Exercise 2: Absorption and Recursive Summation with Three Exam Variations

AFAE - Master 2 CCA

1 Original Exercise

Exercise 1 (Absorption (4 points))**.** Let M be a floating-point number large enough such that $\text{fl}(M + x) = M$. What are the possible values of $\text{fl}\left(\sum_{i=1}^{n} x_i\right)$ where $|x_i| \ll M$, assuming the sum is evaluated using the classical recursive summation algorithm?

2 Solution to Original Exercise

2.1 Understanding the Problem

Key Concept: Absorption

Absorption occurs when adding a small number to a much larger number results in no change:

$$\text{fl}(M + x) = M \quad \text{when } |x| \ll M$$

This happens because x is smaller than the **unit in the last place (ulp)** of M.

2.2 Condition for Absorption

For $\text{fl}(M + x) = M$, we need:

$$|x| < \frac{\text{ulp}(M)}{2}$$

In floating-point arithmetic with precision p (e.g., $p = 53$ for double precision):

$$\text{ulp}(M) = 2^{\lfloor \log_2(M) \rfloor - p + 1}$$

For large M, if $M = 2^E \cdot m$ where $m \in [1, 2)$:

$$|x| < 2^{E-p}$$

2.3 Classical Recursive Summation Algorithm

Algorithm 1 Recursive Summation

```
1: S ← 0
2: for i = 1 to n do
3:     S ← fl(S + x_i)
4: end for
5: return S
```

2.4 Analysis of Possible Values

Critical Observation

The result depends on **when** the partial sum becomes large enough to cause absorption.

2.4.1 Case 1: Early Absorption (Worst Case)

If at some step $k < n$, the partial sum S_k satisfies:

$$|S_k| \geq M \quad \text{and} \quad |x_i| < \text{ulp}(S_k)/2 \text{ for all } i > k$$

Then:

$$\text{fl}(S_k + x_{k+1}) = S_k, \quad \text{fl}(S_k + x_{k+2}) = S_k, \ldots$$

Result: All remaining terms are absorbed!

$$\boxed{\text{fl}\left(\sum_{i=1}^{n} x_i\right) = S_k = \sum_{i=1}^{k} x_i \text{ (with rounding errors)}}$$

2.4.2 Case 2: No Early Absorption

If the partial sum never becomes large enough to cause absorption before all terms are added:

$$|S_i| < M \text{ for all } i < n$$

Then we get the usual result with accumulated rounding errors:

$$\boxed{\text{fl}\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i + \text{(rounding errors)}}$$

2.4.3 Case 3: Sign Cancellation

If terms have mixed signs and cancel each other: - Partial sum may grow large, then shrink - Some terms absorbed, then later terms added

Result: Depends on the specific sequence!

2.5 Comprehensive Answer

Possible Values

The possible values of $\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right)$ are:

1. **Complete Loss of Small Terms:**

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{k} x_i$$

   where $k < n$ is the first index where absorption begins.

2. **Partial Loss:** Various combinations where some terms are absorbed and others are not, depending on the order and signs of x_i.

3. **No Loss (if sum stays small):**

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i + O(\varepsilon_{\mathrm{mach}})$$

4. **Zero (complete cancellation):** If positive and negative terms cancel and the partial sum causes absorption at multiple stages, the result could be 0 or close to 0.

2.6 Key Factors Determining the Result

1. **Order of summation:** Different orders can give different results
2. **Signs of x_i:** All same sign vs. mixed signs
3. **Magnitudes:** How quickly partial sum grows to $|M|$
4. **Number of terms:** How many terms get absorbed

2.7 Concrete Example

Numerical Example

Let:

- Precision: $p = 3$ (for simplicity)
- $M = 2^{10} = 1024$
- $\text{ulp}(M) = 2^{10-3+1} = 2^8 = 256$
- Absorption occurs when $|x_i| < 128$

Suppose: $x_1 = 512, x_2 = 256, x_3 = 128, x_4 = 64, x_5 = 32$
Computation:

$$\begin{aligned}
S_0 &= 0 \\
S_1 &= 0 + 512 = 512 \\
S_2 &= 512 + 256 = 768 \\
S_3 &= 768 + 128 = 896 \\
S_4 &= \text{fl}(896 + 64) && \text{(if } 896 \approx 2^{10}, \text{ absorption may start)} \\
S_5 &= \text{fl}(S_4 + 32) && \text{(likely absorbed)}
\end{aligned}$$

If $S_3 = 896$ is rounded to 1024 in our simplified arithmetic, then:

$$\text{fl}\left(\sum_{i=1}^{5} x_i\right) = 1024 \neq 992 \text{ (true sum)}$$

Lost terms: x_4, x_5 (potentially)

3 Exam Variation 1: Specific Numerical Case

Exercise 2 (Double Precision Example)**.** Consider double precision floating-point arithmetic ($p =$ 53 bits).

(a) Let $M = 2^{60}$. What is $\text{ulp}(M)$?

(b) If we compute $\sum_{i=1}^{1000} x_i$ where each $x_i = 2^5 = 32$, using recursive summation starting from $S_0 = M$, what is the final result $\text{fl}(M + \sum_{i=1}^{1000} x_i)$?

(c) How many terms are lost to absorption?

(d) What should be the result if we computed $\sum_{i=1}^{1000} x_i$ first, then added M?

3.1 Solution to Variation 1

Solution 1. (a) Computing ulp(M):
For $M = 2^{60}$ in double precision ($p = 53$):

$$\text{ulp}(M) = 2^{60-53+1} = 2^8 = 256$$

For absorption: $|x| < \frac{\text{ulp}(M)}{2} = 128$
(b) Recursive summation $S_0 = M$:
Starting from $S_0 = M = 2^{60}$:

$$\begin{aligned} S_1 &= \text{fl}(M + x_1) = \text{fl}(2^{60} + 32) \\ &= 2^{60} \quad \text{(since } 32 < 128\text{, absorbed!)} \\ S_2 &= \text{fl}(S_1 + x_2) = \text{fl}(2^{60} + 32) = 2^{60} \\ &\vdots \\ S_{1000} &= 2^{60} \end{aligned}$$

Result:

$$\boxed{\text{fl}\left(M + \sum_{i=1}^{1000} x_i\right) = 2^{60}}$$

(c) Terms lost:
All 1000 terms are lost to absorption!
True sum: $2^{60} + 1000 \times 32 = 2^{60} + 32000$
Computed sum: 2^{60}
Error: 32000 (absolute)
(d) Correct order of operations:
If we compute $\sum_{i=1}^{1000} x_i$ first:

$$\sum_{i=1}^{1000} 32 = 32000$$

Then:

$$\text{fl}(M + 32000) = \text{fl}(2^{60} + 32000)$$

Since $32000 = 2^{15} - 2^{10} \approx 2^{15}$ and $2^{15} > \text{ulp}(M)/2 = 128$:

$$\text{32000 is NOT absorbed!}$$

Actually: $32000/256 = 125$, so it fits in the ulp.
More precisely:

$$\text{fl}(2^{60} + 32000) = 2^{60} + \text{fl}(32000) \approx 2^{60} + 32768$$

(Rounded to nearest multiple of $\text{ulp}(M) = 256$)

$$\boxed{\text{Result} \approx 2^{60} + 32768}$$

Lesson: Order matters! Sum small numbers first, then add to large numbers.

4 Exam Variation 2: Compensated Summation

Exercise 3 (Kahan Summation Algorithm)**.** Consider the classical recursive summation that suffers from absorption. The **Kahan compensated summation** algorithm attempts to reduce this error.

Algorithm 2 Kahan Summation

$S \leftarrow 0$
$c \leftarrow 0$ ▷ Compensation for lost low-order bits
for $i = 1$ to n **do**
  $y \leftarrow x_i - c$ ▷ Correct the term
  $t \leftarrow \mathrm{fl}(S + y)$ ▷ Add corrected term
  $c \leftarrow \mathrm{fl}((t - S) - y)$ ▷ Compute lost bits
  $S \leftarrow t$
end for
return S

(a) Explain in your own words what the compensation variable c represents.

(b) Consider summing $n = 4$ terms: $x_1 = 1.0, x_2 = 10^{-16}, x_3 = 10^{-16}, x_4 = -1.0$ in double precision ($\varepsilon_{\text{mach}} \approx 2.22 \times 10^{-16}$).

Compare the results of:

- Classical recursive summation
- Kahan summation

(c) Does Kahan summation completely solve the absorption problem when M is very large? Explain.

4.1 Solution to Variation 2

Solution 2. (a) What c represents:

The compensation variable c stores the **rounding error** from the previous addition that was lost due to absorption or limited precision.

When we compute $t = \mathrm{fl}(S + y)$:

- The true value is $S + y$
- But we get t with some rounding error
- The lost part is: $(S + y) - t$

By computing $c = (t - S) - y$, we capture this lost information and **subtract it from the next term** to compensate.

(b) Numerical comparison:

Classical Recursive Summation:

$$\begin{aligned}
S_0 &= 0 \\
S_1 &= \mathrm{fl}(0 + 1.0) = 1.0 \\
S_2 &= \mathrm{fl}(1.0 + 10^{-16}) = 1.0 \quad \text{(absorbed!)} \\
S_3 &= \mathrm{fl}(1.0 + 10^{-16}) = 1.0 \quad \text{(absorbed!)} \\
S_4 &= \mathrm{fl}(1.0 + (-1.0)) = 0.0
\end{aligned}$$

Result: $\boxed{0.0}$

But true sum: $1.0 + 10^{-16} + 10^{-16} - 1.0 = 2 \times 10^{-16}$

Error: 2×10^{-16} (100% relative error!)
Kahan Summation:

$$
\begin{aligned}
& S_0 = 0, \quad c_0 = 0 \\
\text{Step 1:} \quad & y_1 = 1.0 - 0 = 1.0 \\
& t_1 = \text{fl}(0 + 1.0) = 1.0 \\
& c_1 = \text{fl}((1.0 - 0) - 1.0) = 0 \\
& S_1 = 1.0 \\
\text{Step 2:} \quad & y_2 = 10^{-16} - 0 = 10^{-16} \\
& t_2 = \text{fl}(1.0 + 10^{-16}) = 1.0 \\
& c_2 = \text{fl}((1.0 - 1.0) - 10^{-16}) = -10^{-16} \\
& S_2 = 1.0 \\
\text{Step 3:} \quad & y_3 = 10^{-16} - (-10^{-16}) = 2 \times 10^{-16} \\
& t_3 = \text{fl}(1.0 + 2 \times 10^{-16}) = 1.0 \\
& c_3 = \text{fl}((1.0 - 1.0) - 2 \times 10^{-16}) = -2 \times 10^{-16} \\
& S_3 = 1.0 \\
\text{Step 4:} \quad & y_4 = -1.0 - (-2 \times 10^{-16}) = -1.0 + 2 \times 10^{-16} \\
& t_4 = \text{fl}(1.0 + (-1.0 + 2 \times 10^{-16})) = 2 \times 10^{-16} \\
& S_4 = 2 \times 10^{-16}
\end{aligned}
$$

Result: $\boxed{2 \times 10^{-16}}$ (Correct!)
(c) Limitations with very large M:
No, Kahan summation does **not completely solve** the absorption problem when M is very large.
Reason:
When M is so large that $\text{ulp}(M) > |x_i|$:

- The compensation c itself can be absorbed!
- If $|c| < \text{ulp}(M)/2$, then c is also lost
- The algorithm can still fail

Example:
If $M = 2^{60}$ and $x_i = 1$:

- $\text{ulp}(M) = 2^8 = 256$
- Both $x_i = 1$ and compensation $c \approx 1$ are absorbed
- Kahan summation gives same result as classical: M

Better solution: Use **pairwise summation** or sort numbers by magnitude and sum small numbers first.

5 Exam Variation 3: Theoretical Analysis

Exercise 4 (Error Bound with Absorption). Consider computing $S_n = \sum_{i=1}^{n} x_i$ using recursive summation where:

- All $x_i > 0$
- $x_i = x$ for all i (identical terms)
- Starting partial sum: $S_0 = M$ where $M \gg x$

(a) Determine the condition on n and x such that at least one term is absorbed.

(b) Assuming $x < \frac{\text{ulp}(M)}{2}$ and all terms are absorbed, what is the absolute error?

(c) What is the relative error?

(d) If we want the relative error to be less than 10^{-6}, what constraint must be satisfied?

(e) Propose a modified algorithm that avoids this absorption problem.

5.1 Solution to Variation 3

Solution 3. (a) Condition for absorption:

Absorption occurs when:

$$x < \frac{\text{ulp}(M)}{2}$$

Since $S_0 = M$, the first term x_1 is absorbed if:

$$\boxed{x < \frac{\text{ulp}(M)}{2}}$$

For floating-point number $M = 2^E \cdot m$ with precision p:

$$\text{ulp}(M) = 2^{E-p+1}$$

So condition becomes:

$$\boxed{x < 2^{E-p}}$$

Note: This doesn't depend on n directly, but if partial sum grows, ulp changes.

(b) Absolute error when all terms absorbed:

If all n terms are absorbed:

$$\begin{aligned}
\text{Computed sum} &= M \\
\text{True sum} &= M + nx \\
\text{Absolute error} &= |M - (M + nx)| = nx
\end{aligned}$$

$$\boxed{\text{Absolute error} = nx}$$

(c) Relative error:

$$\text{Relative error} = \frac{|M - (M + nx)|}{|M + nx|} = \frac{nx}{M + nx}$$

If $M \gg nx$ (which is our assumption):

$$\boxed{\text{Relative error} \approx \frac{nx}{M}}$$

(d) Constraint for relative error $< 10^{-6}$:
We want:

$$\frac{nx}{M} < 10^{-6}$$

Therefore:

$$\boxed{nx < 10^{-6}M \quad \text{or} \quad n < \frac{10^{-6}M}{x}}$$

Example: If $M = 10^{12}$ and $x = 1$:

$$n < \frac{10^{-6} \times 10^{12}}{1} = 10^{6}$$

So we can sum up to 1 million terms before relative error exceeds 10^{-6}.
(e) Modified algorithm to avoid absorption:

Algorithm: Sum Small Terms First

Strategy: Separate large and small numbers, sum small ones first.

Algorithm 3 Absorption-Resistant Summation

Input: M (large value), $x_1, \ldots, x_n$ (small values)
$S_{\text{small}} \leftarrow 0$
for $i = 1$ to n **do**
  $S_{\text{small}} \leftarrow \text{fl}(S_{\text{small}} + x_i)$
end for
$S_{\text{total}} \leftarrow \text{fl}(M + S_{\text{small}})$
return S_{total}

Why this works:

- Summing small terms first: $S_{\text{small}} = nx$
- Then: $M + nx$ where nx may be large enough to not be absorbed
- If $nx > \text{ulp}(M)/2$, the sum is not absorbed!

Numerical verification:
Using example from Variation 1:

- $M = 2^{60}$, $x = 32$, $n = 1000$
- $S_{\text{small}} = 1000 \times 32 = 32000$
- $\text{ulp}(M) = 256$, so $32000/256 = 125$ ulps
- $32000 > 128$ (threshold for absorption)
- Result: $\text{fl}(2^{60} + 32000) \approx 2^{60} + 32768$

Much better than losing all 1000 terms!

Alternative: Pairwise Summation

- Divide terms into pairs and sum recursively
- Reduces error accumulation
- Better numerical stability

Alternative: Kahan Summation

- As discussed in Variation 2
- Compensates for lost bits
- May still fail for very large M

6 Summary and Key Takeaways

Essential Points for Exam

1. **Absorption occurs when:** $|x| < \frac{\text{ulp}(M)}{2}$
2. **Classical recursive summation:** Can lose many terms if partial sum becomes large early
3. **Possible results vary:** Depends on order, signs, and when absorption starts
4. **Order matters:** Sum small numbers first, then add to large numbers
5. **Kahan summation helps:** But doesn't solve extreme absorption cases
6. **Relative error formula:** $\approx \frac{nx}{M}$ when all n terms of size x are absorbed
7. **Prevention strategies:**
   - Sum small numbers first
   - Use pairwise summation
   - Use compensated algorithms (Kahan)
   - Sort by magnitude