EECE5640 HW3

Quinn Arbolante

March 2025

Question 1

Part A

The single-precision version of my implementation is `sinx_sp.cpp`. The double-precision version of my implementation is `sinx_dp.cpp`. I chose the following values to compute $x = 9, 11, 13, 17$. I measure the absolute error of my computed value with $std :: sin()$, and I compute $sin(x)$ with 10, 12, 14, 16, 18, and 20 terms.

The results for `sinx_sp.cpp` are the following:

- Error of $sin(9)$: 1.840349 (10 terms), 0.041531 (12 terms), 0.000589 (14 terms), 0.000103 (16 terms), 0.000099 (18 terms), 0.000099 (20 terms)
- Error of $sin(11)$: 116.148356 (10 terms), 5.940489 (12 terms), 0.159389 (14 terms), 0.003321 (16 terms), 0.000646 (18 terms), nan (20 terms)
- Error of $sin(13)$: 3585.307862 (10 terms), 364.641236 (12 terms), 19.23652 (14 terms), 0.584749 (16 terms), nan (18 terms), nan (20 terms)
- Error of $sin(17)$: 840109.9136 (10 terms), 260157.3511 (12 terms), 41285.31985 (14 terms), 3714.130399 (16 terms), nan (18 terms), nan (20 terms)

The results for `sinx_dp.cpp` are the following:

- Error of $sin(9)$: 1.84025 (10 terms), 0.041432 (12 terms), 0.00049 (14 terms), 0.000003 (16 terms), 0 (18 terms), 0 (20 terms)
- Error of $sin(11)$: 116.147484 (10 terms), 5.939594 (12 terms), 0.158494 (14 terms), 0.002426 (16 terms), 0.000023 (18 terms), 0 (20 terms)
- Error of $sin(13)$: 3585.303275 (10 terms), 364.635922 (12 terms), 19.231236 (14 terms), 0.579464 (16 terms), 0.010711 (18 terms), 0.000128 (20 terms)
- Error of $sin(17)$: 840109.8999 (10 terms), 260157.0935 (12 terms), 41285.04874 (14 terms), 3713.85615 (16 terms), 203.860743 (18 terms), 7.223645 (20 terms)

Unfortunately, due to the high range in error, these values don't look great when plotted (I could've tried a logarithmic scale in the y-axis, but I couldn't figure out how to do it in Google Sheets).

Conclusions

Both implementations converge to the answer at about the same rate. However, the single-precision implementation appears to have a lower bound for the error, as additional terms don't decrease its error for $sin(9)$. Additionally, for larger values of x, the single-precision implementation fails to compute terms resulting in nan results. I believe this is because the terms become too big to store as `floats`.

The double-precision implementation is able to reach an error of 0, and it does not fail to compute terms even for large values of x. I believe this is because the additional bits that *double* has to store a wider range of numbers lets it not run into the same issues that the single-precision implementation had.

Part B

The IEEE 754 single-precision representations are:

1. 2.1 = 0x40066666 ($s = 0$, $q = 128$, $c \approx 0.05$, so $2.1 = 1.05 \times 2^1$)
2. 6300 = 0x45c4e000 ($s = 0$, $q = 139$, $c \approx \frac{6300}{4096} - 1$, so $6300 = \frac{6300}{4096} \times 2^{12}$)
3. -1.044 = 0xbf85a1cb ($s = 0$, $q = 127$, $c \approx 0.044$, so $-1.044 = -1.044 \times 2^0$)

The IEEE 754 double-precision representations are:

1. 2.1 = 0x4000CCCCCCCCCCCD
2. 6300 = 0x40B89C0000000000
3. -1.044 = 0xBFF0B4395810624E

With similar reasoning as single-precision.