Assignment 2 Report

Mohana Datta Yelugoti

November 11, 2019

Contents

1 General Information: 1

2 Question 1: 1

2.1 Challenges Faced: 2

3 Question 2: 3

1 General Information:

How Graphs were plotted:

For both questions 1 and 2, i had taken different set of input's (x), then i calculated e^x in first case and *tan(x)* in second case by implementing power series of both.

Then for verification i got correspoding e^x and *tan(x)* values from Google's calculator.

You can look for that information in the .dat files. They are present in the dat folder. First column is x, second column is my calculation of function of x, third column is Google's calculation of corresponding function of x.

2 Question 1:

Implementation of Exponential series in Cortex-M4.

2.1 Challenges Faced:

One of the most important challenge was deciding how many iterations are needed for minimising the error.

The theoretical answer is Infinity. But, since we are talking about implementation, we need to come up with a way to minimise the error.

If we take a look at e$^{\text{x}}$ at generalised term of e$^{\text{x}}$, *(n + 1)th* term is $\frac{x^n}{(n+1)!}$.

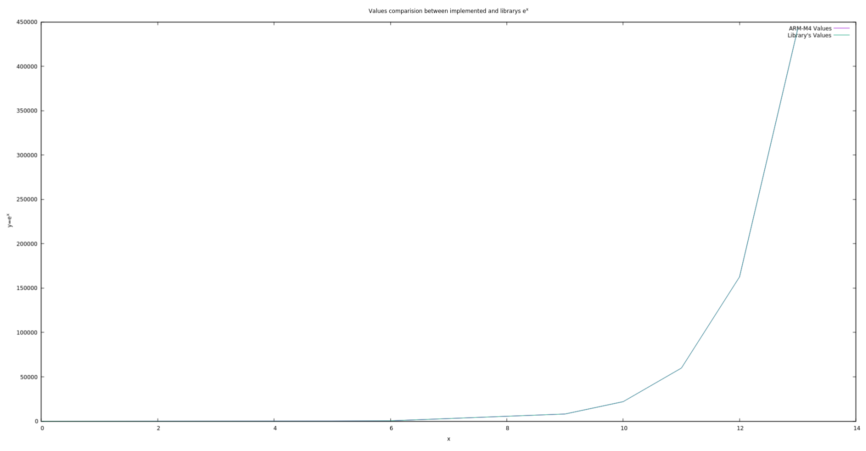
So, we need to consider till $\frac{x^n}{(n+1)!} > 1$ If it's less than 1 error it amounts to will be less, but if it's greater than 1, and if we leave it out, then error will be huge.

If we solve that equation,

$x = \sqrt[\frac{1}{n}]{(n+1)!}$

Now, for any n, this above equation gives, what is the maximum value of x, which when computed till n iterations, gives minimised error value of e$^{\text{x}}$.

In the Assignment, first 'n' is chosen, and then when it gives maximum 'x', values till that 'x' were calculated.



In above image, though it appears as one graph, there are actually two graphs. One graph was calculated with my e^x implementation on ARM-Cortex M4, other was calculated from Google's calculator.

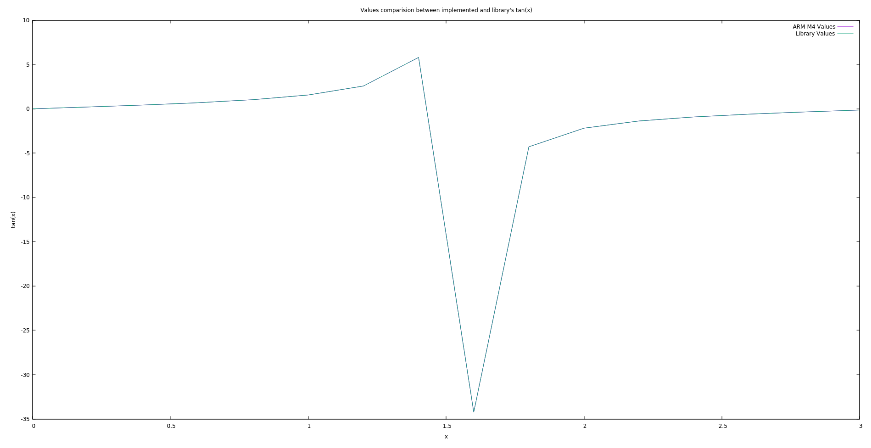
Only few x's in the range [0, 13] were considered. **<u>The values obtained were almost equal</u>**.

3 Question 2:

tan(x) was calculated first by calculating *sin(x)* and *cos(x)* and then dividing them.

After a series of trial and errors, I decided to go with **20** terms. If i took too many terms, the regiters were hitting **NaN**.

The challenge faced was enabling, luckily manual[1] tells how to enable this. It also describes different instructions of FPU of ARM processor.



The values obtained from my implementation and from Google's calculator **were almost equal**. The above image is actually **two graphs, but they both overlap, so it appears as a single graph.**