Documentation of the C Code Optimization Assessment

1.1 Summary

Many optimizations were performed to speed up the computation time of the original C code. Most of which include computing known values ahead of time and reducing redundant calculations. A summary of the execution times are given:

- Original C Code: 16 minutes and 27 seconds
- Modified C Code: 4 minutes and 25 seconds

A full analysis is given below.

Note: Certain approximations were made without knowing the context of what problem this code solves. Knowing the full context of the problem may verify or nullify the assumed validity of these approximations.

1.2 Original C Code

The original C code provided ran in approximately 16 minutes and 27 seconds. GPROF was used to conduct a flat profile of the code. This showed that 69.12 % of the execution time was spent on calls to the *exp*, *pow*, and *log* functions. This provided an initial guide to begin optimization.

In addition, numerous constants were defined inside "function_j", which were being recalculated for each loop iteration. These redundancies include static value re-declarations, many multiplication and division operations, and expensive calls to "log" and "tan" functions.

1.3 Optimized C Code

The modified C code ran in approximately 4 minutes and 25 seconds. The first optimization made was to precompute and combine as many of the constants in "function_j" as possible. These calculations allowed numerous constant doubles to be declared at the beginning of the C file.

The second optimization made was to pull repeated calculations up to their respective level in the nested for loops. For example, "function_j" contains the term *pow(f,-5)*. This only needs to be calculated in the outer most loop, but was calculated inside the inner most loop. Similar steps were followed to pull out calculations from the inner-most loop into the fp-loop, including a boolean flag to reduce the number of calculations in determining *sigma* by 50% and pulling the first exponentiation up one level in the loop.

Two approximations were used to further speed up the calculations: (1) a Taylor series approximation and (2) a log-lookup table.

A Taylor series was used to approximate the *pow* function for small powers. This is valid as the upper and lower bounds of the base and power for certain variables are known prior to running the code. This allows us to define the number of series terms required to accurately approximate the *pow* function.

A log-lookup table was used to quickly interpolate values of *ln(x)* during runtime. It was known prior to runtime that the desired range would be in $x \in [0.1, 10]$. A table of known values of *ln(x)* was defined the the "log_table.h" file at 0.1 step increments in x.

The above steps allowed for a 73.15% reduction in execution time.

1.4 Future Optimizations

As stated in the summary, lacking full knowledge of the context of the problem hinders optimization results. Two major optimizations may be possible given a deeper understanding of the context.

Firstly, it may not be necessary to use such small increments in each loop. A step size of 0.01 is relatively small and a larger step may be equally appropriate. For example, using loop steps of 0.1 instead would reduce the number of loop iterations from approximately one billion down to just one million. This would provide major computational savings.

Secondly, there are terms which may be able to be better approximated or avoided. For example, the outer-most loop and middle loop both produce divide- by-zero results. The problem may not need to actually sample down to zero, or if it does, a small argument approximation may be more appropriate.

Knowledge of the problem context is crucial in determining which approximations are valid and which optimizations may be pushed further.