Lab 1 Taylor’s Series and Numerical Precision

Taylor’s Series and Numerical Precision

1. Implement your own programs using floats to approximate the integral of the \( \sin(x) \) using the Reimann Sum from \( x = -0.5 \) to \( x = 0.5 \).
   a. Implement one method using calls to the system’s \( \sin() \) function.
   b. Implement another method using 2 terms in the Taylor’s series expansion for \( \sin(x) \).
   c. Implement another method using 3 terms in the Taylor’s series expansion for \( \sin(x) \).
   d. Implement another method using 4 terms in the Taylor’s series expansion for \( \sin(x) \).
2. Repeat step 1 above, using doubles rather than floats.
3. Try out the different compiler options –O, -O1, -O2, -O3 and –O (or equivalent options in Visual Studio — release versus debug). Provide some explanations and results on how they affect your performance (not why). Use the best one for your table results. Note, that is a capital-O, not a lowercase –o. Your g++ or gcc line would look like: g++ -O3 -o lab1 lab1.cpp -lm. The –lm loads the math library.
4. Analyze your results and explain any unusual results. This is the bulk of your assignment!!
5. Use powers of two for your MAX_ITER, from \( 2^{12} \) up to \( 2^{28} \).

Hints

- Limit MAX_ITER to less than 1,000,000 while debugging. Your program should take less than 5 seconds.
- To force a constant to be in floating-point representation (usually double), use 3.0 rather than a 3 with no decimal point. Adding a float flag forces single-precision, such as 3.0f as opposed to 3.0.
- Provide a table and optionally a plot of your results. Compare these results to an analytical solution.
- Include a pretty print of your code in your write-up.
- Define your own type for real numbers, so you can change it easily. For instance, I used: #define RAC_Real float.
- See /usr/class/sce/rcpp/RESOLVE_Foundation/Miscellaneous/Keywords.h for transitioning from Resolve to C++.
- For timing a function under Windows, you can use the Performance Timer listed below:

```c
#include <windows.h>
LARGE_INTEGER nStartTime, nEndTime, nSystemClockFrequency;
QueryPerformanceFrequency( &nSystemClockFrequency );
QueryPerformanceCounter( &nStartTime );
Reimann_Sum = sinReimann(); // Do some work!!
QueryPerformanceCounter( &nEndTime );
__int64 nTime = nEndTime.QuadPart - nStartTime.QuadPart;
double elapsedTime = (double) nTime * 1.0e-09;
printf("Actual sinf() Integral = %e, Time: %.9f seconds\n", Reimann_Sum, elapsedTime);
```

- For timing a function under UNIX, you can use the system clock utilities as listed below (see man gethrt ime).

```c
#include <sys/times.h>
#include <limits.h>
hrtimes_t start, end;
float elapsedTime;
start = gethrtimes();
Reimann_Sum = sinReimann(); // Do some work!!
end = gethrtimes();
// gethrtimes returns a long long in nanoseconds.
elapsedTime = (float) (end-start) * 1.0e-09;
printf("Actual sinf() Integral = %e, Time: %.9f seconds\n", Reimann_Sum, elapsedTime);
```