Numerical integration

Numerical integration constitutes a broad family of algorithms to compute a numerical approximation to a definite (Riemann) integral. The integral is generally approximated by a weighted sum of function values within the domain of integration,

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \,, \tag{1}$$

often called *quadrature* (*cubature* for multidimensional integrals) or *rule*. The abscissas, x_i, also called *nodes*, and the weights, w_i, of a quadrature are usually optimized — using one of a large number of different strategies — to suit a particular class of integration problems.

The best quadrature algorithm to use for a given integral depends on several factors, in particular on the integrand. Different classes of integrands — smooth, singular, oscillatory, time-consuming, etc. — generally require different quadratures for the most effective calculation.

In the following we shall consider several of the most popular numerical integration algorithms.

Rectangle and trapezium rules

In mathematics, the *Reimann integral* is usually defined in terms of *Riemann sums*. If the integration interval $[a, b]$ is partitioned into n subintervals,

$$a = t_0 < t_1 < t_2 < \cdots < t_n = b \,. \tag{2}$$

the Riemann sum is defined as

$$\sum_{i=1}^{n} f(x_i)\Delta x_i \,, \tag{3}$$

where $t_{i-1} \le x_i \le t_i$ and $\Delta x_i = t_i - t_{i-1}$. Geometrically a Riemann sum can be interpreted as the area of a collection of adjucent rectangles with widths Δx_i and heights $f(x_i)$.

The Rieman integral is defined as the limit of a Riemann sum as the mesh — the length of the largest subinterval — of the partition approaches zero. Specifically, the number denoted as

$$\int_a^b f(x)dx \tag{4}$$

is called the Riemann integral, if for any $\epsilon > 0$ there exists $\delta > 0$ such that for any partition (2) with $\max \Delta x_i < \delta$ we have

$$\left| \sum_{i=1}^{n} f(x_i)\Delta x_i - \int_a^b f(x)dx \right| < \epsilon \,. \tag{5}$$

A definite integral can be interpreted as the net signed area bounded by the graph of the integrand.

Now, the n-point *rectangle quadrature* is simply the Riemann sum (3),

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)\Delta x_i \,. \tag{6}$$

where the node x_i is often (but not always) taken at the middle of the corresponding subinterval, $x_i = t_{i-1} + \frac{1}{2}\Delta x_i$, and the subintervals are often (but not always) chosen equal, $\Delta x_i = (b-a)/n$. Geometrically the n-point rectangle rule is an approximation to the integral given by the area of a collection of n ajucent equal rectangles whose heights are determined by the values of the function (at the middle of the rectangle).

An n-point *trapezium rule* uses instead a collection of trapezia fitted under the graph,

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(t_{i-1}) + f(t_i)}{2}\Delta x_i \,. \tag{7}$$

Importantly, the trapezium rule is an average of two Riemann sums,

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{n} f(t_{i-1})\Delta x_i + \frac{1}{2}\sum_{i=1}^{n} f(t_i)\Delta x_i \ . \tag{8}$$

Rectangle and trapezium quadratures have the important feature of closely following the very mathematical definition of the integral as the limit of the Riemann sums. This means that — disregarding the round-off errors — these two rules cannot fail if the integral exist. Indeed the existence of the integral is the only necessary condition for the convergence.

For certain partitions of the interval the rectangle and trapezium rules coincide. For example, for the nodes

$$x_i = a + (b-a)\frac{i-\frac{1}{2}}{n} \ , \quad i = 1, \ldots, n \tag{9}$$

both rules give the same quadrature with equal weights, $w_i = (b-a)/n$,

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=1}^{n} f\left(a + (b-a)\frac{i-\frac{1}{2}}{n}\right) \ . \tag{10}$$

Rectangle and trapezium quadratures are rarely used on their own — because of the slow convergence — but they often serve as a basis for more advanced quadratures, for example adaptive quadratures and variable transformation quadratures considered below.

Quadratures with regularly spaced abscissas

A quadrature with n predefined abscissas has n free parameters – the weights w_i. A set of n parameters can generally be tuned to satisfy n conditions. The archetypal set of condition in quadratures is that the quadrature integrates exactly a set of n functions,

$$\{\phi_1(x), \ldots, \phi_n(x)\} \ . \tag{11}$$

This leads to a set of n equations,

$$\sum_{i=1}^{n} w_i \phi_k(x_i) = I_k \ , \quad k = 1, \ldots, n \ , \tag{12}$$

where the integrals are assumed to be known. The equations (12) are linear in w_i and can be easily solved. Since integration is a linear operation, the quadrature will then also integrate exactly any linear combination of the functions (11).

Suppose the integrand can be well represented by the first few terms of its Taylor series,

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k \ , \tag{13}$$

where $f^{(k)}$ is the k-th derivative of the integrand. This is often the case for analytic — that is, infinitely differentiable — functions. For such integrands one can obviously choose polynomials

$$\{1, x, x^2, \ldots, x^{n-1}\} \tag{14}$$

as the set of functions to be integrated exactly.

Classical quadratures

Quadratures with regularly spaced abscissas and polynomials as exactly integrable functions are often called *classical quadratures*. An n-point classical quadrature integrates exactly the first n terms of the function's Taylor expansion (13). The x^n order term will not be integrated exactly and will lead to an

Table 1: Maxima script to calculate analytically the weights of a quadrature with predefined abscissas in the interval $[0, 1]$.

```
n: 8; xs: makelist((i-1)/(n-1),i,1,n); /* list of nodes */
ws: makelist(concat(w,i),i,1,n);
ps: append([1],makelist(x^i,i,1,n-1)); /* polynomials */
fs: makelist(buildq([i:i,ps:ps],lambda([x],ps[i])),i,1,n);
integ01: lambda([f],integrate(f(x),x,0,1));
Is: maplist(integ01,fs); /* calculate the integrals */
eq: lambda([f],lreduce("+",maplist(f,xs)*ws));
eqs: maplist(eq,fs)-Is; /* build equations */
solve(eqs,ws); /* solve for the weights */
```

---

error of the quadrature. Thus the error E_n of the n-point classical quadrature is of the order of the integral of the x^n term in 13,

$$E_n \approx \int_a^b \frac{f^{(n)}(a)}{n!}(x-a)^n dx = \frac{f^{(n)}(a)}{(n+1)!}h^{n+1} \propto h^{n+1}\,, \tag{15}$$

where $h = b - a$ is the length of the integration interval. A quadrature with the error of the order h^{n+1} is often called a *degree-n* quadrature.

If the integrand is smooth enough and the length h is small enough a classical quadrature can provide a good approximation. However, for large n the weights of classical quadratures tend to have alternating signs, which leads to large round-off errors, which in turn negates the potentially higher accuracy of the quadrature. Again, if the integrand violates the assumption of Taylor expansion—for example by having an integrable singularity inside the integration interval—the higher order quadratures may perform poorly.

An interested reader can easily generate classical quadratures of any degree n using the Maxima script in Table (1).

Newton-Cotes quadratures

Newton-Cotes quadratures are classical quadratures with equally spaced abscissas,

$$x_i = a + (b-a)\frac{i-1}{n-1}\,, \quad i = 1, \ldots, n\,. \tag{16}$$

Newton-Cotes quadratures are mostly of historical interest nowadays. Alternative methods—such as quadratures with optimized abscissas or variable transformation quadratures—are more stable and accurate and are normally preferred to Newton-Cotes.

An interested reader can generate Newton-Cotes quadratures of any degree n using the Maxima script in Table (1).

Quadratures with optimized abscissas

In quadratures with optimized abscissas not only the weights w_i but also the abscissas x_i are chosen optimally. The number of free parameters is thus $2N$ and one can choose a set of $2N$ functions,

$$\{\phi_1(x), \ldots, \phi_{2N}(x)\}\,, \tag{17}$$

to be integrated exactly. This gives a system of $2N$ equations, linear in w_i and non-linear in x_i,

$$\sum_{i=1}^{N} w_i \phi_k(x_i) = I_k\,, k = 1, \ldots, 2N, \tag{18}$$

where

$$I_k = \int_a^b \phi_k(x)dx. \tag{19}$$

The weights and abscissas can be determined by solving this system of equations.

Here is, for example, a two-point Gauss-Legendre quadrature rule

$$\int_{-1}^{1} f(x)dx \approx f\left(-\sqrt{\tfrac{1}{3}}\right) + f\left(+\sqrt{\tfrac{1}{3}}\right) . \tag{20}$$

Although quadratures with optimized abcissas are generally of much higher order — $2N-1$ compared to $N-1$ for non-optimal abscissas — the optimal points generally can not be reused at the next iteration in an adaptive algorithm.

Gauss quadratures

Gauss quadratures deal with a slightly more general form of integrals,

$$\int_a^b \omega(x)f(x)dx , \tag{21}$$

where $\omega(x)$ is a positive weight function. For $\omega(x) = 1$ the problem is the same as considered above. Popular choices of the weight function include $\omega(x) = (1-x^2)^{\pm 1/2}$, $\exp(-x)$, $\exp(-x^2)$ and others. The idea is to represent the integrand as a product $\omega(x)f(x)$ such that all the difficulties go into the weight function $\omega(x)$ while the remaining factor $f(x)$ is smooth and well represented by polynomials.

Now, an N-point *Gauss quadrature* is a quadrature with optimized abcissas,

$$\int_a^b \omega(x)f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i) , \tag{22}$$

which integrates exactly — with the given weight $\omega(x)$ — a set of $2N$ polynomials of the orders $1, \ldots, 2N-1$.

Fundamental theorem There is a theorem stating that there exists a set of polynomials $p_n(x)$, orthogonal on the interval $[a, b]$ with the weight function $\omega(x)$,

$$\int_a^b \omega(x)p_n(x)p_k(x) \propto \delta_{nk} , \tag{23}$$

and that the optimal nodes for an N-point Gauss quadrature are the roots of the polynomial $p_N(x)$,

$$p_N(x_i) = 0 . \tag{24}$$

The idea behind the proof is in considering an integral

$$\int_a^b \omega(x)q(x)p_N(x)dx = 0 , \tag{25}$$

where $q(x)$ is an arbitrary polynomial of degree less than N. The quadrature should represent this integral exactly,

$$\sum_{i=1}^{N} q(x_i)p_N(x_i) = 0 . \tag{26}$$

Apparently this is only possible if x_i are the roots of p_N ■.

Calculation of nodes and weights A neat algorithm — usually refered to as Golub-Welsch [**?**] algorithm — for calculation of the nodes and weights of a Gauss quadrature is based on the symmetric form of the three-term reccurence relation for orthogonal polynomials,

$$xp_{n-1}(x) = \beta_n p_n(x) + \alpha_n p_{n-1}(x) + \beta_{n-1} p_{n-2}(x) \,, \tag{27}$$

where $p_{-1}(x) \doteq 0$, $p_1(x) \doteq 1$, and $n = 1, \ldots, N$. This reccurence relation can be written in the matrix form,

$$x\mathbf{p}(x) = J\mathbf{p}(x) + \beta_N p_N(x)\mathbf{e}_N \,, \tag{28}$$

where $\mathbf{p}(x) \doteq \{p_0(x), \ldots, p_{N-1}(x)\}^T$, $\mathbf{e}_N = \{0, \ldots, 0, 1\}^T$, and the tridiagonal matrix J — usually refered to as *Jacobi matrix* or *Jacobi operator* — is given as

$$J = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \beta_3 & \\ & & \ddots & \ddots & \\ & & & \beta_{N-1} & \alpha_N \end{pmatrix} . \tag{29}$$

Substituting the roots x_i of p_N — $\{x_i \mid p_N(x_i) = 0\}$ — into the matrix equation (28) leads to eigenvalue problem for the Jacobi matrix,

$$J\mathbf{p}(x_i) = x_i \mathbf{p}(x_i) \,. \tag{30}$$

Thus the nodes of an N-point Gauss quadrature — the roots of the polynomial p_N — are the eigenvalues of the Jacobi matrix J and can be calculated by a standard diagonalization[1] routine ■.

The weights can be obtained considering N integrals,

$$\int_a^b \omega(x) p_n(x) dx = \delta_{n0} \int_a^b \omega(x) dx \,, \; n = 0, \ldots, N-1 \,. \tag{31}$$

Applying our quadrature gives the matrix equation,

$$P\mathbf{w} = \mathbf{e}_1 \int_a^b \omega(x) dx \,, \tag{32}$$

where $\mathbf{w} \doteq \{w_1, \ldots, w_N\}^T$, $\mathbf{e}_1 = \{1, 0, \ldots, 0\}^T$, and

$$P \doteq \begin{pmatrix} p_0(x_1) & \ldots & p_0(x_N) \\ p_1(x_1) & \ldots & p_1(x_N) \\ \ldots & \ldots & \ldots \\ p_{N-1}(x_1) & \ldots & p_{N-1}(x_N) \end{pmatrix} . \tag{33}$$

Equation (32) is linear in w_i and can be solved directly. However, if diagonalization of the Jacobi matrix provided the normalized eigenvectors, the weigths can be readily obtained using the following method.

The matrix P apparently consists of non-normalized column eigenvectors of the matrix J. The eigenvectors are orthogonal and therefore $P^T P$ is a diagonal matrix with positive elements. Multiplying (51) by P^T and then by $(P^T P)^{-1}$ from the left gives

$$\mathbf{w} = (P^T P)^{-1} P^T \mathbf{e}_1 \int_a^b \omega(x) dx \,. \tag{34}$$

From $p_0(x) = 1$ it follows that $P^T \mathbf{e}_1 = \{1, \ldots, 1\}^T$ and therefore

$$w_i = \frac{1}{(P^T P)_{ii}} \int_a^b \omega(x) dx \,. \tag{35}$$

[1] A symmetric tridiagonal matrix can be diagonalized very effectively using the QR/RL algorithm.

Let the matrix V be the set of the normalized column eigenvectors of the matrix J. The matrix V is then connected with the matrix P through the normalization equation,

$$V = \sqrt{(P^T P)^{-1}} P \,. \tag{36}$$

Therefore, again taking into account that $p_0(x) = 1$, equation (51) can be written as

$$w_i = (V_{1i})^2 \int_a^b \omega(x) dx \;\blacksquare. \tag{37}$$

Example: Gauss-Legendre quadrature Gauss-Legendre quadrature deals with the weight $\omega(x) = 1$ on the interval $[-1, 1]$. The associated polynomials are Legendre polynomials $\mathcal{P}_n(x)$, hence the name. Their reccurence relation is usually given as

$$(2n-1)x\mathcal{P}_{n-1}(x) = n\mathcal{P}_n(x) + (n-1)\mathcal{P}_{n-2}(x) \,. \tag{38}$$

Rescaling the polynomials (preserving $p_0(x) = 1$) as

$$\sqrt{2n+1}\mathcal{P}_n(x) = p_n(x) \tag{39}$$

reduces this reccurence relation to the symmetric form (27),

$$xp_{n-1}(x) = \frac{1}{2}\frac{1}{\sqrt{1-(2n)^{-2}}}p_n(x) + \frac{1}{2}\frac{1}{\sqrt{1-(2(n-1))^{-2}}}p_{n-2}(x) \,. \tag{40}$$

Correspondingly the coefficients in the matrix J are

$$\alpha_n = 0 \,, \quad \{\beta_n = \frac{1}{2}\frac{1}{\sqrt{1-(2n)^{-2}}} \;\mid\; n = 1, \ldots, N-1\} \,. \tag{41}$$

The problem of finding the nodes and the weights of the N-point Gauss-Legendre quadrature is thus reduced to the eigenvalue problem for the Jacobi matrix with coefficients (41). The following Octave function

```
function Q = gauss_legendre(f,a,b,N)
beta = .5./sqrt(1-(2*(1:N-1)).^(-2)); % reccurence relation
J = diag(beta,1) + diag(beta,-1);     % Jacobi matrix
[V,D] = eig(J);                       % diagonalization of J
x = diag(D); [x,i] = sort(x);         % sorted nodes
w = V(1,i).^2*2;                      % weights
Q = w*f((a+b)/2+(b-a)/2*x)*(b-a)/2;   % integral
endfunction;
```

calculates the nodes and the weights for the N-point Gauss-Legendre quadrature and then integrates the given function.

Variable transformation quadratures

The idea behind *variable transformation quadratures* is to apply the given quadrature — either with optimimized or regularly spaced nodes — not to the original integral, but to a variable transformed integral [**?**],

$$\int_a^b f(x)dx = \int_{t_a}^{t_b} f\big(g(t)\big)g'(t)dt \approx \sum_{i=1}^{N} w_i f\big(g(t_i)\big)g'(t_i) \,, \tag{42}$$

where the transformation $x = g(t)$ is chosen such that the transformed integral better suits the given quadrature. Here g' denotes the derivative and $[t_a, t_b]$ is the corresponding interval in the new variable.

For example, the Gauss-Legendre quadrature assumes the integrand can be well represented with polynomials and performs poorly on integrals with integrable singularities like

$$I = \int_0^1 \frac{1}{2\sqrt{x}} dx \,. \tag{43}$$

However, a simple varibale transformation $x = t^2$ removes the singularity,

$$I = \int_0^1 dt\,, \tag{44}$$

and the Gauss-Legendre quadrature for the transformed integral gives exact result. The price is that the transformed quadrature performs less effectively on smooth functions.

Some of the popular variable transformation quadratures are Clenshaw-Curtis [**?**], based on the transformation

$$\int_{-1}^1 f(x)dx = \int_0^\pi f(\cos\theta)\sin(\theta)d\theta\,, \tag{45}$$

and "tanh-sinh" quadrature [**?**], based on the transformation

$$\int_{-1}^1 f(x)dx = \int_{-\infty}^\infty f\left(\tanh\left(\frac{\pi}{2}\sinh(t)\right)\right)\frac{\pi}{2}\frac{\cosh(t)}{\cosh^2\left(\frac{\pi}{2}\sinh(t)\right)}dt\,. \tag{46}$$

Generally the equally spaced trapezium rule is used after the transformation.

Adaptive quadratures

The higher order quadratures, say $n > 10$, suffer from round-off errors as the weights w_i generally have alternating signs. Again, using high order polynomials is dangerous as they typically oscillate wildly and may lead to Runge phenomenon. Therefore if the error of the quadrature is yet too big for a sufficiently large n quadrature, the best strategy is to subdivide the interval in two and then use the quadrature on the half-intervals. Indeed, if the error is of the order h^k, the subdivision would lead to reduced error, $2\left(\frac{h}{2}\right)^k < h^k$, if $k > 1$.

An *adaptive quadrature* is an algorithm where the integration interval is subdivided into adaptively refined subintervals until the given accuracy goal is reached.

Adaptive algorithms are usually built on pairs of quadrature rules (preferably using the same points for efficiency) – a higher order rule,

$$Q = \sum_i w_i f(x_i), \tag{47}$$

where w_i are the weights of the higher order rule and Q is the higher order estimate of the integral, and a lower order rule,

$$q = \sum_i v_i f(x_i), \tag{48}$$

where v_i are the weights of the lower order rule and q is the the lower order estimate of the integral. The difference between the higher order rule and the lower order rule gives an estimate of the error,

$$\delta Q = |Q - q| \tag{49}$$

The integration result is accepted, if the error δQ is smaller than tolerance,

$$\delta Q < \delta + \epsilon|Q|\,, \tag{50}$$

where δ is the absolute accuracy goal and ϵ is the relative accuracy goal of the integration.

Otherwise the interval is subdivided into two half-intervals and the procedure applies recursively to subintervals with the same relative accuracy goal ϵ and rescaled absolute accuracy goal $\delta/\sqrt{2}$.

The reuse of the function evaluations made at the previous step of adaptive integration is very important for the efficiency of the algorithm. The equally-spaced abscissas naturally provide for such a reuse.

As a (primitive) example, Table 2 shows an implementation of the described algorithm using

$$x_i = \left\{\frac{1}{6}, \frac{2}{6}, \frac{4}{6}, \frac{5}{6}\right\} \text{ (easily reusable points)}\,, \tag{51}$$

$$w_i = \left\{\frac{2}{6}, \frac{1}{6}, \frac{1}{6}, \frac{2}{6}\right\} \text{ (trapezium rule)}\,, \tag{52}$$

$$v_i = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\} \text{ (rectangle rule)}\,. \tag{53}$$

This implementation calculates directly the Riemann sums and can therefore deal with integrable singularities, although very inefficiently.

Table 2: Recursive adaptive algorithm

```
#include <cmath>
#include <functional> // to be compiled with -std=c++0x
#include <stdexcept>
#include <iostream>
using namespace std;

double adapt(
  const function<double(double)>& f, const double& a, const double& b,
  const double acc, const double eps,
  const double & f2, const double & f3, const int nrec)
{
if(nrec>100000) throw logic_error("adapt: too many recursions");
double f1        = f(a+(b-a)/6), f4 = f(a+5*(b-a)/6);
double Q         = (2*f1+f2+f3+2*f4)/6*(b-a);
double q         = (f1+f2+f3+f4)/4*(b-a);
double tolerance = acc+eps*fabs(Q);
double error     = fabs(Q-q);
if(error < tolerance) return Q;
else{
  double q1=adapt(f,a,a+(b-a)/2,acc/sqrt(2),eps,f1,f2,nrec+1);
  double q2=adapt(f,a+(b-a)/2,b,acc/sqrt(2),eps,f3,f4,nrec+1);
  return q1+q2;
  }
}

double adapt(
   const function<double(double)>& f, const double a, const double b,
   const double acc = 0.001, const double eps = 0.001)
{
double f2=f(a+2*(b-a)/6), f3=f(a+4*(b-a)/6); int nrec=0;
return adapt(f,a,b,acc,eps,f2,f3,nrec);
}

int main()
{
int ncalls=0;
function<double(double)> f = [&](double x){ncalls++; return 1/sqrt(x);};
double q = adapt(f,0,1);
cout << "q= " << q << " ncalls=" << ncalls << "\n";
return 0;
}
```

Gauss-Kronrod quadratures

Gauss-Kronrod quadratures represent a compromise between equally spaced abscissas and optimal abscissas: n points are reused from the previous iteration (n weights as free parameters) and then m optimal points are added (m abscissas and m weights as free parameters). Thus the accuracy of the method is $n + 2m - 1$. There are several special variants of these quadratures fit for particular types of the integrands.

Infinite intervals

One way to calculate an integral over infinite interval is to transform it — by a variable sustitution — into an integral over a finite interval. The latter can then be evaluated by ordinary integration methods. For example,

$$\int_{-\infty}^{+\infty} f(x)dx = \int_{-1}^{+1} f\left(\frac{t}{1-t^2}\right)\frac{1+t^2}{(1-t^2)^2}dt\ , \tag{54}$$

$$\int_{-\infty}^{+\infty} f(x)dx = \int_{0}^{1} \left(f\left(\frac{1-t}{t}\right)+f\left(-\frac{1-t}{t}\right)\right)\frac{dt}{t^2}\ , \tag{55}$$

$$\int_{a}^{+\infty} f(x)dx = \int_{0}^{1} f\left(a+\frac{t}{1-t}\right)\frac{1}{(1-t)^2}dt\ , \tag{56}$$

$$\int_{a}^{+\infty} f(x)dx = \int_{0}^{1} f\left(a+\frac{1-t}{t}\right)\frac{dt}{t^2}\ , \tag{57}$$

$$\int_{-\infty}^{a} f(x)dx = \int_{-1}^{0} f\left(a-\frac{t}{1+t}\right)\frac{-1}{(1+t)^2}dt\ , \tag{58}$$

$$\int_{-\infty}^{a} f(x)dx = \int_{0}^{1} f\left(a-\frac{1-t}{t}\right)\frac{dt}{t^2}\ , \tag{59}$$

$$\tag{60}$$