Numerical integration: Newton-Cotes and Gauß quadrature formulas

Anne Kværnø, André Massing

Sep 14, 2021

1 Newton-Cotes formulas

We have already seen that *given* $n+1$ distinct but otherwise arbitrary quadrature nodes $\{x_i\}_{i=0}^n \subset [a,b]$, we can construct a quadrature rule $\mathrm{Q}[\cdot](\{x_i\}_{i=0}^{n+1}, \{w_i\}_{i=0}^{n+1})$ based on polynomial interpolation which has degree of exactness equals to n.

An classical example was the trapezoidal rule, which are based on the two quadrature points $x_0 = a$ and $x_1 = b$ and which has degree of exactness equal to 1.

The trapezoidal is the simplest example of a quadrature formula which belongs to the so-called **Newton Cotes formulas**.

By definition, **Newton-Cotes formulas** are quadrature rules which are based on **equidistributed nodes** $\{x_i\}_{i=0}^n \subset [a,b]$ and have degree of exactness equals to n.

The simplest choices here — the *closed* Newton-Cotes methods — use the nodes $x_i = a + ih$ with $h = (b-a)/n$. Examples of these are the Trapezoidal rule and Simpson's rule. The main appeal of these rules is the simple definition of the nodes.

If n is odd, the Newton-Cotes method with $n+1$ nodes has degree of precision n; if n is even, it has degree of precision $n+1$. The corresponding convergence order for the composite rule is, as for all such rules, one larger than the degree of precision, provided that the function f is sufficiently smooth.

However, for $n \geq 8$ negative weights begin to appear in the definitions. Note that for a positive function $f(x) \geqslant 0$ we have that the integral $I[f](a,b) \geqslant 0$ But for a quadrature rule with negative weights *we have not necessarily that* $Q[f](a,b) \geqslant 0$*!* This has the undesired effect that the numerical integral of a positive function can be negative. For instance, f could represent a mass distribution, and $I[f]$ is then the total mass. If you use quadrature rules with negative weights, it might happen that you obtain a negative mass, which is completely unphysical.

In addition, negative weights can lead to cancellation errors in the numerical evaluation, which may result in a lower practical accuracy. Since the rules with $n = 6$ and $n = 7$ yield the same convergence order, this mean that it is mostly the rules with $n \leq 6$ that are used in practice.

The *open* Newton-Cotes methods, in contrast, use the nodes $x_i = a + (i + 1/2)h$ with $h = (b-a)/(n+1)$. The simplest example here is the midpoint rule. Here negative weights appear already for $n \geq 2$. Thus the midpoint rule is the only such rule that is commonly used in applications.

2 Gauß quadrature

In the first lecture on quadrature rules, we compared the trapezoidal rule with Gauß-Legendre quadrature rule which are both based on two quadrature points and observed that

- the Gauß-Legendre quadrature was much more accurate than the trapezoidal rule, and more specifically,
- the Gauß-Legendre quadrature *has degree of exactness equal to* 3 *and not only* 1.

Question.

- Is there a general approach to construct quadrature rules $Q[\cdot](\{x_i\}_{i=0}^n, \{w_i\}_{i=0}^n)$ based on $n+1$ nodes with a degree of exactness $> n$?
- What is the maximal degree of exactness we can achieve?

Observation/Intuition:

- If we predefine $n+1$ distinct quadrature points in some way, e.g. by choosing equi-distributed quadrature points, then we have only $n+1$ *free* parameters left, namely the weights $\{w_i\}_{i=0}^n$. In the interpolation-based construction of quadrature rules, those weights are then determined in such a way that we achieve the highest degree of exactness possible for a *predefined, given* set of quadrature points. By setting $w_i := I[l_i], i = 0, \dots n$ where l_i are the Lagrange/cardinal function associated with $\{x_i\}_{i=0}^n$, we ensure that the resulting quadrature rule has a degree of exactness of n, *no matter how we chose the* $n+1$ *distinct quadrature points*!
- But if we don't predefine the quadrature nodes, we have $2n+2$ *free parameters* ($n+1$ nodes and $n+1$ weights) to play with. So we might hope that with $2n+2$ free parameters, we can construct quadrature rules which are exact for $p \in \mathbb{P}_{2n+1}$, as such polynomials are determined by at most $2n+1$ coefficients.

That leads us to the following definition.

Definition 2.1. *Gaussian quadrature.*

A quadrature rule $Q[\cdot](\{x_i\}_{i=0}^n, \{w_i\}_{i=0}^n)$ based on $n+1$ nodes which has degree of exactness equals to $2n+1$ is called a **Gaussian (Legendre) quadrature** (GQ).

With this definition, both the Gauß-Legendre with $n = 2$ quadrature points and the the midpoint rule are Gaussian quadrature rules.

Notice.

We only need to find somehow the *quadrature points* for a Gaussian quadrature. If the quadrature points are known, then the weights are again determined by $w_i = I[l_i]$, since every other choice of the weights will lead to a quadrature rule with has a degree of exactness $< n$!

How can we find these "mystical quadrature points" to construct a Gaussian quadrature rule? For that we need to briefly review the concept of orthogonality for functions, which you already encountered in Calculus 3. Afterwards we take a short dive into the theory of orthogonal polynomials. We take the following constructive approach. By assuming that there exist Gaussian quadratures rule, we will derive certain properties which uniquely characterizes their quadrature points and weights. Then we turn the tables and show that if we define a quadrature rule with weights and points satisfying those characteristic properties, then it must be a Gaussian quadrature rule.

2.1 Orthogonal polynomials

Two functions $f, g : [a, b] \to \mathbb{R}$ are orthogonal if

$$\langle f, g \rangle := \int_a^b f(x)g(x)\,\mathrm{d}x = 0.$$

Usually, it will be clear from the context which interval $[a, b]$ we picked.

Next, we find some clues regarding the properties of the sought quadrature points.

Lemma 2.1. *(From Gaussian quadratures to orthogonality).*

Assume that a given quadrature rule $Q[\cdot](\{x_i, w_i\}_{i=0}^n)$ on $I = [a, b]$ with $n + 1$ points is a Gaussian quadrature rule. Let

$$\omega_{n+1} = \Pi_{i=0}^n (x - x_i)$$

the $n+1$st Newton polynomial associated with the $n+1$ quadrature points x_i. Then $\omega_{n+1}(x)$ is orthogonal on $\mathbb{P}^n(I)$; that is, :

$$\int_a^b p(x)\omega_{n+1}(x)dx = 0 \quad \text{for } p \in \mathbb{P}^n(I). \tag{1}$$

Proof. By definition, a Gaussian quadrature rule with $n+1$ quadrature points has a degree of exactness of $2n + 1$. On the other hand $p(x)\omega_{n+1}(x) \in \mathbb{P}^{2n+1}(I)$ for $p \in \mathbb{P}^n(I)$. Consequently,

$$\int_a^b p(x)\omega_{n+1}(x)\,\mathrm{d}x = Q[p(x)\omega_{n+1}(x)] = \sum_{i=0}^n p(x_i)\omega_{n+1}(x_i) = 0 \tag{2}$$

since by the definition of $\omega_{n+1}(x)$ we have that $\omega_{n+1}(x_i) = 0$ for $i = 0, \ldots n$.

Note that $\omega_{n+1}(x) = c_0 + c_1 x + \ldots c_{n+1}x^{n+1}$ is the unique polynomial of degree $n+1$ with roots given by $\{x_i\}_{i=0}^n$ and a leading coefficient $c_{n+1} = 1$.

Next, we show the converse of the previous lemma.

Lemma 2.2. *(From orthogonality to Gaussian quadratures).*

Let $\{x_i\}_{i=0}^n \subset [a, b]$ be $n + 1$ distinct points and assume that the corresponding (Newton) polynomial $\omega_{n+1}(x) = \Pi_{i=0}^n (x - x_i)$ satisfies the orthogonality property (1). Then the corresponding interpolation based quadrature rule $Q[\cdot](\{x_i, w_i\}_{i=0}^n)$ with weights $w_i = \int_a^b l_i(x)\,\mathrm{d}x$ is a Gaussian quadrature rule, i.e. its degree of exactness is $2n + 1$ (and not only n).

Proof. Defining the weights by $w_i = \int_a^b l_i(x)\,\mathrm{d}x$ for any given quadrature points $\{x_i\}_{i=0}^n$ guarantess that the resulting quadrature the degree of exactness is at least n. We need to show that the additional orthogonality property implies that we in fact can obtain a degree of exactness of $2n+1$; that is, $Q[p] = I[p]$ all $p \in \mathbb{P}^{2n+1}(I)$.

Let $p_n \in \mathbb{P}^n(I)$ be the interpolation polynomial of degree n which satisfies $p(x_i) = p_n(x_i)$ for $i = 0, \ldots n$. Then the polynomial $q(x) := p(x) - p_n(x)$ is of degree $2n + 1$ and satisfies $q(x_i) = 0$ for $i = 1, \ldots, n$. Thus $\{x_i\}_{i=0}^n$ are roots of $q(x)$, and thus we can write $p(x) - p_n(x) = q(x) = \omega_{n+1}(x)r(x)$ for some $r \in \mathbb{P}^n(I)$. Thus

$$\begin{aligned}
\int_a^b p(x) &= \int_a^b \big(\omega_{n+1}(x)r(x) + p_n(x)\big)\,\mathrm{d}x && (3)\\
&= \int_a^b \omega_{n+1}(x)r(x)\,\mathrm{d}x + \int_a^b p_n(x)\,\mathrm{d}x && (4)\\
&= \int_a^b p_n(x)\,\mathrm{d}x && \text{(Orthogonality property of } \omega_{n+1}) && (5)\\
&= Q[p_n] && (Q[\cdot] \text{ has a degree of exactness } = n) && (6)\\
&= Q[p_n] + Q[\omega_{n+1}r] && (\text{since } \omega_{n+1}(x_i) = 0, i = 0, \ldots, n) && (7)\\
&= Q[\underbrace{p_n + \omega_{n+1}r}_{=p}] = Q[p]. && && (8)
\end{aligned}$$

Intermediate summary.

The last two lemma reveal that the location of the quadrature points for a Gaussian rule is closely linked to the orthogonality of certain polynomials. The next 2 steps are now to

- construct such orthogonality polynomials p_{n+1} for $n = 0, 1, 2, 3 \ldots$, and
- to show that they always have $n+1$ **distinct real roots** $\{x_i\}_{i=0}^n$ **residing** in (a, b).

So these $n+1$ roots will be then the quadrature points to our Gaussian quadrature rule.

The next lemma shows that on a given interval and any given degree k, we can always construct a polynom $p_k \in \mathbb{P}^k(I)$ which is orthogonal to all $p \in \mathbb{P}^{k-1}(I)$.

Lemma 2.3. *Orthogonal polynomials on* $[a, b]$.

There is a sequence of $\{p_k\}_{k=1}^\infty$ of polynomials satisfying

$$p_0(x) = 1, \tag{9}$$
$$p_k(x) = x^k + r_{k-1}(x) \quad \text{for } k = 1, 2, \ldots \tag{10}$$

with $r_{k-1} \in \mathbb{P}_{k-1}$ and satisfying the *orthogonality property*

$$\langle p_k, p_l \rangle = \int_a^b p_k(x) p_l(x) dx = 0 \quad \text{for } k \neq l, \tag{11}$$

and that every polynomial $q_n \in \mathbb{P}_n$ can be written as a linear combination of those orthogonal polynomials up to order n. In other words

$$\mathbb{P}_n = \text{Span}\{p_0, \ldots, p_n\}$$

Proof. We start from the sequence $\{\phi_k\}_{k=0}^\infty$ of monomials $\phi_k(x) = x^k$ and apply the Gram-Schmidt orthogonalization procedure:

$$\begin{aligned}
\widetilde{p}_0 &:= 1 \\
\widetilde{p}_1 &:= \phi_1 - \frac{\langle \phi_1, \widetilde{p}_0 \rangle}{\|\widetilde{p}_0\|^2} \widetilde{p}_0 \\
\widetilde{p}_2 &:= \phi_2 - \frac{\langle \phi_2, \widetilde{p}_0 \rangle}{\|\widetilde{p}_0\|^2} \widetilde{p}_0 - \frac{\langle \phi_2, \widetilde{p}_1 \rangle}{\|\widetilde{p}_1\|^2} \widetilde{p}_1 \\
&\ldots \\
\widetilde{p}_k &= \phi_k - \sum_{j=0}^{k-1} \frac{\langle \phi_k, \widetilde{p}_j \rangle}{\|\widetilde{p}_j\|^2} \widetilde{p}_j
\end{aligned}$$

By construction, $\widetilde{p}_n \in \mathbb{P}_n$ and $\langle p_k, p_l \rangle = 0$ for $k \neq l$. Since $\widetilde{p}_k(x) = a_k x^k + a_{k-1} x^{k-1} + \ldots a_0$, we simply define $p_k(x) = \widetilde{p}_k / a_k$ to satisfy (10).

Lemma 2.4. *Roots of orthogonal polynomials.*

Each of the polynomials p_k defined in Theorem 2.3 has k **distinct real roots**.

Proof. Take $k > 0$ and denote by $x_0, x_1, \ldots, x_{l-1} \in (a, b)$ the l points at which p_k switches sign in $[a, b]$. Clearly, these are roots for the polynomial p_k. The polynomial $q_l(x) = \Pi_{i=0}^{l-1}(x - x_i)$ switches the sign at exactly the same points as p_k. Thus the polynomial $p_k(x) q_l(x)$ must be either strictly positive or strictly negative except for the points $x_0, \ldots, x_{l-1}$ where it is 0. Consequently, $\int_a^b p_k(x) q_l(x)\, \mathrm{d}x$ is either > 0 or < 0. But if $l < k$, this contradicts the orthogonality property for for p_k, namely that $\int_a^b p_k(x) q_l(x)\, \mathrm{d}x = 0$ for all q_l with a degree $l < k$. Thus $k = l$.

All the previous lemma can be now summarized in the final statement regarding the construction of Gaussian quadrature rules.

Theorem 2.1. *Construction of Gaussian quadrature with $n+1$ nodes.*

For every $n \in \mathbb{N}$, there exists an orthogonal polynomial $p_{n+1} \in \mathbb{P}_{n+1}$ on $[a,b]$ satisfying

$$\langle p_{n+1}, q \rangle = 0 \quad \forall\, q \in \mathbb{P}_n.$$

The polynomial p_{n+1} has $n+1$ distinct and real roots $\{x_i\}_{i=0}^n$ all residing in (a,b). Define the weights $\{w_i\}_{i=0}^n$ by

$$w_i = \int_a^b \ell_i(x)\,\mathrm{d}x.$$

where $\{\ell_i\}_{i=0}^n$ are the $n+1$ cardinal functions associated with $\{x_i\}_{i=0}^n$. Then the resulting quadrature rule $Q[\cdot](\{x_i, \omega_i\}_{i=0}^n)$ is a Gaussian quadrature.

Proof. This is just a summary of 4 the previous lemma.

Recipe 1 to construct a Gaussian quadrature with $n+1$ nodes.

To construct a Gaussian formula on $[a,b]$ based on $n+1$ nodes you proceed as follows

1. Construct a polynomial $p_{n+1} \in \mathbb{P}_{n+1}(a,b)$ such that $\displaystyle\int_a^b p_{n+1}(x)q(x)\,\mathrm{d}x \quad \forall\, q \in \mathbb{P}_n$. You can start from the monomials $\{1, x, x^2, \ldots, x^{n+1}\}$ and use Gram-Schmidt to orthogonalize them.
2. Determine the $n+1$ **real** roots $\{x_i\}_{i=0}^n$ of p_{n+1} which serve then as quadrature nodes.
3. Calculate the cardinal functions $\ell_i(x)$ associated with $n+1$ nodes $\{x_i\}_{i=0}^n$ and then the weights are given by $w_i = \displaystyle\int_a^b \ell_i(x)\,\mathrm{d}x$.

This is the recipe you are asked to use in Exercise set 10. Alternatively one can start from a reference interval, leading to

Recipe 2 to construct a Gaussian quadrature with $n+1$ nodes.

To construct a Gaussian formula on $[a,b]$ based on $n+1$ nodes you proceed as follows

1. Construct a polynomial $p_{n+1} \in \mathbb{P}_{n+1}(-1,1)$ which satisfies $\displaystyle\int_{-1}^1 p_{n+1}(x)q(x)\,\mathrm{d}x \quad \forall\, q \in \mathbb{P}_n$.
2. You determine the $n+1$ **real** roots $\{x_i\}_{i=0}^n$ of p_{n+1} which serve then as quadrature nodes.
3. Calculate the cardinal functions $\ell_i(x)$ associated with $n+1$ nodes $\{x_i\}_{i=0}^n$ and then the weights are given by $w_i = \displaystyle\int_a^b \ell_i(x)\,\mathrm{d}x$.
4. Finally, transform the resulting Gauß quadrature formula to the desired interval $[a,b]$ via $x_i = \dfrac{b-a}{2}\widehat{x}_i + \dfrac{b+a}{2}, \quad w_i = \dfrac{b-a}{2}\widehat{w}_i \quad \text{for } i = 0, \ldots n.$

3 Two examples revisited

3.1 Example: Revisiting the midpoint rule

The midpoint rule is in fact the Gauß-Legendre rule with 1 node x_0, which explains why it is has a degree of exactness to $2n+2 = 2\cdot 0 + 1 = 1$ (and not 0 like the left and right endpoint rules).

Let's consider $[a, b] = [0, 1]$. We need to find one quadrature point, which is the simple root of an orthogonal linear polynomial p_1. To construct this polynomial, we start from the first two monomials $\phi_0(x) = 1$ and $\phi(x) = x$. Using the Gram-Schmidt algorithm as outlined above, we set

$$p_0 = \phi_0 = 1 \tag{12}$$

$$p_1 = \phi_1 - \frac{\langle \phi_1, p_0 \rangle}{\|p_0\|^2} p_0 = x - \frac{\int_0^1 x \cdot 1 \, \mathrm{d}x}{\int_0^1 1 \cdot 1 \, \mathrm{d}x} = x - \frac{0.5}{1} \tag{13}$$

Now $p_1(x)$ has the root $x_0 = 0.5$ which defines our single quadrature point. In this case, the cardinal function $l_0(x) = 1$ and thus the weight w_0 corresponding to $x_0 = 0.5$ is $w_0 = \int_0^1 l_0(x) \, \mathrm{d}x = 1$.

3.2 Example: Revisiting Gauß-Legendre quadrature with 2 nodes

We will now derive the Gauß-Legendre quadrature with 2 nodes we encountered in the previous lectures, again for $[a, b] = [0, 1]$. Today we will calculate everything manually first, and then later do it programmatically in Python.

The 2 nodes x_0, x_1 we are looking for at the 2 roots for the quadratic polynomial $p_2(x)$ satisfying $\int_0^1 p_2(x) q_(x) \, \mathrm{d}x = 0$ for all $q \in \mathbb{P}^1(I)$. Again, we start from the monomials $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$. To compute the first two orthogonal polynomials p_0 and p_1, we perform exactly the same calculations as in the previous example, and add a third Gram-Schmidt step leading to

$$p_0 = \phi_0 = 1 \tag{14}$$

$$p_1 = \phi_1 - \frac{\langle \phi_1, p_0 \rangle}{\|p_0\|^2} p_0 = x - \frac{\int_0^1 x \cdot 1 \, \mathrm{d}x}{\int_0^1 1 \cdot 1 \, \mathrm{d}x} = x - 0.5 \tag{15}$$

$$p_2 = \phi_2 - \frac{\langle \phi_2, p_0 \rangle}{\|p_0\|^2} p_0 - \frac{\langle \phi_2, p_1 \rangle}{\|p_1\|^2} p_1 \tag{16}$$

$$= x^2 - \frac{\int_0^1 x^2 \cdot 1 \, \mathrm{d}x}{\int_0^1 1 \cdot 1 \, \mathrm{d}x} - \frac{\int_0^1 x^2 \cdot (x - 0.5) \, \mathrm{d}x}{\int_0^1 (x - 0.5) \cdot (x - 0.5) \, \mathrm{d}x} (x - 0.5) \tag{17}$$

Computing the various integrals gives us

$$\int_0^1 x^2 \cdot 1 \, \mathrm{d}x = \frac{1}{3}, \qquad \int_0^1 1 \cdot 1 \, \mathrm{d}x = 1 \tag{18}$$

$$\int_0^1 x^2 \cdot (x - 0.5) \, \mathrm{d}x = \frac{1}{4} - \frac{1}{6} = \frac{1}{12}, \qquad \int_0^1 (x - 0.5) \cdot (x - 0.5) \, \mathrm{d}x = \frac{1}{12} \tag{19}$$

and thus

$$p_2(x) = x^2 - \frac{1}{3} - \frac{12}{12}(x - \frac{1}{2}) = x^2 - x + \frac{1}{6}. \tag{20}$$

Now a general 2nd order polynomial $p_2(x) = x^2 + px + q$ has the 2 roots

$$x_0 = -\frac{p}{2} - \sqrt{\left(\frac{p}{2}\right)^2 - q} = \frac{1}{2} - \sqrt{\frac{1}{12}} = \frac{1}{2} - \frac{\sqrt{3}}{6} \tag{21}$$

$$x_1 = -\frac{p}{2} + \sqrt{\left(\frac{p}{2}\right)^2 - q} = \frac{1}{2} + \frac{\sqrt{3}}{6}. \tag{22}$$

Next, to compute the final weights, we need the cardinal functions

$$l_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad l_1(x) = \frac{x - x_0}{x_1 - x_0} \tag{23}$$

Now inserting the found values x_0 and x_1 shows (after some simple computations) that $w_0 = \int_1^1 l_0(x) \, \mathrm{d}x = 0.5$ and $w_1 = \int_0^1 l_1(x) \, \mathrm{d}x = 0.5$.

Let's redo these computations in form of a little Python program which can be easily generalized to at up to 5 nodes. We will use `sympy` Python module for symbolic calculations quite a bit. We start with the snippets

```
from sympy.abc import x # Denote our integration variable x
from sympy import integrate
```

Spend a minute and have look at integrate submodule.

First we construct the first 3 orthogonal polynomials (order 0, 1, 2) on $[0, 1]$. Spend 2 minutes to understand the code below:

```
# Interval
a, b = 0, 1

# Define scalar product:
def scp(p,q):
    return integrate(p*q, (x, a, b))

# Define monomials up to order 2
mono = lambda x,m: x**m
def mono(x,m):
    return x**m

phis = [ mono(x,m) for m in range(6)]
print(phis)
```

Construct orthogonal polynomials (not normalized)

```
ps = []
for phi in phis:
    ps.append(phi)
    for p in ps[:-1]:
        ps[-1] = ps[-1] - scp(p, ps[-1])/scp(p, p)*p

print("ps")
print(ps)
```

Now write a code snippet to check whether they are actually orthogonal.

```
for p in ps:
    for q in ps:
        int_p_q = scp(p,q)
        print("int_p_q = {}".format(int_p_q))
```

Compute the roots of the second order polynomial. Of course you can do it by hand but le'ts us `sympy` for it. Spend a minute a have a look at solve submodule.

```
from sympy.solvers import solve

print(ps[-1])
xqs = solve(ps[-1])
print(xqs)
```

Next construct the cardinal functions ℓ_0 and ℓ_1 associated with the 2 roots.

```
# Non-normalized version
L_01 = (x-xqs[1])
print(L_01)
print(L_01.subs(x, xqs[1]))
print(L_01.subs(x, xqs[0]))
# Normalize
L_01 /= L_01.subs(x, xqs[0])
print(L_01.subs(x, xqs[0]))

# Non-normalized version
L_11 = (x-xqs[0])
# Normalize
```

```
L_11 /= L_11.subs(x, xqs[1])

Ls = [L_01, L_11]
print(Ls)
```

Finally, compute the weights.

```
ws = [integrate(L, (x, a, b)) for L in Ls ]
print(ws)
```