8. Gaussian quadrature

Let f be a continuous function on $[-1,1]$. Consider the general quadrature rule with N nodes[10]:

$$\int_{-1}^{1} f(x)\,dx = \sum_{n=1}^{N} c_n\, f(x_n) + R_N(f). \tag{35}$$

Set $L(f) = \int_{-1}^{1} f(x)\,dx$ and $Q_N(f) = \sum_{n=1}^{N} c_n\, f(x_n)$. Now consider the $2\,N$ equations below in which we regard the nodes x_n and the weights c_n as unknowns:

$$\begin{aligned}
Q(1) &= \sum_{n=1}^{N} c_n = L(1) = 2 \\
Q(x) &= \sum_{n=1}^{N} c_n\, x_n = L(x) = 0 \\
Q(x^2) &= \sum_{n=1}^{N} c_n\, x_n^2 = L(x^2) = \frac{2}{3} \\
&\cdots \\
Q(x^{2\,N-2}) &= \sum_{n=1}^{N} c_n\, x_n^{2\,N-2} = L(x^{2\,N-2}) \\
Q(x^{2\,N-1}) &= \sum_{n=1}^{N} c_n\, x_n^{2\,N-1} = L(x^{2\,N-1})
\end{aligned}$$

If these equations can be solved, the resulting quadrature rule will have order of accuracy $2\,N-1$. This is the highest order of accuracy for a fixed number of nodes.

Of course, at this point it is far from obvious that the equations $Q(x^k) = L(x^k)$, $k = 0, \ldots, 2\,N-1$ can be solved. Nor is it obvious that the nodes thus obtained all lie within $[-1,1]$. In the next section we will show that the solution exists for $N = 2, 3$. We will then show that the solution—Gaussian quadrature—exists for all N and is unique.

[10]There exist more general quadrature rules than Equation (35). However, they involve derivatives of the function which are usually not available.

8.1. **Gaussian quadratures with 2 and 3 nodes.** For $N = 2$ the Gaussian quadrature is defined by the following four equations:

$$\begin{aligned} c_1 + c_2 &= 2 \\ c_1\, x_1 + c_2\, x_2 &= 0 \\ c_1\, x_1^2 + c_2\, x_2^2 &= \frac{2}{3} \\ c_1\, x_1^3 + c_2\, x_2^3 &= 0 \end{aligned}$$

It is not hard to solve this system by hand. Indeed, by symmetry, $c_1 = c_2$ and $x_1 = -x_2$. With this assumption two of the equations (for odd monomials) are satisfied while the remaining two equations read $2\, c_1 = 2$ and $2\, c_1\, x_1^2 = \frac{2}{3}$. This leads to

$$Q_2(f) = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right), \tag{36}$$

which is a quadrature of order 3, by construction.

When $N = 3$ the defining system of equations becomes:

$$\begin{aligned} c_1 + c_2 + c_3 &= 2 \\ c_1\, x_1 + c_2\, x_2 + c_3\, x_3 &= 0 \\ c_1\, x_1^2 + c_2\, x_2^2 + c_3\, x_3^2 &= \frac{2}{3} \\ c_1\, x_1^3 + c_2\, x_2^3 + c_3\, x_3^3 &= 0 \\ c_1\, x_1^4 + c_2\, x_2^4 + c_3\, x_3^4 &= \frac{2}{5} \\ c_1\, x_1^5 + c_2\, x_2^5 + c_3\, x_3^5 &= 0 \end{aligned}$$

Here symmetry suggests that we set $x_2 = 0$, $x_1 = -x_3$, and $c_1 = c_3$. Then three of the equations (for odd monomials) are satisfied, and the remaining three equations can be written as:

$$\begin{aligned} 2\, c_1 + c_2 &= 2 \\ 2\, c_1\, x_1^2 &= \frac{2}{3} \\ 2\, c_1\, x_1^4 &= \frac{2}{5} \end{aligned}$$

Dividing the third equation by the second leads to $x_1^2 = \frac{3}{5}$. If we number the nodes from left to right, then $x_1 = -\sqrt{\frac{3}{5}}$. Now, from the second equation $c_1 = \frac{5}{9}$; consequently, $c_2 = \frac{8}{9}$. We conclude that:

$$Q_3(f) = \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right). \tag{37}$$

By construction, Q_3 has order 5.

8.2. **Application of Newton's method.** For large number of nodes, finding Gaussian quadrature by hand quickly becomes unmanageable. As one alternative to manual computations, we can consider solving the equations numerically. Let

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \\ c_1 \\ \vdots \\ c_N \end{bmatrix}, \quad F(\mathbf{X}) = \begin{bmatrix} c_1 + \ldots + c_N - 2 \\ c_1\, x_1 + \ldots + c_N\, x_N \\ c_1\, x_1^2 + \ldots + c_N\, x_N^2 - \frac{2}{3} \\ \vdots \\ Q_N(x^{2N-2}) - L(x^{2N-2}) \\ Q_N(x^{2N-1}) - L(x^{2N-1}) \end{bmatrix}$$

Notice that finding Gaussian nodes and weights is equivalent to solving $F(\mathbf{X}) = \mathbf{0}$.

Recall that our choice method for solving *scalar* equations $f(x) = 0$ is Newton's method. It stands to reason that there should be an extension of Newton's method to vector-valued functions. To see what it is, let JF denote the *Jacobian* of F:

$$JF = \left[\frac{\partial F_i}{\partial X_j}\right]_{i,j=1}^{2N}$$

Note that the Jacobian matrix is formed by differentiating the i-th component of the (vector-valued) function F with respect to the j-th component of the (vector) variable X. For instance, for $N = 2$ the Jacobian of F is the following four-by-four matrix:

$$JF = \begin{bmatrix} 0 & 0 & 1 & 1 \\ c_1 & c_2 & x_1 & x_2 \\ 2\,c_1\,x_1 & 2\,c_2\,x_2 & x_1^2 & x_2^2 \\ 3\,c_1\,x_1^2 & 3\,c_2\,x_2^2 & x_1^3 & x_1^3 \end{bmatrix}$$

The Jacobian matrix is the generalization of the derivative; in fact, it is *the* derivative of a vector-valued function which suggests the following familiar construction. Let $\mathbf{X}_*$ be the (vector) root of F and let $\mathbf{X}_n$ be the n-th approximation to $\mathbf{X}_*$. The tangent space approximation of F near $\mathbf{X}_n$ is given by:

$$F(\mathbf{X}) \approx F(\mathbf{X}_n) + JF(\mathbf{X}_n)\,(\mathbf{X} - \mathbf{X}_n).$$

Notice the analogy with the equation of the tangent line

$$f(x) \approx f(x_n) + f'(x_n)\,(x - x_n)$$

and the equation of the tangent plane

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \approx f\left(\begin{bmatrix} x_n \\ y_n \end{bmatrix}\right) + \nabla f\left(\begin{bmatrix} x_n \\ y_n \end{bmatrix}\right) \cdot \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_n \\ y_n \end{bmatrix}\right).$$

Now, since $F(\mathbf{X}_*) = \mathbf{0}$, we have: $F(\mathbf{X}_n) + JF(\mathbf{X}_n)\,(\mathbf{X}_* - \mathbf{X}_n) \approx \mathbf{0}$. Therefore $\mathbf{X}_* \approx \mathbf{X}_n - JF^{-1}(\mathbf{X}_n)\,F(\mathbf{X}_n)$. Reasoning as in the scalar case, we are led to the following iterative root-finding scheme

$$\mathbf{X}_{n+1} = \mathbf{X}_n - JF^{-1}(\mathbf{X}_n)\,F(\mathbf{X}_n), \tag{38}$$

where the superscript in JF^{-1} stands for the matrix inverse. Equation (38) is Newton's method for systems of equations.

It is easy to implement (38) in Maple where the Jacobian matrix can be computed symbolically. However, some care needs to be exercised with the starting values. Recall that scalar Newton's method is guaranteed to converge only if the starting guess is "close enough" to the root. The same holds for systems of equations. For Gaussian quadrature, we can guess the nodes to be equispaced in $[-1, 1]$ and to have the same unit weight. This, of course, is far from truth but it works. For instance, for $N = 3$, five iterations of Newton's method (38) with the starting guess

$$\mathbf{X}_0 = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

leads to the following approximation:

$$\mathbf{X}_5 = \begin{bmatrix} -0.7745966691 \\ 2.290086622 \times 10^{-10} \\ 0.7745966691 \\ 0.5555555558 \\ 0.8888888884 \\ 0.5555555557 \end{bmatrix}$$

Comparison with our previous result shows that these are the nodes and weights of Q_3 with 8 digits of accuracy.

For $N = 4$ (with similarly constructed starting guess) Newton's method quickly converges to

$$\mathbf{X}_* \approx \begin{bmatrix} -0.8611363111 \\ -0.3399810426 \\ 0.3399810428 \\ 0.8611363111 \\ 0.3478548462 \\ 0.6521451543 \\ 0.6521451538 \\ 0.3478548462 \end{bmatrix}$$

And things work well for $N = 5, 6$. Unfortunately, for $N = 7$ the method diverges and tweaking the starting guess does not improve the situation.

The reason why Newton's method diverges for $N > 6$ lies in the Jacobian matrix. Its determinant, which can be found in closed form, quickly approaches zero with $N \to \infty$. While JF is theoretically invertible for all N, in practice the inversion is severely unstable. This leads to divergence for all sensible starting values.

At this point we could consider other numerical schemes for solving quadrature equations—ones that do not require inversion of the Jacobian. However, it is better to change tactics entirely, as we do in the next section. Here we conclude with a remark that, despite occasional failures, Newton's method is still the default root-finding scheme in many practical situations.

8.3. **Legendre polynomials.** In Maple it is possible to solve systems of equations symbolically with the `solve` command. For instance, feeding `solve` the equations defining Q_2 produces the following result:

$$\left\{x_2 = -\mathit{RootOf}(-1 + 3\,\_Z^2), c_1 = 1, x_1 = \mathit{RootOf}(-1 + 3\,\_Z^2), c_2 = 1\right\}$$

Unlike a human solver, `solve` does not make any simplifying assumptions based on symmetry. This leads to very long computations for $N = 3, 4$ and failure for $N > 4$. While `solve` is incapable of dealing with the complexity of $2\,N$ polynomial equations for large N, it does not give up without giving us an important clue. Notice that for small N, the routine returns the nodes expressed as roots of polynomials.

Inspired by the output of `solve`, let us define the *monic nodal polynomial* for Q_N by:

$$p_N = (x - x_1)\,\ldots\,(x - x_N) = \prod_{n=1}^{N}(x - x_n).$$

The roots of p_N are the nodes of Q_N, hence the moniker *nodal*; the adjective *monic* refers to the fact that the highest coefficient in p_N (the one in front of x^N) is 1.

Clearly, the knowledge of p_N is equivalent to knowing the nodes x_n. However, once we know the nodes, we can find the weights c_n by integrating the interpolating polynomial. Therefore p_N completely defines the Gaussian quadrature Q_n.

Introducing nodal polynomials is worthwhile just by virtue of the fact that it is conceptually easier to search for one unknown—the nodal polynomial p_N—rather than N unknowns—the nodes. However that is just one of the reasons. It turns out that nodal polynomials have a very important property called *orthogonality.* To see it, let p_N be the nodal polynomial (for Q_N) and let q be *any* polynomial of degree less than N. Since Q_N has order of accuracy $2\,N-1$ and the degree of $p_N\,q$ is at most $2\,N-1$, we must have:

$$L(p_N\,q) = \int_{-1}^{1} p_N(x)\,q(x)\,dx = Q_N(p_N\,q).$$

On the other hand, since p_N vanishes at the nodes:

$$Q_N(p_N\,q) = \sum_{n=1}^{N} p_N(x_n)\,q(x_n) = 0.$$

We conclude that

$$\int_{-1}^{1} p_N(x)\,q(x)\,dx = 0$$

for all q with degree lower that N. If we regard functions on $[-1,1]$ as elements of a functional *vector space* and the integral

$$\int_{-1}^{1} f(x)\,g(x)\,dx$$

is a sort of functional dot product, we can say that p_N is *orthogonal* to all lower degree polynomials. We will pursue this line of reasoning in the next handout, after an appropriate review of linear algebra. In the meantime, notice that orthogonality of p_N to all lower degree polynomials implies orthogonality to monomials x^k for $k=0,\ldots,N-1$. This suggests the following (symbolic) computational scheme:

(1) Given a positive integer N, set

$$p_N(x) = a_0 + a_1\,x + \ldots + a_{N-1}\,x^{N-1} + x_N.$$

(2) Form linear equations for the coefficients a_n by setting

$$\int_{-1}^{1} p_N(x)\, x^k\, dx = 0$$

for $k = 0, \ldots, N-1$.

(3) Solve the linear equations symbolically (using `solve`) or numerically.

The following code implements this strategy in Maple:

```
> Q := (f,g) -> int(f(x)*g(x),x=-1..1);
> N := 2;
> p := unapply(add(a[n]*x^n,n=0..N),x);
> a[N] := 1;
> for n from 1 to N do
>     Eq[n] := Q(p,x->x^(n-1));
> end do;
> assign(solve({seq(Eq[n],n=1..N)}));
> p(x);
```

The output of the last command is

$$x^2 - \frac{1}{3},$$

which we recognize as the (monic) nodal polynomial for Q_2. Running the code with $N = 3$ gives

$$p_3 = x^3 - \frac{3}{5}\, x^2.$$

For $N = 10$ the output, produced almost instantaneously, is

$$p_{10} = x^{10} - \frac{45}{19}\, x^8 + \frac{630}{323}\, x^6 - \frac{210}{323}\, x^4 + \frac{315}{4199}\, x^2 - \frac{63}{46189}.$$

If we now want the nodes, we can use the `RootOf` command as follows.

```
> nodes := evalf(allvalues(RootOf(p(x))));
> sort([nodes]);
```

This results in the nodes of Q_N computed with machine precision. For instance, the nodes of Q_{10} are:

$-.9739065285, -.8650633667, -.6794095683, -.4333953942, -.1488743390,$

$.1488743390, .4333953942, .6794095683, .8650633667, .9739065285$

The nodal polynomials for Gaussian quadrature are scalar multiples of Legendre polynomials. For this reason the Gaussian quadrature for $\int_{-1}^{1} f(x)\, dx$ is often called Gauss-Legendre quadrature.

Legendre polynomials form one very important family of *orthogonal polynomials*. They arise not just in quadrature but also as solutions of various partial differential equations of mathematical physics and in other contexts. There exist much better ways both for computing Legendre polynomials and for finding their roots. We will explore these topics in the next handout.

Exercises

(1) Consider the following quadrature rule:

$$\int_{-1}^{1} f(x)\,(1-x^2)\,dx = \sum_{n=1}^{N} c_n\, f(x_n) + R_N(f).$$

Find the nodes and weights that maximize the degree of accuracy for $N = 2, 3, 4$: you can use Newton's method, for instance. Find the (exact) monic nodal polynomials for $N = 2, 3, 4$.

(2) Use Newton's method to solve the following system of equations:

$$x^x + y^y = 1000 \quad x^y + y^x = 100.$$

(3) Let Q denote the Gauss-Legendre quadrature with two nodes

$$Q(f) = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

and let $\widetilde{Q}_N$ denote the *extension* of Q obtained by adding N additional nodes within $[-1, 1]$:

$$\widetilde{Q}_N(f) = a_1\, f\left(-\frac{1}{\sqrt{3}}\right) + a_2\, f\left(\frac{1}{\sqrt{3}}\right) + \sum_{n=1}^{N} c_n\, f(x_n).$$

The parameters in $\widetilde{Q}_N$ are chosen so that its degree of accuracy is maximal; these parameters involve N new nodes $\{x_n\}$, N corresponding weights $\{c_n\}$, and also two new weights a_1 and a_2 for the nodes of Q.

(a) Find $\widetilde{Q}_1$. Bear in mind that the added nodes must be within $[-1, 1]$.

(b) Find the degree of accuracy of $\widetilde{Q}_1$ and the error term.

(c) Suppose we know $Q(f)$ and $\widetilde{Q}_1(f)$. Derive an approximation for $\int_{-1}^{1} f(x)\,dx$ and an error estimate.

(d) Repeat the above with $\widetilde{Q}_2$.