MATH 458 Homework 2

1a) Relative error is most useful when comparing numbers with vastly different sizes. After all, relative error is designed to be compared with something else. For example, having an absolute error of 5 on 100 (5%) is completely different than 5 on 90,000 (0.0055%), despite them having the same absolute error.

Absolute error is sometimes more useful when no comparison is being made, or the comparison is equal in magnitude. Since absolute error is more tangible (being in the same units), it can be used in more simpler measurements. For instance, measuring a pencil with a ruler can have an error or +- 0.1 cm; it’s more easily understood, and does not misrepresent the magnitude of the error.

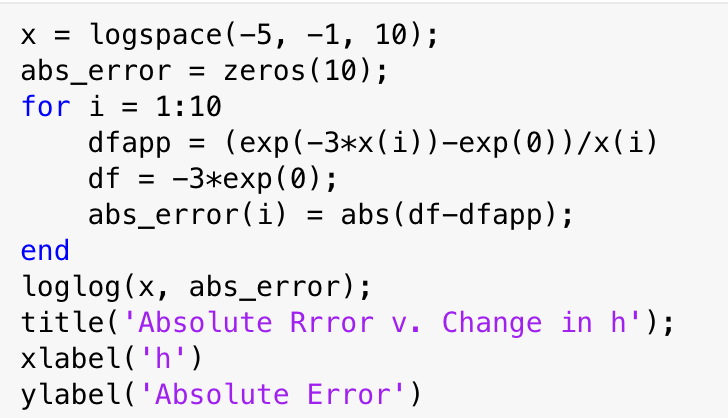
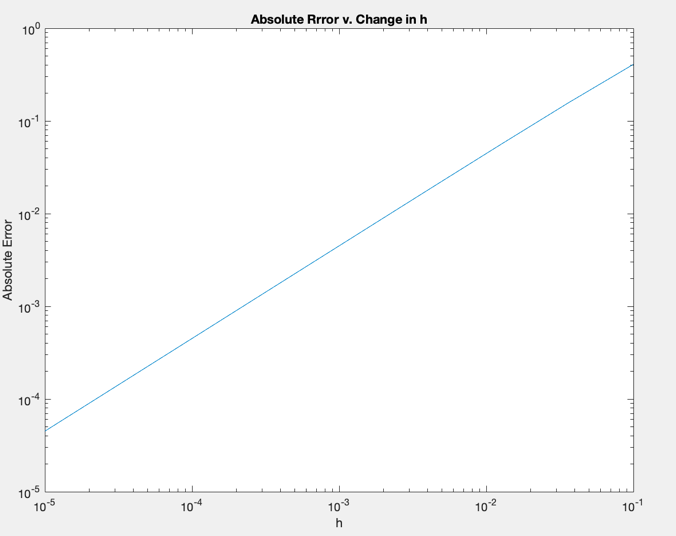
b) Roundoff errors are usually made by computers because they have finite memory and needs to truncate an irrational number. Discretization errors (and approximation errors in general) are human made in the sense that we need to chunk up some continuous measurements in discrete parts.

c) Accuracy is related to the errors that accumulate during calculations, whether they’re roundoff errors or discretization errors. Meanwhile, efficiency measures the time it takes for the algorithm to compute. There are many ways to program the same solution, but different methods take longer than others, and when computing swaths of data that time may be very noticeable. Robustness, on the other hand, is how versatile the algorithm is in handling different types of inputs.

d) Problem conditioning is properly utilizing a problem that would not produce drastic differences at small changes. A classic ill-conditioned problem is evaluating tan(x) near x = pi/2 because the change is so drastic. On the other hand, algorithm stability is when the entire algorithm actually produces the exact result at small changes. An algorithm can only be stable if every step is well-conditioned.

e) In a floating point system, roundoff errors will always occur in every arithmetic operation because the computer has finite memory. This is most often linear and adds up every time an operation is done. This is tolerable if the linear rate is moderate, and thus won’t accumulate that much error in multiple iterations of the algorithm.

2.



The results make mathematical sense. As we shrink the h exponentially, the accuracy becomes exponentially better because of the definition of the derivative

This means that the derivative approximation will be more accurate as h approaches 0. In the graph this means that the smaller the h, the smaller the absolute error.

3a)

b) Start with

, where represents the error accumulated in every iteration.

Notice that this forms a geometric series with r = 0.1. For a given , at any , the error is bounded by

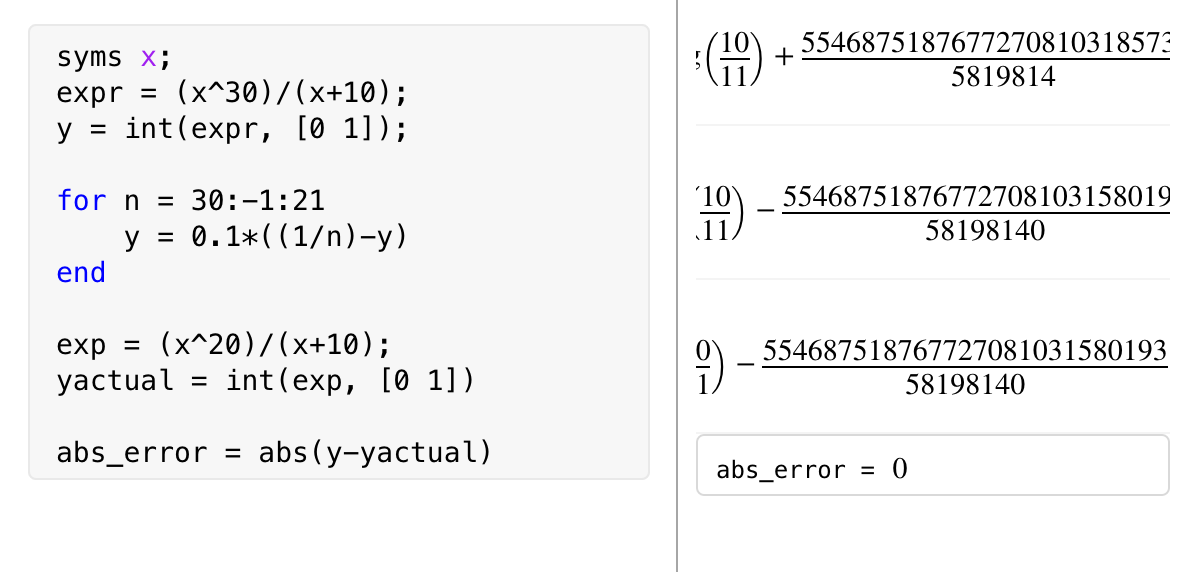
We know for sure that the error is bound by at any step. Substituting to , we get

Finally, we must show that the above quantity is less than at an .

Because this quantity is always positive and can be upper-bounded

There will always exist an for any , hence proven.

c) The algorithm is stable because it can take indefinite iterations and still be reasonably accurate. This is because the construction, going backwards in n, allows each roundoff error to be divided by 10 in every iteration.

d) 

Using the backwards recursion approach, I found that it doesn’t really matter which n1 I started on because the error propagation is so small. I tested this starting with 25, then went up to 30, and went up to 50 and the absolute error was still 0 (or at least computationally negligible.

4a) Beta is the base of the number system (humans are conditioned to use base 10 while computers use base 2). T is the precision (i.e. number of digits), L is the lower bound on the exponent of the base, while U is the upper bound of that. For instance, if beta = 10, L = -1, U = 2, then the highest number that can be represented is 999 and the lowest positive number is 0.1.

b) i.

ii.

e.g. x = 23.367488203

Chopping: 2.336 \* 101

Rounding: 2.337 \* 101

c) Rounding unit is a bound on the relative error in a given decimal system. This is important because no matter how carefully we do a certain operation, we are guaranteed an error at most the size of the machine epsilon. The IEEE in particular has a rounding unit standard, so it is a known number that’s universally used in many machines.

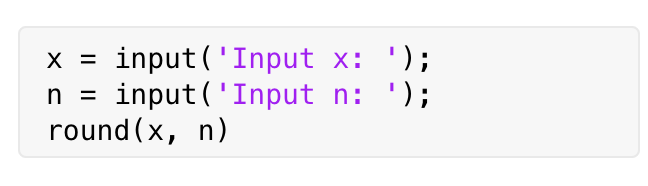
d) Overflow: when a number is too large to be represented in a certain decimal system

Underflow: when a number is too small to be represented in a certain decimal system

Overflow is often fatal in calculations because the machine does not know what to do with the large number. But underflow is not as damaging because the machine usually rounds to 0 and continues normally.

e) Cancellation error is when we have a subtraction operation and, due to some roundoff error, the two values are so close that they cancel out. Hence, some problems need rephrased algorithms for the actual value to be calculated. A common example is in estimating the derivative of a function using

At small h. If the numerator cancels out because the difference in h is so small, then it will be magnified by the denominator.



5.

6.

Most suitable for computation:

The reason being is that is prone is 1.7977e+308, having x = 1.338e+154 (just slightly more than its square root to cancellation errors when x is very large. Given that the largest positive floating-point) will essentially nullify the effect of -1 in the square root.

Which is a fatal error.

7a)

is numerically difficult because , so there will be no solutions to the system of linear equations.

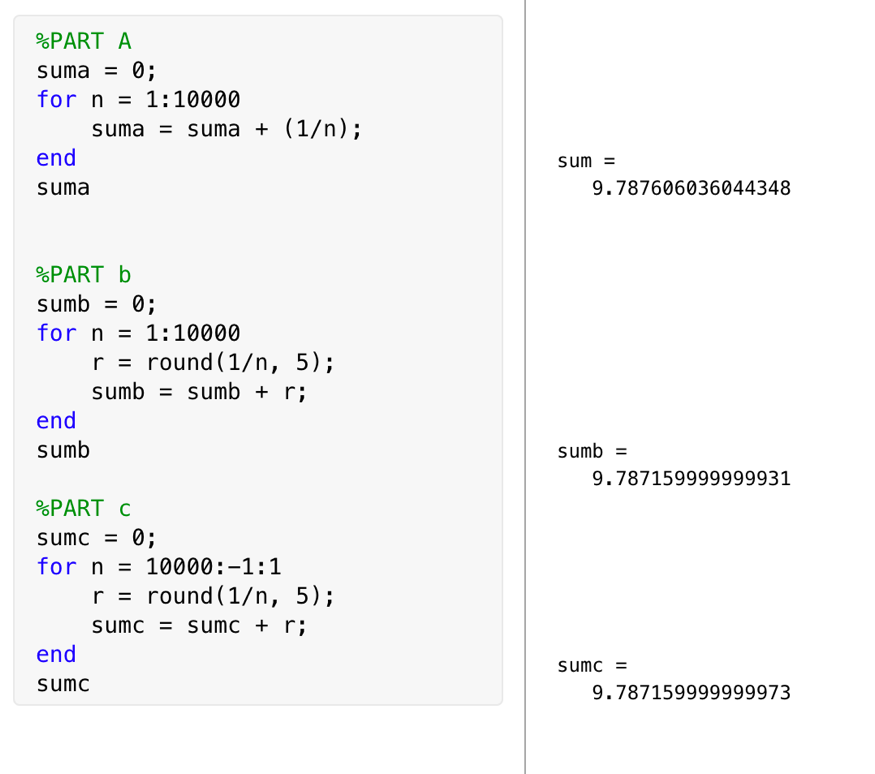
b)

Summing the two equations,

Since a and b are always positive, the denominator will never be 0 and z will have a real value. Hence, this formula is stable.

c) This is true. Part a proved that the linear system is ill-conditioned. But finding x+y as a pair, as part b showed, is always possible for sufficiently large, positive integers of a and b.

8.

Part a definitely produced the most accurate results because the algorithm did not force each iteration to be rounded by 5. Instead, it used MATLAB’s standard accuracy. Part b and c, on the other hand, produced a value that is actually very close to 5 decimal points. (Since I displayed the number as a long, there might be small computational mistakes here and there.) This makes sense because I forced the algorithm to round the numbers to the nearest 5th decimal. There aren’t too many differences with part b and part c because the rounding method did not change; I merely flipped the order or summation, which shouldn’t have much of an impact. Instead, the small differences in the 15th decimal place and onwards are probably minor roundoff errors that propagated throughout the summation.