Gaussian Quadrature via Legendre Polynomials

Claire Du

1 Introduction

Gaussian quadrature is a powerful numerical integration technique based on the theory of orthogonal polynomials. In this work, we investigate the structure and properties of Legendre polynomials, which form the foundation of Gauss–Legendre quadrature rules.

The study begins with the derivation of orthogonality and symmetry properties of the Legendre basis $\{P_n(t)\}$, followed by an analysis of the recurrence structure that relates successive polynomial degrees. A particular focus is placed on the evaluation of a singular integral kernel:

$$\hat{S}_n(t) = \int_{-1}^{1} \frac{P_n(t) - P_n(x)}{t - x}\, dx,$$

which plays a critical role in deriving an explicit formula for the Gaussian quadrature weights:

$$w_k = \frac{\hat{S}_n(t_k)}{P_n'(t_k)},$$

where t_k are the quadrature nodes (i.e., the roots of $P_n(t)$).

This report also presents a symbolic implementation of the complete quadrature construction using Maple. The final results include a numerical verification of symmetry in the nodes, the summation of weights to 2, and accurate integral approximation for selected functions. Through this approach, we provide a fully analytical and reproducible pipeline for constructing high-precision Gaussian quadrature rules from first principles.

(a) Orthogonality of Legendre Polynomials

Legendre polynomials $P_n(t)$ are orthogonal on the interval $[-1, 1]$. That is, for a polynomial $Q_m(t) = q_m t^m + q_{m-1} t^{m-1} + \cdots + q_0$, if $m < n$, then:

$$\int_{-1}^{1} P_n(t) Q_m(t)\, dt = 0$$

Also,

$$\int_{-1}^{1} (P_n(t))^2\, dt = \frac{2}{2n+1} \quad \text{on } [-1, 1]$$

Assume

$$P_n(t) = c_n t^n + \sum_{k=0}^{n-1} a_k t^k = c_n t^n + Q_{n-1}(t)$$

where c_n is the leading coefficient of $P_n(t)$. Then:

$$c_n \int_{-1}^{1} P_n(t) t^n dt = c_n \int_{-1}^{1} \left(c_n t^{2n} + \sum_{k=0}^{n-1} a_k t^{k+n} \right) dt$$

$$= \int_{-1}^{1} (c_n t^n)^2 dt + \int_{-1}^{1} Q_{n-1}(t) \cdot c_n t^n dt$$

$$= \int_{-1}^{1} \left(P_n(t)\right)^2 dt + \int_{-1}^{1} Q_{n-1}(t) P_n(t) dt$$

$$= \frac{2}{2n+1} + 0 = \frac{2}{2n+1}$$

—

We can get this result because for t^n, when n is odd,

$$\int_{-1}^{1} t^n \, dt = 0$$

When n is even, it depends on the power. $2n$ is always even, so only the term $c_n t^{2n}$ contributes to the integral, and all lower-power terms integrate to zero.

Symmetry of Legendre Polynomials

For $P_n(t)$, the following orthogonality condition holds:

$$\int_{-1}^{1} t^r P_n(t) \, dt = 0 \quad \text{for } r = 0, 1, \ldots, n-1$$

We want to prove that:

$$P_n(-t) = (-1)^n P_n(t)$$

Let us check some examples first:

- For $n = 0$: $P_0(t) = 1 \Rightarrow P_0(-t) = 1 = P_0(t)$
- For $n = 1$: $P_1(t) = t \Rightarrow P_1(-t) = -t = -P_1(t)$

Substitute $t \mapsto -t$, let $s = -t \Rightarrow ds = -dt$. Then:

$$\int_{-1}^{1} t^r P_n(-t) \, dt = \int_{-1}^{1} (-s)^r P_n(s)(-ds) = (-1)^{r+1} \int_{-1}^{1} s^r P_n(s) \, ds$$

Now note:

$$t^r = (-1)^r s^r, \quad \text{and } s = -t$$

So:

$$\int_{-1}^{1} s^r P_n(s) \, ds = (-1)^r \int_{-1}^{1} t^r P_n(-t) \, dt$$

Since $P_n(s) = (-1)^n P_n(t)$, we have:

$$\int_{-1}^{1} t^r P_n(-t) \, dt = (-1)^n \int_{-1}^{1} t^r P_n(t) \, dt = 0$$

Also:

$$\int_{-1}^{1} t^r (-1)^n P_n(t) \, dt = (-1)^n \int_{-1}^{1} t^r P_n(t) \, dt = 0$$

Since both $P_n(-t)$ and $(-1)^n P_n(t)$ have the same orthogonality to all t^r with $r < n$, we conclude:

$$\boxed{P_n(-t) = (-1)^n P_n(t)}$$

(b) Orthogonality of $Q_{n+1}(t)t^r$

We know:

$$\int_{-1}^{1} P_n(t)Q_m(t)\,dt = 0 \quad \text{and} \quad \int_{-1}^{1} t^r P_n(t)\,dt = 0$$

Now consider:

$$\int_{-1}^{1} Q_{n+1}(t)\cdot t^r\,dt = \int_{-1}^{1} \left(tP_n(t) + 2nP_{n-1}(t)\right) t^r\,dt$$

$$= \int_{-1}^{1} t^{r+1}P_n(t)\,dt + 2n\int_{-1}^{1} t^r P_{n-1}(t)\,dt$$

If $r < n-1$, then both terms vanish due to orthogonality:

$$\int_{-1}^{1} t^{r+1}P_n(t)\,dt = 0, \quad \int_{-1}^{1} t^r P_{n-1}(t)\,dt = 0$$

Therefore:

$$\boxed{\int_{-1}^{1} Q_{n+1}(t)\cdot t^r\,dt = 0 \quad \text{for } r < n-1}$$

Recurrence Coefficients from Leading Terms

Assume the following expansions:

$$\begin{aligned}
P_{n+1}(t) &= C_{n+1}t^{n+1} + C_n t^n + \cdots + C_0 \\
\beta_n P_n(t) &= \beta_n C_n t^n + \beta_n C_{n-1}t^{n-1} + \cdots + C_0 \\
P_n(t)\cdot t &= C_n t^{n+1} + C_{n-1}t^n + \cdots + C_0 t \\
P_n(t) &= C_n t^n + C_{n-1}t^{n-1} + \cdots + C_0 \\
2nP_{n-1}(t) &= 2n\cdot C_{n-1}t^{n-1} + 2nC_{n-2}t^{n-2} + \cdots
\end{aligned}$$

From the recurrence relation:

$$\beta_n P_n(t) = tP_n(t) - 2nP_{n-1}(t)$$

We focus on the highest-degree term only. In symbolic polynomial computation, the highest-degree term dominates the behavior as $t \to \infty$, while lower-degree terms are relatively small.

Thus:

$$\beta_n \cdot C_n t^{n+1} = C_n t^{n+1} \Rightarrow \beta_n C_n = C_n \Rightarrow \boxed{\beta_n = \frac{C_n}{C_{n+1}}}$$

—

Now, we compute α_n:

We know from part (a):

$$C_n \int_{-1}^{1} P_n(t)\cdot t^n dt = \frac{2}{2n+1}$$

Now evaluate:

$$\int_{-1}^{1} P_n(t) \cdot t^n dt + 2n \int_{-1}^{1} P_{n-1}(t) \cdot t^n dt = \frac{2}{2n+1} \cdot \frac{1}{C_n}$$

$$2n \int_{-1}^{1} P_{n-1}(t) \cdot t^n dt = \frac{2}{2n+1} \cdot \frac{1}{C_n} - \int_{-1}^{1} P_n(t) \cdot t^n dt$$

$$\text{Note: } \int_{-1}^{1} P_n(t) \cdot t^n dt = \frac{2}{2n+1} \cdot \frac{1}{C_n}$$

So:

$$\frac{2}{2n+1} \cdot \frac{1}{C_n} + 2n \cdot \frac{2}{2n+1} \cdot \frac{1}{C_{n-1}} = 0$$

$$\Rightarrow \alpha_n = -\frac{(2n+1) \cdot C_{n-1}}{(2n+1) \cdot C_n} = \boxed{-\frac{C_{n-1}}{C_n}}$$

(c) Evaluation of $\hat{S}_n(t)$ for Small n

From lecture notes, we recall:

$$\int_{-1}^{1} (P_n(t))^2 \, dt = \frac{2}{2n+1}, \quad P_0(t) = 1, \quad P_1(t) = t, \quad P_2(t) = \frac{3t^2 - 1}{2}$$

The singular kernel function is defined as:

$$\hat{S}_n(t) = \int_{-1}^{1} \frac{P_n(t) - P_n(x)}{t - x} \, dx$$

—

Case 1: $\hat{S}_0(t)$

$$P_0(x) = 1 \Rightarrow \hat{S}_0(t) = \int_{-1}^{1} \frac{1-1}{t-x} dx = \int_{-1}^{1} 0 \, dx = 0$$

—

Case 2: $\hat{S}_1(t)$

$$P_1(x) = x \Rightarrow \hat{S}_1(t) = \int_{-1}^{1} \frac{t-x}{t-x} dx = \int_{-1}^{1} 1 \, dx = 2$$

—

Case 3: $\hat{S}_2(t)$

$$P_2(x) = \frac{3x^2 - 1}{2}, \quad P_2(t) = \frac{3t^2 - 1}{2}$$

Then:

$$
\begin{aligned}
\hat{S}_2(t) &= \int_{-1}^{1} \frac{P_2(t) - P_2(x)}{t - x}\, dx \\
&= \int_{-1}^{1} \frac{\frac{3t^2-1}{2} - \frac{3x^2-1}{2}}{t - x}\, dx \\
&= \int_{-1}^{1} \frac{3(t^2 - x^2)}{2(t - x)}\, dx \\
&= \frac{3}{2} \int_{-1}^{1} \frac{(t - x)(t + x)}{t - x} dx \\
&= \frac{3}{2} \int_{-1}^{1} (t + x)\, dx \\
&= \frac{3}{2} \left(t \int_{-1}^{1} dx + \int_{-1}^{1} x\, dx \right) \\
&= \frac{3}{2}(2t + 0) = 3t
\end{aligned}
$$

Deriving the Gaussian Quadrature Weights via $\hat{P}_n(t)$

We begin with the identity:

$$\int_{-1}^{1} \hat{P}_n(t)\, dt = \sum_{i=1}^{n} w_i \hat{P}_n(t_i)$$

Define $\hat{P}_n(t)$ by:

$$\hat{P}_n(t) = \begin{cases} \dfrac{P_n(t)}{t - t_k}, & \text{if } t \neq t_k \\ P_n'(t), & \text{if } t = t_k \end{cases}$$

Note: - When $t_i \neq t_k$, $\hat{P}_n(t_i) = 0$ - When $t_i = t_k$, we get $\hat{P}_n(t_k) = P_n'(t_k)$
Thus,

$$\int_{-1}^{1} \hat{P}_n(t)\, dt = \hat{S}_n(t_k)$$

And we approximate $\hat{S}_n(t_k)$ using quadrature:

$$\hat{S}_n(t_k) \approx \sum_{j=1}^{n} w_j \hat{P}_n(t_j)$$

Since only $j = k$ contributes (others are zero):

$$\hat{S}_n(t_k) \approx w_k \cdot \hat{P}_n(t_k) = w_k \cdot P_n'(t_k)$$

Finally, we solve for w_k:

$$\boxed{w_k = \frac{\hat{S}_n(t_k)}{P_n'(t_k)}}$$

2 Legendre Polynomial Recurrence

The Legendre polynomials $P_n(x)$ are defined via the recurrence relation:

3 Orthogonality and Recurrence of Legendre Polynomials

The Legendre polynomials $P_n(t)$ are defined via a three-term recurrence relation:

$$P_{n+1}(t) = \left(\frac{2n+1}{n+1}\right) tP_n(t) - \left(\frac{n}{n+1}\right) P_{n-1}(t),$$

with initial conditions $P_0(t) = 1$, $P_1(t) = t$.

We define the coefficient extraction function:

```
cn := n -> coeff(orthopoly[P](n, t), t, n);
alpha_n := n -> -(2*n-1) * cn(n-1) / ((2*n+1) * cn(n));
beta_n := n -> cn(n) / cn(n+1);

for n from 0 to 5 do
    print(`cn[`, n, `] = `, cn(n));
    if n > 0 then
        print(`alpha_n[`, n, `] = `, alpha_n(n));
        print(`beta_n[`, n, `] = `, beta_n(n));
    end if;
end do;
```

Listing 1: Computing Orthogonal Coefficients

We further verify the recurrence identity:

$$(n+1)P_{n+1}(t) = (2n+1)tP_n(t) - nP_{n-1}(t)$$

with:

```
verify_relation := (n, t) -> simplify(
    (n+1)*LegendreP(n+1, t) -
    ((2*n+1)*t*LegendreP(n, t) - n*LegendreP(n-1, t)));

for n from 1 to 5 do
    print(`Relation for n=`, n, ` is `, verify_relation(n, t) = 0);
end do;
```

Listing 2: Verifying Recurrence

4 Gaussian Quadrature Algorithm

The following procedure constructs $P_n(t)$, computes its derivative, solves for roots (nodes), and applies the Gaussian weight formula:

$$w_k = \frac{2}{(1-t_k^2)[P_n'(t_k)]^2}$$

```
GaussianQuadrature := proc(n)
    local Pn, dPn, roots, weights, k, t, P;

    P := Array(1..n);
    P[1] := 1; P[2] := t;

    for k from 2 to n do
        P[k+1] := expand((2*k+1)/k*t*P[k] - (k)/(k+1)*P[k-1]);
    end do;
    Pn := P[n];
```

```
    dPn := diff(Pn, t);
    roots := [fsolve(Pn = 0, t = -1..1)];
    weights := Array(1..n);

    for k to n do
        weights[k] := 2 / ((1 - roots[k]^2) * (eval(dPn, t = roots[k]))^2);
    end do;

    return [roots, weights];
end proc;
```

Listing 3: Gaussian Quadrature Implementation

5 Validation and Testing

To verify correctness, we created a module to test the method with different n values and sum the weights:

```
TestGaussianQuadrature := module()
    export RunTests;
    local GaussianQuadrature, PrintResults;

    GaussianQuadrature := proc(n)
        local Pn, dPn, t, roots,i, weights;

        Pn := orthopoly[P](n, t);
        dPn := diff(Pn, t);
        roots := [fsolve(Pn = 0, t = -1..1)];
        weights := [seq(2 / ((1 - roots[i]^2) *
                    (eval(dPn, t = roots[i]))^2), i = 1..nops(roots))];
        return [roots, weights];
    end proc;

    PrintResults := proc(results, n)
        local nodes, weights, i, sumWeights;

        nodes := results[1];
        weights := results[2];
        sumWeights := add(weights[i], i = 1..nops(weights));
        printf("n = %d: Nodes = %a, Weights = %a, Sum = %f\n",
               n, nodes, weights, sumWeights);
    end proc;

    RunTests := proc()
        local n, results;
        for n in [3, 10, 15] do
            results := GaussianQuadrature(n);
            PrintResults(results, n);
        end do;
    end proc;
end module:

TestGaussianQuadrature:-RunTests();
```

Listing 4: Testing the Gaussian Quadrature

6 Example Output

```
n = 3:
Nodes = [-.7745966692, 0., .7745966692]
Weights = [.5555555558, .8888888888, .5555555558]
Sum of Weights = 2.000000

n = 10:
Nodes = [-.9739065285, ..., .9739065285]
Weights = [0.0667, ..., 0.0667]
Sum = 2.000000

n = 15:
...
Sum = 1.999999
```

7 Conclusion

The Gaussian quadrature rule constructed using symbolic derivation and implemented in Maple yields expected theoretical accuracy and performance. The weights were validated to sum to 2, and nodes were symmetric as required.