# 1 Numerical quadrature

In this section, we want to develop methods to approximate the definite interval of a function $f(x)$ on the interval $[a,b]$, given smoothness assumptions on $f$. In particular, we want to use a method known as *numerical quadrature*, aka the Nystrom method. In general, what this means is: given a set of $n+1$ **quadrature nodes** $\{x_j\}_{j=0}^n \in [a,b]$ and corresponding **quadrature weights** $\{w_j\}_{j=0}^n$,

$$\int_a^b f(x)dx \simeq Q[f] = \sum_{j=0}^n w_j f(x_j) \tag{1.1}$$

Note that this is *not the only way* to approximate integrals. The *Galerkin method*, for example, uses function approximation $f(x) \simeq \sum_{j=0}^m a_j\phi_j(x)$ for a given basis of functions $\{\phi_j(x)\}_{j=0}^m$ (e.g. polynomials, splines, trigonometric poly, rational). The integral of $f(x)$ is then approximated by the integral of $\sum_{j=0}^m a_j\phi_j(x)$, which may be written as a linear combination of the integrals of $\phi_j$ (which are easier to compute).

## 1.1 Newton-Cotes Interpolation-based quadrature

Since the Nystrom method (quadrature) uses a weighted sum of function evaluations, it is natural to propose the following: given a choice of $n+1$ nodes $x_j$ in $[a,b]$, which we will WLOG assume to be distinct and sorted in ascending order, compute the unique polynomial interpolant of $f(x)$. This interpolant may be expressed in the Lagrange basis. Then,

$$\int_a^b f(x)dx \simeq \int_a^b \sum_{j=0}^n f(x_j)L_j(x) = \sum_{j=0}^n \left(\int_a^b L_j(x)dx\right) f(x_j) \tag{1.2}$$

This gives us a quadrature rule with weights $w_j = \int_a^b L_j(x)dx$. If $f \in C^{n+1}([a,b])$, we know the interpolant is close to the function (and have a formula for the error). We can use this to determine how close the integral of the interpolant is to the integral of $f(x)$.

### Trapezoidal rule

The trapezoidal rule for one interval $[a,b]$ is simply the Newton-Cotes rule applied to the linear interpolant for $f(x)$ at nodes $x_0 = a, x_1 = b$. The region under this linear interpolant is a trapezoid, with area $\frac{f(b)+f(a)}{2}(b-a)$.

Let's first compute this using linear interpolation and Lagrange polynomials. The linear interpolant is

$$p_1(x) = f(a)\frac{b-x}{b-a} + f(b)\frac{x-a}{b-a} = \frac{1}{(b-a)}\left(f(a)(b-x) + f(b)(x-a)\right) \tag{1.3}$$

We integrate:

$$\int_a^b p_1(x)dx = \frac{1}{(b-a)}\left(f(a)\int_a^b (b-x)dx + f(b)\int_a^b (x-a)dx\right) \tag{1.4}$$

$$= \frac{1}{(b-a)}\left(f(a)(-1)\frac{(b-x)^2}{2}|_a^b + f(b)\frac{(x-a)^2}{2}|_a^b\right) \tag{1.5}$$

$$= \frac{1}{(b-a)}\left(f(a)\frac{(b-a)^2}{2} + f(b)\frac{(b-a)^2}{2}\right) \tag{1.6}$$

$$= (b-a)\frac{f(a)+f(b)}{2} \tag{1.7}$$

### Simpson rule

The Simpson rule for one interval $[a, b]$ is Newton Cotes applied to a quadratic interpolant at *equispaced* nodes $x_0 = a, x_1 = (a+b)/2, x_2 = b$. The area under the corresponding quadratic can be found using either geometric or analytic arguments. We can use symmetry arguments for the Lagrange polynomials to find that $w_0 = w_2$. Finding the weights $w_0, w_1$ means computing:

$$w_0 = w_2 = \int_a^b \frac{(x - \frac{a+b}{2})(x-b)}{\frac{(b-a)^2}{2}}dx = \frac{2}{(b-a)^2}\int_a^b (x - \frac{a+b}{2})(x-b)dx \tag{1.8}$$

$$w_1 = \int_a^b \frac{(x-a)(x-b)}{-\frac{(b-a)^2}{2}}dx = \frac{2}{(b-a)^2}\int_a^b (x-a)(b-x)dx \tag{1.9}$$

We can compute these integrals directly, or use a change of variables to $[-1, 1]$. Either way, we get that $w_0 = w_2 = \frac{1}{3}\frac{(b-a)}{2}$ and $w_1 = \frac{4}{3}\frac{(b-a)}{2}$.

**Note:** For every example derived using this interpolation-based rule, the weights are constant multiples of the spacing between subsequent points. If we call the spacing $h$, then

- Trapezoidal rule is $T[f] = w_0 f(x_0) + w_1 f(x_1)$ for vector of nodes $\mathbf{x} = [a \quad b]$ and vector of weights $\mathbf{w} = h[\frac{1}{2} \quad \frac{1}{2}]$ with $h = b - a$.
- Simpsons rule is $S[f] = w_0 f(x_0) + w_1 f(x_1) + w_2 f(x_2)$ for vector of nodes $\mathbf{x} = [a \quad \frac{a+b}{2} \quad b]$ and vector of weights $\mathbf{w} = h[\frac{1}{3} \quad \frac{4}{3} \quad \frac{1}{3}]$ with $h = (b-a)/2$.
- Newton-Cotes for an equispaced set of $n+1$ nodes is $N[f] = \sum_{j=0}^{n}$ for vector of nodes $\mathbf{x} = [a \quad a+h \quad \dots \quad b]$ and vector of weights $\mathbf{w} = h[\omega_0 \quad \omega_1 \quad \dots \omega_n]$ with $h = (b-a)/n$ and $\omega_j$ constant (compute or look up in a table).

### Error analysis for Newton Cotes

Given that the Newton-Cotes rules are interpolation based, we may use our results on the interpolation error to derive theorems on the corresponding quadrature error. We recall that, assuming $f \in C^{n+1}([a, b])$, we have that the interpolation error $E_n[f](x) = f(x) - \sum_{j=0}^{n} f(x_j)L_j(x)$ is of the form

$$E_n[f] = \frac{f^{n+1}(\eta_x)}{(n+1)!}\prod_{j=0}^{n}(x - x_j) = \frac{f^{n+1}(\eta_x)}{(n+1)!}\Psi_n(x) \tag{1.10}$$

For some $\eta_x \in (a, b)$. We can try to use this to derive a formula for the quadrature error:

$$\int_a^b E_n[f] = \int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = \frac{1}{(n+1)!}\int_a^b f^{n+1}(\eta_x)\Psi_n(x) \tag{1.11}$$

To pursue this analysis further, we want to be able to pull the $n+1$ derivative "out of the integral". This can be done using a generalized Mean Value Theorem (for integration), but only in certain cases.

**Error analysis for Trapezoidal rule:** For the case of Trapezoidal rule, $\Psi_1(x) = (x-a)(x-b) \leq 0$. Because $\Psi(x)$ is always non-positive, we can argue that by MVT there exists $\alpha \in [a,b]$ such that

$$\int_a^b E_n[f] = \int_a^b f(x)dx - T[f] = \frac{f''(\alpha)}{2}\int_a^b (x-a)(x-b)dx = \frac{-f''(\alpha)}{12}(b-a)^3 \tag{1.12}$$

Note that this means **Trapezoidal rule integrates linear functions exactly, and the error is proportional to the spacing $h$ to the power** $3 = n+2$.

**Error analysis for Simpsons rule:** If we attempt to use the same method for Simpsons rule, we run into an issue: $\Psi_2(x)$ changes sign, and integrates to 0 due to symmetry. As we will see, what this means is that we can find an error bound involving not the third, but the fourth derivative, and we gain one power of the spacing $h = (b-a)/2$ in the corresponding error estimate.

There are two ways to prove this. The first is to use more sophisticated arguments based on interpolation to make a similar analysis as we did for trapezoidal. The second, which is arguably easier, is to show that Simpson integrates cubics exactly, and then use Taylor expansions to derive an error bound:

First, we establish the fact that Simpsons rule, although designed for quadratic polynomials, integrate cubics exactly. For this, we consider the cubic $q(x) = (x - \frac{a+b}{2})^3$ centered around the midpoint. We then see that integral and quadrature match:

$$\int_a^b q(x)dx = 0 = \sum_{j=0}^{2} w_j q(x_j) = w_0 q(x_0) + w_2 q(x_2) = q(x_0)(w_0 - w_2) \tag{1.13}$$

This happens because the weights for Simpson are even ($w_{n/2-i} = w_{n/2+i}$). But then, we have

**Theorem 1.1** *Let $p \in \mathcal{P}_3$. The Simpsons rule on the interval $[a,b]$ integrates $p$ exactly.*

The proof of this is simply that, given $p \in \mathcal{P}_3$, we can write it as $p(x) = Cq(x) + r(x)$ where $r(x) \in \mathcal{P}_2$. Since Simpsons integrates $q$ and $r$ exactly, it integrates $p$ exactly!

**Taylor expansion:** Since Simpsons rule integrates polynomials up to degree 3 exactly, assuming $f \in C^4([a,b])$, we approximate it using the *third order* Taylor polynomial $\tau_3(x)$ centered at $x = a$ and use the corresponding Taylor residual formula. Then,

$$f(x) = \tau_3(x) + \frac{f^{(4)}(\eta_x)}{4!}(x-a)^4 \tag{1.14}$$

$$\int_a^b f(x)dx = \int_a^b \tau_3(x)dx + \int_a^b \frac{f^{(4)}(\eta_x)}{4!}(x-a)^4 dx \tag{1.15}$$

$$\int_a^b f(x)dx = \int_a^b \tau_3(x)dx + \frac{f^{(4)}(\eta)}{4!}\int_a^b (x-a)^4 dx \tag{1.16}$$

$$\tag{1.17}$$

for some $\eta \in [a,b]$ (using the generalized MVT, since $(x-a)^4 \geq 0$). Now, if we apply the Simpsons rule to $f(x)$, the part corresponding to $\tau_3$ is exactly equal to the integral of $\tau_3$, and so we can ignore them and write the error in terms of integrating $(x-a)^4$!

$$\begin{aligned}\int_a^b (x-a)^4 dx - S[(x-a)^4] &= \frac{(b-a)^5}{5} - \frac{(b-a)}{2}\left(\frac{(b-a)^4}{3} + \frac{(b-a)^4}{48}\right) \\ &= (b-a)^5\left(\frac{1}{5} - \frac{1}{6} - \frac{1}{24}\right) = -\frac{(b-a)^5}{120}\end{aligned}$$

And so,

$$\int_a^b f(x)dx - S[f] = -\frac{(b-a)^5}{120}\frac{f^4(\eta)}{4!} = \frac{-1}{90}\left(\frac{(b-a)}{2}\right)^5 f^4(\eta) \tag{1.18}$$

where we again write this as a product of the spacing $h = (b-a)/2$ and a constant involving the fourth derivative of $f$ at some $\eta \in [a,b]$.

We can generalize this reasoning to all Newton Cotes formulas, to derive the following theorem:

**Theorem 1.2 (Newton-Cotes error analysis)** *We consider using the Newton-Cotes interpolation based rule for a set of $n+1$ equispaced points using the polynomial interpolant in $\mathcal{P}_n$.*

- ***n+1 is even:*** *The rule integrates polynomials in $\mathcal{P}_n$ exactly. Using the Taylor approximation centered at a, we conclude that for $M_n \in \mathbb{R}, \eta \in [a,b]$,*

$$\int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = M_n f^{n+1}(\eta)\left(\frac{b-a}{n}\right)^{n+2} \tag{1.19}$$

- ***n+1 is odd:*** *The rule integrates polynomials in $\mathcal{P}_{n+1}$ exactly. Using the Taylor approximation centered at a, we conclude that for $M_n \in \mathbb{R}, \eta \in [a,b]$,*

$$\int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = M_n f^{n+2}(\eta)\left(\frac{b-a}{n}\right)^{n+3} \tag{1.20}$$

*where, for the odd case (like midpoint rule or Simpsons), we use a symmetry argument for the weights and apply the rule to $(x - \frac{a+b}{2})^{n+1}$ to show we "gain an order" in exactness.*

**(EXTRA): Sketch of interpolation arguments.** Once we have determined what degree of polynomials our Newton Cotes quadrature rules integrate exactly, we can use this fact to derive general error estimates based on the interpolation error formula. Two ideas one can pursue are as follows:

- Because the polynomial $\Psi_2(x)$ itself is a cubic (and the next Newton polynomial), we can add an extra interpolation node (either another observation, or a derivative evaluation ala Hermite). The extra interpolation term $c_2\Psi_2(x)$ adds nothing to either the integral or the Simpson quadrature rule. Using standard interpolation error estimates, we get the error estimate for Simpsons (or odd numbered rules).

- If you follow the procedure in *A proof of the Newton-Cotes quadrature formulas with error term* by Hayes and Rubin (American Mathematical Monthly 1970), they find error bounds of the form:
$$\int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = \frac{f^m(\alpha)}{m!}\int_a^b \left(\int_a^t \Psi_n(t)dt\right)dx \tag{1.21}$$
the reason they can work with the antiderivative of $\Psi$ is because it is either non-negative or non-positive.

## 1.2 Composite rules for Newton-Cotes

While it is possible to increase $n$ for Newton Cotes, there are two limitations to consider: for one, we know interpolation on equispaced nodes can fall prey of the Runge phenomenon; also, some of the Newton-Cotes weights for high enough $n$ are negative. As a result, the quadratures are no longer stable and can result in loss of significance due to cancellation.

Instead, we consider a partition of $[a, b]$ into $m$ sub-intervals $I_j = [x_{j-1}, x_j]$ of size $h_j = x_j - x_{j-1}$. We apply our quadrature rule of choice on each sub-interval, and add the results.

**Composite Trapezoidal** For example, composite Trapezoidal yields:

$$\int_a^b f(x)dx = \sum_{j=1}^{m}\int_{I_j} f(x)dx \simeq \sum_{j=1}^{m} h_j\left(\frac{f(x_{j-1}) + f(x_j)}{2}\right) \tag{1.22}$$

If the partition of our interval is equispaced, then $h_j = h = (b-a)/m$, and

$$\int_a^b f(x)dx \simeq h\sum_{j=1}^{m}\frac{f(x_{j-1}) + f(x_j)}{2} = h\left(\frac{f(a) + f(b)}{2} + \sum_{j=1}^{m-1} f(x_j)\right) \tag{1.23}$$

This can be simply implemented, once again, by forming a vector of weights $\mathbf{w}$ and a vector of evaluations $\mathbf{f} = f(\mathbf{x})$. For composite trapezoidal with $n+1$ equispaced points ($N$ subintervals), this gives us:

$$\begin{aligned}\mathbf{w} &= h\begin{bmatrix}\frac{1}{2} & 1 & \cdots & 1 & \frac{1}{2}\end{bmatrix}\\ \mathbf{f} &= \begin{bmatrix}f(a) & f(x_1) & \cdots & f(x_{n-1}) & f(b)\end{bmatrix}\\ T_h[f] &= \mathbf{w}^T\mathbf{f}\end{aligned}$$

**Composite Simpsons** To be consistent, we consider $n + 1$ equispaced points, where $n$ is even. We then apply Simpson's rule to each of the $m = n/2$ subintervals $I_j = [x_{2j-2}, x_{2j}]$. This gives us:

$$\int_a^b f(x)dx \simeq h \sum_{j=1}^{n/2} \left( \frac{f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})}{3} \right) \tag{1.24}$$

$$= \frac{h}{3} \left( f(a) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n) \right) \tag{1.25}$$

This can be implemented by changing *one line* in the composite trapezoidal pseudocode:

$$\begin{aligned} \mathbf{w} &= \frac{h}{3} \begin{bmatrix} 1 & 4 & 2 & \cdots & 4 & 1 \end{bmatrix} \\ \mathbf{f} &= \begin{bmatrix} f(a) & f(x_1) & \cdots & f(x_{n-1}) & f(b) \end{bmatrix} \\ S_h[f] &= \mathbf{w}^T \mathbf{f} \end{aligned}$$

**Error analysis for composite rules**

The way we do error analysis for composite rules essentially "adds up" the error formulas for each sub-interval, and then uses the Intermediate Value Theorem to simplify.

**Trapezoidal rule**

$$E_h^T[f] = \int_a^b f(x)dx - T_h[f] \tag{1.26}$$

$$= -\frac{h^3}{12} \sum_{j=0}^{n} f''(\alpha_j) \tag{1.27}$$

$$= \left( -\frac{h^3}{12} \frac{(b-a)}{h} \right) \left( \frac{1}{n} \sum_{j=0}^{n} f''(\alpha_j) \right) \tag{1.28}$$

In the last step, we multiply and divide by $n = (b-a)/h$. By Intermediate Value Theorem, since $f''(x)$ is assumed to be continuous, there exists $\alpha \in [a, b]$ such that $f''(\alpha)$ is equal to the average in this error estimate. Thus,

$$E_h^T[f] = -\frac{(b-a)f''(\alpha)}{12} h^2 \tag{1.29}$$

This means that, as we increase the number of points, the error will go to zero like $O(h^2) = O(n^{-2})$. If we double the number of points, we can expect the error to go down by a factor of 4.

**Simpson's rule**

$$E_h^S[f] = \int_a^b f(x)dx - S_h[f] \tag{1.30}$$

$$= -\frac{h^5}{90}\sum_{j=0}^{n/2} f^{(4)}(\alpha_j) \tag{1.31}$$

$$= \left(-\frac{h^5}{90}\frac{(b-a)}{2h}\right)\left(\frac{2}{n}\sum_{j=0}^{n} f^{(4)}(\alpha_j)\right) \tag{1.32}$$

$$= -\frac{(b-a)f^{(4)}(\alpha)}{180}h^4 \tag{1.33}$$

once again, the last step uses the IVT and continuity of the fourth derivative to find $\alpha$. This means if we double the number of points, we can expect the error to go down by a factor of 16.

We can extend this argument to conclude:

**Theorem 1.3 (Composite Newton-Cotes error analysis)** *We consider using the Newton-Cotes composite rule of degree $k$ for an equispaced partition of $[a, b]$ into $n+1$ points, where $n$ is divisible by $k$.*

- ***k+1 is even:*** *for $M_k \in \mathbb{R}, \eta \in [a, b]$,*

$$\int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = \frac{M_n(b-a)}{k} f^{k+1}(\eta)h^{k+1} \tag{1.34}$$

- ***k+1 is odd:*** *For $M_k \in \mathbb{R}, \eta \in [a, b]$,*

$$\int_a^b f(x)dx - \sum_{j=0}^{n} w_j f(x_j) = \frac{M_k(b-a)}{k} f^{k+2}(\eta)h^{k+2} \tag{1.35}$$

## 1.3 More on Trapezoidal rule and higher order methods

The error estimate above tells us that the error for composite Trapezoidal is $O(h^2)$, and so convergence of this rule as a function of $n$ is rather slow. In particular, to reach an error tolerance $\varepsilon = 10^{-p}$, we need $n \sim \frac{1}{\sqrt{\varepsilon}} = 10^{p/2}$ nodes.

We want to improve upon this and create higher-order methods based on the Trapezoidal rule. To do so, it is useful to write down the error for this rule as an asymptotic expansion in $h$. This result is a consequence of the Euler-Maclaurin formula:

**Theorem 1.4** *Let $f \in C^{2p}([a, b])$ for $p \in \mathbb{N}$, and $T_h[f]$ the composite trapezoidal rule applied to $f$ on $n+1$ equispaced nodes. Then,*

$$\int_a^b f(x)dx - T_h[f] = \sum_{k=1}^{p} \frac{B_{2k}}{(2k)!}\left(f^{2k-1}(b) - f^{2k-1}(a)\right)h^{2k} + R_p \tag{1.36}$$

*where $B_{2k}$ is the $2k$ Bernoulli number, and a formula for $R_p$ can be written involving the integral of the product of a Bernoulli function and $f^p(x)$.*

Note that if $f \in C^\infty([a,b])$, this gives us an asymptotic expansion for the error involving even powers of $h$. That is,

$$\int_a^b f(x)dx - T_h[f] = -\frac{f'(b)-f'(a)}{12}h^2 + \frac{f'''(b)-f'''(a)}{720}h^4 + \cdots \tag{1.37}$$

This suggests a number of interesting things about this rule:

- The error for Trapezoidal is entirely due to boundary effects. This suggests adding either the terms from the Euler-Maclaurin formula or extra nodes / weights to *increase* the order of the Trapezoidal rule. There are a number of so-called "corrected Trapezoidal rules", see papers by Rokhlin, Fornberg, Morales, etc.
- For *smooth periodic* functions (where $f$ and its derivatives are periodic and continuous in the interval of periodicity), this implies that the error for trapezoidal goes to zero *faster than any power of $h$*. This is known as *spectral convergence*, and is the reason why we use trapezoidal rule for periodic functions (e.g. to approximate Fourier coefficients or compute the Fourier transform).

**Richardson extrapolation**

Another way we can increase the order of convergence for a method such as that of Trapezoidal rule is generally known as Richardson extrapolation. It is a method that is generally useful in a setting where we have, for a quantity of interest $M$:

- An approximation method $N_1(h)$ that depends on a discretization parameter $h \to 0$. Examples include finite difference methods, Taylor approximation and numerical quadratures.
- An asymptotic error expansion for $M - N_1(h) = K_1 h + K_2 h^2 + K_3 h^3 + \cdots$

The idea then is to compare the asymptotic formulas for $h$ and $h/2$. This yields:

$$M - N_1(h) = K_1 h + K_2 h^2 + K_3 h^3 + \cdots \tag{1.38}$$
$$M - N_1(\frac{h}{2}) = K_1\frac{h}{2} + K_2\frac{h^2}{4} + K_3\frac{h^3}{8} + \cdots \tag{1.39}$$
$$\tag{1.40}$$

If we take twice the second equation minus the first equation, that eliminates the $O(h)$ terms. Thsi gives us:

$$M - \left(2N_1\left(\frac{h}{2}\right) - N_1(h)\right) = C_2 h^2 + C_3 h^3 + \cdots \tag{1.41}$$

with $C_j = K_j(\frac{1}{2^{j-1}} - 1)$. This means that $N_2(h) = 2N_1(\frac{h}{2}) - N_1(h)$ is a method based on $N_1(h)$ and $N_1(h/2)$ that is $O(h^2)$. We can iterate this process once again:

$$M - N_2(h) = C_2 h^2 + C_3 h^3 + \cdots \tag{1.42}$$

$$M - N_2(\frac{h}{2}) = C_2 \frac{h^2}{4} + C_3 \frac{h^3}{8} + \cdots \tag{1.43}$$

$$3M - \left(4N_1\left(\frac{h}{2}\right) - N_1(h)\right) = D_3 h^3 + D_4 h^4 \cdots \tag{1.44}$$

$$\tag{1.45}$$

In general, this gives us an approximation

$$N_j(h) = \frac{2^{j-1} N_{j-1}\left(\frac{h}{2}\right) - N_{j-1}(h)}{2^{j-1} - 1} \tag{1.46}$$

that is ultimately based on $N_1(h), N_1(h/2), \ldots, N_1(h/2^{j-1})$ and is $O(h^j)$

**Romberg quadrature**

Romberg quadrature is the application of Richardson extrapolation to the composite Trapezoidal rule. We need to modify Richardson a little bit because the asymptotic expansion for Trapezoidal only has even powers of $h$. Given $h = (b-a)/n$, we define the Romberg rule of degree 1 as $R_1(h) = T_h[f]$, and for $j > 1$,

$$R_j(h) = \frac{4^{j-1} R_{j-1}\left(\frac{h}{2}\right) - R_{j-1}(h)}{4^{j-1} - 1} \tag{1.47}$$

which is based on $R_1(h), R_1(h/2), \ldots, R_1(h/2^{j-1})$ and is $O(h^{2j})$.

**Practical implementation and pseudocode:** Given $n = 2^L n_0$ for $n_0 \in \mathbb{N}$, we can compute the trapezoidal rule for spacings $h_0 = (b-a)/n_0$ and $h_j = h_{j-1}/2$ for $j = 1, \ldots, L-1$. That is, we can go from the coarsest to the finest mesh. We then use the formula for the Romberg rule $L$ times, until we compute $R_L(h_0)$. This can be arranged on a triangular matrix ala Newton interpolation coefficients:

$$\begin{bmatrix} R_1(h_0) & R_2(h_0) & R_3(h_0) & \cdots & R_L(h_0) \\ R_1(\frac{h_0}{2}) & R_2(\frac{h_0}{2}) & R_3(\frac{h_0}{2}) & \cdots & \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ R_1(\frac{h_0}{2^{L-1}}) & R_2(\frac{h_0}{2^{L-1}}) & & \cdots & \\ R_1(\frac{h_0}{2^L}) & & & \cdots & \end{bmatrix} \tag{1.48}$$

We note that the best Romberg rules given this data are situated on the antidiagonal of this triangular matrix. If we stop at degree $p < L$, we get $R_p(h_0/2^{L-p+1})$ which is $O(h^{2p})$.

**Extra: deriving nodes and weights for Romberg quadratures.** For a given Romberg quadrature as we have derived them above, we can go through the process of figuring out nodes and weights formulas. For instance, since $R_2(h)$ is derived from a linear combination of $T_h$ and $T_{h/2}$, the quadrature nodes are simply the $2n+1$ equispaced nodes $x_j = a + (h/2)j, j = 0, \ldots, 2n$. If we line up nodes for both trapezoidal rules and add up corresponding weights, we get

$$\mathbf{w}_2 = \frac{h}{6}\begin{bmatrix} 1 & 4 & 2 & \cdots & 2 & 4 & 1 \end{bmatrix}$$

which is the Simpson rule! We can iterate on this process: $R_3(h)$ is also a Newton-Cotes rule (Boole's rule, obtained from quartic polynomial interpolants)

$$\mathbf{w}_3 = \frac{2h}{45}\begin{bmatrix}7 & 32 & 12 & 32 & 14\cdots & 14 & 32 & 12 & 32 & 7\end{bmatrix}$$

From $R_4$ onwards, however, Romberg is not Newton-Cotes (not interpolation based), and unlike NC, it preserves a very important property for stability: all of its weights are non-negative. For instance,

$$\mathbf{w}_4 = \frac{h}{2835}\begin{bmatrix}217 & 1024 & 352 & 1024 & 436 & 1024 & 352 & 1024 & 434 & \cdots\end{bmatrix}$$

## 1.4 Gaussian quadratures

The Newton-Cotes quadratures are interpolation-based, and so once we decide on $k+1$ interpolation nodes $x_j$, this immediately sets the quadrature weights $w_j = \int_a^b L_j(x)dx$. We recall the following error estimates, where $h = (b-a)/n$:

- For a rule of degree $k$ with $k+1$ even (e.g. Trapezoidal), it is exact for polynomials in $\mathcal{P}_k$. The error for one interval is $O(h^{k+2})$; for the composite rule, it is $O(h^{k+1})$.
- For a rule of degree $k$ with $k+1$ odd (e.g. Simpsons), it is exact for polynomials in $\mathcal{P}_{k+1}$. The error for one interval is $O(h^{k+3})$; for the composite rule, it is $O(h^{k+2})$.

Based on this, we could ask: for a single interval, is this the best we can do for a rule with $k+1$ nodes and as many weights? If we are free to determine nodes and weights ($2k+2$ unknowns), can we make a quadrature rule that is exact for polynomials of degree higher than $k$ and $k+1$?

The answer is yes: Gaussian quadrature is a method that integrates polynomials up to degree $2k+1$ exactly. As a result, the error for a single interval is $O(h^{2k+3})$, and for the composite rule this is $O(h^{2k+2})$. This is optimal: no quadrature rule with $k+1$ points can do better than this.

We have a formula for a general quadrature rule $Q[f] = \sum_{j=0}^{k} w_j f(x_k)$, which is determined by $2k+2$ unknowns $(x_j, w_j)_{j=0}^{k}$. Consider the case $k=1$, and interval $[-1,1]$. We have 4 unknowns, and so, this is enough to ask that the rule be exact for polynomials up to degree 3:

$$\begin{aligned}
w_0 + w_1 &= \int_{-1}^{1} dx = 2\\
w_0x_0 + w_1x_1 &= \int_{-1}^{1} xdx = 0\\
w_0x_0^2 + w_1x_1^2 &= \int_{-1}^{1} x^2dx = \frac{2}{3}\\
w_0x_0^3 + w_1x_1^3 &= \int_{-1}^{1} x^3dx = 0
\end{aligned}$$

This is a *non-linear* system of 4 equations in the 4 unknowns. It has a unique solution (up to re-labeling of the weights and nodes) with $w_0 = w_1 = 1$ and $x_0 = \frac{-1}{\sqrt{3}}, x_1 = \frac{1}{\sqrt{3}}$. This system is small enough that it can be done by hand using algebra. The trick is:

1. If we multiply eq (ii) by $x_0$ and separately, multiply eq (ii) by $x_1$, we obtain two equations. If we add them, we can use eqs (i) and (iii) to obtain $x_0 x_1 = -1/3$.
2. We use a similar trick, but now multiplying eq (ii) by $x_0$ and again but by $x_1$, and adding them up. Using equations (ii) and (iv), we get that $x_0 + x_1 = 0$.
3. These two equations are enough to determine $x_0, x_1$. We can plug these in and find the weights.

This rule is exact for polynomials up to degree 3, and so using the Taylor expansion error analysis technique, we can conclude that the error is of the form $Cf^4(\eta)(b-a)^5$, like that of Simpsons rule. Except this quadrature rule uses only two nodes! This was quite cumbersome to solve. However, it turns out we can derive this same quadrature rule using an orthogonal basis of polynomials.

**Gauss-Legendre quadrature**

We are interested in computing $\int_{-1}^{1} f(x)dx$. We consider the basis of orthogonal polynomials for the weight function $w(x) = 1$, that is, the Legendre polynomial basis $P_j(x)$. Now, given a rule with $n+1$ nodes (and as many weights), consider the $(n+1)$ Legendre polynomial $P_{n+1}(x)$. Recall that, by construction,

- $< P_{n+1}, P_j >= \int_{-1}^{1} P_{n+1}(x)P_j(x)dx = 0 \quad \forall j = 0, 1, \ldots, n$

- For all $q \in \mathcal{P}_n$, $< P_{n+1}, q >= 0$.

Now, take $p \in \mathcal{P}_{2n+1}$. Using polynomial division, it can be written as $p(x) = P_{n+1}(x)q(x) + r(x)$, where both quotient $q(x)$ and residual $r(x)$ are polynomials in $\mathcal{P}_n$. Now, consider the integral of $p(x)$:

$$\int_{-1}^{1} p(x)dx = \int_{-1}^{1} P_{n+1}(x)q(x)dx + \int_{-1}^{1} r(x)dx = \int_{-1}^{1} r(x)dx \tag{1.49}$$

due to the orthogonality condition for $P_{n+1}$. This means the integral of $p$ can be written as the integral of a polynomial of degree $n$. This is promising, as we only have $n+1$ nodes.

Now, we write the quadrature rule for $p(x)$:

$$Q[f] = \sum_{j=0}^{n} w_j P_{n+1}(x_j)q(x_j) + \sum_{j=0}^{n} w_j r(x_j) \tag{1.50}$$

At this point, we note that

- Given a set of $n+1$ distinct nodes $x_j$, we can *always* find $w_j$ such that the rule is exact for polynomials in $\mathcal{P}_n$ (we solve an invertible linear system for the weights).
- If we do, then
$$\sum_{j=0}^{n} w_j r(x_j) = \int_{-1}^{1} r(x)dx = \int_{-1}^{1} f(x)dx \tag{1.51}$$
- So, out of all possible choices, we must pick nodes such that for all $q \in \mathcal{P}_n$,
$$\sum_{j=0}^{n} w_j P_{n+1}(x_j)q(x_j) = 0 \tag{1.52}$$

The only choice here is to pick $\{x_j\}_{j=0}^n$ to be *the* $n+1$ *distinct roots of* $P_{n+1}$. We can determine the weights either using Lagrange interpolation or solving a linear system to make the rule exact up to degree $n$ (there's also a formula for them! More on this below).

**Gaussian quadrature for weight function $w(x)$:** say we now are interested in computing $\int_a^b f(x)w(x)dx$ for integrable weight function $w(x) \geq 0$. We consider the corresponding basis of orthogonal polynomials $\Phi_j(x)$ (e.g. Chebyshev, Laguerre, Hermite, etc). We want to construct a quadrature rule with $n+1$ nodes that is exact for these integrals if $f(x)$ is a polynomial of degree up to $2n+1$.

All of the construction above works exactly the same: we set $x_j$ to be the $n+1$ zeroes of $\Phi_{n+1}(x)$. We then set the weights $w_j$ such that the rule satisfies $\sum_{j=0}^n w_j r(x_j) = \int_a^b r(x)w(x)dx$ for all $r \in \mathcal{P}_n$.

For all of this to work, we must prove a few theorems:

**Theorem 1.5 (Roots of orthogonal polynomial are simple)** *Let $\Phi_j$ be a collection of orthogonal polynomials on $[a,b]$ with respect to the weight function $w(x)$. Then, $\Phi_n$ has $n$ distinct real roots $\{x_j\}_{j=0}^n$ in $[a,b]$.*

The proof of this statement is by contradiction. Assume $\Phi_n$ only has $\ell < n$ distinct roots inside $[a,b]$ (this covers both multiple roots and roots outside the interval). This means there are at most $q \leq \ell$ locations $\{x_j\}_{j=1}^q$ in $[a,b]$ such that $\Phi_n$ changes sign. Define $\Psi_q(x) = \prod_{j=1}^q (x-x_j) \in \mathcal{P}_q$. Now, let's look at $\Phi_n(x)\Psi_q(x)$:

- This product, by construction, *does not change sign* in $[a,b]$
- That means $\int_a^b \Phi_n(x)\Psi_q(x)w(x)dx \neq 0$!

This is a contradiction, since it means $< \Phi_n, \Psi_q >_w \neq 0$.

We have the following results (combining our construction with the previous theorem):

**Theorem 1.6 (Gaussian quadrature error exactness)** *Suppose $I_w[f]$ is the Gaussian quadrature rule for $[a,b]$ and weight function $w(x)$. Then, it is exact for polynomials up to degree $2n+1$.*

**Theorem 1.7 (Gaussian quadrature error estimate)** *Suppose $f \in C^{2n+2}([a,b])$. Then, there exists $\eta \in [a,b]$ such that the error satisfies:*

$$E_w[f] = \int_a^b f(x)w(x)dx - I_w[f] = \frac{f^{2n+2}(\eta)}{(2n+2)!}\int_a^b \Psi(x)^2 w(x)dx \tag{1.53}$$

The proof of this statement is via Hermite interpolation. Applying the quadrature rule to the Hermite interpolant $h(x) \in \mathcal{P}_{2n+1}$ defined on the $n+1$ Gaussian-quadrature nodes, this rule is *exact* for the interpolant. Also, the quadrature rule for $f(x)$ is the same as that for $h(x)$ (again, by construction). So,

$$E_w[f] = \int_a^b (f(x)-p(x))w(x)dx = \int_a^b \frac{f^{2n+2}(\eta_x)}{(2n+2)!}\Psi(x)^2 w(x)dx \tag{1.54}$$

Once again, because $\Psi(x)^2 w(x) \geq 0$, by generalized integral MVT, there exists $\eta \in [a,b]$ such that

$$E_w[f] = \frac{f^{2n+2}(\eta)}{(2n+2)!}\int_a^b \Psi(x)^2 w(x)dx \tag{1.55}$$

**Implementation / pseudocode for Gaussian quadrature** Given $[a,b]$ and weight function $w(x)$

1. Pick (or build) the corresponding family of orthogonal polynomials up to $\Phi_{n+1}$.
2. Find the roots of $P_{n+1}$ (either using Newton-Raphson, or via the Golub-Welsch algorithm)
3. Find the weights (via integrals of Lagrange, linear system, or a formula)

**How to find the roots:** If you can't find them on a table (to machine precision or higher), there are a few very efficient ways to compute the roots of $\Phi_{n+1}$ accurately.

If we want to find the roots via Newton-Raphson, one handy result is known as the *interlacing theorem*: that is, if you know the $n$ roots $x_j^n$ for $\Phi_n$, this defines a partition $a < x_1^n < \cdots < x_n^n < b$ involving $n+1$ subintervals. The *interlacing theorem* states there is a single root $x_j^{n+1}$ for $\Phi_{n+1}$ on each subinterval of this partition. We can pick the midpoint of each interval and run the Newton method.

The Golub Welsch algorithm, on the other hand, is a way to write down a so-called Jacobi matrix $\mathbf{J}_n$ such that its eigenvalues $\{\lambda_j^n\}$ are the roots of $\Phi_{n+1}$. The $n+1$ rows of this matrix correspond to the recursion relation

$$\Phi_j(\lambda) = (\lambda - \beta_j)\Phi_{j-1}(\lambda) - \gamma_j \Phi_{j-2}(\lambda) \quad j = 1, \ldots, n+1 \tag{1.56}$$

Given the vector $\mathbf{p} = \begin{bmatrix}\Phi_0(\lambda) & \Phi_1(\lambda) & \cdots & \Phi_n(\lambda)\end{bmatrix}$, we can write this in matrix form as $\mathbf{Jp} = \lambda\mathbf{p} - \Phi_{n+1}(\lambda)\mathbf{e}_n$, where

$$\mathbf{J}_n = \begin{bmatrix} \beta_1 & 1 & 0 & \cdots & 0 & 0 \\ \gamma_2 & \beta_2 & 1 & \cdots & 0 & 0 \\ 0 & \gamma_3 & \beta_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_n & 1 \\ 0 & 0 & 0 & \cdots & \gamma_{n+1} & \beta_{n+1} \end{bmatrix} \tag{1.57}$$

This matrix is tri-diagonal (and so, upper Hessenberg), but non-symmetric. One can modify the recursion relation so that the resulting matrix is tri-diagonal and symmetric; this is equivalent to applying a similarity transformation on the matrix $\mathbf{J}$ above of the form $\mathbf{DJD^{-1}}$ where $\mathbf{D}$ is *diagonal*, so that the resulting matrix is symmetric. Having a symmetric tri-diagonal matrix allows us to compute its eigenvalues *very efficiently* using methods like the QR algorithm and Krylov subspace methods.

**How to find the weights:** We have reviewed two methods to find the weights, namely, to find the (weighted) integrals of the Lagrange polynomials, or to explicitly write down a linear system for them that imposes the rule is exact for polynomials up to degree $n$. However, it turns out we can also use the properties of orthogonal polynomials to find formulas for the weights! For the Gauss-Legendre weights, given the quadrature nodes $x_j$,

$$w_j = \frac{2}{(1 - x_j^2)[P'_{n+1}(x_j)]^2} \tag{1.58}$$

Note that all this involves is computing the derivatives of the $n+1$ Legendre polynomial at each of its roots. We can use formulas for Legendre polynomials and their derivatives to do this efficiently.

For a general orthogonal polynomial $\Phi_{n+1}$, the weights can be expressed as a quotient of integrals:

$$w_j = \frac{a_{n+1}}{a_n} \frac{\int_a^b \Phi_n^2 w(x)dx}{\Phi'_{n+1}(x_j)\Phi_n(x_j)} \tag{1.59}$$

where $a_k$ is the leading coefficient of $\Phi_k$. Note the integral in the numerator is $\|\Phi_n\|_w^2$ (this is known for established families of polynomials, and is not hard to compute), and this formula again involves evaluating the derivatives of the $n+1$ polynomial at the nodes.

## 1.5 Adaptive quadrature

Imagine that we are tasked with integrating a function $f(x)$ in the interval $[a, b]$. Let's say we pick a composite quadrature, and we are now tasked to decide what the number of points (total number of degrees of freedom) should be to compute this integral for a given target accuracy $\varepsilon$.

Now, assume this function is "boring" in some parts of $[a, b]$ (it is very smooth and not varying too much), and "interesting" in others (more peaked, more oscillatory, derivatives are large in absolute value). From our error estimates for Newton-Cotes and Gaussian quadrature, we know that even if the assumptions on $f(x)$ are met, we require many more points for the quadrature to converge to a desired target accuracy for an "interesting" function than for a "boring" one.

This tells us that if we pick a uniform partition to define the "panels" for our composite quadrature, we run one of two risks. If we pick too coarse of a discretization (based on the boring part), we will have huge errors coming from the "interesting" parts of $f(x)$. If we pick too fine of a discretization (based on the interesting part), we are incredibly wasteful (the panels in the boring part are way over-resolved).

This suggests an **adaptive** approach: we want to figure out an automatic way to only put quadrature points where they are needed to reach a desired accuracy. And, of course, we want some guarantees that this adaptive method will give us as good of a result as the fine, uniform discretization.

**Sketch of a general adaptive algorithm:**

Say you pick a composite quadrature of your liking: it could be Newton-Cotes, Gaussian quadrature, or something else. Given an interval $I$, we denote the *single interval* quadrature rule applied to this $Q_I[f]$. Given a partition of $[a, b]$ into intervals $I_j = [x_{j-1} \;\; x_j]$, the composite quadrature is then the sum of $Q_{I_j}[f]$. Unless required, from now, we will drop the $[f]$ to make notation simple.

The idea is as follows: For a given interval $I_0 = [a, b]$, let us compare the quadrature applied to it with a composite quadrature using *two identical panels* $I_1 = [a, m]$ and $I_2 = [m, b]$ where $m = (a + b)/2$. That is, we compare $Q_{I_0}$ with $Q_{I_1} + Q_{I_2}$. We claim that:

- $|Q_{I_0} - (Q_{I_1} + Q_{I_2})|$ is (asymptotically) proportional to an estimate of the absolute error of approximating our integral with either $I_0$ or $I_1$ plus $I_2$.
- For similar reasons, we claim $\frac{|Q_{I_0} - (Q_{I_1} + Q_{I_2})|}{|Q_{I_1} + Q_{I_2}|}$ is (asymptotically) proportional to the corresponding relative error.

If this is true, we come up with the following algorithm:

1. Start with the interval $[a, b]$. Compute the relevant absolute (or relative) error estimates using $Q_{I_0} - (Q_{I_1} + Q_{I_2})$.

2. **IF** if this is accurate enough, return $I = Q_{I_1} + Q_{I_2}$.
3. **ELSE**, if this is not accurate enough, subdivide $I_0$ into two intervals $I_1$ and $I_2$.
4. For each of the intervals which haven't returned an integral yet, repeat the process outlined above.

Note that the condition to stop and return an integral estimate for a given interval should include a maximum number of times we can subdivide, or a maximum number of points (overall); perhaps even that intervals are not allowed to be smaller than some size.

This is done for two reasons: one, if the assumptions behind the quadrature are not met ($f(x)$ is not sufficiently smooth), this could run forever. More importantly, we want to set limits to the amount of time, resources and maximum degrees of freedom we are allowed to spend on this.

**Example: Adaptive Simpson's rule**

Given a quadrature of choice like Simpson, we can apply the relevant error estimates to analyze the difference between the one and two interval quadratures. When going over this, you should also recall the derivation of Romberg quadrature from Trapezoidal, as some of the analysis is similar.

Let us denote $Q_I$ as $S_I$ for Simpson's rule. We know that, for some $\eta, \eta_2 \in [a, b]$:

$$S_{[a,b]} - \int_a^b f(x)dx = \frac{f^4(\eta)}{90}\left(\frac{b-a}{2}\right)^5 \tag{1.60}$$

$$(S_{[a,m]} + S_{[m,b]}) - \int_a^b f(x)dx = \frac{f^4(\eta_2)(b-a)}{180}\left(\frac{b-a}{4}\right)^4 = \frac{1}{2^4}\left(\frac{f^4(\eta_2)}{90}\left(\frac{b-a}{2}\right)^5\right) \tag{1.61}$$

Taking the difference between these two:

$$S_{[a,b]} - (S_{[a,m]} + S_{[m,b]}) = \left(f^4(\eta) - \frac{1}{2^4}f^4(\eta_2)\right)\frac{1}{90}\left(\frac{b-a}{2}\right)^5 \tag{1.62}$$

by continuity of $f^4(x)$, this shows us that

$$I^{est}_{[a,b]} = S_{[a,b]} - (S_{[a,m]} + S_{[m,b]}) \simeq \frac{2^4-1}{2^4}\frac{f^4(\eta)}{90}\left(\frac{b-a}{2}\right)^5 \tag{1.63}$$

That is, 15/16 of the quadrature error for $[a, b]$ or 15 times the error for the composite rule with two intervals $[a, m]$ and $[m, b]$. If, for instance, I want the absolute error to be less than $\varepsilon$, I can make part of the stopping criterion to be $S_{[a,b]} - (S_{[a,m]} + S_{[m,b]}) < \frac{1}{15}\varepsilon$.

**Note:** Some implementations of adaptive Simpson quadrature return $(S_{[a,m]} + S_{[m,b]}) + \frac{1}{15}(S_{[a,b]} - (S_{[a,m]} + S_{[m,b]})$. This should remind you of one round of Romberg quadrature, and is using the difference between these two as a correction to improve upon the accuracy and convergence of the Simpson's rule.

**Implementation of adaptive quadrature:**

Regardless of the composite rule of choice, there are (largely) two ways of implementing this adaptive quadrature idea: Recursive and Non-recursive. It is generally recommended to be very careful with recursive implementation, as it might lead to numerically instability as errors propagate.

For either implementation, we can think of it in terms of a binary tree structure of intervals, where we say $I$ is a parent of $I_1$ and $I_2$ if these result from the bisection of $I$. The adaptive quadrature algorithms then generate this tree and add up the results from each one of the tree's leaves (terminal nodes).

**Recursive algorithm:** The main idea behind a recursive implementation is for the main adaptive step to take in data from $[a, b]$, perform the error estimate check, and either directly return $Q_{I_1} + Q_{I_2}$ (and terminate), OR call itself (hence the recursion) for $I_1$ and for $I_2$, and add the results from both. A pseudocode could be written as follows:

```
1. Given information for $[a,b]$
2. Compute $I^{est}_{[a,b]} = Q_{[a,b]} - (Q_{[a,m]} + Q_{[m,b]})$
3. **IF** $I^{est}_{[a,b]} < TOL(\varepsilon)$ or $Level > L_{MAX}$
4.     return $I_{FIN} = Q_{[a,m]} + Q_{[m,b]}$
5. **ELSE:**
6.     $I_L$ = Adaptive_Quad(Left interval info)
7.     $I_R$ = Adaptive_Quad(Right interval info)
8.     return $I_{FIN} = I_L + I_R$
```

As written, this algorithm is traversing the tree all the way to the left-most leaf, and then making its way across to the right.

**Non-recursive algorithm:** The idea behind non-recursive implementation is to create the tree structure level by level by writing down a list of intervals "to be processed". For each interval $I$ in this list, we compute the error estimate and decide whether to subdivide or not. If we don't subdivide, we add the integral estimate to our overall result, and delete $I$ from the list. If we decide to subdivide, we remove parent interval $I$ and add the two children intervals $I_1$ and $I_2$ to the list.

A pseudocode could be written as follows:

```
1. Set Interval_List = $\{[a,b]\}$, Levels = $\{0\}$, $Sum = 0$
2. **WHILE** Interval_List is not empty
3.     Read interval $I_k = [a,b]$
4.     Compute $I^{est}_{[a,b]} = Q_{[a,b]} - (Q_{[a,m]} + Q_{[m,b]})$
5.     **IF** $I^{est}_{[a,b]} < TOL(\varepsilon)$ or $Level > L_{MAX}$
6.         $Sum = Sum + (Q_{[a,m]} + Q_{[m,b]})$
```

```
7.          Remove interval from list, record endpoints if needed
8.      ELSE:
9.          Remove interval from list. Add $[a, m], [a, b]$ to the list.
10. Return $Sum$.
```