LU-TP 21-04
March 2021

Squaring VEGAS
Multidimensional Integration Using 2D Correlations

Philip Siemund

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Rikkert Frederix

Abstract

Multidimensional integrals are used in many areas of physics; electrodynamics, quantum mechanics and statistical physics, to name a few. Many integrals can not be solved analytically, but can instead be approximated numerically. VEGAS is a Monte Carlo integration algorithm, which specializes it in approximating multidimensional integrals. By treating each integration variable independently, VEGAS only offers good approximations if the characteristic regions of the integrand's graph align with the coordinate axes. This thesis presents a modified VEGAS called VEGAS squared, which tracks possible correlations between the integration variables pairwise. When integrating Gaussian functions in two and three dimensions, for which the central parts of the graphs misalign with the coordinate axes, VEGAS squared produces a standard error by a factor of 2.5 less than what VEGAS does.

From Monte Carlo to VEGAS to VEGAS squared

A definite integral represents the area under the graph of the function, known as the integrand, that is to be integrated. A double definite integral represents a volume and anything beyond represents a hypervolume. The latter integrals are referred to as multidimensional integrals and can be found in many areas of physics; electrodynamics, quantum mechanics and statistical physics, to name a few. Many integrals do not have an exact solution in which the value of the integral can be expressed with a finite number of symbols. These integrals can instead be approximated numerically; through solutions that can be shown to be almost exact.

Monte Carlo integration, named after the Monte Carlo Casino in Monaco, is a numerical integration technique in which the integral is approximated by the mean of random evaluations of the integrand. More generally, the method of using random sampling as a means to estimate a deterministic quantity, such as a hypervolume, is known as the Monte Carlo method. This method is said to originate from scientists working at the Manhattan project, where they were challenged with testing their theories about neutron diffusion in fissionable material while simultaneously not affording to run countless experiments. Since then it has become a very popular tool in making theoretical predictions and its applications reach beyond sciences.

The Monte Carlo method uses a brute force approach. Consider rolling two dice. To find out the different probabilities of the outcomes, one can construct a simple table. There will be thirty-six outcomes in total and finding the probabilities from the table is straightforward. This is the mathematical approach and no dice were rolled. Another approach is to actually roll the dice, say ten thousand times, and make a note of the outcome with each roll. This is what the Monte Carlo method does; your results will be approximate, meaning there will be an uncertainty, and they can always be improved by continuing rolling the dice.

The uncertainty in the approximation indicates how well a numerical integration technique performs. There are different strategies to reduce the uncertainty when using Monte Carlo integration. VEGAS is a Monte Carlo integration algorithm that tries to reduce the uncertainty by evaluating the integrand where it is large. This strategy has made VEGAS very efficient in higher dimensions compared to other Monte Carlo integration algorithms using other strategies. However, VEGAS only offers good approximations if the characteristic regions of the integrand's graph align with the coordinate axes of a given coordinate system. The purpose of this thesis is to modify VEGAS to improve its approximation of integrals for which it performs less well and thereby possibly contribute to a more accurate prediction somewhere. The modified VEGAS is called VEGAS squared.

Contents

1 Introduction **4**

2 Theory **5**

2.1 Monte Carlo integration 5

2.2 Importance and stratified sampling 7

2.3 Curse of dimensionality 8

2.4 VEGAS algorithm 9

2.4.1 Optimal number of bins in VEGAS 10

2.5 VEGAS squared 11

2.6 2D correlations 14

2.7 VEGAS squared algorithm 15

3 Result **18**

4 Conclusion **19**

5 References **20**

Appendix Python Code **21**

VEGAS 21

VEGAS squared 24

List of acronyms

MC	Monte Carlo
i.v.('s)	integration variable(s)
nD	n-dimension(s)(al)
LLN	law of large numbers
r.v.('s)	random variable(s)
pdf(s)	probability density function(s)
i.i.d.	independent and identically distributed

1 Introduction

VEGAS [1] is a Monte Carlo (MC) integration algorithm, which specializes it in approximating multidimensional integrals. By treating each integration variable (i.v.) independently, VEGAS only offers good approximations if the characteristic regions of the integrand's graph align with the coordinate axes of a given coordinate system. The purpose of this thesis is to modify VEGAS to improve its approximation of integrals for which it performs less well, that is, where the characteristic regions of the integrand's graph do not align with the coordinate axes, or equivalently, if the level sets corresponding to the characteristic region display correlations between the i.v.'s. Using the programming language Python, a self-written VEGAS will be implemented and subsequently modified. The modified VEGAS is called VEGAS squared.

The thesis is structured as follows. The Theory section, section 2, starts with a brief overview of how integrals can be approximated numerically. This is followed by subsection 2.1 on MC integration and in turn by subsection 2.2 on two variance reduction techniques, namely importance and stratified sampling. Next, subsection 2.3 highlights some of the challenges in approximating multidimensional integrals, specifically the phenomenon known as the curse of dimensionality. The VEGAS algorithm is presented in subsection 2.4, which is a rewrite of the algorithm as presented in [2]. Then follows subsection 2.5, in which the necessary changes that need to be made in VEGAS in order to transition to VEGAS squared are discussed. Subsection 2.6 defines the 2D correlations used in VEGAS squared. Finally, the VEGAS squared algorithm is presented in subsection 2.7.

In the Result section, VEGAS squared is compared to VEGAS. This involves comparing standard errors when integrating a 2D (two-dimensional) Gaussian function. Since a Gaussian function is commonly used in approximating parts of other functions, it lends itself well for testing. Gaussian functions can also be rotated so that the characteristic regions of their graphs do align with the coordinate axes, however, for many other functions this is not possible due to the correlations between the variables. The computational costs of the two implementations for a fixed number of points will not be compared, since in practice they would be negligible compared to the computational cost of evaluating the integrand.

2 Theory

In numerical integration, the integral is approximated by a weighted sum of a finite number of evaluations of the integrand. How the weights and points for the evaluations are determined is what characterizes a specific technique. The goal is to get an error in the approximation less than a given threshold with as few points as possible. For example, the numerical approximation of the Riemann integral follows from its definition.

Definition 1. Let $f(x)$ be a function defined on the interval $[a, b]$. Let $\mathcal{P}_N$ be a partition of $[a, b]$ consisting of N intervals $[x_{k-1}, x_k]$ and $k = 1, \ldots, N$. Then, for all choices of $c_k \in [x_{k-1}, x_k]$ and all possible sets $\mathcal{P}_N$ for which the largest interval approaches 0 as $N \to \infty$, the Riemann integral I is defined as

$$\lim_{N\to\infty} \sum_{k=1}^{N} f(c_k)(x_k - x_{k-1}) = I = \int_a^b f(x)\mathrm{d}x \ . \tag{2}$$

The weighted sum approximates the integral and is known as the Riemann sum. The weights are $x_k - x_{k-1}$ and the points c_k are predetermined. When f is only evaluated at x_{k-1} or x_k, this is known as a left and right Riemann sum respectively. The trapezoidal rule is a numerical integration technique that averages the left and right Riemann sums. Its rate of convergence ϵ_N, i.e. how fast the error in the approximation approaches 0, is of order $N^{-2/n}$ [3], where n is the number of dimensions. This means, for some positive constant C and a given threshold δ such that $\epsilon_N \leq \delta$, N must satisfy

$$\epsilon_N \leq CN^{-2/n} \leq \delta \implies N \geq \left(\frac{C}{\delta}\right)^{n/2} \ . \tag{3}$$

For a fixed ϵ_N, the number of function evaluations grows exponentially with n. MC integration will be shown to have a rate of convergence of order $N^{-1/2}$, independent of dimension. This makes it the preferred numerical integration technique in higher dimensions.

2.1 MC integration

The trapezoidal rule uses a deterministic approach, meaning that each point where the function is evaluated is known beforehand. MC integration, on the other hand, is nondeterministic; it uses random points when evaluating the function. This alludes to introducing MC integration from a probabilistic standpoint.

The motivation behind MC integration arises out of two basic concepts in probability theory; expectation and law of large numbers (LLN). The strong LLN relies on convergence in p'th mean; a form of stochastic convergence. Intuitively, the expectation describes where the distribution of a random variable (r.v.) is centered, and the LLN says that the probability of an event approximately equals the proportion of times the event has occurred in a large number of trials.

Definition 4. Let X be a continuous r.v. defined on the interval $[a,b]$ with probability density function (pdf) $g(x)$. Then its expectation $\mathrm{E}[X]$ is defined as

$$\mathrm{E}[X] = \int_a^b x g(x) \mathrm{d}x \ . \tag{5}$$

Definition 6. Let X be a r.v. and $\{X_k\}_{k\geq 1}$ a sequence of r.v.'s. Also, let $p > 0$ and $\mathrm{E}[|X|^p], \mathrm{E}[|X_k|^p] < \infty$. If $\mathrm{E}[|X_k - X|^p] \to 0$ as $k \to \infty$, X_k converges in p'th mean to X, denoted $X_k \xrightarrow{L^p} X$. In particular, for $p = 2$, it is called convergence in quadratic mean.

Theorem 7. *Let $X_1, \ldots, X_N$ be independent and identically distributed (i.i.d.) r.v.'s and $\mathrm{E}[X_i] = \mu$. Then the arithmetic mean of $X_1, \ldots, X_N$ converges in quadratic mean to μ, that is,*

$$\frac{1}{N}\sum_{i=1}^{N} X_i \xrightarrow{L^2} \mu \ . \tag{8}$$

Note that the i.i.d. r.v.'s $X_1, \ldots, X_N$ can be seen as a sample of a single r.v. X, in which case the capital letters become lower case ones, and the arithmetic mean becomes the sample mean. Thus, let X be uniformly distributed over $[a,b]$, so that $g(x) = 1/(b-a)$, and define the r.v. $f(X)$, which is a r.v. if the function f is well-behaved[1]. Using the definition of expectation and LLN, it follows that a reasonable approximation I_N of the integral I in equation 2 is given by

$$\int_a^b f(x)\mathrm{d}x \approx I_N = \frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i) \ . \tag{9}$$

The weighted sum is the approximation of the integral in MC integration. That the rate of convergence of I_N goes as $N^{-1/2}$ follows from the variance of I_N, $\mathrm{Var}[I_N]$, which in turn is determined by the variance of $f(X)$, $\mathrm{Var}[f(X)]$. The square root of $\mathrm{Var}[I_N]$ is the standard error u_N. If $\mathrm{Var}[f(X)]$ is given by the biased sample variance, then

$$\begin{aligned} u_N = \sqrt{\mathrm{Var}[I_N]} = \sqrt{\frac{(b-a)^2}{N}\mathrm{Var}[f(X)]} &\approx \frac{(b-a)}{\sqrt{N}}\sqrt{\left(\sum_{i=1}^{N}\frac{f(x_i)^2}{N}\right) - \left(\sum_{i=1}^{N}\frac{f(x_i)}{N}\right)^2} \\ &= \frac{1}{\sqrt{N}}\sqrt{(b-a)^2\left(\sum_{i=1}^{N}\frac{f(x_i)^2}{N}\right) - I_N^2} \ . \end{aligned} \tag{10}$$

If $\mathrm{Var}[f(X)]$ is bounded as $N \to \infty$, then u_N approaches 0 as $N^{-1/2}$. The proportionality constant $N^{-1/2}$ provides a simple check that the numerical integration algorithm behaves correctly, i.e. u_N should decrease by a factor of $N^{-1/2}$ when increasing N and vice versa.

[1]By the law of the unconscious statistician, see theorem 7.11 in [4], $\mathrm{E}[f(X)]$ is simply given by replacing x by $f(x)$ in Definition 4.

Another sound inspection related to equation 10 is comparing the integral of $f_1(x) = 1/\sqrt{x}$ and $f_2(x) = 1/\sqrt{x + 0.001}$ over the interval $[0, 1]$, both of which exist. For small x, the latter integrand is bounded by $1/\sqrt{0.001}$, whereas the former approaches infinity. Thus the variance can get much larger for f_1 over $[0, 1]$, which implies a larger u_N.

By reducing $\text{Var}[f(X)]$ in equation 10, one reduces u_N. Hence different variance reduction techniques have been developed to make MC integration more efficient. Two such techniques are highlighted in the next section; importance and stratified sampling.

2.2 Importance and stratified sampling

Importance sampling is a variance reduction technique used by VEGAS. It samples where it is important, which is accomplished by replacing the uniform distribution with a non-uniform one. Consider a very large and narrow Gaussian function defined on the interval $[0, 1]$. Recall that in MC integration one approximates the expectation by the sample mean, and thus the contribution to I_N should come from the part of the function where it steeply rises and reaches its peak. This part, however, corresponds to a relatively small interval on the x-axis. A uniform pdf is not ideal in this case since the length of the interval equals the probability of finding a point there. Rather, a pdf $h(x)$ that looks just like the function is more suitable. Thus I_N in equation 9 becomes

$$\int_a^b f(x)\mathrm{d}x = \int_a^b \frac{f(x)}{h(x)} h(x)\mathrm{d}x \approx I_N = \frac{(b-a)}{N} \sum_{i=1}^{N} \frac{f(x_i)}{h(x_i)} \ . \tag{11}$$

In two and higher dimensions, VEGAS defines the joint pdf associated with the multivariate integrand as a product of the marginal pdfs of the r.v.'s. That is, $h(x, y) = h(x)h(y)$ in 2D. This is equivalent to all r.v.'s being independent[2].

Another variance reduction technique is called stratified sampling. One partitions the domain in subdomains, i.e. strata, of possible different sizes. MC integration is performed in each subdomain with the original number of points divided by the number of subdomains. The sizes of the subdomains are varied and minimized when the contributions to the variance from each subdomain are equal. Consider again a very large Gaussian-like function, but with a peak stretching over a large part of the interval over which the function is defined. Importance sampling is not ideal in this case, since then the contribution to I_N will mainly come from the value of the function at the peak. By dividing the domain into subdomains where the function is constant and non-constant, contributions to I_N from both parts will be considered.

In summary, importance sampling increases the density of points where the integrand is large. Stratified sampling increases the density of points where the integrand changes rapidly. In the case of a large and narrow Gaussian function, these techniques are equivalent.

[2]This follows from theorem 5.18 in [4].

2.3 Curse of dimensionality

Although MC integration has a rate of convergence independent of dimension, the space of higher dimensions poses another problem; sparseness. When the number of dimensions increase, so do the degrees of freedom. There are two ways to move away from the center of a line, but an infinite number of ways to move away from the center of a square. In a sense, the points can thus differ in more ways from each other as the dimensions increase. For a fixed number of random points, this leads to a larger average distance between the points. As each point also has a higher probability for any of its components to lie at the edge of an axis with more axes, the points gather at the edge of the integration domain. The probability that points lie at the center of the domain approaches 0. The curse of dimensionality derives its name from the fact that this happens regardless of how the points are distributed and counteracting it requires an exponential increase of points. In general, a high density of points in each part of the integration domain will give a fast convergence, but as the number of dimensions grow, this becomes close to impossible.

To illustrate the above, consider the volume of a unit hypercube in n dimensions (nD), which equals 1. The fraction of its volume occupied by another hypercube inside of it is simply the volume $V(r,n) = r^n$ of the other hypercube, where r is its side length. Figure 1 shows how $V(r,n)$ depends on r for different n. In 10D, a side length of 0.8 only corresponds to approximately 0.11 of the volume of the unit hypercube. Specifically, it is the fraction that goes to 0 as $n \to \infty$, regardless of the side lengths of the two hypercubes. A point with any of its components in the interval $[0.8, 1]$ will most likely be far away from an already sparse set of points.

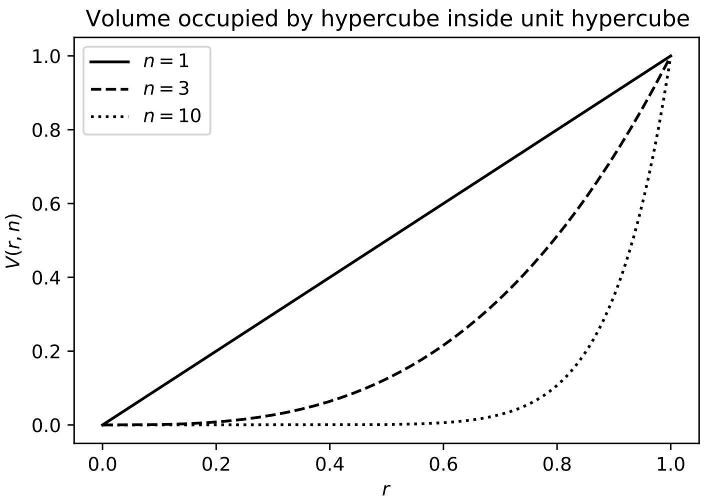


Figure 1: As n increases, so does the sparseness of a fixed number of points in the integration domain.

2.4 VEGAS algorithm

In words, there are two unit hypercubes; the x-cube, the integration domain, and the y-cube, where random points are generated in. In the x-cube the bins are varied in size and in the y-cube they are fixed. The first iteration starts with the bins in the x-cube having the same length as those in the y-cube. One generates N random points in the y-cube and maps those to points in the integration domain. This map assigns different probabilities to each bin in the x-cube, but in VEGAS the bins in the y-cube are fixed, so each bin in the x-cube gets on average the same number of points. The integral is then approximated by the sample mean of the random evaluations of the function and the error by the standard error. For each iteration, one stores the integral in each bin on each axis. This information is then used to change the bin sizes in the x-cube, so that the integral in each bin is approximately equal. Thus, a large bin corresponds to a region where the function is small and vice versa.

In symbols, and following the notation of [2], consider the nD unit hypercube for $n \geq 1$, and let $f(x^1, \ldots, x^n)$ be an integrable function in this domain, where x^k denotes the k'th coordinate. Divide the interval $[0, 1]$ for each coordinate in m bins of variable length. Let y^k denote the k'th coordinate in another unit hypercube, and divide the interval $[0, 1]$ for all of these in m fixed bins of length $1/m$. Define n piecewise linear and strictly increasing functions $h^k(y^k)$ by

$$h^k\left(\frac{l}{m}\right) = x_l^k \ , \tag{12}$$

where $l = 0, \ldots, m$ and $k = 1, \ldots, n$. Thus, the point l/m on every y^k-axis maps to the point x_l^k on every x^k-axis, each marking the possible end and beginning of a fixed bin and variable sized bin respectively. If y^k is given and y is a random point in the interval $[0, 1]$ satisfying $(l-1)/m < y < l/m$, then

$$\left.\frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k}\right|_{y^k=y} = (x_l^k - x_{l-1}^k)m \ . \tag{13}$$

The first iteration starts with $x_l^k = l/m$. A set $\mathcal{Y}_N$ of N random points $\mathbf{y}$ are generated in the unit hypercube $0 \leq y^k \leq 1$. The integral of $f(x^1, \ldots, x^n)$ is approximated by I_N in equation 9, and the error u_N by equation 10,

$$\int f(x^1, \ldots, x^n) \prod_{k=1}^{n} \mathrm{d}x^k = \int f(h(y^1), \ldots, h(y^n)) \prod_{k=1}^{n} \frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k} \mathrm{d}y^k$$

$$\approx I_N = \frac{1}{N} \sum_{\mathbf{y} \in \mathcal{Y}_N} f(h(y^1), \ldots, h(y^n)) \prod_{k=1}^{n} \frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k} \ , \tag{14}$$

$$u_N \approx \frac{1}{\sqrt{N}} \sqrt{\left(\sum_{\mathbf{y} \in \mathcal{Y}_N} \frac{\left(f(h(y^1), \ldots, h(y^n)) \prod_{k=1}^{n} \frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k} \right)^2}{N} \right) - I_N^2} \ . \tag{15}$$

For each point $\mathbf{y}$ and each of its components y, the cumulative sum of the integral less than a factor m^n as well as the number of points in a given bin on the y^k-axis are stored in the arrays R_{lk} and N_{lk}. These arrays are used to compute the average integral in each bin on the x^k-axes, and the cumulative sum of these are stored in the array I_{lk}, where $I_{0k} = 0$. That is,

$$R_{lk} = \sum_{\mathbf{y} \in \mathcal{Y}_N} 1\{(l-1)/m < y < l/m\} f(h(y^1), \ldots, h(y^n)) \prod_{k=1}^{n} (x_l^k - x_{l-1}^k) \ , \tag{16}$$

$$N_{lk} = \sum_{\mathbf{y} \in \mathcal{Y}_N} 1\{(l-1)/m < y < l/m\} \ , \tag{17}$$

$$I_{lk} = \sum_{j=1}^{l} \frac{R_{jk}}{N_{jk}} \ , \tag{18}$$

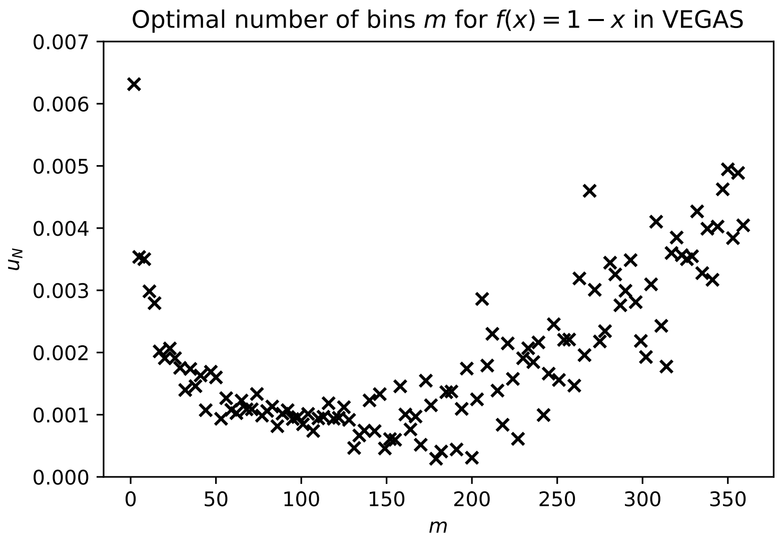
where $l = 1, \ldots, m$ and $k = 1, \ldots, n$. Finally, one defines n piecewise linear and strictly increasing functions $i^k(x^k)$ similar to $h^k(y^k)$, mapping x_l^k to I_{lk}. Then for the next iteration, the new x_l^k in equation 12 are found by solving the equation

$$i^k(x_l^k) = i^k(1)\frac{l}{m} \ , \tag{19}$$

where $l = 0, \ldots, m$ and $k = 1, \ldots, n$. Since the bins on the y^k-axes are fixed, the number of points will on average be the same in each bin on the x^k-axes. When all R_{jk}/N_{jk} are equal, the graphs of $i^k(x^k)$ approach a straight line and x_l^k no longer change.

2.4.1 Optimal number of bins in VEGAS

A high density of points in each part of the integration domain is a preferred outcome. For a fixed number of points N in VEGAS, one would expect to have a large u_N for a few bins, but likewise for many bins, since the density in a single bin will be low in both cases due to each bin having on average the same number of points. Clearly, u_N is minimized for a specific m given N points and an integrable function. Figure 2 shows that the minimum error in VEGAS is reached for different number of bins when integrating over a 1D and 2D domain, approximately 175 and 125 respectively. The fewer bins in 2D is due to the fact that, when adding another axis, each prior bin grows in height, and thus the sparseness of the points increases.



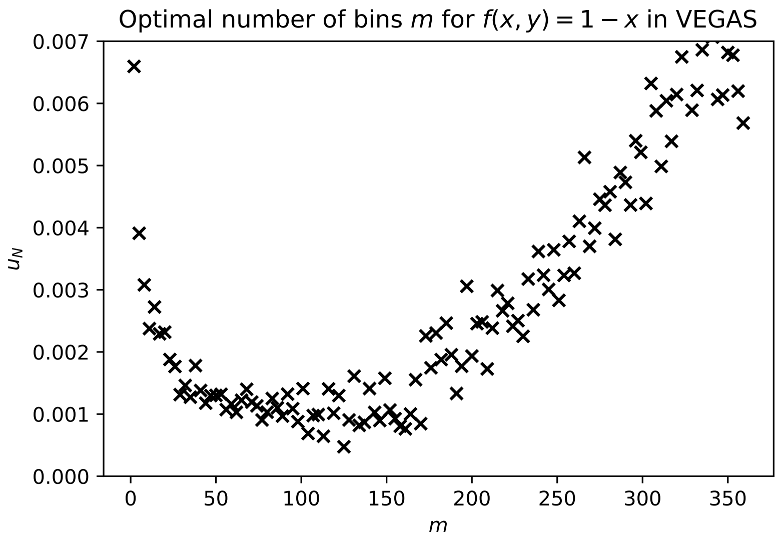


Figure 2: The optimal number of bins m in VEGAS when integrating $f(x) = 1 - x$ and $f(x, y) = 1 - x$ over the unit interval and square respectively after five iterations and $N = 1000$.

2.5 VEGAS squared

Suppose the characteristic regions of a function's graph do not align with the coordinate axes, but for example with the hypersurface $y = x$. Moreover, suppose the function is only large within a finite space close to this curve. A Gaussian function satisfies these criteria, specifically $G_1(x, y)$ with $a = 500, b = -495$, $c = 500$, $A = 3$ and $B = 0$ in the general, 2D

Gaussian function $G(x,y)$ centered at $(0.5, 0.5)$,

$$G(x,y) = Ae^{-a(x-0.5)^2-2b(x-0.5)(y-0.5)-c(y-0.5)^2} + B \ . \tag{20}$$

The fact that $G_1(x,y)$ is a narrow Gaussian should make important sampling ideal, but in VEGAS it is implemented in such a way that the density of points in parts of the integration domain increase by changing the sizes of the bins on each axis, independently. Each bin on each axis also receives on average the same number of points. This culminates in a large number of points not landing within the narrow stretch where the function is large, see Figure 3. More points will be needed to reach a good approximation of the integral, but in higher dimensions, this may have little effect.

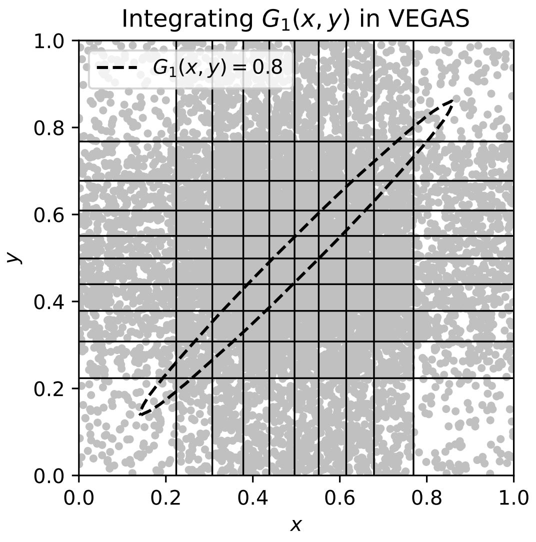


Figure 3: The bin sizes and distribution of points in VEGAS when integrating $G_1(x,y)$ over the unit square after five iterations, $m = 10$ and $N = 10000$. A grey dot represents a point.

The idea behind VEGAS squared is to find the parts of the integration domain where the i.v.'s make $G_1(x,y)$ large and only increase the density of points in those parts. The information where $G_1(x,y)$ is large is partly given by VEGAS, but only for the bins on each axis. It needs to be refined and the bins provide the structure for this refinement; the grid of rectangles in Figure 3. Then, to increase the density of points in those parts where $G_1(x,y)$ is large, the i.v.'s have to depend on each other.

If the bins are varied in size, only variable sized hyperrectangles which belong to small bins will have a high density of points. This will lead to a collection of hyperrectangles in the integration domain where the concentration of points is high, which is why VEGAS is good at approximating integrals where the characteristic regions of the integrand's graph do align with the coordinate axes – the bins on each axis may adjust to increasing or decreasing the dimensions of the characteristic regions of the integrand's graph and ensure that the corresponding part of the domain receives the main portion of the points, provided m and N are chosen reasonably. This is shown in Figure 4, where $G_2(x, y)$ is the rotated version of $G_1(x, y)$, that is with $a = 5, b = 0, c = 995$, $A = 3$ and $B = 0$ in equation 20. For $G_1(x, y)$, however, variable sized bins have to be rejected, mainly due to the variable sized grid of rectangles they give rise to; tracking where $G_1(x, y)$ is large becomes problematic without a fixed, common area for each rectangle, in which case the rectangles become squares.

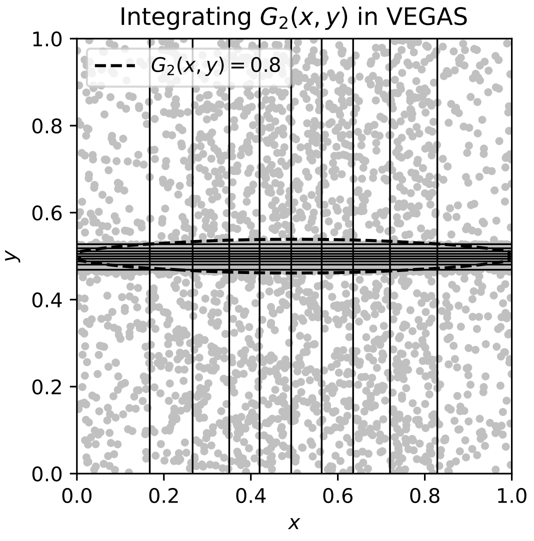


Figure 4: The bin sizes and distribution of points in VEGAS when integrating $G_2(x, y)$ over the unit square after five iterations, $m = 10$ and $N = 10000$. When the dimensions of the characteristic region enclosed by the level curve change, so do the bins correspondingly.

When the bin sizes are fixed, the points need to be, possibly differently, distributed in the bins on each axis, whereas in VEGAS the points were on average the same for each variable sized bin. Clearly, these are equivalent approaches in controlling the density of points over the integration domain, and thus one may expect the advantages in one approach to be the disadvantages in the other and vice versa. For example, if the size of a whole number of bins do not match the dimensions of the domain where $G_2(x, y)$ is large, there will be squares in which $G_2(x, y)$ is both large and small. These squares will be filled with points, but only a fraction of them effectively determine the resulting, poor approximation.

As an intermediate step, the distribution of points for the bins on each axis may be determined similar to the sizes of the bins in VEGAS; through the integral, i.e. the sample mean, of the function in each bin. Adopting this approach, the integral in each bin remains almost unchanged after the first iteration, and using it to optimize the distributions for the next iteration will have no effect. To increase the density of points in a given hypercube in the integration domain where the function is large, the distributions have to redetermine each other.

In VEGAS, an evaluation of the integrand at a point lying in a hyperrectangle of the integration domain is associated with the bins the hyperrectangle belongs to. However, the value of the integrand could also be associated with the rectangles in each plane corresponding to two i.v.'s. Applying this to an integration domain made up of hypercubes, one can study how the i.v.'s are correlated, pairwise. These are the so-called 2D correlations and they determine the distributions for each r.v. corresponding to an i.v. Not all distributions can determine each other; a natural choice is to let each added i.v. depend on the existing ones, making one i.v. independent of all the others. Its r.v. has a distribution determined by the intermediate step suggested in the previous paragraph.

To introduce what the 2D correlations are, consider a 2D domain and let i be an observation of the discrete r.v X indicating the bin in which the random number x' of the i.v. x is found and whose distribution is determined by the integral in each bin on the x-axis. The distribution of the discrete r.v. Y indicating the bin of the random number y' of the i.v. y is then determined by the integral in each square above the bin i on the x-axis. Thus y' will most likely be found in a bin where the function is large for both i.v.'s, which is the desired outcome.

With m bins in nD, the above implies that one must track the integral in $\binom{n}{2}m^2$ squares, which is significantly larger than the nm bins VEGAS tracks the integral in. The curse of dimensionality ensures that this number grows exponentially if one tracks correlations beyond 2D correlations, which is not feasible. Note also from the above that in 3D and in all higher dimensions, the distribution of the r.v. corresponding to the added i.v. is determined by the 2D correlations collected from each existing i.v. These must be combined into a total correlation.

2.6 2D correlations

The 2D correlations are no more than conditional probabilities and how to obtain a total correlation follows from two related concepts, namely conditional independence and Bayes' theorem. In the following definitions and theorem, it is assumed a probability space is given; an outcome space Ω, a σ-algebra $\mathcal{F}$ and a probability measure P. A, B and C are elements of $\mathcal{F}$.

Definition 21. If $P(B) \neq 0$, then the conditional probability of A given B, $P(A|B)$, is defined as

$$P(A|B) = \frac{P(A \cup B)}{P(B)} \ . \tag{22}$$

Definition 23. If $P(C) \neq 0$, A and B are conditionally independent given C if

$$P(A \cap B|C) = P(A|C)P(B|C) \ . \tag{24}$$

Theorem 25. *If $P(B) \neq 0$, then the conditional probability $P(A|B)$ is given by*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \ . \tag{26}$$

For the r.v.'s X and Y of the next to last paragraph in the previous subsection, given the observation i of X, the 2D correlation $P(Y = j|X = i)$ is defined as the integral in the square above bin i and bin j, on the x- and y-axes respectively, divided by the integral in bin i. Note that it is a function of j only. Furthermore, the r.v.'s corresponding to the existing i.v.'s will be conditionally independent given the r.v. of an added i.v. Thus, in 3D, if Z is the counterpart to X and Y, given the observations i and j of X and Y respectively, and in turn the 2D correlations $P(Z = k|X = i)$ and $P(Z = k|Y = j)$, the total correlation $P(Z = k|X = i \cap Y = j)$ is given by

$$\begin{aligned} P\left(Z = k|X = i \cap Y = j\right) &= \frac{P(X = i \cap Y = j|Z = k)P(Z = k)}{P(X = i \cap Y = j)} \\ &= \frac{P\left(X = i|Z = k\right) P\left(Z = k\right) P\left(Y = j|Z = k\right) P\left(Z = k\right)}{P\left(Z = k\right) P\left(X = i \cap Y = j\right)} \\ &= \frac{P\left(X = i\right) P\left(Y = j\right)}{P\left(Y = j \cap Y = j\right)} \frac{P\left(Z = k|X = i\right) P\left(Z = k|Y = j\right)}{P\left(Z = k\right)} \\ &\propto \frac{P\left(Z = k|X = i\right) P\left(Z = k|Y = j\right)}{P\left(Z = k\right)} \ , \end{aligned} \tag{27}$$

where the first equality follows from Theorem 25, the second from Definition 23 and the third from Definition 21 [5]. The proportionality constant can be omitted since, in normalizing the probabilities, it is absorbed by the normalization constant. Equation 27 determines how the 2D correlations from different planes are combined and the generalization of it is straightforward.

2.7 VEGAS squared algorithm

In words, and like in VEGAS, there are two unit hypercubes; the x-cube, the integration domain, and the y-cube, where random points are generated in. However, the bins in the x-cube are now fixed, whereas those in the y-cube, the probabilities, are varied. As in VEGAS, the first iteration starts with a uniform distribution. One generates N random points in the y-cube and maps those to points in the integration domain. The integral is approximated by the sample mean and the error by the standard error. For each iteration, one stores the integral in each square in each plane. This information is then used to update the bin sizes in the y-cube when determining the bins of each component of a generated point. This will lead to squares with few points corresponding to where the function is small and vice versa.

In symbols, consider the nD unit hypercube for $n \geq 2$ and let $f(x^1, \ldots, x^n)$ be an integrable function in this domain, where $x^k = x^k(x^1, \ldots, x^{k-1})$ denotes the k'th coordinate. Divide the interval $[0, 1]$ for each coordinate in m bins of length $1/m$. Let y^k denote the k'th coordinate in another unit hypercube, and divide the interval $[0, 1]$ for all of these in m variable sized bins. Define n piecewise linear and strictly increasing functions $h^k(y_l^k)$ by

$$h^k(y_l^k) = \frac{l}{m} \ , \tag{28}$$

where $l = 0, \dots, m$ and $k = 1, \dots, n$. Thus, the point y_l^k on every y^k-axes maps to the point l/m on every x^k-axes, each marking the possible end and beginning of a variable sized and fixed bin respectively. If y^k is given and y is a random point in the interval $[0, 1]$ satisfying $y_{l-1}^k < y < y_l^k$, then

$$\left.\frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k}\right|_{y^k=y} = \frac{1}{(y_l^k - y_{l-1}^k)m} \ . \tag{29}$$

The first iteration starts with $y_l^k = l/m$. A set $\mathcal{Y}_N$ of N random points $\mathbf{y}$ are generated in the unit hypercube $0 \le y^k \le 1$. The integral of $f(x^1, \dots, x^n)$ is approximated by I_N in equation 9, and the error u_N by equation 10,

$$I_N = \frac{1}{N} \sum_{\mathbf{y} \in \mathcal{Y}_N} f(h(y^1), \dots, h(y^n)) \prod_{k=1}^{n} \frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k} \ , \tag{30}$$

$$u_N \approx \frac{1}{\sqrt{N}} \sqrt{\left(\sum_{\mathbf{y} \in \mathcal{Y}_N} \frac{\left(f(h(y^1), \dots, h(y^n)) \prod_{k=1}^{n} \frac{\mathrm{d}h^k(y^k)}{\mathrm{d}y^k} \right)^2}{N} \right) - I_N^2} \ . \tag{31}$$

For each point $\mathbf{y}$ and its components y, the cumulative sum of the value of the integral less than a factor $\prod_{k=1}^{n} = 1/(y_l^k - y_{l-1}^k)$ as well as the number of points in a column above a given square in all the planes of the unit hypercube $0 \le x^k \le 1$ are stored in the collections of arrays $R_{ij}^{12}, ..., R_{ij}^{uv}$ and $N_{ij}^{12}, ..., N_{ij}^{uv}$ respectively, for which there are $\binom{n}{2}$ arrays of each collection. If Y^k is a r.v. indicating the bin where the component y corresponding to y^k was generated in, and $R_{ij}^{(k-1)k}$ represents the array for the $x^{k-1}x^k$-plane, then

$$R_{ij}^{(k-1)k} = \sum_{\mathbf{y} \in \mathcal{Y}_N} 1\{Y^k = i \cap Y^{k-1} = j\} f(h(y^1), \dots, h(y^n)) \frac{1}{m^n} \ , \tag{32}$$

$$N_{ij}^{(k-1)k} = \sum_{\mathbf{y} \in \mathcal{Y}_N} 1\{Y^k = i \cap Y^{k-1} = j\} \ , \tag{33}$$

where $0 \le i, j \le m - 1$ and $1 \le u < v \le n$. In the $x^{k-1}x^k$-plane, the integral in each square $I_{ij}^{(k-1)k}$ is simply $R_{ij}^{(k-1)k}/N_{ij}^{(k-1)k}$. For the next iteration, $y_{l+1}^1 = y_l^1 + C^1 \sum_i^{m-1} I_{il}^{12}$ in equation 28, where $y_0^1 = 0$ and C^1 normalizes the sum of $\sum_i^{m-1} I_{il}^{12}$ over l to 1. The unnormalized correlation $P(Y^k = i | Y^{k-l} = j)$ is defined as

$$P(Y^k = i | Y^{k-l} = j) = I_{ij}^{(k-l)k} \ , \tag{34}$$

where $0 \leq i, j \leq m-1$ and $l = 1, ..., k-1$. In the next iteration, given $Y^1 = j$, it determines y_l^2. Then, given $Y^2 = i$, it together with $Y^1 = j$ determines y_l^3. In general, and using equation 27,

$$y_{l+1}^k = y_l^k + C^k \frac{P\left(Y^k = l | Y^{k-1} = j'\right) 1\{Y^{k-1} = j'\} \cdots P\left(Y^k = l | Y^1 = j\right) 1\{Y^1 = j\}}{P\left(Y^k = l\right)^{n-2}} \tag{35}$$

where $y_0^k = 0$, $0 \leq l, j, j' \leq m-1$ and C^k normalizes the sum of the total correlation over l to 1.

3 Result

Table 1 shows the value of u_N for VEGAS and VEGAS squared respectively when integrating $G_1(x, y)$ over the unit square. The error in VEGAS squared is approximately less by a factor of 2.5 after the first iteration. Figure 5 shows the distribution of points over the integration domain after the fifth iteration. The same results were obtained when integrating $G_1(x, y)$ over the unit cube, and replacing the x and y coordinate respectively by the coordinate z.

Table 1: u_N in VEGAS and VEGAS squared respectively when integrating $G_1(x, y)$ over the unit square.

	VEGAS	VEGAS squared
Iteration	u_N	u_N
1	± 0.004106	± 0.004301
2	± 0.003638	± 0.001452
3	± 0.003512	± 0.001507
4	± 0.004176	± 0.001485
5	± 0.003736	± 0.001490

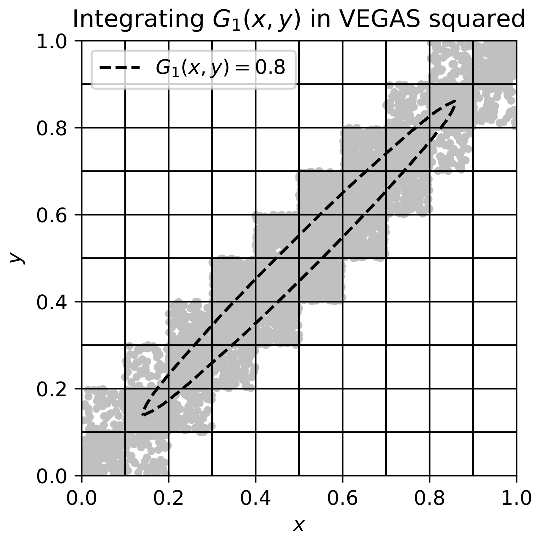


Figure 5: The bin sizes and distribution of points in VEGAS squared when integrating $G_1(x, y)$ over the unit square after five iterations, $m = 10$ and $N = 10000$.

4 Conclusion

Figure 5 confirms that the algorithm of VEGAS squared works in 2D as intended. That similar figures can be produced in 3D is a testament to that the algorithm can be applied in higher dimensions. Table 1, on the other hand, confirms the intention of the thesis – to improve the approximation of integrals in which the i.v.'s display correlations. It should be noted that, when integrating functions like $G_2(x, y)$, whose graph's characteristic region does align with the coordinate axes, the approximation and error in VEGAS squared will greatly depend on the number of bins m. Unlike in VEGAS, the fixed bins can not adjust to the dimensions of the domain where $G_2(x, y)$ is large, leading to a decreased control of the distribution of points and ultimately a less accurate approximation. $G_1(x, y)$ is thus an ideal case and it remains to test how VEGAS squared performs when integrating functions, possibly not Gaussian, with correlations between the variables other than simply $y = x$.

5 References

[1] G Peter Lepage. "A new algorithm for adaptive multidimensional integration". In: *Journal of Computational Physics* 27.2 (1978), pp. 192–203. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(78)90004-9. URL: http://www.sciencedirect.com/science/article/pii/0021999178900049.

[2] P. Nason. *MINT: a Computer Program for Adaptive Monte Carlo Integration and Generation of Unweighted Distributions*. 2007. arXiv: 0709.2085 [hep-ph].

[3] Magnus Wiktorsson. *Monte Carlo and Empirical Methods for Stochastic Inference (MASM11/FMSN50)*. http://www.maths.lth.se/matstat/kurser/fmsn50masm11/2020/material/L1.pdf. Accessed: 2020–11-07. 2020.

[4] Dragi Anevski. *A Concise Introduction to Mathematical Statistics*. Lund, Sweden: Studentlitteratur, 2017.

[5] Till Hoffmann (https://stats.stackexchange.com/users/17643/till-hoffmann). *Proper way to combine conditional probability distributions of the same random variable conditioned on a discrete variable ? (based on assumptions)*. Cross Validated. URL:https://stats.stackexchange.com/q/112361 (version: 2020-07-23). eprint: https://stats.stackexchange.com/q/112361. URL: https://stats.stackexchange.com/q/112361.

Appendix Python Code

VEGAS

```python
import numpy as np


it=5                                        #it>=1
N=10000                                     #N>=1
m=10                                        #m>=2
n=2                                         #n>=1


x_i=np.linspace(np.zeros(n),np.ones(n),m+1)
y_i=np.linspace(0,1,m+1)


A=3
B=0
a=500
b=-990
c=500


def f(arr):                                 #define appropriately according to n
    x=arr[0]
    y=arr[1]
    return A*np.exp(-a*(-0.5 + x)**2-b*(-0.5 + x)*(-0.5 + y)-c*(-0.5 + y)**2)+B


for dummy1 in range(0,it):


  I=0
  u=0
  N_l=np.zeros([m,n])
  R_l=np.zeros([m,n])


  for dummy2 in range(0,N):


    y=np.random.uniform(0,1,n)
    Y=np.argmax(np.repeat(y_i[:, np.newaxis], len(y), axis=1) >= y, axis=0)


    h_y_k=np.zeros(n)                   #equation 12
    p=1
    for i,k in zip(Y,range(0,n)):
      h_y_k[k]=x_i[i-1,k]+(x_i[i,k]-x_i[i-1,k])*m*(y[k]-(i-1)/m)
      p*=x_i[i,k]-x_i[i-1,k]


    I+=f(h_y_k)*p*m**n                  #equation 14 and 15
    u+=(f(h_y_k)*p*m**n)**2


    for i,k in zip(Y,range(0,n)):       #equation 16 and 17
      R_l[i-1,k]+=f(h_y_k)*p
      N_l[i-1,k]+=1                     #exiting 1st for-loop


  I_N=I/N                               #equation 14 and 15
  u_N=np.sqrt((u/N-I_N**2)/N)
  print("{}      {:3f} +\- {:3f}".format(dummy1+1,I_N,u_N))


  N_l=np.where(N_l>0,N_l,1)             #equation 18
  R_l=np.where(R_l>10**-8,R_l,10**-8)
  I_l=np.r_[np.zeros([1,n]),np.cumsum(np.divide(R_l,N_l),axis=0)]


  x_i_new=np.zeros([m+1,n])             #equation 19
  for k in range(0,n):
    for l in range(0,m+1):
      r=l/m*I_l[-1,k]
      i=np.argmax(I_l[:,k]>=r)
      x_i_new[l,k]=x_i[i-1,k]+((r-I_l[i-1,k])/(I_l[i,k]-I_l[i-1,k]))*(x_i[i,k]-x_i[i-1,k])
  x_i=x_i_new                           #exiting 2nd for-loop
```

VEGAS squared

```python
import numpy as np


it=5                                    #it>=1
N=10000                                 #N>=1
m=10                                    #m>=2
n=2                                     #n>=2


y_i=np.linspace(np.zeros(n),np.ones(n),m+1)
x_i=np.linspace(0,1,m+1)
I_ij=np.ones([n-1,n-1,m,m])


A=3
B=0
a=500
b=-990
c=500


def f(arr):                             #define appropriately according to n
    x=arr[0]
    y=arr[1]
    return A*np.exp(-a*(-0.5 + x)**2-b*(-0.5 + x)*(-0.5 + y)-c*(-0.5 + y)**2)+B


for dummy1 in range(0,it):


  I=0
  u=0
  N_ij=np.zeros([n-1,n-1,m,m])
  R_ij=np.zeros([n-1,n-1,m,m])


  for dummy2 in range(0,N):


    y=np.random.uniform(0,1,n)
    Y=[1]*n


    tot_prob_Y1=np.sum(I_ij[0,0],axis=0)/np.sum(np.sum(I_ij[0,0],axis=0))
    y_i[:,0][y_i[:,0]>0]=np.cumsum(tot_prob_Y1)
    Y[0]=np.argmax(y_i[:,0]>=y[0])
    for k in range(0,n-1):              #equation 35
      corr=np.vstack([I_ij[i,k,:,Y[i]-1] for i in range(0,k+1)])
      tot_prob_Yi=np.sum(I_ij[0,k],axis=1)
      tot_corr=(np.prod(corr,axis=0)/tot_prob_Yi**k)[::-1]
      y_i[:,k+1][y_i[:,k+1]>0]=np.cumsum(tot_corr/np.sum(tot_corr))
      Y[k+1]=np.argmax(y_i[:,k+1]>=y[k+1])


    h_y_k=np.zeros(n)                   #equation 28
    p=1
    for i,k in zip(Y,range(n)):
      h_y_k[k]=x_i[i-1]+(y[k]-y_i[i-1,k])*(x_i[i]-x_i[i-1])/(y_i[i,k]-y_i[i-1,k])
      p*=1/(m*(y_i[i,k]-y_i[i-1,k]))


    I+=f(h_y_k)*p                       #equation 30 and 31
    u+=(f(h_y_k)*p)**2


    for r in range(0,n-1):              #equation 32 and 33
      for k in range(r,n-1):
        R_ij[r,k,m-Y[k+1],Y[r]-1]+=f(h_y_k)*(1/m)**n
        N_ij[r,k,m-Y[k+1],Y[r]-1]+=1    #exiting 1st for-loop


  I_N=I/N                               #equation 30 and 31
  u_N=np.sqrt((u/N-I_N**2)/N)
  print("{}      {:3f} +\- {:3f}".format(dummy1+1,I_N,u_N))


  N_ij=np.where(N_ij>0,N_ij,1)
  R_ij=np.where(R_ij>10**-8,R_ij,10**-8)
  I_ij=np.divide(R_ij,N_ij)             #exiting 2nd for-loop
```