

Numerical Integration and Monte Carlo Integration

Anders W. Sandvik, Department of Physics, Boston University

1 Introduction

A common numerical task in computational physics is to evaluate integrals that cannot be solved analytically. Numerical integration is an important subfield of numerical analysis, and a wide variety of elaborate schemes have been developed for treating many different types of integrals, including difficult ones with integrable singularities.

Here we will begin by deriving the basic trapezoidal and Simpson's integration formulas for functions that can be evaluated at all points in the integration range, i.e, there are no singularities in this range. We will also show how the remaining discretization errors present in these low-order formulas can be systematically extrapolated away to arbitrarily high order; a method known as Romberg integration. Multi-dimensional integrals can be calculated "dimension-by-dimension" using these one-dimensional methods.

When one encounters a difficult case that cannot be solved with the elementary schemes discussed here, e.g., due to integrable singularities that cannot be transformed away, the best approach may be to search for an appropriate "canned" subroutine in a software library, e.g., in the book "Numerical Recipes", or the extensive software repository GAMS (Guide to Available Mathematical Software), available on-line from NIST at gams.nist.gov.

In addition to elementary numerical integration of one- and multi-dimensional integrals, we will here also consider *Monte Carlo integration*, which is a stochastic scheme. i.e., one based on random numbers. Generation of random numbers (or, more accurately, pseudo-random numbers) is also an important element of many computational physics techniques; we will here discuss the basics of *random number generators*.

2 Elementary algorithms for one-dimensional integrals

Elementary schemes for evaluating an integral of the form

$$I = \int_a^b f(x)dx, \tag{1}$$

where $f(x)$ can be evaluated at all points in the closed integration range $[a, b]$, are based on the function values $f_i = f(x_i)$ at evenly spaced points between a and b , including also the limiting points a, b themselves. Approximating the function between points $x_i, x_{i+1}, \dots, x_{i+n}$ with an interpolating polynomial of order n , the integral can be calculated. In Fig. 1 this scheme is illustrated for the cases $n = 1$ and $n = 2$. The point spacing is denoted h ;

$$h = x_{i+1} - x_i. \tag{2}$$

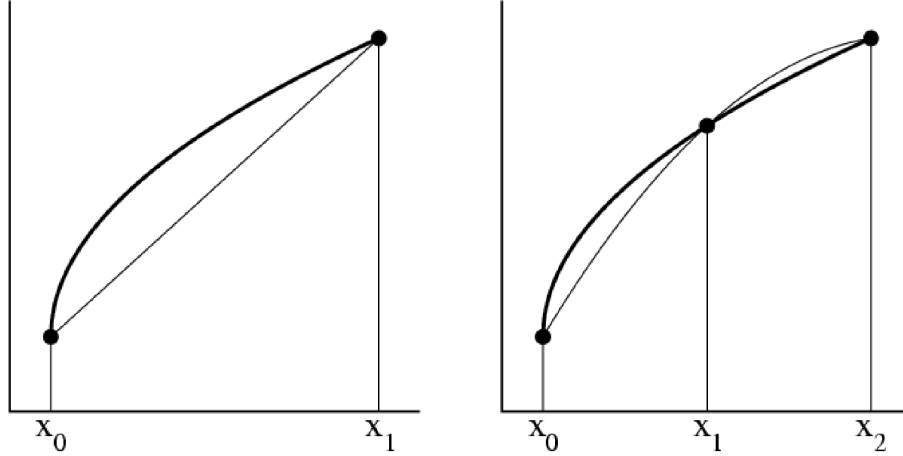


Figure 1: Graphical representation of the trapezoidal (left) and Simpson's (right) integration schemes. The bold curve shows the function $f(x)$ that is to be integrated. The thin curves are the interpolating polynomials of order $n = 1$ (left) and $n = 2$ (right).

For $n = 1$, the interpolating polynomial for an interval $[x_0, x_1]$ is simply a straight line,

$$f(x_0 + \delta) = a + b\delta, \quad 0 \leq \delta \leq h, \quad (3)$$

where clearly $a = f_0$ and $b = (f_1 - f_0)/h$. The integral in the interval is hence approximated by

$$\int_{x_0}^{x_1} f(x)dx = h(a + bh/2) = \frac{h}{2}[f_0 + f_1] + O(h^3), \quad (4)$$

where $O(h^3)$ denotes a correction (error of the $n = 1$ estimate) of order h^3 . Clearly this result could have been obtained directly as the area under the trapezoid between x_0 and x_1 . The reason for carrying out the formal manipulations above was that this easily generalizes to higher orders.

Consider the case $n = 2$, where the function between x_0 and x_2 is approximated as

$$f(x_0 + \delta) = a + b\delta + c\delta^2, \quad 0 \leq \delta \leq 2h, \quad (5)$$

where the coefficients a, b , and c have to be determined so that $f(x_0) = f_0$, $f(x_0 + h) = f_1$, and $f(x_0 + 2h) = f_2$, i.e.,

$$\begin{aligned} f_0 &= a \\ f_1 &= a + bh + ch^2 \\ f_2 &= a + 2bh + 4ch^2, \end{aligned} \quad (6)$$

which is easily solved to give

$$\begin{aligned} a &= f_0, \\ b &= -(3f_0 - 4f_1 + f_2)/2h, \\ c &= (f_0 - 2f_1 + f_2)/2h^2. \end{aligned} \quad (7)$$

The $n = 2$ approximation to the integral is hence

$$\int_{x_0}^{x_2} f(x)dx = 2h(a + bh + (4/3)ch^2) = \frac{h}{3}[f_0 + 4f_1 + f_2] + O(h^5), \quad (8)$$

which is called Simpson's rule. One might have expected the error in this case to be $O(h^4)$, but because of the symmetry of the formula there is a cancellation of the corrections at this order, and the error is in fact $O(h^5)$.

In order to estimate an integral over an extended range $[a, b]$ discretized in N intervals ($N + 1$ points), one writes it as a sum of integrals of the form (4) or (8), which give, respectively, the extended trapezoidal rule,

$$\int_{x_0}^{x_N} f(x)dx = h\left(\frac{1}{2}f_0 + f_1 + f_2 + \dots + f_{N-1} + \frac{1}{2}f_N\right) + O(h^2), \quad (9)$$

and the extended Simpson's rule;

$$\int_{x_0}^{x_N} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \dots + 2f_{N-2} + 4f_{N-1} + f_N) + O(h^4). \quad (10)$$

In Simpson's rule N must be even. Note that the errors in these extended formulas are larger by a factor $1/h$ than in the corresponding single-segment formulas (4) and (8), because the number of points $N \propto 1/h$.

There are also formulas—open integration formulas—that do not involve the end-points of integration. In the so-called mid-point rule, all the points are weighted equally;

$$\int_{x_0}^{x_N} f(x)dx = h(f_{1/2} + f_{3/2} + \dots + f_{N-3/2} + f_{N-1/2}) + O(h^2). \quad (11)$$

Other low-order open formulas are

$$\int_{x_0}^{x_N} f(x)dx = h\left(\frac{3}{2}f_1 + f_2 + f_3 + \dots + f_{N-2} + \frac{3}{2}f_{N-1}\right) + O(h^2), \quad (12)$$

$$\int_{x_0}^{x_N} f(x)dx = h\left(\frac{23}{12}f_1 + \frac{7}{12}f_2 + f_3 + f_4 + \dots + f_{N-3} + \frac{7}{12}f_{N-2} + \frac{23}{12}f_{N-1}\right) + O(h^3), \quad (13)$$

where all the interior points are weighted with 1. Open formulas can be used in cases where there are integrable singularities at the end-points, but note that in such cases the discretization errors given above do not apply—there will be larger contributions to the errors coming from the singularities. If a singularity cannot be transformed away, it may be possible to subtract off an asymptotic divergent form, which itself may be analytically integrable, leaving an integral with no singularities that can be treated numerically. In cases where such tricks are not possible, the above open formulas (or ones of higher order) can be used but the convergence may be very slow. More sophisticated methods may then be necessary.

For many integrals without singularities, Simpson's formula (10) and the corresponding open formula (13) are perfectly adequate and there is no need to go to higher orders. However, if a large number of integrals have to be evaluated to high accuracy, higher-order formulas can be used. Although such formulas can be derived using the above scheme with $n = 3, 4, \dots$, it is in practice often better to use the Romberg scheme discussed next.

3 Romberg integration

Romberg's method is based on the fact that the error in the extended trapezoidal rule is a polynomial in h , with the leading-order term $\propto h^2$. In fact, the polynomial contains only even powers of h , as discussed, e.g., in *Numerical Recipes*. The error is hence a polynomial in h^2 . Furthermore, if one uses the trapezoidal rule several times and doubles the number of points in each step, one can start from the previous result divided by two and only add the contributions from the new points (which fall between the ones of the previous step). Explicitly, if the result of the trapezoidal rule in step k is $I(k)$, then the formula for the next step with twice as many intervals is

$$I_{k+1} = \frac{I_k}{2} + h(f_1 + f_3, \dots, f_{N-1}), \quad (14)$$

where h and N are those of step $k + 1$. The doubling procedure is illustrated in the upper part of Fig. 2. This trick speeds up the calculation by a factor of 2.

Starting with some N_0 (which can even be the smallest possible; $N_0 = 1$), one generates a series of approximants $I_0, I_1, \dots, I_k, \dots, I_n$ with discretizations $h = h_0/2^k$. Knowing that the approximants approach the true integral I_∞ as a polynomial in $h^2 \propto 1/2^{2k}$, one can construct an interpolating order- n polynomial in $1/2^{2k}$ that goes through all the approximants. Evaluating (extrapolating) this polynomial at $k = \infty$ ($h = 0$), one obtains an approximation to the integral with an error of order $h^{2(k+1)}$.

To illustrate this in the simplest case, $n = 1$, the two integrals and their errors are

$$\begin{aligned} I_0 &= I_\infty + \epsilon h_0^2 + \dots, \\ I_1 &= I_\infty + (\epsilon/4)h_0^2 + \dots, \end{aligned} \quad (15)$$

where h_0 is the interval size used in the initial step. Interpolating the leading h -dependence by a straight line in h^2 ,

$$I(h) = I_\infty + bh^2, \quad (16)$$

we have the equation set

$$\begin{aligned} I_0 &= I_\infty + bh_0^2, \\ I_1 &= I_\infty + bh_0^2/4, \end{aligned} \quad (17)$$

which when solved for I_∞ and b gives

$$I_\infty = \frac{4}{3}I_1 - \frac{1}{3}I_0. \quad (18)$$

Using Eqs. (9) and (14), one can easily see that this result is in fact exactly equivalent to the extended Simpson's formula (10).

To go to higher orders, we need the polynomial which interpolates between n integral approximants. There is a simple general formula, called Lagrange's formula, for a polynomial $P(x)$ of order n which goes through $n + 1$ points (x_i, y_i) :

$$P(x) = \sum_{i=0}^n y_i \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}. \quad (19)$$

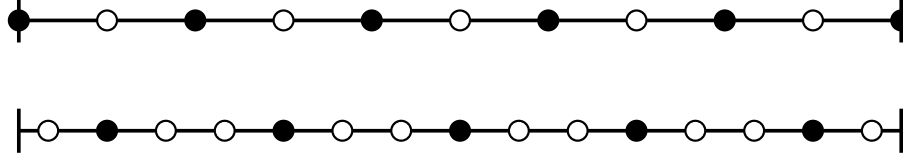


Figure 2: Doubling of the number of points in the trapezoidal formula (top) and tripling of the number of points in the mid-point formula (bottom). The solid and open circles indicate points included in the k th and $(k + 1)$ th step, respectively, and the integration boundaries are indicated with vertical marks.

Here the products run over the factors with $k = 0, 1, \dots, n$, except for $k = i$. With this construction, the polynomial is clearly of order n . For $x = x_l$ (one among the points x_0, \dots, x_n), only the term with $i = l$ is non-zero; all other terms vanish because there is a factor $x_l - x_l$ in the products with $i \neq l$. In the surviving $i = l$ term, all the numerators and denominators in the product cancel to 1, and so this term is by construction equal to y_l . Hence the polynomial indeed goes through all the $n - 1$ points.

In general, Lagrange's formula is not the best way to construct an interpolating polynomial; alternative methods, which suffer less from numerical instabilities, are discussed, e.g., in *Numerical Recipes*. However, in the case of extrapolating integral approximants, we are only interested in the value of the polynomial $P(x = h^2)$ at $x = 0$. The x -values values at which we have evaluated the integrals are given by $x_k = h_0/2^{2k}$, and hence Lagrange's formula reduces to

$$I_\infty = \sum_{i=0}^n I_i \prod_{k \neq i} \frac{-h_0 2^{-2k}}{h_0(2^{-2i} - 2^{-2k})} = (-1)^n \sum_{i=0}^n I_i \prod_{k \neq i} \frac{1}{2^{2(k-i)} - 1}. \quad (20)$$

This expression can be evaluated without numerical instabilities up to the orders needed in most cases; typically not higher than $k \approx 10$. The error convergence $O(h^{2(k+1)})$ is very fast indeed, and Romberg's scheme thus performs very well for the vast majority of integrals without singularities.

Romberg's method can also be used with the mid-point formula (11) instead of the trapezoidal rule, but in that case it is not possible to double the number of points at each step and make use of the calculations carried out with the previous points. However, one can instead triple the number of points, as illustrated in the lower part of Fig. 2. The error then decreases by a factor of 9 in each step, and the Lagrange formula (20) can be used with $2^{2(k-i)}$ replaced by $3^{2(k-i)}$.

In the case of integrable singularities at the end-points, the Romberg scheme breaks down since the errors of the trapezoidal formula then no longer scale as h^2 . However, it may still be useful to carry out a calculation with the mid-point formula (or, preferably, a higher-order open formula) in several steps, and to try to extract the rate of convergence and then do the extrapolation to $h \rightarrow 0$ with this form (if the asymptotic form of the divergence is known, the scaling of the error can be computed explicitly). But, on the other hand, there are more sophisticated schemes for integrable singularities, e.g., *Gaussian quadrature* (see, e.g., *Numerical Recipes*), which is based on non-equally spaced points.

4 Multi-dimensional integrals

Algorithms for multi-dimensional integrals,

$$I = \int_{a_n}^{b_n} dx_n \cdots \int_{a_2}^{b_2} dx_2 \int_{a_1}^{b_1} dx_1 f(x_1, x_2, \dots, x_n), \quad (21)$$

can be carried out "dimension-by-dimension" using one of the one-dimensional formulas discussed above. Consider for simplicity a two-dimensional case,

$$I = \int_{a_y}^{b_y} dy \int_{a_x(y)}^{b_x(y)} dx f(x, y), \quad (22)$$

where we have also allowed the integration limits for the x -integral to depend on y , i.e., the integration range is non-rectangular. From the point of view of evaluating the y -integral, the x -integral can be considered as nothing more than a complicated evaluation of a function of y , i.e., we can write

$$I = \int_{a_y}^{b_y} dy F(y), \quad F(y) = \int_{a_x(y)}^{b_x(y)} dx f(x, y). \quad (23)$$

For fixed y , the function $F(y)$ is just a one-dimensional integral that can be computed using one of the methods discussed above (assuming now again that there are no singularities present).

The processor time needed to compute an N -dimensional integral using this type of method scales as M_1^N , where M_1 is the typical number of operations needed to compute a one-dimensional integral. Using an efficient integrator is clearly critically important for $N > 1$. In practice it is rarely feasible to compute an integral this way if N is larger than, roughly, $5 \sim 6$. However, Monte Carlo methods, to be discussed next, can be used also for very large N .

5 Monte Carlo integration

In physics (as well as in other fields where these methods are used), the term "Monte Carlo" refers to the use of random numbers (in principle, one could use a dice or a roulette wheel). Monte Carlo integration is the simplest of a wide range of "Monte Carlo methods", where averages are calculated using uniform random sampling (in other Monte Carlo methods the sampling can be biased in various ways for increased efficiency).

Monte Carlo integration is based on the simple fact that an integral can be expressed as an average of the integrand over the range, or volume, of integration, e.g., a one-dimensional integral can be written as

$$A = \int_a^b dx f(x) = (b - a) \langle f \rangle, \quad (24)$$

where $\langle f \rangle$ is the average of the function in the range $[a, b]$. A statistical estimate of the average can be obtained by randomly generating N points $a \leq x_i \leq b$ and calculating the arithmetic average

$$\bar{f} = \frac{1}{N} \sum_{i=1}^N f(x_i). \quad (25)$$

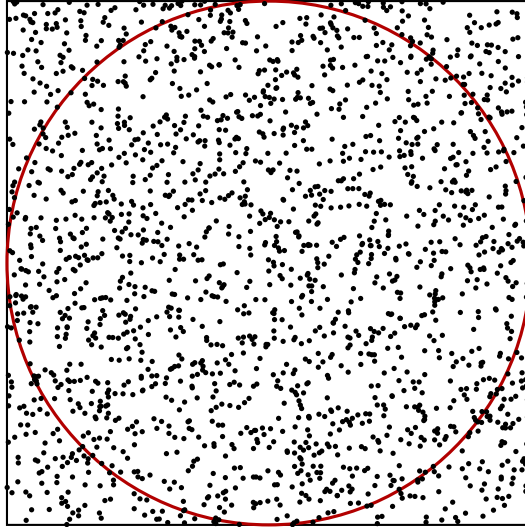


Figure 3: Illustration of the Monte Carlo integration method for finding the area of a circle. With the circle enclosed by a square, the fractional area inside the circle is estimated by generating points inside the square at random and counting the number of points that fall inside the circle. In the case shown, 2000 points were generated, and the fraction of points inside the circle is 0.791, which hence is the estimate of $\pi/4$ obtained in this calculation.

This estimate approaches the true average $\langle f \rangle$ as $N \rightarrow \infty$, with a statistical error (to be defined precisely below) which is proportional to $1/\sqrt{N}$. In one dimension, this rate of convergence is very slow compared to standard numerical integration methods on a mesh using N points. However, in higher dimensions the computational effort of numerical integration increases exponentially with the number of dimensions, whereas the error of the Monte Carlo estimate for an integral in any number of dimensions decreases as $1/\sqrt{N}$. Hence, for high-dimensional integrals Monte Carlo sampling can be more efficient.

The straight-forward unbiased Monte Carlo integration method outlined above only works well in practice if the integrand does not have sharp peaks that dominate the integral. Such peaks will be "visited" infrequently by random sampling and hence the majority of points in the sum (25) will be ones that in fact contribute little to the average. This implies a large prefactor in the $\propto 1/\sqrt{N}$ dependence of the statistical error. There are other Monte Carlo methods that are geared specifically to problems where the integral is dominated by a small fraction of the sampling space; we will discuss such importance sampling methods later. Here we will first analyze some aspects of simple Monte Carlo integration that will be useful for discussing importance sampling as well. We will assume that we have a good random number generator available, with which we can generate uniformly distributed random numbers. Random number generators are discussed in Appendix B.

A commonly used illustration of multi-dimensional Monte Carlo integration is the simple problem of calculating the area of a circle with radius 1. The circle's area can be written as an integral over a square of length 2 within which the circle is enclosed;

$$A = \int_{-1}^1 dy \int_{-1}^1 dx f(x, y), \quad (26)$$

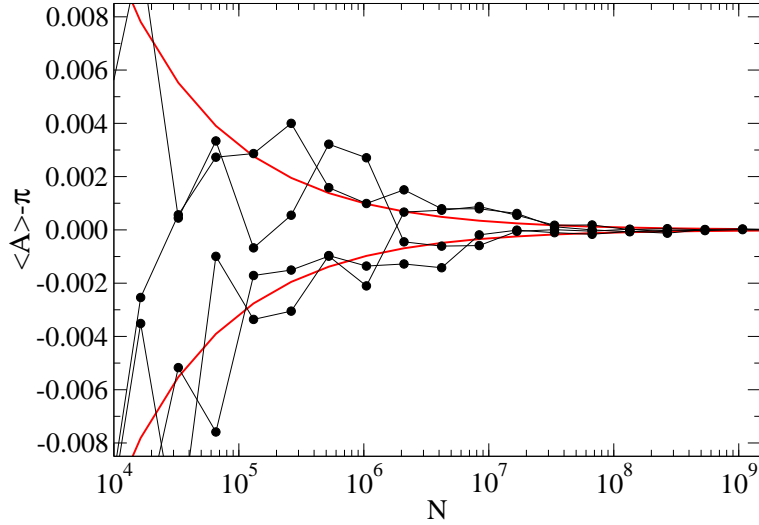


Figure 4: Deviation of the stochastic estimate of π from the exact value as a function of the number of points generated in four independent Monte Carlo integration runs. The smooth curves show the anticipated error scaling $\propto 1/\sqrt{N}$.

where the function is

$$f(x, y) = \begin{cases} 1, & \text{for } x^2 + y^2 \leq 1, \\ 0, & \text{for } x^2 + y^2 > 1. \end{cases} \quad (27)$$

The Monte Carlo integration in this case hence amounts to generating random number pairs, or “points”, (x_i, y_i) in the range $[-1, 1]$ and adding 1 or 0 according to the summation (25) for points falling inside and outside the circle, respectively. The final result is the integration volume, in this case 4, times the average. This procedure is illustrated in Fig. 3. The true area of the circle equals π , and hence this method gives a stochastic estimate of π . In Fig. 4 the average π is graphed as a function of the number of points as the calculation proceeds, for four independent runs. The fluctuations of the average become smaller when more points are included, as the relative contribution of each successive point decreases, and the deviation from the exact results decreases roughly as $1/\sqrt{N}$ as expected. Any deviation from π as $N \rightarrow \infty$ would be due to imperfect random numbers, but almost any random number generator is good enough for this kind of simple problem that no deviations beyond statistical can be detected even in extremely long runs.

Since Monte Carlo sampling can never give an exact result, because of the finite N , it is very important to have a good estimate of the statistical error. It is typically defined as one standard deviation of the probability distribution of the calculated average \bar{A} . If we repeat the calculation M times with the same number of points but with different (independent) sequences of random numbers (as illustrated in Fig. 4 for $M = 4$), we obtain M independent averages

$$\bar{A}_i = \frac{1}{N} \sum_{j=1}^N A_{i,j}, \quad (28)$$

where $A_{i,j}$ are the sampled function values for run i , $i = 1, \dots, M$. The statistical error associated with each of these individual averages is the standard deviation σ of the distribution of these

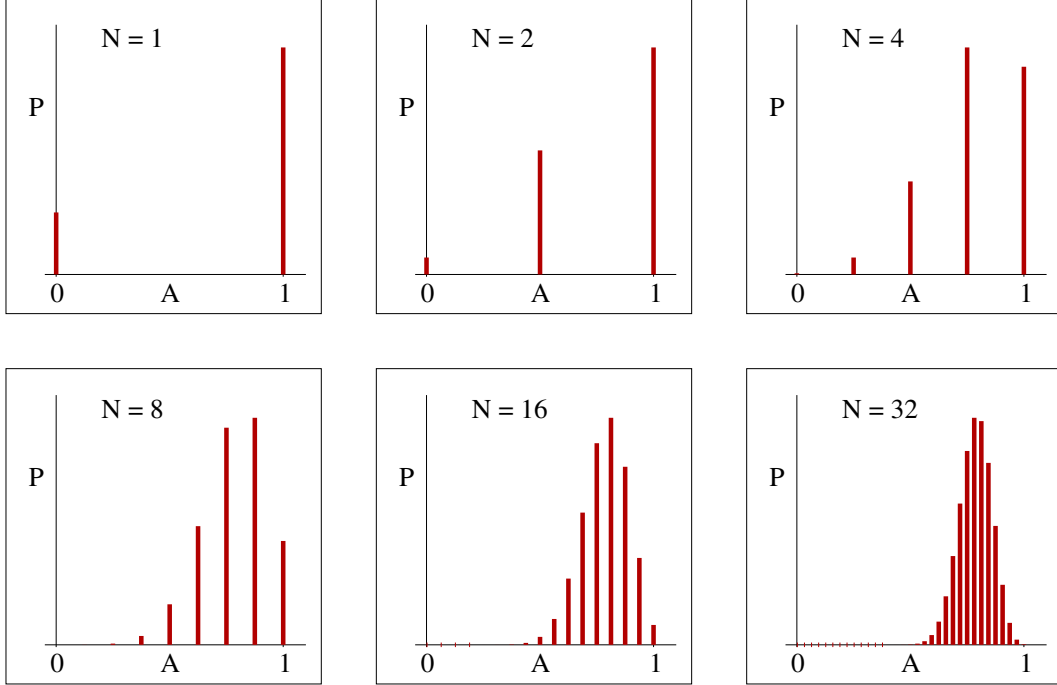


Figure 5: Expected distribution of the average over N samples in the circle area Monte Carlo integration, for $N = 1, 2, 4, 8, 16$ and 32 . The histograms are scaled in such a way that the highest bars have the same size in all the cases.

averages, which can be estimated using the generated data;

$$\sigma = \sqrt{\frac{1}{M} \sum_{i=1}^M (\bar{A}_i^2 - \bar{A}^2)}. \quad (29)$$

However, with M different estimates available, we can clearly obtain a better estimate by taking the average of all of them. We denote this full average \bar{A} ;

$$\bar{A} = \frac{1}{M} \sum_{i=1}^M \bar{A}_i. \quad (30)$$

The statistical uncertainty of this average (the error bar) is $\bar{\sigma} = \sigma/\sqrt{M-1}$ (where the use of $M = 1$ instead of M is discussed in most texts on statistics; it essentially reflects the fact that with just one estimate the uncertainty is completely undetermined), i.e.,

$$\bar{\sigma} = \sqrt{\frac{1}{M(M-1)} \sum_{i=1}^M (\bar{A}_i^2 - \bar{A}^2)}. \quad (31)$$

The result of M independent runs, which in this context normally are referred to as *bins* should thus be quoted as $\bar{A} \pm \bar{\sigma}$.

The standard deviation, as it is normally used in statistics, has its normal meaning only if the bin averages \bar{A}_i obey a gaussian distribution. In that case, we know, e.g., that the probability of

the true value of the integral being within the range $[\bar{A} - \bar{\sigma}, \bar{A} + \bar{\sigma}]$ (“inside the error bars”) is approximately 66%, and that the probability is $\approx 95\%$ for it to be within two error bars. While individual samples $A_{i,j}$ of the function normally follow some distribution which is far from gaussian, the “law of large numbers” guarantees that the distribution of bin averages (28) in fact becomes a gaussian when the number of points N used to calculate them becomes large. This is illustrated in Fig. 5 for the case of the circle area integration. For a single sample ($N = 1$) in each bin, the distribution is bimodal in this case, with probability $P = \pi/4$ for $A = 1$ and $1 - \pi/4$ for $A = 0$. For arbitrary N , the distribution is a binomial;

$$P(A) = \sum_{m=0}^N \frac{N!}{m!(N-m)!} \left(\frac{\pi}{4}\right)^m \delta(NA - m\pi/4). \quad (32)$$

Fig. 5 shows this distribution for several N , with the delta-functions represented by histogram bars of finite width. One can clearly see how a gaussian is gradually emerging; its width decreases as $1/\sqrt{N}$. If N is on the order of 100 or larger, one can in this case safely assume that the distribution is gaussian and use Eq. (29) to calculate the statistical error. In practice, the N used for each bin would be much larger. Note also that the number of bins M should be at least on the order of $10 \sim 100$, in order to constitute a sufficient statistical basis for calculating σ according to Eq. (31), and it should be noted that this is also only an estimate for the true standard deviation—error bars also have error bars. The final statistical error is of course $\sim 1/\sqrt{NM}$, and in practice it does not matter that much how the NM samples are divided into M bins and N samples per bin, as long as N is large enough to produce gaussian bin averages and M is at least 10. In practice, one should not choose M very large if the individual bin averages are also stored on disk, which is often useful in actual applications of Monte Carlo methods and is strongly advocated here.

We have already touched upon the fact that the simple Monte Carlo integration scheme becomes problematic if the integrand has sharp peaks in small regions of the integration space. This is reflected in a long tails in the distribution of the sampled function values, and this in turn implies that large N has to be used to obtain a bin distribution that approaches a gaussian, and the prefactor in the $1/\sqrt{N}$ scaling of the width of this distribution will also be large (implying a large final statistical error).

As a simple example illustrating these issues, we next consider a modification of the circle integration above. Instead of having a flat function $f(x, y) = 1$ inside the circle, we now let f have an integrable $r^{-\alpha}$ singularity at the center of the circle;

$$f(x, y) = f(r) = \begin{cases} r^{-\alpha}, & \text{for } r = \sqrt{x^2 + y^2} \leq 1, \\ 0, & \text{for } r > 1. \end{cases} \quad (33)$$

We must have $\alpha < 2$ for the singularity to be integrable, and the integral then evaluates to $A = 2\pi/(2 - \alpha)$. In this case we can calculate the distribution of function values $P(f)$ inside the circle as

$$P(f)df = P(r) \left(\frac{dr}{df}\right) df. \quad (34)$$

In the circle with radius 1, $P(r) = 2r$, which together with $f(r) = r^{-\alpha}$ gives $P(f) = (2/\alpha)f^{-1-2/\alpha}$ for points inside the circle. We also need to account for the fact that that $f = 0$ outside the circle, i.e., with probability $1 - \pi/4$. Hence the full distribution of f -values is

$$P(f) = (1 - \pi/4)\delta(f) + \frac{\pi}{4} \frac{2}{\alpha} f^{-1-2/\alpha} \Theta(f), \quad (35)$$

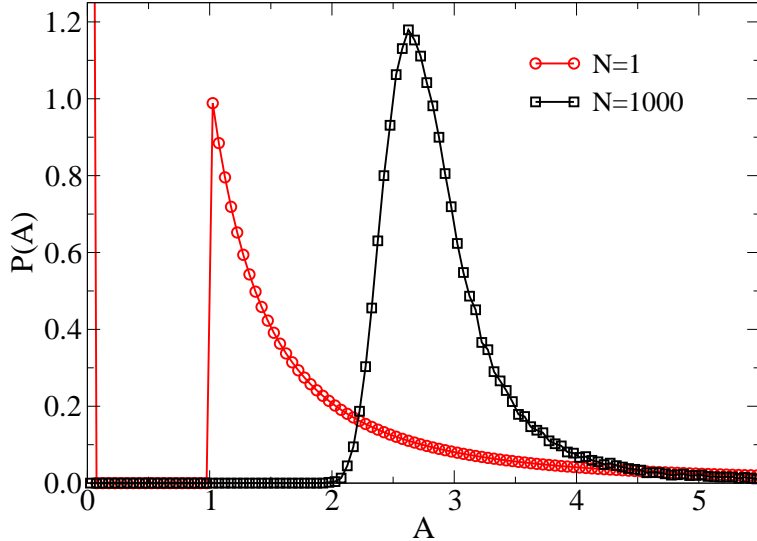


Figure 6: Distribution of individual function values ($N = 1$) and bin averages generated in a Monte Carlo integration of the function (33) with $\alpha = 3/2$, based on 10^5 bins and using $N = 1000$ points per bin (the distribution multiplied by 20 is shown to highlight the broad tail).

where $\Theta(f)$ is the step function which is 0 for $f < 1$ and 1 for $f \geq 1$. This distribution for $\alpha = 3/2$ is shown in Fig. 6. To calculate the distribution of averages over bins with N points, we need to integrate over a product of N of these distributions;

$$P(A) = \int_0^\infty df_N \cdots \int_0^\infty df_2 \int_0^\infty df_1 \times P(f_N) \cdots P(f_2) P(f_1) \delta\left(A - \frac{1}{N}(f_1 + f_2 + \dots + f_N)\right). \quad (36)$$

This probability distribution can be easily evaluated by Monte Carlo sampling. A distribution of bin averages with $N = 1000$ obtained this way is shown in Fig. 6. This distribution still has a clear tail and is far from a gaussian. Hence much larger N has to be used for the bins in order to obtain a reliable result (i.e., a statistically meaningful error bar) for the integral of (33). Here, in this simple case, this could be done without any extra effort just by increasing N while reducing the number of bins (which was as large as $M = 10^5$ in order to results in a reasonably smooth distribution) and keeping NM constant. This would not change the value for A , but the error bar would be more representative of the actual expected error of the estimated average.

It is interesting to see what happens if we try to integrate a divergent integral using the Monte Carlo method. Consider the limiting case $\alpha = 2$ in (33), in which case the integral is logarithmically divergent (as the radius of a circular region excluded from the integration volume goes to zero). Fig. 7 shows the calculated function average versus the number of bins. Comparing with Fig. 4 (showing the evolution of the integral for $\alpha = 0$) shows that the behavior of the divergent integral is much more erratic, with many large jumps. Clearly, these jumps originate from occasional points falling very close to the center of the circle. For a divergent integral, these contributions dominate the integral and hence, although they are relatively rare, they will eventually cause the average to become arbitrarily large and ill defined. The fact that the individual curves in Fig. 7 almost overlap in many regions, in spite of the completely different large fluctuations, reflect the

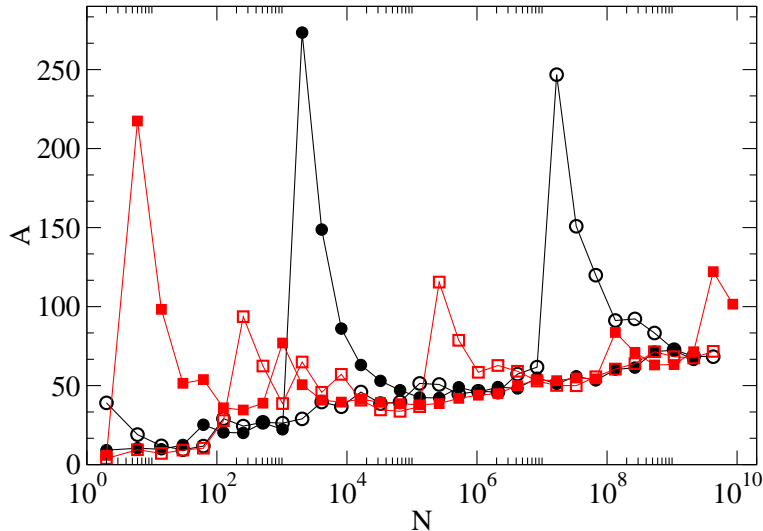


Figure 7: Evolution of the estimated value of the log-divergent integral (33) with $\alpha = 2$ in four independent runs.

fact that the distribution of averages based on N samples has a well defined peak for any finite N , although the integral of these distributions are divergent due to their “fat tails”. In the case of an integrable singularity, occasional large fluctuations will also occur, but the probability of a fluctuation decreases with the size of the fluctuation in such a way that the average is well defined.

6 Random number generators

Sequences of numbers that are almost random, or *pseudo-random numbers*, can be generated using completely deterministic algorithms on the computer. In practice, some flaws can be found in all known random number generators, but even some very simple ones are sufficiently good for most practical purposes (although there are also many examples of calculations that have produced wrong results because of poor random numbers; it is something one should pay attention to in serious work).

In Fortran 90, there is an intrinsic subroutine `random_number(ran)`, which when called assigns a value in the range $[0, 1)$ (i.e., including the value 0 but not 1) to the `real` variable `ran`. Calling this subroutine multiple times generates a sequence of such pseudo-random numbers. The sequence is always the same (starting from the first time the routine is called in a program), but it can be changed by calling the subroutine `random_seed(seeds)`, where `seeds` is an integer vector of size 3 containing three random number *seeds*. While built-in generators (in Fortran 90 as well as in other languages) are useful for simple calculations and testing, they should in general be avoided in serious work; much better ones can be programmed quite easily. The intrinsic `random_number()` may in fact be a good one, but the exact way it generates the number sequence is not specified in the Fortran 90 standard, and hence the quality can vary among systems.

Here we will only discuss the very basics of random number generation; this is in itself a big field in

number theory and applied mathematics. It is actually very simple to generate a sequence of good pseudo-random numbers (very good ones require more sophisticated methods than those discussed here).

Most random number generators do their internal work with integers and at the end convert the result to a floating-point number. One of the most frequently used basic methods is to construct a sequence of integers x_i according to the recurrence relation

$$x_n = \text{mod}(a \cdot x_{n-1} + c, m). \tag{37}$$

With the multiplier a , increment c , and modulus m suitably chosen, and starting with an integer $0 \leq x_0 < m$, this sequence will generate all the integers in the range $0, \dots, m - 1$ in a seemingly random order. While any odd number will do for the increment c , there is a relatively small subset of multipliers with which the full sequence of m numbers is produced (other choices will lead to a *period* of the generator smaller than m). Among the values of a that produce the full period of the sequence, some lead to better (more random) sequences than others; tables of good parameters have been published. It should be noted that on the computer the operation $a * x_{n-1} + c$ leads to overflow if the result is larger than the largest positive value that can be represented with a 32 or 64 bit integer. For the scheme to work as described above, a and c have to be chosen so that overflow does not occur (otherwise negative numbers will also be generated, and parameters that work well without overflow may not give a good sequence with overflow).

To see that the generator (37) indeed works, we can check explicitly what it produces for small m . We consider $m = 4, 8$, and 16 (with this type of recurrence relation, a multiplier of the form $m = 2^k$ turns out to work well), using $c = 1$ and trying all $a = 0, \dots, m - 1$. Giving the initial $x_0 = 0$, these are the sequences produced for $m = 4$:

```

a = 1   x = 0 1 2 3 0
      2   0 1 3 3 3
      3   0 1 0 1 0
      4   0 1 1 1 1

```

There is no sequence with period 4 in this case, except for the trivial one for $a = 0$, which is of course not random. With $m = 8$ we get

```

a = 1   x = 0 1 2 3 4 5 6 7 0
      2   0 1 3 7 7 7 7 7 7
      3   0 1 4 5 0 1 4 5 0
      4   0 1 5 5 5 5 5 5 5
      5   0 1 6 7 4 5 2 3 0
      6   0 1 7 3 3 3 3 3 3
      7   0 1 0 1 0 1 0 1 0
      8   0 1 1 1 1 1 1 1 1

```

Here the sequence for $a = 5$ indeed has period 8, and although the sequence is not ordered it is clearly not very random. With $m = 16$ we get two non-trivial sequences with period 16:

```

a = 5   x = 0 1 6 15 12 13 2 11 8 9 14 7 4 5 10 3 0
        13      0 1 14 7 12 13 10 3 8 9 6 15 4 5 2 11 0

```

In these two sequences the order appears more random, although there is one clear non-random feature; the numbers alternate between odd and even. It is no coincidence that the multipliers that produce the period- m sequences are primes; this will be true for higher m as well. As m increases, one can find more and more sequences with period m , and statistical analyses of these sequences reveal that they also become more random. In terms of the individual bits making up the integers, the high bits are more random than the low bits; the even-odd alternations found above persist for any $m = 2^k$ and correspond to the lowest bit alternating between 0 and 1 in successive numbers.

The integer overflow can be considered a modified modulus 2^{32} (or 2^{64} for 64-bit integers) function, and in fact one can base random number generators also on this automatic feature (it saves time). For example, these two lines of Fortran code constitute a relatively good random number generator (the built-in generator on many systems is of this type):

```

n=69069*n+1013904243
ran=0.5d0+dbple(n)*0.23283064d-9

```

Here the second line converts the number to a double-precision real in the range $[0, 1)$. This generator has only one seed; the initial value of n chosen before invoking the generator the first time (to which the value returns after 2^{32} steps).

A much longer period can be achieved by using 64-bit integers. There are several known multipliers that give the full period 2^{64} . An example is the generator

```

n=n*2862933555777941757+1013904243
ran=0.5d0+dbple(n)*dmul

```

where `dmul` is precalculated as

```

dmul=1.d0/dbple(2*(2_8**62-1)+1)

```

This 64-bit generator is used in many professional applications and is recommended for this course as well.

Pseudo-random numbers can also be generated on the basis of multiplying, adding or subtracting previous numbers of the sequence, e.g..

$$x_n = \text{mod}(x_{n-3} - x_{n-1}, m). \quad (38)$$

With a properly chosen m (typically of the form $2^m - p$, where p is a prime number) this kind of generator can in fact have a period much larger than m .

By combining several different random number generator one can achieve significant improvements to the randomness and the period. For example, the following lines of Fortran code implement a random number generator combining sequences of the forms (37) and (38):

```
mzran=iir-kkr
IF (mzran.lt.0) mzran=mzran+2147483579
iir=jjr; jjr=kkkr; kkr=mzran
nnr=69069*nnr+1013904243
mzran=mzran+nnr
ran=0.5d0+mzran*0.23283064d-9
```

This generator requires four different seeds; the initial values for the integers `iir`, `jj`, `kk`, `nn`. Its period is claimed to be larger than 10^{28} by its creators; G. Marsiglia and A. Zaman, in *Computers in Physics*, **8**, 117 (1994). It has also performed well in several tests. The full implementation of this generator and its initialization can be found on the course web site.

One often needs random numbers distributed according to some other distribution (e.g., Gaussian) than the "box" between 0 and 1. There are various ways to accomplish this starting from a uniform generator; see, e.g., *Numerical Recipes*.