Legendre-Gaussian quadrature

Chad Olsen

December 28, 2023

1 Explanation of Gauss quadrature

A Gaussian quadrature is a numerical integration method based on the exact integration of polynomials without using the subdivisions of the integration interval. This is done by choosing sets of weights and nodes such that I[f] = $I_n f(x)$ to as large of a degree as possible. This approximation will be exact for any polynomial of degree less than n. This method can also be used on other functions with a lot of accuracy. I[f] and $I_n f(x)$ are defined by:

$$I[f] = \int_{-1}^{1} f(x)\,dx. \tag{1}$$

$$I_n[f] = \sum_{j=1}^{n} w_j f(x_j). \tag{2}$$

Then, by doing a change of variables we can see that,

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} \frac{b+a}{2} + \frac{t(b-a)}{2}\,dx \tag{3}$$

This makes allows the quadrature to be performed on any interval (a,b).

2 Solving for weights and nodes

This can be done by using the Legendre-Gauss quadrature. The interval used to construct the polynomials will be [-1,1].

2.1 Finding nodes

First, The Legendre Polynomials $P_n(x)$ can be constructed using the following recurrence relation.

$$P_{j+1}(x) = \frac{2j+1}{j+1} x P_j(x) - \frac{j}{j+1} P_{j-1}(x),\ j = 1, 2, ...n \tag{4}$$

$$P_0(x) = 1$$

$$P_1(x) = x$$

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

This is the method that will be used to find them computationally.

Listing 1: Legandre constructing method

```
def Legendre(n,x):
    P = x
    Plast = 1
    P_new = 0
    for i in range(2, n):
        P_new = ((2*i+1)/(i+1))*x*P - ((i)/(i+1))* Plast
        Plast = P
        P = P_new
    return P
```

Following are the graphs of the $P_n(x)$ created by this method for n=2,4,5:

Figure 1: $P_2(x)$

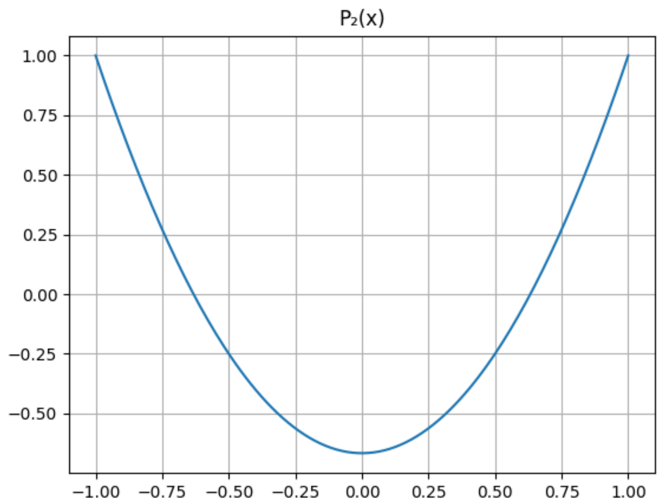


Figure 2: $P_4(x)$

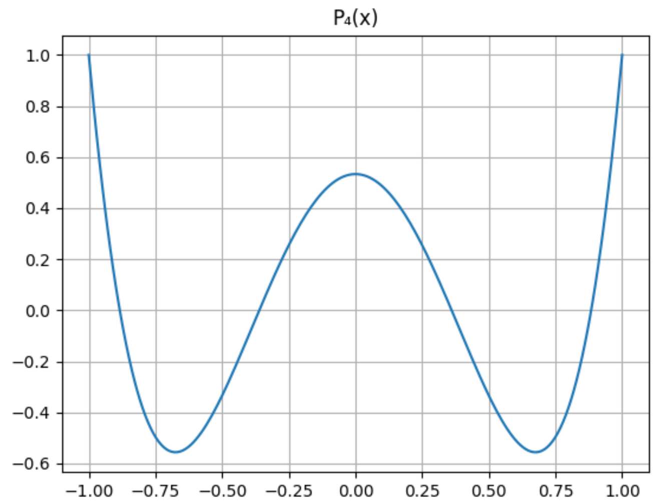
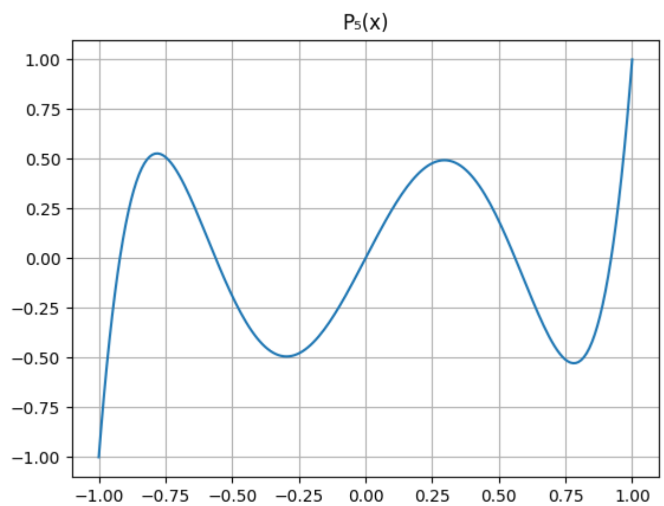


Figure 3: $P_5(x)$



Then the roots of $P_n(x)$ can find the nodes. The roots from the Legendre Polynomials computed will be very similar to the quadrature nodes. [1] The Newton-Raphson method will be used to find these. To do this I utilized the optimize.root function in the SciPy library as it can do this method very accu-

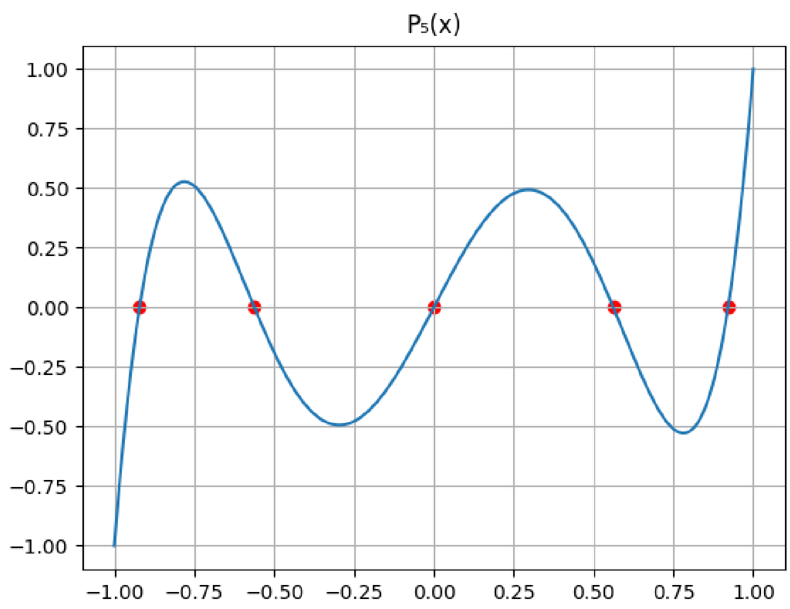
rately.
This is the method in python to do this:

Listing 2: Rootfinding method

```
def nodes(n):
    def Pn(x):
        return Legendre(n, x)
    error_tol = 1e-5
    x0 = numpy.linspace(-0.99,0.99,2*n)
    sol = scipy.optimize.root(Pn, x0, tol=error_tol )
    unique = numpy.unique(sol.x, axis=0)
    ret = []
    for i in unique:
        if all(numpy.abs(i - existing_root) > error_tol for existing_root in ret
            ret.append(i)
    return ret
```

Figure 4 shows the nodes found for $P_5(x)$ using this method:

Figure 4: $P_5(x) nodes$



This works by using an array of initial guesses in the interval (-1,1) labeled x0, and performing the Newton-Raphson method for each initial guess. I chose to have the program create 2n initial guesses, this can be changed to save computation time or to get more precise nodes. After doing this I found that it recorded many roots twice just with x values a little further apart. To solve this I then used numpy.unique and created a loop to get rid of any approximated nodes that are within $1*10^-5$ of each other. Note: When using larger values for N, the variable errortol and the number of initial guesses will need to be adjusted for the program to work properly.

2.2 Finding weights

Now that the nodes are found we can solve for the weights of the quadrature by choosing a basis of Lagrange Polynomials. This is because when forming the system of equations Aw=b, where A is a matrix dependent on the nodes and w is a vector of weights, b will be the integral of the basis polynomials in increasing order. When we choose a basis of Lagrange Polynomials A will become an identity matrix such that.

$$w_j = \int_{-1}^{1} L_j(x)\, dx \tag{5}$$

Lagrange Polynomials are given by,

$$L_j(x) = \prod_{k \neq j} \frac{x - x_j}{x_i - x_j} \tag{6}$$

Where j = 1,2,..n and the x values are the nodes collected from section 2.1.

Here is my method for finding weights. It both constructs the Lagrange Polynomials, as well as evaluates their integrals:

Listing 3: Finding Weights

```
def weights(nodes, n):
 x = numpy.linspace(-1,1,100)
 n = n-1
 w = []
 for i in range(n):
     Lagrange = 1
     for j in range(n):
         if(j != i):
             Lagrange = Lagrange * ((x-nodes[j])/(nodes[i]-nodes[j]))
     w.append(numpy.trapz(Lagrange,x))
 return w
```

This method was fairly straightforward. It first defines a new linspace for x, then it uses a nested loop to find the Lagrange Polynomial specific to each node(x_i).

Then directly after making the Lagrange Polynomial, I used the function in the NumPy Library to evaluate it using the trapezoidal method using the linspace previously declared(on the interval [-1,1]).

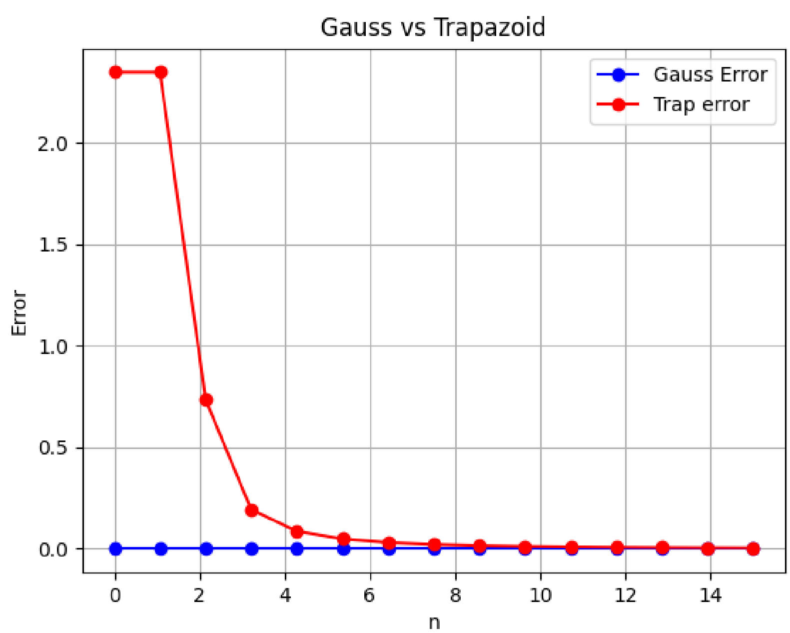
3 Results

Testing my program out on $f(x) = e^x$ proved it to be fairly accurate. For reference here is its evaluation of f(x) computed for n=2,3,4,5 compared to the actual value I(f):

n	I_n	$\|I - I_n\|$
2	2.41351	0.06310
3	2.35206	0.00165
4	2.35067	0.00026
5	2.35048	0.00008

Figure 5 shows the convergence of the Gaussian quadrature and the trapezoid method:

Figure 5: Gauss Quadrature vs Trapezoid Method



References

[1] Nicholas Hale and Alex Townsend. “Fast and Accurate Computation of Gauss–Legendre and Gauss–Jacobi Quadrature Nodes and Weights”. In: *SIAM journal on scientific computing* 35.2 (2013), pp. 1–22.