Floating-Point Programming Exercise

February 6, 2025

1 Exercise Tasks

Download the float-exercise.c file from Avenue which has the code template shown below. Implement the following tasks. Note, we include the math library (math.h) for functions like cos(x). When you compile, you need the -lm flag.

```
gcc float-exercise.c -o float-ex -lm
```

1. **Catastrophic Cancellation**: Compute $(1 - \cos x)$ for $x \approx 0$ using first a naive/direct calculation and then a stable version using a trig identity (Hint: how can we relate it back to sin() to prevent cancellation errors?). Compare the results, and explain what is happening in terms of underflow or overflow.
2. **Floating-Point Summation**: Compare naive summation vs pairwise summation for 1,000,000 elements of 0.1. Try to understand how pairwise sum works and gives us a more stable solution.

Code Template

```c
#include <stdio.h>
#include <math.h>

// Task 1: 1 - cos(x) calculations
double one_minus_cos_naive(double x) {
    // Direct calculation
}

double one_minus_cos_stable(double x) {
    // Numerically stable version (use some trig. identity)
}

// Task 2: Summation methods
double naive_sum(const double arr[], int n) {
    // Standard summation
}

double pairwise_sum(const double arr[], int n) {
    // Implemented for you
    if(n == 1) return arr[0];
    int m = n/2;
    return pairwise_sum(arr, m) + pairwise_sum(arr+m, n-m);
}

int main() {
    // Task 1 Test: Small angle value
    double x = 1e-8;
    printf("Task 1: 1 - cos(%.1e)\n", x);
    printf("Naive:  %.15e\n", one_minus_cos_naive(x));
    printf("Stable: %.15e\n\n", one_minus_cos_stable(x));

```

```
    // Task 2 Test: Large array summation
    const int size = 1000000;
    double arr[size];
    for(int i = 0; i < size; i++) {
        arr[i] = 0.1;
    }
    printf("Task 2:\nNaive sum:   %.15f\n", naive_sum(arr, size));
    printf("Pairwise sum: %.15f\n", pairwise_sum(arr, size));
    return 0;
}
```