Computational Physics Project B2: Solving Quantum Systems Numerically Using Newton-Coates Rules and Monte Carlo Methods

I Kit Cheng
CID: 00941460
Imperial College London
(Dated: 20^{th} November 2017)

Newton-Coates rules (trapezoidal and Simpson's) and Monte Carlo Methods (fixed and adaptive importance sampling) were used in Python to numerically integrate a non-analytic 1D Gaussian function representing the probability density of finding a particle at location z (in arbitrary units). Considering a single particle initially well-localised in space, the probability of finding the particle in the range $z \in [0, 2]$ at some fixed point in time was found to be 0.497661 ± 10^{-6}. The results from different numerical methods used were consistent to at least 2d.p. of this value. For a relative accuracy of $\epsilon = 10^{-6}$, the computation time of Simpson's rule averaged twice as fast as trapezoidal rule. Furthermore, Simpson's required 65 function evaluations in total as opposed to 513. The relative accuracy achieved for MC methods varied between algorithms. In a 5 minute time window: the flat sampling achieved $\epsilon = 10^{-3}$, the linear sampling achieved $\epsilon = 10^{-4}$, the Metropolis algorithm achieved $\epsilon = 10^{-4}$ and the adaptive MC achieved $\epsilon = 10^{-6}$. In general, the fastest of the MC methods (adaptive MC) required a computation time that was 5 orders of magnitude longer than the fastest NC method (Simpson's rule), and 4 orders of magnitude more function evaluations for the same relative accuracy.

I. INTRODUCTION

Many physical systems like a quantum mechanical particle are described by non-analytic functions, thus numerical techniques are very powerful for obtaining a solution regardless of the form of the function.

This project investigates Newton-Coates rules, Monte Carlo and Markov-Chain Monte Carlo methods for integrating the square of the absolute value of a Gaussian wave-packet in position-space. Physically, the integral value represents the probability of finding the particle between $z = a$ and $z = b$ in position-space.

Newton-Coates rules are deterministic methods for estimating integrals. It works by summing the area of strips under the integrand which tend to the integral as the number of strips tend to infinity. In contrast, Monte Carlo integration is stochastic where random samples of the integrand are used to estimate the integral. As the number of samples tend to infinity, the exact integral is obtained.

The rest of this paper is organised as follows. The general theory and implementation of each algorithm is presented in Section 2. Section 3 outlines the validation of the code and Section 4 presents a summary of the results from the numerical experiments together with a discussion.

II. NUMERICAL METHODS

An `integrator` package was created which contains module files named `NC.py` with `NC` class (for Newton-Coates integration) and `MC.py` with `MC` class (for Monte Carlo integration). Objects of these classes are integrators which could perform integration on any user-supplied 1D function and automatically refines the result to a user-specified relative accuracy ϵ (see below for definition). They also contain methods that can print a summary of the integration results and produce plots.

For the purpose of this project, the integral to be solved is

$$I = \int_0^2 |\Psi(z)|^2 dz, \tag{1}$$

where $I \equiv P$ is the probability of finding the particle between $z = 0$ and $z = 2$, Ψ is the particle wavefunction given by

$$\Psi(z) = \frac{1}{\pi^{1/4}} e^{ia(z)} e^{-z^2/2}, \tag{2}$$

where z is a point in 1-dimensional space. The lower and upper bounds of the integral are denoted a and b respectively here onward.

A. Newton-Coates - Extended trapezoidal rule

The extended trapezoidal rule estimates the integral I to a certain relative accuracy ϵ by splitting the area below the integrand $f = |\Psi^2|$ into $n - 1$ trapezoids of height h and sum them up. This requires a total of n evenly spaced points where the integrand is evaluated. The estimate is given by

$$I \approx h[\frac{1}{2}f(x_0) + f(x_1) + ... + f(x_{n-2}) + \frac{1}{2}f(x_{n-1})], \tag{3}$$

where x_0 and x_{n-1} are a and b respectively. The relative error is defined as

$$err = \frac{I_n - I_{n-1}}{I_{n-1}}, \tag{4}$$

where I_{n-1} and I_n are successive refined estimates of I.

The algorithm has three main steps: To start, a) evaluate the first estimate I_1 with one trapezoid of height $h_1 = b - a$, requiring the integrand value at $f(a)$ and $f(b)$. Then, b) evaluate the mid-point $f(\frac{a+b}{2})$ to find I_2 with two trapezoids of height $h_2 = h_1/2$. Finally, c) If $err < \epsilon$, return the result. If not, the algorithm loops back to keep evaluating intermediate points doubling the number of trapezoids (i.e. h halves) at each iteration until $err < \epsilon$.

The routine implemented this by first defining an initial height $h = b - a$, a counter to record number of iterations and three more variables to store the sum f_{prev} defined in Eqn. 3, step-size h_{prev}, and number of points n_{prev} from the previous iteration. The algorithm described above was then performed under a while loop with a maximum limit of $1e10$ iterations to prevent an infinite loop. A manual exception would be raised should this limit be exceeded. Within the while loop, a new number of points $n = 2n_{prev} - 1$ was calculated at each iteration, and the height updated to $h = h_{prev}/2$. Using the slice function in Python, every other point starting from the second was selected to avoid evaluating the integrand twice at the same value of z. The integrand was evaluated at the selected new points and summed together with the previous sum, reweighing the whole sum with new h to give a refined estimate of I.

Since the trapezoidal method approximates the tops of the integrand as straight lines, this linear interpolation results in a local truncation error (i.e. after one iteration) that scales as $\mathcal{O}(h^3)$ and a global truncation error (i.e. cumulative error after many iterations) that goes as $\mathcal{O}(h^2)$. For a general d-dimensional integrand, the function needs to be evaluated at $N \approx h^{-d}$ points to calculate the integral. This leads to an error which scales as $\mathcal{O}N^{-2/d}$; becoming more significant at higher dimensions [1].

B. Newton-Coates - Extended Simpson's rule

The extended Simpson's rule uses an additional point requiring three integrand evaluations per sub-integral (i.e. per strip). It therefore improves on the trapezoidal rule by approximating the tops of the equal-width strips with quadratic curves rather than straight lines. This leads to a local and global truncation error that scales as $\mathcal{O}(h^5)$ and $\mathcal{O}(h^4)$ respectively. Therefore, the d-dimensional error scales as $\mathcal{O}N^{-4/d}$. There is an effective error cancellation between successive iterations of the trapezoidal rule T_i and T_{i+1} such that [1]

$$S_i = \frac{4}{3}T_{i+1} - \frac{1}{3}T_i, \tag{5}$$

where S_i is the estimation of the integral at the i^{th} iteration.

The Simpson's integration routine was simply piggy-backed onto the implementation of trapezoidal routine. The only extra line in the routine was Eqn. 5 and an extra variable S_{prev} to store the previous Simpson's estimate for calculating relative error using Eqn. 4.

C. Monte Carlo integration

The key concept in MC integration is writing the integrand f as a product of two functions Q and P, where $Q = f/P$, and P is the sampling pdf (aka the target density). Ideally, the target pdf should be as close to the integrand as possible to maximise efficiency- this is importance sampling. Then, the integral

$$I = \int_a^b f(z)dz, \tag{6}$$

becomes

$$I = \int_a^b Q(z)P(z)dz =< Q > . \tag{7}$$

Here, $Q = f/P$ is the integrand weighted by the pdf at position z and $< Q >$ is the expectation value of Q. By taking N random samples of z from a fixed pdf and evaluating Q at each point, $< Q >$ can be estimated using the mean formula (and hence an estimate for I),

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} Q_i = \bar{Q}, \tag{8}$$

such that $\hat{I} = I$ when N tends to infinity. Note that the pdf must be normalised over the integration domain (by definition). The estimated variance of I is given by

$$\hat{\sigma}_I^2 = \frac{1}{N}\sigma_Q^2, \tag{9}$$

where σ_Q^2 is the unbiased variance of Q,

$$\sigma_Q^2 = \frac{1}{N-1}(\sum_{i=1}^{N} Q_i^2 - N\bar{Q}^2). \tag{10}$$

When the pdf P is flat and normalised in the integration volume V (i.e. $P = 1/V$), the basic Monte Carlo estimate is obtained from Eqn. 8 and Eqn. 9 giving

$$\hat{I} = \frac{V}{N}\sum_{i=1}^{N} f_i \pm \frac{V}{\sqrt{N}}\sigma_f. \tag{11}$$

Physically, Eqn.11 says that the mean value of a function multiplied by some volume is the integral of the function inside this volume. This concept could be generalised to d-dimensional integrals where z is replaced by $\vec{z}$ and V becomes some hyper-volume with d-dimensions. The weighted mean of `niter` estimates of I is given by [2]

$$< \hat{I} >= \frac{\sum_i \hat{I}_i/\hat{\sigma}_{Ii}^2}{\sum_i 1/\hat{\sigma}_{Ii}^2}, \tag{12}$$

with a corresponding error of

$$\sigma_{<\hat{I}>} = \sqrt{\frac{1}{\sum_i 1/\hat{\sigma}_{Ii}^2}} \tag{13}$$

where the sum is from $i = 1$ to $i =$`niter`. The total number of samples taken would be (`niter` $\times N$). This weighted average technique was inspired by Lepage [4].

In MC integration, the integrator makes `niter` estimates of the integral using `neval` samples of the integrand taken randomly from a fixed pdf; known as fixed importance sampling, except for a flat pdf- known as uniform sampling. Importance sampling places more samples in peak regions of the integrand where the integral depends on more, thus faster convergence to true integral. The integrator returns the weighted average `wgtQmean` of all `niter` estimates, together with a relative error `wgterr/wgtQmean` on the weighted mean. Note that Eqn. 4 was not used for relative error because the use of random numbers could result in very little change in successive integral estimates even though both are far from the convergence value. The use of relative error could blow up if the true integral converges to zero. However, since the integrand of interest is a Gaussian the integral over a finite region will always be greater than zero.

To implement this, a large number of $Q(z)$ was computed where z was sampled from a) a flat sampling $pdf(z) = 0.5$ and b) a linear sampling $pdf(z) = -0.48z + 0.98$, both normalised over the interval $z \in [0, 2]$. For general integration limits, the flat distribution takes the form $pdf(z) = 1/(b-a)$. The transformation method was used to map the uniform pdf over the interval $[0, 1]$ onto the linear one. By definition the pdf's area is one. Thus, the area of the desired pdf from 0 to z must equal to the area under the uniform pdf from 0 to x. Mathematically,

$$\int_a^z (A\tilde{z} + B)d\tilde{z} = \int_0^x d\tilde{x}. \tag{14}$$

Rearranging and using the quadratic formula gives $z = z(x)$. The coefficients A and B were generalised for any arbitrary integration limits. This was achieved by using the ratio of the pdf values at both ends as a constraint and then normalising. See `pdf` routine in source code for their expressions. Using `numpy.random.uniform(0,1)`, samples of z were produced. Initially, the algorithm kept generating $Q(z)$ and calculating its mean until the required accuracy was satisfied. However, it was found that by taking several iterations with a fixed number of Q(z) sampled and then computing an error weighted average over all the iterations improved the accuracy of the result and decreased total computation time.

Monte Carlo integration has a big advantage over trapezoidal rule when the dimension of the problem becomes large because the error of the integral always scales as $\mathcal{O}(N^{-1/2})$ as seen in Eqn.9, whereas it grows with the dimension in trapezoidal rule. Thus for 4 or more dimensions the MC method is as accurate if not more than the trapezoidal method for the same number of samples [1].

The caveat of the basic MC is the use of transformation method to map the uniform distribution on interval [0,1] to the desired sampling pdf of arbitrary range. This may not be analytically possible in more complicated pdfs. To avoid this, an adaptive MC or Metropolis Algorithm could be used.

D. Adaptive Monte Carlo (AMC)

The AMC algorithm is based on the basic MC method described above, but differs in that it learns about the integrand as it aquires samples in each iteration. The algorithm then divides the integration region with maximum standard deviation (i.e. region where the integrand value varies most) into two equal halves and performs basic MC in each, ensuring only important regions are sampled more. This process is repeated until the total standard deviation from all the subregions is less than some relative accuracy. Using the Riemann integral principle, the total integral is simply the sum of the sub-integrals from each sub-region. The total error is found from adding in quadrature [3].

The algorithm is implemented as an `adapt=True` or `False` option inside the existing `MCInt` method of the MC class. Parameters for the number of random points N in each new region and the maximum number of iterations T are set when the `MC` object is initialised. The algorithm could be summarised into 5 main steps: 1) add lower and upper bounds of integration into `A` and `B` arrays respectively. 2) Get esimates of integral I and error E using Eqn. 11 and add them into `TM` and `TE` arrays respectively. 3) Split region (with max error) into two equal halves $c = (a + b)/2$, update A and B arrays ($A = [a, c], B = [c, b]$), get new estimates of I and E in the two new regions using Eqn. 11 (where V is now the width of the subregion) and update TM and TE arrays. 4) Find index ai of region with highest error using `numpy.argmax(TE)` function in Python, and increment to next iteration. 5) The total integral is $\hat{I} = \sum TM_i$ and error $\hat{E} = \sqrt{\sum TE_i^2}$. If the relative error $\hat{E}/\hat{I}$ satisfies relative accuracy, return results. Otherwise, goto step Furthter details of the algorithm are in Alrefaei (2007) [3].

E. Monte Carlo Markov Chain - Metropolis Algorithm

The Metropolis algorithm (MA) is very powerful as it can generate samples from any target density P which can be un-normalised, a huge advantage over non-adaptive MC that requires using transformation method (see Section 3C).

In MA, new samples of the integrand are proposed randomly using a normal distribution centred on the previous z-value. If the new value `znew` lies outside the integration domain, a periodic boundary condition is used to add or subtract integer multiples of $(b-a)$ until it is inside the region. If P at `znew` is greater than that at `z`, the proposed step is accepted. Otherwise, it is accepted with a probability `P(znew)/P(z)` by comparing to a random number between 0 and 1. If the step is rejected, then `znew = z` (i.e. stay at current z). The variance of the Gaussian proposal is adjusted for a reasonable acceptance rate of $\approx 70\%$; decrease variance for higher acceptance and vice versa. This sampling method results in non-independent samples z_i, contrary to the transformation method.

To get an estimate of the integral, Eqn. 8 is used. However, the pdf must first be normalised if not already. From the list of z-samples obtained, `numpy.hist` is used to extract the normalised density in each bin and the values of bin edges. By identifying which bin each z-value belongs to, the density value P at z is found. This allows $Q(z)$ to be calculated. The algorithm continues as in MC integration using Eqns. 8,9 and 10 to update the integral and error at each iteration. Finally, Eqns. 12 and 13 are used to give a final weighted estimate of both the integral and error.

III. CODE VALIDATION

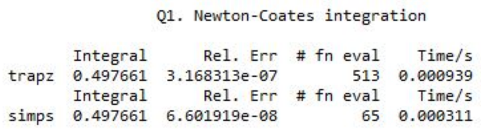
The integral values obtained were validated against the output of `scipy.integrate.quad` integration function. They were consistent up to six decimal places when a relative accuracy of $\epsilon = 10^{-6}$ was achievable. In addition, the MC and MCMC results agreed with the Newton-Coates results too. Since these methods are independent, the result obtained is likely to be reliable.

The sampling distribution was validated by comparing the histogram of all the samples with the desired pdf which showed good agreement.

IV. RESULTS AND DISCUSSION

A. Newton-Coates

Results from NC integrator using the trapezoidal and Simpson's rule are summarised in Fig. 1. The probability of finding the particle between $z = 0$ and $z = 2$ (i.e. the solution to Eqn. 1) was found to be 0.497661 (to 6d.p.) with a relative accuracy of $\epsilon = 10^{-6}$. The number of function evaluations for trapezoidal rule was 513 as compared to only 65 for Simpson's rule; approximately 8 times less. The computational time of Simpson's was on average 2 times faster. The error in trapezoidal and Simpson's fell as N^{-2} and N^{-4} respectively, as expected from theory (see Fig.2). Fig. 3 shows that Simpson's converged much more sharply than trapezoidal even though its initial estimate was worse. This was again expected due to a more accurate quadratic approximation of function tops.



```
                 Q1. Newton-Coates integration

         Integral      Rel. Err   # fn eval    Time/s
trapz    0.497661  3.168313e-07         513  0.000939
         Integral      Rel. Err   # fn eval    Time/s
simps    0.497661  6.601919e-08          65  0.000311
```

FIG. 1. A table summarising the integration results of NC integrator for a relative accuracy of $\epsilon = 10^{-6}$.

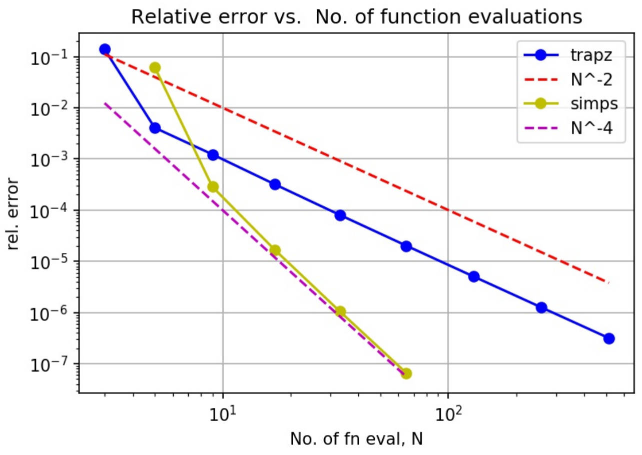


FIG. 2. A plot showing the relative error against number of function evaluations (for $\epsilon = 10^{-6}$). The trapezoidal rule (blue) shows a scaling proportional to N^{-2} (red). The Simpson's rule (yellow) shows a scaling proportional to N^{-4} (magenta) after approximately 9 function evaluations. The first data point is at N = 3.

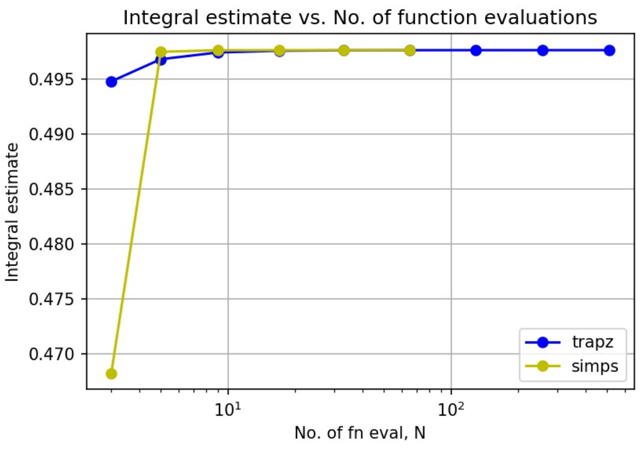
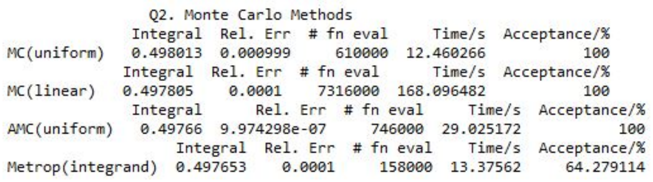


FIG. 3. A plot showing the integral estimate against number of function evaluations (for $\epsilon = 10^{-6}$). The trapezoidal rule (blue) shows a gradual convergence whereas the Simpson's rule (yellow) show a much sharper convergence. The first data point is at N = 3.

B. Monte Carlo

Results from MC integrator using a flat and linear sampling pdf are summarised in Fig.4. For a given user specified accuracy ϵ, the linear importance sampling was superior over the flat sampling with almost an order of magnitude improvement in both the iterations required and computation time (see Fig. 5). For $\epsilon = 10^{-3}$, the probability obtained was 0.498 which agrees up to 2d.p. with the results from NC integrator. The error decreased as $N^{-1/2}$ as expected from Eqn. 11. In a 5 minute time window, the maximum relative accuracy achieved by the flat and linear were 10^{-3} and 10^{-4} respectively. However, the computation time was found to depend significantly on the computer used. Higher relative accuracies could not be achieved because the number of function evaluation grew as ϵ^{-2} for both flat and linear sampling (as expected from Eqn. 9, see Fig. 6). This means a 10 times decrease in ϵ results in a 100 times increase in function evaluations required. Since the computation time is roughly proportional to the number of samples, the time required also increases by 100 times making it impractical. Based on its performance so far and results from NC integrator, it is expected that the flat and linear pdf algorithms would require a total 10^{12} and 10^{11} function evaluations respectively to achieve $\epsilon = 10^{-6}$.



```
                  Q2. Monte Carlo Methods
                Integral  Rel. Err  # fn eval      Time/s  Acceptance/%
MC(uniform)     0.498013  0.000999     610000   12.460266           100
               Integral  Rel. Err  # fn eval      Time/s  Acceptance/%
MC(linear)     0.497805    0.0001    7316000  168.096482           100
                Integral      Rel. Err  # fn eval      Time/s  Acceptance/%
AMC(uniform)     0.49766  9.974298e-07     746000   29.025172           100
                     Integral  Rel. Err  # fn eval      Time/s  Acceptance/%
Metrop(integrand)    0.497653    0.0001     158000   13.37562     64.279114
```

FIG. 4. A table summarising the integration results of MC integrator for relative accuracies achievable in a 5 minute time window. Uniform: $\epsilon = 10^{-3}$, Linear: $\epsilon = 10^{-4}$, Adaptive MC: $\epsilon = 10^{-6}$, Metropolis: $\epsilon = 10^{-4}$.

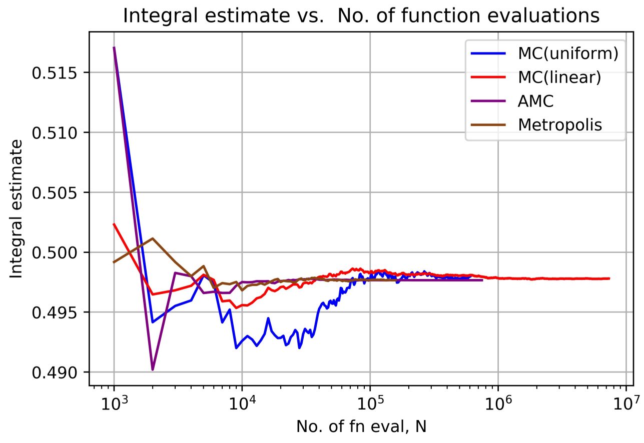


FIG. 5. A plot showing evolution of the integral estimate with number of function evaluations. Uniform sampling took the longest to converge, followed by linear, Metropolis, and AMC. Interestingly, all methods overestimates in the first iteration, overshoots in the next (except for Metropolis), before converging to the the value of the integral.

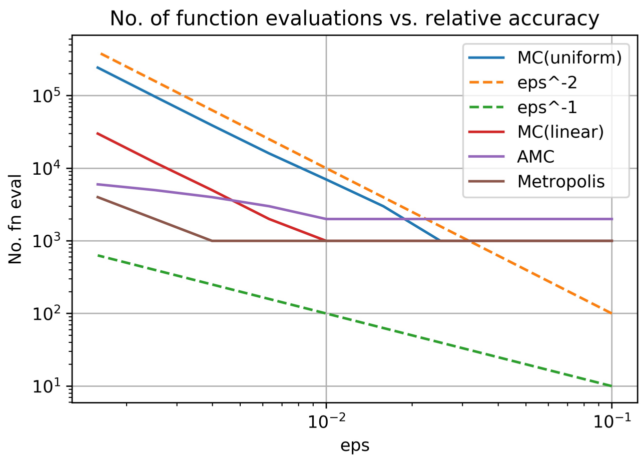


FIG. 6. A plot showing how the no. of function evaluations vary with relative accuracy of the result. The uniform (blue), linear (red) and Metropolis (brown) sampling methods scales as ϵ^{-2}. The AMC (purple) method has a even gentler slope than ϵ^{-1} making it the most efficient method.

C. Adaptive Monte Carlo

The adaptive Monte Carlo (AMC) algorithm allowed relative accuracies up to 10^{-6} to be achieved, giving a probability of 0.497660 (to 6 d.p.) which agreed almost exactly to that obtained by the NC integrator. This is because samples were taken from the most important regions of the integrand, thus not wasting computational resources on non-important regions such as relatively flat regions.

D. Metropolis Algorithm

The metropolis algorithm was used to sample the integrand and the results outperformed the non-adaptive MC method for both flat and linear pdf (see Fig. 4). The sampled distribution is shown in the Figure 7 below along with other distributions obtained from the above algorithms. The number of function evaluations varied randomly with relative accuracy. It was able to achieve $\epsilon = 10^{-6}$.

V. CONCLUSION

In this project, Newton-Coates (NC) and Monte Carlo (MC) methods were used to determine the probability of finding a quantum particle between $z = 0$ and $z = 2$. The probability was 0.497661 (to 6d.p.) with a relative accuracy of $\epsilon = 10^{-6}$. This value is reliable as all of the independent methods investigated converged towards this value. Simpson's rule was the most efficient, followed by trapezoidal rule, adaptive MC, Metropolis, MC with linear importance sampling and the least efficient MC with

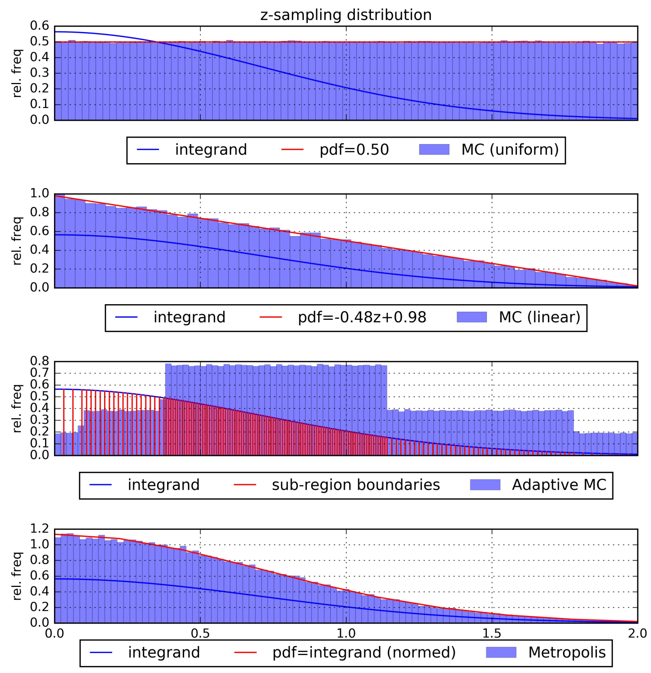


FIG. 7. A plot showing the distribution of the integrand samples using MC and Metropolis sampling methods. They all agree with the desired theoretical pdf.

flat pdf (i.e. no importance sampling). The adaptive importance sampling used in this project greatly improved the MC method allowing it to achieve a relative accuracy of $\epsilon = 10^{-6}$ that would otherwise be impossible inside a 5 minute time window.

Word Count: 2031

[1] Uchida Y., Scott P. (2017) *Lecture Notes 2017*. Blackboard Imperial. Available from:
`https://bb.imperial.ac.uk/webapps/ blackboard/ content/listContent.jsp?course_id=_12315_1content_ id=_1216038_1`
[Accessed 10th November 2017]

[2] Caltech. *The Weighted Mean*. Available from:
`https://projecteuclid.org/download/pdf_1/euclid.bj/1080222083`
[Accessed 12th December 2017]

[3] Alrefaei, M., et al. (2007) An adaptive Monte Carlo integration algorithm with general division approach. *Science Direct*. Available from:
`www.sciencedirect.com` [Accessed 17th December 2017]

[4] Lepage G., (2017) *vegas Documentation*, Available from: http://pythonhosted.org/vegas/ [Accessed 16th December 2017]