Numerical Integration of Functions

Differential equations are the backbone of physics, the mathematical scaffolding upon which we erect models for the evolution of physical systems. For this reason, Physics 305 spends more time on this subject than on any other. We will begin our investigation by looking at the numerical evaluation of integrals, the determination of a numerical approximation to the definite integral

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

The connection to differential equations is, of course, that Q is the value at $x = b$ of the solution to the ordinary differential equation (ODE)

$$\frac{dF(x)}{dx} = f(x) \tag{2}$$

subject to the boundary condition $F(a) = 0$[1]. Most of the mathematical machinery we develop below will carry over directly into our study of ODE's; indeed, some of the most sophisticated methods for integration have been developed for solving differential equations.

Errors

Before we start, a brief discussion of errors is in order. Errors can creep into (even burst upon!) our calculations in a variety of ways. The most obvious way is that *we* make a mistake; this is known in computer parlance as a "bug". A bug occurs when what we have *in fact* told the computer to do is not what we *meant* to tell the computer to do. This could be because of a simple typographic error or, more seriously, because we don't really understand what it is we want the computer to do in the first place. In some sense, bugs are accidental, and we hope that, with sufficient care, they can be avoided or at least corrected. More important to the following discussion are two other forms of error, neither of which is accidental. They are essential features of using a computer to do numerical math – of life in finite precision. The first is roundoff error.

Integer values are represented in the computer by a finite string of binary digits or *bits*, typically of length 32 or 64. $2^{64} = 18446744073709551616$, so if we were to scale numbers on the interval $[0, 1)$ by that number, we would have a resolution of about 5×10^{-20}. One could then ask: what happens if we try to add 10^{-21} to zero? This would correspond to making a change to zero in the 66-th bit out of 64, so the answer is: we would still get zero. Clearly, $0 + 10^{-21} = 0$ is an error, but if we operate with a fixed number of bits (64 in this case), it is an unavoidable error. We simply can't store enough information in 64 bits to precisely evaluate $0 + 10^{-21}$. This form of error is called round-off error. We had to "round-off" the 66 bit result to the available number of bits,

[1]To see this, write the ODE as

$$dF = f\ dx$$

and integrate both sides over $[a, b]$

$$\begin{aligned}\int_{F(a)}^{F(b)} dF &= \int_a^b f(x)dx\\ F(b) - F(a) &= \int_a^b f(x)dx\\ F(b) &= \int_a^b f(x)dx + F(a)\end{aligned}$$

64. Another way of thinking of round-off in this case is that there are 2^{64} numbers which can be represented exactly, and the "gaps" between these numbers are filled with numbers which cannot be represented exactly – we are stuck using only the representable numbers.

Now, using integers to represent real numbers in this way has its limitations. We would very much like to be able to compute on ranges larger than $[0, 1)$, while at the same time to be able to represent numbers much smaller than 5×10^{-20}. To do so, we use *floating-point* numbers. Floating point numbers are represented by a mantissa with a fixed number of digits and an exponent with another fixed number of digits, for example: 5.34231×10^{-17}. Virtually all modern computers use the IEEE754[2] standard for storing *double precision* numbers which calls for a mantissa of 52 bits, an exponent of 11 bits, and a sign bit. This gives a bit better than 15 decimal digits of precision (a 15 decimal digit mantissa) and a range of about 10^{-308} to 10^{+308}. The `float` type in Python uses double precision. This is much more flexible than simple integer scaling, allowing us to represent a much larger range of numbers, but it still exhibits round-off error, albeit in a slightly more complex way. Double-precision numbers still have gaps, but the gaps vary in size depending upon the magnitude of the numbers. For example, for values near 10^{100}, the gaps are approximately 10^{85} in width, and round-off error looks like $10^{100} + 10^{84} = 10^{100}$. For values near 10^{-100}, the gaps are much smaller, around 10^{-115} in magnitude. Every numerical calculation using floating point numbers will thus suffer to some extent from round-off error.

For example, a *finite-difference* approximation (a topic we will get to next week) to the first derivative of a function f might be written as

$$\frac{d}{dx} f(x) \sim \frac{f(x+h) - f(x)}{h}$$

In infinite-precision arithmetic, the approximation becomes increasingly accurate the smaller we make h (in the limit $h \to 0$ this is the definition of the derivative). But what happens if we want to evaluate $f'(x)$ for $x \sim 1$? If $f(x) \sim x$ in magnitude, if we make h smaller than about 10^{-15}, we have $1 + 10^{-15} = 1$, and the numerator is identically zero! Less trivially, let $f(x) = e^x$. If $x = 1$, then we have, to 15 decimal digits,

$$\begin{aligned} \exp(1 + 10^{-12}) =& 2.71828182846176 \\ \exp(1) =& \underline{2.71828182845905} \\ \text{difference} =& \overline{0.0000000000271} \end{aligned} \tag{3}$$

so that $(\exp(1 + 10^{-12}) - \exp(1))/10^{-12} = 2.72 \times 10^{-12}$, a severe loss of accuracy with only three significant digits left out of fifteen!

The second form of error is known as *truncation error*. You will soon see that we make great use of series expansions in developing finite-precision approximations to analytic results. For example, the approximation to the derivative above comes from expanding $f(x + h)$ in a Taylor series about x:

$$f(x+h) = f(x) + hf'(x) + h^2 \frac{f''(x)}{2} + h^3 \frac{f'''(x)}{3!} + \ldots \tag{4}$$

Rearranging terms, we have

$$\frac{f(x+h) - f(x)}{h} = f'(x) + h\frac{f''(x)}{2} + \ldots \tag{5}$$

[2]The Institute of Electrical and Electronics Engineers (IEEE) is a professional association. One of its activities is to promulgate standards so that all manner of equipment and processes can inter-operate safely and effectively. The IEEE standard number 754 defines the layout of floating-point numbers and the algorithms for manipulating them, including, for example, how to do rounding, how the results of undefined operations (like dividing by zero) are represented, and how successively smaller numbers approach zero. Without this, every time you ran a program on a different computer architecture you might get a different result, a sure recipe for chaos.

This is an exact expression. The approximation introduced above was obtained by truncating the series on the RHS after the first term. The sum of the remaining terms is the truncation error: $\epsilon(h) = hf''(x)/2 + \dots$. Because the leading term in the error scales linearly with h (denoted as $\mathcal{O}(h)$), we call this a "first-order accurate approximation". We can reduce this error by reducing the magnitude of h, but as shown above we can only go so far before round-off error takes over as the dominant error and gives us *worse* answers for smaller h.

Much of the art of numerical computation lies in managing these sources of error.

Piecewise Constant Integration

The simplest precise definition of the definite integral of a function over a closed interval $[a, b]$ is in terms of a *Riemann sum* of the function over a *partition* of that interval. A partition is simply a sequence of points x_i

$$a = x_1 \leq x_2 \leq \cdots \leq x_N \leq x_{N+1} = b \tag{6}$$

which divide up the interval $[a, b]$ into a set of n sub-intervals $[x_i, x_{i+1}], i = 1, \dots, N$. If we pick a point in each sub-interval $t_i \in [x_i, x_{x+1}]$, the Riemann sum is

$$\sum_{i=1}^{N} f(t_i)(x_{i+1} - x_i) \tag{7}$$

This is the sum of the signed areas of n rectangles with width $x_{x+1} - x_i$ and height $f(t_i)$. The Riemann integral

$$S = \int_a^b f(x)dx \tag{8}$$

is then defined as the limiting value of the corresponding Riemann sum as the lengths of the intervals all go to zero. If we define the *mesh* of a partition as the maximum length of an interval, $\max_i(x_{i+1} - x_i), i = 1 \dots N1$, a more precise definition of this limit is that the Riemann integral of f equals S if the following condition obtains:

For every $\epsilon > 0$ there exists a δ and an N such that for any partition $x_1, \dots, x_{N+1}$ whose mesh is less than δ and any set of points $t_i \in [x_i, x_{i+1}], i = 1, \dots, N$, we have

$$\left| \sum_{i=1}^{N} f(t_i)(x_{i+1} - x_i) - S \right| < \epsilon$$

Of course, this limit is impossible to achieve in the finite-precision world of numerical methods, so we must be satisfied with some finite approximation. To simplify things, let's assume that our partition is evenly-spaced, with mesh h, so that $h = x_{i+1} - x_i, i = 1, .., N$ and

$$x_i = a + ih, \quad i = 1, \dots, N+1, \quad h = \frac{b-a}{N}$$

(To call out an obvious point, note that N intervals are defined by $N + 1$ mesh-points!) Let's (aribtrarily) pick our t_i to be the beginning of each interval, $t_i = x_i$. Then our finite Riemann sum approximates the integral as

$$\int_a^b f(x)dx \sim \sum_{i=0}^{N} f(x_i)(x_{i+1} - x_i) = h \sum_{i=0}^{N} f_i$$

where we have written f_i for $f(x_i)$.

If our function $f(x)$ happened to be piecewise-constant on the sub-intervals, this would be an exact representation of the integral. Since this is unlikely to be the case, we can estimate the truncation error in the approximation by expanding the function in a Taylor series in each interval,

$$\begin{aligned} f(x) &= f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2} f''(x_i) + \dots \\ &= f_i + (x - x_i)f'_i + (x - x_i)^2 \frac{1}{2} f''_i + \dots \\ &= f_i + hf'_i + h^2 \frac{1}{2} f''_i + \dots \end{aligned} \tag{9}$$

integrating each series, and comparing their sum with our approximation. Integrating the series representation of $f(x)$ over an interval, we have

$$\begin{aligned} \int_{x_i}^{x_{i+1}} f(x)dx &= (x_{i+1} - x_i)f(x_i) + \frac{(x_{i+1} - x_i)^2}{2} f'(x_i) + \frac{(x_{i+1} - x_i)^3}{3!} f''(x_i) + \dots \\ &= hf_i + h^2 \frac{1}{2} f'_i + h^3 \frac{1}{3!} f''_i + \dots \end{aligned} \tag{10}$$

The first term in the integrated expansion is precisely our approximation, so the leading term in the error on interval i is

$$\epsilon_i = h^2 \frac{1}{2} f'_i$$

The integral on $[a, b]$ is then approximated by

$$\int_a^b f(x)dx \sim h \sum_{i=1}^{N} f_i$$

with total error

$$\epsilon = \sum_{i=1}^{N} \epsilon_i = h^2 \sum_{i=1}^{N} \frac{1}{2} f'_i$$

If we write the mean value of $f'(x_i)$ as

$$\langle f'(x_i) \rangle = \frac{1}{N} \sum_{i=1}^{N} f'_i$$

the Mean Value Theorem tells us there must be some $\xi \in (a, b)$ such that

$$\langle f'_i \rangle = f'(\xi)$$

Since $h = (b - a)/N$, we then have

$$\epsilon = Nf'(\xi)\frac{1}{2}h^2 = Nf'(\xi)\frac{1}{2}h\frac{(b-a)}{n} = h(b-a)\frac{1}{2}f'(\xi) = \mathcal{O}(h)$$

All this is just to show that, if we approximate the integral as a piecewise constant function on each interval, the error scales linearly with h (which is to say, with $1/N$): doubling the number of intervals will decrease the error by a factor of two. Note that the error per interval (the *local error*) is $\mathcal{O}(h^2)$, but we have n intervals and $h \propto 1/N$, so that the global error is one power of h smaller than the local error.

The Trapezoidal Rule

What if we take another term in the Taylor series and approximate $f(x)$ on an interval by a piecewise *linear* function? Writing $f(x) = f_i + m_i\, x$ for $x \in [x_i, x_{i+1}]$, we can write the average slope on the interval as $m_i = (f_{i+1} - f_i)/h$, and our approximation to $f(x)$ on an interval is

$$f(x) \sim f_i + \frac{f_{i+1} - f_i}{h}(x - x_i)$$

Integrating this analytically over the interval we have

$$\int_{x_i}^{x_{i+1}} f(x)dx \sim h\frac{1}{2}\left(f_{i+1} + f_i\right)$$

We have now replaced the integral by the sum of the areas of a set of trapezoids of width h and height given by the average of the function at the endpoints of the intervals; this approximation is thus known as the *trapezoidal rule*:

$$\int_a^b f(x)dx \sim h\sum_{i=1}^{N} \frac{1}{2}(f(x_i) + f(x_{i+1})) \tag{11}$$

We can determine the error of this new scheme as follows. In equation (10) we expanded $f(x)$ about the beginning of the interval, x_i. Now expand $f(x)$ *backwards* about the endpoint of the interval, x_{i+1}:

$$f(x) = f_{i+1} - (x - x_{i+1})f'_{i+1} + (x - x_{i+1})^2\frac{1}{2}f''_{i+1} + \dots \tag{12}$$

Integrating this expression over the interval as before, we have

$$\int_{x_i}^{x_{i+1}} f(x)dx = hf_i - h^2\frac{1}{2!}f'_{i+1} + h^3\frac{1}{3!}f''_{i+1} + \dots \tag{13}$$

Averaging this and the result of integrating the forward expansion (equation 10), we have

$$\int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2}(f_i + f_{i+1}) + \frac{h^2}{4}(f'_i - f'_{i+1}) + \frac{h^3}{12}(f''_i + f''_{i+1}) + \dots$$

We can expand the derivatives at x_{i+1} about x_i:

$$\begin{aligned} f'_{i+1} &= f'_i + hf''_i + h^2\frac{1}{2}f'''_i + \dots \\ f''_{i+1} &= f''_i + hf'''_i + h^2\frac{1}{2}f^{(4)}_i + \dots \end{aligned} \tag{14}$$

Substituting for these values and keeping only terms to order h^3, we have

$$\int_{x_i}^{x_{i+1}} f(x)dx = h\frac{1}{2}(f_i + f_{i+1}) - h^3\frac{1}{12}f''_i + \dots \tag{15}$$

The *local error* – the error per interval – is now $\mathcal{O}(h^3)$.

Summing over all of the intervals, we obtain the *extended* trapezoidal rule (extended to the entire integral over $[a, b]$, that is)

$$\begin{aligned} \int_a^b f(x)dx &= \sum_{i=1}^{N+1} \frac{1}{2}h(f_i + f_{i+1}) \\ &= h\left(\frac{1}{2}f_1 + f_2 + \dots + f_N + \frac{1}{2}f_{N+1}\right) \end{aligned} \tag{16}$$

Summing the leading term in the error on each interval, we have

$$\epsilon = -\frac{h^3}{12}\sum_{i=1}^{N} f_i'' \tag{17}$$

Using the Mean Value Theorem once again, there must be some $\xi \in [a,b]$ such that $Nf''(\xi) = \sum_{i=1}^{N} f_i''$, and so the global error is

$$\epsilon = -N\frac{h^3}{12}f''(\xi) = -\frac{(b-a)}{h}\frac{h^3}{12}f''(\xi) = \mathcal{O}(h^2) \tag{18}$$

For the trapezoidal rule, doubling the number of intervals decreases the global error by a factor of four.

One can carry on with this approach, approximating the function on an interval by polynomials of increasing order and obtaining *quadrature formulae*[3] of increasing accuracy with the same number of points (known as Simpson's rule, Bode's rule, *etc.*). In practice, however, the trapezoidal rule will usually be sufficient for most purposes since we can always increase the number of points as necessary, especially with the following "tricks".

Recursive Trapezoidal Rule

We can write the trapezoidal rule using N intervals ($N+1$ points) to integrate the function $f(x)$ over $[a,b]$ as

$$I_n = h_n\left(\frac{1}{2}f(a) + \sum_{k=1}^{2^n-1} f(a+kh) + \frac{1}{2}f(b)\right) \tag{19}$$

with $N = 2^n$ and $h_n = (b-a)/2^n$. With this, we can generate the sequence of approximations

$$\begin{aligned}
I_0 &= h_0\left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right)\\
I_1 &= h_1\left(\frac{1}{2}f(a) + f(a+h_1) + \frac{1}{2}f(b)\right)\\
I_2 &= h_2\left(\frac{1}{2}f(a) + f(a+h_2) + f(a+2h_2) + f(a+3h_2) + \frac{1}{2}f(b)\right)
\end{aligned} \tag{20}$$

The thing to notice is that each successive approximation equals half the previous approximation plus additional terms evaluated at the midpoints of the previous approximation's intervals. For $n > 0$, we have that

$$I_n = \frac{1}{2}I_{n-1} + h_n\sum_{j=1}^{2^{n-1}} f(a+(2j-1)h_n) \tag{21}$$

It is often the case that the computational effort required to evaluate $f(x)$ is significant. Note that computing I_n requires evaluating the function 2^n+1 times. To compute the next approximation, I_{n+1}, we need only evaluate the function 2^n more times, at the points halfway between the points already computed. Thus, computing I_n and I_{n+1} requires roughly 2^{n+1} function evaluations. If we had computed these quantities separately using the trapezoidal rule, we would have performed $3(2^n)$ evaluations – we've saved 50% of the effort.

[3]*Quadrature* is an old-fashioned name for integration.

An estimate of the absolute error is the difference between successive approximations, $|I_{n+1} - I_n|$; the relative error can be estimated by $|(I_{n+1} - I_n)/I_n|$. This gives us a robust algorithm for choosing the number of points required to evaluate an integral to some requested accuracy – keep incrementing n, doubling the number of intervals, until the error criterion is satisfied. The real utility of this technique becomes apparent in the next section, however.

Romberg Integration

For functions which are sufficiently smooth, which is to say functions which have continuous derivatives to all orders on $[a, b]$, another expression for the error in the trapezoidal rule comes from the Euler-Maclauren Formula, which can be written as

$$\begin{aligned}\int_a^b f(x)dx =& \sum_{i=1}^{N} \frac{1}{2} h\left(f_i + f_{i+1}\right) \\ &- \sum_{k=1}^{\lfloor 2p \rfloor} \frac{B_{2k} h^{2k}}{(2k)!} \left(f^{(2k-1)}(b) - f^{(2k-1)}(a)\right) \\ &- (b-a) P_{2p+1}(\xi) h^{2p+1} f^{(2p+1)}(\xi), \qquad \text{for some } \xi \in [x_1, x_{n+1}]\end{aligned} \tag{22}$$

where the B_k are the Bernoulli numbers, the $P_k(x)$ are the Bernoulli polynomials (don't worry for now what these are), and $\lfloor x \rfloor$ is the floor function, the greatest integer $\leq x$. The Euler-Maclaurin formula expresses the integral of a function in terms of a finite sum of evenly-spaced function evaluations (the trapezoidal rule, in fact!) plus a series representation of the error with a remainder term (which can be shown to be at most twice the magnitude of the last term in the error series[4]).

While deriving this formula is beyond the scope of this course, it shows that there are only even powers of h in the error, which we can thus write notionally as

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \frac{1}{2} h(f_i + f_{i+1}) - \sum_{k=1}^{\lfloor 2p \rfloor} C_k h^{2k} \tag{23}$$

where the C_k depend only upon the derivatives of f. We can then write any two successive trapezoidal rule approximations, with their error terms, as

$$\begin{aligned} I_{n-1} &= I + C_1 h_{n-1}^2 + \mathcal{O}(h^4) \\ I_n &= I + C_1 (\frac{1}{2} h_{n-1})^2 + \mathcal{O}(h^4) \end{aligned} \tag{24}$$

where I is the exact value of the integral. Ignoring the terms $\mathcal{O}(h^4)$, we can solve these two equations for I and C_1. Since we are ignoring terms $\mathcal{O}(h^4)$, I is no longer the exact value of the integral, but it will be an improved approximation, since the value of C_1 which results cancels the $\mathcal{O}(h^2)$ error terms. We will call this improved approximation $I_{n,1}$ (and hereafter call the original trapezoidal rule results $I_{n-1,0}$ and $I_{n,0}$)

$$I_{n,1} = \frac{4I_{n,0} - I_{n-1,0}}{3} + \mathcal{O}(h^4) \tag{25}$$

Thus, taking the appropriate linear combination of $I_{n,0}$ and $I_{n-1,0}$, each correct to $\mathcal{O}(h^2)$, we have derived an approximation correct to $\mathcal{O}(h^4)$.

[4] I have used the Mean Value theorem again in the last line to assert an appropriate value of ξ exists.

Of course, we can compute another $\mathcal{O}(h^4)$ result using $I_{n+1,0}$

$$I_{n+1,1} = \frac{4I_{n+1,0} - I_{n,0}}{3} + \mathcal{O}(h^4) \tag{26}$$

but now, with two $\mathcal{O}(h^4)$ approximations, we can eliminate the next term in the error series, the one proportional to h^4

$$\begin{aligned} I_{n,1} &= I + C_2(h_n)^4 + \mathcal{O}(h^6) \\ I_{n+1,1} &= I + C_2(\frac{1}{2}h_n)^4 + \mathcal{O}(h^6) \end{aligned} \tag{27}$$

and solve to obtain the $\mathcal{O}(h^6)$ approximation

$$I_{n+1,2} = \frac{16I_{n+1,1} - I_{n,1}}{15} + \mathcal{O}(h^6) \tag{28}$$

and so on.

The general result is that the $\mathcal{O}(h^{2k})$ approximation for 2^n intervals is

$$I_{n,k} = \frac{4^k I_{n,k-1} - I_{n-1,k-1}}{4^k - 1}, \quad k = 1, \ldots, n \tag{29}$$

This expression generates a tableau of approximations which looks like

$$\begin{array}{llll} I_{0,0} & & & \\ I_{1,0} & I_{1,1} & & \\ I_{2,0} & I_{2,1} & I_{2,2} & \\ I_{3,0} & I_{3,1} & I_{3,2} & I_{3,3} \end{array}$$

where the first column represents a series of $\mathcal{O}(h^2)$ evaluations, each with twice the number of intervals as in the row above, the second column is the resulting $\mathcal{O}(h^4)$ extrapolations, and so on, with the final result an evaluation accurate to $\mathcal{O}(h^8)$. We can use the last two values in a row to estimate the fractional (relative) error in the result at that n, as before

$$\epsilon \sim \frac{I_{n,n} - I_{n,n-1}}{I_{n,n-1}}$$

Clearly, in finite precision this procedure cannot be continued too far, or roundoff error will begin to dominate the result instead of truncation error.

This is an example of *Richardson extrapolation*, a technique to accelerate the convergence of a sequence. In this case, we have a sequence of trapezoidal rule estimates I_n which converges as h^2 for $h = (b-a)/2^n$, or $1/2^{2n}$. Applying Richardson extrapolation provides us with a new estimate, $I_{n,n}$, which converges much faster. To find out how much faster, combine this with our recursive trapezoidal rule (19),

$$I_n = h_n \left(\frac{1}{2} f(a) + \sum_{k=1}^{2^n - 1} f(a + kh) + \frac{1}{2} f(b) \right)$$

to produce an algorithm known as *Romberg Integration*

```
nextTrapezoidal($f, x_{min}, x_{max}, n$):
    """
    Return the difference between the trapezoidal rule approx-
    imation for 2^n intervals and one-half the trapezoidal
    rule for 2^(n-1) intervals.
    """
    $N = 2^n$
    $h = (x_{max} - x_{min})/N$
    $I_n = 0$
    for $j = 1$ to $N/2$:
        $I_n = I_n + f(x_{min} + (2j-1)h)$

    return $hI_n$

romberg($f, x_{min}, x_{max}, n_{max}, \epsilon$):
    """
    Use Richardson extrapolation to compute the integral of f(x)
    from x_min to x_max with a relative error less than epsilon,
    using trapezoidal rules with up to 2^n_max intervals.
    """
    $I_{0,0} = \frac{1}{2}(x_{max} - x_{min})(f(x_{min}) + f(x_{max}))$

    for $n = 1$ to $n_{max}$:
        $I_{n,0} = \frac{1}{2} I_{n-1,0}$ + nextTrapezoidal($f, x_{min}, x_{max}, n$)

        for $k = 1$ to $n$:
            $q = 4^k$
            $I_{n,k} = (qI_{n,k-1} - I_{n-1,k-1})/(q-1)$

        if $|I_{n,n} - I_{n,n-1}| < \epsilon |I_{n,n-1}|$:
            return $I_{n,n}$

    print "failured to converge!"
```

As an example, we will compute the *error function* defined by

$$erf(x) = \int_0^x \frac{2}{\sqrt{\pi}} e^{-y^2} dy \tag{30}$$

for $x = 1$. The resulting tableau for $(m, n) \in [0, 4]$ is

0.771743332258054				
0.825262955596749	0.843102830042981			
0.838367777441205	0.842736051389357	0.842711599479115		
0.841619221244768	0.842703035845956	0.842700834809729	0.842700663941961	
0.842430505490232	0.842700933572054	0.842700793420461	0.842700792763488	0.842700793268671

to 15 digits: erf(1) = 0.842700792949715

The result is almost magic. The first column gives the five trapezoidal rule evaluations with 1, 2, 4, 8, and 16 intervals (using a total of $2^4 + 1 = 17$ function evaluations in all). The final result (the lower-right value in the tableau) has an absolute error of $\sim 3 \times 10^{-10}$, while the error of the 16-interval trapezoidal rule is $\sim 3 \times 10^{-4}$. To have obtained the same accuracy as Romberg integration using only the trapezoidal rule, one would have needed $2^{14} = 16384$ intervals, a thousand times more work!

Of course, Romberg integration is not a panacea. The constants C_k are related to the derivatives of the integrand, and the procedure will only work if the function is sufficiently smooth (has well-behaved derivatives up to some order). Richardson extrapolation can be applied to many problems of this sort in addition to numerical integration, and we will run into it again when we look at more advanced methods of integrating differential equations.

Gaussian Quadrature

Without going into the details here, there is another approach to numerical integration which is worth mentioning briefly. In the above discussion, we have kept the abscissæ fixed, with a constant stepsize between them. We then defined "weights" of various complexity to give us ever-increasing accuracy. We thus wrote the numerical integral of $f(x)$ as

$$\int_a^b f(x)dx \sim \sum_{i=1}^{N} w_i f(x_i). \tag{31}$$

There is no reason (except simplicity!), to determine the weights in such a simple manner. If we allow ourselves to choose the x_i in some other way, we will have twice the number of degrees of freedom to achieve greater accuracy from the same number of points. One common application of this idea is known as *Gaussian integration* (quadrature is an archaic term for integration). Here, one chooses the abscissæ *and* weights to make the integral exact for integrals that are a polynomial times some function $W(x)$. The "weighting function" $W(x)$ can also be chosen to remove integrable singularities from the integral. Thus, we can find a set of abscissæ and weights such that

$$\int_a^b W(x)g(x) \sim \sum_{i=1}^{N} w_i g(x_i) \tag{32}$$

is exact if $f(x)$ is a polynomial of some order. We can then use this procedure for integrals of a function $f(x)$ where $g(x) = f(x)/W(x)$ is "close" to a polynomial. The resulting accuracy is, again, almost magical. One can get fabulous accuracy for some functions with *very* many fewer function evaluations than for the trapezoidal rule or its derivatives, or even Romberg integration.

The theory of how to find these weights, though beyond the scope of this course, is clever (going back to Gauss) and very useful. There are many routines on the web which you can use to find these "Gaussian Quadratures" for a given function $W(x)$, and weights and abscissæ are tabulated in many places for "popular" weighting functions $W(x)$. The Wikipedia entry for Gaussian Quadrature would be a good place to begin an investigation.