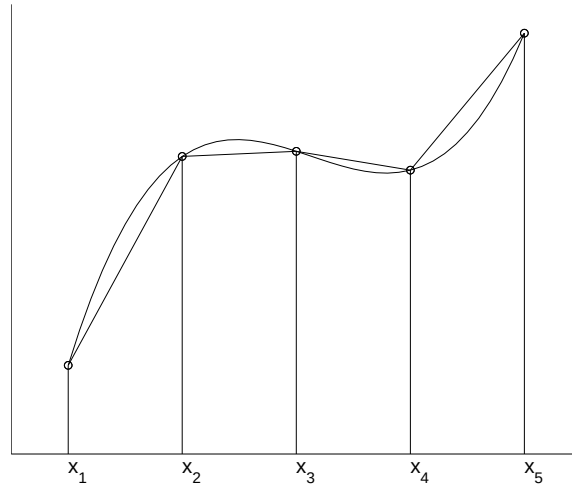


The basic idea to calculate an integral $\int_a^b f(x)dx$ numerically is to break up the interval $[a, b]$ into sub-intervals $a = x_1 < x_2 < x_3 < \dots < x_{N-1} < x_N = b$ (partition of $[a, b]$). On each of these sub-intervals we approximate the function by a simpler function, a constant, a straight line or a parabola for instance, which we can integrate easily.



The trapezoidal approximation to the integral of a function $f(x)$

Let $f_i = f(x_i)$, $h_i = x_{i+1} - x_i$ and $h = \max h_i$ then we have the following approximations:

Riemann Sum:

$$\int_a^b f(x)dx = \sum_{i=1}^{N-1} f_i h_i + O(h)$$

Trapezoidal Rule:

$$\int_a^b f(x)dx = \sum_{i=1}^{N-1} \frac{f_i + f_{i+1}}{2} h_i + O(h^2)$$

Midpoint Rule:

$$\int_a^b f(x)dx = \sum_{i=1}^{N-1} f_{i+1/2} h_i + O(h^2)$$

Simpson's formula:

$$\int_a^b f(x)dx = \sum_{i=1}^{N-1} \frac{f_i + 4f_{i+1/2} + f_{i+1}}{6} h_i + O(h^4)$$

The “big O” notation indicates how the error behaves with h as $h \rightarrow 0$. $O(h)$ for the Riemann Sum means that the error goes to zero like h , for instance $3h + 27h^2$ and $\sin(5h)$ are $O(h)$.

Note that if $TR(N)$ is the value computed using the trapezoidal rule with N segments and $SR(N)$ is the value obtained using Simpson's rule with the same segments then

$$SR(N) = \frac{4 TR(2N) - TR(N)}{3}$$

where $TR(2N)$ is the trapezoidal rule approximation computed using the $2N$ intervals obtained by splitting each of the N intervals in half. From the error estimates, $TR(2N)$ is 4 times more accurate than $TR(N)$.

The trapezoidal and Simpson rules are built into Matlab (“help **trapz**”, “help **quad**”).

This is nice and easy, but in practice there are many complications such as integrable singularities. See PS4, problem 3 and its solution for instance.

- There are fancy integration methods called *Gauss quadrature*. There are many such methods: Gauss-Legendre, Gauss-Chebyshev,... Those methods apply when the integral is of the form $\int_a^b w(x)g(x)dx$ where $g(x)$ is a “nice function” (*i.e.* a function that has many, many derivatives such as a polynomial or $\sin x$) and $w(x) \geq 0$ is called the weight function. Gauss-Legendre applies when $w(x) = 1$ and Gauss-Chebyshev when $w(x) = 1/\sqrt{1-x^2}$ when $a = -1, b = 1$ (note: $[a, b]$ can always be mapped to $[-1, 1]$ by the linear transformation: $x = (a+b)/2 + u(b-a)/2$, so x in $[a, b]$ means u is in $[-1, 1]$). Gauss quadrature gives fantastic accuracy when you use the right method for the right function but will give lousy results otherwise. So you need to know what you are doing. A gauss quadrature formula will look like $\int_a^b w(x)g(x)dx \approx \sum_{i=1}^N w_i g(x_i)$ where the *weights* w_i and abscissas x_i are chosen so that this formula is *exact* for any polynomial of degree $2N - 1$ or less. The x_i are not equispaced and the weights w_i are NOT equal to the size of the sub-intervals, $w_i \neq (x_{i+1} - x_i)$. For instance $\int_{-1}^1 P_3(x)dx = P_3(-1/\sqrt{3}) + P_3(1/\sqrt{3})$ for any polynomial $P_3(x)$ of degree 3 or less.

- If the function $f(x)$ is only known at a set of discrete values x_i then we are restricted in what we can do. We still can use the trapezoidal rule and this amounts to doing linear interpolation between the data points. Another good option is to use *spline interpolation* and integrate the spline interpolant. Remember that spline interpolation consist in representing the function by piecewise polynomials of 3rd degree between any two data points and imposing continuity of the 1st and 2nd derivatives at the data points. So between x_i and x_{i+1} , the function is represented by $S_i(x) \equiv c_1(x - x_i)^3 + c_2(x - x_i)^2 + c_3(x - x_i) + c_4$. It is easy to integrate this over the subinterval, let $h_i = x_{i+1} - x_i$ then

$$\int_{x_i}^{x_{i+1}} S_i(x)dx = c_1 \frac{h_i^4}{4} + c_2 \frac{h_i^3}{3} + c_3 \frac{h_i^2}{2} + c_4 h_i.$$

To compute the integral over the full domain, we just need to add those pieces together.

Matlab’s **spline** function returns the “pp” form (pp for “piecewise polynomial”) of the spline interpolant, meaning all the coefficients c_k for each interval. If x and y are two vectors of length N then

```
>> pp=spline(x,y);
>> [x,c,L,k]=unmkpp(pp);
```

returns the x ’s in the vector x and the coefficients in the L by k matrix “c,” where L is the number of intervals and $k=4$ is the number of coefficients per interval.

The function **splineq** posted on the “lectures” web page implements this spline integration method, download it! Example: $\int_0^\pi \sin x dx$

```
>> x=[0:0.1:1]*pi; y=sin(x); [trapz(x,y), splineq(x,y)]
ans =
    1.9835    2.0000
```