**Question 1:**

The first question asked for the LU decomposition of matrix A, done by hand. The following

steps were performed, bearing in mind A = LU.

1. Obtain the row-echelon form of A via row-reduction, which yields U
   1. -1R1 + R2 and -2R1 + R3 are the first 2 row operations
   2. Now we can express U = l1\*A, which is the multiplication representation of the 2 row operations
   3. Then -2/3(R2)+(R3) was performed
   4. Now we can express U = l2 \* l1 \* A
2. Multiplying the two matricies l2 and l1 will yield l-1, which is equivalent to L.
   1. Thus, L = l1-1 \* l2-1 , so the inverses of l1 and l2 were calculated.

In the end, the LU decomposition resulted in:

\*highlighted numbers were the unknowns that we were asked to solve for

**Question 2:**

To determine the solution of Ax, Ax was set equal to be, then the following steps were taken:

1. If Ax = b and A = LU from LU decomposition, then LUx = b
2. Ly = b and Ux = y, y was solved first in the former equation, then its values were used in Ux = y to solve for x, which is the solution.

Thus, using the L and U matricies found using LU decomposition, the solution is:

**Question 3:**

To obtain the row-echelon form of matricies B and C as part of the LU decomposition process, the exact same 3 row-reduction operations performed in Question 1 are conducted. This implies that matricies A, B, and C all share the same L matrix. Obtaining the row-echelon forms of B and C reveals that both matricies share near identical U matricies. Thus, it is safe to say that the LU decompositions of B and C are similar to one another.

**Figure 1:** Composite Midpoint Rule Error Convergence

\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Figure 2:** Composite Trapezoid Rule Error Convergence



\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Figure 3:** Composite Simpson Formula Error Convergence



\*the y-axis’s scale that’s clipped by the plot’s title is supposed to read: 0 x 10-4

**Table 1:** Error Values for Composite Newton-Cotes methods with Respect to N

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Midpoint Error** | **Trapezoid Error** | **Simpson’s Error** |
| 17 | -0.000390633268067 | 0.000573053429959 | -0.000114515002546 |
| 33 | -2.34078647070e-05 | 3.01931351884e-05 | -6.73544644502e-06 |
| 65 | -1.49151675898e-06 | 1.81188560955e-06 | -4.26933215003e-07 |
| 129 | -9.51189536025e-08 | 1.12110846828e-07 | -2.71895927995e-08 |
| 257 | -6.02159477836e-09 | 6.98940283428e-09 | -1.72065917070e-09 |
| 513 | -3.79023035180e-10 | 4.36558345029e-10 | -1.08295594714e-10 |

As demonstrated visually by Figures 1 to 3 (the error plots) above, Simpson’s formula converged the fastest of the three Newton-Cotes methods and is in line with the theoretical error calculated in Table 1. This is probably due to the Trapezoidal rule being an average of the left and right Riemann Sums; as a result, the Composite Trapezoid Rule performs poorly when it comes to approximating very curvy equations, like ours which involves a few sine and cosines. The same idea occurs with the Composite Midpoint Rule as it samples the midpoint between two subintervals and uses that lone point as the basis of interpolation.

Simpson’s Rule, however, places a parabola across every two subintervals on top of the fact that it also utilizes at least twice the number of subintervals compared to the other two. Generally, using more subintervals to approximate integrals yields a more accurate result, and such is the case we see here. These points explain why Simpson’s Formula is superior to the Midpoint and Trapezoid Rules.

The next component of Question 3 was to estimate the explicit errors for the Trapezoid Rule and Simpson’s Formula, which was done through estimating the appropriate derivatives.

To estimate the derivative f’’(ζ) for the Composite Trapezoid Rule, for each N-value, the second derivative of f(x) was obtained by passing f(x) into the diff() routine and setting the nth derivative to 2. Then f’’(x) was calculated across the domain of a to b; the maximum f’’(x) was obtained, and it’s respective x value was verified to fall between a and b. This maximum f’’(x) value thus became the derivative estimation f’’(ζ) for the Composite Trapezoid Rule. Obtaining f(4)(φ), the derivative estimation for Composite Simpson’s, used the exact same methodology, except we were operating with the 4th, not the 2nd, derivative of f(x).

In our case, the derivative estimations results were:

f’’(ζ) = f2\_eta = 8.6612

f(4)(φ) = f4\_eta = 384.7914

Δx values were calculated for both Composite Simpson’s and Composite Trapezoid. The derivative estimations and Δx were then plugged into the explicit error formulas for each respective method, and the explicit error obtained.

Calculating the errors via Richardson’s Extrapolation was an extension of what has already been calculated. Due to Richardson’s Extrapolation using both Δx and Δx/2, we needed to calculate the composite integration of 2\*N using the Simpson’s and Trapezoid, as a half-step in x is equivalent to doubling the number of subintervals; we already calculated the composite integration of N in the previous component of this question. Obtaining the 2\*N integration used the same methods in the previous component: calculating the appropriate {wi, xi} pairs using the functions developed in Question 1, then feeding the generated {wi, xi} pairs into the weightPairSum function.

Calculating Richardson’s Extrapolation error was only a matter of plugging the integral calculation using N and integral calculation of 2\*N for a Newton-Cotes method into the error formula, which I’ve hard coded into a separate function called RichardsonExtrap().

These calculations resulted in the values found in the table below:

**Table 2:** Error Values for Composite Trapezoid and Composite Simpson’s Formula methods using both Explicit Error and Richardson’s Extrapolation Error Formulas with Respect to N

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Explicit Error** | | **Richardson’s Extrapolation** | |
| **N** | **Trapezoid Rule** | **Simpson’s Formula** | **Trapezoid Rule** | **Simpson’s Formula** |
| 17 | 0.6993538615661 | 0.0009790763802 | 0.0007285920531 | -0.000144746249 |
| 33 | 0.1748384653915 | 6.895352924e-05 | 3.798786843e-05 | -8.44533574e-06 |
| 65 | 0.0437096163478 | 4.580984607e-06 | 2.270955285e-06 | -5.34096383e-07 |
| 129 | 0.0109274040870 | 2.952931808e-07 | 1.403061892e-07 | -3.39939620e-08 |
| 257 | 0.0027318510217 | 1.874475488e-08 | 8.741642669e-09 | -2.15093721e-09 |
| 513 | 0.0006829627554 | 1.180708796e-09 | 5.458543247e-10 | -1.35366681e-10 |

Overall, Richardson’s Extrapolation Error calculations were more accurate than the errors yielded by the Explicit Error formula. The Composite Trapezoid error benefitted more from Richardson’s Error calculations than Composite Simpson’s Formula did.

The final component of Question 3 examines the Gaussian Quadrature implemented back in Question 2. To confirm the question’s assumption that Gaussian Quadrature performs rather poorly though plots and numerical data, we placed the values of N into a vector and pre-emptively allocated vectors to hold integral calculations, both exact and Gaussian, along with error calculations (exact subtracted by Gaussian).

By using a for-loop to iterate through each N-value, the Gaussian Quadrature integration was calculated; subtracting this value from the exact integration over the same [a b] domain yielded the error.

Plotting the error demonstrates what the question already confirmed: Gaussian Quadratures is a terrible method to estimate the integral of the function given in this assignment. The only N where the error is ‘better’ is N=4.

**Figure 4:** Gaussian Quadrature Error Convergence



Gaussian Quadrature works poorly with our given formula f(x) = 1 + sin(x)\*cos(2x/3)\*sin(4x) because our {wi, xi } pairs did not maximize the order of accuracy of the Gaussian Quadrature Formula. The {wi, xi } pairs were determined largely by the number of points to sample (N) and the domain that we wanted to integrate over. To make the Gaussian Quadrature give better results, a and b should be chosen such that the mapped xi should maximize the order of accuracy while wi remains constant for each N.

Sources of Information:

Sources of information include the slides on Numerical Interpolation posted on Canvas, as well as MatLab’s documentation on routines such as saveas() and figures(). Tajo, as always, was superbly helpful in his office hours in clarifying what he was looking for with our plots and the formula for Richardson’s Extrapolation. Also, I did browse the Discussion forum on Canvas for any tips but didn’t find them super helpful this time around. I did not collaborate with any classmates on this assignment.

As far as source codes went, I implemented all the functions that didn’t already come with MatLab. For the Newton-Cotes and Gaussian quadratures, I used the formulas in the Numerical Interpolation slides on Canvas and implemented them. I also browsed the links at the end of **Numerical\_Integration(a)-11.pdf** to gain further insight on the quadrature methods.