1 Numerical integration: Part IV

1.1 Newton-Cotes and Gauß quadrature formulas

Anne Kværnø, André Massing

Date: Feb 1, 2021

If you want to have a nicer theme for your jupyter notebook, download the cascade stylesheet file tma4125.css and execute the next cell:

```python
from IPython.core.display import HTML
def css_styling():
    try:
        with open("tma4125.css", "r") as f:
            styles = f.read()
            return HTML(styles)
    except FileNotFoundError:
        pass  # Do nothing

# Comment out next line and execute this cell to restore the default notebook

# style
css_styling()
```

2 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 \( Q[·]|\{x_i\}_{i=0}^{n+1}, \{w_i\}_{i=0}^{n+1} \) 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 \( f(x) \geq 0 \) we have that the integral \( I[f](a,b) \geq 0 \) But for a quadrature rule with negative weights we have not necessarily that \( Q[f](a,b) \geq 0 \)! This has the undesired effect that the numerical integral of a positive function can be negative.

In addition, this 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 means 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.

## 3 Gauß quadrature

Last lecture, when comparing the trapezoidal rule with Gauß-Legendre quadrature rule, both based on two quadrature nodes, we observed that * the Gauß-Legendre quadrature was much more accurate than the trapezoidal rule,

- the Gauß-Legendre quadrature has degree of exactness equal to 3 and not only 1.

So obviously the position of the nodes matters!

**Questions:** *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?

**Intuition:** If we don’t predefine the quadrature nodes, we have \( 2n + 2 \) parameters (\( n + 1 \) nodes and \( n + 1 \) weights) in total.

With \( 2n + 2 \) parameters, we might hope that we can construct quadrature rules which are exact for \( p \in \mathbb{P}_{2n+1} \).

### 3.1 Definition 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).

### 3.2 Orthogonal polynomials

To construct Gaussian quadrature rule, we need to briefly review the concept of orthogonality, which we introduced when we learned about Fourier series.
Two functions \( f, g : [a, b] \rightarrow \mathbb{R} \) are orthogonal if
\[
\langle f, g \rangle := \int_a^b f(x)g(x) \, dx = 0.
\]
Usually, it will be clear from the context which interval \([a, b]\) we picked.

### 3.3 Theorem 1: Orthogonal polynomials on \([a, b]\)

There is a sequence of \( \{p_k\}_{k=1}^{\infty} \) of polynomials satisfying
\[
p_0(x) = 1, \tag{1}
\]
\[
p_k(x) = x^k + r_{k-1}(x) \quad \text{for } k = 1, 2, \ldots \tag{2}
\]
with \( r_{k-1} \in \mathbb{P}_{k-1} \) and …
satisfying the orthogonality property
\[
\langle p_k, p_l \rangle = \int_a^b p_k(x)p_l(x) \, dx = 0 \quad \text{for } k \neq l, \tag{3}
\]
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:
\[
\begin{align*}
\tilde{p}_0 &:= 1 = \phi_0 \\
\tilde{p}_1 &:= \phi_1 - \frac{\langle \phi_1, \tilde{p}_0 \rangle}{\lVert \tilde{p}_0 \rVert^2} \tilde{p}_0 \\
\tilde{p}_2 &:= \phi_2 - \frac{\langle \phi_2, \tilde{p}_0 \rangle}{\lVert \tilde{p}_0 \rVert^2} \tilde{p}_0 - \frac{\langle \phi_2, \tilde{p}_1 \rangle}{\lVert \tilde{p}_1 \rVert^2} \tilde{p}_1 \\
\vdots \\
\tilde{p}_k &:= \phi_k - \sum_{j=0}^{k-1} \frac{\langle \phi_k, \tilde{p}_j \rangle}{\lVert \tilde{p}_j \rVert^2} \tilde{p}_j
\end{align*}
\]

By construction, \( \tilde{p}_n \in \mathbb{P}_n \) and \( \langle p_k, p_l \rangle = 0 \) for \( k \neq l \). Since \( \tilde{p}_k(x) = a_kx^k + a_{k-1}x^{k-1} + \ldots a_0 \), we simply define \( p_k(x) = \tilde{p}_k/a_k \) to satisfy (2).
3.4 Theorem 2: Roots of orthogonal polynomials
Each of the polynomials $p_n$ defined in Theorem 1: Orthogonal polynomials on $[a,b]$ has $n$ distinct real roots.

Proof. Without proof, will be added later for the curious among you.

3.5 Theorem 3: Construction of Gaussian quadrature
Let $p_{n+1} \in \mathbb{P}_{n+1}$ be a polynomial on $[a,b]$ satisfying

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

Set $\{x_i\}_{i=0}^n$ to be the $n+1$ real roots of $p_{n+1}$ and define the weights $\{w_i\}_{i=0}^n$ by

$$w_i = \int_a^b \ell_i(x) \, dx.$$

where $\{\ell_i\}_{i=0}^n$ are the $n+1$ cardinal functions associated with $\{x_i\}_{i=0}^n$. The resulting quadrature rule is a Gaussian quadrature.

Proof. Without proof, will be added later for the curious among you.

Recipe 1 to construct a Gaussian quadrature.

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

1. Construct a polynomial $p_{n+1} \in \mathbb{P}_{n+1}$ on the interval $[a,b]$ which satisfies

$$\int_a^b p_{n+1}(x)q(x) \, dx \quad \forall \ q \in \mathbb{P}_n.$$ 

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 $w_i = \int_a^b \ell_i(x) \, dx$.

This is the recipe you are asked to use in Exercise set 3. Alternatively one can start from a reference interval, leading to

Recipe 2 to construct a Gaussian quadrature.

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

1. Construct a polynomial $p_{n+1} \in \mathbb{P}_{n+1}$ on the reference interval $[-1,1]$ which satisfies

$$\int_{-1}^1 p_{n+1}(x)q(x) \, dx \quad \forall \ q \in \mathbb{P}_n.$$ 

2. You determine the $n+1$ real roots $\{\hat{x}_i\}_{i=0}^n$ of $p_{n+1}$ which serve then as quadrature nodes.
3. Calculate the cardinal functions $\ell_i(x)$ associated with $n + 1$ nodes $\{\hat{x}_i\}_{i=0}^n$ and then the weights are given by $\hat{w}_i = \int_{-1}^{1} \ell_i(x) \, dx$.

4. Finally, transform the resulting Gauß quadrature formula to the desired interval $[a, b]$ via

$$x_i = \frac{b - a}{2} \hat{x}_i + \frac{b + a}{2}, \quad w_i = \frac{b - a}{2} \hat{w}_i \quad \text{for } i = 0, \ldots, n.$$ 

3.6 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.

Today we will use the sympy quite a bit, and start with the snippets

```python
3: 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:

9: # 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(3)]
print(phis)
[1, x, x**2]

Construct orthogonal polynomials (not normalized)

[]: # Insert code here

10: ps = []
# Use Gram-Schmidt
for phi in phis:
    ps.append(phi)
for p in ps[:-1]:
\[
ps[-1] = ps[-1] - \frac{scp(p, ps[-1])}{scp(p, p)}p
\]

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

```python
ps
[1, x - 1/2, x**2 - x + 1/6]
```

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

```
# Insert code here
```

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

```
int_p_q = 1
int_p_q = 0
int_p_q = 0
int_p_q = 0
int_p_q = 1/12
int_p_q = 0
int_p_q = 0
int_p_q = 0
int_p_q = 1/180
```

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

```
from sympy.solvers import solve
# Insert code here
```

```
[18]: from sympy.solvers import solve
```

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

```
x**2 - x + 1/6
[1/2 - sqrt(3)/6, sqrt(3)/6 + 1/2]
```

Next constructe the cardinal functions \( \ell_0 \) and \( \ell_1 \) associated with the 2 roots.

```
[14]: # 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
```

6
L_01 /= L_01.subs(x, xqs[0])
print(L_01.subs(x, xqs[0]))

x - 1/2 - sqrt(3)/6
0
-sqrt(3)/3
1

# Non-normalized version
L_11 = (x-xqs[0])
# Normalize
L_11 /= L_11.subs(x, xqs[1])

Ls = [L_01, L_11]
print(Ls)

[-sqrt(3)*(x - 1/2 - sqrt(3)/6), sqrt(3)*(x - 1/2 + sqrt(3)/6)]

Finally, compute the weights.

# Insert code here

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

[1/2, 1/2]

3.7 Exercise: Now construct a Gaussian quadrature for n=3 or n=4 points.