A Comparison of Two-Dimensional Numerical Integration Techniques in C

Atharv Thakur

(Dated: May 27, 2024)

This paper covers numerical integration in the case of two dimensions with C implementations of the Midpoint Rule, Trapezoidal Rule, Simpson's rule, two versions of Monte Carlo methods, and a Quasi-Monte Carlo method using the Halton sequence. It aims to identify the optimal numerical integration technique for certain simple physics problems — such as calculating the electric flux through a plane or calculating the area under a three-dimensional gaussian distribution — for use in an educational physics simulator application. Expectedly, the more standard approaches prove generally superior to the pseudorandom and quasi-random approaches, with their own differences characterized primarily by modest tradeoffs in time in exchange for accuracy. Complete code, data, and graphs can be viewed on the official Github page for this paper.

I. Introduction

Numerical integration techniques are tools used to approximate the results of integrals without performing any actual integration. Their use of exclusively arithmetic operations often makes them preferable to direct integration, as computers can easily make use of them, enabling efficient computation of nonelementary integrals. A variety of numerical integration techniques exist, and their utility is determined primarily by their deviation from the accepted result, i.e. error, and their execution time.

Runtime and error margins depend not only on the numerical integration method being used, but also on the specific function being integrated, as well as integration bounds. As a result, certain methods perform better under certain types of conditions. In order to clarify this notion, consider the case of Monte Carlo methods. Unlike deterministic quadrature methods, which suffer from Bellman's curse of dimensionality, the random nature of Monte Carlo methods allows them to efficiently approximate integrals of higher dimensions. However, when computing integrals of lower dimensions, these strengths are not as pronounced, and thus deterministic methods are preferred for their accuracy.

Factors unique to a function also affect the performance and accuracy of numerical integration. Approximations of discontinuous functions and functions with high magnitude derivatives, for instance, tend to result in greater error than approximations of well-behaved functions with relatively lesser derivative magnitudes. Collectively, these factors make it difficult to identify a "superior" numerical integration technique for use in a certain scenario, unless sufficient details are known beforehand..

Fortunately, this paper addresses a specific use case: computing common two-dimensional integrals for use in an educational physics simulation program. Specifically, it focuses on integrating the two-dimensional Gaussian function, as well as the electric field from a point charge across a plane. Given our conditions, we are able to select and test a few common quadrature techniques for use in this program. Ultimately, we determine that the textbook standard techniques clearly outperform the pseudorandom and quasi-random techniques.

However, there exist only modest differences between the standard techniques themselves, which in turn necessitates further analysis.

II. Methods

This paper tests the Midpoint Rule, Trapezoidal Rule, Simpson's rule, two versions of Monte Carlo methods, and a Quasi-Monte Carlo method using the Halton sequence for use in approximating two-dimensional integrals of the Gaussian function and the electric field from a point charge over a plane. In order to do this, we must first programmatically implement these methods in a manner that can be conveniently called from our C# based simulator application. Due to its exceptional performance and portability, the C language was chosen for our implementations. Next, we must determine the semantics of each algorithm. Although the Midpoint Rule extends effortlessly to two dimensions, and extending Monte Carlo or Quasi-Monte Carlo methods is similarly trivial, extending the Trapezoidal Rule and Simpson's Rule to two dimensions requires slightly more conceptual effort. Fortunately, their formulae prove straightforward to implement once derived [1].

The standard quadrature methods — the Midpoint Rule, Trapezoidal Rule, and Simpson's Rule — all have time complexities of $O(n^2)$ due to breaking up the integration region into n^2 two-dimensional shapes of equal area, and then weighting the contributions of those shapes to the integral value depending on their location. The selected Monte Carlo methods and Quasi-Monte Carlo method, meanwhile, have time complexities of O(n), due to collecting n samples of size 100 and 50 respectively in order to find an average function value for their approximation.

Monte Carlo methods draw on values selected from a probability distribution in order to estimate an average value for a function, then subsequently multiply that value by the total area of the integration region in order to approximate an integral value. The two Monte Carlo methods used in this paper select pseudorandom numbers from a Gaussian distribution and a uniform distribution respectively. A Quasi-Monte Carlo method, meanwhile, uses a deterministic sequence of numbers that mimics a pseudorandom sequence, but offers greater accuracy due to guaranteeing an even spread of point selections. This paper uses the Halton sequence, which tends to be the preferred sequence for Quasi-Monte Carlo integrations over a small number of dimensions [2].

After selecting and implementing our methods to compare, we must then create equal grounds to compare them on. Let the the integral

$$\iint_D f_n(x, y)\, dA \qquad (1)$$

represent the general form of an integral to be approximated with these methods. The notation f_n in Eq. (1) refers to the nth function we will be testing, where n=1 refers to the function for the magnitude of the electric field from a point charge above the rectangular region D, and n=2 refers to the two-dimensional Gaussian function. The region D takes on the forms $[0, a] \times [0, a]$, $[0, a] \times [0, b]$, $[-a, a] \times [0, a]$, $[-a, a] \times [0, b]$, $[-a, a] \times [-a, a]$, and $[-a, a] \times [-b, b]$, where $a \in \{10, 100, 1500\}$, and $b \in \{15, 150, 1500\}$. These bounds cover regions across a single quadrant, two quadrants, and four quadrants, alongside a variety of sizes. Performing approximations of Eq. (1) across all the

aforementioned bounds results in a data set that accounts for many of the potential variations in use that would occur in our physics simulator application. Thus, by constructing this data set for each numerical integration method with samples $\epsilon \{256, 512, 1024, 2048\}$, we can make reasonable initial comparisons of their viability in our application by analyzing their runtimes and error percentages. To account for its relatively smaller O(n) time complexity, the Quasi-Monte Carlo method uniquely used samples $\epsilon \{256^2, 512^2, 1024^2, 2048^2\}$. However, all graphs that this method appears in reflects the square root of its samples instead in order to allow for consistent graphical comparisons to the other methods. Although the Monte Carlo methods also have an O(n) time complexity, their operations are more expensive to perform, and thus their data was gathered with samples $\epsilon \{256, 512, 1024, 2048\}$ as well in order to allow them to compete fairly with the runtimes of the other methods..

After confirming the general superiority of the standard quadrature techniques for use in this lower dimensional scenario, we then collect more detailed data on those methods in order to facilitate deeper analysis. Allowing $a = 100$ and $b = 150$, we use Eq. (1) with all forms of D to collect information on the runtime and error percentage of the standard quadrature methods across $\{1024 + 32k \mid k \epsilon \{0, 1, \dots 32\}\}$ samples. In this data gathering procedure, we also run each iteration of Eq. (1) thirty times and use the average runtime in order to present more accurate estimations.

All data gathering was performed on the lowest tier of Github codespace available freely to the public as of the writing of this paper. This system was chosen intentionally so as to provide easily replicable results. More precisely, all code was executed on an Ubuntu 20.04.2 virtual machine provided by Github with a single core of an AMD EPYC 7763 processor. Furthermore, all code was compiled by the GNU Compiler Collection C compiler version 9.4.0 with optimizations set at an O_3 level.

III. Results

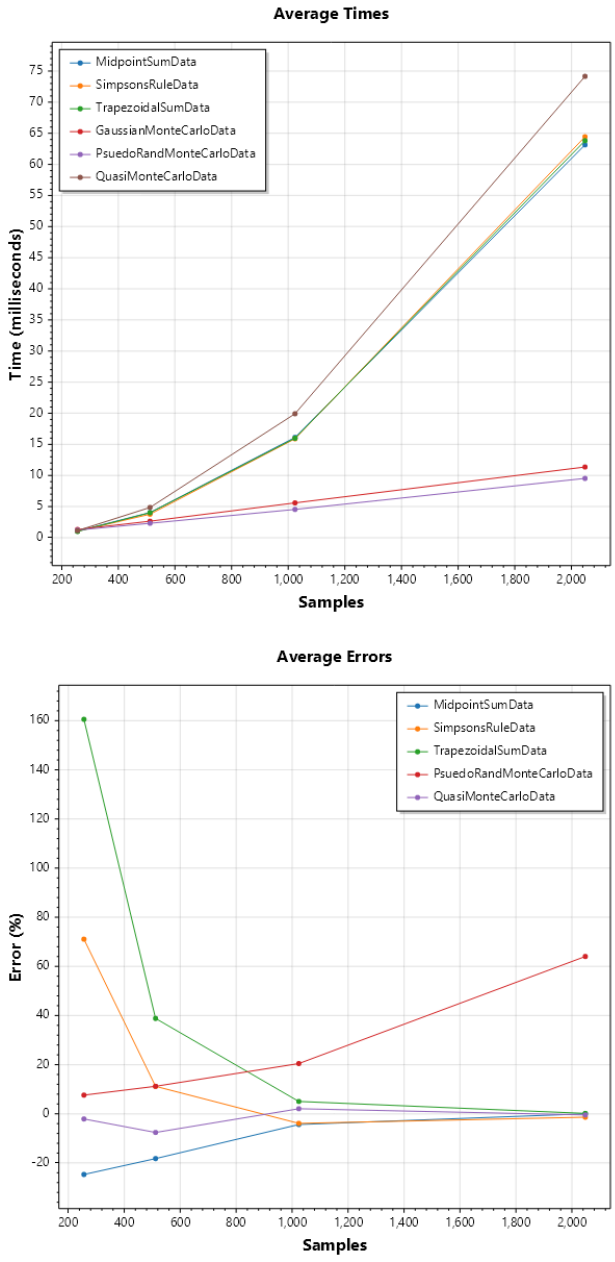
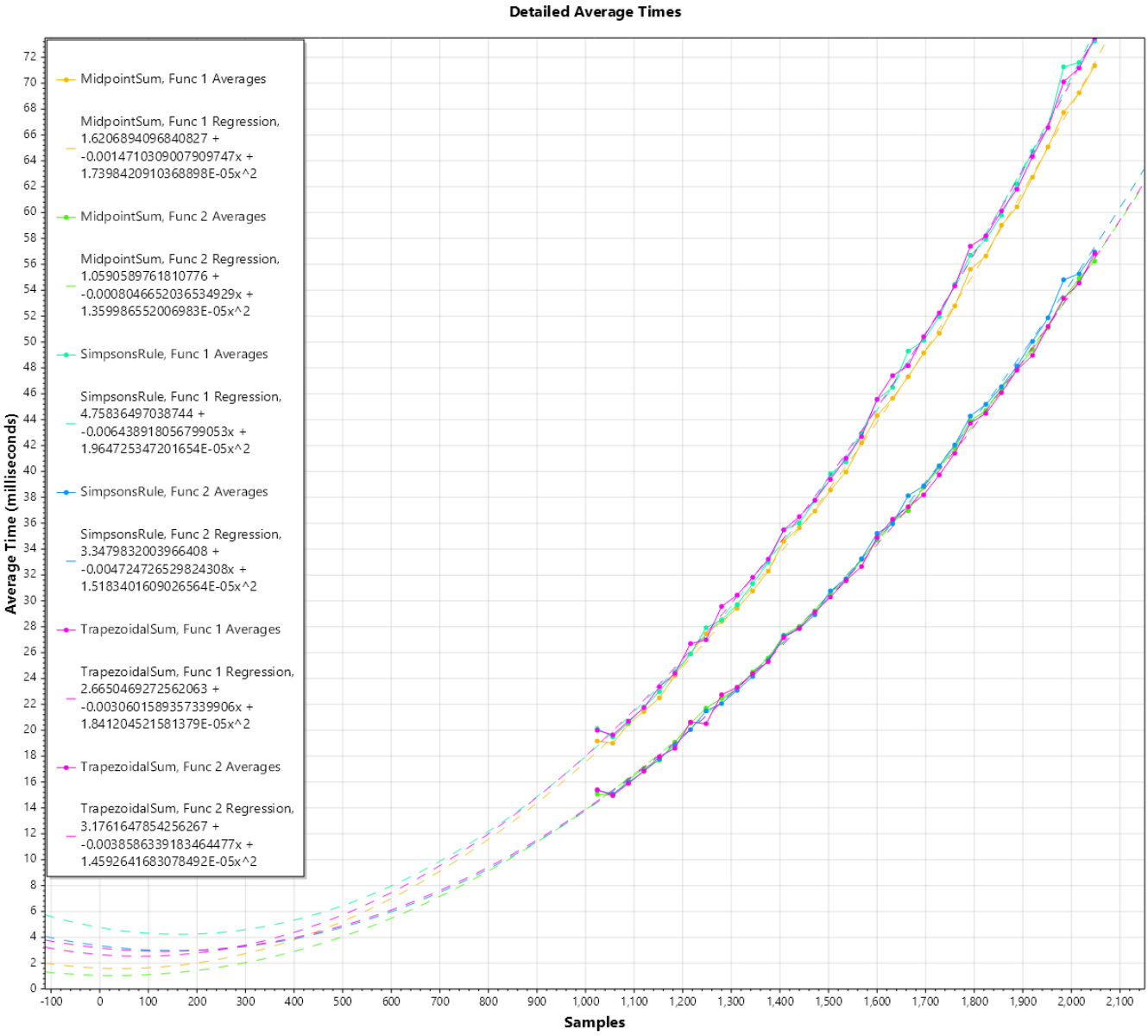


FIG. 1. Graphs of the average runtime and error percentage of each numerical integration method across all bounds versus the amount of samples used.


Detailed Average Times
Average Time (milliseconds)
Samples
MidpointSum, Func 1 Averages
MidpointSum, Func 1 Regression, 1.6206894096840827 + -0.0014710309007909747x + 1.7398420910368898E-05x^2
MidpointSum, Func 2 Averages
MidpointSum, Func 2 Regression, 1.0590589761810776 + -0.0008046652036534929x + 1.359986552006983E-05x^2
SimpsonsRule, Func 1 Averages
SimpsonsRule, Func 1 Regression, 4.75836497038744 + -0.006438918056799053x + 1.964725347201654E-05x^2
SimpsonsRule, Func 2 Averages
SimpsonsRule, Func 2 Regression, 3.3479832003966408 + -0.004724726529824308x + 1.5183401609026564E-05x^2
TrapezoidalSum, Func 1 Averages
TrapezoidalSum, Func 1 Regression, 2.6650469272562063 + -0.0030601589357339906x + 1.841204521581379E-05x^2
TrapezoidalSum, Func 2 Averages
TrapezoidalSum, Func 2 Regression, 3.1761647854256267 + -0.0038586339183464477x + 1.4592641683078492E-05x^2

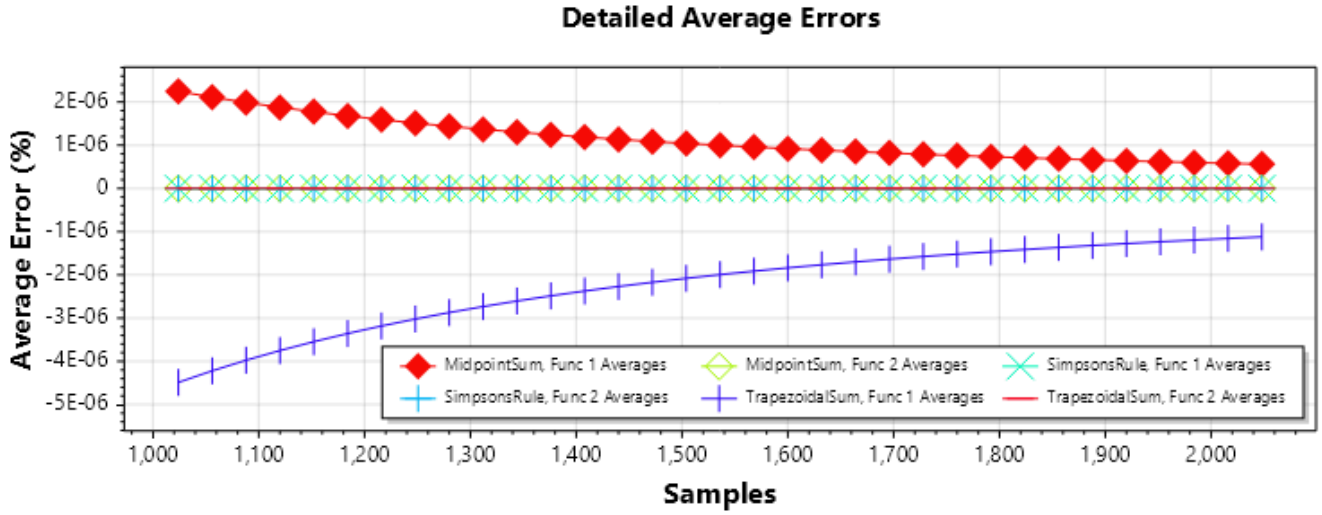

Detailed Average Errors
Average Error (%)
Samples
MidpointSum, Func 1 Averages
MidpointSum, Func 2 Averages
SimpsonsRule, Func 1 Averages
SimpsonsRule, Func 2 Averages
TrapezoidalSum, Func 1 Averages
TrapezoidalSum, Func 2 Averages

FIG. 2. Graphs of the average runtimes and error percentages from the more detailed set of data for the standard quadrature methods. A quadratic regression curve accompanies the runtime graph, as all of these methods are of time complexity $O(n^2)$ and have visually similar results, whereas no regression curve is necessary for the error graph since all significant differences are clearly apparent.

IV. Analysis

As no major outliers exist in the data sets, using the average across every combination of bounds proves to be an effective method of judging general performance and accuracy. Furthermore, performing regression analysis on the detailed data allows us to easily compare each method quantitatively, for it is otherwise difficult to draw definite conclusions from the appearance of the graphs. Before moving onto closer analysis of the standard quadrature methods, however, we must first justify the elimination of the Monte Carlo methods and Quasi-Monte Carlo method from potential use in our physics simulator.

The Monte Carlo method which drew from the Gaussian distribution performs reasonably well on low sample counts, however begins to drastically overestimate on higher sample counts. Due to reaching error margins of far greater orders of magnitude than the other methods, it even had to be removed from the average errors graph to allow the other methods to remain visible. For this reason, we choose to disregard it for the rest of this study.

The Monte Carlo method drawing from a uniform probability distribution suffered a similar fate, although not quite as dramatic. Despite possessing far smaller runtimes than the deterministic methods, its error margins actually increased as more samples were taken due to the nature of the functions being tested. Thus, it remained competitive at only around 512 samples or less, a count which fails to provide an adequate approximation for our purposes with any of the tested numerical integration methods. Consequently, we must eliminate this method as well.

The Quasi-Monte Carlo method proved a better competitor than its pseudorandom counterparts, however nonetheless fell short of the mark. Despite purporting the lowest average error margin at 1024 samples, the error margins of the deterministic techniques roughly equalize as sample counts continue to increase. More significantly, the Quasi-Monte Carlo method consistently had the longest runtime out of any method, even when disregarding the time required to precompute the Hatlon sequence for a given number of samples. Although the Quasi-Monte Carlo method proves promising with its low error margins, it provides diminishing returns with the greater sample counts necessary for this use case. In addition, the Quasi-Monte Carlo method costs more time and memory than the other deterministic methods, and could even result in memory leaks depending on implementation, ultimately leading to its elimination.

With those competitors out of the way, we can now proceed to a deeper analysis of the remaining three methods, those that have been thus far referred to as the standard methods. As seen in the first graph from Fig. 2, the Midpoint Rule typically runs the fastest for function 1, with both the Trapezoidal Rule and Simpson's Rule falling behind it at roughly equivalent runtimes. Selecting the fastest method for function 2 proves more difficult, as both the data and regression curves are nearly unified for much of the data set. However, a similar pattern to function 1

emerges, as the Midpoint Rule appears to ever so slightly edge out the speed of the Trapezoidal Rule, and Simpson's Rule falls clearly behind both others once again. This pattern of runtimes is consistent with our expectations in light of the implementational differences of each method, as the Midpoint Rule requires the least operations, the Trapezoidal Rule requires a few more, and Simpson's Rule requires by far the most. When we move to consider errors, however, the existing hierarchy undergoes drastic change, and not in the way one may initially expect. The Trapezoidal Rule consistently suffers from the greatest error magnitude for function 1, and always underestimates the accepted value. The Midpoint Rule, meanwhile, always overestimates the accepted value for function 1, albeit with less magnitude of error than the Trapezoidal Rule. In stark contrast to both of these methods, Simpson's Rule appears to have an insignificant error margin for function 1, even at the smallest samples count. Exact calculations put the numerical error from Simpson's Rule on function 1 at an order of magnitude of 10^{-12}, which is entirely negligible considering that the accepted values used in calculations are only known to an order of 10^{-11}. With this in mind, Simpson's Rule clearly emerges as the victor from an errors perspective, as all of the standard methods boast negligible error for function 2 when applied with the established conditions. At this point, it is important to note the severity of differences between the rankings for each metric. Although the Midpoint Rule consistently outperformed Simpson's Rule in speed, it did so only by a matter of a few milliseconds at a time. This difference would certainly grow more pronounced across higher sample counts and additional calls, however considering that scenario is unnecessary for our purposes. We successfully reach negligible error margins for both functions using Simpson's Rule with the tested sample counts, and thus the use of higher sample counts is unnecessary. Furthermore, the physics simulator will not need to successively call a numerical integration method more than a few times, meaning that the runtime difference will likely be unnoticeable. Furthermore, if we were to use either of the other standard methods, they would require significantly higher sample counts to achieve similar error margins to Simpson's Rule on function 1, which would in turn cause them to perform slower than Simpon's Rule when actually implemented into the simulator. In consideration of all of these factors, we find that Simpson's Rule is the optimal numerical integration technique for our purposes.

V. Conclusion

This paper primarily concerns itself with identifying the best numerical integration technique for use in an educational physics simulator, a very specific use case. For this reason, we must take care to note that every numerical integration technique tested in this paper has its own purposes, even if it proved suboptimal within the scope of our research. The Monte Carlo methods and Quasi-Monte Carlo method, for instance, are ideal for higher dimensional scenarios where standard techniques become grossly inefficient. However, considering that this paper tested only two-dimensional scenarios, one could easily predict the relative inefficacy of these methods. Additionally, their ineffectiveness is partly due to implementation. Drawing from a Gaussian probability distribution caused massive overestimations with our tested functions, however it could be optimal in other scenarios. Furthermore, the Quasi-Monte Carlo method only had an O(n) time complexity, unlike the other deterministic methods, and thus there was more room for experimentation with sample counts than this paper chose to explore due to concerns over comparison. As for the standard methods, although

Simpson's Rule proved optimal for use in the simulator due to its negligible error margins across all sample counts 1024 and greater, this precision is only preferred due to its minimal difference in execution time from the other methods. If a program required computing many different integrals at once, or if an error margin on the order 10^{-6} was equally negligible, then either of the other methods would be preferable. Finally, the specific functions and bounds used played a pivotal role in determining the optimal quadrature method. Both functions are monotonically decreasing when converted to functions of distance from the origin, and since the integration region is a simple plane, the integral is symmetric across axes. These facts would not hold true for certain more complicated functions or boundaries. In addition, both functions are mathematically well behaved, which makes them well suited for approximation through standard quadrature techniques. With the culmination of these factors and limitations in mind, however, we can clearly see that Simpson's Rule proves the superior numerical integration technique for our purposes. The simplicity of our integrals allows the standard numerical integration techniques to shine, which includes Simpson's Rule. Then, across all the collected data for 1024 samples and up, Simpson's Rule yields approximations with by far the lowest error margins in exchange for a minor increase in computational time. Furthermore, this increased cost is offset by the fact that Simpson's Rule requires far less samples to yield a satisfactory result than the other tested methods, implying that it may also be the most time efficient method considering the limited use cases we are concerned with. Of course, there remains room for improvement in designing the optimal quadrature method for all iterations of Eq. (1), such as taking advantage of the symmetric nature of many of its boundaries. However, when selecting from common existing methods of numerical integration, as we have done in this paper, it is clear that Simpson's Rule is the superior approximation technique for generic use in our simulator.

---

[1] D. Keffer, "Numerical Techniques for the Evaluation of Multi-Dimensional Integral Equations," in *ChE 505*. University of Tennessee, 1999. [Online]. Available: http://utkstair.org/clausius/docs/che505/pdf/IE_eval_N-Dints.pdf

[2] W. J. Morokoff and R. E. Caflisch, "Quasi-Monte Carlo Integration," *Journal of Computational Physics*, vol. 122, no. 2, pp. 218–230, Dec. 1995, doi: https://doi.org/10.1006/jcph.1995.1209.