# Contents

# Chapter 1

# Introduction

Computation of closed-form antiderivatives that are encountered in many engineering phenomena, is not always possible. The lack of an analytical solution motivates a numerical approximation of the integral. *Numerical integration* or *quadrature* is the computation of an approximate solution to the *definite* integral $\int_a^b f(x)dx$ to a considerable degree of accuracy. Depending on the behaviour of $f(x)$ in the range of integration, various methods may be applied. In this work, methods pertaining to one-dimensional integrals are described.

The function $f(x)$ is known as the *integrand*. Numerical quadrature methods can be defined in general as the weighted sum of evaluations of the integrand at a finite set of points called *integration points* or *nodes*.

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

The nodes($x_i$) and weights($w_i$) vary according to the method and desired accuracy.

The present work focuses on quadrature rules that are built on interpolating functions that are computationally efficient to integrate. *Polynomials* are a suitable choice for the interpolating function. The methods that are described in this document are:

- Composite-Trapezoidal method
- Composite-Simpson method
- Generalized Closed Newton-Cotes method
- Gauss-Chebyshev method
- Gauss-Hermite method
- Gauss-Legendre method

- Gauss-Laguerre method
- Gauss-Lobatto method

The corresponding methods have been implemented in MATLAB in an intuitive manner. The aim of this implementation is to compute the nodes and weights according to the user input at run-time. This document explains the underlying theory of the above mentioned methods.

# Chapter 2

# Description

In this chapter, the description of methods mentioned in Chapter 1, along with the underlying theory and necessary mathematical preliminaries are presented.

## 2.1 Composite-Trapezoidal method

The Composite-Trapezoidal method, an extension of the 2-point Trapezoidal rule, approximates the area under the graph of the function $f(x)$ as a series of trapezoids and computes the area (Fig 2.1). This is achieved by dividing the given interval into $n$ sub-intervals, apply the 2-point trapezoidal rule for each sub-interval and perform a summation over the results.

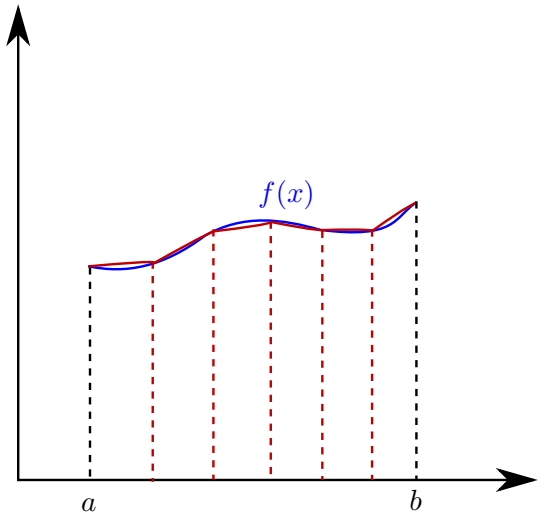


FIGURE 2.1: Illustration of Composite-trapezoidal rule

This enhances the accuracy of the result considerably. Also, this rule converges faster when the integrand is a periodic function. The interpolating function is a polynomial of degree 1.

To compute the numerical integral of a function $f(x)$ from $a$ to $b$ using the Composite-Trapezoidal method, the following procedure can be adopted:

- Divide the range of integration into $n$ sub-intervals, that gives a step size $h = (b-a)/n$
- The integration points or nodes (excluding the limits) can be written as $x_k = a + kh$ $(k = 1, 2, 3, ..., n-1)$
- The integral can now be evaluated as

$$\int_a^b f(x)dx \approx \frac{(b-a)}{n}\left(\frac{f(a)}{2} + \sum_{k=1}^{n-1} f(x_k) + \frac{f(b)}{2}\right) \tag{2.1}$$

This method is implemented in the MATLAB function `CTrap`.

## 2.2 Composite-Simpson method

This is an extension of the 3-point Simpson method. If the integrand is a non-smooth function (oscillatory, discontinuous at a few points) over the given range, the accuracy of the result greatly reduces. A solution for such problems is dividing the given range into an even number of sub-intervals, apply the 3-point rule on each sub-interval and perform a summation over the results.

To compute the numerical integral of a function $f(x)$ from $a$ to $b$ using the Composite-Simpson's method, the following procedure can be adopted:

- Divide the range of integration into $n$ sub-intervals, giving a step size $h = (b-a)/n$. Make sure that $n$ is **even**.
- The integral can be now evaluated as

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(a) + 2\sum_{k=1}^{(n-2)/2} f(x_{2k}) + 4\sum_{l=1}^{n/2} f(x_{2l-1}) + f(b)\right) \tag{2.2}$$

- The interpolating function is a polynomial of degree 2 which is essentially a *parabola*. This in fact motivates $n$ to be even.

The above method is implemented in the MATLAB function `CSimp`.

## 2.3 Generalized Closed Newton-Cotes quadrature method

This is a generalized formula that encompasses a lot of other methods like Trapezoidal, Simpson's, Boole's rule etc for evaluating the approximate value of definite integrals. To compute the numerical integral of a function $f(x)$ from $a$ to $b$ using a Newton-Cotes quadrature rule where the interpolating function is a polynomial of degree $m$, the following procedure can be adopted:

- The Closed Newton-Cotes rule of degree $m$ can be written as

$$\int_a^b f(x)dx \approx \sum_{i=0}^{m} A_i f(x_i) \tag{2.3}$$

  where $x_i$ are the nodes and $A_i$ are the corresponding coefficients.

- Without loss of generality, polynomials $x^0$, $x^1$, $x^2$, ... ,$x^m$ can be chosen as $f(x)$ which will lead to linear system of equations in $A_i$

$$\begin{aligned} A_0 + A_1 + \cdots + A_m &= b - a \\ A_0 x_0 + A_1 x_1 + \cdots + A_m x_m &= (b^2 - a^2)/2 \\ &\vdots \\ A_0 x_0^m + A_1 x_1^m + \cdots + A_m x_m^m &= (b^{m+1} - a^{m+1})/(m+1) \end{aligned}$$

- The coefficients when divided by the step size $h = (b-a)/m$ provide the corresponding weights of the quadrature.

- Substituting $b = a+mh$ and $x_i = a+ih$ into the above system of equations and simplifying further gives a linear system of equations in $w_i$

$$\begin{aligned} w_0 + w_1 + \cdots + w_m &= m \\ w_1 + 2w_2 + \cdots + mw_m &= m^2/2 \\ &\vdots \\ w_1 + 2^m w_2 + \cdots + m^m w_m &= m^{m+1}/(m+1) \end{aligned}$$

- This system can be written in matrix form $\mathbf{VW} = \mathbf{M}$.

  Here $\mathbf{V}$ is the transpose of the *Vandermonde* matrix, $\mathbf{W}$ is the vector of unknowns $w_i$ and $\mathbf{M}$ is the vector of constant terms. This system can be solved for $w_i$

- With the nodes and weights in hand, the quadrature values can be calculated analogous to the above methods.

This method has been implemented as a composite rule in the MATLAB function `NC`.

The limitation of this generalized formula is, however, that the integration points are to be equally spaced for any degree. This is because polynomial interpolation using a high degree causes oscillation at the ends of the interval (*Runge's phenomenon*). Also, the vandermonde matrix is ill-conditioned. Though, for a degree $m \leq 10$ the system is fairly stable.

## 2.4 Gaussian Quadrature

Until now, procedures contained equally spaced abscissas at which function evaluations take place. However, in Gaussian quadrature, this is not true. There is a freedom in the choice of nodes as well, consequently allowing to perform a higher order quadrature with the same number of function evaluations (compared to Newton-Cotes method).

Regardless of the type of method, the integrand is desired to be smooth over the interval which allows a well-approximated polynomial as well. But, in practice the integrand is seldom smooth. This issue is addressed in Gaussian quadrature using a so-called *weighing function* using which integrable singularities are removed. Conventionally, the domain of integration is taken as [-1,1]. The Gaussian quadrature can be represented as,

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} W(x)g(x)dx \approx \sum_{i=0}^{n-1} w_i g(x_i) \tag{2.4}$$

Also, it has been well established that the nodes $x_i$ can be evaluated from the roots of an *orthogonal polynomial*.

In this context, the **inner product** of two functions $p(x)$ and $q(x)$ over a weight function $W(x)$ is defined as

$$\langle p, q \rangle = \int_a^b W(x)p(x)q(x)dx \tag{2.5}$$

With this definition in mind, using the *Gram-Schmidt process*, a set of orthogonal polynomials $p_j(x)$ can be generated as,

$$p_n(x) := x^n - \sum_{k=0}^{n-1} \frac{\langle x^n, p_k \rangle}{\langle p_k, p_k \rangle} p_k(x) \tag{2.6}$$

However, (2.6) is numerically unstable. Hence, to construct the orthogonal polynomials for our purpose, the following theorem is used.

**Theorem 2.1.** *Monic orthogonal polynomials are given by the three term recurrence relation,*

$$\begin{gathered} p_{-1}(x) \equiv 0 \\ p_0(x) \equiv 1 \\ p_{n+1}(x) = (x - a_n)p_n(x) - b_n p_{n-1}(x) \end{gathered}$$

*where (for,$n \geq 1$)*

$$a_n := \frac{\langle p_n, xp_n \rangle}{\langle p_n, p_n \rangle} \tag{2.7}$$

$$b_n := \frac{\langle p_n, p_n \rangle}{\langle p_{n-1}, p_{n-1} \rangle} \tag{2.8}$$

Based on above definitions, the nodes $x_i$ and weights $w_i$ can be computed from eigenvalue analysis of the tridiagonal *Jacobi matrix* given by,

$$J_n = \begin{pmatrix} a_0 & \sqrt{b_1} & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & \\ & \sqrt{b_2} & \ddots & \ddots & \\ & & \ddots & a_{n-2} & \sqrt{b_{n-1}} \\ & & & \sqrt{b_{n-1}} & a_{n-1} \end{pmatrix} \tag{2.9}$$

The nodes $x_i$ are nothing but the eigenvalues of the above matrix. Let $\mathbf{V} = V_{ij}$ be the matrix of eigen vectors. The weights are given by

$$w_i = V_{1j}^2 \int_{\mathbb{S}} W(x) dx \tag{2.10}$$

where $\mathbb{S} \subset \mathbb{R}$ is the domain of integration.

In the present work, Gaussian quadrature based on the classical orthogonal polynomials have been implemented.

### 2.4.1 Gauss-Chebyshev quadrature

This method uses the *Chebyshev polynomials of the first kind.* Using the recurrence relation, they can be written as,

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{2.11}$$

This method is used for evaluating the approximate value of the integral of the kind:

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} \tag{2.12}$$

where, the weighing function is $1/\sqrt{1-x^2}$

The chebyshev nodes in the interval (-1,1) are given by

$$x_j = cos\left(\frac{2j-1}{2n}\pi\right), j = 1, 2, ..., n \tag{2.13}$$

and the weights are all the same and equal to $\pi/n$

The implementation of this method can be found in the MATLAB function `GCheby`

### 2.4.2 Gauss-Hermite quadrature

The Gauss-Hermite quadrature is used to evaluate the approximate value of the integral of the kind

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx \tag{2.14}$$

where the weighing function is $W(x) = e^{-x^2}$

This method uses the *physicist's Hermite polynomials* that are given by,

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \tag{2.15}$$

The recurrence relation can be written as,

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x) \tag{2.16}$$

Note that, the leading coefficient of $H_n$ is $2^n$. In order to make the polynomial monic, it has to be divided by the same amount

$$\frac{H_{n+1}(x)}{2^{n+1}} = x\frac{H_n(x)}{2^n} - \frac{n}{2}\frac{H_{n-1}(x)}{2^{n-1}} \tag{2.17}$$

writing it in the form similar to Theorem 2.1,

$$p_{n+1}(x) = (x-0)p_n(x) - \frac{n}{2}p_{n-1}(x) \tag{2.18}$$

Hence, $a_n = 0$ and $b_n = n/2$. With these values, the tridiagonal Jacobi matrix can be generated and the nodes and weights can be computed as shown in (2.7) and (2.8).

The implementation of this procedure can be seen in the MATLAB function `GHerm`.

### 2.4.3 Gauss-Laguerre quadrature

The Gauss-Laguerre quadrature is used to evaluate the approximate value of the integral of the kind

$$\int_0^\infty e^{-x} f(x) dx \tag{2.19}$$

where the weighing function is $W(x) = e^{-x}$

This method uses the *Laguerre polynomials* defined by the formula,

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n}(e^{-x} x^n) \tag{2.20}$$

The recurrence relation can be written as

$$(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x) \tag{2.21}$$

The leading coefficient is again not 1. Making the relevant changes and rewriting it in the form of Theorem 2.1, we get $a_n = 2n+1$ and $b_n = n^2$. The corresponding Jacobi matrix can be constructed and the nodes and weights can be computed from (2.7) and (2.8).

This method has been implemented in the MATLAB function `GLagu`

### 2.4.4 Gauss-Legendre quadrature

The Gauss-Legendre quadrature is the simplest form where the weighing function, $W(x) = 1$. The related polynomials are called *Legendre polynomials*. They may be written using the formula,

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left((x^2-1)^n\right) \tag{2.22}$$

The recurrence relation can be written as,

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x) \tag{2.23}$$

From (2.18) it is clear that the leading coefficient is not 1. After rewriting the polynomials in monic form we get $a_n = 0$ and $b_n = n^2/(4n^2-1)$. The corresponding Jacobi matrix can be constructed and the nodes and weights can be computed from (2.7) and (2.8).

This method has been implemented in the MATLAB function `GLege`

### 2.4.5 Gauss-Lobatto quadrature

This procedure is similar to Gauss-Legendre with the key difference being, the interval end points are also considered as nodes. This leads to a slight modification in the tridiagonal Jacobi matrix as shown below.

$$J_n = \begin{pmatrix} a_0 & \sqrt{b_1} & & & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & & & \\ & \sqrt{b_2} & \ddots & \ddots & & & \\ & & \ddots & a_{n-2} & \sqrt{b_{n-1}} & & \\ & & & \sqrt{b_{n-1}} & a_{n-1} & \sqrt{b_n} & \\ & & & & \sqrt{b_n} & a_n & \sqrt{b_{n+1}} \\ & & & & & \sqrt{b_{n+1}} & a_{n+1} \end{pmatrix} \tag{2.24}$$

Compared to (2.7), a few extra terms can be observed. Also, $b_{n+1} = (n+1)/(2n+1)$ can be obtained. The rest of the procedure is as explained above.

This method has been implemented in the MATLAB function `GLoba`