LECTURE 25: *Gaussian quadrature: nodes, weights; examples; extensions*

3.5 *Computing Gaussian quadrature nodes and weights*

When first approaching Gaussian quadrature, the complicated characterization of the nodes and weights might seem like a significant drawback. For example, if one approximates an integral with an $(n+1)$-point Gaussian quadrature rule and finds the accuracy insufficient, one must compute an entirely new set of nodes and weights for a larger n from scratch.

Many years ago, one would need to look up pre-computed nodes and weights for a given rule in book of mathematical tables, and one was thus limited to using values of n for which one could easily find tabulated values for the nodes and weights.

However, in a landmark paper of 1969, Gene Golub and John Welsch found a nice characterization of the nodes and weights in terms of a symmetric matrix eigenvalue problem. Given the existence of excellent algorithms for computing such eigenvalues, one can readily compute Gaussian quadrature nodes for arbitrary values of n. Some details of this derivation are left for a homework exercise, but we summarize the results here.

G. H. Golub and J. H. Welsch, "Calculation of Gauss Quadrature Rules," *Math. Comp.* 23 (1969) 221–230.

In general, such algorithms require $\mathcal{O}(n^3)$ operations to compute all eigenvalues and eigenvectors of an $n \times n$ matrix; for the modest values of n most common in practice (n in the tens or at most low hundreds), this expense is not onerous. Exploiting the structure of $\mathbf{J}_n$, this algorithm could be sped up to $\mathcal{O}(n^2)$.

One can show, via the discussion in Section 2.5, that, given values $\phi_{-1}(x) = 0$ and $\phi_0(x) = 1$ the subsequent orthogonal polynomials can be generated via the three-term recurrence relation

$$\phi_{k+1}(x) = x\,\phi_k(x) - \alpha_k \phi_k(x) - \beta_k \phi_{k-1}(x). \tag{3.4}$$

The values of $\alpha_0, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$ follow from the Gram–Schmidt process used in Section 2.5. Later we will also need the definition

A good source for the values of $\{\alpha_k\}$ and $\{\beta_k\}$ is Table 1.1 in Walter Gautschi's *Orthogonal Polynomials*, Oxford University Press, 2004.

$$\beta_0 := \langle 1, 1 \rangle = \int_a^b w(x)\,\mathrm{d}x.$$

Given a fixed value of n, collect the coefficients $\{\alpha_k\}_{k=0}^n$ and $\{\beta_k\}_{k=1}^n$ and use them to populate the *Jacobi matrix*

$$\mathbf{J}_n = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \ddots & \ddots & \\ & & \ddots & \alpha_{n-1} & \sqrt{\beta_n} \\ & & & \sqrt{\beta_n} & \alpha_n \end{bmatrix}. \tag{3.5}$$

The following theorem (whose proof is left for as a homework exercise) gives a ready way to compute the roots of ϕ_{n+1}.

Theorem 3.9. Let $\{\phi_j\}_{j=0}^{n+1}$ denote a sequence of monic orthogonal polynomials generated by the recurrence relation (3.4). Then λ is a root of ϕ_{n+1} if and only if λ is an eigenvalue of $\mathbf{J}_n$ with corresponding eigenvector

$$\mathbf{v}(\lambda) = \begin{bmatrix} \phi_0(\lambda) \\ \phi_1(\lambda)/\sqrt{\beta_1} \\ \phi_2(\lambda)/\sqrt{\beta_1\beta_2} \\ \vdots \\ \phi_n(\lambda)/\sqrt{\beta_1\cdots\beta_n} \end{bmatrix}. \tag{3.6}$$

Theorem 3.9 thus gives a convenient way to compute the nodes of a Gaussian quadrature rule. Given the nodes, one could compute the weights using either of the formulas (3.3) involving Lagrange basis functions. However, Golub and Welsch proved that these same weights could be extracted from the eigenvectors of the Jacobi matrix $\mathbf{J}_n$. Label the eigenvalues of $\mathbf{J}_n$ as

$$\lambda_0 < \lambda_1 < \cdots < \lambda_n$$

and the corresponding eigenvectors, given by the formula (3.6), as

$$\mathbf{v}_0 = \mathbf{v}(\lambda_0), \quad \mathbf{v}_1 = \mathbf{v}(\lambda_1), \quad \ldots, \quad \mathbf{v}_n = \mathbf{v}(\lambda_n).$$

Then the weights for $n+1$-point Gaussian quadrature can be computed as

$$w_j = \beta_0 \frac{1}{\|\mathbf{v}_j\|_2^2}, \tag{3.7}$$

Note: assumes $(\mathbf{v}_j)_1 = \phi_0(\lambda_j) = 1$.

where $\|\cdot\|_2$ is the vector 2-norm.

The formula (3.7) relies on the specific form of the eigenvector in (3.6). If the eigenvector is normalized differently (e.g., MATLAB's `eigs` routine gives unit eigenvectors, $\|\mathbf{v}_j\|_2 = 1$, then one should use the general formula

$$w_j = \beta_0 \frac{(\mathbf{v}_j)_1^2}{\|\mathbf{v}_j\|_2^2}, \tag{3.8}$$

where $(\mathbf{v}_j)_1$ is the first entry in the eigenvector $\mathbf{v}_j$.

3.5.1 Examples of Gaussian Quadrature

Let us examine four well-known Gaussian quadrature rules.

method	*interval*, (a, b)	*weight*, $w(x)$
Gauss–Legendre	$(-1, 1)$	1
Gauss–Chebyshev	$(-1, 1)$	$\dfrac{1}{\sqrt{1-x^2}}$
Gauss–Laguerre	$(0, \infty)$	e^{-x}
Gauss–Hermite	$(-\infty, \infty)$	e^{-x^2}

The weight function plays an essential role here. For example, a Gauss–Chebyshev quadrature rule with $n+1 = 5$ points will exactly integrate polynomials of degree $2n+1 = 9$ *times the weight function*. Thus this rule *will* exactly integrate

$$\int_{-1}^{1} \frac{x^9}{\sqrt{1-x^2}}\,\mathrm{d}x,$$

but it *will not* exactly integrate

$$\int_{-1}^{1} x^{10}\,\mathrm{d}x.$$

This is a subtle point that many students overlook when first learning about Gaussian quadrature.

Example 3.2. Gauss–Legendre Quadrature
When numerical analysts speak of "Gaussian quadrature" without further qualification, they typically mean Gauss–Legendre quadrature, i.e., quadrature with the weight function $w(x) = 1$ (perhaps over a transformed domain (a, b); see Section 3.5.2.) As discussed in Section 2.5.1, the orthogonal polynomials for this interval and weight are called *Legendre polynomials*. To construct a Gaussian quadrature rule with $n+1$ points, determine the roots of the degree-$(n+1)$ Legendre polynomial, then find the associated weights.

First consider $n = 1$. The quadratic Legendre polynomial is

$$\phi_2(x) = x^2 - 1/3,$$

and from this polynomial one can derive the 2-point quadrature rule that is exact for cubic polynomials, with roots $\pm 1/\sqrt{3}$. This agrees with the special 2-point rule derived in Section 3.4.1. The values for the weights follow simply, $w_0 = w_1 = 1$, giving the 2-point Gauss–Legendre rule

$$I_n(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3})$$

that exactly integrates polynomials of degree $2n+1 = 3$, i.e., all cubics.

Recall that Simpson's rule also exactly integrates cubics, but it requires *three* f evaluations, rather than the two f evaluations required of this rule.

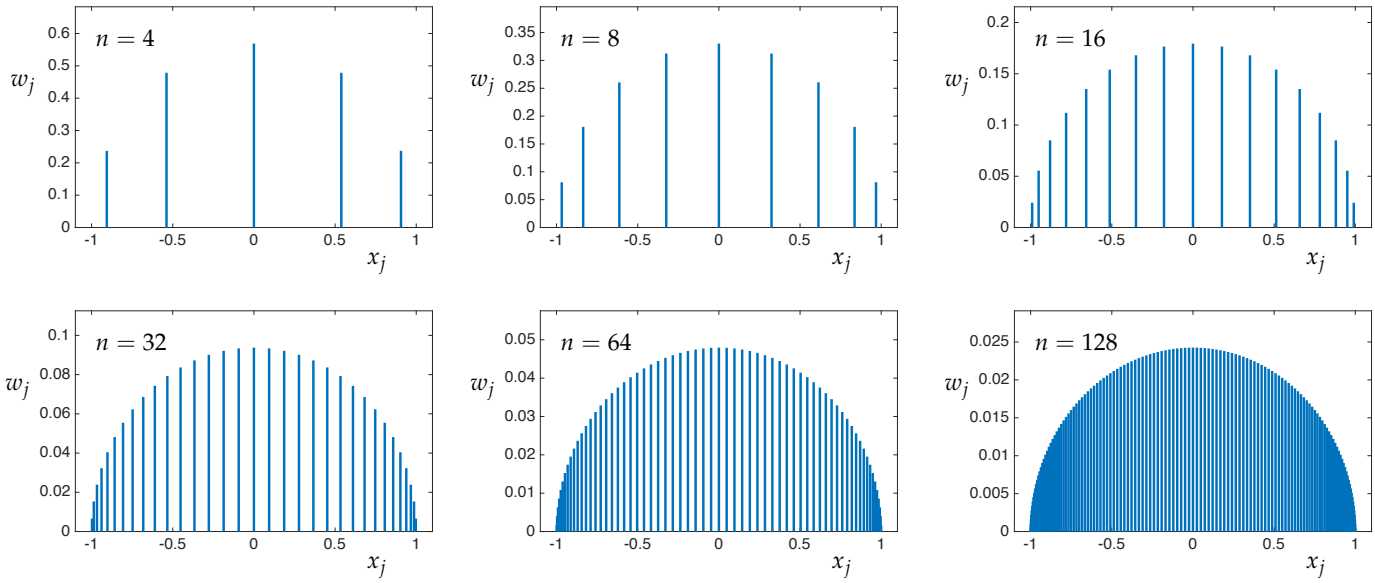

Figure 3.9: Nodes and weights of Gauss–Legendre quadrature, for various values of n. In each case, the location of the vertical line indicates x_j, while the height of the line shows w_j.

For Gauss–Legendre quadrature rules based on larger numbers of points, we can compute the nodes and weights using the symmetric eigenvalue formulation discussed in Section 3.5. For this weight, one can show

$$\begin{aligned} \alpha_k &= 0, && k = 0, 1, \ldots; \\ \beta_0 &= 2; \\ \beta_k &= \frac{k^2}{4k^2 - 1}, && k = 1, 2, 3, \ldots. \end{aligned}$$

Figure 3.9 shows the nodes and weights for six values of n, as computed via the eigenvalue problem. Notice that the points are not uniformly spaced, but are slightly more dense at the ends of the interval. Moreover, the weights are smaller at these ends of the interval.

The table below shows nodes and weights for $n = 4$, as computed in MATLAB.

j	*nodes*, x_j	*weights*, w_j
0	−0.906179845938664	0.236926885056189
1	−0.538469310105683	0.478628670499366
2	0.000000000000000	0.568888888888889
3	0.538469310105683	0.478628670499367
4	0.906179845938664	0.236926885056189

Example 3.3. Gauss–Chebyshev quadrature
Another popular class of Gaussian quadrature rules use as their nodes the roots of the Chebyshev polynomials. The standard degree-

k Chebyshev polynomial is defined as

$$T_k(x) = \cos(k \cos^{-1} x),$$

which can be generated by the recurrence relation

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x).$$

with $T_0(x) = 1$ and $T_1(x) = x$. These Chebyshev polynomials are orthogonal on $(-1, 1)$ with respect to the weight function

$$w(x) = \frac{1}{\sqrt{1-x^2}}.$$

The degree-$(n+1)$ Chebyshev polynomial has the roots

$$x_j = \cos\Big(\frac{(j+1/2)\pi}{n+1}\Big), \qquad j = 0, \ldots, n.$$

In this case all the weights work out to be identical; one can show

$$w_j = \frac{\pi}{n+1}$$

for all $j = 0, \ldots n$. Figure 3.10 shows these nodes and weights. One can also define the *monic* Chebyshev polynomials according to the recurrence (3.4) with

See Süli and Mayers, Problem 10.4 for a sketch of a proof.

$$\begin{aligned}
&\alpha_k = 0, && k = 0, 1, \ldots; \\
&\beta_0 = \pi; \\
&\beta_1 = 1/2; \\
&\beta_k = 1/4, && k = 2, 2, 3, \ldots.
\end{aligned}$$

The resulting polynomials are scaled versions of the usual Chebyshev polynomials $T_{k+1}(x)$, and thus have the same roots.

Again, we emphasize that the weight function plays a crucial role: the Gauss–Chebyshev rule based on $n+1$ interpolation nodes will exactly compute integrals of the form

$$\int_{-1}^{1} \frac{p(x)}{\sqrt{1-x^2}}\,\mathrm{d}x$$

for all $p \in \mathcal{P}_{2n+1}$. For a general integral

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\,\mathrm{d}x.$$

the quadrature rule should be implemented as

$$I_n(f) = \sum_{j=0}^{n} w_j f(x_j);$$

Note that the $1/\sqrt{1-x^2}$ component of the integrand is not evaluated here; its influence has already been incorporated into the weights $\{w_j\}$.

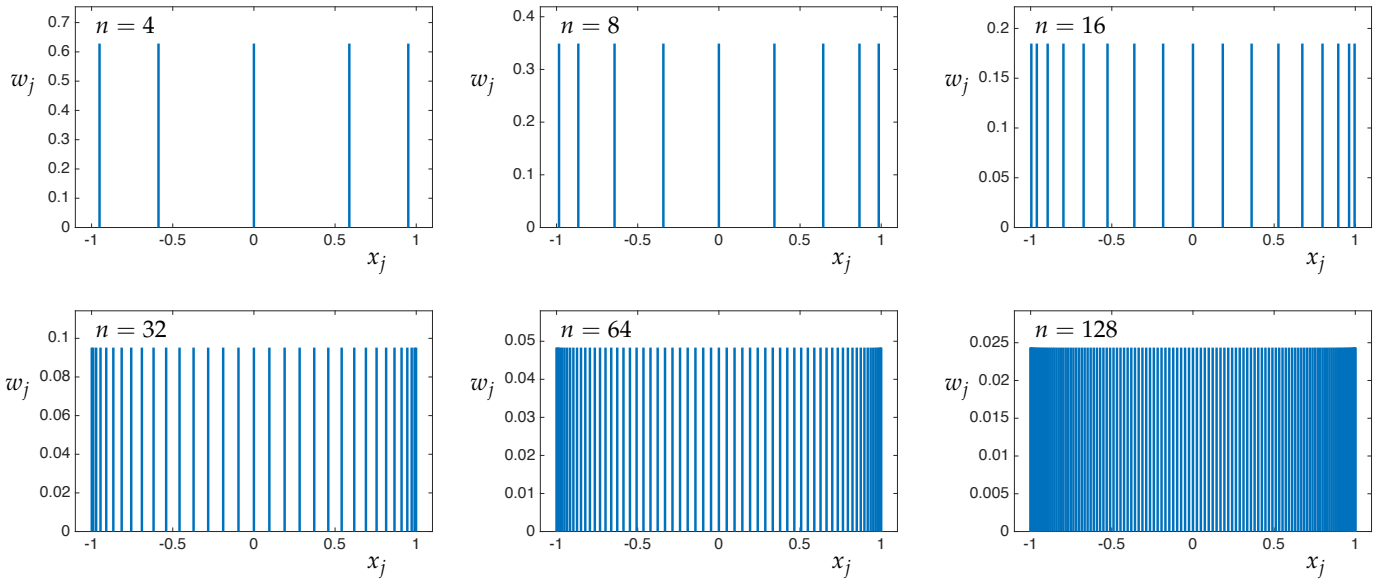

Figure 3.10: Nodes and weights of Gauss–Chebyshev quadrature, for various values of n. In each case, the location of the vertical line indicates x_j, while the height of the line shows w_j. The nodes, roots of Chebyshev polynomials, cluster toward the end of the intervals; the weights are the same for all the nodes, $w_j = \pi/(n+1)$.

The Chebyshev weight function $w(x) = 1/(1-x^2)$ blows up at ± 1, so if the integrand f does not balance this growth, adaptive Newton–Cotes rules will likely have to place many interpolation nodes near these singularities to achieve decent accuracy, while Gauss–Chebyshev quadrature has no problems. Moreover, in this important case, the nodes and weights are trivial to compute, thus allaying the need to solve the eigenvalue problem.

It is worth pointing out that Gauss–Chebyshev quadrature is quite different than Clenshaw–Curtis quadrature. Though both use Chebyshev points as interpolation nodes, only Gauss–Chebyshev incorporates the weight function $w(x) = (1-x^2)^{-1/2}$ in the weights $\{w_j\}$. Thus Clenshaw–Curtis is more appropriately compared to Gauss–Legendre quadrature. Since the Clenshaw–Curtis method is not a Gaussian quadrature formula, it will generally be exact only for all $p \in \mathcal{P}_n$, rather than all $p \in \mathcal{P}_{2n+1}$.

Example 3.4. Gauss–Laguerre quadrature
The Laguerre polynomials form a set of orthogonal polynomials over $(0, \infty)$ with the weight function $w(x) = \mathrm{e}^{-x}$. The accompanying quadrature rule approximates integrals of the form

$$\int_0^\infty f(x)\mathrm{e}^{-x}\,\mathrm{d}x.$$

The recurrence (3.4) uses the coefficients

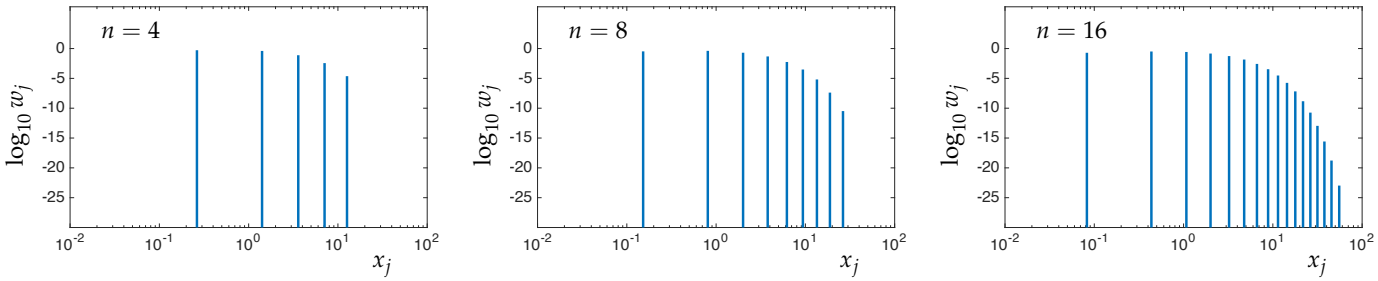

Figure 3.11: Nodes and weights of Gauss–Laguerre quadrature, for various values of n. In each case, the location of the vertical line indicates x_j, while the height of the line shows $\log_{10} w_j$. Note that the horizontal axis is scaled logarithmically. As n increases the quadrature rule includes larger and larger nodes to account for the infinite domain of integration; however, the weights are exceptionally small for the larger nodes. For example for $n = 16$, $w_{16} \approx 10^{-23}$.

$$\begin{aligned} \alpha_k &= 2k+1, \qquad && k = 0, 1, \ldots; \\ \beta_0 &= 1; \\ \beta_k &= k^2, && k = 1, 2, 3, \ldots. \end{aligned}$$

Figure 3.11 shows the nodes and weights for several values of n. Since the domain of integration $(0, \infty)$ is infinite, the quadrature nodes x_j get larger and larger. As the nodes get larger, the corresponding weights decay rapidly. When $w_j < 10^{-15}$, it becomes difficult to reliably compute the weights by solving the Jacobi matrix eigenvalue problem. To get the small weights given here, we have used Chebfun's `lagpts` routine routine, which uses a more efficient algorithm of Glaser, Liu, and Rokhlin (2007).

Example 3.5. Gauss–Hermite quadrature
The Hermite polynomials are orthogonal polynomials over $(-\infty, \infty)$ with the weight function $w(x) = \mathrm{e}^{-x^2}$. This quadrature rule approximates integrals of the form

$$\int_{-\infty}^{\infty} f(x) \mathrm{e}^{-x^2} \, \mathrm{d}x.$$

The Hermite polynomials can be generated using the recurrence (3.4) with coefficients

$$\begin{aligned} \alpha_k &= 0, \qquad && k = 0, 1, \ldots; \\ \beta_0 &= \sqrt{\pi}; \\ \beta_k &= k/2, && k = 1, 2, 3, \ldots. \end{aligned}$$

Figure 3.12 shows nodes and weights for various values of n. Though the interval of integration is infinite, the nodes do not grow as rapidly as for Gauss–Laguerre quadrature, since the Hermite weight $w(x) = \mathrm{e}^{-x^2}$ decays more rapidly than the Laguerre weight $w(x) = \mathrm{e}^{-x}$. (Again, the nodes and weights in the figure were computed with Chebfun's implementation of the Glaser, Liu, and Rokhlin algorithm.)

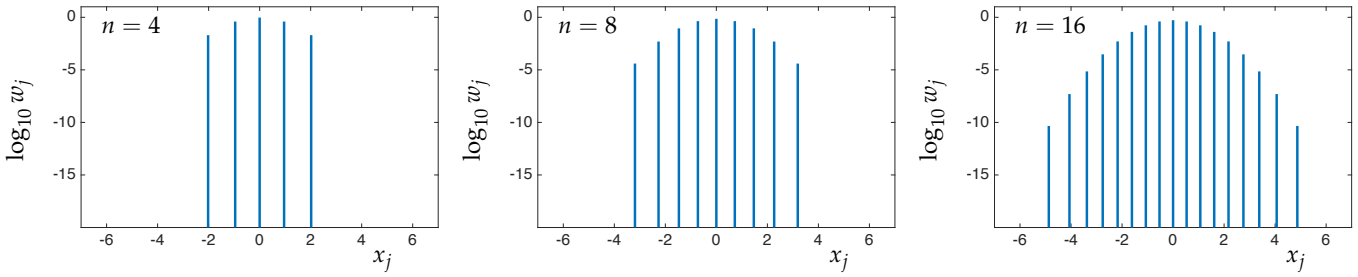

Figure 3.12: Nodes and weights of Gauss–Hermite quadrature, for various values of n. In each case, the location of the vertical line indicates x_j, while the height of the line shows $\log_{10} w_j$. Contrast this plot to Figure 3.11. Though the domain of integration is infinite in both cases, the weight function here, e^{-x^2}, decays much more rapidly than e^{-x}, explaining why the largest nodes are smaller than seen in Figure 3.11 for Gauss–Laguerre quadrature.

3.5.2 *Changing variables to transform domains*

One notable drawback of Gaussian quadrature is the need to precompute (or look up) the requisite weights and nodes. If one has a quadrature rule for the interval $[c, d]$, and wishes to adapt it to the interval $[a, b]$, there is a simple change of variables procedure to eliminate the need to recompute the nodes and weights from scratch. Let τ be a linear transformation taking $[c, d]$ to $[a, b]$,

$$\tau(x) = a + \left(\frac{b-a}{d-c}\right)(x-c)$$

with inverse $\tau^{-1} : [a, b] \to [c, d]$,

$$\tau^{-1}(y) = c + \left(\frac{d-c}{b-a}\right)(y-a).$$

Then we have

$$\begin{aligned}\int_a^b f(x)w(x)\,\mathrm{d}x &= \int_{\tau^{-1}(a)}^{\tau^{-1}(b)} f(\tau(x))w(\tau(x))\tau'(x)\,\mathrm{d}x \\ &= \left(\frac{b-a}{d-c}\right)\int_c^d f(\tau(x))w(\tau(x))\,\mathrm{d}x.\end{aligned}$$

The quadrature rule for $[a, b]$ takes the form

$$\widehat{I}(f) = \sum_{j=0}^{n} \widehat{w}_j f(\widehat{x}_j),$$

for

$$\widehat{w}_j = \left(\frac{b-a}{d-c}\right) w_j, \qquad \widehat{x}_j = \tau^{-1}(x_j),$$

where $\{x_j\}_{j=0}^n$ and $\{w_j\}_{j=0}^n$ are the nodes and weights for the quadrature rule on $[c, d]$.

Be sure to note how this change of variables alters the weight function. The transformed rule will now have a weight function

$$w(\tau(x)) = w(a + (b-a)(x-c)/(d-c)),$$

not simply $w(x)$. To make this concrete, consider Gauss–Chebyshev quadrature, which uses the weight function $w(x) = (1-x^2)^{-1/2}$ on

$[-1, 1]$. If one wishes to integrate, for example, $\int_0^1 x(1-x^2)^{-1/2}\,\mathrm{d}x$, it *is not* sufficient just to use the change of variables formula described here. To compute the desired integral, one would have to adjust the nodes and weights to accommodate $w(x) = (1-x^2)^{-1/2}$ on $[0, 1]$.

Composite rules Employing this change of variables technique, it is simple to devise a method for decomposing the interval of integration into smaller regions, over which Gauss quadrature rules can be applied. (The most straightforward application is to adjust the Gauss–Legendre quadrature rule, which avoids complications induced by the weight function, since $w(x) = 1$ in this case.) Such techniques can be used to develop Gaussian-based adaptive quadrature rules.

3.5.3 Gauss–Radau and Gauss-Lobatto quadrature

Some applications make it necessary or convenient to force one or both of the end points of the interval of integration to be among the quadrature points. Such methods are known as Gauss–Radau and Gauss–Lobatto quadrature rules, respectively; rules based on $n+1$ interpolation points exactly integrate all polynomials in $\mathcal{P}_{2n}$ or $\mathcal{P}_{2n-1}$: each quadrature node that we fix decreases the optimal order by one.