Higher-order quadrature rules for data sets

Berkers, Abby

Award date:
2018

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Department of Mathematics and Computer Science

Higher-Order Quadrature Rules for Data Sets

A. Berkers
0951825

Supervisor:
ir. A.W. Eggels
dr. M.E. Hochstenbach

July 6, 2018

Contents

Chapter 1

Introduction

In this report we will discuss several approximations of the Riemann–Stieltjes integral [21]. Its formulation allows for both continuous and discrete distributions, and it defines the expectation over such a distribution or data set. These integrals are often used in computations that involve data sets. We will only discuss integrals in one dimension. Extending our methods to more dimensions is a topic for further research; this has a wide range of applications. Being able to approximate these type of integrals is needed when the function f is expensive to evaluate, i.e., the number of times it can be evaluated is limited by time or money. Oftentimes in practice, these functions will be computer simulations. Simulations are often used in multiple fields. One example from fluid dynamics is the aerodynamics when building an aircraft, see e.g., [19]. Another example, from stochastic simulation, is optimising traffic flow, see e.g., [5]. Furthermore, this type of integral has many applications in probability theory as the law of the unconscious statistician, see e.g., [16].

The approximations we will discuss will be quadrature rules, as the idea of constructing quadrature rules for data arose when working with Gauss quadrature rules, see [25, Sec. 5.7]. For quadrature rules, we will be evaluating the function in several points and assigning associated weights. Of course we are interested in doing this such that we obtain a good approximation of the Riemann–Stieltjes integral.

In Chapter 2 we will first state this more explicitly, and then we will discuss several known methods that can be useful. Among these methods are existing quadrature rules and Monte Carlo integration. In Chapter 3 we will describe our own proposed methods. Here, we will first describe methods to find the optimal points for function evaluations (representatives), and later the methods for finding the associated weights. In Chapter 4 we will discuss the performances of these methods, based on how well they approximate the Riemann–Stieltjes integral. Finally, in Chapter 5, we will give a short conclusion and a discussion regarding our work.

Chapter 2

Problem Description

The goal is to find an approximation of the *Riemann–Stieltjes integral* [21, p.298-330]

$$\mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(x)\,\mathrm{d}M_X(x)\,, \tag{2.1}$$

where $f(x)$ is a continuous function we wish to integrate over the data X, with X a one dimensional distribution. $M_X(x)$ is the *cumulative distribution function* of X and if $M_X(x)$ is continuously differentiable with $M_X'(x) = \mu_X(x)$, then $\mu_X(x)$ is the *probability density function* of X. When this exists, and $M_X(x)$ is continuously differentiable [23], we may write (2.1) as

$$\mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(x)\,\mathrm{d}M_X(x) = \int_{-\infty}^{\infty} f(x)\mu_X(x)\,\mathrm{d}x\,. \tag{2.2}$$

Let us assume that when $M_X(x)$ is known, then it is continuously differentiable and thus $\mu_X(x)$ exists and it is also known. Note that in our problem, the distribution of X might be unknown, or $\mu_X(x)$ or $f(x)$ might be difficult — or even impossible — to integrate analytically. The approximation we are looking for should be precise enough using a sufficiently large realisation of X. Let $\boldsymbol{x}$ be an arbitrary realisation of X of size n. The goal is then to find an approximation for (2.1) or (2.2) that will integrate a polynomial of degree k exactly. Since we are looking for a *quadrature rule* [17], this approximation will be of the form

$$\sum_{i=1}^{N} w_i f(z_i)$$

where the $z_i \in \boldsymbol{x}$ are *representatives* of the data with corresponding *weights* $w_i \geq 0$ and N the number of representatives used. Since evaluating f might be expensive, we wish to keep N small.

Because we are dealing with data, it seems intuitive that we are looking for a method that will represent this data, i.e., plotting the weights against the representatives will look somewhat like the density of the data. The methods we will propose in Section 3 will first use the given data to find representatives, and use these representatives to find the corresponding weights. We will first discuss several existing methods.

2.1 Existing Methods and Their Limitations

Of course there already exist methods that could be useful. We will go over these methods in the next sections, and some of our proposed methods in Section 3 will use the ideas of these methods.

2.1.1 Gaussian Quadrature Rules

Gaussian *quadrature rules* are approximations of the form

$$\int_a^b g(x)\,\mathrm{d}x = \int_a^b f(x)\,\omega(x)\,\mathrm{d}x \approx \sum_{i=1}^{N} w_i f(z_i)\,,$$

where $f(x)$ is a polynomial and $\omega(x) \geq 0$ is a *weight function* that generates a class of orthogonal polynomials [14]. In our case, $\omega(x)$ is the density function of our data X. Thus to use this method, the density must be known.

Definition 2.1 (Orthogonal polynomials)**.** A class of *orthogonal polynomials* is a set P of polynomials such that each pair of polynomials $p_n, p_m \in P$ in the set are orthogonal with respect to some inner product, see, e.g., [9]. This inner product depends on a non-decreasing function $W\colon \mathbb{R} \to \mathbb{R}$ and is defined by

$$(g,h) \coloneqq \int g(x)\,h(x)\,\mathrm{d}W(x)\,,$$

where $(\cdot,\cdot)$ denotes the inner product. Let $p_n \in P$ be of the form $p_n(x) = c_n \prod_{i=1}^{n}(x - r_i)$, with $c_n > 0$ a constant. Each $p_n \in P$ has n real roots $r_1 < r_2 < \cdots < r_n$. If P is a set of orthogonal polynomials, the following relations hold:

$$(p_i, p_j) = \begin{cases} 0 & \text{if } p_i \neq p_j\,, \\ 1 & \text{if } p_i = p_j\,, \end{cases} \qquad \text{for all } p_i, p_j \in P\,.$$

Because of the second relation, the polynomials are also *orthonormal polynomials*. This means that, besides the polynomials being orthogonal, they are also normalised. If we would only require that they are orthogonal, the second relation would require the inner

product to be nonzero. Notice that if W is continuously differentiable on its domain, and $\frac{\mathrm{d}}{\mathrm{d}x}W(x) = \omega(x)$, the inner product is given by

$$(p_1, p_2) \coloneqq \int p_1(x)\, p_2(x)\, \omega(x)\, \mathrm{d}x\,.$$

In this case ω is called a *weight function*.

The orthogonal polynomials can be used to compute the Gauss quadrature rules, as is done in [14]. The representatives are the roots of these orthogonal polynomials. The weights and the representatives are fixed, given a weight function $\omega(x)$. The Gaussian quadrature rules will be exact as long as f is a polynomial of degree at most $2N-1$ [25, Sec. 5.7]. We will now give two examples of specific Gaussian quadrature rules, and we will use them as a basis for the methods we propose. These quadrature methods will generate representatives that are most likely not in $\boldsymbol{x}$. Therefore, to be able to use these methods on data, we choose the data points that are closest to the representatives generated by the quadrature rules.

Gauss–Legendre Quadrature

When $\omega(x) = 1$ we have the *Gauss–Legendre quadrature rule*. To derive this quadrature rule we use the interval $[-1, 1]$, and this derivation is similar to the one in [25, Sec. 5.7]. Suppose we want to integrate a polynomial of degree three exactly. Since polynomials are made of *monomials*, e.g., $f(x) = x^3$, this implies that our approximation should be exact for all monomials with degree less than or equal to three. This gives us the following constraints:

$$\begin{aligned}
f(x) = 1: &\qquad \sum_{i=1}^{N} w_i = \int_{-1}^{1} 1\,\mathrm{d}x = 2\,,\\
f(x) = x: &\qquad \sum_{i=1}^{N} w_i\, z_i = \int_{-1}^{1} x\,\mathrm{d}x = 0\,,\\
f(x) = x^2: &\qquad \sum_{i=1}^{N} w_i\, z_i^2 = \int_{-1}^{1} x^2\,\mathrm{d}x = \frac{2}{3}\,,\\
f(x) = x^3: &\qquad \sum_{i=1}^{N} w_i\, z_i^3 = \int_{-1}^{1} x^3\,\mathrm{d}x = 0\,.
\end{aligned}$$

Take $N = 2$. By symmetry it holds that $w_1 = w_2 = 1$. Then $z_1 = -z_2$ and this gives a solution using the third or fourth equation. This can be extended to an arbitrary interval

$[a, b]$ by using a linear transformation. This gives

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}\left(w_1 f\left(\frac{b-a}{2}z_1 + \frac{b-a}{2}\right) + w_2 f\left(\frac{b-a}{2}z_2 + \frac{b-a}{2}\right)\right).$$

A third-degree polynomial can be integrated exactly using the two-point Gauss quadrature rule.

In general, we can integrate a polynomial of degree $2N-1$ exactly using N points. In that case it has to hold that

$$\sum_{i=1}^{N} w_i\, z_i^k = \int_a^b x^k\,\mathrm{d}x = \frac{a^{k+1} - b^{k+1}}{k+1}$$

for $k = 0, \ldots, 2N-1$. The corresponding quadrature rule is

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}\sum_{i=1}^{N} w_i f\left(\frac{b-a}{2}z_i + \frac{b-a}{2}\right).$$

Gauss–Hermite Quadrature

When integrating a polynomial over a distribution, we can choose $\omega(x)$ to be the probability density function of this distribution. For the *Gauss–Hermite quadrature rule* the weight function is e^{-x^2} [22], which we can transform to use these quadrature rules for a normal distribution. The class of orthogonal polynomials are the Hermite polynomials given by

$$H_N(x) = (-1)^N e^{x^2} \cdot \frac{\mathrm{d}^N}{\mathrm{d}x^N}\left(e^{-x^2}\right),$$

Then the Gauss–Hermite quadrature rule is given by

$$\int_{-\infty}^{\infty} f(x)\, e^{-x^2}\,\mathrm{d}x \approx \sum_{i=1}^{N} w_i\, f(x_i), \qquad w_i = \frac{\sqrt{\pi}\, 2^{N-1} N!}{N^2 (H_{N-1}(x_i))^2}, \tag{2.3}$$

where x_i are the roots of $H_N(x)$ [2, Sec. 25.4.46]. Let $Y \sim \mathcal{N}(\mu, \sigma^2)$ be a normally distributed with mean μ and variance σ^2. Then, using the transformation $x = \frac{y-\mu}{\sqrt{2}\,\sigma}$,

$$\int_{-\infty}^{\infty} f(y)\,\frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{\mu-y}{2\sigma^2}\right)^2}\,\mathrm{d}y = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty} f(\mu + \sqrt{2}\,\sigma x) e^{-x^2}\,\mathrm{d}x\,.$$

Finally, define the function $h(x) = f(\mu + \sqrt{2}\,\sigma x)$ and use (2.3) to obtain that

$$\int_{-\infty}^{\infty} f(y)\,\frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{\mu-y}{2\sigma^2}\right)^2}\,\mathrm{d}y \approx \frac{1}{\sqrt{\pi}}\sum_{i=1} w_i h(x_i)\,,$$

with w_i as in (2.3) and x_i the roots of $H_{N-1}(x)$. The $\sqrt{\pi}$ in w_i cancels with the one in the fraction, and thus

$$\mathbb{E}[f(Y)] \approx \sum_{i=1} w_i \, f(\mu + \sqrt{2}\,\sigma x_i)\,, \qquad\qquad w_i = \frac{2^{N-1}N!}{N^2(H_{N-1}(x_i))^2}\,.$$

Note that only the representatives depend on this specific distribution, since they are translated and scaled with the mean and standard deviation respectively. The weights only depend on the distribution in the sense that we can only do this for a normal distribution, but they are independent of the mean and the standard deviation.

Limitations When Applied to Data

When applying these quadrature rules to data, assumptions will have to be made about the probability density function of the underlying data. If it is possible to make an assumption about the data — for example by fitting a distribution — then the density function might still be difficult to use for constructing orthogonal polynomials. However, this is the technique proposed by [14], and for some weight functions the quadrature rules are already known, see e.g., [2, Sec. 25.4].

2.1.2 Monte Carlo

The *Monte Carlo integral* is defined by, see e.g., [7],

$$\int_{-\infty}^{\infty} f(x)\,\mu_X(x)\,\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)\,,$$

where $\mu_X(x)$ is the probability density function of X and $x_1, \ldots, x_n$ is a random sample of X.

Notice that this approximation can be viewed as a quadrature rule with n representatives, and each representative having corresponding weight $\frac{1}{n}$.

When f is expensive or difficult to evaluate, one would rather avoid n f-evaluations.

2.2 Summary

In this chapter we have stated our goal; find an approximation to $\mathbb{E}[f(X)]$ of the form $\sum_{i=1}^{N} w_i \, f(z_i)$. To do so, we need to find weights w_i and representatives z_i. This will be discussed in the next chapter.

Next, we explained two already existing methods. A Gaussian quadrature rule is a method to approximate an integral using weights and representatives — just as we are aiming for. The only difference is that these quadrature rules can be constructed for known

weight functions, see e.g, [14], but in our case the weight function is often unknown. Then we shortly discussed Monte Carlo integration, which is often used for approximating integrals like ours. However, when f is expensive to evaluate, Monte Carlo integration is not desirable due to its many evaluations of f.

In the next chapter we will explore ways of finding weights and representatives, sometimes using the ideas of the existing methods discussed in this chapter.

Chapter 3

Proposed Methods

As mentioned in the previous chapter, our approximation of (2.1) will be of the form $\sum_{i=1}^{N} w_i f(z_i)$. The methods described below are methods to find suitable values of the representatives and their corresponding weights. The method we propose consists of two phases. In the first phase, we will search for suitable representatives, and in the second phase we will search for weights corresponding to the representatives found in the first phase.

In Section 3.1 we propose methods to find representatives. These methods generally depend on the data, the number of representatives N we are interested in, and the order k of the polynomial we wish to integrate exactly. Note that often there is a relation between N and k, and that to integrate exactly, N will have to be at least $k+1$. We will see this in Section 3.2.2. Furthermore, $N \ll n$, with n the size of the data.

In Section 3.2 we propose methods to find weights corresponding to given representatives. These methods generally depend on the data, the representatives $\boldsymbol{z}$ (and thus N).

Finally, the methods of these two phases will be combined, giving one method that consists of two phases, where in each phase one of the discussed methods has been chosen. We will discuss their performances in Chapter 4.

3.1 Methods to find Representatives

Obviously, we will use the data $\boldsymbol{x}$ to find the representatives. Some of the methods proposed below make assumptions about the given data, e.g., it follows a certain distribution, and others do not.

When looking for representatives, it is intuitive to split our data into groups, and take the data point that is closest to the mean of each group as the representatives. We are looking for N representatives, so we have to split our data in N groups. To decide how we will split our data, we need a *penalty function* g, which depends on the representatives. For

this g, we want to find the arguments that minimise it, i.e., the representatives that take the lowest penalty under g. This penalty function can have different forms, and different approaches to this penalty function will be described below in the first two methods. The first is based on the assumption that the density function is known and builds a penalty function based on this theoretical distribution, rather than the actual data. The second will build a similar penalty function, yet this penalty function uses the data, and does not need an assumption about the distribution.

3.1.1 Using the Theoretical Distribution to derive a Minimisation Problem

We start with the idea of minimising the total squared distance to cluster centroids. Assume we know our data follows distribution X with known probability density function $\mu_X(x)$. Let $\boldsymbol{x}$ be an arbitrary realisation of X, of size n. Let $\boldsymbol{z} \in \mathbb{R}^N$ be the vector with representatives $(z_1, \ldots, z_N)^\mathsf{T}$, which are the variables for which we like to minimise the penalty function. For each $x \in \boldsymbol{x}$, define the function h as

$$h(x, \boldsymbol{z}) = \min_{z \in \boldsymbol{z}} |x - z|^2 = \begin{cases} (x - z_1)^2 & x \leq \frac{z_1 + z_2}{2} \\ \quad \vdots & \\ (x - z_i)^2 & \frac{z_{i-1} + z_i}{2} < x \leq \frac{z_i + z_{i+1}}{2}, i \in \{2, \ldots, N-1\} \\ \quad \vdots & \\ (x - z_N)^2 & x > \frac{z_{N-1} + z_N}{2} \end{cases} \tag{3.1}$$

to measure the distance of x to the closest representative. It is important to note that when choosing h this way, we assume that it is optimal to assign a data point $x \in \boldsymbol{x}$ to the cluster of the representative it is closest to. This does not have to be the case when splitting data into groups in general. Recall that $\mu_X(x)$ is known. We now can define the penalty function $g \colon \mathbb{R}^N \to \mathbb{R}$ as

$$g(\boldsymbol{z}) = \int_{-\infty}^{\infty} h(x, \boldsymbol{z})\, \mu_X(x)\, \mathrm{d}x\,. \tag{3.2}$$

We then are interested in the value of $\boldsymbol{z}$ that minimises g. The elements of $\boldsymbol{z}$ are taken from the data set $\boldsymbol{x}$, and thus there will always exist a $\boldsymbol{z}$ that gives the minimum value for $g(\boldsymbol{z})$. In particular, this leads to an optimisation problem in N variables which is very likely to involve complicated analytical expressions.

Example 3.1. Let $N = 3$ and $X \sim \mathcal{N}(0, 1)$, so

$$\mu(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,.$$

Note that the standard normal distribution is symmetric around 0, so when choosing three representatives it is natural to choose the middle representative as 0, with the first and the third representative mirrored around zero. This gives the choice $\boldsymbol{z} = (-z, 0, z)^{\mathsf{T}}$. Using this to simplify h we get

$$h(x,z) = \begin{cases} (x+z)^2 & \text{if } x < -\frac{z}{2} \\ x^2 & \text{if } -\frac{z}{2} \leq x \leq \frac{z}{2} \\ (x-z)^2 & \text{if } x > \frac{z}{2} \end{cases} .$$

Definition 3.2 (Error function)**.** Let $x \in \mathbb{R}$. The *error function* and the *complementary error function* are defined by, see e.g., [11, Chap. 3]

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t \,, \qquad \operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \, \mathrm{d}t \,,$$

respectively.

Now, the analytic expression for $g(z)$ is given by

$$g(z) = 1 - 2\sqrt{\frac{2}{\pi}} e^{-\frac{z^2}{8}} z + z^2 \operatorname{erfc}\left(\frac{z}{2\sqrt{2}}\right) ,$$

and its derivative is

$$g'(z) = -2\sqrt{\frac{2}{\pi}} e^{-\frac{z^2}{8}} + 2z \operatorname{erfc}\left(\frac{z}{2\sqrt{2}}\right) ,$$

by the product rule for derivatives. In Figure 3.1 we see that g has a minimum, and we will use the bisection method on $g'(z)$ to find the minimum. This is shown in Figure 3.2, and gives the minimum value for $z \approx 1.2240$.

The previous example shows that even if we choose a symmetric distribution and the number of representatives is small, this method is very likely to be complicated.

Example 3.3. Let $N = 3$ and $X \sim \mathcal{N}(0,1)$, as in the previous example. However, we will choose $\boldsymbol{z} = (z_1, z_2, z_3)^{\mathsf{T}}$ without the assumptions that $z_1 = -z_3$ and $z_2 = 0$. Then h will be

$$h(x,\boldsymbol{z}) = \begin{cases} (x-z_1)^2 & \text{if } x < \frac{z_1+z_2}{2} \\ (x-z_2)^2 & \text{if } \frac{z_1+z_2}{2} \leq x \leq \frac{z_2+z_3}{2} \\ (x-z_3)^2 & \text{if } x > \frac{z_2+z_3}{2} \end{cases} .$$

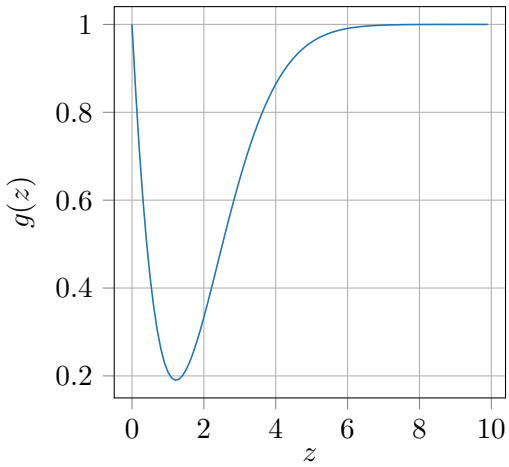


Figure 3.1: Plotting $g(z)$ on the interval $[0, 10]$ we see it has a minimum.

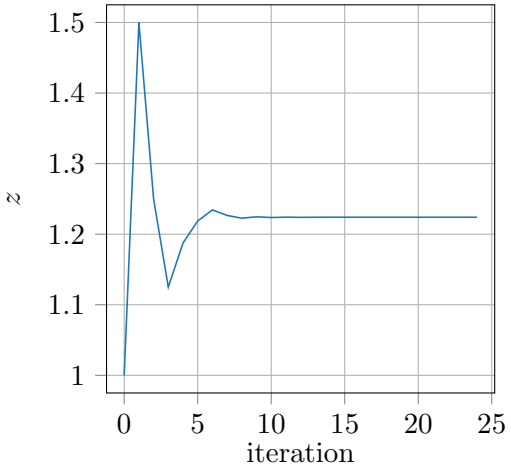


Figure 3.2: Using bisection on $g'(z)$ converges to the minimum.

Then $g\colon \mathbb{R}^3 \to \mathbb{R}$ is even uglier than in the previous example, and given by

$$g(\boldsymbol{z}) = \frac{1}{2}\left(\left(z_2^2 - z_1^2\right)\operatorname{erfc}\left(\frac{z_1+z_2}{2\sqrt{2}}\right) + \left(z_3^2 - z_2^2\right)\operatorname{erfc}\left(\frac{z_2+z_3}{2\sqrt{2}}\right)\right)$$
$$+ z_1^2 + \sqrt{\frac{2}{\pi}}e^{-\frac{1}{8}(z_1+z_2)^2}(z_1 - z_2) + \sqrt{\frac{2}{\pi}}e^{-\frac{1}{8}(z_2+z_3)^2}(z_2 - z_3) + 1\,.$$

This leads to a minimization problem in three dimensions. Minimizing this function (with Mathematica) gives $\boldsymbol{z} = (-1.22401, 4.57 \cdot 10^{-16}, 1.22401)^{\mathsf{T}}$, which is roughly the same result as in the previous example.

3.1.2 Gradient Descent

Computing the integral (3.2) from the previous section is complicated. If it is possible to compute this integral, it is likely that it is difficult to find the values of $\boldsymbol{z}$ that minimise it, at least analytically. Furthermore, when we do not know what the distribution of our data is, the previous method leads us nowhere. Using the same h as before, we define $\tilde{g}\colon \mathbb{R}^n \to \mathbb{R}$ as

$$\tilde{g}(\boldsymbol{z}) = \sum_{x \in \boldsymbol{x}} h(x, \boldsymbol{z})\,.$$

Note that this is an approximation of the previous g, where we use all our data without making an assumption about its distribution. As before, we wish to find the $\boldsymbol{z}$ that minimise

$\tilde{g}(\boldsymbol{z})$. We will do this using iterative Gradient Descent with the recursion [10]

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{z}^{(k)} - \frac{1}{n}\nabla g(\boldsymbol{z}^{(k)}) = \boldsymbol{z}^{(k)} - \frac{1}{n}\sum_{x\in\boldsymbol{x}} \nabla h(x, \boldsymbol{z}^{(k)}) \,,$$

where the gradient of h is

$$\nabla h(\boldsymbol{z}) = \begin{bmatrix} -2(x-z_1)\mathbb{1}\left\{x \leq \frac{z_1+z_2}{2}\right\} \\ \vdots \\ -2(x-z_i)\mathbb{1}\left\{\frac{z_{i-1}+z_i}{2} < x \leq \frac{z_i+z_{i+1}}{2}\right\} \\ \vdots \\ -2(x-z_N)\mathbb{1}\left\{x > \frac{z_{N-1}+z_N}{2}\right\} \end{bmatrix} ,$$

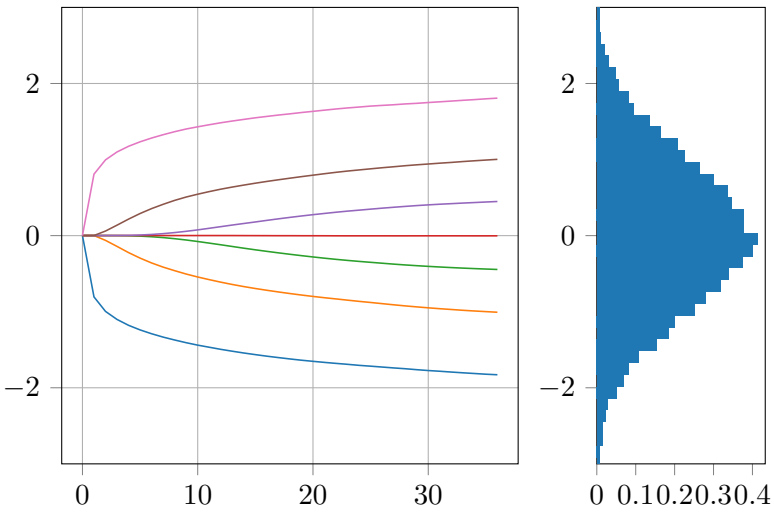
with $\mathbb{1}\{\cdot\}$ the *indicator function*. The step size $\frac{1}{n}$ is determined by trial. It should be small enough for the method to converge, but big large enough so the convergence is not too slow. The method converges when using step size $\frac{1}{n}$, though this is by no means the optimal step size. There might exist step sizes which yield faster convergence to a minimum. Note that $\boldsymbol{y}^{(k+1)}$ will most likely not consist of data points. Therefore we will choose $\boldsymbol{z}^{(k+1)}$ to be the data points that are closest to $\boldsymbol{y}^{(k+1)}$, so

$$\boldsymbol{y}^{(k+1)} = \arg\min_{x\in\boldsymbol{x}} \left|x - \boldsymbol{z}^{(k+1)}\right| .$$

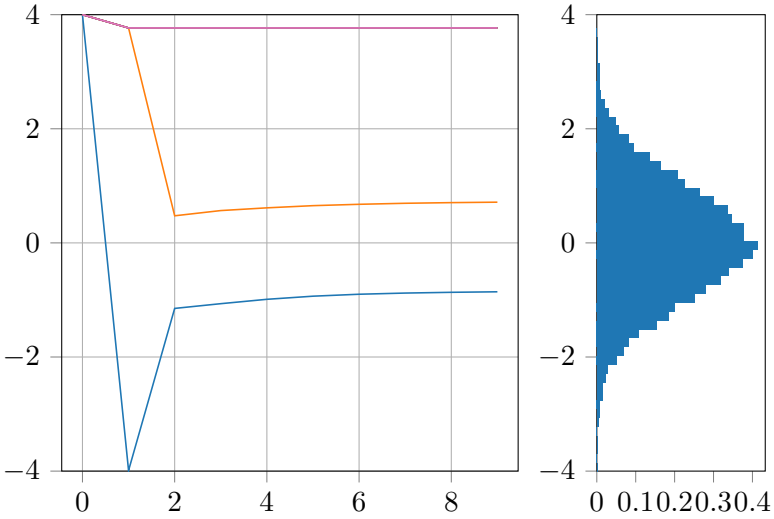
The stopping criterion is

$$\left|z_i^{(k+1)} - z_i^{(k)}\right| < \varepsilon_a \,, \quad \forall_{i\in\{1,\ldots,N\}} \,,$$

where ε_a is the accuracy we aim for. Note that with this criterion we might not end up at the minimum of f if we choose $\boldsymbol{z}^{(0)}$ somewhat random. However, we are probably able to set $\boldsymbol{z}^{(0)}$ to a good guess of the eventual $\boldsymbol{z}$. The advantage of this method is that this works for any distribution, as long as the initial guess $\boldsymbol{z}^{(0)}$ is good enough; see, e.g., Figure 3.3. In Figure 3.3a the initial $\boldsymbol{z}$ have been chosen as the mean of the data, and we see that the seven representatives spread out over the data. As there are more data points around the centre (in 0), one would expect that the representatives are closer to each other around the centre and this is indeed what happens. Now consider Figure 3.3b, where the representatives do not spread out over the data. In the last iteration, each $\boldsymbol{z}$ moved less than ε_a, and $g(\boldsymbol{z})$ evaluated in $\boldsymbol{z}^{(k)}$ and $\boldsymbol{z}^{(k+1)}$ hardly differs, thus the algorithm terminates. However, the minimum we have been looking for has not been reached. Instead we have reached another point $\boldsymbol{z}$ for which $g(\boldsymbol{z})$ does not differ more than $2\varepsilon_a$. Recalling Example 3.1 and Figure 3.1, the algorithm might have terminated for $z = 6$. This is obviously not a minimum, but the value of $f(z)$ for z in a neighbourhood of 6 differs only a little.



(a) Taking $\boldsymbol{z}^{(0)}$ as the mean of the data, for each point. The 7 points spread out over the data.



(b) Taking $\boldsymbol{z}^{(0)}$ as a point outside the tails of the data, in this case as 4. Two of the points end up actually representing the distribution, the other five stay at the end of the tail.

Figure 3.3: Different outcomes for different guesses of $\boldsymbol{z}^{(0)}$. On the left is the progress of $\boldsymbol{z}^{(i)}$ for each iteration. On the right is the histogram of the data the algorithm runs on, to get the idea of where the $\boldsymbol{z}$ are at each iteration.

3.1.3 Clustering

In the previous section we have been looking for the best representatives by minimising the sum of the squared distances to the closest representatives. A different way of finding the representatives that minimise this criterion is using *Lloyd's algorithm* [18] as described in [4], an implementation of *k-means*.

Let $\boldsymbol{y} = (y_1, \ldots, y_N)$ be given initial *centroids*, where a centroid is the centre of a cluster. Then the algorithm will switch between the following two phases until the clusters do not change anymore.

1. Given $\boldsymbol{y}$, assign each data point to the cluster of the closest representative. That is, the cluster C_i is the set of points $x \in \boldsymbol{x}$ that are closer to y_i than to y_j for all $j \neq i$.

2. Recalculate $y_i \in \boldsymbol{y}$ as the mean of each cluster C_i. Thus

$$y_i = \frac{1}{|C_i|} \sum_{c \in C_i} c \,.$$

The implementation [20] used here uses the *k-means++* algorithm [4] to choose the initial centroids, instead of choosing them arbitrarily. This improves the speed and accuracy of the algorithm [4].

In our implementation, we want the centroids to be data points. Therefore, we choose the representatives as

$$z_i = \arg\min_{x \in x} |x - y_i| \,, \text{for } i = 1, \ldots, N \,.$$

3.2 Methods to Find Corresponding Weights

Let us assume that we have found the representatives $\boldsymbol{z} = (z_1, \ldots, z_N)^\mathsf{T}$ with one of the methods from above, i.e., $\boldsymbol{z}$ is known. In this section we will propose methods that find weights corresponding to the given representatives. We will see three different methods. The first will simply take the normalised sizes of the clusters as weights. The second will impose constraints on the weights to ensure that the error is small. An extension of the second method will try to find the weights that satisfy these constraints, and are as close to the real weights (from the first method) as possible. The third and final method will base the weights on a polynomial approximation of f and the given data.

3.2.1 Cluster Size

Every data point $x \in \boldsymbol{x}$ can be assigned to a cluster. Let us assume this is done with one of the methods described in Section 3.1. Using one of these methods, we have found

representatives $z_1, \ldots, z_N$ and these can be used to construct the clusters $C_1, \ldots, C_N$ by assigning each $x \in \boldsymbol{x}$ to the cluster with the closest representative. Using the method in 3.1.3 we have found the clusters $C_1, \ldots, C_N$, in which case we do not have construct the clusters anymore.

When looking at these clusters, it is natural that each representative represents as much of the data as there are data points in its cluster. Let $|C_i|$ be the size of the cluster represented by z_i. Then the weights w_i will be computed by

$$w_i = \frac{|C_i|}{n},$$

where n is the size of the data $\boldsymbol{x}$. The weight w_i is the fraction of the data represented by z_i, and by construction it holds that

$$\sum_{i=1}^{N} w_i = 1.$$

Note that this method is rather fast. The running time of this method is $\mathcal{O}(nN)$, where $\mathcal{O}$ is the *big-O symbol* [6]. It has to do N comparisons for n points in the data set to assign each point to a cluster and determine the cluster size. When having found a cluster size, it has to do one operation, namely division by n.

3.2.2 Solving an Underdetermined System

If we want our method to be exact for polynomials of a certain degree, say k, then we have to impose constraints on either the weights or the representatives to ensure that this method is exact. Since we have used our representatives to represent our data, we will impose these constraints on the weights associated with these representatives.

Definition 3.4. We refer to a method as *exact* when the resulting approximation is equal to the approximation by Monte Carlo integration. That is

$$\sum_{i=1}^{N} w_i f(z_i) = \frac{1}{n} \sum_{x \in \boldsymbol{x}} x_i.$$

The data set $\boldsymbol{x}$ is all available information, and thus we cannot guarantee an approximation that is better than using all that information.

Similar as for the Gaussian quadrature rules explained in Section 2.1.1 we will create a system of equations. Suppose we wish to integrate a polynomial of degree k exactly ($k \geq 2$), then we want the integrals over the monomials 1, x, ..., x^k to be exact. To formulate the constraints that follow from this, we need the following definition.

Definition 3.5 (kth sample moment [1])**.** Let X be a random variable, and let $\boldsymbol{x}$ be a realisation of X, of size n. The *kth sample moment* $\widetilde{\mathbb{E}}\big[X^k\big]$ is defined by

$$\widetilde{\mathbb{E}}\big[X^k\big] \coloneqq \frac{1}{n}\sum_{x\in\boldsymbol{x}} x^k \approx \mathbb{E}\big[X^k\big]\,, \qquad k\in\mathbb{N}_0\,.$$

Now the constraints are

$$\begin{aligned}
f(x)=1: &\qquad \sum_{i=1}^{N} w_i = \int_{-\infty}^{\infty}\mu(x)\,\mathrm{d}x = 1 = \widetilde{\mathbb{E}}\big[X^0\big]\,,\\
f(x)=x: &\qquad \sum_{i=1}^{N} w_i z_i = \int_{-\infty}^{\infty} x\mu(x)\,\mathrm{d}x = \mathbb{E}[X] \approx \frac{1}{n}\sum_{x\in\boldsymbol{x}} x = \widetilde{\mathbb{E}}[X]\,,\\
&\;\;\vdots\\
f(x)=x^k: &\qquad \sum_{i=1}^{N} w_i z_i^k = \int_{-\infty}^{\infty} x^k\mu_X(x)\,\mathrm{d}x = \mathbb{E}\big[X^k\big] \approx \frac{1}{n}\sum_{x\in\boldsymbol{x}} x^k =: \widetilde{\mathbb{E}}\big[X^k\big]\,.
\end{aligned}$$

We already have found representatives, so the z_i are fixed and we want to find the weights w_i by solving this system of equations. Writing this as a matrix system gives $\boldsymbol{Z}\boldsymbol{w}=\boldsymbol{b}$ with

$$\boldsymbol{Z} = \begin{bmatrix} 1 & \cdots & 1\\ z_1 & \cdots & z_N\\ \vdots & & \vdots\\ z_1^k & \cdots & z_N^k \end{bmatrix}\,, \qquad \boldsymbol{b} = \begin{bmatrix} 1\\ \widetilde{\mathbb{E}}[X]\\ \vdots\\ \widetilde{\mathbb{E}}\big[X^k\big] \end{bmatrix}\,. \tag{3.3}$$

From now on when we talk about the matrix $\boldsymbol{Z}$, we mean the $\boldsymbol{Z}$ as in (3.3). Note that $\boldsymbol{Z}$ is a transposed *Vandermonde matrix*, with the following definition of such a matrix, see e.g., [8].

Definition 3.6 (Vandermonde matrix)**.** Let $\boldsymbol{A}\in\mathbb{R}^{N\times k+1}$. $\boldsymbol{A}$ is called a *Vandermonde matrix* if

$$\boldsymbol{A} = \begin{bmatrix} 1 & z_1 & \cdots & z_1^k\\ 1 & z_2 & \cdots & z_2^k\\ \vdots & \vdots & & \vdots\\ 1 & z_N & \cdots & z_N^k \end{bmatrix}\,.$$

The determinant of this matrix is given by [8]

$$\det(\boldsymbol{A}) = \det(\boldsymbol{A}^{\mathsf{T}}) = \prod_{i<j}(z_j - z_i)\,.$$

Note that the Vandermonde matrix and its transpose are only nonsingular when $z_i \neq z_j$ for all $i \neq j$. This means that when there are two or more identical representatives, there does not exist a solution to (3.3). When we take $N = k + 1$ we get $\boldsymbol{Z} \in \mathbb{R}^{N \times N}$, and there exists exactly one solution. However, we cannot guarantee that $\boldsymbol{w} \geq 0$ element-wise. Therefore we choose $N > k + 1$ and solve the underdetermined system. This does not guarantee that $\boldsymbol{w} \geq \mathbf{0}$; we will get back to this later. First, we will show the following result.

Lemma 3.7. Let f be a polynomial of degree k. Then determining the weights solving the system in (3.3) will give the same final result for the approximation of the integral as Monte Carlo integration, and thus this method is exact. That is

$$\sum_{i=1}^{N} w_i f(z_i) = \frac{1}{n} \sum_{i=1}^{n} f(x_i)\,,$$

where $\boldsymbol{z}$ are computed using any method and the corresponding weights $\boldsymbol{w}$ are computed solving (3.3).

Proof. Let $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_k x^k$. This gives

$$\begin{aligned}\frac{1}{n} \sum_{i=1}^{n} f(x_i) &= \frac{1}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i + \cdots + \beta_k x_i^k) \\ &= \beta_0 + \beta_1 \frac{1}{n} \sum_{i=1}^{n} x_i + \cdots + \beta_k \frac{1}{n} \sum_{i=1}^{n} x_i^k\,.\end{aligned}$$

By construction it holds that $\frac{1}{n} \sum_{i=1}^{n} x_i^j = \sum_{i=1}^{N} w_i z_i^j$, for $j = 0, 1, \ldots, k$. Using this gives

$$\begin{aligned}\beta_0 + \beta_1 \frac{1}{n} \sum_{i=1}^{n} x_i + \cdots + \beta_k \frac{1}{n} \sum_{i=1}^{n} x_i^k &= \beta_0 \sum_{i=1}^{N} w_i + \beta_1 \sum_{i=1}^{N} w_i z_i + \cdots + \beta_k \sum_{i=1}^{N} w_i z_i^k \\ &= \sum_{i=1}^{n} w_i f(z_i)\,.\end{aligned}$$

And thus $\frac{1}{n} \sum_{i=1}^{n} f(x_i) = \sum_{i=1}^{N} w_i f(z_i)$. □

Finding a Positive Solution

We can use the algorithm as outlined in [12] to find a positive solution to a system of the form $\boldsymbol{Z}\boldsymbol{x} = \mathbf{0}$, if one exists. From now on, we will refer to this algorithm with *Dines' algorithm*. We will elaborate on Dines' algorithm and why it works below, but first we will

rewrite our system $\boldsymbol{Zw} = \boldsymbol{b}$ so it is of the from $\widetilde{\boldsymbol{z}}\widetilde{\boldsymbol{w}} = \boldsymbol{0}$. To do this, let us take the vector $\boldsymbol{b}$ to the left-hand side, and add a 1 to the end of $\boldsymbol{w}$. This gives the system

$$\begin{bmatrix} 1 & \cdots & 1 & -1 \\ z_1 & \cdots & z_N & -\widetilde{\mathbb{E}}[X] \\ \vdots & & \vdots & \vdots \\ z_1^k & \cdots & z_N^k & -\widetilde{\mathbb{E}}\big[X^k\big] \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ 1 \end{bmatrix} = \boldsymbol{0}\,. \tag{3.4}$$

This is similar to solving the system

$$\widetilde{\boldsymbol{z}}\widetilde{\boldsymbol{w}} = \begin{bmatrix} 1 & \cdots & 1 & -1 \\ z_1 & \cdots & z_N & -\widetilde{\mathbb{E}}[X] \\ \vdots & & \vdots & \vdots \\ z_1^k & \cdots & z_N^k & -\widetilde{\mathbb{E}}\big[X^k\big] \end{bmatrix} \begin{bmatrix} \widetilde{w}_1 \\ \vdots \\ \widetilde{w}_N \\ \widetilde{w}_{N+1} \end{bmatrix} = \boldsymbol{0}\,, \tag{3.5}$$

with the requirement that $\widetilde{w}_{N+1} \neq 0$.

Claim 3.8. Finding a positive solution to (3.5) gives us a positive solution to (3.4) and thus to the original system as in (3.3).

Proof. Let $\widetilde{\boldsymbol{w}}^+$ be a positive solution to (3.5), thus each element of $\widetilde{\boldsymbol{w}}^+$ is strictly greater than 0. Now take $\boldsymbol{w}^+ = \frac{1}{\widetilde{w}^+_{N+1}}\widetilde{\boldsymbol{w}}^+$, this is a scalar-vector multiplication, so

$$\widetilde{\boldsymbol{z}}\boldsymbol{w}^+ = \widetilde{\boldsymbol{z}}\frac{1}{\widetilde{w}^+_{N+1}}\widetilde{\boldsymbol{w}}^+ = \frac{1}{\widetilde{w}^+_{N+1}}\widetilde{\boldsymbol{z}}\widetilde{\boldsymbol{w}}^+ = \boldsymbol{0}\,.$$

Thus $\boldsymbol{w}^+$ is a positive solution to (3.4), and the first N elements of $\boldsymbol{w}^+$ are a positive solution to the original system. □

The idea of Dines' algorithm is to reduce the $k+1$ equations to a single equation by first reducing to k equations, then to $k-1$ equations, and so on, until we are left with one equation. In every step of this process we can check if a positive solution still exists. If it does, the algorithm continues, and if a positive solution does not exist, it will terminate. Then there exists a positive solution to the final single equation if and only if there exists a positive solution to the original system of equations, and we can use the positive solution to the final equation to construct a solution to the original system.

In the following paragraphs we will discuss how the reduction works, how to determine if a positive solution still exists, and how to recover a solution.

Reducing m equations to $m-1$ equations [12] Given a system with m equations of l variables,

$$\sum_{j=1}^{l} a_{ij}x_i = 0\,, \qquad i = 1,\dots,m\,,$$

we will show how to reduce these m equations to $m-1$ equations. While doing this (on the left), we will also perform the steps at the $k+1$ equations of N variables of the system in (3.5) (on the right).

To begin, we write the first equation in a different form. Let $P \subset \{1,\dots,l\}$ be the set of indices such that $a_{1p}, p \in P$ is positive, and similarly define Q as the set of negative indices. We then write

$$\sum_{p\in P} a_{1p}x_p = -\sum_{q\in Q} a_{1q}x_q\,, \qquad\qquad \sum_{i=1}^{N} \widetilde{w}_i = \widetilde{w}_{N+1}\,. \tag{3.6}$$

Using the same indices, the other $m-1$ equations are written as

$$\sum_{q\in Q} a_{rq}x_q = -\sum_{p\in P} a_{rp}x_p\,, \qquad\qquad -\widetilde{\mathbb{E}}[X^s]\widetilde{w}_{N+1} = -\sum_{i=1}^{N} z_i^s\widetilde{w}_i\,, \tag{3.7}$$

with $r = 2,\dots,m$ and $s = 1,\dots,k$. Now we obtain $m-1$ equations by multiplying the left side of (3.6) with the left side of (3.7), and the right side of (3.6) with the right side of (3.7), for $r = 2,\dots,m$. This gives the equations

$$\Big(\sum_{p\in P} a_{1p}x_p\Big)\Big(\sum_{q\in Q} a_{rq}x_q\Big) = \Big(\sum_{q\in Q} a_{1q}x_q\Big)\Big(\sum_{p\in P} a_{rp}x_p\Big) \tag{3.8}$$

with $r = 2,\dots,m$ and for our system

$$-\sum_{i=1}^{N} \widetilde{w}_i \cdot \widetilde{\mathbb{E}}[X^s] = -\sum_{i=1}^{N} z_i^s\widetilde{w}_i\,,$$

with $s = 1,\dots,k$. Rewriting (3.8) gives

$$\sum_{p\in P}\sum_{q\in Q} a_{1p}a_{rq}x_px_q = \sum_{q\in Q}\sum_{p\in P} a_{1q}a_{rp}x_qx_p\,.$$

Let $j = pq \in P\times Q$, and define for notation $\sum_j = \sum_p\sum_q$ and $x_j = x_px_q$ for some $p\in P, q\in Q$. Using this to rewrite the previous equation gives

$$\sum_j a_{1p}a_{rq}x_j = \sum_j a_{1p}a_{rp}x_j\,.$$

Bringing the right-hand side to the left-hand side we have

$$\sum_j (a_{1p}a_{rq} - a_{1q}a_{rp})x_j = 0\,, \tag{3.9}$$

for $r = 2, \ldots, m$. In our example we have

$$\sum_{i=1}^{N} (z_i^s - \widetilde{\mathbb{E}}[X^s])\widetilde{w}_i\widetilde{w}_{N+1} = 0\,, \qquad s = 1, \ldots, k\,. \tag{3.10}$$

We now have $m-1$ equations in the variables x_j for $j \in P \times Q$. Observe that we have $|P \times Q| = |P| \times |Q| \geq n-1$ variables, given that we had n variables in the previous system of equations. We can do this reduction again to obtain a system of $m-2$ equations, and we can continue doing this reduction until there one final equation left.

Checking if a Positive Solution Exists After every reduction step, we will check if a positive solution exists. In the solution, all $\widetilde{w}_i$ have to be positive. It is easy to see that if in one of the equations all the coefficients have the same sign, a positive solution does not exist. Thus to check if a positive solution does exist, we check that every equation has at least one sign disagreement.

Recovering the Solution Let the solution $\overline{x}_j = \overline{x}_{pq}$ of $m-1$ equations be given. By the reduction, the system of $m-1$ equations is of the form as in (3.9). Rewriting this equation gives

$$\sum_{q\in Q} a_{rq}\Big(\sum_{\in P} a_{qp}\overline{x}_{pq}\Big) + \sum_{p\in P} a_{rp}\Big(\sum_{q\in Q} a_{1q}\overline{x}_{pq}\Big)\,.$$

Substituting

$$\overline{x}_p = \sum_{q\in Q} a_{1q}\overline{x}_{pq}\,, \qquad \overline{x}_q = \sum_{p\in P} a_{1p}\overline{x}_{pq} \tag{3.11}$$

in the previous equation gives

$$\sum_{p\in P} a_{rp}\overline{x}_p + \sum_{q\in Q} a_{rq}\overline{x}_q = \sum_{i=1}^{n} a_{ri}\overline{x}_i = 0\,,$$

with $r = 1, \ldots, m$, and n the number of variables in these equations. And thus the expression in (3.11) is a solution to the m equations.

In our example, recovering the solution to the $k+1$ equations given the solution to the k equations is rather simple. Given a solution $\widetilde{w}_{i(N+1)} = \widetilde{w}_i \widetilde{w}_{N+1}$ for $i = 1, \ldots, N$, we obtain the solution to the original system by

$$w_i = \frac{\widetilde{w}_{i(N+1)}}{\widetilde{w}_{N+1}} = \widetilde{w}_i \,.$$

This is similar to setting $\widetilde{w}_{N+1} = 1$.

To get a solution for the system of $m-1$ equations, we start with solving the final single equation. From there we use the recovery from above to obtain a solution to the system of two equations. We can apply the recovery to the solution for the current system until we reach the system of $m-1$ equations.

Let the final equation be given by (3.9), for $r = m = 2$, so

$$\sum_j (a_{1p}a_{2q} - a_{1q}a_{2p})x_j \,.$$

Let J_p be the set of indices for which $a_{1p}a_{2q} - a_{1q}a_{2p} \geq 0$, and let J_q be the set of indices with $a_{1p}a_{2q} - a_{1q}a_{2p} < 0$. Then the solution to this system is

$$\begin{cases} x_j = \sum_{i \in J_q} a_{1p}a_{2q} - a_{1q}a_{2p} & \text{if } j \in J_p \\ x_j = \sum_{i \in J_p} a_{1p}a_{2q} - a_{1q}a_{2p} & \text{if } j \in J_q \end{cases} .$$

General Algorithm to find the Minimum Number of Representatives when using Dines Now we have a method that will find a positive solution if it exists, we can use this to find suitable weights when we wish to integrate a polynomial of degree k exactly. We can use Algorithm 1 to always find representatives and corresponding weights, given data and the degree k. There are situations where Dines' algorithm will never find a positive solution

Algorithm 1 Using Dines' algorithm to find weights.

```
procedure DinesFindWeights(x, k)
    N ← k + 1
    repeat
        find z_1, ..., z_N using one of the methods described in Section 3.1
        build a matrix z as in (3.5)
        use Dines' algorithm to find corresponding weights, if they exist
        N ← N + 1
    until Dines' algorithm found positive solution or N ≥ U
end procedure
```

(because it does not exist). Recall that the determinant of the Vandermonde matrix is given

by $\det(\boldsymbol{Z}) = \prod_{i<j}(z_j - z_i)$. Thus if there are non-distinct representatives, there does not exist a solution. Because of this, it is useful to include an upper bound $U < n < \infty$ for the number of representatives one would try, to make sure that the algorithm does not run forever.

So far we have only discussed finding a positive solution to the underdetermined system, using Dines' algorithm. As one would expect, this probably does not assign the weights to a cluster that correspond to the actual cluster sizes, see Figure 3.4. We wonder if it is possible to come closer to these 'real' weights while retaining that the weights are a solution to our system. This means that we will look into the solution space of our system, to see if there are weights that better represent this data. Note that this should not change the eventual approximation, since the requirements on the weights do not change. This is only to see how close we can get to the 'real' weights.

Minimisation Problem to find Intuitive Positive Solution

Given a matrix $\boldsymbol{Z}$ and a vector $\boldsymbol{b}$ as in (3.3) we have seen that we can find a positive solution using Dines' algorithm, if one exists. Let us now use this solution to find a more intuitive one. To do this, we need the following definitions from linear algebra, see e.g., [13].

Definition 3.9 (Null space)**.** Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. The *null space* of $\boldsymbol{A}$ is the set of vectors $\boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{y} \neq \boldsymbol{0}$ for which it holds that $\boldsymbol{Ay} = \boldsymbol{0}$. We denote the null space of $\boldsymbol{A}$ by $\mathcal{N}(\boldsymbol{A})$.

Definition 3.10 (Null basis)**.** Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, with $m < n$. Then there exists a matrix $\boldsymbol{B}$ such that the columns of $\boldsymbol{B}$ span $\mathcal{N}(\boldsymbol{A})$. This matrix $\boldsymbol{B}$ is called the *null basis* of $\boldsymbol{A}$.

Definition 3.11 (Rank)**.** Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. The *rank* of $\boldsymbol{A}$ is the *row rank* or the *column rank rank! column*, where the row rank is the number of independent rows of $\boldsymbol{A}$ and the column rank is the number of independent columns of $\boldsymbol{A}$. We denote the rank of $\boldsymbol{A}$ by $\operatorname{rank}(\boldsymbol{A})$. Note that in particular, $\operatorname{rank}(\boldsymbol{A}) \leq \min\{m, n\}$.

Definition 3.12 (Vector norm)**.** A function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is called a *vector norm* if it satisfies

(i) $\|\boldsymbol{x}\| \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\|\boldsymbol{x}\| = 0 \iff \boldsymbol{x} = \boldsymbol{0}$,

(ii) $\|\alpha\boldsymbol{x}\| = |\alpha| \cdot \|\boldsymbol{x}\|$ for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$,

(iii) $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}$.

The norm is a way to define the distance between vectors, as the absolute value is only defined on scalars. In particular, the *2-norm* is given by

$$\|\boldsymbol{x}\|_2 = \sqrt{\textstyle\sum_{i=1}^n x^2} = \sqrt{(\boldsymbol{x}, \boldsymbol{x})}\,,$$

where $(\boldsymbol{x}, \boldsymbol{x})$ is the inner product of $\boldsymbol{x}$ with itself.

Let us now define the minimisation problem. Let C_i be the set of points $x \in \boldsymbol{x}$ that have the point z_i as closest representative, so

$$C_i = \{x \in \boldsymbol{x} \mid \arg\min_{z \in \boldsymbol{z}} |z - x| = z_i\},$$

and $\boldsymbol{c} \in \mathbb{R}^N$ be the vector $(|C_1|, |C_2|, \ldots, |C_N|)^\mathsf{T}$. The goal is to minimise $\|\boldsymbol{w} - \boldsymbol{c}\|_2^2$, with $\boldsymbol{w} \in \mathbb{R}^N$ a solution to the system $\boldsymbol{Z}\boldsymbol{w} = \boldsymbol{b}$ and furthermore $\boldsymbol{w} \geq 0$ element-wise. Recall that in the previous section we found that if a positive solution exists, it is possible to find a positive solution with Dines' algorithm. Without loss of generality, let us assume such a solution exists and call this solution $\boldsymbol{u}$. Then we want to find $\boldsymbol{v} \in \mathcal{N}(\boldsymbol{Z})$ that minimises $\|\boldsymbol{u} + \boldsymbol{v} - \boldsymbol{c}\|_2^2$, $\boldsymbol{u} + \boldsymbol{v} \geq 0$ element-wise. Define the function $f\colon \mathbb{R}^N \to \mathbb{R}$ by $f(\boldsymbol{v}) = \|v - (c - u)\|_2^2 = \sum_{i=1}^N (v_i - (c_i - u_i))^2$. Note that it holds that $f(\boldsymbol{v}) \geq \boldsymbol{0}$ for all $\boldsymbol{v} \in \mathbb{R}^N$. Furthermore, $\boldsymbol{c}$ and $\boldsymbol{u}$ are known vectors, so $\boldsymbol{c} - \boldsymbol{u}$ is a known, and fixed, vector in $\mathbb{R}^N$. Let $\boldsymbol{B}$ be the null basis of $\boldsymbol{Z}$. Then the minimisation problem is

$$\underset{\boldsymbol{v} \in \langle \boldsymbol{B} \rangle}{\text{minimise}}\, f(\boldsymbol{v}), \qquad u_i + v_i \geq 0, \text{ for } i = 1, \ldots, N.$$

where $\langle \boldsymbol{B} \rangle$ denotes the space spanned by the columns of $\boldsymbol{B}$. Let $V \subset \langle \boldsymbol{B} \rangle$ be the set of allowed solutions in the space spanned by $\boldsymbol{B}$. The structure of the method we propose to solve this problem with is given in Algorithm 2. We will explain these steps further in the following paragraphs.

Algorithm 2 Use the solution found by Dines' algorithm to find more intuitive weights.

1: **procedure** MINIMISE($\boldsymbol{Z}$, $\boldsymbol{u}$)
2: Find $\boldsymbol{B}$ such that $\langle \boldsymbol{B} \rangle = \mathcal{N}(\boldsymbol{Z})$.
3: $\boldsymbol{c}' \leftarrow$ projection of $\boldsymbol{c} - \boldsymbol{u}$ on $\langle \boldsymbol{B} \rangle$.
4: Find the point on $V \subset \langle \boldsymbol{B} \rangle$ closest to $\boldsymbol{c}'$, where $\boldsymbol{v}$ satisfies $\boldsymbol{u} + \boldsymbol{v} \geq \boldsymbol{0}$ element-wise for all $\boldsymbol{v} \in V$.
5: **end procedure**

Finding $\boldsymbol{B}$ The columns of $\boldsymbol{B}$ are the vectors $\boldsymbol{v}$ that satisfy $\boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0}$, by the definition of the null space.

Lemma 3.13. Let $\boldsymbol{v} \in \mathbb{R}^N$. If $\boldsymbol{Z}^\mathsf{T}\boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0}$, then also $\boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0}$.

Proof. Let us assume that $\boldsymbol{Z}^\mathsf{T}\boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0}$ and $\boldsymbol{Z}\boldsymbol{v} \neq \boldsymbol{0}$. This implies that $\boldsymbol{y} = \boldsymbol{Z}\boldsymbol{v}$ is a vector for which it holds that $\boldsymbol{Z}^\mathsf{T}\boldsymbol{y} = \boldsymbol{0}$. Recall that $\boldsymbol{Z}^\mathsf{T}$ is a Vandermonde matrix, and thus $\boldsymbol{Z}^\mathsf{T}$ has full column rank. This means that the columns of $\boldsymbol{Z}^\mathsf{T}$ are linearly independent, and so the only vector $\boldsymbol{y}$ that can satisfy $\boldsymbol{Z}^\mathsf{T}\boldsymbol{y} = \boldsymbol{0}$ is the zero vector. The result is that

$\boldsymbol{Z}^\mathsf{T}\boldsymbol{y} = \boldsymbol{0} \implies \boldsymbol{y} = \boldsymbol{0}$, and this is in contradiction with the assumption that $\boldsymbol{y} = \boldsymbol{Z}\boldsymbol{v} \neq \boldsymbol{0}$. And thus

$$\boldsymbol{Z}^\mathsf{T}\boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0} \implies \boldsymbol{Z}\boldsymbol{v} = \boldsymbol{0}\,.$$

□

The previous lemma enables us to find the columns of $\boldsymbol{B}$ by finding the eigenvectors to the eigenvalue 0 for the matrix $\boldsymbol{Z}^\mathsf{T}\boldsymbol{Z}$. Note that, because $\boldsymbol{Z}$ has full row rank, the geometric multiplicity of the eigenvalue 0 of $\boldsymbol{Z}^\mathsf{T}\boldsymbol{Z}$ is $N-(k+1)$. By the rank-nullity theorem it holds that $\operatorname{rank}(\boldsymbol{Z}) + \operatorname{nul}(\boldsymbol{Z}) = N$, where $\operatorname{nul}(\boldsymbol{Z})$ is the dimension of the null space of $\boldsymbol{Z}$. $\boldsymbol{Z}$ has full row rank, and thus $\operatorname{rank}(\boldsymbol{Z}) = k+1$. Obviously the dimension of the null space is then $N-(k+1)$, and as the number of columns in $\boldsymbol{B}$ is $N-(k+1)$, $\langle \boldsymbol{B} \rangle$ will span the null space of $\boldsymbol{Z}$. Now we have a representation of the null space, we can use this to create a projection onto the null space.

Projecting $\boldsymbol{c}-\boldsymbol{u}$ on $\langle \boldsymbol{B} \rangle$ Let $\boldsymbol{c}'$ be the projection of $\boldsymbol{c}-\boldsymbol{u}$ onto $\langle \boldsymbol{B} \rangle$. Then $\boldsymbol{c}'$ is a linear combination of the columns of $\boldsymbol{B}$, so

$$\boldsymbol{c}' = \sum_{i=1}^{N-(k+1)} \beta_i \boldsymbol{b}_i\,,$$

where $\boldsymbol{b}_i$ is the ith column of $\boldsymbol{B}$, and $\beta_i \in \mathbb{R}$ the length of $\boldsymbol{c}-\boldsymbol{u}$ along $\boldsymbol{b}_i$. Recall from linear algebra the *inner product*, which satisfies

$$(\boldsymbol{b}_i, \boldsymbol{c}-\boldsymbol{u}) = \|\boldsymbol{b}_i\|_2 \, \|\boldsymbol{c}-\boldsymbol{u}\|_2 \cos(\theta)\,,$$

where θ is the angle between $\boldsymbol{b}_i$ and $\boldsymbol{c}-\boldsymbol{u}$. Dividing this by $\|\boldsymbol{b}_i\|$ gives

$$\frac{(\boldsymbol{b}_i, \boldsymbol{c}-\boldsymbol{u})}{\|\boldsymbol{b}_i\|_2} = \|\boldsymbol{c}-\boldsymbol{u}\|_2 \cos(\theta) =: \beta_i\,,$$

which is exactly the length of $\boldsymbol{c}-\boldsymbol{u}$ in the direction of $\boldsymbol{b}_i$.

Finding closest point to $\boldsymbol{c}'$ To argue that finding the closest point to $\boldsymbol{c}'$ gives the minimises f, we need that f is convex.

Claim 3.14. $f(\boldsymbol{v})$ is *convex*, that is, V is a *convex set* and

$$\forall_{\boldsymbol{v}_1, \boldsymbol{v}_2 \in V} \forall_{t \in [0,1]} \colon f(t\,\boldsymbol{v}_1 + (1-t)\,\boldsymbol{v}_2) \leq t f(\boldsymbol{v}_1) + (1-t) f(\boldsymbol{v}_2)\,.$$

When this inequality holds with $<$ instead of $\leq$, then we say that f is *strictly convex*.

Proof. A set is a convex set if for any two points $\boldsymbol{v}, \boldsymbol{y}$ in the set, the segment from $\boldsymbol{v}$ to $\boldsymbol{y}$ lies completely in the set. The set V is a subset of $\mathbb{R}^N$ and in each dimension consists of a halfplane, thus it is convex.

Let $g\colon \mathbb{R} \to \mathbb{R}$ be $g(x) = x^2$ and let $l\colon \mathbb{R}^N \to \mathbb{R}$ be $l(\boldsymbol{v}) = \|\boldsymbol{v}\|_2$. We will first show that both $g(x)$ and $l(\boldsymbol{v})$ are convex, and then that $(g \circ l)(\boldsymbol{v}) = f(\boldsymbol{v})$ is convex.

- $g(x) = x^2$. The second derivative $g''(x) = 2 > 0$, so g is strictly convex [24, Chap. 2].
- $l(\boldsymbol{v}) = \|\boldsymbol{v}\|_2$. Take $\boldsymbol{v}, \boldsymbol{y} \in \mathbb{R}^N$ arbitrary. By the properties of the 2-norm it holds that

  $$\|\alpha \boldsymbol{v}\|_2 = |\alpha| \, \|\boldsymbol{v}\|_2 \qquad \text{and} \qquad \|\boldsymbol{v} + \boldsymbol{y}\|_2 \leq \|\boldsymbol{v}\|_2 + \|\boldsymbol{y}\|_2 \ ,$$

  for all $\alpha \in \mathbb{R}$. Now for all $t \in [0, 1]$ it holds that

  $$\|t\boldsymbol{v} + (1-t)\boldsymbol{y}\|_2 \leq \|t\boldsymbol{v}\|_2 + \|(1-t)\boldsymbol{y}\|_2 = t \, \|\boldsymbol{v}\|_2 + (1-t) \, \|\boldsymbol{y}\|_2 \ .$$

Note that $l(\boldsymbol{v}) \geq$ for all $\boldsymbol{v} \in \mathbb{R}^N$ and that for $x \geq 0$ g is non-decreasing. Now for the composition $(g \circ l)(\boldsymbol{v}) = f(\boldsymbol{v}) = \|\boldsymbol{v}\|_2^2$ we have that

$$\begin{aligned} \|t\boldsymbol{v} + (1-t)\boldsymbol{y}\|_2^2 &= (g \circ l)(t\boldsymbol{v} + (1-t)\boldsymbol{y}) \\ &\leq g(tl(\boldsymbol{v}) + (1-t)l(\boldsymbol{y})) \qquad (l \text{ convex}, g \text{ non-decreasing on } [0, \infty)) \\ &< tg(l(\boldsymbol{v})) + (1-t)g(l(\boldsymbol{y})) \qquad (g \text{ strictly convex}) \\ &= t \, \|\boldsymbol{v}\|_2^2 + (1-t) \, \|\boldsymbol{y}\|_2^2 \ . \end{aligned}$$

Thus $f(\boldsymbol{v})$ is strictly convex. □

Note that $f(\boldsymbol{v})$ is also convex on $\mathbb{R}^N$, as $\mathbb{R}^N$ is also a convex set.

Remark 3.15. Since f is strictly convex on its domain $\mathbb{R}^N$ it has exactly one minimum, and so this is a global minimum [24, Chap. 2].

Let $\boldsymbol{v} \in V$ be the unique minimiser of f on V. Then $\boldsymbol{v}$ is either the minimiser of f on $\mathbb{R}^N$, or $\boldsymbol{v}$ is on the boundary of V.

Definition 3.16. Define $|\boldsymbol{c}' - V| = \sqrt{\sum_{i=1}^N d_i^2}$ as the distance from $\boldsymbol{c}'$ to V, with d_i the distance between $\boldsymbol{c}'$ and V in dimension i. Observe that in each dimension, $V_i \coloneqq \{v \in \mathbb{R}\colon v \geq -u_i\}$. For each d_i, one of the following holds

- $d_i = 0$ if $c_i' \in V_i$, and so $c_i' \geq -u_i$.
- $d_i \neq 0$ if $c_i' \notin V_i$. In this case, $-u_i - c_i' > 0$, and this is the distance between u_i and c_i'. Thus $d_i = -u_i - c_i'$.

Combining these two cases gives

$$d_i = \min\{-u_i - c'_i, 0\}\,,$$

and thus

$$\left|\boldsymbol{c}' - V\right| = \sqrt{\textstyle\sum_{i=1}^{N}(\min\{-u_i - c'_i, 0\})^2}\,.$$

Then, by construction, for the closest point, $\boldsymbol{v}$, we have that $v_i = \max\{c'_i, -u_i\}$ for $i \in \{1, \ldots, N\}$.

Summary and comparison to Dines' weights We have now found the weights that are a solution to our system, and are closest to the 'real' weights of all the solution. By looking at the null space of our matrix, we have found the subset of the null space where the solutions are still valid. That is, where the weights are greater than or equal to zero. From this set, we could then take the weights that were closest to the difference between the solution we already had, and the 'real' weights.

Let us take $N = 5$ and $k = 2$, and let $\boldsymbol{x}$ be a data set of size $n = 5000$. In Figure 3.4 we see the difference between using just Dines to determine the weights, and optimising these weights to the real weights. We observe that, indeed, the optimised weights look more like the real weights.

3.2.3 Lagrange Interpolation

In this section we will determine the weights based on both the function f and the data. We use *Lagrange interpolation* as described in [25] to approximate the function f with a degree N polynomial. We will use properties of this interpolation polynomial — it can be constructed using Lagrange basis polynomials — to construct weights.

To determine the *Lagrange interpolation polynomial* L_N we need to choose N points $\{z_1, \ldots, z_N\}$ for which we know the value of $f(z_i)$. It makes sense to choose these z_i as the representatives of our data, since we will have to choose these at some point anyway. L_N is a polynomial of degree $N - 1$, thus

$$L_N(x) = \sum_{k=0}^{N-1} c_{k+1} x^k\,,$$

where c_k are the coefficients of this polynomial. We approximate f, so it should hold that $f(z_i) = L_N(z_i)$ for $i = 1, \ldots, N$. This directly gives a linear system by writing out $c_1 + c_2 z_i + c_2 z_i^2 + \cdots + c_N z_i^{N-1} = f(z_i)$ for $i = 1, \ldots, N$. Using the *Lagrange basis polynomials* $L_{k,N}$ given by

$$L_{k,N}(x) = \frac{(x - z_1) \cdot \cdots \cdot (x - z_{k-1})(x - z_{k+1}) \cdot \cdots \cdot (x - z_N)}{(z_k - z_1) \cdot \cdots \cdot (z_k - z_{k-1})(z_k - z_{k+1}) \cdot \cdots \cdot (z_k - z_N)}$$

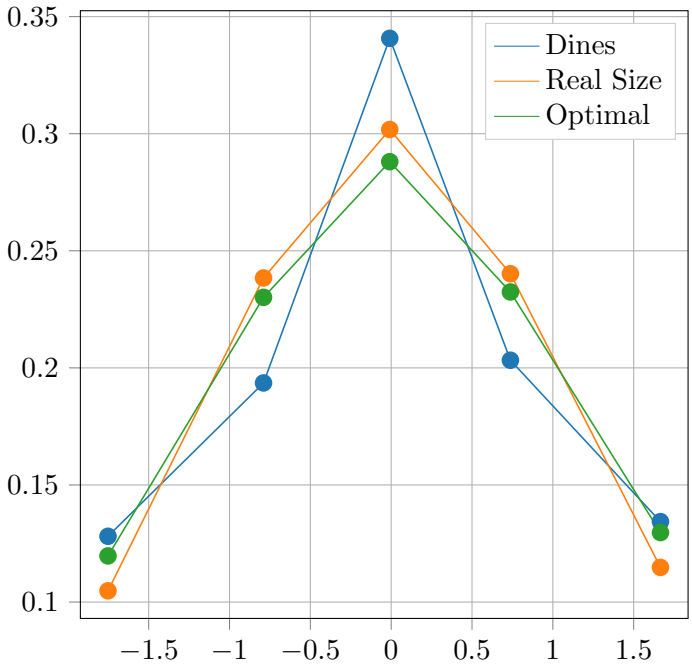


Figure 3.4: Connecting the five points (z,w) for each method to clarify the differences.

we can write the interpolation polynomial as

$$L_N(x) = \sum_{k=1}^{N} f(z_k) L_{k,N}(x)\,. \tag{3.12}$$

Substituting f by $L_N(x)$ in the integral (2.1) gives

$$\int_{-\infty}^{\infty} L_N(x)\mu(x)\,\mathrm{d}x = \int_{-\infty}^{\infty} \sum_{k=1}^{N} f(z_k) L_{k,N}(x)\mu(x)\,\mathrm{d}x = \sum_{k=1}^{N} w_k f(z_k)$$

with

$$w_k = \int_{-\infty}^{\infty} L_{k,N}(x)\mu(x)\,\mathrm{d}x\,.$$

Notice that this is an instance of our original problem, and that we are certain that $L_{k,N}(x)$ is a polynomial of degree $N-1$. We could solve this problem using any of the methods proposed in this report, but for simplicity we will use Monte Carlo integration as in Section 2.1.2. Note that $L_{k,N}$ is a polynomial of degree $N-1$, and thus easy to evaluate. Finally, this gives

$$\mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(x)\,\mathrm{d}M_X(x) \approx \sum_{k=1}^{N} \frac{1}{n} \sum_{i=1}^{n} L_{k,N}(x_i)\, f(z_k) \tag{3.13}$$

as the approximation generated by this method.

Furthermore, it is important to note that these weights can be negative, since $L_{k,N}(x)$ can be negative and $\mu(x)$ is never negative since it is a density function.

Lemma 3.17. Let f be a polynomial of degree k, $\boldsymbol{x}$ a data set of size n, and $z_1, \ldots, z_N$ be given, with $k \leq N-1$. Using the approximation above gives the same result as Monte Carlo integration, and thus, by Lemma 3.7, also the same result when determining the weights by solving the system in (3.3). That is

$$\sum_{k=1}^{N} \frac{1}{n} \sum_{i=1}^{n} L_{k,N}(x_i)\, f(z_k) = \frac{1}{n} \sum_{i=1}^{n} f(x_i)\,.$$

Furthermore, when $k = N-1$, the weights when solving a system or using Lagrange interpolation are the same.

Proof. Rewriting equation (3.13) gives

$$\sum_{k=1}^{N} \frac{1}{n} \sum_{i=1}^{n} L_{k,N}(x_i)\, f(z_k) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{N} f(z_k)\, L_{k,N}(x_i)\,.$$

Using (3.12) we obtain

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{N} f(z_k)\, L_{k,N}(x_i) = \frac{1}{n}\sum_{i=1}^{n} L_N(x_i)\,.$$

The Lagrange polynomial was constructed using N points and the value of the polynomial at those N points, by solving a system of N equations with N unknowns. The matrix corresponding to this system is a Vandermonde matrix, and thus the resulting polynomial is uniquely defined. Since f is a polynomial of degree $\leq N-1$, it holds that $L_N(x) = f(x)$.

When determining the weights by solving a system, we impose the following requirements.

$$\sum_{\ell=1}^{N} w_\ell z_\ell^m = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{for } m = 0, \ldots, k. \tag{3.14}$$

The Lagrange weights are given by

$$w_\ell = \sum_{i=1}^{n} L_{\ell,N}(x_i) \qquad \text{for } \ell = 1, \ldots, N\,,$$

substituting this in the left-hand side of (3.14) gives

$$\sum_{\ell=1}^{N} \frac{1}{n}\sum_{i=1}^{n} z_\ell^m L_{\ell,N}(x_i) = \frac{1}{n}\sum_{i=1}^{n}\sum_{\ell=1}^{N} z_\ell^m L_{\ell,N}(x_i)\,,$$

for $m = 0, \ldots, k$. Define $P_m(x) = \sum_{\ell=1}^{N} z_\ell^m L_{\ell,N}(x) - x^m$, $m = 0, \ldots, k$. Then $P(x)$ is a polynomial of degree $N-1$, due to the Lagrange basis polynomials. In particular, $P(x) = 0$ when $x = z_1, \ldots, z_N$, i.e., $P(x)$ has N roots. A polynomial of degree p has at most p roots, and since $P(x)$ has more than $N-1$ roots it must hold that $P(x) = 0$ for all x. And thus $\sum_{\ell=1}^{N} z_\ell^m L_{\ell,N}(x) = x^m$ for $m = 0, \ldots, k$. This gives that equation (3.14) holds for the Lagrange weights. When $k = N-1$ the solution to this system is unique, and thus the Lagrange weights are also the weights obtained by solving the system. □

3.3 Method Overview

We will give a short overview of the methods discussed in the previous sections. A quick overview is given in Table 3.1.

3.3.1 Methods to find Representatives

Theoretical Distribution

We used the assumption that the density of the data is known, and used this to determine the points that would best represent the data.

Advantages The integral that determines $g(\boldsymbol{z})$ is difficult to compute analytically. However, when $g(\boldsymbol{z})$ is computed for a given distribution and a given number of representatives, it can be used again and again. So once this integral has been computed once, this is a fast method.

Disadvantages The density of the data has to be known. Even when it is known, if might not be a nice function, and the integral that determines $g(\boldsymbol{z})$ can get really complicated.

Gradient Descent

Since the integral from the previous method is so complicated, we have been looking for a different way to minimise the function $g(\boldsymbol{z})$, not necessarily analytically. We defined a function $\tilde{g}(\boldsymbol{z})$ that approximated $g(\boldsymbol{z})$ using the data. Then we used Gradient Descent to find the minimum of this function.

Advantages There is no need for an assumption about the underlying density of the data.

Disadvantages When the initial guess for the representatives is bad, the solution found by this method might not be very good at all.

It is not guaranteed that we find N distinct representatives.

Clustering

We used the existing k-means (or Lloyd's) algorithm to split the data into clusters. We then took the points closest to the centres of these clusters as our representatives.

Advantages We do not need an assumption about the underlying data. There exists a number of (optimised) implementations of this algorithm, and we will use one of them [20].

3.3.2 Methods to find Weights

Cluster Size

To determine the weights given to a representative, we argued that it is natural to assign a weight that corresponds to the amount of data represented by this representative.

Advantages This method is fast.

Disadvantages It does not take into account that we wish to integrate over polynomials exact. Thus most likely, the approximation of the integral when using this method will not be very good.

Solving an Underdetermined System

The idea is that we wish to integrate polynomials of a certain degree k exactly. We enforced this requirement by formulating constraints that led to a system of equations. We then formulated a minimisation problem to have the weights be more intuitive, as they were for the Cluster Size method.

Advantages This guarantees an approximation that is 'good' for polynomials of degree k. Even though this approximation might not be exact, because the constraints are based on the sample moments of the data, instead of the analytical value.

Disadvantages There might not always exist a positive solution to the system of constraints. In this case we cannot approximate the integral using this method, and we are left with nothing.

Lagrange

We can use Lagrange interpolation to approximate our expensive function f with a polynomial. We use the Lagrange basis polynomials to find the weights. Polynomials are typically easy to evaluate, and we could use, e.g., Monte Carlo integration to integrate this polynomial. Note that the real f is still used to compute the approximation, as in (3.13).

Advantages The Lagrange basis polynomials are easy to compute.

Disadvantages The interpolation polynomial can suffer from *Runge's phenomenon* [25, Sec. 2.3]. The effect of Runge's phenomenon might not be very big here, since in most cases the tails of this polynomial will get small because of the density of the data. We will not go into this here, but it is important to keep in mind that this could be a problem.

With Table 3.1 in mind we expect the combination of Clustering with Solving a System to be the most practical and give the best results. This is because, first of all finding representatives using Theoretical Distribution is not very practical. Second of all Gradient Descent might not lead us to N distinct representatives, and so we expect Clustering to be the best method out of these three. For the latter, Cluster Size simply does not take the eventual approximation into account, and thus we expect Solving a System to give better results.

We will implement these methods in the next chapter, and discuss the results.

Method	**Goal**	**Assumptions**	**Advantages**	**Disadvantages**
Theoretical Distribution (3.1.1)	Representatives	Underlying data	Fast once integral has been computed	Complicated integral
Gradient Descent (3.1.2)	Representatives		No assumption about distribution of data	Might not find distinct representatives
Clustering (3.1.3)	Representatives		No assumption about distribution of data	
Cluster size (3.2.1)	Weights	Points are represented by closest representative	Fast	Does not impose requirements on the weights
Solving (underdetermined) system (3.2.2)	Weights	Points are represented by closest representative	Guarantees 'good' approximation	Positive solution does not always exist
Lagrange (3.2.3)	Weights		Basis polynomials are easy, an explicit formula is given	Runge's phenomenon might increase error

Table 3.1: An overview of the different proposed methods.

Chapter 4

Implementation and Performance

Now we have the methods, theoretically, but we have not gone into their performance yet. We will do so in this chapter, by means of simulations. This means that we will implement the methods from the previous chapters and compare their performances. We will shortly discuss the implementations used, in Section 4.1, and we will discuss the performance of the methods in Section 4.2.

4.1 Implementation

These methods were implemented in Python 3.6, using an Anaconda interpreter. The only method for which an existing implementation is used is Clustering (Section 3.1.3) where the K-Means algorithm from [20] is used. The other methods were implemented from scratch, using the NumPy package.

4.2 Performance

There are two performance measures we use to compare our methods. The first is how much the resulting representatives and weights look like the (empirical) density they represent, call this the *method shape*. The second is how well they approximate the integral of interest, recall that this is

$$\mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(x)\,\mathrm{d}M_X(x)\,.$$

The error then is the absolute difference between the analytical value of the integral, and the value of the approximation. We call this error the *method error*.

Recall that k is the order of accuracy we aim for, thus that the method error should be small as long as the degree of f is $\leq k$, when f is a polynomial. When f is not a polynomial,

we expect this error to be greater. Furthermore, N is the number of representatives we use, i.e., how many f evaluations each method will do.

4.2.1 Method Shape

We will first look into the method shapes. To do this, we will

1. Fix $k \in \mathbb{N}_+$, $N \in \{3, 6\}$, and $n \in \mathbb{N}_+$.
2. Generate a data set of size n.
3. Run all the methods to find representatives on this data set. That is, the representatives will be found by using the theoretical distribution (Section 3.1.1), by using Gradient Descent (Section 3.1.2), or by clustering the data (Section 3.1.3).
4. Run all the methods to find corresponding weights on each of the results for the representatives. Recall that finding weights can be done by using the sizes of the clusters (Section 3.2.1), by solving a system (Section 3.2.2) which can be optimised to the shape, or by using a Lagrange interpolation polynomial to approximate f (Section 3.2.3).

Consider Figure 4.1, where we see the shapes of the different methods dependent on the number of representatives. In Figure 4.1a we see a situation where $N = k + 1$. The first thing we note is that we see only two methods. This is due to the fact that $N = k+1$. This means that there is a unique solution to the system described in Section 3.2.2, and thus this unique solution is also the optimal. Furthermore, recall from Lemma 3.17 that the weights from the Lagrange method are equal to the weights when solving a system. Indeed, in the figure we see only the Lagrange weights and the Cluster Size weights.

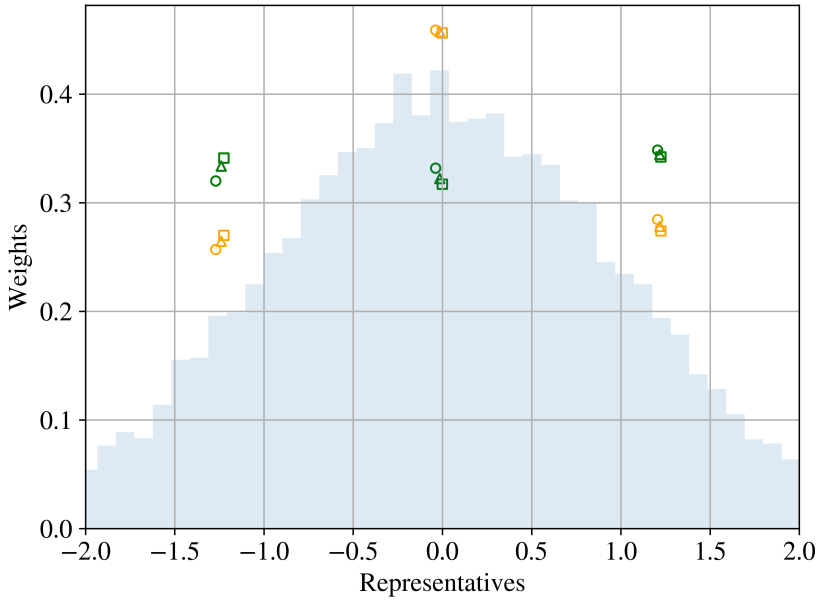
The second thing we note is that the weights found by the Cluster Size are high in the centre, and lower on the sides, nicely following the histogram of the data. However, the Lagrange weights do not follow this, and they are rather equal.

In Figure 4.1b we see a situation where $N > k + 1$. In this case the Dines weights can be optimised, and the Lagrange weights are also different from the Dines weights. We see that the Optimised weights are lower in the tails and the centre, and a bit higher for the representatives in between, compared to the Dines weights. These Optimised weights are close the Cluster Size weights, but they are not the same.

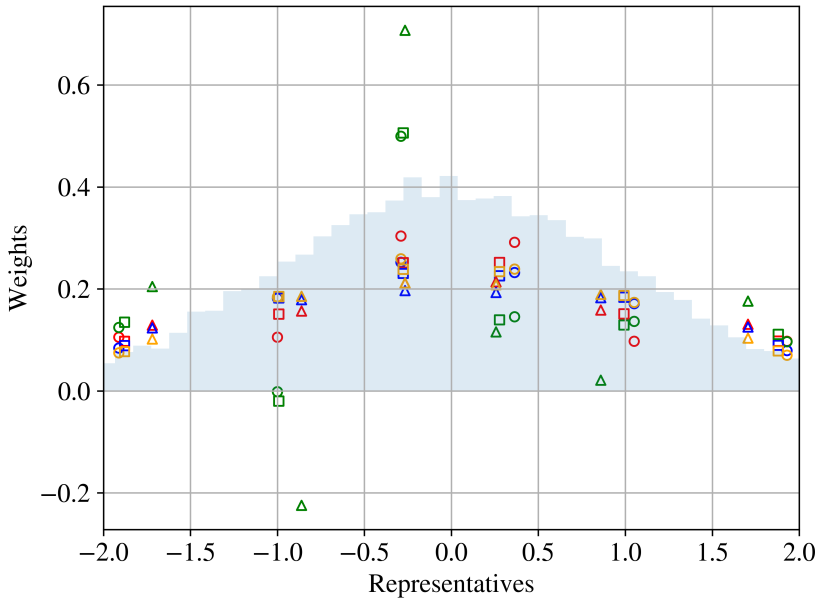
Our next observation is that the Lagrange weights are not all positive, and we have seen in Section 3.2.3 that they do not need to be.

The final thing we notice is that the Gradient Descent representatives are a little bit more in the centre than the Theoretical or Cluster representatives.

In both cases, the Cluster Size weights are not the same as the Dines weights, and thus they are not a solution to the system, and that the Method Error will be larger for this method than for the methods using Dines or the Optimised weights.



(a) $N = 3$



(b) $N = 6$

(c) Legend to these figures.

Figure 4.1: Representatives against their weights for each method, with the density of the data in the background. In both figures, $n = 10000$ *and* $k = 2$.

We will discuss the influence the Method Shape has on the Method Error in the next section, by discussing the different methods.

4.2.2 Method Error

We are interested in the Method Error for the different methods. We will discuss two instances of the function to integrate over. One is a polynomial of degree 2, namely $f(x) = x^2 + x + 1$. The other is not a polynomial, but it can be written as a power series, this function is $g(x) = e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{1}{2}x^2 + \mathcal{O}(x^3)$ [3, Sec. 4.10], where $\mathcal{O}$ is the *big-O symbol* [6].

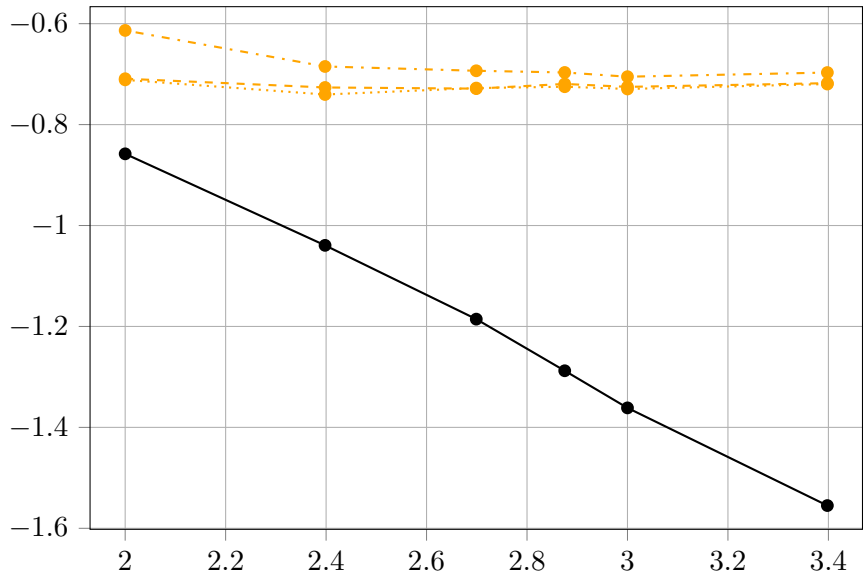
The results will be presented in the form of log-log plots. The Method Error will (hopefully) be between zero and one, and the natural logarithm amplifies this region. For example, $\log_{10}(1) = 0$ and $\lim_{x \to 0} \log_{10}(x) = -\infty$. Furthermore, the derivative of $\log_{10}(x)$ is $1/x > 0$, and thus the logarithm is increasing. This means that when the error is smaller, the logarithm is also smaller. Thus plotting the logarithm of the error instead of the error itself will make it easier to compare the errors of the different methods.

Integrating a polynomial of degree k

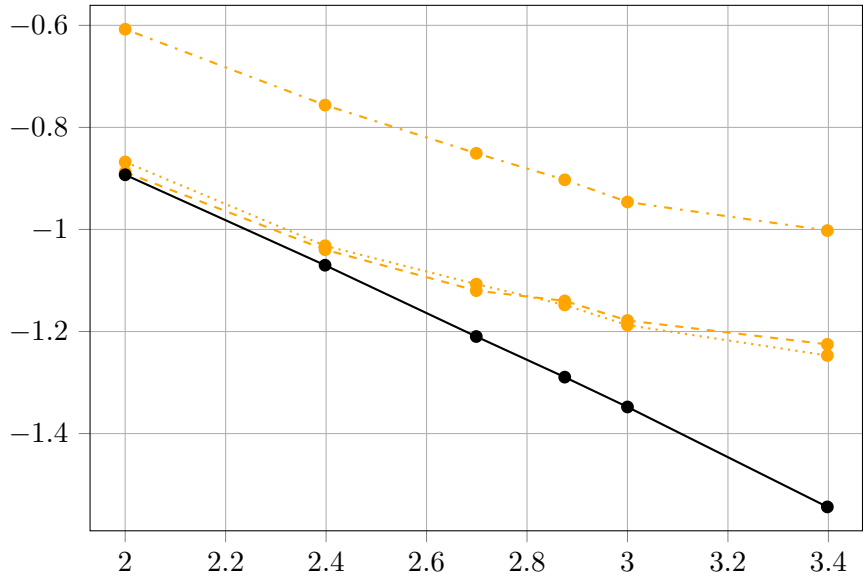
Recall from Lemma 3.7 and Lemma 3.17 that using Dines and Lagrange to construct the weights will gave the same final result as approximating the integral using Monte Carlo (MC) integration, when f is a polynomial of degree at most k. In this case $k = 2$. Therefore, to analyse the accuracy of these methods, we use only one of them. We will pick the one that is the least time consuming. Since we integrate over a polynomial of degree 2, which is not expensive to evaluate, and the maximum size of the data set we use is $n = 2500$, Monte Carlo will be the least time consuming. As in Figure 4.2, we only have to compare Monte Carlo to the methods using the Cluster Size (CS) to determine the weights.

In Figure 4.2 we see the results when integrating over $f(x) = x^2 + x + 1$ when using three representatives (Figure 4.2a) and when using six representatives (Figure 4.2b). Of course, Monte Carlo integration is independent of the number of representatives, and we see that the results for this method are the same in both figures. In both figures, we see that — when using Cluster Size to determine the weights — using Gradient Descent to determine the representatives gives less good results than using Clustering or the Theoretical Distribution.

As expected, the methods that behave like Monte Carlo give better results than the methods using Cluster Size. Furthermore, the slope of this line is $-\frac{1}{2}$ in the log-log scales. This corresponds to a convergence rate $\frac{1}{\sqrt{n}}$ of the error to zero, with n the size of the data set. This is what one would expect when using a Monte Carlo method, due to the *Law of Large Numbers*, see e.g., [15, Chap. 2]. The convergence rate of the Cluster Size methods is always less than for the Monte Carlo methods, as we see in both figures.



(a) $N = 3$



(b) $N = 6$

(c) Legend to these figures.

Figure 4.2: Log-log plot using the base 10 real logarithm of the Method Error for the different methods. In both figures, $n \in \{100, 250, 500, 750, 1000, 2500\}$, $k = 2$ and integrating over $f(x) = x^2 + x + 1$.

Not integrating over a polynomial of degree k

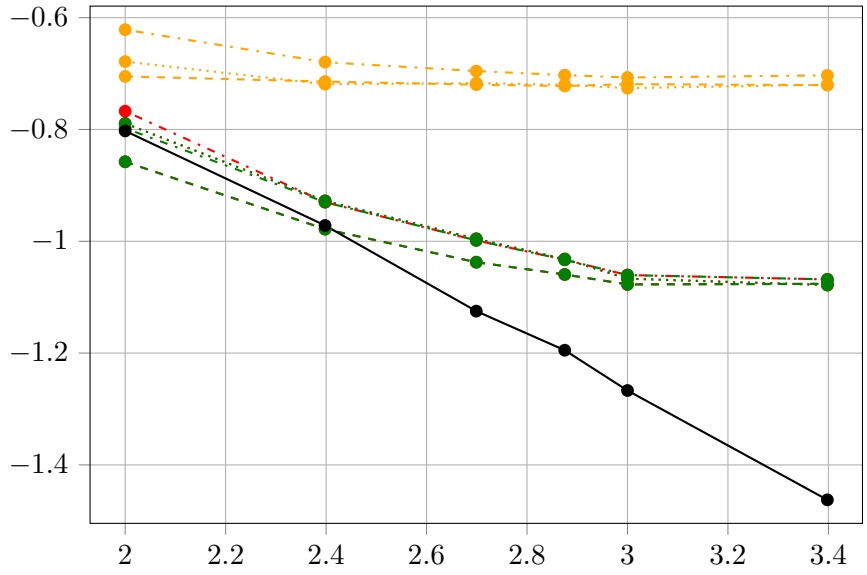
When not integrating over a polynomial of a certain degree, Lemmas 3.7 and 3.17 do not hold. And so, in Figure 4.3, we compare all methods individually. It should be noted that in both figures 4.3a and 4.3b the results for Monte Carlo are the same (with convergence rate $\frac{1}{\sqrt{n}}$).

In Figure 4.3a, where the number of representatives is three, we see an obvious difference between the methods using Cluster Size or Dines or Lagrange. Based on this figure, we might pick Cluster Size as the worst method to determine the weights, as expected. The methods using Dines and Lagrange behave very similar, and even give the same results for Gradient Descent and Clustering.

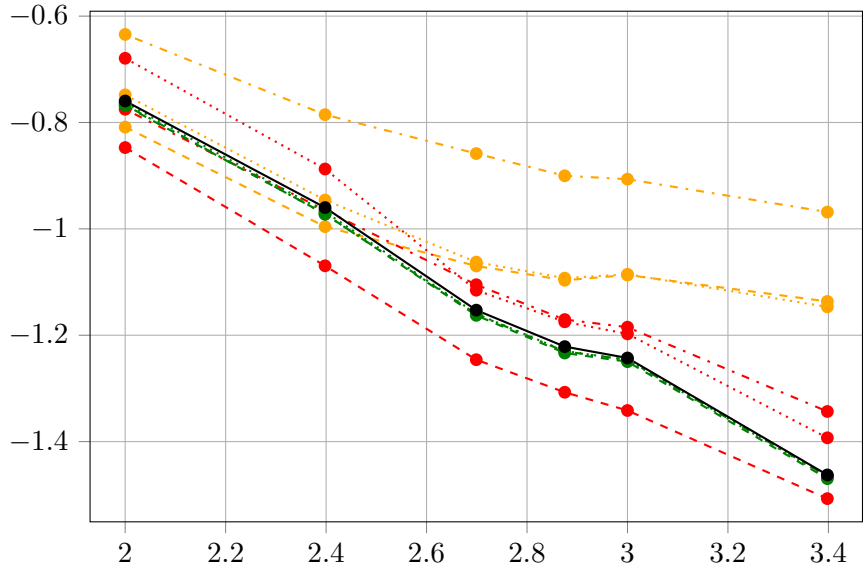
When taking the number of representatives to be six, as in Figure 4.3b, the results look different. The first thing we observe is that both Lagrange and Dines behave more like Monte Carlo than in the previous figure. This might imply that increasing the number of representatives improves the results, which is natural. Furthermore, for bigger n, the Lagrange methods behave almost exactly as Monte Carlo.

From this figure we can say that using the Theoretical Distribution to find the representatives and using Dines to find the corresponding weights is the best method. However, we have seen before that Theoretical Distribution is not very useful in practice. Therefore we might suggest the next method with good results, using Lagrange to find the weights.

Finally, from Figure 4.3 we can conclude that increasing the number of representatives will probably give better results. Furthermore, the best results are obtained using Lagrange or Dines to find the weights. For Dines, the best method to find the representatives is Clustering.



(a) $N = 3$



(b) $N = 6$

(c) Legend to these figures.

Figure 4.3: Log-log plot using the base 10 real logarithm of the Method Error for the different methods. In both figures, $n \in \{100, 250, 500, 750, 1000, 2500\}$, $k = 2$ *and integrating over* $f(x) = e^x$.

Chapter 5

Conclusion

We have seen various ideas to find weights and representatives for the approximation of $\mathbb{E}[f(x)]$. In the previous chapter we have seen how the different methods performed, and we concluded that the best results are obtained using either Lagrange (3.2.3) or Dines (3.2.2) to find the weights. For Lagrange the method of finding representatives is irrelevant, and for Dines the best representative finding method to be used is Clustering (3.1.3).

Discussion and further research The method(s) we have seen in this report are based on the Gaussian quadrature rules (Section 2.1.1). This is one possible approach, and most likely not the only one. We did not look into any other method, and thus cannot be sure that there exist no other methods that might be faster, easier, or more accurate.

In Chapter 4 we have only verified the performance of the methods for the normal distribution, and for relatively easy functions. From this we cannot draw the conclusion that the performance will be similar for any wild distribution or function. More research will have to be done to verify if this is the case, and if there exist situations where our method does not work.

Also, we have based the performance of the methods mostly on the results, and not on the time these methods needed. The reason for this is that we have only tested our methods on toy functions and data sets, and we did not expect our methods to be faster than Monte Carlo integration for these relatively easy problems.

Finally, our method is only constructed for one dimension. In practice and real life, however, a lot of data will depend on multiple parameters, and thus be multi-dimensional. It would be interesting to see if our method can be extended to more dimensions.

Bibliography

[1] F. Abramovich and Y. Ritov. *Statistical Theory: A Concise Introduction*. Chapman and Hall/CRC, 2013.

[2] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*. Dover Publications Inc., 1st edition, 1965.

[3] R. Adams and C. Essex. *Calculus: A Complete Course*. Pearson Education Canada, seventh edition, 2009.

[4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[5] J. Barceló et al. *Fundamentals of Traffic Simulation*, volume 145. Springer, 2010.

[6] N. G. D. Bruijn. *Asymptotic Methods in Analysis*. DOVER PUBN INC, 2010.

[7] R. E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 7:1, 1998.

[8] P. Cameron. Note on linear algebra, 2005. `http://www.maths.qmul.ac.uk/~pjc/notes/linalg.pdf`.

[9] T. S. Chihara. *An Introduction to Orthogonal Polynomials*. Courier Corporation, 2011.

[10] H. B. Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[11] A. A. M. Cuyt, V. Petersen, B. Verdonk, H. Waadeland, and W. B. Jones. *Handbook of Continued Fractions for Special Functions*. Springer-Verlag GmbH, 2008.

[12] L. L. Dines. On positive solutions of a system of linear equations. *The Annals of Mathematics*, 28(1/4):386, 1926.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. JHU Press, 4 edition, 2012.

[14] G. H. Golub and J. H. Welsch. Calculation of Gauss Quadrature Rules. *Mathematics of Computation*, 23(106):221–221, May 1969.

[15] C. Graham and D. Talay. *Stochastic Simulation and Monte Carlo Methods*. Springer Berlin Heidelberg, 2013.

[16] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

[17] P. K. Kythe and P. Puri. *Computational Methods for Linear Integral Equations*. Birkhuser Boston, 2002.

[18] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[19] O. D. L. Mejia and J. A. E. Gomez, editors. *Numerical Simulation of the Aerodynamics of High-Lift Configurations*. Springer International Publishing, 2018.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] K. A. Ross. *Elementary Analysis*. Springer New York, 2013.

[22] N. M. Steen, G. D. Byrne, and E. M. Gelbard. Gaussian quadratures for the integrals $\int_0^\infty e^{-x^2} f(x)\,\mathrm{d}x$ and $\int_0^b e^{-x^2} f(x)\,\mathrm{d}x$. *Mathematics of Computation*, 23(107):661–661, 1969.

[23] D. W. Stroock. *A Concise Introduction to the Theory of Integration*. Springer Science & Business Media, 1998.

[24] H. Tuy. *Convex Analysis and Global Optimization*. Springer International Publishing, 2016.

[25] C. Vuik, F. Vermolen, M. van Rijzen, and M. Vuik. *Numerical Methods for Ordinary Differential Equations*. Delft Academic Press, 2nd edition, 2015.

Index