Computional Homework8 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Introduction to Gauss Quadrature

1.1 Fundamental concepts of Gauss Quadrature

Suppose $f(x)$ is a polynomial of degree at most $2n-1$. We seek quadrature weights $\{A_i\}$ and nodes $\{x_i\}$ such that

$$\int_a^b \omega(x)f(x)\,dx = \sum_{i=1}^{n} A_i f(x_i),$$

where $\omega(x)$ is a given weight function.

Let $\{S_k(x)\}_{k=0}^{\infty}$ denote a sequence of orthogonal polynomials with respect to the inner product

$$\langle S_k, S_m\rangle = \int_a^b \omega(x)S_k(x)S_m(x)\,dx.$$

By the division algorithm for polynomials: Suppose $p(x), s(x) \in \mathcal{P}(\mathbf{F})$ with $s(x) \neq 0$. Then there exist unique polynomials $q(x), r(x) \in \mathcal{P}(\mathbf{F})$ such that

$$p(x) = s(x)q(x) + r(x), \qquad \deg r < \deg s.$$

For our purposes, write

$$f(x) = Q_{n-1}(x)S_n(x) + R_{n-1}(x),$$

where $\deg Q_{n-1} \leq n-1$, $\deg R_{n-1} < n$.

Plugging into the integral, and using the orthogonality:

$$\int_a^b \omega(x)f(x)\,dx = \int_a^b \omega(x)Q_{n-1}(x)S_n(x)\,dx + \int_a^b \omega(x)R_{n-1}(x)\,dx.$$

Since $R_{n-1}(x)$ is a polynomial of degree at most $n-1$ and $S_n(x)$ is orthogonal to all polynomials of lower degree,

$$\int_a^b \omega(x)Q_{n-1}(x)S_n(x)\,dx = 0,$$

thus

$$\int_a^b \omega(x) f(x)\, dx = \int_a^b \omega(x) R_{n-1}(x)\, dx.$$

The quadrature rule must then be exact for all polynomials of degree at most $2n-1$, hence,

$$\sum_{i=1}^{n} A_i f(x_i) = \sum_{i=1}^{n} A_i \left(Q_{n-1}(x_i) S_n(x_i) + R_{n-1}(x_i)\right).$$

If we choose x_i to be the roots of $S_n(x)$, i.e., $S_n(x_i) = 0$ for $i = 1, \ldots, n$, then

$$\sum_{i=1}^{n} A_i Q_{n-1}(x_i) S_n(x_i) = 0,$$

so

$$\int_a^b \omega(x) R_{n-1}(x)\, dx = \sum_{i=1}^{n} A_i R_{n-1}(x_i).$$

Because R_{n-1} is an arbitrary degree $n-1$ polynomial, we can construct the weights A_i by requiring the quadrature to be exact for the Lagrange basis polynomials $l_j(x)$ for $j = 1, 2, \ldots, n$:

Let

$$l_j(x) = \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i}, \qquad l_j(x_i) = \delta_{ij}.$$

Then,

$$A_j = \int_a^b \omega(x) l_j(x)\, dx.$$

Moreover, since x_j is a root of $S_n(x)$, we have

$$S_n(x) = \prod_{i=1}^{n} (x - x_i), \qquad S_n'(x_j) = \prod_{\substack{i=1 \\ i \neq j}}^{n} (x_j - x_i).$$

Hence, the Lagrange polynomial can be written as

$$l_j(x) = \frac{S_n(x)}{(x - x_j) S_n'(x_j)}.$$

Thus, the integral becomes:

$$\int_a^b \omega(x) f(x)\, dx = \sum_{i=1}^{n} A_i f(x_i),$$

where

$$A_j = \int_a^b \omega(x) l_j(x)\, dx = \int_a^b \omega(x) \frac{S_n(x)}{(x - x_j) S_n'(x_j)}\, dx.$$

To transform the interval $[a, b]$ to $[-1, 1]$, let

$$x = \frac{b-a}{2}\eta + \frac{b+a}{2}, \qquad \eta \in [-1, 1].$$

Then,

$$\int_a^b \omega(x) f(x)\, dx = \frac{b-a}{2} \int_{-1}^{1} \Omega(\eta) F(\eta)\, d\eta,$$

where $\Omega(\eta) = \omega(x(\eta))$, $F(\eta) = f(x(\eta))$.

Finally, the Gauss quadrature rule becomes:

$$\int_a^b \omega(x) f(x)\, dx \approx \frac{b-a}{2} \sum_{i=1}^{n} a_i F(\eta_i),$$

where η_i are the roots of the orthogonal polynomial S_n with respect to the inner product

$$\langle S_k, S_m \rangle = \int_a^b \omega(x) S_k(x) S_m(x)\, dx.$$

2 Numerical Integration: Quadrature Formulas, Nodes, and Weights

We now illustrate the application and effectiveness of Gauss quadrature rules with two examples. The first uses Gauss-Legendre quadrature (weight $\omega(x) = 1$), and the second uses Gauss-Chebyshev quadrature (weight $\omega(x) = \frac{1}{\sqrt{1-x^2}}$).

2.1 Example 1: Gauss-Legendre Quadrature

We approximate

$$I_1 = \int_0^{\pi} e^{3x} \cos(x)\, dx$$

using Gauss-Legendre quadrature with $n = 2, 3, 4$ nodes. The true value is computed exactly as

$$I_1 = -3717.7943.$$

```
def If(x):
    return 1/10*(np.sin(x)+3*np.cos(x))*np.exp(3*x)


I_f = If(np.pi) - If(0)
```

The quadrature approximations and errors are summarized below:

n	True Value	Numerical Value	Absolute Error	Relative Error
2	-3717.7943	-2082.9894	1634.8050	0.4397
3	-3717.7943	-3504.0510	213.7433	0.0575
4	-3717.7943	-3711.2726	6.5217	0.00175

```python
def f(x):
    return np.exp(3*x)*np.cos(x)

class GaussQuadrature:
    def __init__(self,f,n,a,b,weight="1"):
        self.f=f
        self.n=n
        self.a=a
        self.b=b
        self.weight=weight

    def cal_quad(self):
        if self.weight=="1":
            nodes,weights=roots_legendre(self.n)
            xn=(self.b-self.a)/2*nodes+(self.b+self.a)/2
            return (self.b-self.a)/2*sum(self.f(xn)*weights)

GQ = GaussQuadrature(f, 2, 0, np.pi)
#GQ = GaussQuadrature(f, 3, 0, np.pi)
#GQ = GaussQuadrature(f, 4, 0, np.pi)
Q_f = GQ.cal_quad()
I_f = If(np.pi) - If(0)
abs_err_f = abs(Q_f - I_f)
rel_err_f = abs_err_f / abs(I_f)
print(" 【Gauss-Legendre on [0, pi]】 ")
print("True value       :", I_f)
print("Numerical value :", Q_f)
print("Absolute error   :", abs_err_f)
print("Relative error   :", rel_err_f)
print()
```

As the number of nodes n increases, the Gauss-Legendre quadrature’ s accuracy improves dramatically:

- With $n = 2$, the error is large, because the function $e^{3x}\cos(x)$ is highly non-polynomial and varies rapidly.
- At $n = 3$, accuracy already improves by more than an order of magnitude.
- By $n = 4$, the quadrature nearly matches the true value (relative error $< 0.2\%$).

2.2 Example 2: Gauss-Chebyshev Quadrature for Weighted Integral

For the weighted integral:

$$I_2 = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}\sqrt{16-x^2}}\,dx$$

Reduction to Elliptic Integral (Sketch) To compute

$$I_2 = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}\sqrt{16-x^2}}\,dx,$$

make the substitution $x = \sin\theta$. The bounds change to $\theta \in [-\pi/2, \pi/2]$, and the integral simplifies:

$$I_2 = \int_{-\pi/2}^{\pi/2} \frac{1}{\sqrt{16-\sin^2\theta}}\,d\theta = 2\int_{0}^{\pi/2} \frac{1}{\sqrt{16-\sin^2\theta}}\,d\theta$$

Using a standard formula for elliptic integrals:

$$\int_{0}^{\pi/2} \frac{1}{\sqrt{a^2-b^2\sin^2\theta}}\,d\theta = \frac{1}{a}K\left(\frac{b}{a}\right)$$

Setting $a = 4$, $b = 1$, we get:

$$I_2 = \frac{1}{2}K\left(\frac{1}{4}\right)$$

where $K(k)$ is the complete elliptic integral of the first kind.

```
k = 1/4
m = k ** 2
K_k = ellipk(m)
I_wg_true = 0.5 * K_k
```

we apply both Gauss-Legendre (which does not take the singular weight into account) and Gauss-Chebyshev (matching the weight):

Gauss-Legendre Quadrature ($n = 2$) for the weighted function:

True value	0.798121
Numerical value	0.618853
Absolute error	0.179268
Relative error	0.2246

```
def wg(x):
    return 1/(np.sqrt(1-x**2)*np.sqrt(16-x**2))

GQ = GaussQuadrature(wg, 2, -1, 1)
Q_wg = GQ.cal_quad()
abs_err_wg = abs(Q_wg - I_wg_true)
rel_err_wg = abs_err_wg / abs(I_wg_true)
print(" 【Gauss-Legendre on [-1, 1] for wg(x)】 ")
print("True value      :", I_wg_true)
print("Numerical value :", Q_wg)
print("Absolute error  :", abs_err_wg)
print("Relative error  :", rel_err_wg)
```

Gauss-Chebyshev Quadrature ($n = 2$):

True value	0.798121
Numerical value	0.797965
Absolute error	0.000156
Relative error	0.00020

```
def g(x):
    return 1/np.sqrt(16-x**2)

GQ = GaussQuadrature(g, 2, -1, 1, "cheby")
Q_g = GQ.cal_quad()
abs_err_g = abs(Q_g - I_wg_true)
rel_err_g = abs_err_g / abs(I_wg_true)
print(" 【Gauss-Chebyshev with weight on [-1, 1]】 ")
print("True value      :", I_wg_true)
print("Numerical value :", Q_g)
print("Absolute error  :", abs_err_g)
print("Relative error  :", rel_err_g)
```

```
print()
```

Observations:

- For the second case, Gauss-Chebyshev quadrature almost exactly matches the true value even with $n = 2$, demonstrating how matching the weight to the quadrature rule dramatically improves accuracy.
- Applying Gauss-Legendre quadrature to weighted integrals (with a singular weight) gives large error, further showing the importance of adapting the quadrature rule to the specific weight function of the integral.