Basics on orthogonal polynomials and Gaussian rules

Alvise Sommariva

Padua, autumn, 2024

Doctoral Program in Mathematical Sciences, Padua (I), Autumn 2024

"Numerical cubature and its applications"

Purpose

In this presentation we consider the basics of the Gaussian rules, for numerical integration over an interval Ω via a weighted sum, that is

$$\int_{\Omega} f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k).$$

Differently from Newton-Cotes type formulas, we do not assume the interval to be bounded or closed.

Typical intervals Ω are

- bounded, as the unit interval $[-1, 1]$;
- the half-line $[0, +\infty)$;
- the real-line $(-\infty), +\infty)$.

For details, see e.g. [1, p.95], [2].

Definition (Weight function)

Let $w : (a, b) \to \mathbb{R}$ be a function such that

- w è is nonnegative in (a, b) (not necessarily bounded)),
- $\int_a^b |x|^n \, w(x) \, dx < +\infty$ for all $n \in \mathbb{N}$;
- $\int_a^b g(x) \, w(x) \, dx = 0$ for some continuous and nonegative function g implies $g \equiv 0$ in (a, b).

Such a w is named weight function.

Example

The classical weight functions are

1. $w(x) = 1$ with $x \in [-1, 1]$ (Legendre weight);
2. $w(x) = \frac{1}{\sqrt{1-x^2}}$ con $x \in (-1, 1)$ (Chebyshev weight);
3. $w(x) = (1 - x^2)^{\gamma-(1/2)}$ con $x \in (-1, 1)$, $\gamma > (-1/2)$ (Gegenbauer weight);
4. $w(x) = (1 - x)^{\alpha} \cdot (1 + x)^{\beta}$ con $x \in (-1, 1)$, $\alpha > -1$, $\beta > -1$ (Jacobi weight);
5. $w(x) = \exp(-x)$ con $x \in (0, +\infty)$ (Laguerre weight);
6. $w(x) = \exp(-x^2)$ con $x \in (-\infty, +\infty)$ (Hermite weight);

Definition (Orthogonal polynomials)

A triangular family of polynomials $\{\phi_k\}_{k=0,\ldots,n}$ (that is $\deg(\phi_k) = k$) is orthogonal w.r.t. the weight function w on its reference interval (a, b) if and only if

$$(\phi_i, \phi_j)_{2,w} = c_i \, \delta_{i,j}, \quad i, j = 0, \ldots, n$$

where

- $(f, g)_{2,w} := \int_a^b f(x)\, g(x) dx$,
- $\delta_{i,j}$ is the Kronecker delta,
- $c_i > 0$, $i = 0, \ldots, n$.

The family is orthonormal if $c_i = 1$ for $i = 0, \ldots, n$.

Remark

- *Orthogonal polynomials were already known by Legendre, Laplace (1810) and Jacobi (1826), that using them proved the existance of Gaussian rules in* $[-1,1]$*, for* $w(x)=1$ *(result known already to Gauss (1814)).*
- *The term orthogonal polynomials was introduced by Schmidt in 1905.*
- *The Jacobi weight* $w(x)=(1-x)^{\alpha}\cdot(1+x)^{\beta}$ *with* $x\in(-1,1)$*,* $\alpha>-1$*,* $\beta>-1$ *was introduced by Jacobi (1859) and studied by Mehler in numerical cubature (1864).*
- *The Laguerre weight function was introduced by this scientist in 1879.*
- *The Hermite weight function was investigated in 1864, though already known to Lagrange (1762), Abel (1826), Murphy (1835), Chebyshev (1859) and the relative orthogonal polynomials used by Laplace (1810).*

For many other historical details see A Survey of Gauss-Christoffel Quadrature Formulae by W. Gautschi.

In this framework it is fundamental the following theorem.

Teorema (Zeros of orthogonal polynomials, Christoffel 1877)

Let $\{\phi_k\}_{k=0,\dots,n}$ be a triangular family of orthogonal polynomials in (a, b) w.r.t. the weight function "w". The zeros of ϕ_n

- *are exactly n,*
- *they have multiplicity 1,*
- *belong to the open interval (a, b).*

Definition (Monic polynomial)

A polynomial $p_n(x) = \sum_{k=0}^{n} a_k x^k$ is monic if $a_n = 1$.

Theorem (Three terms recursion (Christoffel 1877, Darboux 1878, Stieltjes 1884))

Let $\{\phi_k\}_{k=0,\dots,n}$ be a triangular family of monic polynomial, orthogonal in (a, b) w.r.t. a weight function $w : (a, b) \to$ 'R.

Let $\phi_{-1}(x) = 0$, $\phi_0(x) = 1$, Then for $n \geq 1$

$$\phi_{n+1}(x) = (x - \beta_n)\phi_n(x) - \gamma_n \phi_{n-1}(x)$$

where

$$\beta_n = \frac{(x\phi_n, \phi_n)_{2,w}}{(\phi_n, \phi_n)_{2,w}}, \quad \gamma_n = \frac{(\phi_n, \phi_n)_{2,w}}{(\phi_{n-1}, \phi_{n-1})_{2,w}}.$$

Remark

Observe that

- *given ϕ_0 and ϕ_1, the procedure determines the triangular family of orthogonal monic polynomials of degree $n+1$, as soon as the coefficients α_k, β_k, γ_k are available.*
- *if ϕ_n is such that $(\phi_n, \phi_k) = 0$ for $k = 0, \ldots, n-1$ then for $\tau \neq 0$ also $\tilde{\phi}_n = \tau\phi_n$ is such that*

$$(\tilde{\phi}_n, \phi_k) = (\tau\phi_n, \phi_k) = \tau(\phi_n, \phi_k) = 0, \;\; k = 0, \ldots, n-1$$

is an orthogonal polynomial of degree n.

Remark

The case in which the family of orthogonal polynomials $\{\hat{\phi}_k\}_{k=0,\dots,n}$ *is asked to be orthonormal can be easily derived by that in monic form.*

If $\gamma_0 = \int_a^b w(x)dx$ *and all the monic polynomials of degree* $0, \dots, k$ *are available, then one may evaluate*

$$\gamma_j = \frac{(\phi_j, \phi_j)_{2,w}}{(\phi_{j-1}, \phi_{j-1})_{2,w}}, \quad j = 0, \dots, k.$$

Since

$$\|\phi_k\|_{2,w}^2 = \gamma_0 \cdot \dots \cdot \gamma_k, \qquad \hat{\phi}_k = \frac{\phi_k}{\|\phi_k\|_{2,w}},$$

one can determine the triangular family of orthonormal polynomials w.r.t. the weight function w.

Gaussian rules

Problema. (Formulas with n nodes and ADE equal to $2n-1$)

Do exist $x_1, \ldots, x_n$ and weights $w_1, \ldots, w_n$ (the so called Gauss-weight name) for which the relative quadrature rules of interpolatory type have ADE $\delta = 2n-1$, that is they compute exactly $\int_a^b p(x)w(x)\,dx$ for any polynomial p whose degree is inferior or equal to $2n-1$?

Teorema (Existence and uniqueness of Gaussian rules (Jacobi, 1826))

For each $n \geq 1$ exist and are unique $x_1, \ldots, x_n$ and weights $w_1, \ldots, w_n$ such that the degree of exactness of the relative rule is $2n-1$, that is

$$I_w(p_{2n-1}) := \int_a^b p_{2n-1}(x)w(x)\,dx = \sum_{k=1}^{n} w_k p_{2n-1}(x_k), \quad p_{2n-1} \in \mathbb{P}_{2n-1}$$

The nodes are the zeros of an orthogonal polynomial of degree n (w.r.t. w),

$$\phi_n(x) = A_n \cdot (x - x_1) \cdot \ldots \cdot (x - x_n)$$

and the weights are

$$w_i = \int_a^b L_i(x)w(x)dx = \int_a^b L_i^2(x)w(x)dx > 0, \quad i = 1, \ldots, n$$

Gaussian rules in Matlab

With the exception of few cases (as in the case of Chebyshev weight function), the nodes and the weights of a Gaussian rule are not explicit.

- Many years ago they were listed in specific manuals and the user had to copy them in his own files (inevitable the introduction of typos!).
- Next they were computed and listed in cards.
- In 1969 Golub and Welsch introduced in Calculation of Gauss Quadrature Rules a novel algorithm for the computation of the nodes and the weights, based on numerical linear algebra. Walter Gautschi and collaborators wrote a collection of codes for the computation of Gaussian rules for a large variety of weight functions.
- In last years there have been an increasing interest on algorithms for the computation of Gaussian rules up to several thousands points (see, e.g., Fast and Accurate Computation of Gauss-Legendre and Gauss-Jacobi Quadrature Nodes and Weights by N. Hale and A. Townsend and references therein). Specific routines from Chebfun package are legpts.m jacpts.m, lagpts.m, jacpts.m.

We briefly introduce the approach used by Golub-Welsch, in the implementation by W. Gautschi.
First it computes *the recurrence coefficients*, by the routine r_jacobi.m.

The call is

```
ab=r_jacobi(n,a,b)
```

that determines n recurrence coefficients of orthogonal monic polynomials relatively to the Jacobi weight function

$$w(x) = (1-x)^a (1+x)^b.$$

Gaussian rules in Matlab: recurrence

Below we give the code of the function `r_jacobi`

```
function ab=r_jacobi(N,a,b)

nu=(b-a)/(a+b+2);
mu=2^(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+2);
if N==1
 ab=[nu mu]; return
end

N=N-1;
n=1:N;
nab=2*n+a+b;
nuadd=(b^2-a^2)*ones(1,N)./(nab.*(nab+2));
A=[nu nuadd];
n=2:N;
nab=nab(n);
B1=4*(a+1)*(b+1)/((a+b+2)^2*(a+b+3));
B=4*(n+a).*(n+b).*n.*(n+a+b)./((nab.^2).*(nab+1).*(nab-1));
abadd=[mu; B1; B'];
ab=[A' abadd];
```

In particular

- a, b are the α and β exponents of Jacobi weight function (and not the interval extrema!);
- The recurrence coefficients are stored in the matrix `ab`.
- Notice that `r_jacobi` uses the recursive formula

$$\begin{aligned} &\phi_{n+1}(x) = (x - \overline{\alpha}_n)\phi_n(x) - \overline{\beta}_n\phi_{n-1}(x), \;\; k = 1, 2, \ldots \\ &\phi_{-1}(t) = 0, \; \phi_0(t) = 1. \end{aligned} \tag{1}$$

to generate the orthogonal polynomials.

In particular:

1. The first column of `ab` contains the terms $\alpha_0, \ldots, \alpha_{n-1}$,
2. The second of `ab` contains the terms $\beta_0, \ldots, \beta_{n-1}$.

Gaussian rules in Matlab

Next one calls gauss.m as

```
xw=gauss(N,ab)
```

defined as

```
function xw=gauss(N,ab)
N0=size(ab,1); if N0<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
  J(n,n-1)=sqrt(ab(n,2));
  J(n-1,n)=J(n,n-1);
end
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];
```

- This routine, for n equal to the number of rows of `ab` provides a matrix `xw` of dimension $n \times 2$ in which
  - the first column stores the n nodes,
  - the second column stores the n weights,

  of the Gaussian rule relatively to the choosen Jacobi-weight.
- For details see Algorithm 726: ORTHPOL–a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules.

Gaussian rules in Matlab

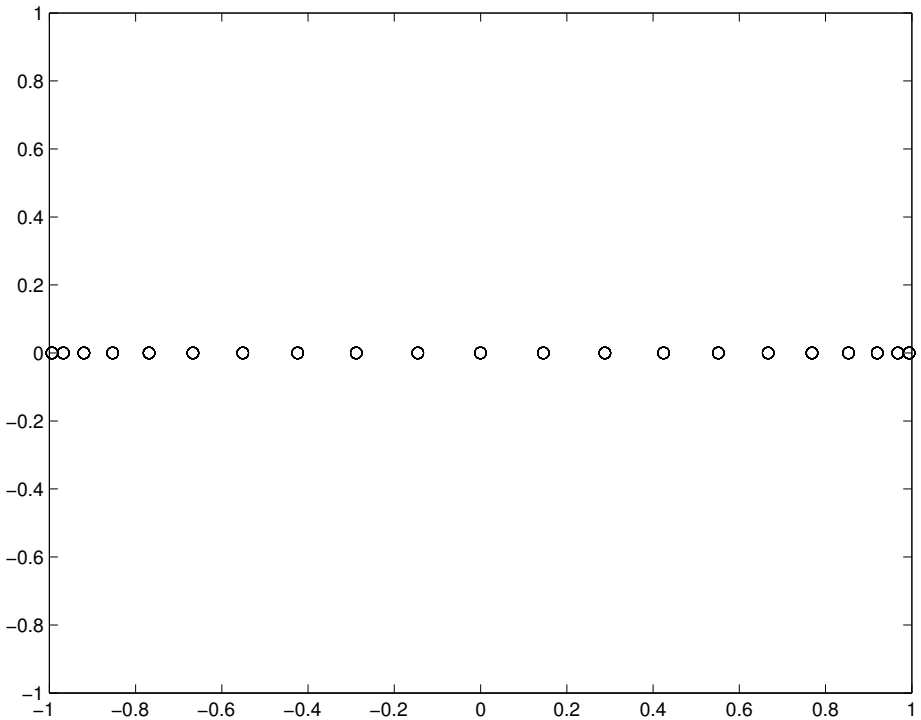

Figure: Distribution of 20 nodes of the Gauss-Legendre rule of degree 39.

.

Gaussian rules in Matlab

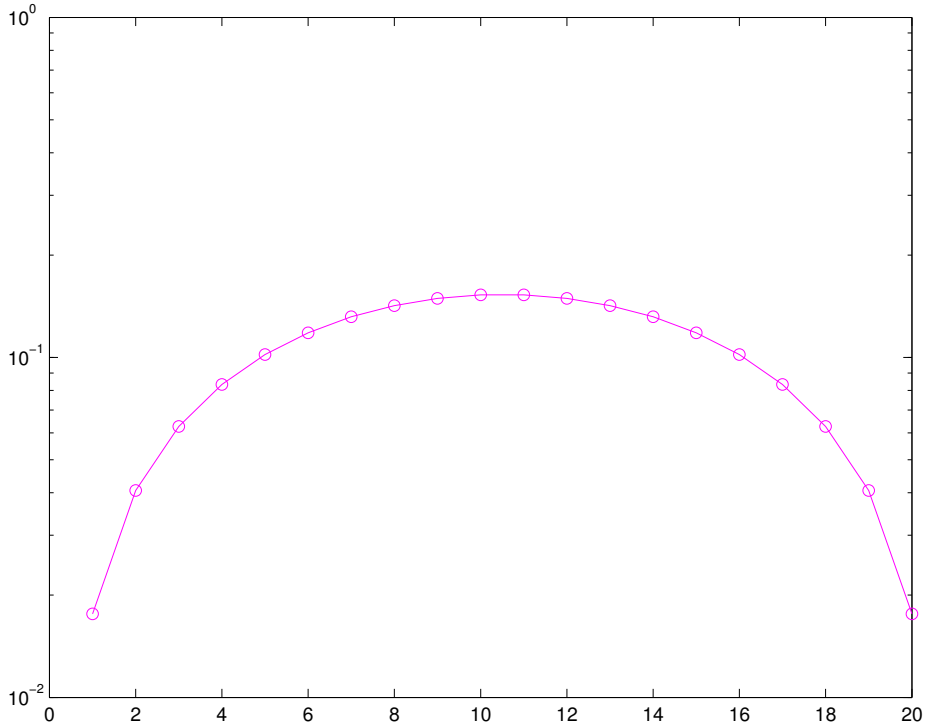

Figure: Values of 20 weights of the Gauss-Legendre rule of degree 39.

.

Gaussian rules in Matlab: Gauss-Jacobi

Now we can write integration_gauss_jacobi.m,

```
function [I,x,w]=integration_gauss_jacobi(N,alpha,beta,f)
% INPUT:
% N: number of nodes.
% alpha,beta: w(x)=((1-x)^alpha)*((1+x)^beta)
% f: function such that the integrand is f(x) w(x) in (-1,1).
% OUTPUT:
% I: approximation of I(f*w,a,b)
% x,w: nodes and weights of the rule.
ab=r_jacobi(N,alpha,beta); % recursive terms
xw=gauss(N,ab); % matrix of nodes and weights
x=xw(:,1); w=xw(:,2);% nodes and weights
fx=feval(f,x); % function evaluation
I=w'*fx; % value of the integral
```

In the case of Gauss-Legendre weight, one can use the formula in a generic bounded interval (a, b) by scaling the formula by scale_formula.m

```
function [x,w]=scale_formula(x,w,a,b)
% INPUT:
% x,w: nodes and weights of Gauss-Legendre rule,
% a,b: integration extrema
% OUTPUT:
% x,w: nodes and weights of the Gauss-Legendre formula in [a,b].

 x=((a+b)/2)+((b-a)/2)*x; % scale nodes [a,b].
 w=((b-a)/2)*w; % scale weights in [a,b].
```

Gaussian rules in Matlab: Gauss-Jacobi

For a variety of test functions taken from Is Gauss Quadrature Better than Clenshaw–Curtis?, we can use f.m defined as

```
function fx=f(x)

example=1;

switch example
case 1
    fx=x.^20;
case 2
     fx=exp(x);
case 3
    fx=exp(-x.^2);
case 4
    fx=1./(1+16*(x.^2));
case 5
    fx=exp(-x.^(-2));
case 6
    fx=abs(x); fx=fx.^3;
case 7
    fx=x.^(0.5);
case 8
    fx=exp(x).*(sqrt(1-x));
end
```

Many useful routines are available at the following homesite.

Remark

We briefly descrive the scaling of Gaussian rules.

- *Let* $\gamma(t) = (a(1-t)/2) + (b(1+t)/2)$,
- *if* $\{t_k\}$ *and* $\{w_k\}$ *are the nodes and weights of Gauss-Legendre rule*

$$\int_a^b f(x)dx = \int_{-1}^{1} \frac{b-a}{2} f(\gamma(t))dt \approx \sum_{k=1}^{n} \frac{b-a}{2} w_k f(\gamma(t_k))$$

and thus $\int_a^b f(x)dx \approx \sum_{k=1}^{n} w_k^* f(x_k^*)$ *with*

$$w_k^* = \frac{b-a}{2} w_k, \quad x_k^* = \gamma(t_k) = (a(1-t_k)/2) + (b(1+t_k)/2).$$

Gaussian rules in Matlab: example 1

As first test we consider

$$\int_{-1}^{1} x^{20}\, dx \approx 0.09523809523809523300$$

obtaining

```
>> format long e;
>> N=11; ajac=0; bjac=0; a=-1; b=1; g=@(x) x.^20;
>> [I_jac,x_jac,w_jac]=integration_gauss_jacobi(N,ajac,bjac,g);
>> I_jac
I_jac =
     9.523809523809568e-02
>> length(x_jac)
ans =
    11
>> (0.09523809523809568-0.09523809523809523300)/0.09523809523809523300 % relative
      error
ans =
     4.662936703425657e-15
>>
```

Remark

From the theory we know that a Gauss-Legendre rule with 11 *nodes integrates exactly* $\int_{-1}^{1} x^{20}\, dx$ *and consequently it is not surprise an error of* 10^{-15}.

Gaussian rules in Matlab: example 2

Let us consider the integral

$$\int_{-1}^{1} \exp(x)\sqrt{1-x}\,dx = 1.779143654691910. \qquad (2)$$

To achieve this result we can use for instance Chebfun suite or Matlab built in `integral`, getting

```
>> format long e;
>> f=@(x) exp(x).*(1-x).^(1/2);
>> F=chebfun(f,'splitting','on'); S=sum(F)
S = 1.779143654691910e+00
>> I=integral(f,-1,1,'AbsTol',10^(-15),'RelTol',10^(-15))
I = 1.779143654691910e+00
```

Gaussian rules in Matlab: example 2

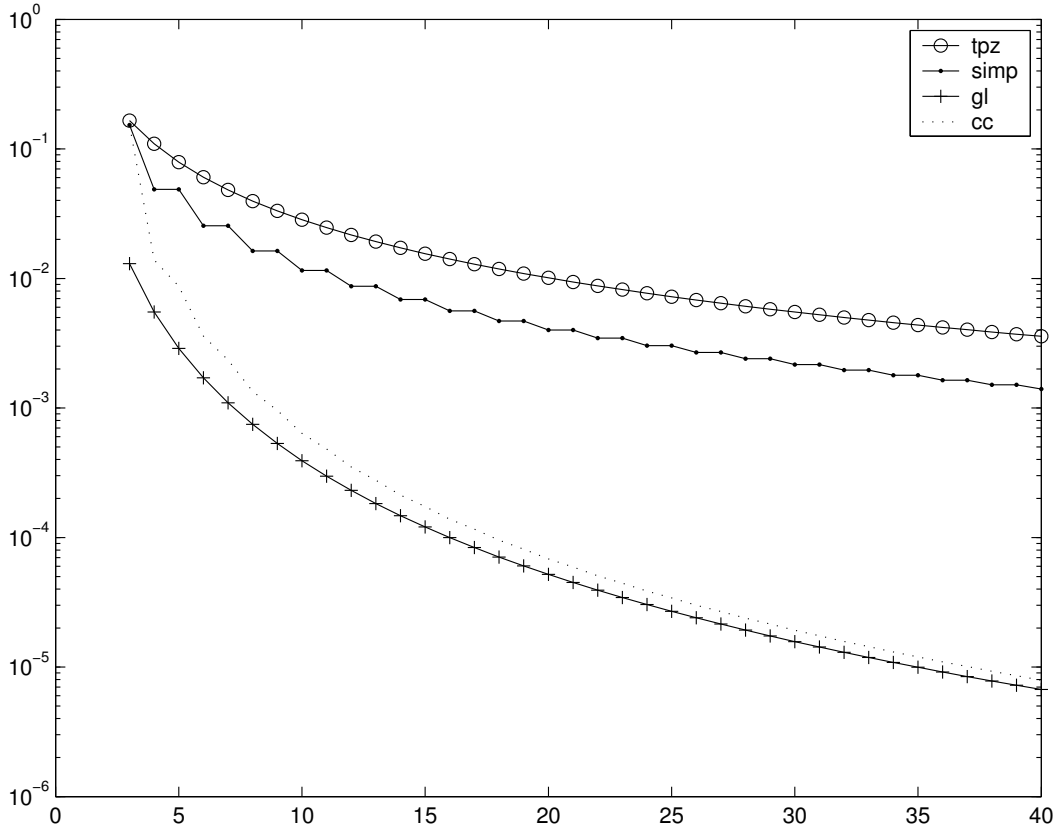

Figure: Plot of the absolute quadrature error of the trapezoidal and Cavalieri-Simpson composite rule composta, Gauss-Legendre and Clenshaw-Curtis formula for computing $\int_{-1}^{1} \exp(x)\sqrt{1-x}\,dx$. In abscissa the number of nodes.

Esempio 2

Now observe that $w(x) = \sqrt{1-x}$ is a Gauss-Jacobi weight

$$w(t) = (1-t)^{\alpha}(1+t)^{\beta}$$

where $\alpha = 1/2$ and $\beta = 0$.

Consequently, if

- $g(x) = \exp(x),\ \ w_1(x) = \sqrt{1-x}$,
- $f(x) = \exp(x),\ \ w_2(x) = 1$,

then

$$\int_{-1}^{1} \exp(x)\sqrt{1-x}dx = \int_{-1}^{1} g(x)w_1(x)dx = \int_{-1}^{1} f(x)w_2(x)dx$$

Thus the integrand can be written as $f \cdot w_1$ where $f(x) = \exp(x)$ and $w_1(x) = \sqrt{1-x}$.

Gaussian rules in Matlab: example 2

```
>> a=-1; b=1;
>> [I_jac,x_jac,w_jac]=integration_gauss_jacobi(10,1/2,0,@exp);
>> fprintf('\n \t [GAUSS-JACOBI]: %15.20f',I_jac);
   [GAUSS-JACOBI]: 1.77914365469190930000
 >>1.77914365469190979259117902999-1.779143654691909300
ans = 4.440892098500626e-016
>> length(x_jac)
ans = 10
>> [I_jac,x_jac,w_jac]=...
 integration_gauss_jacobi(10,0,0,inline('exp(x).*sqrt(1-x)'));
>> fprintf('\n \t [GAUSS-LEGENDRE]: %15.20f',I_jac)
   [GAUSS-LEGENDRE]: 1.77984112101478020000
 >>1.77914365469190979259117902999-1.779841121014780200
ans =-6.9747e-004
>> length(x_jac)
ans =  10
>>
```

The formulas above have the same cardinality, but give different errors

- circa $4.44 \cdot 10^{-16}$ per Gauss-Jacobi con $a = 1/2$, $b = 0$,
- circa $2.52 \cdot 10^{-3}$ per Gauss-Legendre (cioè Gauss-Jacobi con $a = 0$, $b = 0$).

The key is that using the right weight function w, with a regular f, gives better results than using a less regular integrand fw w.r.t. to Gauss-Legendre weight.

Fixed an n-point Gaussian rule, the purpose of Gauss-Kronrod rules is to add $n+1$ points to achieve a rule with ADE equal to $3n+1$ if n is even, $3n+2$ if n is odd (see, e.g. [1, p. 106]).

- The price that one pays is the full rule in general does not preserve for the first n points the weights. The idea is that with few function evaluations, i.e. $2n+1$, one has two rules with different order, both providing an approximation of the integral as well as an error estimate.
- The codes are a complicated modification of the Golub-Welsch approach, and were introduced in Calculation of Gauss-Kronrod quadrature rules by D.P. Laurie.
- Kronrod extensions exist not only for the usual Gauss-Legendre weight but also for the Gauss-Gegenbauer weight function $w(x) = (1-x^2)^\mu$ with $-1/2 \leq \mu \leq 3/2$.

Our purpose is to introduce the routines to run the numerical experiments.
We start by its core, that is the procedure r_kronrod.
D. Laurie comments his code as follows:

- `ab=R_KRONROD(n,ab0)` produces the alpha- and beta-elements in the Jacobi-Kronrod matrix of order $2n+1$ for the weight function (or measure) w.
- The input data for the weight function w are the recurrence coefficients of the associated orthogonal polynomials, which are stored in the array `ab0` of dimension

$$\lceil (3n/2) + 1 \rceil \times 2.$$

- The alpha-elements are stored in the first column, the beta-elements in the second column, of the

$$(2n+1) \times 2$$

array `ab`.

Gauss-Kronrod rules: Matlab implementation

The routine r_kronrod consists in this code.

```
function ab=r_kronrod(N,ab0)

if length(ab0)<ceil(3*N/2)+1, error('array ab0 too short'), end
a=zeros(2*N+1,1); b=a;
k=0:floor(3*N/2); a(k+1)=ab0(k+1,1);
k=0:ceil(3*N/2); b(k+1)=ab0(k+1,2);
s=zeros(floor(N/2)+2,1); t=s; t(2)=b(N+2);
for m=0:N-2,
  k=floor((m+1)/2):-1:0; l=m-k;
  s(k+2)=cumsum((a(k+N+2)-a(l+1)).*t(k+2)+b(k+N+2).*s(k+1)-b(l+1).*s(k+2));
  swap=s; s=t; t=swap;
end
j=floor(N/2):-1:0; s(j+2)=s(j+1);
for m=N-1:2*N-3,
  k=m+1-N:floor((m-1)/2); l=m-k; j=N-1-l;
  s(j+2)=cumsum(-(a(k+N+2)-a(l+1)).*t(j+2)-b(k+N+2).*s(j+2)+b(l+1).*s(j+3));
  j=j(length(j)); k=floor((m+1)/2);
  if rem(m,2)==0, a(k+N+2)=a(k+1)+(s(j+2)-b(k+N+2)*s(j+3))/t(j+3);
  else b(k+N+2)=s(j+2)/s(j+3);
  end
  swap=s; s=t; t=swap;
end
a(2*N+1)=a(N)-b(2*N+1)*s(2)/t(2);
ab=[a b];
```

Gauss-Kronrod rules

The previous routine is used by kronrod to provide the desired rule. D. Laurie provides these comment to the routine.

- `xw=KRONROD(n,ab)` generates the $(2n+1)$-point Gauss-Kronrod quadrature rule for the weight function w encoded by the recurrence matrix ab of order

$$\lceil (3n/2) + 1 \rceil \times 2$$

containing in its first and second column respectively the alpha- and beta-coefficients in the three-term recurrence relation for w.

- The $2n+1$ nodes, in increasing order, are output into the first column, the corresponding weights into the second column, of the

$$(2n+1) \times 2$$

array `xw`.

Gauss-Kronrod rules

Below you find the Matlab code of kronrod.

```
function xw=kronrod(N,ab)

ab0=r_kronrod(N,ab);
if(sum((ab0(:,2)>0))~=2*N+1) error('Gauss-Kronrod does not exist'), end
J=zeros(2*N+1);
for k=1:2*N
    J(k,k)=ab0(k,1);
    J(k,k+1)=sqrt(ab0(k+1,2));
    J(k+1,k)=J(k,k+1);
end
J(2*N+1,2*N+1)=ab0(2*N+1,1);
[V,D]=eig(J);
d=diag(D);
e=ab0(1,2).*(V(1,:).^2);
[x,i]=sort(d);
w=e(i)';
xw=[x w];
```

Gauss-Kronrod rules: demo

To see how these rules work we consider the demo demo_kronrod.

```
function demo_kronrod(f,alpha,beta,N)

% Integration of a function 'f' in [-1,1] and Gauss-Kronrod rules.
if nargin < 1, f=@(x) exp(x).*sqrt(1-x.^2); end
if nargin < 3, alpha=0; beta=0; end % Jacobi weight
if nargin < 4, N=10; end

dim=ceil(3*N/2)+1; % length Kronrod rule
ab=r_jacobi(dim,alpha,beta); % Gauss-Legendre recurrence (as required by kronrod)
xwg=gauss(N,ab); xwk=kronrod(N,ab);

g=@(x) f(x).*(1-x).^alpha.*(1+x).^beta;
I=integral(g,-1,1,'AbsTol',10^(-15),'RelTol',10^(-15));

fk=feval(f,xwk(:,1)); % this evaluation includes that at Gauss-Legendre nodes!

Sg=(xwg(:,2))'*fk(2:2:end);
Sk=(xwk(:,2))'*fk;

fprintf('\n * Gauss-Legendre rule \n \n'); xwg
fprintf('\n * Gauss-Legendre-Kronrod rule \n \n'); xwk
fprintf('\n \t I gauss : %1.15e',Sg);
fprintf('\n \t I kronr : %1.15e',Sk);
fprintf('\n \t I exact : %1.15e',I);
fprintf('\n \n');
fprintf('\n \t E gauss : %1.3e',abs(I-Sg));
fprintf('\n \t E estim : %1.3e',abs(Sk-Sg));
fprintf('\n \n');
```

Gauss-Kronrod rules: demo

We run this first experiment.

```
>> clear all
>> f=@(x) exp(x).*sqrt(1-x.^2); alpha=0; beta=0; N=5;
>> demo_kronrod(f,alpha,beta,N);

 * Gauss-Legendre rule
xwg =
   -9.0618e-01   2.3693e-01
   -5.3847e-01   4.7863e-01
    4.1434e-17   5.6889e-01
    5.3847e-01   4.7863e-01
    9.0618e-01   2.3693e-01

 * Gauss-Legendre-Kronrod rule
xwk =
   -9.8409e-01   4.2582e-02
   -9.0618e-01   1.1523e-01
   -7.5417e-01   1.8680e-01
   -5.3847e-01   2.4104e-01
   -2.7963e-01   2.7285e-01
   -1.9783e-16   2.8299e-01
    2.7963e-01   2.7285e-01
    5.3847e-01   2.4104e-01
    7.5417e-01   1.8680e-01
    9.0618e-01   1.1523e-01
    9.8409e-01   4.2582e-02

         I gauss : 1.783762504838484e+00
         I kronr : 1.775930588360792e+00
         I exact : 1.775499689212181e+00

         E gauss : 8.263e-03
         E estim : 7.832e-03
```

We observe that

- nodes of Gauss-Legendre rule are nested in those of Gauss-Kronrod rule, though the weights are different;
- the error estimate is quite good though the error is large;
- one can use a better Gegenbauer weight function that is

$$(1-x^2)^{1/2} = (1-x)^{1/2}(1+x)^{1/2}, \quad x \in (-1,1).$$

  and so one intends to make experiments with this new weight function, setting the Jacobi exponents as $\alpha = 1/2$, $\beta = 1/2$ and $f(x) = \exp(x)$.
- it is proven that this weight function admits Kronrod extension.

Gauss-Kronrod rules: demo

Thus we run this second experiment.

```
>> clear all
>> f=@(x) exp(x); alpha=1/2; beta=1/2; N=5;
>> demo_kronrod(f,alpha,beta,N);

 * Gauss-Legendre rule
xwg =
   -8.6603e-01   1.3090e-01
   -5.0000e-01   3.9270e-01
   -5.1486e-17   5.2360e-01
    5.0000e-01   3.9270e-01
    8.6603e-01   1.3090e-01

 * Gauss-Legendre-Kronrod rule
xwk =
   -9.6593e-01   1.7537e-02
   -8.6603e-01   6.5450e-02
   -7.0711e-01   1.3090e-01
   -5.0000e-01   1.9635e-01
   -2.5882e-01   2.4426e-01
   -3.4235e-17   2.6180e-01
    2.5882e-01   2.4426e-01
    5.0000e-01   1.9635e-01
    7.0711e-01   1.3090e-01
    8.6603e-01   6.5450e-02
    9.6593e-01   1.7537e-02

         I gauss : 1.775499688781380e+00
         I kronr : 1.775499689212182e+00
         I exact : 1.775499689212181e+00

         E gauss : 4.308e-10
         E estim : 4.308e-10
```

Gauss-Kronrod rules: demo

We observe that

- nodes of the Gauss-Jacobi rule are nested in those of Gauss-Jacobi-Kronrod rule, though the weights are different;
- the error estimate is quite good but this time the error is rather small (due to the good choice of the weight function).

Anti-Gaussian rules

An alternative to Gauss-Kronrod rules are the so called anti-Gaussian rules introduced in Anti-Gaussian quadrature formulas by D. Laurie.

- An anti-Gaussian quadrature formula is an $(n+1)$-point formula of degree $2n-1$ which integrates polynomials of degree up to $2n+1$ with an error equal in magnitude but of opposite sign to that of the n-point Gaussian formula.
- In the algorithm,
  1. by `r_jacobi` one determines N+1 recursive coefficients $\boldsymbol{\alpha} = (\alpha_k)_{k=1,\ldots,N+1}$, $\boldsymbol{\beta} = (\beta_k)_{k=1,\ldots,N+1}$ w.r.t. the weight w;
  2. sets $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}$,
  3. sets $\boldsymbol{\beta}^*_k = \boldsymbol{\beta}_k$ for $k = 1, \ldots, N$, $\boldsymbol{\beta}^*_{N+1} = 2\boldsymbol{\beta}_{N+1}$;
  4. computes the Gaussian rule S_N^G with N nodes as well as the anti-Gaussian S_N^{AG} rule with $N+1$ nodes;
  5. the error $I(f) - S_N^G(f)$ is equal to $-(I(f) - S_N^{AG}(f))$ and thus, easily,
$$I(f) - S_N^G(f) = (S_N^{AG}(f) - S_N^G(f))/2.$$

Anti-Gaussian rules in Matlab

```
function demo_antigaussian(f,alpha,beta,N)

% Integration of a function 'f' in [-1,1] and Gauss-Kronrod rules.
if nargin < 1, f=@(x) exp(x); end
if nargin < 3, alpha=0; beta=0; end % Jacobi weight
if nargin < 4, N=3; end

dim=N+1; % length Kronrod rule
ab=r_jacobi(dim,alpha,beta); % Gauss-Legendre recurrence (as required by kronrod)
xwg=gauss(N,ab(1:N,:));
ab_agss=ab; ab_agss(end,2)=2*ab_agss(end,2);
xwag=gauss(N+1,ab_agss);

g=@(x) f(x).*(1-x).^alpha.*(1+x).^beta;
I=integral(g,-1,1,'AbsTol',10^(-15),'RelTol',10^(-15));

fg=feval(f,xwg(:,1)); % this evaluation includes that at Gauss-Legendre nodes!
Sg=(xwg(:,2))'*fg;

fg=feval(f,xwag(:,1)); % this evaluation includes that at anti-gauss nodes!
Sag=(xwag(:,2))'*fg;

fprintf('\n * Gauss-Legendre rule \n \n'); xwg
fprintf('\n * Gauss-Legendre-anti rule \n \n'); xwag
fprintf('\n \t I gauss     : %1.15e',Sg);
fprintf('\n \t I agaus     : %1.15e',Sag);
fprintf('\n \t I exact     : %1.15e',I);
fprintf('\n \n');
fprintf('\n \t I-Sg        : %1.3e',(I-Sg));
fprintf('\n \t I-Sag       : %1.3e',(I-Sag));
fprintf('\n \t (Sg+Sag)/2  : %1.3e',(Sag-Sg)/2);
fprintf('\n \n');
```

Anti-Gaussian rules in Matlab

Here, we test the anti-Gaussian rules with $N+1$ points on $f(x) = \exp(x)$, that though it is not a polynomial of degree $2N-1$ allows realiable error estimates (why does it happen?).

```
>> demo_antigaussian

 * Gauss-Legendre rule
xwg =
   -0.7746    0.5556
   -0.0000    0.8889
    0.7746    0.5556

 * Gauss-Legendre-anti rule
xwag =
   -0.9643    0.1998
   -0.4294    0.8002
    0.4294    0.8002
    0.9643    0.1998

   I gauss     : 2.350336928680012e+00
   I agaus     : 2.350467853389318e+00
   I exact     : 2.350402387287603e+00

   I-Sg        : 6.546e-05
   I-Sag       : -6.547e-05
   (Sg+Sag)/2  : 6.546e-05

>>
```

Bibliography


P.J. Davis and P. Rabinowitz, Methods of Numerical Integration, Dover 1984.


W. Gautschi, Orthogonal polynomials: applications and computation, Acta Numerica (1996), pp.45–119.