Project 4: Numerical Integration

Erik Gabrielsen

Math 3316

December 2, 2015

Overview:

In this Project, I designed a function that would approximate an integral using a method of my choosing as well as a test function to ensure that it works correctly and efficiently. I will then incorporate this function into an adaptive Numerical Integration that requests to find a result of a desired accuracy using the least possible computational effort. I will then apply these two methods to model the concentrations of carbon relative to iron in an alloy during the process of carburizing. Finally, I use a root finding function to find an approximation of the temperature of carbon need to reach a certain carbon concentration at a certain time.

Part I – High-Order Numerical Integration

In Part I, I was provided a composite Gaussian numerical integration formula with 2 nodes that approximates a function over an integral [a, b] with a convergence of *O(h4).*  My task is to create a composite numerical integration routine that approximates the integral with a convergence of *O(h8)* in a function with the following signature:

double composite\_int(Fcn& f, const double a, const double b,

const int n);

My first thought was to add one additional node to the original 2 node Gaussian method. This increased the convergence but only to a convergence of *O(h6).* Therefore, I used 4 nodes in my version of the Gaussian numerical integration. To test the convergence of this routine I created a file (test\_int.cpp) which would run the function several times with different values of n nodes to ensure that it converges correctly. Using 10 different n values in the range [20, 200] I determined that the method converges at a rate of *O(h8)* as desired. Here is the output from the test routine:



As the number of nodes increased, the approximation increased in accuracy, as expected as you are reducing the amount of error during each subinterval. I chose to solve solve for the eight optimal evaluation points myself and incorporate the values as constants in order to save computation time in calculating the approximation of the integral, rather than reevaluating these constants every time the function is called.

Code: *composite\_int.cpp*

*test\_int.cpp*

Part II – Adaptive Numerical Integration

In Part II, I was to design an adaptive numerical integration function called *adaptive\_int.cpp* that will compute the integrand of a function along the interval [a, b] by calling my *composite\_int()* function in an adaptive manner. The goal is to create such a routine so that the result uses the least possible computational effort. I will achieve this by minimizing n, but not working overtime to do so. I then created a test function called *test\_adapt.cpp* that uses my *adaptive\_int()* function to integrate the same problem as in part I but this time stopping when the increase in accuracy is less than a combination of given relative and absolute tolerances. By doing this, the routine will refrain from under-guessing or over-guessing the correct value of n. I now used the tolerance with atoli = rtoli/1000, and rtol = {10−2 , 10−4 , 10−6 , 10−8 , 10−10 , 10−12}. My method does achieve the desired relative error at each tolerance level, however when the relative tolerance reaches 10-10 the approximation becomes too accurate as the value of n doubles and over-guesses the correct value of n. Here are the results of the *test\_adapt.cpp* routine:

