 97,200(2) + (8) = 196,568, which is exactly our answer. Thus, $r_3 = 2$.

$\implies \sqrt[3]{6,028,568} = r_1 r_2 r_3 = 182$

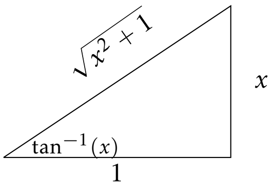
QUESTION 3

Part (a): Consider the following diagram, for a known angle θ



One sees then that $\cos(\theta) = \frac{1}{\sqrt{\tan^2(\theta)+1}}$ and $\sin(\theta) = \frac{\tan(\theta)}{\sqrt{\tan^2(\theta)+1}}$

We can draw a similar diagram for $\arctan(x)$, where x is unknown:



And conclude that

$$\tan^{-1}(x) = \cos^{-1}\left(\frac{1}{\sqrt{x^2+1}}\right) = \sin^{-1}\left(\frac{x}{\sqrt{x^2+1}}\right)$$

Rearranging, one yields

$$\cos^{-1}(x) = \tan^{-1}\left(\frac{\sqrt{1-x^2}}{x}\right) \qquad \sin^{-1}(x) = \tan^{-1}\left(\frac{x}{\sqrt{1-x^2}}\right)$$

Lastly, we have that $e^{a\ln(x)} = \left(e^{\ln(x)}\right)^a = x^a$ for any $a \in \mathbb{R}$, and thus we've found "elementary" expressions for $\cos, \sin, \cos^{-1}, \sin^{-1}$, and x^a.

Part (b): We will reduce the argument of $\tan(x)$ to $x \in [0, \frac{\pi}{4}]$, or roughly 0.79.

$\frac{\pi}{4} < x \leqslant \frac{\pi}{2}$ One has $\tan(x) = \frac{\sin(x)}{\cos(x)} \overset{\star}{=} \frac{-\cos(x-\frac{\pi}{2})}{\sin(x-\frac{\pi}{2})} = \frac{\cos(\frac{\pi}{2}-x)}{\sin(\frac{\pi}{2}-x)} = [\tan(\frac{\pi}{2}-x)]^{-1}$, which is in our domain $y \in [0, \frac{\pi}{4}]$.

$\frac{\pi}{2} < x \leqslant \pi$ Write $\tan(x) = -[\tan(x-\frac{\pi}{2})]^{-1}$, from $\star$. These inputs span $y \in (0, \frac{\pi}{2}]$, but the case $y \in (\frac{\pi}{4}, \frac{\pi}{2}]$ is taken care of by the point above.

$\pi < x < 2\pi$ Note that $\tan(x) = \tan(x-\pi)$, which has inputs in $y \in (0, \pi)$, but the case $y \in (\frac{\pi}{4}, \pi)$ has been taken care of by above.

Taking repeated compositions of the equations above, one derives the following (awful) table.

$$\tan(x) = \begin{cases} \tan(x) & x \in [0, \frac{\pi}{4}] \\ [\tan(\frac{\pi}{2} - x)]^{-1} & x \in (\frac{\pi}{4}, \frac{\pi}{2}] \\ -[\tan(x - \frac{\pi}{2})]^{-1} & x \in (\frac{\pi}{2}, \frac{3\pi}{4}] \\ -\tan(\pi - x) & x \in (\frac{3\pi}{4}, \pi] \\ \tan(x - \pi) & x \in (\pi, \frac{5\pi}{4}] \\ [\tan(\frac{3\pi}{2} - x)]^{-1} & x \in (\frac{5\pi}{4}, \frac{3\pi}{2}] \\ -[\tan(x - \frac{3\pi}{2})]^{-1} & x \in (\frac{3\pi}{2}, \frac{7\pi}{4}] \\ -\tan(2\pi - x) & x \in (\frac{7\pi}{4}, 2\pi) \end{cases}$$

QUESTION 4

The following facts will be useful:

$$|x_1 \otimes ... \otimes x_n - x_1 ... x_n| \leqslant 2n\varepsilon |x_1 ... x_n| \qquad \text{(naive algorithm; machine epsilon)}$$

$$|\tilde{x_1} ... \tilde{x_n} - x_1 ... x_n| \leqslant \frac{n\varepsilon}{1 - n\varepsilon} |x_1 ... x_n| \qquad \text{(input epsilon)}$$

$$\frac{|x_1 \oplus ... \oplus x_n - x_1 - ... - x_n|}{|x_1 + ... + x_n|} \leqslant \frac{|x_1| + ... + |x_n|}{|x_1 + ... + x_n|} 2n\varepsilon \qquad \text{(naive algorithm; machine epsilon)}$$

$$\frac{|\tilde{x_1} + ... + \tilde{x_n} - x_1 - ... - x_n|}{|x_1 + ... + x_n|} \leqslant \frac{|x_1| + ... + |x_n|}{|x_1 + ... + x_n|} \varepsilon \qquad \text{(input epsilon)}$$

Error in x

Suppose we want to compute $\ln(y)$. The Gregory series gives us

$$\ln\left(\frac{1+z}{1-z}\right) = 2\left(z + \frac{z^3}{3} + \frac{z^5}{5} + ...\right)$$

Thus, we solve $y = \frac{1+x}{1-x}$ to start, and find $x = \frac{y-1}{y+1}$. Suppose y is given with some small input error ε. For simplicity, we suppose that ±1 operation can be done exactly.

$$\left|\frac{y-1}{y+1} - (\tilde{y} - 1) \odiv (\tilde{y} + 1)\right| \leqslant \left|\frac{y-1}{y+1} - \frac{\tilde{y}-1}{\tilde{y}+1}\right| + \left|\frac{\tilde{y}-1}{\tilde{y}+1} - (\tilde{y} - 1) \odiv (\tilde{y} + 1)\right|$$

For the first term, one calculates an asymptotic condition number $\kappa = \frac{yf'(y)}{f(y)}$, where $f'(y) = \frac{2}{(y+1)^2} \implies \left|\frac{y-1}{y+1} - \frac{\tilde{y}-1}{\tilde{y}+1}\right| \leqslant \frac{2y}{y^2-1} \varepsilon \left|\frac{y-1}{y+1}\right|$

The second term just evaluates to $\leqslant \varepsilon \left|\frac{y-1}{y+1}\right|$, from axiom 1. Combining, one yields the relative error for x, the input to our series:

$$\varepsilon_x = \left(\frac{2y}{y^2 - 1} + 1\right) \varepsilon \approx \left(\frac{2}{y} + 1\right) \varepsilon$$

There's a cancellation of digits at $y = 1$ from the $(y-1)$ term in the numerator of x, which manifests in our expression for ε_x. We wish to move on with the analysis, though, so the assumption that ±1 is exact allows us to approximate ourselves out of this blow-up spot.

Error in series terms

Define the following:

$$z_n = b_1 + ... + b_n \qquad \tilde{z_n} = \tilde{b_1} + ... + \tilde{b_n} \qquad \hat{z_n} = \tilde{b_1} \oplus ... \oplus \tilde{b_n}$$

where $b_i := \frac{2x^i}{2i-1}$. Assume that integer division/multiplication can be done exactly, and that the error in $b_i \leqslant$ error in b_n term.

(As in class)

Let $\tilde{b_i} = \frac{2\overbrace{\tilde{x}\otimes\ldots\otimes\tilde{x}}^{i \text{ times}}}{2i-1}$ and $\overline{b_i} = \frac{2\tilde{x}^i}{2i-1}$.

$$\begin{aligned}|b_n - \tilde{b_n}| &\leqslant |b_n - \overline{b_n}| + |\overline{b_n} - \tilde{b_n}| \\ &= \left|\frac{2x^n}{2n-1} - \frac{2\tilde{x}^n}{2n-1}\right| + \left|\frac{2\tilde{x}^n}{2n-1} - \frac{2\tilde{x}\otimes\ldots\otimes\tilde{x}}{2n-1}\right| \\ &\leqslant \frac{n\varepsilon_x}{1-n\varepsilon_x}\left|\frac{2x^n}{2n-1}\right| + 2n\varepsilon\left|\frac{2\tilde{x}^n}{2n-1}\right|\end{aligned}$$

We know that $\frac{-1}{2} \leqslant x \leqslant \frac{1}{2}$, so our initial input y is bounded below by $\frac{1}{3}$. Lastly, assume $n\varepsilon \leqslant \frac{1}{2}$, as usual. Then we have:

$$n\varepsilon_x = \left(\frac{2}{y} + 1\right) n\varepsilon \leqslant 7n\varepsilon \leqslant \frac{7}{2}$$

Thus, we can take $n\varepsilon_x \leqslant \frac{7}{2}$. Recall our expression for $|b_n - \tilde{b_n}|$. Subbing in our bound on $n\varepsilon_x$, we get

A note on asymptotic approximation: $\varepsilon\tilde{x} = x(1+\varepsilon_x)\varepsilon$, which will produce terms on the order ε^2, which we throw away.

$$|b_n - \tilde{b_n}| \leqslant -\frac{2}{5}n\varepsilon_x\left|\frac{2x^n}{2n-1}\right| + 2n\varepsilon\left|\frac{2\tilde{x}^n}{2n-1}\right| \overset{\text{Asympt.}}{\leqslant} n\left(-\frac{2}{5}\varepsilon_x + 2\varepsilon\right)\left|\frac{2x^n}{2n-1}\right| = n\left(-\frac{2}{5}\varepsilon_x + 2\varepsilon\right)|b_n|$$

Error in series

Recall our definitions for z_n, $\tilde{z_n}$, and $\hat{z_n}$, and let $\varepsilon_b := -\frac{2}{5}\varepsilon_x + 2\varepsilon$

$$\begin{aligned}|z_n - \hat{z_n}| &\leqslant |z_n - \tilde{z_n}| + |\tilde{z_n} - \hat{z_n}| \\ &\leqslant \overbrace{\frac{|b_1| + \ldots + |b_n|}{|b_1 + \ldots + b_n|}\varepsilon_b|z_n|}^{\text{error propagation}} + \overbrace{\frac{|b_1| + \ldots + |b_n|}{|b_1 + \ldots + b_n|}2n\varepsilon|\tilde{z_n}|}^{\text{naive summation}} \\ &= n\left(\frac{-2}{5}\varepsilon_x + 2\varepsilon\right)|z_n| + 2n\varepsilon|\tilde{z_n}| \\ &\leqslant n\left(\frac{-2}{5}\varepsilon_x + 2\varepsilon\right)|z_n| + 2n\varepsilon|z_n|(1+\varepsilon'_b) \quad \text{where } |\varepsilon'_b| \leqslant \varepsilon_b \\ &= n\left(\frac{-2}{5}\varepsilon_x + 2\varepsilon\right)|z_n| + |z_n|(2n\varepsilon + \overset{\text{LOT} \approx \varepsilon^2}{\cancel{2n\varepsilon\varepsilon'_b}}) \\ &= n\left(\frac{-2}{5}\varepsilon_x + 4\varepsilon\right)|z_n| = n\varepsilon\left(-\frac{2}{5}\frac{2}{y} - \frac{2}{5} + 4\right)|z_n| \leqslant n\varepsilon\left(9 - \frac{2}{y}\right)|z_n|\end{aligned}$$

If $y = \frac{1+x}{1-x}$ is bounded by $x \in [\frac{-1}{2}, \frac{1}{2}]$, then $y \in [\frac{1}{3}, 3]$, and we conclude

$$|z_n - \hat{z_n}| \leqslant \frac{25}{3}n\varepsilon|z_n|$$

Truncation Error

Let $\ln(y)$ be the exact result. $|\ln(y) - \hat{z_n}| \leqslant |\ln(y) - z_n| + |z_n - \hat{z_n}| \leqslant R_n + \frac{25}{5} n\varepsilon |z_n|$, where R_n is the Lagrange error bound on $2x + \frac{2x^3}{3} + ...$

We can bound R_n by the Lagrange error for $2(x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} ...) = -2\ln(1-x)$, to simplify our derivations. The following are derivatives of this function:

0	$-2\ln(1-x)$
1	$\frac{2 \cdot 0!}{1-x}$
2	$\frac{2 \cdot 1!}{(1-x)^2}$
3	$\frac{2 \cdot 2!}{(1-x)^3}$
$\vdots$	$\vdots$
n	$\frac{2(n-1)!}{(1-x)^n}$
$n+1$	$\frac{2n!}{(1-x)^{n+1}}$

The maximum value for the $n+1^{\text{th}}$ derivative on $x \in [-\frac{1}{2}, \frac{1}{2}]$ occurs at $-\frac{1}{2}$, where $f^{(n+1)}(\frac{-1}{2}) = \frac{2n!}{(\frac{3}{2})^{n+1}}$. Thus:

$$R_n \leqslant \frac{\frac{2n!}{(\frac{3}{2})^{n+1}} {\frac{1}{2}}^{n+1}}{(n+1)!} = \frac{2}{n+1} {\frac{1}{3}}^{n+1}$$

Bounding $|z_n|$ by its value at $\frac{1}{2}$, we get

$$|\ln(y) - \hat{z_n}| \leqslant R_n + \frac{25}{3} n\varepsilon |z_n| \leqslant \frac{2}{n+1} {\frac{1}{3}}^{n+1} + \frac{25}{3} n\varepsilon |z_n|$$

$$= \frac{2}{n+1} {\frac{1}{3}}^{n+1} + \frac{25}{3} n\varepsilon \cdot 2 \sum_{i=1}^{n} \frac{1}{(2i-1)} {\frac{1}{2}}^{2i-1}$$

Argument Reduction

We'd like to reduce x to the interval $[\frac{-1}{2}, \frac{1}{2}]$, which means $y = \frac{1+x}{1-x}$ should be reduced to the interval $[\frac{1}{3}, 3]$. One can perform repeated pseudo divisions to find $k_0, ..., k_n$ such that

$$y = 2^{k_0} \cdot (1.1)^{k_1} \cdot ... \cdot (1 + 10^{-n})^{k_n}$$

Since $y_i := (1 + 10^{-i}) \in [\frac{1}{3}, 3]$, $\ln(y_i)$ can be computed as above, and we have

$$\ln(y) = \sum_{i=0}^{n} k_i \ln(y_i)$$

Question 5

Part (a): Consider the summation algorithm

$$\begin{aligned} y_n &= x_n - c_{n-1} \\ \tilde{s_n} &= \tilde{s}_{n-1} + y_n \qquad\qquad c_0 = \tilde{s_0} = 0 \\ c_n &= (\tilde{s}_n - \tilde{s}_{n-1}) - y_n \end{aligned}$$

Instead of computing `sum = sum + x_i` repeatedly (naive), we will add on a *corrected* x_i, denoted y_i, i.e. `sum = sum + y_i` . This corrected number will somehow store information about round-off error that occurred previously, and rectify it.

Consider the operation `sum = sum + y_i` . One usually expects a running sum to be larger than any particular y_i. With finite float-point representation, lower-order digits will thus be lost (e.g. 2010.1 + 2.1383 = 2012.2 will lose $\varepsilon = .0383$, where we can store 5 digits of precision). One would like to add back ε in some meaningful way.

c_i, this correction term, will first find the difference between the old and new summation, which we expect to have low precision (e.g. 2012.2 - 2010.1 = 2.1). Thus, $(\tilde{s_n} - \tilde{s}_{n-1}) - y_n$ can store the lower order digits we missed (e.g. 2.1000 - 2.1383 = -.0383 = ε).

This algorithm manages to capture and store round-off error in the c_i terms. $\tilde{s_n}$ had lost these digits, so the first step, which we previously glossed over, will add back $|c_i|$ (c_i is negative with round-down error) to the next sum index.

Even though $\tilde{s_n} + y_n$ will always lose precision, corrected or otherwise, the act of “carrying along” lost digits ensures that the sum will never stray too far.

So: in every way, this is better than the naive summation, which will accumulate error and do nothing with it!

Part (b): With each step in the algorithm being inexact, we have

$$\begin{aligned} y_n &= (x_n - c_{n-1})(1+\alpha_i) \\ \tilde{s_n} &= (\tilde{s}_{n-1} + y_n)(1+\delta_i) \qquad |\delta_i|, |\alpha_i|, |\gamma_i|, |\gamma_j| \leqslant \varepsilon \\ c_n &= [(\tilde{s}_n - \tilde{s}_{n-1})(1+\gamma_i) - y_n](1+\gamma_j) \end{aligned}$$

We want to express $\tilde{s_n}$ as a sum of $|x_n|$, where x_i have some attached error $(1+\delta_1)...(1+\delta_k)$.

$$\begin{aligned} \tilde{s_1} &= (\tilde{s_0} + y_1)(1+\delta_1) = y_1(1+\delta_1) \\ &= (x_1 - c_0)(1+\delta_2)(1+\delta_1) \leqslant x_1(1 + 2\varepsilon + O(\varepsilon^2)) \end{aligned}$$

With each subsequent $\tilde{s_i}$, one will introduce additional error into the x_1 term. Thus, it will be sufficient to find the error associated with the x_1, and bound all x_i by this. Consider:

I could not get the following to work in any reasonable way, so everything below is extraneous to the answer provided thus far.

$$\begin{aligned} \tilde{s_n} &= y_1(1+\delta_n)...(1+\delta_1) + ... + y_n(1+\delta_1) \qquad \text{as in naive algorithm} \\ &= x_1(1+\delta_n)...(1+\delta_1)(1+\alpha_1) + ... + x_n(1+\delta_1)(1+\alpha_n) \\ &- [c_1(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) + ... + c_{n-1}(1+\delta_1)(1+\alpha_n)] \qquad \text{remove } i = 2, 3, ... \text{ terms} \\ &= x_1(1+\delta_n)...(1+\delta_1)(1+\alpha_1) - c_1(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &= x_1(1+\delta_n)...(1+\delta_1)(1+\alpha_1) - c_1(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &= x_1(1+\delta_n)...(1+\delta_1)(1+\alpha_1) - [(\tilde{s}_1 - \cancel{\tilde{s}_0})(1+\gamma_1) - y_1](1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &= x_1(1+\delta_n)...(1+\delta_1)(1+\alpha_1) - [\tilde{s}_1(1+\gamma_1) - (x_1 - \cancel{c_0})(1+\alpha_3)](1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &= x_1[(1+\delta_n)...(1+\delta_1)(1+\alpha_1) + (1+\alpha_3)(1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2)] \\ &- \tilde{s_1}(1+\gamma_1)(1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &= x_1[(1+\delta_n)...(1+\delta_1)(1+\alpha_1) + (1+\alpha_3)(1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2) \\ &- \underbrace{(1+\delta_N)(1+\alpha_4)}_{\text{from } \tilde{s_1} \to y_1 \to x_1 \text{ estimation, } |\delta_N| \leqslant \varepsilon} (1+\gamma_1)(1+\gamma_2)(1-\delta_{n-1})...(1+\delta_1)(1+\alpha_2)] \\ &\leqslant x_1[\text{degree 1 term} + (n+2-1)\varepsilon \text{ terms} + \left[\binom{n+1}{2} - 1\right] \varepsilon^2 \text{terms}] \\ &\leqslant x_1(1 + C\varepsilon + O(\varepsilon^2)) \end{aligned}$$