Project 1

Van Tran

9/17/15

Math 3315

Part A:

In the first part of the project we used the Taylor series to estimate the value of a function by using the nested multiplication algorithm and deriving the derivatives of the function by hand and putting them in vectors. The nested multiplication function was used to reduce the cost of estimating the function via Taylor series. We calculated the Taylor series for p4, p8, and p12 and found each of their respective errors and found that as we increase the pn of the Taylor series, the value gets closer to the actual value (and of course error decreases as well).

I started by first opening both the Matrix and Python notebook examples to see how to use the new tools provided by the instructor. I then implemented the nested multiplication algorithm shown on page 8 of the book. I then created a vector to be used for the x values to be used in the Taylor series and created a vector of coefficients to be used in the nested multiplication algorithm. I then used these to find p4, p8, p12 and then found the absolute value of pn-cos(x) to find their absolute errors. I then displayed these results using the Python notebook.

The best estimation of cos(-3) is p4(-3). This makes sense because we found the Taylor series at a=0 which makes estimations around that value more precise the more pn we use for the Taylor series. So p12 is more precise the closer x is to 0, so it makes sense that it is not the best estimation as x = -3. Yes, my plots are consistent with my derivation. The purpose of this part was to show how although using computers can almost never get the exact answer, as computers get stronger and as better algorithms are designed, we can get closer and closer to the answer, which basically gives us the answer for all intents and purposes.

Part B:

In part B of the project we calculated the error in a forward finite difference approximation. We approximated a derivative of the function by using the definition of a derivative:

(f(a+h) –f(a))/h (lim h to 0)

However we cannot evaluate the function when h is zero so we make h as small as possible to get the best answer. However we also have floating point errors which we have to take into account. On paper as h gets smaller the error will always get smaller, but when we add in floating point errors, the error behaves differently.

I started by first creating the array of n values to use to create the decrementing h array. Then I made functions for f(x), f’(x), and f’’(x) and used these functions to estimate the relative error r and the upper bound relative error R.

As h decreases, r and R get smaller and smaller, until the end where r and R sharply spike up as h gets extremely close to 0. The values n=25 and h= 1.49012e-08

gives the best approximation of the derivative. This is because as h gets smaller although the closer the derivative estimate is, the floating point error increases. At n=25 is the point where the sum of the floating point error and derivative estimate error are the smallest. The purpose of this part was to show that floating point error has to be taken into account when using computers and change the behavior of the general trend of errors on paper.



