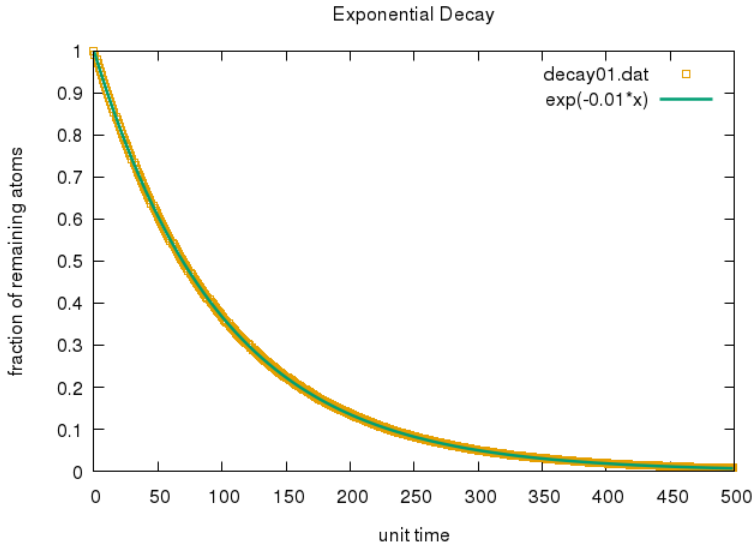

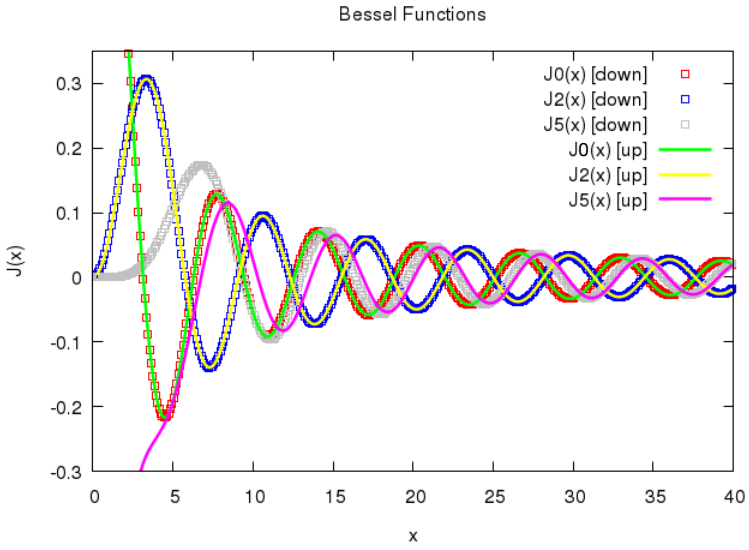

The algorithms give different results for some cases is because as we recurse using the "up" algorithm to recurse up to a larger order *n* of *Jn(x), e.g. n* = 5 in this case, the process suffers from subtractive cancelation where the difference of two "large" functions is used to produce a "small" value for *Jn(x)* such that the precision is reduced. Hence the "up" algorithm has lost precision and significance in its values produced. Whereas using the "down" algorithm avoids subtractive cancelation by taking small values of Jn+1(x) and Jn(x) to produce a larger Jn-1(x) by addition such that the loss of precision is less significant and the resulting values are more precise and trustworthy.

The value I determined for the floating point number is $1.23977661 \times 10^{-5}$

In the overflow/underflow test, for floats and doubles, the underflow limits were greater in magnitude compared to the overflow limits. That is, underflows occurred at loop number 148 for floats and 1073 for doubles, where as overflows occurred at loop number 126 for floats and 1022 for doubles. Hence, floats and doubles are more capable of a higher precision for negative numbers than positive numbers, in terms of the mantissa of the scientific notation. These numbers corresponded to the biggest and smallest numbers, 3.402923×10^{38} and 1.401298×10^{-45} respectively for floats, and 1.797693×10^{308} and $4.940656 \times 10^{-324}$ respectively for doubles, the 32-bit system can store.

As for integers, the overflow occurred at $2147483647 = 2^{31} - 1 = 2^{32-1} - 1$ and underflow occurred at $-2147483648 = -2^{31} = -2^{32-1}$, which makes sense since the system is 32-bit and these are the biggest and smallest integers, respectively, it can store.

As for unsigned integers, the overflow occurred at $4294967295 = 2^{32}$, which again makes sense since the system is 32-bit and it's the biggest unsigned integer it can store.

As for the "float vs. double" test, the "real time" varied generally randomly throughout all the runs. However, the "user/sys times" were shorter for floats compared to doubles, which makes sense since floats are less precise, hence take up fewer bits and require less time to compute. These times were also generally shorter for optimized compared to unoptimized executions, for both floats and doubles.