3sk3

**How are you supposed to use the processors?**

Split the series into 4 parts, 1 part for each processor

**How many n's are we supposed to be using for the project?**

i used 100, it let me see the error trend pretty nicely and also took v little time

**did anyone get clarifications if it actually has to be performed by 4 seperate processes the "Simulate your parallel addition algorithm on a serial computer" is throwing me off also anyone manage to meet the bonus precision or just get close?**

i just had a function and split the size into 4 and called the function 4 times for each block ig

**can anyone provide some tips on answering questions 4a and 4b on project 1?**

I think I used the notes

**ayo brudda can i use the exponentiation in project**

i did, not sure if youre allowed though

**are we allowed to use a function to convert binary/decimal or are we supposed to do everything in the ieee 32 bit format?**

the latter, i have assumed



**How do you define/use a 32 bit ieee float in matlab?**

if you force single(x) it will use single precision 32 bit float

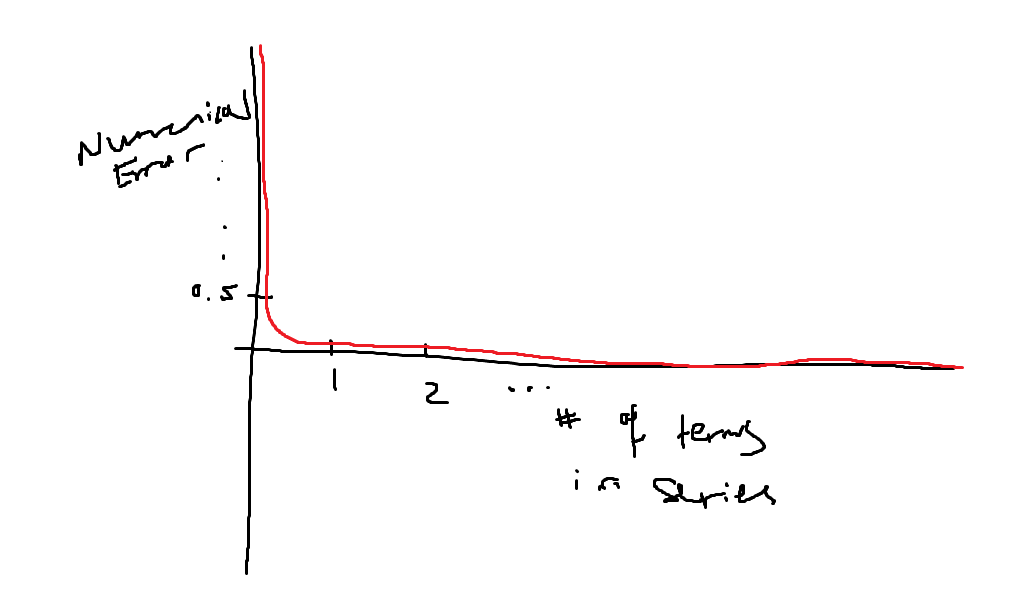
you dont need to do anything as matlab counts it as singles by def

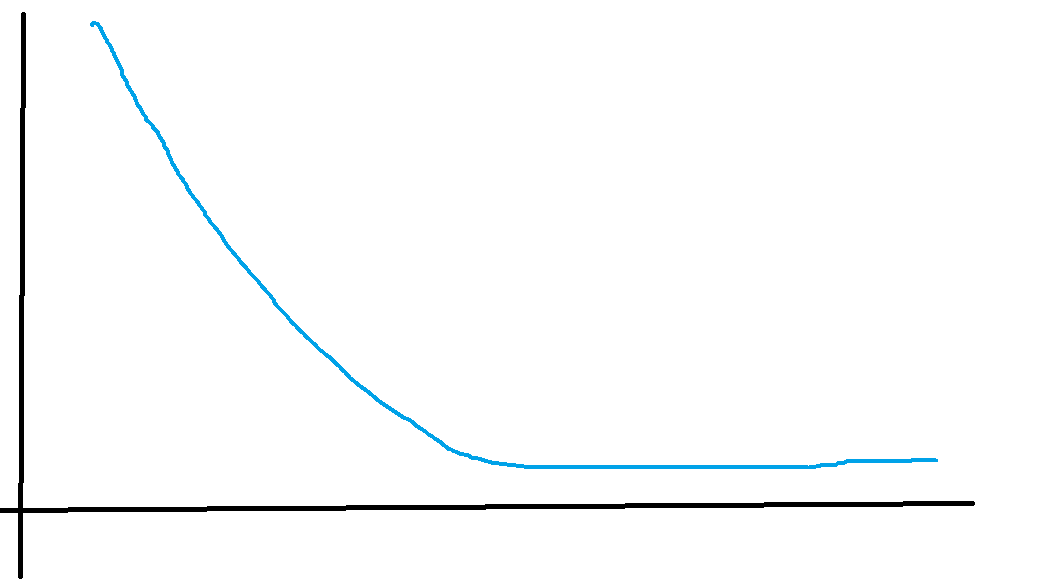
**True haha but when I did that I could only represent numbers up to the 7th decimal digit**

**So then how can we check if we achieved precision of 10^-9?**

i think calculate true error

btw, when calculating error, do i have to \* 100? cause without \*100 my precision is good





#include <stdio.h>

#include <math.h> // only for log calculation for error comparison

float calculatePartialSum(int, int);

void main()

{

unsigned int n = 1e7; // number of terms

// unsigned int n = 4294967295; // maximum number of terms

int quarter = n / 4;

int halfway = n / 2;

int three\_fourths = 3 \* n / 4;

float processor\_1 = 0.0;

float processor\_2 = 0.0;

float processor\_3 = 0.0;

float processor\_4 = 0.0;

float half\_sum\_1 = 0.0;

float half\_sum\_2 = 0.0;

processor\_1 = calculatePartialSum(1, quarter); // from 2 to quarter - 1

processor\_2 = calculatePartialSum(quarter, halfway); // from quarter to halfway - 1

processor\_3 = calculatePartialSum(halfway, three\_fourths); // from halfway to three\_fourths - 1

processor\_4 = calculatePartialSum(three\_fourths, n + 1); // from three\_fourths to n

half\_sum\_1 = processor\_1 + processor\_2;

half\_sum\_2 = processor\_3 + processor\_4;

float sum = half\_sum\_1 + half\_sum\_2;

printf("\n");

printf("------------------------------------------------------\n");

printf("\n");

printf("Ln2 approximation with %d terms: %.10f\n", n, sum);

printf("\n");

printf("------------------------------------------------------\n");

printf("\n");

printf("Ln2 approximation using C built-in log function: %.10f\n", log(2));

printf("\n");

}

float calculatePartialSum(int lower\_limit, int upper\_limit)

{

// assuming the first element of any part (lower\_limit) is even => negative

// int sign = -1;

float partial\_sum = 0.0;

for (int n = lower\_limit; n < upper\_limit; n++)

{

float term = (float)(1 / (float)n) \* pow(-1.0, n - 1);

partial\_sum = (float)partial\_sum + term;

}

return partial\_sum;

}