Efficient computation of (classical) Gaussian quadrature rules

A. Gil, D. Ruiz-Antolín, J. Segura, N. M. Temme

Departamento de Matemáticas, Estadística y Computación
Universidad de Cantabria, Spain

SIAM Annual Meeting 2016

Given $I(f) = \int_a^b f(x)w(x)dx$, with $w(x)$ a weight function, the n-point quadrature rule

$$Q_n(f) = \sum_{i=1}^{n} w_i f(x_i) \tag{1}$$

is a Gaussian quadrature if $I(f) = Q_n(f)$ for f any polynomial with $\deg(f) \leq 2n - 1$.

Gaussian quadrature rules are optimal in a very specific sense and they are one of the more widely used methods of integration.

The question is, how to compute the nodes x_i and weights (or Christoffel numbers) w_i?

1. The nodes x_i, $i = 1, \ldots, n$ of the Gaussian quadrature rule are the roots of the (for instance monic) orthogonal polynomial satisfying
$$\int_a^b x^i p_n(x) w(x) dx = 0,\ i = 0, \ldots, n-1. \tag{2}$$
2. The Christoffel-Darboux formula gives the following expression for the weights in terms of monic polynomials
$$w_j = -\frac{||p_n||^2}{p_n'(x_j)p_{n+1}(x_j)},\ ||p_n||^2 = \int_a^b p_n(x)^2 w(x) dx$$
3. Recurrence relation for monic polynomials
$$p_{k+1}(x) = (x - B_k)p_k(x) - A_k p_{k-1}(x), \quad k = 1, 2, \ldots, \tag{3}$$
where $A_0 p_{-1} \equiv 0$, $A_k = \dfrac{||p_k||^2}{||p_{k-1}||^2}$, $k \geq 1$, $B_k = \dfrac{\langle xp_k, p_k \rangle}{||p_k||^2}$, $k \geq 0$, and
$$< f, g >= \int_a^b f(x)g(x)w(x)dx,\ ||f|| =< f, f >$$

The Golub-Welsch algorithm

Let

$$J = \begin{pmatrix} \beta_0 & \alpha_1 & 0 & \dots & 0 \\ \alpha_1 & \beta_1 & \alpha_2 & & \\ 0 & \alpha_2 & \beta_2 & & \vdots \\ \vdots & & & \ddots & \alpha_{n-1} \\ 0 & \dots & & \alpha_{n-1} & \beta_{n-1} \end{pmatrix}$$

$\alpha_i = \sqrt{A_i}$, $\beta_i = B_i$. Then the n different eigenvalues of J are the nodes. Furthermore, if $\vec{\Phi}^{(j)}$ is an eigenvector with eigenvalue the node x_j:

$$w_j = \mu_0 \frac{(\Phi_1^{(j)})^2}{||\vec{\Phi}^{(j)}||_E^2}$$

where $\Phi_1^{(j)}$ is the first component of $\vec{\Phi}^{(j)}$ and $\mu_0 = \int_a^b w(x)dx$.

Complexity: $\mathcal{O}(n^2)$

For the iterative computation of the nodes and weights we need:

1. A method to compute the polynomials $p_n(x)$ and the first derivatives.
2. A method to compute the roots of $p_n(x)$ (nodes x_i)
3. Depending on the selection of the iterative method: good starting values ensuring convergence (this is rarely proved).

Classical Gaussian quadrature

Iterative methods are restricted to the classical cases, characterized by the fact that the OPs are solutions of second order ODEs

$$C(x)y_n''(x) + B(x)y_n'(x) + \lambda_n y(x) = 0$$

(C and B polynomials).
The classical cases are:

1. Hermite: $w(x) = e^{-x^2}$ in $(-\infty, +\infty)$
2. Laguerre: $w(x) = x^{-\alpha}e^{-x}$, $\alpha > -1$, in $(0, +\infty)$
3. Jacobi: $w(x) = (1-x)^{\alpha}(1+x)^{\beta}$, $\alpha, \beta > -1$, in $(-1, 1)$

Apart from being solution of a second order ODE, the coefficients of the three-term recurrence relation are simple, as well as the coefficients in

$$y_n'(x) = a_n(x)y_n(x) + b_n y_{n-1}(x)$$

Recent references on the computation of classical Gauss quadrature

E. Yakimiw, Accurate computation of weights in classical Gauss-Christoffel quadrature rules.
J. Comput. Phys. (1996) Legendre (Hermite and Laguerre $\alpha = 0$ to a lesser extent)

K. Petras, On the computation of the Gauss-Legendre quadrature formula with a given precision.
J. Comput. Appl. Math. (1999) Legendre

P. N. Swarztrauber, On computing the points and weights for Gauss-Legendre quadrature.
SIAM J. Sci. Comput. (2002) Legendre

A. Glaser, X. Liu, V. Rokhlin, A fast algorithm for the calculation of the roots of special functions.
SIAM J. Sci. Comput. (2007) Hermite, Laguerre ($\alpha = 0$), Legendre

J. Segura, Reliable computation of the zeros of solutions of second order linear ODEs using a fourth order method.
SIAM J. Numer. Anal. (2010) Hermite, Laguerre, Jacobi (*)

I. Bogaert, B. Michiels, J. Fostier, J., O(1) computation of Legendre polynomials and Gauss-Legendre nodes and weights for parallel computing.
SIAM J. Sci. Comput. (2012) Legendre

N. Hale, A. Townsend, Fast and accurate computation of Gauss-Legendre and Gauss-Jacobi quadrature nodes and weights.
SIAM J. Sci. Comput. (2013) Jacobi

I. Bogaert, Iteration-free computation of Gauss-Legendre quadrature nodes and weights.
SIAM J. Sci. Comput. (2014) Legendre

A. Townsend, T. Trogdon, S. Olver, Fast computation of Gauss quadrature nodes and weights on the whole real line.
IMA J. Numer. Anal (to appear) Hermite

A. Townsend, The race to compute high-order Gauss-Legendre quadrature.
SIAM News (2015)

P. Bremer, On the numerical calculation of the roots of special functions satisfying second order ordinary differential equations.

Worth noting:

◯ All papers use Newton's method for computing the roots (order 2), with the exception of Yamikiw and Petras papers (higher derivatives are needed) and JS.

Newton's method:

The NM, $x^{(n+1)} = x^{(n)} - \dfrac{f(x^{(n)})}{f'(x^{(n)})}$, has order of convergence 2 because

$$\frac{\epsilon_{n+1}}{\epsilon_n^2} = \frac{f''(\alpha)}{2f'(\alpha)} + \mathcal{O}(\epsilon_n) \text{ as } n \to \infty,\ \epsilon_n = x^{(n)} - \alpha.$$

◯ The only proof of convergence for Newton method is for the Legendre case (Petras, 1999).

If a function satisfies and EDO, use it to speed up the method!

For computing zeros of solutions of

$$w''(x) + B(x)w'(x) + C(x)w(x) = 0 \tag{4}$$

we can take $y(x) = \exp\left(\int \frac{1}{2}B(x)dx\right) w(x)$ and then $y''(x) + A(x)y(x) = 0$, with $A(x) = C(x) - \frac{1}{2}B'(x) - \frac{1}{4}B(x)^2$. Now

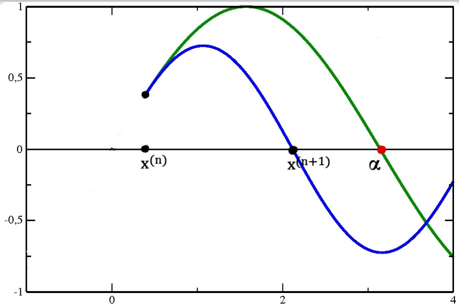
$$\frac{y(x)}{y'(x)} = \frac{w(x)}{\frac{1}{2}B'(x)w(x) + w'(x)}$$

and the Newton method $x^{(n+1)} = x^{(n)} - \frac{y(x^{(n)})}{y'(x^{(n)})}$ is of **third order.**

And we can do better!

Algorithm (Zeros of $y''(x)+A(x)y(x)=0$, $A(x)$ decreasing)

Given $x^{(n)}$, the next iterate $x^{(n+1)}$ is computed as follows: find a solution of the equation $w''(x)+A(x^{(n)})w(x)=0$ such that $y(x^{(n)})w'(x^{(n)})-y'(x^{(n)})w(x^{(n)})=0$ and take as $x^{(n+1)}$ the zero of $w(x)$ closer to $x^{(n)}$ and larger than $x^{(n)}$.



Equations: $y''(x)+A(x)y(x)=0$, $w''(x)+A(x^{(n)})w(x)=0$, $(A'(x)<0)$

The method is equivalent to iterating $x_{n+1} = T(x_n)$ with the following fixed point iteration.

Let $h(x) = y(x)/y'(x)$, $j = \mathrm{sign}(A'(x))$, we define

$$T(x) = x - \frac{1}{\sqrt{A(x)}}\mathrm{arctan}_j(\sqrt{A(x)}h(x))$$

with

$$\mathrm{arctan}_j(\zeta) = \begin{cases} \mathrm{arctan}(\zeta) \text{ if } jz > 0, \\ \mathrm{arctan}(\zeta) + j\pi \text{ if } jz \leq 0, \\ j\pi/2 \text{ if } z = \pm\infty \end{cases}$$

This method converges to α for any x_0 in $[\alpha', \alpha)$ if $A'(x) < 0$, with α' the largest zero smaller than α (analogously for $A'(x) > 0$).

The method has fourth order convergence:

$$\epsilon_{n+1} = \frac{A'(\alpha)}{12}\epsilon_n^4 + \mathcal{O}(\epsilon_n^5),\ \epsilon_k = x_k - \alpha$$

Computing the zeros in an interval where $A(x)$ is monotonic.

The basic algorithm is remarkably simple:

Algorithm

Computing zeros for $A'(x) < 0$

1. *Iterate $T(x)$ starting from $x^{(0)}$ until an accuracy target is reached. Let α be the computed zero.*
2. *Take $x^{(0)} = T(\alpha) = \alpha + \pi/\sqrt{A(\alpha)}$ and go to 1.*

Repeat until the interval where the zeros are sought is swept. For $A'(x) > 0$ the same ideas can be applied but the zeros are computed in decreasing order.
See JS, SIAM J. Numer. Anal. (2010).

Requirement: the monotonicity properties of $A(x)$ should be known in advance in order to compute zeros in sub-intervals where $A(x)$ is monotonic.

Our algorithm has some connection with the Glaser-Liu-Rokhlin algorithm (GLR).

Our algorithm uses $T(x) = x - \frac{1}{\sqrt{A(x)}} \arctan\left(\sqrt{A(x)}\frac{y(x)}{y'(x)}\right)$

In GLR, they consider a Prüfer transformation of the ODE $p(x)u''(x) + q(x)u'(x) + r(x) = 0$, defining

$$\theta(x) = \arctan\left(\sqrt{\frac{r}{p}}\frac{u}{u'}\right)$$

Once a zero α is computed, the next one is estimated by integrating the first order ODE satisfied by $x(\theta)$ with initial value $x(0) = \alpha$. The next zero is given by $x(\pi)$. This is used as starting value for the Newton method (and the OPs are computed by Taylor series).

Differently from GLR our algorithm has guaranteed fourth order convergence. No ODE integration is required for a first estimation of the zeros.

The only numerical concern will be to compute accurately the OPs (provided we have an ODE $y''(x) + A(x)y(x) = 0$ with known monotonicity properties of $A(x)$).

Gauss-Hermite quadrature

The function $y(x) = C_n e^{-x^2/2} H_n(x)$ satisfies

$$y_n''(x) + A(x) y_n(x) = 0,\ A(x) = 2n + 1 - x^2.$$

The nodes are symmetric around the origin.

We start $x = 0$ and compute zeros in increasing order until we have computed $\lfloor n/2 \rfloor$ zeros. The first step is

$$x = T_{-1}(0) = \begin{cases} \dfrac{\pi}{\sqrt{2n+1}}, & n \text{ odd } (h(0^+) = 0^+) \\ \dfrac{\pi}{2\sqrt{2n+1}}, & n \text{ even } (h(0^+) = +\infty) \end{cases}$$

As $n \to +\infty$ the coefficient $A(x)$ is essentially constant for not too large x. In this sense, the method will be asymptotically exact.

Methods are available for computing efficiently and reliably $y_n(x)$, also for large n:

Computing the Real Parabolic Cylinder Functions U(a,x), V(a,x).
A. Gil , J. Segura, N.M. Temme.
ACM Trans. Math. Softw. 32(1) (2006) 70-101

Algorithm 850: Real Parabolic Cylinder Functions U(a,x), V(a,x).
A. Gil, J. Segura, N.M. Temme.
ACM Trans. Math. Softw. 32(1) (2006) 102-112

This algorithm uses two different asymptotic approximations for large n (in terms of elementary or Airy functions)

A simpler approach is also possible: use local Taylor series.

By differentiating the ODE satisffied by $y(x) = C_n e^{-x^2/2} H_n(x)$ we have

$$y^{(k+2)} + (2n + 1 - x^2)y^{(k)} - 2kxy^{(k-1)} - k(k-1)y^{(k-2)} = 0$$

Stability: Perron-Kreuser theorem does not give conclusive information.

All solutions of this difference equation satisfy: $\limsup_{k\to+\infty} \left(|y^{(k)}|/(k!)^{2/3}\right)^{1/k} = 1$

Use the derivatives to compute

$$y(x_0 + h) = \sum_{k=0}^{\infty} \frac{y^{(k)}(x_0)}{k!} h^k$$

and similarly for $y'(x_0 + h)$, truncating for a given precision.

In our case h will be always less than the maximal distance between zeros of $H_n(x)$.

Algorithm for Gauss-Hermite based on local Taylor series.

As before $y_n = (2^n n!)^{-1/2} e^{-x^2/2} H_n(x)$.

1. With $x = \pi/\sqrt{2n+1}$ for n odd and $x = \pi/(2\sqrt{2n+1})$ for n even. Let $i = 1$.
2. Iterate the fixed point method

$$T(x) = x - \frac{1}{\sqrt{A(x)}} \arctan_{-1}(\sqrt{A(x)}h(x))$$

   until convergence is reached within a given accuracy. The values of $y(x)$ and $y'(x)$ are computed from Taylor series, centered at the previous point.

   Let x_i be the resulting zero (node)
3. The corresponding weight is given by $w_i = 2e^{-x_i^2}/(y_n'(x_i))^2$.
4. Set $x = x_i + \pi/\sqrt{A(x_i)}$, $i = i + 1$, and go to 2 if $i \leq \lfloor n/2 \rfloor$.

Some features of the algorithm

1. The Taylor algorithm is nearly as fast for large n as the algorithm based on asymptotics for Hermite, but much simpler (around 60 code lines).
2. The method does not need initial estimations for the roots and they don't improve significantly the performance. Typically 2 iterations are needed for full double accuracy, except for the largest zeros, which require 3.
3. Full double precision accuracy is obtained for all the nodes and for any n (differently from GLR).
4. There is some degradation in relative accuracy for the normalized weights $\tilde{w}_i$ ($\tilde{w}_i = e^{x_i^2} w_i$), as n becomes large, particularly for the weights for the largest nodes, which in any case are two orders of magnitud smaller errors than those in GLR. The errors range from 10^{-15} for 10^3 nodes (or lower) to 10^{-11} for 10^6 nodes.
5. It appears to be faster than GLR, particularly for extended (quadruple) precision. In double precision, each node/weight is computed in less than $0.5\mu s$ (in my laptop).

Gauss-Laguerre quadrature

Take $z(x) = \sqrt{x}$, then
A good starting point consists in considering $y(z) = z^{\alpha+1/2} e^{-z^2/2} L_n^{(\alpha)}(z^2)$, which satisfies $\ddot{y}(z) + A(z)y(z) = 0$ with

$$A(x) = A(z(x)) = -x + 2L + \frac{\frac{1}{4} - \alpha^2}{x}, x = z^2.$$

The coefficient $A(x)$ is decreasing for positive x if $|\alpha| \leq 1/2$ and has a maximum at $x_e = \sqrt{\alpha^2 - 1/4}$ if $|\alpha| > 1/2$.

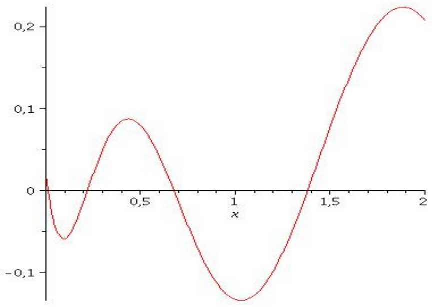
For large n, the above change $z(x) = \sqrt{x}$ is an interesting transformation because the method becomes asymptotically exact as $n \to +\infty$.

Other changes for which the Liouville transformations of the Laguerre ODE leads to a simple analysis are $z(x) = x^m$, $m \in \mathbb{R}$ and $z(x) = \log(x)$ (A. Deaño, A. Gil, JS, 2004).

The question is now: how to compute $L_n^{(\alpha)}(x)$.

Differently form Hermite, we can not start by computing the smallest root using Taylor series ($x = 0$ is a singular point of the ODE). In fact, we will need alternatives to the Taylor series for computing more than a zero...

Plot of $L_{20}^{(-0.8)}(x)$.



The first three or four zeros can not be computed with Taylor series.

A new (and first) algorithm for the efficient computation of Laguerre polynomials including high orders: A. Gil, J. Segura, N. M. Temme, "Efficient computation of Laguerre polynomials" (submitted)

Apart from recurrences for moderate degree ($n \leq 200$), our algorithm uses:

1. A simple expansion for small x in terms of Bessel functions:
$$L_n^{(\alpha)}(x) = \left(\frac{x}{n}\right)^{-\frac{1}{2}\alpha} e^{\frac{1}{2}x} \left(J_\alpha\left(2\sqrt{nx}\right) A(x) - \sqrt{\frac{x}{n}} J_{\alpha+1}\left(2\sqrt{nx}\right) B(x) \right)$$
$A(x)$ and $B(x)$ are given as asymptotic expansions in powers of n^{-1}.
2. A not so simple expansion in terms of Bessel functions for not so small x (from Frenzen & Wong, 1988)
3. Uniform expansion in terms of Airy functions (from Frenzen & Wong, 1988)

Additional expansions in terms of Bessel functions or in terms of Hermite polynomials can be considered for large α, but the coefficients are hard to compute.

Recurrence over α is also possible, but the stability has to be carefully analyzed.

With this we have built an algorithm for the computation of Gauss-Laguerre quadratures for $-1 < \alpha \leq 20$ and unrestricted *n*.

Performance:

1. Typically 2 iterations are neeeded for full double accuracy, except for the largest and smallest zeros, which require 3.
2. Double precision accuracy is obtained for all the nodes and for any *n* (differently from GLR, for which only $\alpha = 0$ was considered).
3. There is some degradation in relative accuracy for the normalized weights $\tilde{w}_i$ ($\tilde{w}_i = e^{x_i} w_i$) as *n* becomes large. For 10^n nodes, the worst relative accuracy of the normalized weights is of the order of 10^{n-16}.
4. The algorithm is nearly as fast as Hermite's

Gauss-Jacobi quadrature

We will talk about the Jacobi case some other day...

We only mention that the most appropriate starting point is the ODE satisfied by $y(\theta)=\left(\sin\frac{\theta}{2}\right)^{\alpha+1/2}\left(\cos\frac{\theta}{2}\right)^{\beta+1/2}P_n(\cos\theta)$:

$$\frac{d^2y}{d\theta^2}+\frac{1}{4}\left[L^2+\frac{\frac{1}{4}-\alpha^2}{\sin^2(\theta/2)}+\frac{\frac{1}{4}-\beta^2}{\cos^2(\theta/2)}\right]y=0$$

$L=2n+\alpha+\beta+1$

The change $x=\cos\theta$ is not the only possibility (see Deaño, Gil, Segura (2004)) but in this variable the method is asymptotically exact as $n\to+\infty$.

Status of the algorithms and software

1. Maple codes for all classical quadrature formulas and for various changes of variable are available (reliable but slow).
2. A Fortran 95 code is available for Gauss-Hermite. The code outperforms the Glaser-Liu-Rokhlin algorithm and it is simpler.
3. A Fortran 95 algorithm for Gauss-Laguerre is available for $-1 < \alpha < 20$ and practically unlimited orders.
4. An exponential-type algorithm based on Taylor series for Laguerre is also available but it is not stable for all parameters (but it is an interesting approach for large α).
5. There is a need of effective asymptotic expansions for Laguerre polynomials with large α.
6. Jacobi: Maple codes are available. Fortran codes will be soon available.

Our goal: to offer a Fortran package for a fast, accurate and reliable computation of classical gaussian quadratures.

THANK YOU!