# Gauss-Legendre Quadrature in 1D and 2D

March 5, 2020

## 0.1 Gauss-Legendre Quadrature Rule in 1D and 2D

In general a quadrature rule is an approximation of the definite integral of a function $f(x)$ over an interval $[a, b]$ by a sum of weighted functions at certain points within the domain of integration. The goal is to find such weights and points within the domain. In general, a quadrature is in the form of:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where $w_i$'s are the weights and $x_i$'s are the fixed points (the **roots** of a polynomial belonging to a class of orthogonal polynomials) within the domain. In the case of Gaussian quadrature, $a = -1, b = 1$, and the quadrature is exact (approximation becomes equality) up to polynomials of degree $2n-1$. Gaussian-Legendre quadrature is a special form of Gaussian quadrature that utilizes orthogonal polynomials that are called Legendre polynomials, denoted by $P_n(x)$. These polynomials are defined as an orthogonal system (orthogonal with respect to $L_2$ inner product) satisfying:

$$< P_m(x),\ P_n(x) > = \int_{-1}^{1} P_m(x)P_n(x)dx = 0 \ \text{ if } \ n \neq m$$

Some of the first Legendre polynomials are: $P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{3x^2-1}{2} \ldots$. Rest of the legendre polynomials can be found using **Bonnet's recursion formula**:

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

or **Rodrigues' formula**:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n$$

With the *n-th* polynomial normalized to given $P_n(1) = 1$, the *i-th* Gauss node $x_i$ is the *i-th* root of $P_n(x)$, and the weights are given by:

$$w_i = \frac{2}{(1-x_i^2)(P'_n(x_i))^2}$$

When the integral is taken over a random interval $[a, b]$, interval can easily be changed to $[-1, 1]$ following way:

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\Big(\frac{b-a}{2}z + \frac{b+a}{2}\Big)dz = c\int_{-1}^{1} f\Big(cz+d\Big)dz$$

where $c = \dfrac{b-a}{2}$, $d = \dfrac{b+a}{2}$. Similarly, gaussian quadrature can be generalized to any interval using the transformation above:

$$\int_a^b f(x)dx = c\int_{-1}^{1} f\Big(cz+d\Big)dz \approx c\sum_{i=1}^{n} w_i f(cx_i + d) = \sum_{i=1}^{n} \tilde{w_i} f(cx_i + d)$$

where $\tilde{w_i} = c * w_i$.

### 0.1.1 Newton's Method for Root Finding

Newthon-Raphson method is a root-finding algorithm producing successively better approximations to the zeroes (roots) of a real-valued function $f(x)$. Starting with an initial guess $x_0$ (assuming it is close enough), the iteration is given by the formula approximates to a root of the function $f(x)$:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{for} \quad n = 0, 1, 2, \ldots$$

We will use this method in order to approximate the roots of legendre polynomials and find $x_i$ values for the Gauss-Legendre quadrature.

[20]:
```
import numpy as np
import pandas as pd
```

[21]:
```
class Quadrature():

    def __init__(self, a=-1, b=1, n=2, tolerance=10**-5):
        self.a = a
        self.b = b
        self.dim = n
        self.tol = tolerance
        self.x = np.zeros((n,1))
        self.w = np.zeros((n,1))
```

[22]:

```python
class Gauss_Legendre_1D(Quadrature):

    def __init__(self, a, b, n, tolerance):
        super().__init__(a, b, n, tolerance)



    def quad_1D(self):
        m = int((self.dim+1)/2)
        c = (self.b-self.a)/2
        d = (self.b+self.a)/2

        for i in range(m):

            z = np.cos(np.pi*(i+3/4)/(self.dim+1/2))
            delta = self.tol + 1

            while delta > self.tol:

                P1 = 0
                P2 = 1

                for j in range(self.dim):

                    P3 = ((2*j+1)*z*P2 - j*P1)/(j+1)      # Bonet's recursion␣
↪formula

                    dP = self.dim * (z*P3 - P2)/(z**2-1)
                    P1, P2 = P2, P3

                z_old = z
                z = z_old - P3/dP                         # Newton's update for␣
↪root finding
                delta = abs(z - z_old)


            self.x[i] = d - c*z
            self.x[self.dim-1-i] = d + c*z
            self.w[i] = 2*c/((1-z**2)*(dP**2))           # Weight formula (5)␣
↪combined with (7)
            self.w[self.dim-1-i] = self.w[i]

        df = pd.DataFrame(data=np.hstack((self.x, self.w)),␣
↪columns=['$x_i$','$w_i$'])

        return df
```

[158]:
```python
class Gauss_Legendre_2D(Gauss_Legendre_1D):

    def __init__(self, a, b, n, tolerance):
        super().__init__(a, b, n, tolerance)

    def quad_2D(self):
        df = super().quad_1D()
        self.x = np.zeros((self.dim**2, 2)) # Notice here x := (x ,y) in 2D
        self.w = np.zeros((self.dim**2, 1))
        k = 0
        for i in range(self.dim):
            for j in range(self.dim):
                self.w[k]    = df.iloc[:, 1][i] * df.iloc[:, 1][j]
                self.x[k, 0] = df.iloc[:, 0][i]
                self.x[k, 1] = df.iloc[:, 0][j]
                k += 1

        DF = pd.DataFrame(data=np.hstack((self.x, self.w)),␣
↪columns=['$x_i$','$y_i$','$w_i$'])

        return DF
```

[162]:
```python
a, b, n, tolerance = (-1, 1, 3, 10**(-6))

gl1 = Gauss_Legendre_1D(a, b, n, tolerance)
gl1.quad_1D() # Gives the roots - weight pair for 1D
```

[162]:

| | \$x_i\$ | \$w_i\$ |
|---|---|---|
| 0 | -0.774597 | 0.555555 |
| 1 | 0.000000 | 0.888889 |
| 2 | 0.774597 | 0.555555 |

[163]:
```python
gl2 = Gauss_Legendre_2D(a, b, n, tolerance)
gl2.quad_2D() # Gives the roots - weight pair for 2D
```

[163]:

| | \$x_i\$ | \$y_i\$ | \$w_i\$ |
|---|---|---|---|
| 0 | -0.774597 | -0.774597 | 0.308642 |
| 1 | -0.774597 | 0.000000 | 0.493827 |
| 2 | -0.774597 | 0.774597 | 0.308642 |
| 3 | 0.000000 | -0.774597 | 0.493827 |
| 4 | 0.000000 | 0.000000 | 0.790123 |
| 5 | 0.000000 | 0.774597 | 0.493827 |
| 6 | 0.774597 | -0.774597 | 0.308642 |
| 7 | 0.774597 | 0.000000 | 0.493827 |
| 8 | 0.774597 | 0.774597 | 0.308642 |