Lecture 26: More on Gaussian Quadrature [draft]

4.4.3. Examples of Gaussian Quadrature.

Gauss–Legendre quadrature. The best known Gaussian quadrature rule integrates functions over the interval $[-1, 1]$ with the trivial weight function $w(x) = 1$. As we saw in Lecture 19, the orthogonal polynomials for this interval and weight are called *Legendre polynomials*. To construct a Gaussian quadrature rule with $n + 1$ points, we must determine the roots of the degree-$(n + 1)$ Legendre polynomial, then find the associated weights.

First, consider the case of $n = 1$. The quadratic Legendre polynomial is

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

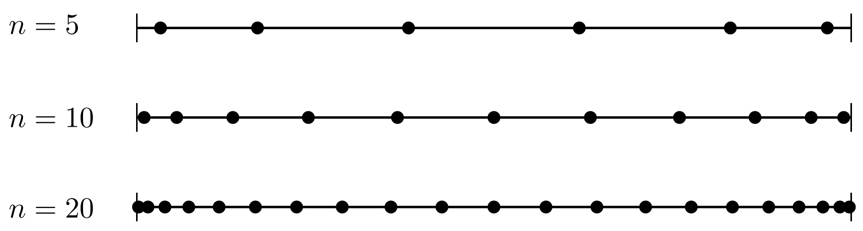
and from this polynomial one can derive the 2-point quadrature rule that is exact for cubic polynomials, with roots $\pm 1/\sqrt{3}$. This agrees with the special 2-point rule derived in Section 4.4.1. The values for the weights follow simply, $w_0 = w_1 = 1$, giving the 2-point Gauss–Legendre rule

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

For Gauss–Legendre quadrature rules based on larger numbers of points, there are various ways to compute the nodes and weights. Traditionally, one would consult a book of mathematical tables, such as the venerable *Handbook of Mathematical Functions*, edited by Milton Abramowitz and Irene Stegun for the National Bureau of Standards in the early 1960s. Now, one can look up these quadrature nodes and weights on web sites, or determine them to high precision using a symbolic mathematics package such as Mathematica. Most effective of all, one can compute these nodes and weights by solving a symmetric tridiagonal eigenvalue problem. See Trefethen and Bau, Lecture 37, for details. When $n = 5$, we obtain (from Mathematica)

j	nodes, x_j	weights, w_j
0	-0.9324695142031520	0.1713244923791703
1	-0.6612093864662645	0.3607615730481386
2	-0.2386191860831969	0.4679139345726910
3	0.2386191860831969	0.4679139345726910
4	0.6612093864662645	0.3607615730481386
5	0.9324695142031520	0.1713244923791703

Notice that the Gauss–Legendre nodes by no means uniformly distributed: like Chebyshev points for optimal interpolation, Legendre points for optimal quadrature cluster near the ends, as seen below (computed with Trefethen's `gauss.m` from *Spectral Methods in MATLAB*).



Given these values, we can implement the associated Gauss-Legendre quadrature rule with the following MATLAB code. (This includes a change of variables technique, described in Section 4.4.4 below, to allow integration over general intervals $[a, b]$, rather than $[-1, 1]$.)

```
  function intf = guasslengendre(f,a,b)
% Approximate the integral of f from a to b using a 6-point Gauss-Legendre rule.
% f is either the name of a function file, or an inline function.

 nodes   = [-0.9324695142031520; -0.6612093864662645; -0.2386191860831969;
             0.2386191860831969;  0.6612093864662645;  0.9324695142031520];
 weights = [ 0.1713244923791703;  0.3607615730481386;  0.4679139345726910;
             0.4679139345726910;  0.3607615730481386;  0.1713244923791703];

% change of variables from [-1,1] to [a,b]
 ab_nodes   = a + (b-a)*(nodes+1)/2;
 ab_weights = weights*(b-a)/2;

% apply Guass-Legendre rule
 intf = sum(ab_weights.*feval(f,ab_nodes));     % requires f to work for vectors
```

We can test this code on $\int_0^1 x^j \, \mathrm{d}x = 1/(j+1)$. Since this quadrature rule uses $n+1 = 6$ points, it should be exact for polynomials of degree $2n+1 = 11$. Indeed, that is what we see (up to rounding error) in the table below: There are negligible errors for $j = 1, \ldots, 11$, but at $j = 12$, suddenly some significant error appears. Even this discrepancy is fairly small, remarkable given that we are evaluating the integrand at only six points.

```
    j      Gauss-Legendre            error
----------------------------------------------------
    1   0.49999999999999994   0.00000000000000006
    2   0.33333333333333326   0.00000000000000006
    3   0.24999999999999994   0.00000000000000006
    4   0.20000000000000001   0.00000000000000000
    5   0.16666666666666663   0.00000000000000003
    6   0.14285714285714285   0.00000000000000000
    7   0.12499999999999999   0.00000000000000001
    8   0.11111111111111110   0.00000000000000000
    9   0.09999999999999999   0.00000000000000001
   10   0.09090909090909090   0.00000000000000001
   11   0.08333333333333331   0.00000000000000001
   12   0.07692298682558422   0.00000009009749270
   13   0.07142798579486889   0.00000058563370253
   14   0.06666454418815693   0.00000212247850974
   15   0.06249433000097107   0.00000566999902893
```

Gauss–Chebyshev quadrature. Another popular class of Gaussian quadrature rules use as their nodes the roots of the Chebyshev polynomials. Recall that the degree-k Chebyshev polynomial is defined as

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

These are orthogonal polynomials 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

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

One can determine the associated weights to be

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

for all $j = 0, \ldots n$. (See Süli and Mayers, Problem 10.4 for a sketch of a proof.)

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

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

one *does not* include $1/\sqrt{1-x^2}$: the weight function is absorbed into the quadrature weights $\{w_j\}$.

Notice that the Chebyshev weight function blows up at ± 1, so if the integrand f doesn't 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 consult numerical tables or employ other fancy tools.

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 $p \in \mathcal{P}_{2n+1}$.

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

$$\int_{0}^{\infty} f(x)\mathrm{e}^{-x}\,\mathrm{d}x.$$

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

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

4.4.4. Variations on the theme.

Change of variables. One notable drawback of Gaussian quadrature is the need to pre-compute (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 $[a, b]$,

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

Then we have

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

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

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

for

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

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

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. See the above MATLAB code for an implementation.) Such techniques can be used to develop Gaussian-based adaptive quadrature rules.

Gauss–Radau and Gauss-Lobatto quadrature. In applications, it is sometimes 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.

[†]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 $[0, 1]$.