Quadrature Analysis, Program 3

Colton Piper

November 20th, 2017

1 Executive Summary

This report focuses on five basic quadrature methods, being two closed Newton-Cotes Methods, two open Newton-Cotes Methods, and a two-point Gauss-Legendre Method. The closed methods are the Trapezoidal rule and Simpsons rule. The open methods are the midpoint rule and the two point open rule. We discuss errors and how small intervals need to do be to be with a certain tolerance of error. Our code also refines the sub-intervals while still using the evaluations of the last iteration.

2 Statement of the Problem

First code the five quadrature methods that accepts a uniform mesh of any size and returns the approximated integral. Next we will make five routines that have a refining process to get a more accurate approximation. The refining process increases the number of sub-intervals within the interval for the integral. Now we will increase the number of sub-intervals by a factor of k which we choose to be the smallest integer such that we get complete reuse of the function evaluations evaluated in the last iteration. The Gaussian method we will just take the $k = 2$ as the refinement integer.

Next we will discuss the methods we used in our code to efficiently reuse the each function evaluation. Then for each integral in the problems list we will estimate the sub-interval size needed to satisfy various error demands using the simple composite quadrature methods. Next we will run the codes for various error requirements, summarize appropriately

and concisely your observations, compare the observed performance to predictions. Then comment on any behavioral differences observed between the problems and explain them based on differences in the functions being integrated. Discuss and compare the number of function evaluations. Then after we will do the same analysis for using our refining methods.

3 Description of the Mathematics

3.1 Non-Composite Rules

The mathematics of all these quadratures are fairly simple mathematically. The Newton-Cotes methods are just integrating the interpolating polynomial for however many points you want to include in the interval. The first closed rule is the trapezoidal rule. So we use a linear interpolating polynomial that interpolates at the boundary of the interval. Thus we have

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b f(a) + \frac{f(b)-f(a)}{b-a}(x-a)dx \\ &= f(a)x + \frac{f(b)-f(a)}{b-a}(\frac{x^2}{2} - ax)\Big|_a^b \\ &= \frac{b-a}{2}(f(a)+f(b)).\end{aligned}$$

Next we have the Simpsons rule which uses a quadratic interpolating polynomial that interpolates at the boundary of the interval and the midpoint of the interval. Thus we have

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b f(a) + \frac{f(b)-f(a)}{b-a}(x-a) + \frac{f[x_m,b]-f[a,x_m]}{b-a}(x-a)(x-x_m)dx \\ &= \frac{b-a}{3}\big[f(a)+4f(x_m)+f(b)\big]\end{aligned}$$

The first open rule is the midpoint rule which could be also called the rectangle rule as we just multiply the interval width by the height, which is the function evaluated at the

midpoint of the interval. Thus we have

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b f(\frac{a+b}{2})dx \\ &= (b-a)f(\frac{a+b}{2})\end{aligned}$$

The next open rule uses two points equally space apart in the interval. It is equivalent to the trapezoidal rule but you extend the interpolating polynomial to the entire interval.

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b f(x_0) + \frac{f(x_1)-f(x_0)}{x_1-x_0}(x-x_0)dx \\ &= f(x_0)x + \frac{f(x_1)-f(x_0)}{x_1-x_0}(\frac{x^2}{2} - (x_0)x)\Big|_a^b \\ &= \frac{x_1-x_0}{2}(f(x_0)+f(x_1))\end{aligned}$$

where $x_0 = a + \frac{b-a}{3}$ and $x_1 = b - \frac{b-a}{3}$. The last method is not a Newton-Cotes quadrature method.

It is a method that is constructed to attain the maximum degree of exactness. For a two point Gaussian quadrature we can and will attain a degree of exactness of 3. Thus our quadrature will integrate exactly up to a cubic function. Thus if $f(x) = a_3x^3+a_2x^2+a_1x+a_0$ then we have where the interval is $[-1, 1]$

$$\int_{-1}^1 a_3x^3 + a_2x^2 + a_1x + a_0dx = w_0f(x_0) + w_1f(x_1)$$

$$\frac{a_3}{4}x^4 + \frac{a_2}{3}x^3 + \frac{a_1}{2}x^2 + a_0x\Big|_{-1}^1 = w_0(a_3x_0^3 + a_2x_0^2 + a_1x_0 + a_0) + w_1(a_3x_1^3 + a_2x_1^2 + a_1x_1 + a_0).$$

If we group together all of the constant coefficients we get 4 equations which we can then

use to solve w_0, w_1, x_0, x_1. We get the system

$$\begin{aligned} w_0x_0^3 + w_1x_1^3 &= 0 \\ w_0x_0^2 + w_1x_1^2 &= \frac{2}{3} \\ w_0x_0 + w_1x_1 &= 0 \\ w_0x_0 + w_1x_1 &= 2. \end{aligned}$$

Solving this we get that $x_0 = -\sqrt{\frac{1}{3}}$, $x_1 = \sqrt{\frac{1}{3}}$ and $w_0 = w_1 = 1$. Now if we need to integrate on $[a, b]$ we will have $x_0 = \frac{b-a}{2}\big(-\sqrt{\frac{1}{3}}\big) + \frac{a+b}{2}$ and $x_1 = \frac{b-a}{2}\big(\sqrt{\frac{1}{3}}\big) + \frac{a+b}{2}$. So $\int_a^b f(x) \approx \frac{b-a}{2}\big(f(x_0) + f(x_1)\big)$.

3.2 Composite Rules

Next we will discuss the composite rules with a certain number of sub-intervals within the interval of integration and applying our methods to each one of the sub-intervals. The composite trapezoidal rule will add all the inner boundary function evaluations twice thus when we distribute in the half we will get

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\bigg(0.5 * f(a) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + 0.5 * f(x_n)\bigg).$$

Next the composite Simpson's rule is

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}\bigg(f(a) + 4f(x_1) + 2f(x_2) + \cdots + 4f(x_{n-1}) + f(x_n)\bigg).$$

Then the midpoint rule is

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\bigg(f(\frac{a+x_1}{2}) + f(\frac{x_2+x_1}{2}) + \cdots + f(\frac{x_{n-1}+x_{n-2}}{2}) + f(\frac{x_n+x_{n-1}}{2})\bigg).$$

The last composite Newton-Cotes rule is the two point open rule which formulates to the same as the midpoint rule except has a half in front and inside has all the inner point

evaluations inside each sub-interval. Then Gaussian rule is the same as the two point rule except the equally spaced points are replaced with the two point Gaussian points inside the sub-interval.

3.3 Errors

The errors for all these rules are just

$$|E_n(f)| = |\int_a^b \omega_n(x) f[x_0, x_1, \ldots, x_n, x]dx|.$$

This form is not very helpful, but using the Integral Mean Value Theorem and the divided difference form of the Taylor Expansion we can get a more useful form. This new form has to do with the derivative of the function at a point within the interval. So we can see that from the notes that we will get

$$\begin{aligned}
E_{trap} &= -\frac{h^3}{12} f''(\zeta_{trap}) && \text{where } h = (b-a) \\
E_{mid} &= \frac{h^3}{3} f''(\zeta_{mid}) && \text{where } h = \frac{(b-a)}{2} \\
E_{simp} &= -\frac{h^5}{90} f^{(4)}(\zeta_{simp}) && \text{where } h = \frac{(b-a)}{2} \\
E_{open2} &= \frac{3h^3}{4} f''(\zeta_{open2}) && \text{where } h = \frac{(b-a)}{3} \\
E_{gauss} &= \frac{(b-a)^5(2!^4)}{5(4!)^3} f^{(4)}(\zeta) = \frac{(b-a)^5}{4320} f^{(4)}(\zeta).
\end{aligned}$$

Our routines though use composite methods while these errors are only for non-composite intervals. A composite method has n sub-intervals and thus the error will be the sum of all the errors for each sub-intervals. So we will have a E_i and a ζ_i for each sub-interval where ζ_i is in that sub-interval. The notes give us that the trapezoidal composite error will be

$$E_{ct} = -\frac{(b-a)^3}{12n^2} f''(\zeta)$$

for some $\zeta \in [a,b]$ and $h = \frac{b-a}{n}$. To get that we need the Discrete Mean Value Theorem. Thus let us do the same for the midpoint rule, simpsons rule, and the two point open rule. So for the midpoint rule we have.

$$\begin{aligned} E_{cm} &= \frac{h^3}{3}\sum_{i=0}^{n-1} f''(\zeta_i) \\ &= \frac{h^3}{3} n f''(\zeta) \\ &= \frac{(b-a)^3}{24n^2} f''(\zeta). \end{aligned}$$

Next is the simpsons composite error. So we have

$$\begin{aligned} E_{cs} &= -\frac{h^5}{90}\sum_{i=0}^{n-1} f^{(4)}(\zeta_i) \\ &= -\frac{h^5}{90} n f^{(4)}(\zeta) \\ &= -\frac{(b-a)^5}{2880n^4} f^{(4)}(\zeta). \end{aligned}$$

Then we have the error for the composite two point open rule. So we have

$$\begin{aligned} E_{co2} &= \frac{3h^3}{4}\sum_{i=0}^{n-1} f''(\zeta_i) \\ &= \frac{3h^3}{4} n f''(\zeta) \\ &= \frac{(b-a)^3}{36n^2} f''(\zeta). \end{aligned}$$

Lastly we have the error for the Guassian Quadrature Rule. So we have

$$\begin{aligned} E_{cg} &= \frac{(b-a)^5}{4320n^5}\sum_{i=0}^{n-1} f^{(4)}(\zeta_i) \\ &= \frac{(b-a)^5}{4320n^5} n f^{(4)}(\zeta) \\ &= \frac{(b-a)^5}{4320n^4} f^{(4)}(\zeta). \end{aligned}$$

The ζ is going to be some number within the interval $[a, b]$. Thus we have our composite Newton-Cotes errors. If we solve for the number of sub-intervals, n, then we could find out how many intervals we need to get to within a certain error. We will have to bound the derivative as well. But solving for n we get

$$
\begin{aligned}
n_{trap} &= \sqrt{\frac{(b-a)^3 f''(\zeta)}{-12E}} \\
n_{mid} &= \sqrt{\frac{(b-a)^3 f''(\zeta)}{24E}} \\
n_{simp} &= \sqrt{\sqrt{\frac{-(b-a)^5 f^{(4)}(\zeta)}{2880E}}} \\
n_{open2} &= \sqrt{\frac{(b-a)^3 f''(\zeta)}{36E}} \\
n_{gauss} &= \sqrt{\sqrt{\frac{(b-a)^5 f^{(4)}(\zeta)}{4320E}}}.
\end{aligned}
$$

So now we can bound our $f^{(m)}(\zeta)$ and we will get our minimum interval size that guarantees it is within our chosen error.

4 Description of the Algorithms and Implementation

4.1 Trapezoidal Rule

First all of these routines are made for uniform sub-intervals. Let us start with the trapezoidal rule which is integrating a linear interpolating polynomial over each of the sub-intervals using the endpoints as the interpolating points. Now because this is made for a uniform sub-interval we know that each interval width dx will be the same. We also know that the single trapezoidal rule has the form $\frac{dx}{2}(f(x_1) + f(x_2))$. So a composite trapezoidal rule will be $dx(0.5 * f(x_1) + f(x_2) + f(x_2) + \cdots + f(x_{n-1}) + 0.5 * f(x_n))$. Thus we can add the two evaluations at the end points divide by two, then implement a simple for loop to add the evaluations inside the interval to the sum and lastly multiply by dx.

The refinement code first uses the simple composite trapezoidal rule to solve for the initial

sum. Then we found smallest factor we could multiply the amount of sub-intervals by and still reuse all the already evaluated points is 2. Double the intervals still yields with the old evaluations being used because the boundaries of the old sub-intervals are still boundaries for the new sub-intervals, thus they will be used for evaluations in this iteration. Now a new iteration yields new evaluations as well. Now these evaluations will be spaced apart by the dx of the last iteration. So we can implement a for loop that evaluates at $x + dx/2 + i * dx$ for $i = 0$ to the last new evaluation point. Those new evaluations added together then multiplied by this iteration's dx will be the new stuff needed to add the old sum divided by 2. The recursive formula looks like this

$$sum_{i+1} = 0.5 * (sum_i) + dx_{i+1}\left(\sum_{k=1}^{2^{i-1}*n} f_{i+1,k}\right)$$

where n is the number of initial sub-intervals, and f_i is the new evaluations of the i^{th} iteration.

4.2 Simpson's Rule

Simpsons composite rule is integrating the quadratic interpolant that interpolates the boundary points of the sub-interval and the midpoint inside the sub interval. The easy formula for Simpsons rule is $\frac{dx}{3}(f(x_0) + 4f(x_1) + f(x_2))$. Then the composite rule will be $\frac{dx}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n))$. Thus we can implement a for loop that uses has modular arithmetic to determine when to multiply by 2 or 4. Now the refinement factor for Simpsons is 2. This is because the boundaries of the sub-intervals will stay boundaries and the midpoints will become boundaries for the new the sub-intervals. Now we can see from the composite formula that the midpoints are the function evaluations multiplied by 4 and the functions evaluated at the mesh inside the global interval are the ones multiplied by 2. Thus because the midpoints become boundaries for the next iteration they go from being multiplied by 4 to 2. Thus you will need to subtract the 2 of them when you go to the next iteration if you need to. Thus I have variable that holds the amount needed to be knocked off if needed at the end of the iteration. So subtract the knock off sum, then multiply by a half and then add the sum of

the new evaluations at the new midpoints multiplied by the dx of this iteration divided by 3. That will give you the sum of this iteration. The recursive formula looks like this

$$sum_{i+1} = 0.5 * \left(sum_i - \frac{dx_i}{3} * 2 * \sum_{k=1}^{2^i * n} f_{i,k} \right) + \frac{dx_{i+1}}{3} \left(\sum_{k=1}^{2^i * n} f_{i+1,k} \right)$$

where n is the number of initial sub-intervals, and f_i is the new evaluations of the i^{th} iteration.

4.3 Midpoint Rule

The next method we implemented is the midpoint rule. This is just the constant function multiplied by the size of the sub-interval. Now because this quadrature method only uses the function evaluation at the midpoint of the sub-interval we cannot use the refinement factor of 2. This is because if the midpoints of the sub-intervals become the boundaries of the new sub-intervals we will not use the evaluations. Thus we will multiply the number of sub-intervals by 3. This will ensure that the midpoint evaluations from the last iteration are used because the midpoints of the sub-intervals will remain midpoints in the next iteration. Now evaluate the new midpoints, which are not space uniformly. They do have a pattern with the first being at $x_0 + dx_{i+1}$ then alternate adding $2 * dx_{i+1}$ then dx_{i+1} until the last new midpoint. The recursive formula is

$$sum_{i+1} = \frac{1}{3}(sum_i) + dx_{i+1} \left(\sum_{k=1}^{3^i * 2 * n} f_{i+1,k} \right)$$

where n is the number of initial sub-intervals and f_i is the new evaluations of the i^{th} iteration.

4.4 Two Point Open Rule

The last Newton-Cotes Quadrature is the two point open rule. This rule is the same as the trapezoidal rule except that instead of using the endpoints of the boundary it uses two

uniform spaced points within the sub-interval. The formula is equivalent to the trapezoidal rule except that it uses the inner points. The refining process is also the same as the trapezoidal rule with doubling the amount of sub-intervals or halfing the spacing of the sub-intervals to ensure reuse of all the previous function evaluations. The only difficult part about the implementation for this functions is that the new evaluations needed to be computed in a new iteration are not uniform across the global interval. But they do have a pattern. Starting at $x_{i,1} = a + \frac{dx_i}{3}$, then $x_{i,2} = x_{i,1} + \frac{4dx_i}{3}$, $x_{i,3} = x_{i,2} + \frac{2dx_i}{3}$, and alternates back and fourth until you reach the final new mesh $x_{i,2^i*n}$ where i is the iteration and n is the amount of initial sub-intervals. The recursive formula looks like this

$$sum_{i+1} = 0.5 * (sum_i) + dx_{i+1}\left(\sum_{k=1}^{2^i*n} f_{i+1,k}\right)$$

where n is the number of initial sub-intervals, and f_i is the new evaluations of the i^{th} iteration.

4.5 Gaussian Rule

The last routine is the Gaussian quadrature routine. This method is similar to the two point open rule but instead of using two equally spaced points in the interval we will use two points that will yield the maximum degree of exactness. The points are easily computed but are irrational thus we will take the refining factor to be 2. Now we cannot reuse the functions already evaluated, but the first attempt is usually very good. Thus the refining iteration just evaluates at the new points.

The nice thing about all of these methods is all of them are constant storage complexity routines. The time complexity is a different story because it depends on how accurate you want your approximation to be. In the next section we get a bound for the minimum amount of sub-intervals needed to be within a certain given error tolerance. If we use that n we will have a linear time complexity because we will have to evaluate some multiple of n functions and many other operations. But it still ends up being linear time complexity. So you have to be careful when talking about time complexity. In the end though using the

routines is very simple and efficient, which means we can refine the mesh if needed to get a better approximation without increasing the amount of work to be done substantially.

5 Description of the Experimental Design and Results

The five integrals given are the first five and the others are my own functions to check my routines. They are

$$\int_0^3 e^x dx = e^3 - 1 \approx 19.0855369232 \tag{1}$$

$$\int_0^{\pi/3} e^{\sin(2x)} \cos(2x) dx = 0.5(e^{\sqrt{3}/2} - 1) \approx 0.6887213376 \tag{2}$$

$$\int_{-2}^1 \tanh(x) dx = ln(\cosh(1) - ln(\cos(-2)) \approx -0.89122191687 \tag{3}$$

$$\int_0^{3.5} x \cos(2\pi x) dx = -\frac{1}{2\pi^2} \approx -0.050660591821169 \tag{4}$$

$$\int_{0.1}^{2.5} x + \frac{1}{x} dx = 3.12 + \ln(\frac{2.5}{0.1}) \approx 6.3388758248682 \tag{5}$$

$$2\int_{-1}^1 \sqrt{1 - x^2} dx = \pi \approx 3.14159265358 \tag{6}$$

$$\tag{7}$$

5.1 Bounding Sub-Intervals

Let us use the equations we derived in the section above. To do this we first must find our bounds on the derivatives for the functions above. We can see that for

$$
\begin{aligned}
|\frac{d^2}{dx^2}[e^x]|, |\frac{d^4}{dx^4}[e^x]| &\leq e^3 \\
|\frac{d^2}{dx^2}[e^{\sin(2x)}\cos(2x)]| &\leq 396.975 \\
|\frac{d^4}{dx^4}[e^{\sin(2x)}\cos(2x)]| &\leq 16.2831 \\
|\frac{d^2}{dx^2}[\tanh(x)]| &\leq 0.7698 \\
|\frac{d^4}{dx^4}[\tanh(x)]| &\leq 4.0859 \\
|\frac{d^2}{dx^2}[x\cos(2\pi x)]| &\leq 138.174 \\
|\frac{d^4}{dx^4}[x\cos(2\pi x)]| &\leq 5454.91 \\
|\frac{d^2}{dx^2}[x+\frac{1}{x}]| &\leq 2000 \\
|\frac{d^4}{dx^4}[x+\frac{1}{x}]| &\leq 2400000 \\
|\frac{d^2}{dx^2}[\sqrt{1-x^2}]| &\leq \\
|\frac{d^4}{dx^4}[\sqrt{1-x^2}]| &\leq e^3.
\end{aligned}
$$

Those are for the intervals of interest which are the integration bounds for each integral. Let us try getting within about 0.1% of the error. Thus for the first integral let us get within 0.02, 0.0007, 0.0009, 0.00005, 0.006, 0.003, for the first, second, third, fourth, fifth, and sixth functions respectively.

From our analysis above and using our chosen error tolerance and bounds on the derivatives we get

Function	e^x	$e^{\sin(2x)}\cos(2x)$	$\tanh(x)$	$x\cos(2\pi x)$	$1+\frac{1}{x}$
Error	0.02	0.0007	0.0009	0.00005	0.006
Trapezoid	48	48	44	3143	620
Midpoint	34	34	32	2222	439
Open2	28	27	26	1815	358
Simpson	4	3	5	67	58
Guassian	3	3	4	61	53

Table 1: Gives analytic calculated minimum amount of sub-intervals needed to be used in the composite method to guarantee the approximation is within the error tolerance.

Now let us use these interval sizes to in our methods and compute the errors. Then we can compare them to each other and see how accurate our bound is. These 5 tables give the errors for each function.

n	Trapezoid	Midpoint	Open 2	Simpson	Gauss
48	0.0062123354	0.0031060161	0.0020707186	0.0000001011	0.0000000674
34	0.0123808786	0.0061898369	0.0041267216	0.0000004016	0.0000002677
28	0.0182543556	0.0091258683	0.0060842678	0.0000008730	0.0000005820
4	0.8863581155	0.4400857457	0.2942283883	0.0020622080	0.0013732723
3	1.5645694658	0.7726326215	0.5176974193	0.0064347410	0.0042813284

Table 2: Errors for methods at sub-interval sizes specified for function (1).

n	Trapezoid	Midpoint	Open 2	Simpson	Gauss
48	0.0001955205	0.0000977651	0.0000651754	0.0000000032	0.0000000021
34	0.0003897132	0.0001948757	0.0001299120	0.0000000127	0.0000000085
27	0.0006180287	0.0003090624	0.0002060286	0.0000000320	0.0000000213
3	0.0508414758	0.0257124994	0.0170635851	0.0001945077	0.0001289233

Table 3: Errors for methods at sub-interval sizes specified for function (2).

n	Trapezoid	Midpoint	Open 2	Simpson	Gauss
44	0.0001353157	0.0000676537	0.0000451036	0.0000000028	0.0000000019
32	0.0002558125	0.0001278913	0.0000852649	0.0000000099	0.0000000066
26	0.0003874718	0.0001937016	0.0001291437	0.0000000229	0.0000000153
5	0.0104030238	0.0051717699	0.0034561437	0.0000198280	0.0000133647
4	0.0161977817	0.0080319146	0.0053728781	0.0000446508	0.0000298724

Table 4: Errors for methods at sub-interval sizes specified for function (3).

n	Trapezoid	Midpoint	Open 2	Simpson	Gauss
3143	2.06679373E-7	1.03339878E-7	6.8893201E-8	1.27E-13	8.4E-14
2222	4.13522224E-7	2.06761874E-7	1.37841042E-7	5.08E-13	337E-13
1815	6.19776018E-7	3.09889715E-7	2.06592681E-7	1.138E-12	7.59E-13
67	4.572759056E-4	2.295616008E-4	1.527899832E-4	6.157653E-7	4.106564E-7
61	5.522717E-4	2.774816E-4	1.846218E-4	8.972E-7	5.984E-7

Table 5: Errors for methods at sub-interval sizes specified for function (4).

n	Trapezoid	Midpoint	Open 2	Simpson	Gauss
620	0.00012465143	6.2318708E-5	4.15477094E-5	4.67358E-9	3.11553E-9
439	0.00024859187	1.24268070E-4	8.28529467E-5	1.857691E-8	1.238315E-8
358	0.00037375295	1.86813529E-4	1.24559441E-4	4.196748E-8	2.797335E-8
58	0.0140187041	6.92569816E-3	4.63951992E-3	5.5769261E-5	3.6950859E-5
53	0.0167388718	8.2513595E-3	5.5324241E-3	7.87176E-5	5.20981E-5.

Table 6: Errors for methods at sub-interval sizes specified for function (5).

These 5 tables give us the errors at the function for the specified number of sub-intervals. Now we need to pay close attention to the errors on the diagonals of the tables. Those are the errors that we analytically showed that should be within our tolerance of error chosen for that function. Now we can see that sometimes the number of sub-intervals solved using our equations derived can be a bit conservative. As we can see in most of these that the lower n values calculated for the midpoint and two point open methods still provide a good enough approximation for the trapezoidal rule with those amount of sub-intervals. Thus we do get a much better accuracy, than expected.

5.2 Sub-intervals from Refining

Now we can look at the refinement methods and see how they compare to the results we just got. So we are going to start with the non-composite method and refine by how how we discussed in section 4. Remember above that we will refine by doubling the amount of sub-intervals for the trapezoid rule, Simpson's rule, and the two point open rule and then we will triple the amount of sub-intervals for the midpoint rule. We do this to reuse all of the function evaluations evaluated in the last iteration. Thus we can see that

Iteration	Sub-Intervals	Error
	Trapezoidal Rule	
0	1	1.5645694658
1	2	0.3959684222
2	4	0.0993004463
3	8	0.0248444903
4	16	0.0062123354
	Midpoint Rule	
0	1	0.7726326215
1	3	0.0880734551
2	9	0.0098141290
	Open Rule 2	
0	1	0.5176974193
1	2	0.1317456456
2	4	0.0330848340
3	8	0.0082805384
	Simpson Rule	
0	1	0.0064347410
	Gauss-Legendre Rule	
0	1	0.0042813284

Table 7: Errors for methods with refinements for function (1).

Iteration	Sub-Intervals	Error
	Trapezoidal Rule	
0	1	0.0508414758
1	2	0.0125644882
2	4	0.0031313963
3	8	0.0007822356
4	16	0.0001955205
	Midpoint Rule	
0	1	0.0257124994
1	3	0.0027895868
2	9	0.0003090624
	Open Rule 2	
0	1	0.0170635851
1	2	0.0041958579
2	4	0.0010442837
3	8	0.0002607756
	Simpson Rule	
0	1	0.0001945077
	Gauss-Legendre Rule	
0	1	0.0001289233

Table 8: Errors for methods with refinements for function (2).

Iteration	Sub-Intervals	Error
	Trapezoidal Rule	
0	1	0.0284110489
1	2	0.0072423560
2	4	0.0018173418
3	8	0.0004547224
	Midpoint Rule	
0	1	0.0139263368
1	3	0.0016114318
2	9	0.0001796245
	Open Rule 2	
0	1	0.0093637183
1	2	0.0024088488
2	4	0.0006054759
	Simpson Rule	
0	1	0.0001861251
	Gauss-Legendre Rule	
0	1	0.0001265998

Table 9: Errors for methods with refinements for function (3).

Iteration	Sub-Intervals	Error
	Trapezoidal Rule	
0	1	2.6715616304
1	2	0.1316937197
2	4	0.0169181795
3	8	0.0036984686
4	16	0.0008955181
5	32	0.0002221175
6	64	0.0000554201
7	128	0.0000138482
	Midpoint Rule	
0	1	2.4081741910
1	3	0.0213718223
2	9	0.0014840304
3	27	0.0001565979
4	81	0.0000173003
	Open Rule 2	
0	1	1.2833197533
1	2	0.0556451957
2	4	0.0060512990
3	8	0.0012557085
4	16	0.0002998967
5	32	0.0000741255
6	64	0.0000184788
	Simpson Rule	
0	1	0.7149289172
1	2	0.0213403338
2	4	0.0007081017
3	8	0.0000387987
	Gauss-Legendre Rule	
0	1	0.5002597898
1	2	0.0148029415
2	4	0.0004772046
3	8	0.0000259372

Table 10: Errors for methods with refinements for function (4).

Iteration	Sub-Intervals	Error
	Trapezoidal Rule	
0	1	2.3006012993
1	2	0.8390312354
2	4	0.2698298524
3	8	0.0770823529
4	16	0.0203295135
5	32	0.0051683594
	Midpoint Rule	
0	1	0.6225388285
1	3	0.1763390457
2	9	0.0294555365
3	27	0.0035983147
	Open Rule 2	
0	1	0.4919665682
1	2	0.2226028613
2	4	0.0815411076
3	8	0.0248214679
4	16	0.0067076111
5	32	0.0017181139
	Simpson Rule	
0	1	0.3518412141
1	2	0.0800960581
2	4	0.0128331864
3	8	0.0014119003
	Gauss-Legendre Rule	
0	1	0.1639237236
1	2	0.0446422293
2	4	0.0079229194
3	8	0.0009152506

Table 11: Errors for methods with refinements for function (5).

Thus we can see from the tables with the iterations that the bounds we calculated for the number of sub-intervals to guaruntee the error within our choice is much higher than needed. In some Cases an order of magnitude higher.

Function	e^x	$e^{\sin(2x)}\cos(2x)$	$\tanh(x)$	$x\cos(2\pi x)$	$1+\frac{1}{x}$
Error	0.02	0.0007	0.0009	0.00005	0.006
Trapezoid	16	16	8	128	32
Midpoint	9	9	9	81	27
Open2	8	8	4	64	32
Simpson	1	1	1	8	8
Guassian	1	1	1	8	8

Table 12: Gives the amount of sub-intervals needed to be used in the composite method to guarantee the approximation is within the error tolerance.

Function	e^x	$e^{\sin(2x)}\cos(2x)$	$\tanh(x)$	$x\cos(2\pi x)$	$1+\frac{1}{x}$
Error	0.02	0.0007	0.0009	0.00005	0.006
Trapezoid	48	48	44	3143	620
Midpoint	34	34	32	2222	439
Open2	28	27	26	1815	358
Simpson	4	3	5	67	58
Guassian	3	3	4	61	53

Table 1: Gives analytic calculated minimum amount of sub-intervals needed to be used in the composite method to guarantee the approximation is within the error tolerance.

Here we can see that the accuracy the a priori bound gives us is much higher than we asked for because it is so conservative. Thus we can say that the a priori bound gives definitely guarantees us a good enough approximation if we want it within a certain error. But it may go overboard and that could take your code longer to run do to the excessive amount of function evaluations. Thus in some cases it may be better to use a refinement method to find the right amount of sub-intervals to use. It could save you a lot of time in function evaluations.

6 Conclusions

Our results illustrates how the five different quadrature methods approximate the integral with only knowing a few points. We started out with coding the methods and found with creative ways we can refine the intervals with making a complete reuse of function evaluations used in the last interval. The creative way only uses constant storage and the time-complexity is slow as well. That was nice to see that adding more sub-intervals will not add much in storage or time as we may need to add more sub-intervals to get a better approximation.

After we did an analysis on the error of these functions and found a bound for the minimum number of sub-intervals needed to have the approximation within a certain error tolerance. The bound was a function of maximum error of the user's choosing, the global interval size and the maximum magnitude of the derivative of the function in the interval being integrated on. The function is very nice, but for larger interpolating polynomials or methods with higher degree of exactness we will need the functions derivatives to be more continuous on the interval. But for our uses they worked very well because we were only using low degree of exactness methods.

However after testing our methods with the amount of sub-intervals calculated we found that the approximations given were far more accurate than we had asked for. The bound in some cases can be very much overkill. If you function is simple enough it is not a problem. But with functions that have hundreds of calculations in them it can become a problem. We then checked our to see how many iterations it took with our refining methods to get within the error we desired. As we suspected this gave a much lower amount of sub-intervals to be with the tolerance. Seeing this we came up with the conclusion that whether to use a refining method or to use the bound we found really depends on the function trying to be integrated. If the function is very complex the refining method might be the better bet as you will most-likely have much less function evaluations. But if the function is simple, the a priori bound on the number of sub-intervals can give you a very accurate result.

7 Program Files

There are eleven routines in the main.cpp. They are called exacEval, trapRule, simpsonRule, midpoint, openRule2, and guassLegendreRule. There are two functions each with those names. One function being the regular composite function and the other with refining. The regular composite function accepts the number of mesh points as an argument, the array pointer of the x values of the mesh, and the function. The function is denoted as an integer and will have to be added to the top in the exactEval function which takes in an x value and an integer corresponding to the function you want that point to be evaluated in. Then for the refining methods you will have those same arguments as the composite plus two more arguments being the maximum error tolerance for the approximation of the integral and then the actual solution of the integral. All these functions return the approximate solution to the integral. Now the refining methods can easily be changed to having the condition be the length from mesh to mesh. As in some cases we may not have the exact solution to the integral or there does not exist an analytical solutin to the integral.