Numerical integration

Introduction

Numerical integration constitutes a broad family of algorithms to compute a numerical approximation to a definite (Riemann) integral.

Generally, the integral is approximated by a weighted sum of function values within the domain of integration,

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

Expression (1) is often referred to as *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 for a given problem depends on several factors, in particular on the integrand. Different classes of integrands generally require different quadratures for the most effective calculation.

A popular numerical integration library is QUADPACK [**?**]. It includes general purpose routines—like QAGS, basen on an adaptive GaussKronrod quadrature with acceleration—as well as a number of specialized routines. The GNU scientific library [**?**] (GSL) implements most of the QUADPACK routines and in addition includes a modern general-purpose adaptive routine CQUAD based on Clenshaw-Curtis quadratures [**?**].

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

Rectangle and trapezium rules

In mathematics, the *Reimann integral* is generally 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 $x_i \in [t_{i-1}, t_i]$ 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 adjucent 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 the average of two Riemann sums,

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

Rectangle and trapezium quadratures both have the important feature of closely following the very mathematical definition of the integral as the limit of the Riemann sums. Therefore—disregarding the round-off errors—these two rules cannot fail if the integral exists.

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 the basis for more advanced quadratures, for example adaptive quadratures and variable transformation quadratures considered below.

Quadratures with regularly spaced abscissas

A quadrature (1) with n predefined nodes x_i 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 conditions 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,

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

where the integrals

$$I_k \doteq \int_a^b \phi_k(x)dx \tag{13}$$

are assumed to be known. 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 functions (11).

A popular choice for predefined nodes is a *closed* set—that is, including the end-points of the interval—of evenly spaced abscissas,

$$x_i = a + \frac{i-1}{n-1}(b-a) \Big|_{i=1,\ldots,n} \ . \tag{14}$$

However, in practice it often happens that the integrand has an integrable sinfularity at one or both ends of the interval. In this case one can choose an *open set* of equidistant nodes,

$$x_i = a + \frac{i - \frac{1}{2}}{n}(b-a) \Big|_{i=1,\ldots,n} \ . \tag{15}$$

The set of functions to be integrated exactly is generally chosen to suite the properties of the integrands at hand: the integrands must be well represented by linear combinations of the chosen functions.

Table 1: Maxima script to calculate analytically the weights of an n-point classical quadrature with predefined abscissas in the interval $[0, 1]$.

```
n: 8; xs: makelist((i-1)/(n-1),i,1,n); /* nodes: adapt to your needs */
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 */
```

Classical quadratures

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

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k \,, \tag{16}$$

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

$$\{1, x, x^2, \ldots, x^{n-1}\} \tag{17}$$

as the set of functions to be integrated exactly.

This leads to the so called *classical quadratures*: quadratures with regularly spaced abscissas and *polynomials* as exactly integrable functions.

An n-point classical quadrature integrates exactly the first n terms of the function's Taylor expansion (16). The x^n order term will not be integrated exactly and will lead to an error of the quadrature. Thus the error E_n of the n-point classical quadrature is on the order of the integral of the x^n term in (16),

$$E_n \approx \int_a^b \frac{f^{(n)}(a)}{n!}(x-a)^n dx = \frac{f^{(n)}(a)}{(n+1)!}h^{n+1} \propto h^{n+1} \,, \tag{18}$$

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 with not so large n can provide a good approximation for the integral. 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.

Classical quadratures are mostly of historical interest nowadays. Alternative methods—such as quadratures with optimized abscissas, adaptive, and variable transformation quadratures—are more stable and accurate and are normally preferred to classical quadratures.

Classical quadratures with equally spaced abscissas—both closed and open sets—are generally referred to as *Newton-Cotes quadratures*. 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,

$$\{\phi_1(x), \ldots, \phi_{2n}(x)\} \,, \tag{19}$$

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

$$\sum_{i=1}^{n} w_i \phi_k(x_i) = I_k \Big|_{k=1,\ldots,2n} , \tag{20}$$

where again

$$I_k \doteq \int_a^b \phi_k(x) dx . \tag{21}$$

The weights and abscissas of the quadrature can be determined by solving this system of equations[1].

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,

$$\int_a^b \omega(x) f(x) dx , \tag{23}$$

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.

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

$$\int_a^b \omega(x) f(x) dx \approx \sum_{i=1}^{N} w_i f(x_i) , \tag{24}$$

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

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 $\omega(x)$,

$$\int_a^b \omega(x) p_n(x) p_k(x) \propto \delta_{nk} . \tag{25}$$

Now, one can prove that the optimal nodes for the N-point Gauss quadrature are the roots of the polynomial $p_N(x)$,

$$p_N(x_i) = 0 . \tag{26}$$

The idea behind the proof is to consider the integral

$$\int_a^b \omega(x) q(x) p_N(x) dx = 0 , \tag{27}$$

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

$$\sum_{i=1}^{N} q(x_i) p_N(x_i) = 0 . \tag{28}$$

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

[1] Here is, for example, an $n = 2$ quadrature with optimized abscissas,

$$\int_{-1}^{1} f(x) dx \approx f\left(-\sqrt{\tfrac{1}{3}}\right) + f\left(+\sqrt{\tfrac{1}{3}}\right) . \tag{22}$$

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,

$$xp_{n-1}(x) = \beta_n p_n(x) + \alpha_n p_{n-1}(x) + \beta_{n-1} p_{n-2}(x) \,, \tag{29}$$

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,

$$x\mathbf{p}(x) = \mathbf{J}\mathbf{p}(x) + \beta_N p_N(x)\mathbf{e}_N \,, \tag{30}$$

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 $\mathbf{J}$ — usually refered to as *Jacobi matrix* or *Jacobi operator* — is given as

$$\mathbf{J} = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \beta_3 & \\ & & \ddots & \ddots & \\ & & & \beta_{N-1} & \alpha_N \end{pmatrix} . \tag{31}$$

Substituting the roots x_i of p_N—that is, the set $\{x_i \mid p_N(x_i) = 0\}$—into the matrix equation (30) leads to eigenvalue problem for the Jacobi matrix,

$$\mathbf{J}\mathbf{p}(x_i) = x_i \mathbf{p}(x_i) \,. \tag{32}$$

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[2] routine ■.

The weights can be obtained considering N integrals,

$$\int_a^b \omega(x) p_n(x) dx = \delta_{n0} \int_a^b \omega(x) dx \,, \; n = 0, \ldots, N-1 \,. \tag{33}$$

Applying our quadrature gives the matrix equation,

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

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

$$\mathbf{P} \doteq \begin{pmatrix} p_0(x_1) & \ldots & p_0(x_N) \\ p_1(x_1) & \ldots & p_1(x_N) \\ \ldots & \ldots & \ldots \\ p_{N-1}(x_1) & \ldots & p_{N-1}(x_N) \end{pmatrix} . \tag{35}$$

Equation (34) 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 $\mathbf{P}$ apparently consists of non-normalized column eigenvectors of the matrix $\mathbf{J}$. The eigenvectors are orthogonal and therefore $\mathbf{P}^T\mathbf{P}$ is a diagonal matrix with positive elements. Multiplying (34) by $\mathbf{P}^T$ and then by $(\mathbf{P}^T\mathbf{P})^{-1}$ from the left gives

$$\mathbf{w} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{e}_1 \int_a^b \omega(x) dx \,. \tag{36}$$

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

$$w_i = \frac{1}{(\mathbf{P}^T\mathbf{P})_{ii}} \int_a^b \omega(x) dx \,. \tag{37}$$

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

Table 2: An Octave function which calculates the nodes and weights of the N-point Gauss-Legendre quadrature and then integrates a given 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;
```

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

$$\mathbf{V} = \sqrt{(\mathbf{P}^T\mathbf{P})^{-1}}\mathbf{P} \,. \tag{38}$$

Therefore, again taking into account that $p_0(x) = 1$, equation (37) can be written as

$$w_i = (V_{1i})^2 \int_a^b \omega(x)dx \,\blacksquare. \tag{39}$$

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

$$(2n-1)x\mathcal{P}_{n-1}(x) = n\mathcal{P}_n(x) + (n-1)\mathcal{P}_{n-2}(x) \,. \tag{40}$$

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

$$\sqrt{2n+1}\mathcal{P}_n(x) = p_n(x) \tag{41}$$

reduces this reccurence relation to the symmetric form (29),

$$xp_{n-1}(x) = \frac{1}{2}\frac{1}{\sqrt{1-(2n)^{-2}}}p_n(x) + \frac{1}{2}\frac{1}{\sqrt{1-(2(n-1))^{-2}}}p_{n-2}(x) \,. \tag{42}$$

Correspondingly, the coefficients in the matrix $\mathbf{J}$ are

$$\alpha_n = 0 \,, \quad \left\{ \beta_n = \frac{1}{2}\frac{1}{\sqrt{1-(2n)^{-2}}} \;\middle|\; n = 1, \ldots, N-1 \right\} \,. \tag{43}$$

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 (43).

As an illustration of this algorithm Table (2) shows an Octave function which calculates the nodes and the weights of the N-point Gauss-Legendre quadrature and then integrates a given function.

Gauss-Kronrod quadratures

Generally, the error of a numerical integration is estimated by comparing the results from two rules of different orders. However, for ordinary Gauss quadratures the nodes for two rules of different orders almost never coinside. This means that one can not reuse the points of the lower order rule when calculating the hihger order rule.

Gauss-Kronrod algorithm [**?**] remedies this inefficiency. The points inherited from the lower order rule are reused in the higher order rule as predefined nodes (with n weights as free parameters), and then m more optimal points are added (m abscissas and m weights as free parameters). The order of the method is $n + 2m - 1$. The lower order rule becomes *embedded*—that is, it uses a subset of the nodes—into the higher order rule. On the next iteration the procedure is repeated.

Patterson [**?**] has tabulated nodes and weigths for several sequences of embedded Gauss-Kronrod rules.

Adaptive quadratures

Higher order quadratures 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's phenomenon. Therefore, if the error of the quadrature is yet too large for a quadrature with sufficiently large n, 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, $2\,(h/2)^k < h^k$, 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 – a higher order rule,

$$Q = \sum_i w_i f(x_i), \tag{44}$$

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,

$$q = \sum_i v_i f(x_i), \tag{45}$$

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,

$$\delta Q = |Q - q|\,. \tag{46}$$

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

$$\delta Q < \delta + \epsilon |Q|\,, \tag{47}$$

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

If the error estimate is larger than tolerance, 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 points x_i are usually chosen such that the two quadratures use the same points, and that the points can be reused in the subsequent recursive steps. 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 an example, Table 3 shows an implementation of the described algorithm using

$$x_i = \left\{\frac{1}{6}, \frac{2}{6}, \frac{4}{6}, \frac{5}{6}\right\} \text{ (easily reusable points)}\,, \tag{48}$$

$$w_i = \left\{\frac{2}{6}, \frac{1}{6}, \frac{1}{6}, \frac{2}{6}\right\} \text{ (trapezium rule)}\,, \tag{49}$$

$$v_i = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\} \text{ (rectangle rule)}\,. \tag{50}$$

During recursion the function values at the points #2 and #3 are inherited from the previous step and need not be recalculated.

The points and weights are cited for the rescaled integration interval $[0, 1]$. The transformation of the points and weights to the original interval $[a, b]$ is given as

$$\begin{aligned} x_i &\to a + (b-a)x_i\,, \\ w_i &\to (b-a)w_i\,. \end{aligned} \tag{51}$$

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

More efficient adaptive routines keep track of the subdivisions of the interval and the local errors [**?**]. This allows detection of singularities and switching in their vicinity to specifically tuned quadratures. It also allows better estimates of local and global errors.

Table 3: Recursive adaptive integrator in C

```
#include<math.h>
#include<assert.h>
double adapt24(double f(double),double a, double b,
double acc, double eps, double f2, double f3, int nrec)
{ assert(nrec<1000000);
  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), q=(f1+f4+f2+f3)/4*(b-a);
  double tolerance=acc+eps*fabs(Q), error=fabs(Q-q);
  if(error < tolerance) return Q;
  else {
    double Q1=adapt24(f,a,(a+b)/2,acc/sqrt(2.),eps,f1,f2,nrec+1);
    double Q2=adapt24(f,(a+b)/2,b,acc/sqrt(2.),eps,f3,f4,nrec+1);
    return Q1+Q2; }
}
double adapt(double f(double),double a,double b,
double acc,double eps)
{ double f2=f(a+2*(b-a)/6),f3=f(a+4*(b-a)/6); int nrec=0;
  return adapt24(f,a,b,acc,eps,f2,f3,nrec);
}
#include<stdio.h>
int main() //uses gcc nested functions
{ int ncalls=0; double a=0,b=1,acc=0.001,eps=0.001;
  double f(double x){ncalls++; return 1/sqrt(x);};//nested function
  double Q=adapt(f,a,b,acc,eps);
  printf("Q=%g ncalls=%d\n",Q,ncalls);
  return 0 ;
}
```

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 [**?**],

$$\int_a^b f(x)dx = \int_{t_a}^{t_b} f\big(g(t)\big)g'(t)dt \approx \sum_{i=1}^{N} w_i f\big(g(t_i)\big)g'(t_i)\,, \tag{52}$$

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

$$I = \int_0^1 \frac{1}{2\sqrt{x}}dx\,. \tag{53}$$

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

$$I = \int_0^1 dt\,, \tag{54}$$

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

$$\int_{-1}^{1} f(x)dx = \int_0^{\pi} f(\cos\theta)\sin\theta d\theta\,, \tag{55}$$

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

$$\int_{-1}^{1} f(x)dx = \int_{-\infty}^{\infty} f\left(\tanh\left(\frac{\pi}{2}\sinh(t)\right)\right)\frac{\pi}{2}\frac{\cosh(t)}{\cosh^2\left(\frac{\pi}{2}\sinh(t)\right)}dt\,. \tag{56}$$

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

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. Table 4 lists several of such transformation.

Table 4: Variable transformations reducing infinite interval integrals into integrals over finite intervals.

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

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

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

$$\int_{a}^{+\infty} f(x)dx = \int_{0}^{1} f\left(a+\frac{1-t}{t}\right)\frac{dt}{t^2}\,, \tag{60}$$

$$\int_{-\infty}^{a} f(x)dx = \int_{-1}^{0} f\left(a-\frac{t}{1+t}\right)\frac{-1}{(1+t)^2}dt\,, \tag{61}$$

$$\int_{-\infty}^{a} f(x)dx = \int_{0}^{1} f\left(a-\frac{1-t}{t}\right)\frac{dt}{t^2}\,. \tag{62}$$