Solving quantum systems numerically

Samuel Lashwood, 01190892

December 11, 2018

Abstract

A project was undertaken to examine various methods of computational numerical integration in order to evaluate a position probability of a quantum wave-packet. Newton-Coates integration was used in the extended trapezium and Simpson's rules, which were found to be very computationally fast and precise. Monte Carlo methods were also employed, which, though slower, allowed for greater accuracy in evaluating the given integral. Various methods of validation were employed and differing implementations of Monte-Carlo methods were compared.

1 Introduction and aims

The analysis in the field of Quantum Mechanics often requires integrals to be performed in order to obtain meaningful and physical results. For the case in which an integral is not analytic, computational methods must be employed. This project investigated different methods of carrying out such an integral in order to obtain a probability of a wave-packet represented particle being in a certain spatial interval. The project aimed to examine and compare Newton-Coates (NC) Integration and Monte-Carlo methods so that the best approach in terms of computational speed, precision and accuracy could be used to evaluate quantum systems. The project also aimed to examine the implementations of the employed methods in terms of efficiency and validity. The given integral that the various methods evaluated was

$$\int_0^2 |\frac{1}{\sqrt[4]{\pi}}e^{ia(z)}e^{-\frac{z^2}{2}}|^2 dz = \int_0^2 \frac{1}{\sqrt{\pi}}e^{-\frac{z^2}{2}}dz. \tag{1}$$

2 Computational Method

Initially, the extended trapezium rule was used to approximate the integral. This algorithm works by considering the integral of a function as finding the area of the function and splitting it into successively smaller rectangular sections such that

$$\int_a^b f(x)dx \approx \frac{b-a}{2N}(f(a) + 2f(a+\frac{b-a}{N}) + 2f(a+2\frac{b-a}{N}) + ... + f(b)). \tag{2}$$

Successive estimations of the value of the integral can be obtained by increasing N. In order to implement the extended trapezium rule such that a result is given to a specified accuracy, an iterative 'while' loop was used in order to continue approximating the given integral whilst the relative error between the previous and current evaluations was greater than the specified accuracy. In successive evaluations, terms were added to the previous evaluation to obtain the next in order to minimise the required number of computational operations. Due to the fact that the extended Simpson's rule basically performs quadratic interpolation for sets of points such that the errors from lower order interpolants (such as those in the extended trapezium rule) will cancel, the extended Simpson's rule can be built from successive trapezium estimations with the correct weightings to cancel those errors. This is summarised by the compact formula

$$S_i = \frac{4}{3}T_{i+1} - \frac{1}{3}T_i, \tag{3}$$

which allows us to easily tweak the trapezium rule implementation to obtain that of Simpson's rule. For the purposes of the project, the extended trapezium rule implementation was duplicated and modified to give two separate functions to perform both integration methods.

Both of these functions took the integrand, limits and required accuracy as arguments and output the integral value and the number of evaluations required to achieve this. Next, a Monte Carlo integration routine was written which also performed a given integral to a specified accuracy. This method was based on the fact that an integral, I, can be expressed in

$$I = \int_V dx\frac{f(x)}{w(x)}w(x) \text{ so that } I \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{w(x_i)} \tag{4}$$

where x_i are drawn from a random distribution according the weight function, termed the sample distribution. This was carried out first using a constant weight function (flat importance sampling) and then using a linear weight function with the formula

$$w(x) = 0.98 - 0.48x. \tag{5}$$

Using the transformation method, a sample distribution was obtained by passing a uniform random distribution, (on the interval [0,1]), through the function

$$x_i = \frac{2\times 0.98 - \sqrt{4\times 0.98^2 - 8\times 0.48\times x}}{2\times 0.48} \tag{6}$$

using the transformation method on (5) and choosing the subtractive root to ensure that the deviate x_is are in the integral domain.

This method was implemented by comparing successive estimates of the integral, obtaining those estimates by adding more random numbers to the list of numbers at which the function was evaluated. This was done in order to obtain the value that the estimate converges to, within the specified input error. It was chosen that 100 random numbers would be added in each iteration, as this quantity is likely to sample the domain space well (according to a weighting function if provided) whilst still minimisng possible bias, see Results and Discussion.

The error associated with these estimations were computed as

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(\frac{f(x_i)}{w(x_i)} - I)^2. \tag{7}$$

This function returned the estimate, the number of function evaluations, the set of x coordinates considered (for validation) and the error on the estimate.

Adaptive step size was also implemented using the Metropolis algorithm [1], where a small step δx is proposed from position and then accepted with probability $P(x+\delta x)/P(x)$ (or 1 if that quantity is larger than 1) according to the weighting PDF 'P'. The Monte Carlo integration continues normally with these successive new steps.This method was implemented to run taking a certain number of steps, and then the whole process was repeated to obtain an average value for the integral.

List comprehensions were used where applicable to compactify the code and function evaluations were minimised by storing outputs as variables when needed multiple times in order to decrease the number of computational operations needed. For investigative purposes, the original Monte-Carlo method (with flat importance sampling) was also implemented as a rejection method: considering a ratio of the number of points generated under the integrand curve to the total number, after generating random coordinates on a set interval.

3 Validation

The NC integration functions were validated by first examining the successive estimates to check that they were converging using simple print statements and then by comparing the estimates obtained with the calculated value given by the integration function in the *Scipy* module. Once these were established as accurate estimates, they were used for comparison with the other methods. This value was found to be 0.4976611.

The Monte Carlo routines were validated in multiple ways. Firstly, the resulting estimations were compared to those of the NC routines to be ensured as correct. Secondly, when building the routine, print statements were used to keep track of the values considered at each step. Furthermore, for the importance sampling routine, a histogram of the considered random deviates was produced such that the correct distribution could be ensured (see figure 1).

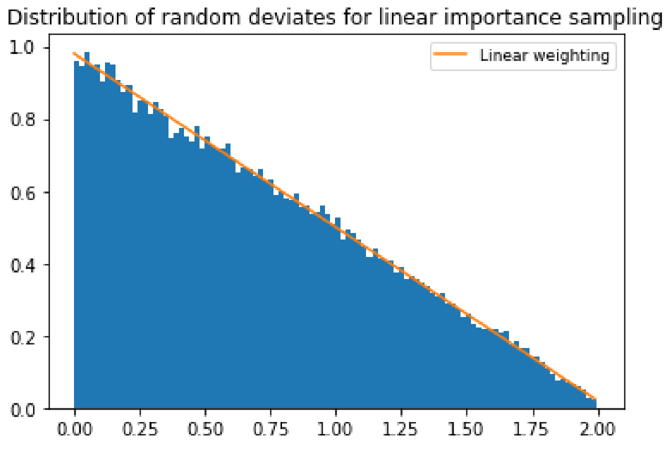

Figure 1: The normalised histogram of the random numbers distributed with PDF in equation 5 using the transformation method, showing the expected linear distribution.

The same method was also employed with the Metropolis algorithm implementation to validate the Metropolis algorithm, which took more time to implement. Though the distribution of random numbers considered was found to be distributed as expected (see Figure 2), the estimate of the integral was found to strongly depend on the size of the small step taken during the random walk. This caused some issues during implementation as unrealistic step sizes were used and thus caused problems. This was discovered during validation by plotting histograms of the points accepted in accordance with the Metropolis probability as, if too small a step size is used, the random walk stays relatively local to the original random number, with the expected deviations, but does not give an accurate answer as it cannot cover the whole of the domain in the number of steps stipulated by the user supplied accuracy. However, if too large a step is used, the algorithm compares steps that would lead out of the integral domain often and thus considers some points far too many times.

During the validation of the Metropolis algorithm, other functions were integrated using the same routine to check the consistency of success. It successfully calculates the integral of the linear weighting function and a quadratic function $y(x) = 4 - x^2$, showing that with that linear weighting, it should be applicable to all decreasing functions, as expected.

Different starting random numbers were also used to test for accidental success, and this yielded similar results each time. In accordance with this, the routine was modified to start from multiple different random numbers in order to reduce the effects of bias towards one number. All points on the Metropolis random walk were considered after some deliberation, as the phenomenon known as 'burn-in' is not applicable here [2] due to there being no effective bias when sampling multiple random start points.

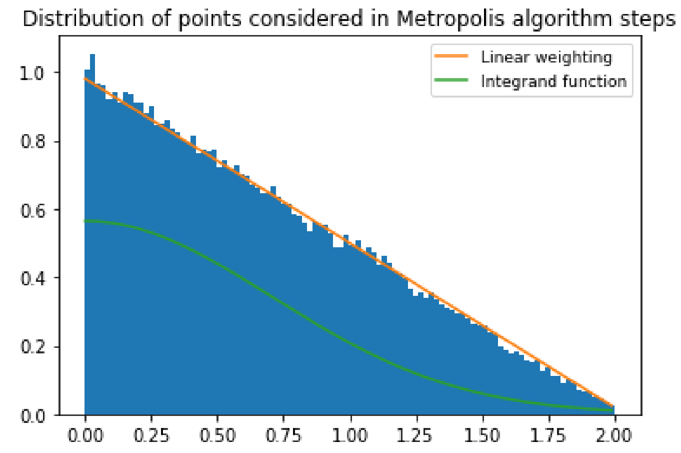

Figure 2: The normalised histogram of the random numbers considered using the metropolis algorithm with the weight function in (5), showing the expected linear distribution with some random fluctuations.

4 Results and discussion

As can be seen in Table 1, the implemented methods were compared on result, error and number of function evaluations required to reach these estimates. It was found that the NC methods achieved the most accurate estimates in the fewest evaluations, though with some of the largest error estimates. As such, and factoring in the ease of implementation, these methods would be used for integrating other 'well behaved' smooth functions if required. The difference between the number of function evaluations between these NC methods highlight the difference in efficiency between these methods. The fact that the lower order error terms associated with the extended trapezium rule cancel for the extended Simpson's rule lead it to converge in a significantly faster manner, thus making it far more efficient. It also obtained a result accurate to one more significant figure than that of the extended trapezium rule.

The methods using importance sampling, though significantly slower than the NC methods, were also compared. The estimate found that using a flat importance sample was more accurate to the true value than that of the linear importance sampling, and came to this estimate in about 20% fewer evaluations. However, the linear method, though less accurate, is more precise; as it carries a smaller error, by nature of the weighting. In light of these results, the flat importance sampling method is more appropriate here. If more time was available, different weighting functions would be investigated. Both of these methods computed the value of the function at many different random points in the domain, so were expected to be less computationally efficient than the NC methods.

As can be seen in Table 2, the number of evaluations required to give an estimate at the specified accuracies using the Monte Carlo methods were also compared. As expected, the errors on the estimates decreased with accuracy and the number of evaluations increased by about a factor of 10 for each corresponding increase in accuracy. This proportionality only consistently began after the 10^{-4} accuracy threshold though, suggesting that accuracies lower than this were obtained sporadically as the algorithm's rate of convergence changed. It was expected that the algorithm would converge quickly to some low accuracy, as the mean of the function is easily calculable for any random number distribution (again, to a certain accuracy). However, when it comes to smaller adjustments to fine tune the estimate to a given accuracy, more and more random samples are required, such that the fractional change in estimate is correct to that accuracy. Figure 3 shows the relationship between accuracy and number of function evaluations as a constant factor increase for accuracies greater than 10^{-5} for the Metropolis algorithm. The smaller accuracies not following this relation is attributed to the nature of random numbers, that leads to the integral domain not being fully 'walked over' by the Metropolis method when not enough steps are taken.

The importance sampling estimations were repeated with fewer random numbers being considered in each iteration. This reduced total number of function calls for a given accuracy, but only by a small amount. It was decided that the routine could be marginally optimized by changing this parameter; so if more time were available, the ideal number of points to consider would be investigated.

As previously alluded to, the random walk nature of the Metropolis method lead to the expectancy of some dependence on the start conditions, for a small number of steps. The same convergence rate examination was carried out with different random number seedings corresponding to the start point but yielded no change in the number of steps required to converge and minimal change to the values for the integrals. However, when the seed that corresponds to the random number generated for the Metropolis comparison step was changed, some variations were seen in both the estimates and the number of required function evaluations, especially in the smaller accuracies. The same linear relation emerges for greater accuracies, as can be seen in figure 4- the same exercise repeated for yet another seed. The behaviour for smaller accuracies fluctuating as such between the seeds supports the hypothesis that not all of the domain can be considered fully in some walks, thus showing dependence on the random walk. The comparison step seed changing allows the random walk to take a different (yet still random) path and thus weight the domain differently. A different path is taken, as the size direction and probability of the small steps change.

The metropolis algorithm was used in the study leading to figure 3, as it is the only Monte Carlo method implemented that will work for accuracies greater than 10^{-6}. The importance sampling based methods did not work to greater accuracies and crashed the Python IDE being used as the computer ran out of memory. This is not unexpected as vast amounts of random numbers are needed to achieve such accurate convergence using importance sampline, as we must consider an average of the integral in

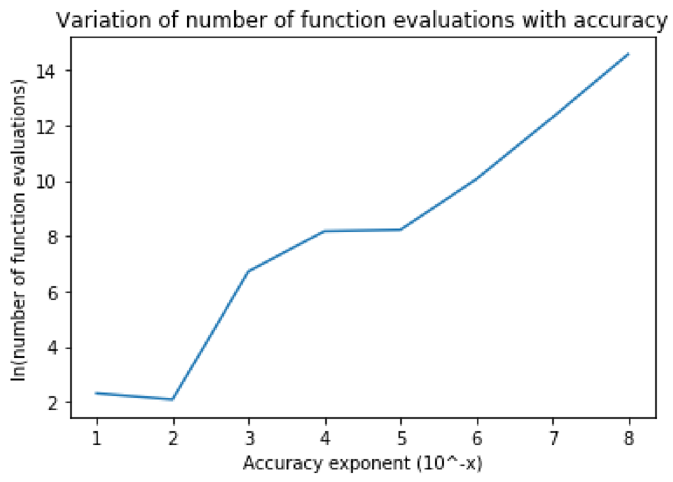

Figure 3: A plot showing the natural logarithm of the number of function evaluations varying with the specified accuracy. For accuracies of 10^{-5} and greater, a linear relation emerges.

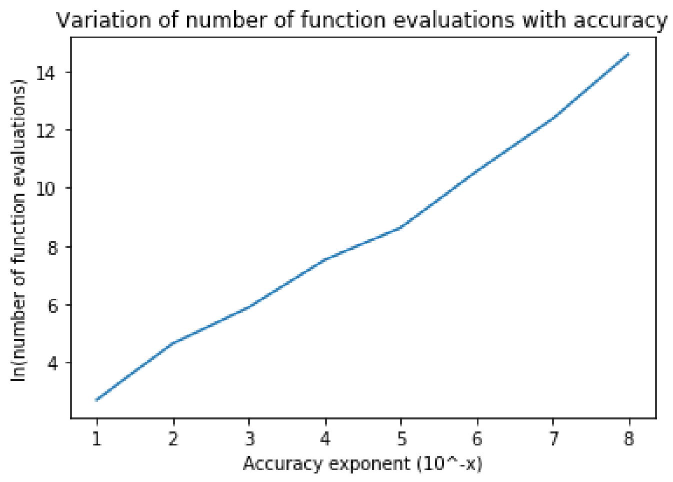

Figure 4: A plot showing the natural logarithm of the number of function evaluations varying with the specified accuracy, using a different comparison seed to that of figure 3. A linear still relation is evident.

question, which requires data points, which in turn will have a non zero standard deviation. If more time was available for further study, this rate of convergence would be investigated further to relate the relationship between accuracy and number of evaluations more closely.

The fact that the Metropolis method is based off of a random walk leads to the smaller number of steps required to achieve the same accuracies as the other Monte Carlo methods, as it can repeatedly sample the integral domain at the points of highest integral function value to obtain an average value for the integral with less variance. However this does lead to slight deviations in the integral estimates for the accuracies specified in the project. It can be seen that, though more time is taken to compute them, higher accuracy estimates are closer to the true value (see table 2). These numbers of function evaluations can be obtained in this manner, unlike that of using importance sampling, thus further highlighting the benefits of the Metropolis algorithm, especially in systems where a weighting function is unsuitable.

The time taken by these methods to produce the estimates of given accuracy was long in comparison with the NC methods, as given that the generation of random numbers is a relatively computationally costly process, the time to process thousands of random numbers was large compared to the convergent NC methods that only call the input function. Where the standard convergence of the NC methods allows them to reach specified accuracies quickly, the nature of random number generation requires many attempts to achieve successive average values of the integral with such little difference.

5 Conclusion

Many methods of numerically evaluating an integral were implemented in Python and, their convergence, computational efficiency, precision and accuracy were compared. It was found that extended Simpson's rule NC integration was the best option for this integral as it was the most precise and computationally efficient. The rates of convergence and number of function calls of various Monte Carlo methods were also examined and it was found that the Metropolis algorithm allowed the greatest accuracy of these methods and yielded a good estimate when enough sample points were considered. These methods were found to be effectively independent of the start conditions (and more dependent on the path of the random walk in the case of the Metropolis method). However, importance sampling methods converged to better estimates as they covered the domain with uniform distributions, though they had larger errors due to greater distribution spread. If more time were available, different weighting functions would be investigated, convergence rates would be examined more fully and the different methods would be applied to respectively more suitable problems. The VEGAS algorithm would also be considered.

6 Tables

Table 1: Results for 10^{-6} accuracy of all methods

Method	Result	Number of evaluations for 10^{-6} Accuracy	Error estimate
Extended Trapezium rule	0.4976616	513	0.0001037
Extended Simpson's rule	0.4976611	65	0.0022700
Flat importance sampling transformation	0.4974825	7587500	5.126303×10^{-8}
Linear importance sampling transformation	0.4976637	9106400	1.479620×10^{-8}
Flat importance sampling rejection method	0.5039080	26200	0.0030494
Metropolis algorithm	0.4981711	23113	5.811280×10^{-6}

Table 2: Results for Monte Carlo methods with varied accuracy

Method	Accuracy	Result	Number of evaluations	Error estimate
Flat importance sampling transformation	10^{-3}	0.4976616	86500	4.506961×10^{-6}
Flat importance sampling transformation	10^{-4}	0.497891	84700	4.588543×10^{-6}
Flat importance sampling transformation	10^{-5}	0.496989	696700	5.581723×10^{-7}
Flat importance sampling transformation	10^{-6}	0.497482	7587500	5.126303×10^{-8}
Linear importance sampling transformation	10^{-3}	0.497009	96900	1.392605×10^{-6}
Linear importance sampling transformation	10^{-4}	0.497250	96100	1.406480×10^{-6}
Linear importance sampling transformation	10^{-5}	0.497543	927600	1.454117×10^{-7}
Linear importance sampling transformation	10^{-6}	0.497663	9106400	1.479620×10^{-8}
Metropolis algorithm	10^{-3}	0.506356	822	0.000167
Metropolis algorithm	10^{-4}	0.543550	3546	2.829189×10^{-5}
Metropolis algorithm	10^{-5}	0.534356	3725	2.984560×10^{-5}
Metropolis algorithm	10^{-6}	0.498281	23113	5.811280×10^{-6}
Metropolis algorithm	10^{-7}	0.0.499139	215206	5.81421×10^{-6}
Metropolis algorithm	10^{-8}	0.496484	2130304	5.81723×10^{-6}
Metropolis algorithm	10^{-9}	0.495770	21125368	5.81214×10^{-6}

7 References

[1] Professor Yoshi Uchida and Dr Pat Scott, *Random Numbers and Monte Carlo Methods.* Imperial College London Computational physics

[2] Charles Geyer, *Burn-in is useless.* Available from http://users.stat.umn.edu/ geyer/mcmc/burn.html [Accessed 05/12/2018]