Uncertainty quantification using high-dimensional numerical integration

Rickard Strandberg
rstran@kth.se

Johan Låås
johanlaa@kth.se

2016-05-20

Abstract

We consider quantities that are uncertain because they depend on one or many uncertain parameters. If the uncertain parameters are stochastic the expected value of the quantity can be obtained by integrating the quantity over all the possible values these parameters can take and dividing the result by the volume of the parameter-space. Each additional uncertain parameter has to be integrated over; if the parameters are many, this give rise to high-dimensional integrals.
This report offers an overview of the theory underpinning four numerical methods used to compute high-dimensional integrals: Newton-Cotes, Monte Carlo, Quasi-Monte Carlo, and sparse grid. The theory is then applied to the problem of computing the impact coordinates of a thrown ball by introducing uncertain parameters such as wind velocities into Newton's equations of motion.

Contents

1 Introduction

A function $f(x)$ meant to realistically model something will more often than not contain some parameters for which the exact values are unknown. This uncertainty in parameters may be caused by error margins in measurements, small differences between samples of materials and a host of other reasons. For each different value a parameter can take, a different answer will be obtained from evaluating $f(x)$.
In lieu of a single answer from the evaluation of $f(x)$, a good approximation would be the most likely outcome of an evaluation. By uncertainty quantification the assumption is made that the parameters are stochastic variables. In this paper we further assume that they are uniformly distributed, that is; the parameters may take on any value in a certain interval with equal probability.

For some randomly distributed parameter α in the interval I_α, with $|I_\alpha|$ denoting the length of I_α. The expectation value of a function $f(x)$ containing the parameter α would be given by

$$E[f] = \frac{1}{|I_\alpha|} \int_{I_\alpha} f(x, \alpha) d\alpha.$$

If $f(x)$ contain an additional parameter $\beta \in I_\beta$, the expectation value would be found by

$$E[f] = \frac{1}{|I_\beta|\,|I_\alpha|} \int_{I_\alpha} \int_{I_\beta} f(x, \alpha, \beta) d\alpha d\beta.$$

Each additional uncertain parameter has to be integrated over. This give rise to high-dimensional integrals.

All methods for computing an integral numerically will involve evaluating the integrand in a set of points $\{x_j\}$. How these nodes[1] are chosen will depend on which method is being used. If the approximation employs n nodes in each direction, the total number of points N will grow exponentially with dimension d; $N = n^d$. Since the bulk of the computational cost will come from evaluating the function, the cost of computation will be directly related to the total amount of nodes.
This exponential dependence on dimension is referred to as *the curse of dimensionality*, and circumventing it will be the main focus of this paper.

The first approach to high-dimensional integration examined is the Newton-Cote's formulae which divides the interval integrated over into n equidistant nodes $\{x_j\}_1^n$, separated by a step size h. The integrand is then interpolated in these nodes using polynomials which are then integrated exactly.

Next a probabilistic approach to the problem of dimensionality is investigated through the Monte Carlo method. This method interprets the integral of $f(x)$ over the domain Ω as the scaled expectation value of $f(x)$. This expectation value can be computed by evaluating the integrand in randomly selected points from within Ω and then taking the mean value of these evaluations.

[1] Throughout this paper we will use points and nodes interchangeably.

A refinement of the Monte Carlo method is developed by the introduction of quasi-randomly selected points. The quasi-random sequences are more uniformly distributed throughout Ω, thereby providing an approximation less sensitive to local variations of the integrand.

The last method covered in the theory is the sparse grid method. This method approximates the integrand by a linear combination of piecewise linear basis functions belonging to different levels of mesh refinement. Some of the terms in the resulting sum contribute only marginally to the approximation and can thus be discarded. This leads to a significant reduction in the total amount of points N in which the function has to be evaluated.

The paper is concluded by solving for the impact coordinates of a thrown ball using the methods developed throughout the theory sections.

2 Theory

In this section we discuss four different ways to approximate high-dimensional integrals. The integral

$$\int_\Omega e^{x_1 \cdot \ldots \cdot x_d} d\mathbf{x} \qquad x_i \in \Omega \qquad d\mathbf{x} := dx_1 ... dx_d \tag{1}$$

will be used in order to plot the exact error[2].

This integral is chosen because the correct value can be approximated to high accuracy by analytic means; something which is rarely possible as the dimension of the integral increase. To obtain the correct value recall the Taylor expansion of e^x:

$$e^{x_1 \cdot \ldots \cdot x_d} = \sum_{i=0}^{\infty} \frac{(x_1 \cdot ... \cdot x_d)^i}{i!}. \tag{2}$$

For the d-dimensional integral over the domain $\Omega = [0, L]^d$ the integral can be rewritten by plugging (2) into (1):

$$\begin{aligned}\int_\Omega e^{x_1 \cdot \ldots \cdot x_d} d\mathbf{x} &= \int_\Omega \sum_{i=0}^{\infty} \frac{(x_1 \cdot ... \cdot x_d)^i}{i!} d\mathbf{x} = \sum_{i=0}^{\infty} \frac{1}{i!} \int_\Omega (x_1)^i \cdot ... \cdot (x_d)^i d\mathbf{x} \\ &= \sum_{i=0}^{\infty} \frac{1}{i!} \int_0^L (x_1)^i dx_1 ... \int_0^L (x_d)^i dx_d = \sum_{i=0}^{\infty} \frac{1}{i!} \frac{L^{(i+1)\cdot d}}{(i+1)^d}.\end{aligned}$$

This sum converges rapidly for moderate values of L.

[2]The test-integrand belongs to the class of infinitely differentiable functions and is therefore a special case. The plots does therefore not constitute proofs but serves to give a graphic representation of the convergence

2.1 Newton-Cotes formulas

The Newton-Cotes formulas are the most basic methods for numerical integration and is characterized by their use of equidistant nodes (Figure 1).
To quantify the error define the order of accuracy as the largest k such that the error ϵ_n satisfies

$$|\epsilon_n| \leq Cn^{-k} = Ch^k$$

for some constant C independent of n.

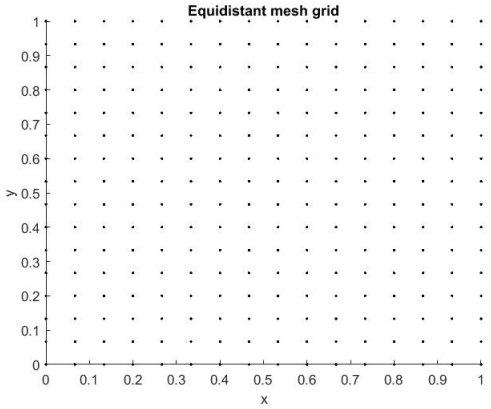

Figure 1: The two-dimensional equidistant grid.

2.1.1 Newton-Cotes formulas in one dimension

The Newton-Cotes formulas in one dimension approximates $\int_a^b f(x)dx$ by splitting the interval $[a, b]$ into $n + 1$ equidistant nodes $\{x_j\}_0^n$ separated by a step length $h = \frac{b-a}{n}$. These nodes are then interpolated with a polynomial of degree n, P_n, which can then be integrated exactly.
The interpolating polynomial should satisfy the condition that $P(x_j) = f(x_j)$.

Consider the Lagrange basis polynomials as defined by

$$L_j(x) := \prod_{j \neq m=0}^{k} \frac{x - x_m}{x_j - x_m} \qquad L_j(x_i) = \begin{cases} 1 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}$$

The condition that $P_n(x_i) = f(x_i)$ can be satisfied with these Lagrange polynomials allowing $P_n(x)$ to be written :

$$P_n(x) = \sum_{j=0}^{n} f(x_j)L_j(x).$$

The integral can then be approximated by:

$$I[f] \approx \int_a^b P_n(x)dx = \int_a^b \sum_{j=0}^{n} f(x_j)L_j(x)dx = \sum_{j=0}^{n} f(x_j)w_j, \tag{3}$$

where

$$w_j = \int_a^b L_j(x)dx.$$

Equation (3) is called a $n+1$ point simple quadrature rule $Q_{n+1}[f]$.

Depending on the degree of P_n, the names of these rules are: the midpoint rule, trapezoidal rule and Simpson's rule for polynomial degree 0,1 and 2 respectively.

The approximation improves as more nodes are considered. This then means that the degree of the interpolating polynomial becomes large as the demands on accuracy increase. High degree interpolating polynomials introduce issues such as Runge's phenomena where the end-points of the interpolating function will oscillate, leading to a bad approximation.

Instead of one quadrature formula for the entire interval it is therefore more practical to split the interval into smaller intervals with equidistant nodes and apply a simple quadrature rule to each interval and sum these up. These are called composite quadrature rules. An illustration of the one-dimensional trapezoidal composite quadrature rule is plotted in Figure 2.

If a simple quadrature rule is of order k, the composite rule will be of order $k+1$. This means that the composite quadrature rule of order $k+1$ will integrate polynomials of degree $k+1$ or lower exactly[7].

On the interval $[a, b]$ split into n subintervals, with a distance between nodes h (step size), given by $h = \frac{(b-a)}{n}$. The composite trapezoidal rule in the x direction is defined by:

$$T_x[f] := \sum_{j=1}^{n} T_j[f] := \sum_{j=1}^{n} \frac{h}{2}(f(x_{j-1}) + f(x_j)). \tag{4}$$

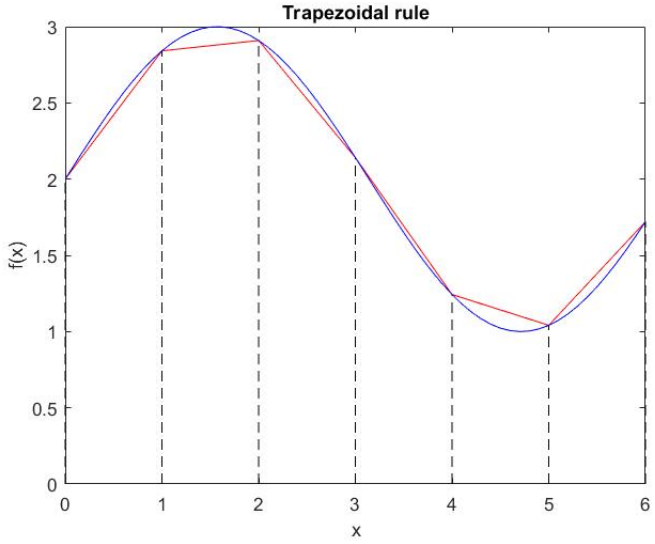

Figure 2: The composite trapezoidal rule - piecewise linear interpolation.

2.1.2 Generalization to higher dimensions

To generalize the trapezoidal rule to higher dimensions consider the integral of $f(x, y)$ over a rectangular domain $\Omega = [a, b] \times [c, d]$

$$I[f] = \int\int_{\Omega} f(x, y)dxdy \Longleftrightarrow \int_a^b \Big(\int_c^d f(x, y)dy \Big) dx. \tag{5}$$

Define the inner integral $F(x) := \int_c^d f(x, y)dy$. The integral can then be written

$$I[f] = \int_a^b F(x)dx. \tag{6}$$

Dividing the interval into n subintervals admits the definition of the step size in the x direction $h_x = \frac{b-a}{n}$ and the nodes $x_j = a + jh_x$. The trapezoidal rule can then be written:

$$I[f] \approx h_x \big(\frac{1}{2} F(x_0) + F(x_1)....F(x_{n-1}) + \frac{1}{2} F(x_n) \big).$$

To calculate the inner function $F(x_j)$ use the trapezoidal rule once more. Let $y_j = c + jh_y$, $h_y = \frac{d-c}{m}$;

$$F(x_j) = \int_c^d f(x_j, y)dy \approx h_y \big(\frac{1}{2} f(x_j, y_0) + f(x_j, y_1) + ... + f(x_j, y_{m-1}) + \frac{1}{2} f(x_j, y_m) \big),$$

$$I[f] \approx \sum_{j=0}^{n} w_j^x F(x_j) h_x = \sum_{j=0}^{n} \sum_{k=0}^{m} w_j^x w_k^y f(x_j, y_k) h_x h_y,$$

where the weights are given by:

$$w_j^x = \begin{cases} \frac{1}{2} & j = 0, n \\ 1 & j \neq 0, n \end{cases} \qquad w_k^y = \begin{cases} \frac{1}{2} & k = 0, m \\ 1 & k \neq 0, m \end{cases}$$

The two-dimensional trapezoidal rule is thus defined by:

$$T_x T_y[f] := \sum_{j=0}^{n} \sum_{k=0}^{m} w_j^x w_k^y f(x_j, y_k) h_x h_y. \tag{7}$$

The pattern is the same for arbitrary dimensions. For d dimensions, with each dimension having n_i points. The d-dimensional trapezoidal rule will consist of d sums,

$$I[f] \approx \sum_{i_1=0}^{n_1} ... \sum_{i_d=0}^{n_d} w_{i_1} ... w_{i_d} f(x_{i_1}, ..., x_{i_d}) h_{i_1} ... h_{i_d}.$$

2.1.3 Error of the one-dimensional Trapezoidal-rule

Theorem 1. *Suppose that $f \in C^2$ and $|f''(x)| \leq M \ \forall x \in [a, b]$.*
The error of the trapezoidal quadrature rule applied to $f(x)$ is then:

$$\epsilon[f] = T_x(f) - \int_a^b f(x)dx = R_f(h) \cdot h^2,$$

where

$$|R_f(h)| \leq \frac{(b-a)}{12} M.$$

Proof. Define $x_i = a + \frac{i(b-a)}{n}$ and let c_i be the midpoint of each subinterval so that

$$c_i = \frac{(x_{i-1} + x_i)}{2} \qquad \Rightarrow \qquad x_i - c_i = c_i - x_{i-1} = \frac{h}{2}.$$

The definition of the composite trapezoidal quadrature rule given by equation (4) in section 2.1.1. allows for the error to be written:

$$\epsilon[f] = \left| \sum_{i=1}^{n} \left(\frac{h}{2} \left(f(x_{i-1}) + f(x_i) \right) - \int_{x_{i-1}}^{x_i} f(x)dx \right) \right| := \left| \sum_{i=1}^{n} L_i \right|.$$

Applying integration by parts backwards on L_i gives:

$$L_i = \int_{x_{i-1}}^{x_i} (x - c_i) f'(x)dx.$$

Use integration by parts on L_i to obtain:

$$L_i = \int_{x_{i-1}}^{x_i} (x - c_i) f'(x)dx = \left(\frac{x^2}{2} - c_i x \right) f'(x) \Big|_{x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} \left(\frac{x^2}{2} - c_i x \right) f''(x)dx.$$

The definition $c_i = \frac{(x_{i-1}+x_i)}{2}$ yields:

$$\begin{aligned} L_i &= \frac{x_{i-1} \cdot x_i}{2} \left[f'(x_{i-1}) - f'(x_i) \right] - \int_{x_{i-1}}^{x_i} \left(\frac{x^2}{2} - c_i x \right) f''(x) dx \\ &= \frac{x_{i-1} \cdot x_i}{2} \int_{x_i}^{x_{i-1}} f''(x) dx - \int_{x_{i-1}}^{x_i} \left(\frac{x^2}{2} - c_i x \right) f''(x) dx. \end{aligned}$$

Rewriting $x_{i-1} \cdot x_i = (c_i - \frac{h}{2})(c_i + \frac{h}{2}) = c_i^2 - \frac{h^2}{4}$ produces

$$\frac{1}{2} \left(\left(\frac{h}{2} \right)^2 - c_i^2 \right) \int_{x_{i-1}}^{x_i} f''(x) dx - \int_{x_{i-1}}^{x_i} \left(\frac{x^2}{2} - c_i x \right) f''(x) dx.$$

Which can be reshaped into

$$L_i = \frac{1}{2} \int_{x_{i-1}}^{x_i} \left(\left(\frac{h}{2} \right)^2 - (x - c_i)^2 \right) f''(x) dx.$$

By using the restriction $|f''(x)| \leq M \ \forall x \in [a, b]$ the original expression can be rewritten

$$\begin{aligned} |\epsilon[f]| &\leq \sum_{i=1}^{n} |L_i| \leq \frac{1}{2} \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} \left| \left(\frac{h}{2} \right)^2 - (x - c_i)^2 \right| |f''(x)| \, dx \\ &\leq \frac{M}{2} \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} \left(\left(\frac{h}{2} \right)^2 - (x - c_i)^2 \right) dx \\ &= \frac{M}{2} \sum_{i=1}^{n} \frac{1}{6} h^3 = \frac{(b-a)}{12} M \cdot h^2. \end{aligned}$$

□

This result shows that the error decreases as $O(h^2)$ and since $h^2 \propto n^{-2}$ this also reveals that increasing the number of nodes by a factor 2 reduces the error by $\frac{1}{4}$.

2.1.4 Error of the two-dimensional Trapezoidal-rule

Theorem 2. *Suppose* $|f_{xx}(x, y)| \leq M_x$ *and* $|f_{yy}(x, y)| \leq M_y \ \forall (x, y) \in [a, b] \times [c, d]$ *where* M_x *and* M_y *are constants.*

Then the error of the two dimensional trapezoidal rule as defined by equation (7) *in section* 2.1.2 *is:*

$$|\epsilon[f]| = \left| T_x(T_y f(x, y)) - \int_a^b \int_c^d f(x, y) dx dy \right| \leq C(h_x^2 + h_y^2),$$

where C *is some constant.*

Proof. Recall the notation used in equation (5) and (6) of section 2.1.2;

$$I = \int\int_{\Omega} f(x,y)dxdy = \int_a^b \left(\int_c^d f(x,y)dy \right) dx := \int_a^b F(x)dx.$$

The trapezoidal rule applied to the double integral can then be written

$$T_x F(x) = \int_a^b F(x)dx + h_x^2 R_F(h_x),$$

$$T_y f(x,y) = \int_c^d f(x,y)dy + h_y^2 S(x,h_y) = F(x) + h_y^2 S(x,h_y).$$

Observe that the term containing $S(x,h_y)$ show up as the error term resulting from a one-dimensional trapezoidal rule and is therefore, by Theorem 1, bounded by $|S(x,h_y)| \leq \frac{(d-c)}{12} M_y$. Combining these two expressions:

$$\begin{aligned} T_x T_y f(x,y) &= T_x \left(F(x) + h_y^2 S(x,h_y) \right) = T_x F(x) + h_y^2 T_x S(x,h_y) \\ &= \int_a^b F(x)dx + h_x^2 R_F(h_x) + h_y^2 T_x S(x,h_y) \\ &= \int_a^b \left(\int_c^d f(x,y)dy \right) dx + h_x^2 R_F(h_x) + h_y^2 T_x S(x,h_y). \end{aligned}$$

For the coefficient of h_y^2;

$$|T_x S(x,h_y)| \leq \sum |h_x S(x_j,h_y) w_j| \leq \sum \left| h_x \big(\frac{(d-c)}{12} M_y \big) w_j \right| = D$$

where D is a constant.
Rearranging this yields the error:

$$\epsilon[f] = T_x T_y f(x,y) - \int_a^b \int_c^d f(x,y)dydx \leq h_x^2 R_f(h_x) + h_y^2 D \Rightarrow O(h_x^2 + h_y^2).$$

□

The errors in one and two dimensions suggests that the error of the trapezoidal rule in d dimensions should be of the form $c(h_1^2 + h_2^2 + ... + h_d^2)$ and this turns out to indeed be the case.

The total number of points for d dimensions of n nodes $N = n^d$, combined with $h^2 \propto n^{-2}$, allows for the error to be recast in terms of N: $O(N^{-2/d})$.

In general, the error for a k order method will behave as $O(N^{-k/d})$.
This result proves the veracity of the claims made in the introduction; for a specified error, the cost of computation grows exponentially with dimension.

2.2 Monte Carlo method

2.2.1 Integral interpreted as an expectation value

The expectation value interpretation of an integral opens up for the possibility of a probabilistic approach to the problem of high-dimensional integration. The integral of $f(x)$ over the interval $x \in \Omega$ is then taken to represent the scaled mean value of $f(x)$ on that interval:

$$I[f] = \int_\Omega f(x)dx = |\Omega| \cdot \bar{f}$$

where $|\Omega|$ denotes the volume measure of Ω. For convenience define $\Omega = [0, 1]$. Let x be a stochastic variable uniformly distributed on the unit interval. The integral can then be interpreted as the expectation value of $f(x)$

$$I[f] = E[f(x)].$$

Generalizing $x \to \mathbf{x} \in \mathbb{R}^d$ to be the set of stochastic variables distributed uniformly on the unit hypercube $\Omega = [0, 1]^d \in \mathbb{R}^d$ (A visual representation for $\Omega \in \mathbb{R}^2$ is displayed in Figure 3) the integral can be written:

$$I[f] = E[f(\mathbf{x})] = \int_\Omega f(\mathbf{x})d\mathbf{x}.$$

A sample mean of function evaluations performed in the randomly[3] selected nodes $\{x_i\}_1^N$ allows for the approximation of the expectation value of f on Ω:

$$E[f(x)] \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) = I_N[f]. \tag{8}$$

Equation (8) is what is known as the Monte Carlo method and will converge with probability one by the strong law of large numbers:

$$\lim_{N\to\infty} I_N[f] \to I[f].$$

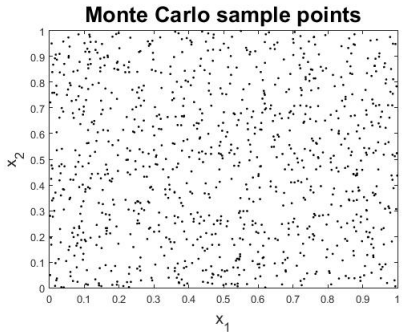

Figure 3: Two-dimensional grid of pseudo-random points.

[3]In practical uses these nodes will be generated by a computer and are therefore not truly random. They will however posses the properties of random numbers for a reasonable sample size. Computer-generated random numbers are referred to as pseudo-random in order to make this distinction clear.

2.2.2 Theoretical error of the Monte Carlo method

For a large number of points N the error of the Monte Carlo method applied to the function f is:

$$\epsilon_N[f] = I_N[f] - I[f] \approx \sigma N^{-1/2}\nu \tag{9}$$

where

$$\sigma = \big(\int_\Omega (f(x) - \bar{f})^2 dx\big)^{1/2},$$
$$\nu \in N(0,1).$$

A proof of the normal distribution of ν is given in [4].
As for the proof of the dependence on N:

Theorem 3. *The standard deviation of the Monte-Carlo method is* $O\left(\sigma N^{-1/2}\right)$.

Proof. Define $\xi_i = \frac{f(x_i)-\bar{f}}{\sigma}$ for which the following expectation values can be calculated:

$$E[\xi_i] = 0, \quad E[\xi_i^2] = \int \frac{(f(x_i) - \bar{f})^2}{\sigma^2} dx = 1, \quad E[\xi_i \xi_j] = 0 \quad if \quad i \neq j.$$

The last result follow because the x_i are independent (which consequently mean that the ξ_i are too).
Form the sum

$$S_N = \frac{1}{N}\sum_1^N \xi_i = \frac{1}{\sigma}\Big(\frac{1}{N}\sum_1^N f(x_i) - \bar{f}\Big) = \frac{\epsilon_N}{\sigma} \tag{10}$$

which has the variance

$$\begin{aligned} E[S_N^2] &= E\Big[(\frac{1}{N})^2(\sum_1^N \xi_i)^2\Big] = \frac{1}{N^2}\big(E\big[\sum_1^N \xi_i^2\big] + E\big[\sum_1^N \sum_{j\neq i} \xi_i \xi_j\big]\big) \\ &= \frac{1}{N^2}(\sum_1^N 1 + 0) = \frac{1}{N}. \end{aligned} \tag{11}$$

Combining (10) and (11) produces:

$$E\left[\epsilon_N^2\right] = \frac{\sigma^2}{N}$$

the square root of which is the standard deviation:

$$\sqrt{E[\epsilon_N^2]} = \sigma N^{-1/2}.$$

□

2.2.3 Empirical error of the Monte Carlo method

The result in the previous section is useful insofar as it reveals that the error depend on $N^{-1/2}$. The actual error does however not only contain the standard deviation which by itself is hard to calculate, but it is also normally distributed.

An empirical error is computed by performing M calculations with different sets of N values of x: $x_i \in \Omega$, $i = 1, ..., N$, which gives M different approximations of the integral, $I_N^{(j)}$, $j = 0, 1, ..., M$. The sample variance of ϵ_N is then formed by:

$$\tilde{\sigma}_N^2 = \frac{1}{M-1}\sum_{j=1}^{M}(I_N^{(j)} - \bar{I}_N)^2, \qquad \bar{I}_N = \frac{1}{M}\sum_{j=1}^{M} I_N^{(j)}.$$

The empirical error can thus be stated as $\tilde{\epsilon}_N = \nu\tilde{\sigma}_N$.
The error in approximating the test integral is shown in Figure 4, while the empirical standard deviation is plotted in Figure 5.

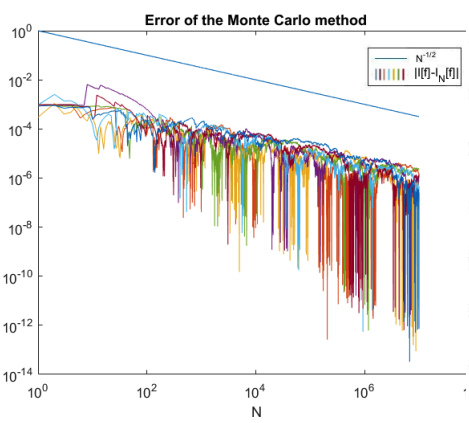

Figure 4: The difference between the N point Monte Carlo approximation of the test-integral and the true value. Computed seven times.

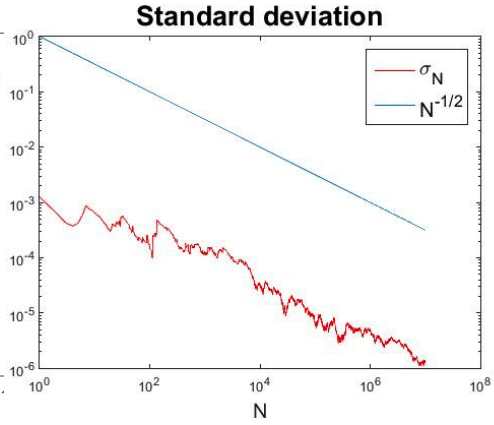

Figure 5: Standard deviation of the seven Monte Carlo approximations.

2.3 Quasi-Monte Carlo method

2.3.1 From pseudo to quasi - uniformity of points

The Monte Carlo method relied on the randomness of the sampling to make use of the statistical properties of a random sample. However, these random and pseudo-random sequences tend to cluster in places.
The significance of clustering is that some sections of the domain are excessively sampled while some sections are not sampled at all. Clustering may therefore introduce errors if a function varies much locally; this can be observed through the standard deviation term showing up in equation (9) of section 2.2.2.
The issue of clustering can be rectified by the use of quasi-random sequences which are designed to distribute points more uniformly over the domain (Figure

6). These sequences retain enough statistical properties to be used in the Monte Carlo method despite being deterministic.

In order to quantify the clustering, define a measure of how uniformly the points are distributed as

$$R_N(J) = \frac{1}{N}\#\{\mathbf{x}_n \in J\} - vol(J)$$

where $J \subset \Omega$ (Figure 7), N is the total number of points, Ω the unit hypercube, and $vol(J)$ is a volume-measure of J.
Let E be the set of all rectangular subsets of Ω and define the discrepancy:

$$D_N = \sup_{J \in E} \mid R_N(J) \mid .$$

Also define the starred discrepancy:

$$D_N^* = \sup_{J \in E^*} \mid R_N(J) \mid, \qquad E^* = \{J(0, \mathbf{x})\}.$$

The starred definition imposes the constraint that the rectangular subset E be anchored with a corner in the origin (Figure 8). This will become useful in proving the error in section 2.3.2.
Expressed in words, $R_N(J)$ is the percentage of the total amount of points lying in a domain J, minus the volume of J. D_N is the rectangular subset of J with largest difference between the percentage of points in the rectangle and the volume of the rectangle.

A random sequence has the discrepancy $N^{-1/2}$ while a low discrepancy sequence is defined as having discrepancy $D_N \leq c(logN)^k N^{-1}$.

There are several methods for generating low-discrepancy quasi-random sequences, such as Sobolt and Halton [6]. The only practical difference between applying Quasi-Monte Carlo as opposed to Monte Carlo is thus whether or not the sequence of points used in equation (8) is chosen to be quasi-random or pseudo-random.

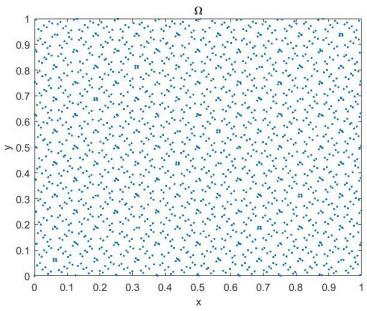

Figure 6: A quasi-random sequence in two dimensions.

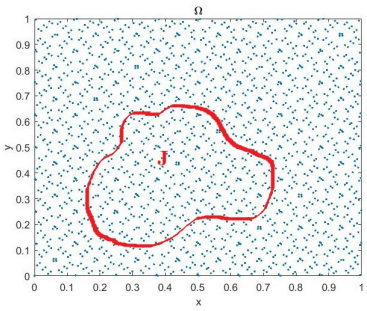

Figure 7: The set J.

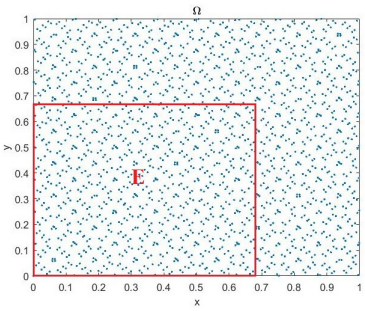

Figure 8: The set E^*.

2.3.2 Koksma-Hlawka inequality

The error of the Monte Carlo method was proven by the use of the statistical properties inherent in random sequences. With quasi-random sequences these results no longer apply.

Define the Hardy-Krause variation on $\Omega = [0,1]^d$ as

$$V[f] = \int_\Omega \left| \frac{\partial^d f}{\partial t_1 ... \partial t_d} \right| \, dt_1 ... dt_d + \sum_{i=1}^{d} V[f_1^{(i)}]$$

where the sum term restricts the function on the boundary.
There is then the more general theorem:

Theorem 4. *The error of the Monte Carlo approximation using a sequence of points with discrepancy D_N^* can be estimated as*

$$\epsilon[f] = |I[f] - I_N[f]| \leq V[f] D_N^*,$$

where $V[f]$ is the variation in the Hardy-Krause sense.

Proof. Express $R_N(J(0, \mathbf{x}))$ in the following form:

$$R_N(\mathbf{x}) = \left\{ \frac{1}{N} \sum_{n=1}^{N} \delta(\mathbf{x} - \mathbf{x}_n) - 1 \right\} d\mathbf{x}.$$

Consider a function f which is zero on the edge of the hypercube Ω. The error $\epsilon[f]$ is then given by

$$\begin{aligned} \epsilon[f] &= \left| \int_\Omega f(\mathbf{x}) d\mathbf{x} - \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x}_n) \right| = \left| \int_\Omega \left\{ 1 - \frac{1}{N} \sum_{n=1}^{N} \delta(\mathbf{x} - \mathbf{x}_n) \right\} f(\mathbf{x}) d\mathbf{x} \right| \\ &= \left| \int_\Omega R(\mathbf{x}) df(\mathbf{x}) \right| \leq (\sup_{\mathbf{x}} R(\mathbf{x})) \int_\Omega |\, df(\mathbf{x})| = D_N^* V[f]. \end{aligned}$$

□

This error depends on the discrepancy of the sequence of points chosen and on the variance of the integrand. In theory these can be calculated exactly which gives a hard bound on the error rather than the probabilistic error obtained for the Monte Carlo method.

2.3.3 Quasi-Monte Carlo error

The Koksma-Hlawka inequality proved the error in terms of the discrepancy. It is thus applicable for the Monte Carlo method as well as for the for Quasi-Monte Carlo method. To find the error of the Quasi-Monte Carlo method, apply Theorem 4 on a quasi-random low discrepancy sequence to obtain:

$$\epsilon_N[f] \leq cV[f](logN)^d N^{-1}.$$

And since c and $V[f]$ do not depend on N the error of the Quasi-Monte Carlo method is $O((logN)^d N^{-1})$.

The Koksma-Hlawka result gives an upper bound on the error but often overestimates. For an exact value of the error Wozniakowski has shown the following result:

Theorem 5.

$$E[\epsilon[f]^2]^{1/2} = T_N^* \qquad T_N^* = \Big[\int_\Omega R_N(J(0,\boldsymbol{x}))^2 d\boldsymbol{x}\Big]^{1/2}.$$

The proof of this theorem is given in [1].

Since the Koksma-Hlawka inequality is based on discrepancy it can also be applied to a random sequence thereby providing the error of the Monte Carlo method. A random sequence has the the discrepancy $cN^{-1/2}$ thus yielding the result $\epsilon[f] \leq cV[f]N^{-1/2}$ which is what one would expect from previous results.

The error of the Quasi-Monte Carlo method, $O((logN)^d N^{-1})$, has an exponential term involving d. This term however is logarithmic and will be dominated by the N^{-1} term in the denominator. The error will therefore behave as N^{-1} for large N. By comparison the Monte Carlo method had an error of $O(N^{-1/2})$, making the convergence of the Quasi-Monte Carlo method faster. Furthermore, the error is a strict upper bound as opposed to the probabilistic error of the Monte Carlo method.

The plots below display the errors in approximating the test integral for seven calculations (Figure 9) and the empirical standard deviation of the approximation (Figure 10).

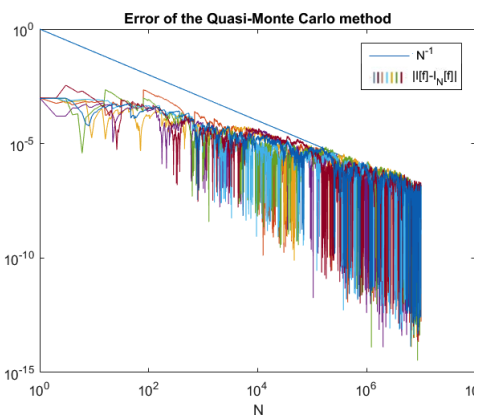

Figure 9: The difference between the N point Quasi-Monte Carlo approximation of the test-integral and the true value. Computed seven times.

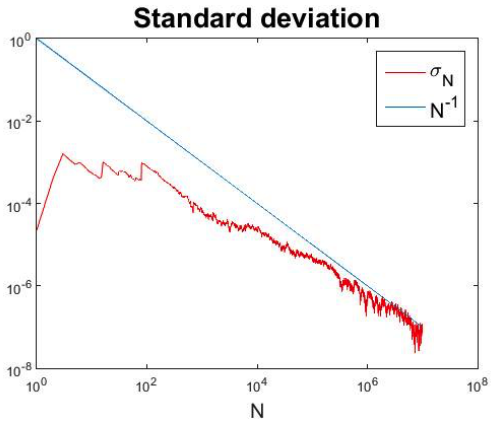

Figure 10: Standard deviation of the seven Quasi-Monte Carlo approximations.

2.4 Sparse grid method

Sparse grid methods are based on decomposition of vector spaces. Let $f(x) \in V$ where V is some vector space, then $f(x)$ can be expressed as a linear combination of the basis vectors spanning V. If V is decomposed into a set of subspaces representing different levels of refinement; it can be shown that the projection of $f(x)$ onto some of these subspaces contribute significantly less to the overall approximation of $f(x)$ than do others. These less contributing subspaces can consequently be discarded thereby reducing the total number of points needed to approximate $f(x)$.

2.4.1 Interpolation by piecewise linear basis functions

In order to interpolate a one-dimensional function $f(x)$ on $[0, 1]$, with $f(0) = f(1) = 0$, begin by considering the piecewise linear function:

$$\Phi(x) = \begin{cases} 1 - |x| & x \in [-1, 1] \\ 0 & x \notin [-1, 1] \end{cases}$$

If the interval is divided into subintervals of length (step size) $h_l = 2^{-l}$, where the level $l \in \mathbb{N}$ and $i = 0, ..., 2^l$, each level will correspond to a grid Θ_l with the grid nodes given by $x_{l,i} = i \cdot h_l$. Each of these nodes can be assigned a piecewise linear basis function $\Phi_{l,i}(x) := \Phi\left(\frac{x - i \cdot h_l}{h_l}\right)$ with support in $[x_i - h_l, x_i + h_l]$. These $\Phi_{l,i}$ will then make up a nodal basis for the interval $x \in [0, 1]$.

The number of nodes and consequently the step size will depend on which level is currently considered.
If only points interior to Ω are considered[4] the space V_l can be written as a span of basis-functions such that:

$$V_l = span\{\Phi_{l,i} : i = 1, ..., 2^l - 1\}.$$

The plots of the basis functions spanning V_1 and V_2 are shown below in Figure 11 and Figure 12 respectively.

[4]The reason for this has to do with homogeneous boundary conditions on $f(x)$.

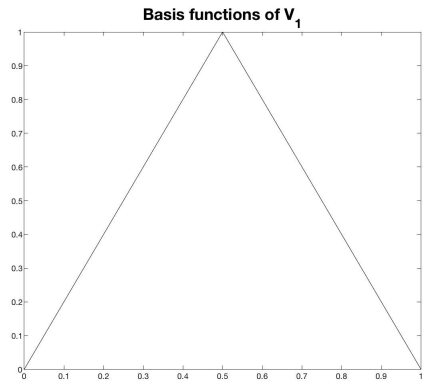

Figure 11: Basis functions of V_1: $\Phi_{1,1}$.

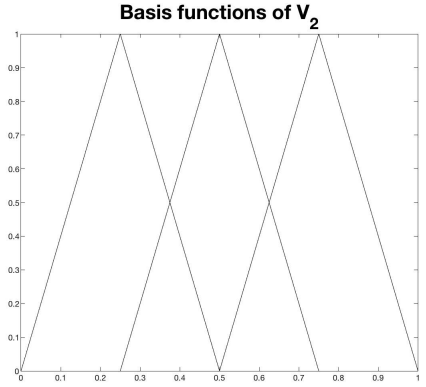

Figure 12: Basis functions of V_2: $\Phi_{2,1}$(left), $\Phi_{2,2}$(mid) and $\Phi_{2,3}$ (right).

A level l approximation of $f(x)$, denoted $u_l(x)$, can be formed by a linear combination of the basis-functions spanning V_l:

$$f(x) \approx u_l(x) = \sum_{j=1}^{2^l-1} f(x_j)\Phi_{l,j}(x)$$

This expression is reminiscent of the interpolation in section 2.1.1; it should therefore not come as a surprise that the piecewise linear basis $\Phi_{l,j}$ is called the Lagrange nodal basis.

The current construction of V_l involves overlapping basis functions. By defining the hierarchical increment space as

$$W_l := span\{\Phi_{l,i} : i \in I_l\},$$

$$I_l = \{i \in \mathbb{N} : i = 1, ..., 2^l - 1, i = odd\},$$

V_l can be constructed by a tensor product of hierarchical increment spaces whose supports are mutually disjoint

$$V_l = \bigotimes_{k \leq l} W_k.$$

A graphic illustration of the basis functions spanning the hierarchical basis corresponding to level 1 and 2 is displayed in Figure 13 and Figure 14.

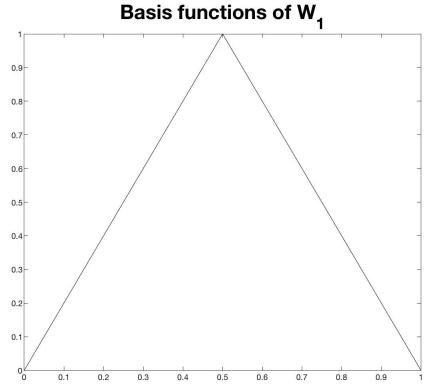

Figure 13: Basis functions of W_1: $\Phi_{1,1}$.

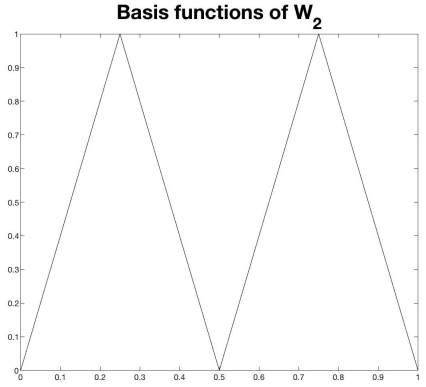

Figure 14: Basis functions of W_2: $\Phi_{2,1}$(left) and $\Phi_{2,3}$ (right).

In the W_l basis the level l approximation of $f(x)$ can be written:

$$f(x) \approx u_l(x) = \sum_{k=1}^{l} \sum_{i \in I_k} v_{k,i} \cdot \Phi_{k,i}(x).$$

Generalizing the piecewise linear interpolation to higher dimensions is done by introducing multi indices.

Define the multi index level as $\mathbf{l} := (l_1, ..., l_d) \in \mathbb{N}^d$ and let the domain be the d-dimensional unit hypercube $\Omega = [0,1]^d$. Further define the operation $\mathbf{a} \leq \mathbf{b} \Rightarrow a_j \leq b_j \forall 1 \leq j \leq d$ for $\mathbf{a}, \mathbf{b} \in \mathbb{N}^d$

The multi-dimension equivalent to step size is the mesh size $\mathbf{h_l} := (h_{l_1}, ..., h_{l_d}) := \mathbf{2^{-l}} := (2^{l_1}, ..., 2^{l_d})$ for which the grid points are given by $x_{\mathbf{l},\mathbf{i}} = (x_{l_1,i_1}, ..., x_{l_d,i_d}) = \mathbf{i} \cdot \mathbf{h_l}$, $\mathbf{1} \leq \mathbf{i} \leq \mathbf{2^l} - \mathbf{1}$.
These definitions allows for the description of the level $\mathbf{l}$ grid $\Theta_\mathbf{l}$ of mesh size $\mathbf{h_l}$ where each grid point $x_{\mathbf{l},\mathbf{i}}$ will have the basis functions $\Phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) := \prod_{j=1}^{d} \Phi_{l_j,i_j}(x_j)$, $\mathbf{x} \in \mathbb{R}^d$, associated with it.

As was the case in one dimension these basis functions will span a space:

$$V_\mathbf{l} = span\{\Phi_{\mathbf{l},\mathbf{i}} : \mathbf{1} \leq \mathbf{i} \leq \mathbf{2^l} - \mathbf{1}\}$$

with the hierarchical increment space defined as:

$$W_\mathbf{l} := span\{\Phi_{\mathbf{l},\mathbf{i}} : \mathbf{i} \in \mathbf{I_l}\},$$

$$\mathbf{I_l} = \{\mathbf{i} \in \mathbb{N}^d : \mathbf{1} \leq \mathbf{i} \leq \mathbf{2^l} - \mathbf{1}, i_j = odd \quad \forall \quad 1 \leq j \leq d\}.$$

Analogous to one dimension $V_\mathbf{l}$ can now be composed by the tensor product of the increment spaces $W_\mathbf{l}$:

$$V_\mathbf{l} = \bigotimes_{\mathbf{k} \leq \mathbf{l}} W_\mathbf{k}.$$

The function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$, can then be approximated by:

$$f(\mathbf{x}) \approx u_{\mathbf{l}}(\mathbf{x}) = \sum_{\mathbf{k}=\mathbf{1}}^{l} \sum_{\mathbf{i} \in \mathbf{I_k}} v_{\mathbf{k},\mathbf{i}} \cdot \Phi_{\mathbf{k},\mathbf{i}}(\mathbf{x}).$$

It can now be shown that some of the terms in this sum can be dropped without incurring much error in the approximation of $u_{\mathbf{l}}(\mathbf{x})$. More precisely, it can be shown that the hierarchical coefficients decay as $|v_{\mathbf{l},\mathbf{i}}| = O\left(2^{-2|\mathbf{l}|_1}\right)$ [5], where $|\mathbf{l}|_1 := \sum_j^d l_j$.
An optimization of cost to benefit ratio for these coefficients will lead to the sparse grid space of level l

$$\hat{V}_l = \bigotimes_{|\mathbf{l}|_1 \leq l+d-1} W_{\mathbf{l}}.$$

Compared to the original, full space of level l:

$$V_{\mathbf{l}} = \bigotimes_{\mathbf{k} \leq \mathbf{l}} W_{\mathbf{k}}.$$

This, sparser, grid will have a computational order of $O(2^l l^{d-1})$ with accuracy $O(h_l^2 l^{d-1})$; as opposed to the full grid computational order of $O(2^{ld})$ with accuracy $O(h_l^2)$ [9].

2.4.2 Chebyshev polynomials

In the previous section the nodes were spaced equidistant.
We can instead use the zeroes of Chebyshev polynomials as nodes. These nodes have two convenient properties that the equidistant nodes lack.

Firstly, if the zeroes of the Chebyshev polynomials are used, the nodes will be nested which means that the nodes of lower levels are also included in the higher levels.
Secondly, smaller approximation error is incurred if the nodes are zeroes of the Chebyshev polynomial.

A plot of level $l = 0$ through $l = 6$ using chebyshev nodes is presented in Figure 15.

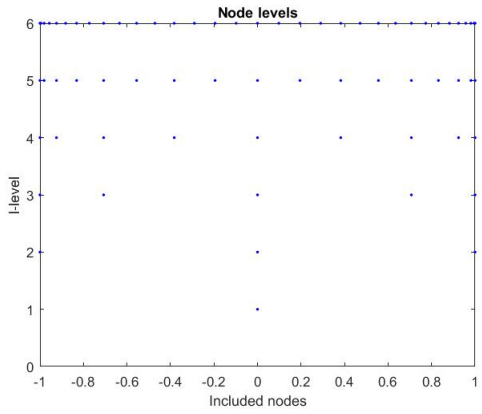

Figure 15: Chebyshev nodes.

Below is shown a sparse grid of level 7 using Chebyshev nodes in two dimensions and three dimensions respectively.

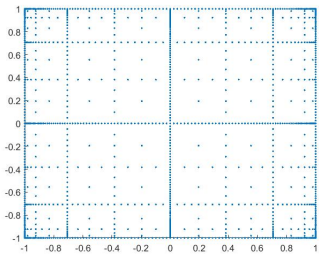

Figure 16: Two-dimensional sparse grid.

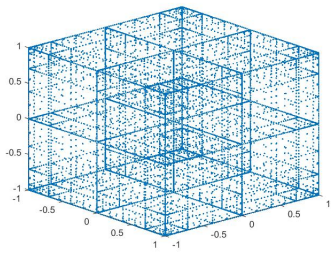

Figure 17: Three-dimensional sparse grid.

2.4.3 Integration on a sparse grid

Consider the problem of integrating a function on a sparse grid.
For a grid of level l and dimension d the number of points are denoted n_l^d. The quadrature rule in one dimension is then written as follows:

$$Q_l^1 f := \sum_{i=1}^{n_l^1} w_{li} f(x_{li}).$$

Define the difference quadrature formula as

$$\Delta_k^1 f := (Q_k^1 - Q_{k-1}^1) f, \qquad Q_0^1 f := 0.$$

Then Smolyak found[8] that one could construct the level l sparse grid quadra-

ture rule for a d-dimensional function on $[0,1]^d$ as:

$$\begin{aligned}\hat{Q}_l^d f :&= \sum_{|\mathbf{k}|_1 \leq l+d-1} (\Delta_{k_1}^1 \bigotimes \ldots \bigotimes \Delta_{k_d}^1) f \\ &= \sum_{|\mathbf{k}|_1 \leq l+d-1} ((Q_{k_1}^1 - Q_{k_1-1}^1) \bigotimes \ldots \bigotimes (Q_{k_d}^1 - Q_{k_d-1}^1)) f\end{aligned}$$

which can be rewritten as a linear combination of one-dimensional quadrature rules:

$$\hat{Q}_l^d f = \sum_{l \leq |\mathbf{k}|_1 \leq l+d-1} (-1)^{l+d-|\mathbf{k}|_1-1} \binom{d-1}{|\mathbf{k}|_1 - l} \left(Q_{k_1}^1 \bigotimes \ldots \bigotimes Q_{k_d}^1\right) f.$$

2.4.4 Sparse grid error

The error will depend on which basis is used for the sparse grid, how many times differentiable the function is and what quadrature rule is being used. For the Chebyshev sparse grid with the Smolyak construction quadrature rule and a r times differentiable function the error will be[2]:

$$O\big(N^{-r}\big(log(N)\big)^{(d-1)(r+1)}\big).$$

The error when computing the test integral with the sparse grid method is plotted in Figure 18.

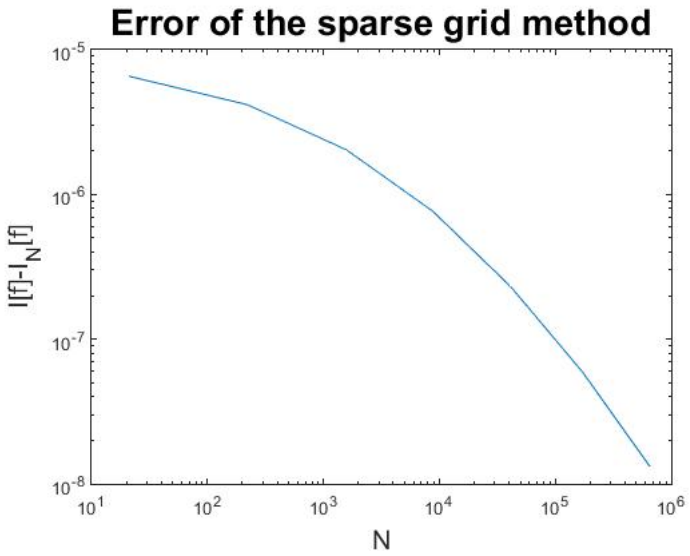

Figure 18: The difference between the N point sparse grid approximation of the test-integral and the true value.

3 How to throw a ball

3.1 Problem statement

The problem is to find the coordinates of impact for a ball thrown from the origin with some initial velocity. The ball will obey Newton's equations of motion with gravitational forces and forces due to wind:

$$m\frac{d^2\mathbf{x}}{dt^2} = -c\left|\frac{d\mathbf{x}}{dt} - \mathbf{w}(z)\right|\frac{d\mathbf{x}}{dt} - g\mathbf{e}_z$$

where $\mathbf{w}(z)$ is the wind velocity defined as:

$$\mathbf{w}(z) = (w_0 + w_1 z)\mathbf{e}_y$$

with the initial values:

$$\begin{cases}\mathbf{x}(0) = H\mathbf{e}_z \\ \frac{d\mathbf{x}(0)}{dt} = v_0\hat{\mathbf{n}}(\theta,\alpha)\end{cases}$$

Denote the coordinate where the ball hits the ground by $\mathbf{X}^* = (X^*, Y^*)$.
This coordinate is a function of all the parameters such that $\mathbf{X}^* = \mathbf{X}^*(m, c, w_0, w_1, g, H, v_0, \theta, \alpha)$.

If the parameters are taken to be stochastic variables uniformly distributed throughout their respective intervals the expectation value of the impact coordinates will be given by:

$$E[\mathbf{X}^*] = \frac{1}{V}\int_{I_m}\int_{I_c}\int_{I_{w_0}}\int_{I_{w_1}}\int_{I_g}\int_{I_H}\int_{I_{v_0}}\int_{I_\theta}\int_{I_\alpha}\mathbf{X}^*(m, c, w_0, w_1, g, H, v_0, \theta, \alpha)\quad dmdcdw_0dw_1dgdHdv_0d\theta d\alpha$$

where V is the volume of the domain.

The parameter intervals are shown in the table below.

m	$\in$	$[0.4, 0.6]$	α	$\in$	$[35°, 55°]$
V_0	$\in$	$[20, 30]$	θ	$\in$	$[40°, 50°]$
H	$\in$	$[2.7, 3.3]$	c	$\in$	$[0.01, 0.02]$
w_0	$\in$	$[0.098, 0.102]$	g	$\in$	$[9.77, 9.87]$
w_1	$\in$	$[0.05, 0.1]$			

3.2 Trapezoidal rule

The practical implementation of the trapezoidal rule for high-dimensional integrals can be directly adopted from the theory by employing nested sums. The problem involved a 9-dimensional integral which by using the trapezoidal rule on a n^9 grid can be approximated by:

$$\sum_{I_1}^{n}...\sum_{I_9}^{n} w_{I_1}...w_{I_9}f(x_{I_1},...,x_{I_9})h_{I_1}...h_{I_9}$$

where the index is defined by

$$I := (i_m, i_{V_0}, i_H, i_{w_0}, i_{w_1}, i_\alpha, i_\theta, i_c, i_g).$$

The curse of dimensionality was discussed in the introduction and it was shown that for the choice of n nodes per direction, the resulting total number of nodes would be $N = n^d$; which in this case translates to $N = n^9$. The total number of points N for the first five grids using n nodes in each direction are displayed in the table below.

n	N
2	512
3	19 683
4	262 144
5	1 953 125
6	10 077 696

The rapid growth of N means that the problem can only be solved by the trapezoidal method if a very coarse grid is used.

For two nodes which corresponds to the start and endpoint of the interval in each dimension, we obtained the following answer as the impact coordinate:

$$E[\mathbf{X}^*] = (46.464762257851014, 56.239686132477907).$$

While for four nodes the answer was:

$$E[\mathbf{X}^*] = (45.656057286682369, 54.777822933694480).$$

Higher number of nodes than this proved impossible since the total number of points grew too large.

3.3 Monte Carlo

Of the methods covered in this paper, the Monte-Carlo method is by far the easiest to implement.
`MATLAB` has a built in method for generating pseudo-random sequences and doing this once for each dimension provides the pseudo-random d-dimensional coordinates in which the function is evaluated. The expectation value is then formed by summing these function-evaluations and dividing by the number of points.

For a sequence of 361249 points the Monte Carlo method calculated the impact coordinate:

$$E[\mathbf{X}^*] = (45.562466123185963, 54.608090981379242)$$

with the error shown in Figure 19.

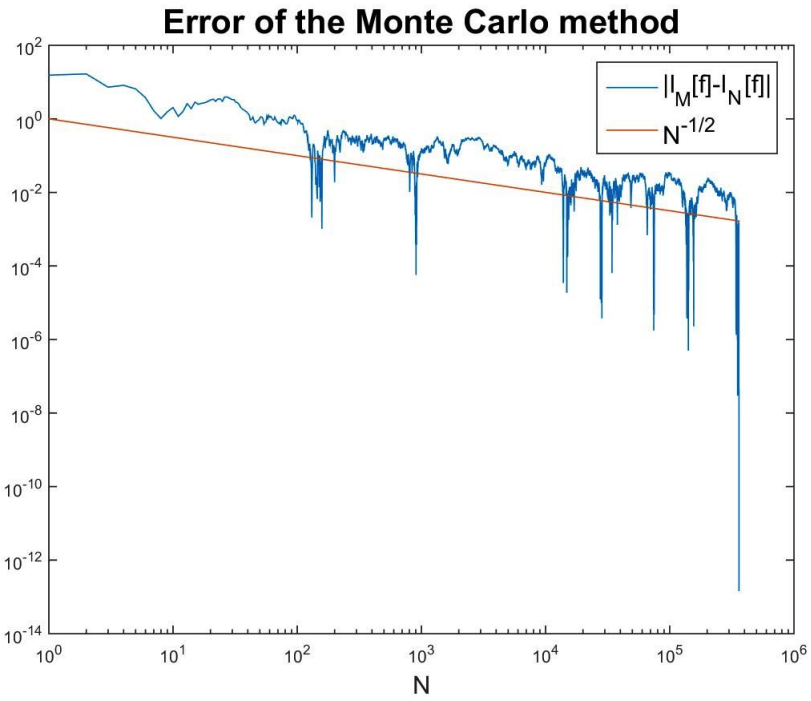

Figure 19: The difference between the N point approximation of the impact coordinate and the $M = \sup N$ point approximation using the Monte Carlo method.

3.4 Quasi-Monte Carlo

Implementation of the Quasi-Monte Carlo method is nearly as straight forward as that of the Monte Carlo. The only difference is how the d-dimensional coordinates are generated. This is done by the built in function `sobolset`. The dimension is specified when the function is called thus guaranteeing that the nodes in each direction are not the same.
One thing to keep in mind is that these sequences are deterministic. When running consecutive evaluations the results will therefore be identical to each other. To obtain different sequences for each trial, choose to skip some of the points in the generated sequence.

Quasi Monte Carlo using 361249 nodes gave the answer:

$$E[\mathbf{X}^*] = (45.539267438784286, 54.577233126045329),$$

with the error presented in Figure 20.

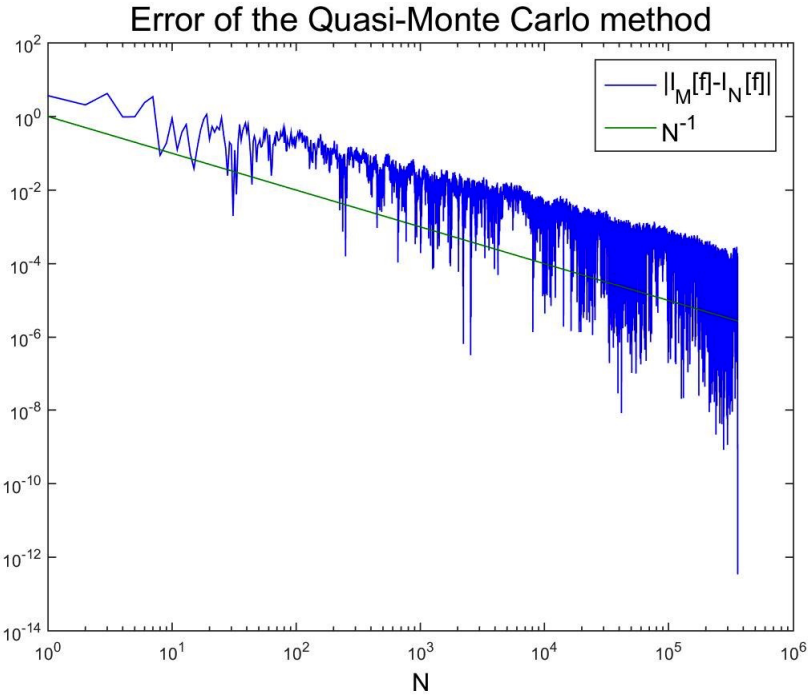

Figure 20: The difference between the N point approximation of the impact coordinate and the $M = \sup N$ point approximation using the Quasi-Monte Carlo method.

3.5 Sparse grid

The sparse grid method is the most challenging method to implement but once implemented it is also the most powerful.

The level l corresponds to how many nodes are used in each dimension. We used the zeroes of the Chebyshev polynomials as nodes (Figure 15). These can be generated by the free `MATLAB` package `chebfun`.[3]
If $l = 3$ there would be 5 nodes which are the zeroes of the degree 5 Chebyshev polynomial. These nodes are then placed in a vector denoted W_l.

The grid is constructed by adding together all the tensor-products of these vectors W_l (in `MATLAB` the equivalent of tensor multiplication is performed by the function `ndgrid`) such that the sum of the levels l are equal to $l + d$ where d is the dimension; for our problem $d = 9$. A few nodes will occur multiple times and are removed with the built in `MATLAB` function `unique`.

The weights are computed in a similar way but rather than satisfying the above equality the tensor product now has to satisfy the condition that the sum of the subscripts be less than or equal to $l + d$: $\sum_i l_i \leq l + d$. This is done by first calculating the weights for the lowest level $l = 1$. As the next level weights are calculated the difference is examined and duplicates are discarded.

The integrand is a second order differential equation. To solve for the impact coordinate we initially used `ODE45`. While this method gave an answer close to those of the other methods, the error plot did not coincide well with the theory (Figure 21). Without comparing the result to those given by the other methods we would have no way of assessing the veracity of the result.

The error behaved as predicted when the sparse grid method was applied to the test-integral. We thus conclude that the error is made in connection with the integrand. The error improves significantly (Figure 22) if a Runge-Kutta 4th order method (RK4) with fixed time step is used as opposed to `ODE45` which has a variable time step. Further reducing the fixed time step improves the error even further although at great cost to computational time.

Below are shown the results obtained by using 19 and 100897 nodes with the `ODE45` and RK4 respectively.

	ODE45
19 nodes	$E[\mathbf{X}^*] = (45.57986220630692, 54.626206031338867)$
100897 nodes	$E[\mathbf{X}^*] = (45.539694069466975, 54.5782838325925)$

	RK4
19 nodes	$E[\mathbf{X}^*] = (45.531503116980126, 54.536620421334284)$
100897 nodes	$E[\mathbf{X}^*] = (45.497265419819804, 54.497749774031199)$

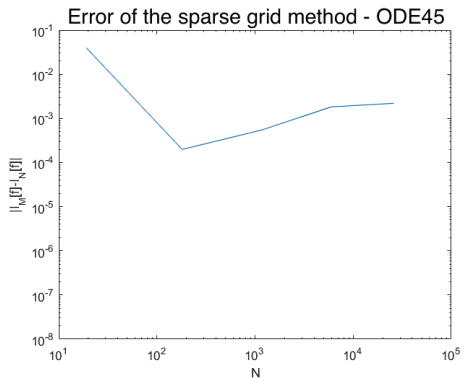

Figure 21: The difference between the N point approximation of the impact coordinate and the $M = \sup N$ point approximation using the sparse grid method and ODE45.

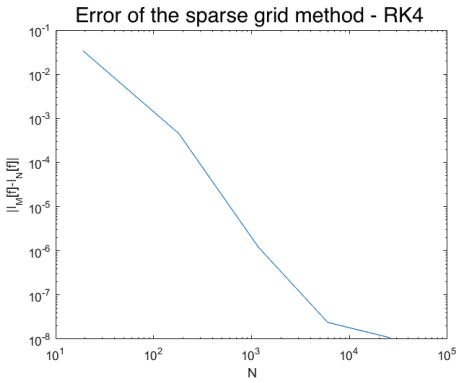

Figure 22: The difference between the N point approximation of the impact coordinate and the $M = \sup N$ point approximation using the sparse grid method and RK4.

4 Comparison

Our findings have coincided closely with the theory. The trapezoidal rule quickly reaches values of N too large for the computer to handle and gives an answer which deviates somewhat from the answers acquired by the other methods, although surprisingly little considering the course grid used.
The two Monte Carlo methods produces good answers for a reasonable N with Quasi-Monte Carlo converging somewhat faster, again coinciding closely with the theory developed. The sparse grid method converges to a seemingly accurate answer significantly faster than the others methods despite using far fewer nodes.

The sparse grid method displayed issues with the convergence when the second order differential equation was calculated with the `ODE45` method. This issue was resolved by the introduction of a Runge-Kutta 4th order method. Since the `ODE45` method worked properly for the other methods we are led to believe that the issue was caused by too large derivatives arising as a consequence of the adaptive step length employed by the `ODE45`.

Below are the results for the different methods.

Method	**Nodes**	$\mathbf{E}[X^*]$	$\mathbf{E}[Y^*]$
Trapezoid	262144	45.656057286682369	54.777822933694480
Monte Carlo	361249	45.562466123185963	54.608090981379242
Quasi-Monte Carlo	361249	45.539267438784286	54.577233126045329
sparse grid (RK4)	100897	45.497265419819804	54.497749774031199

A comparison of the actual errors for the different methods applied to the test integral is shown in Figure 23 while a comparison of the empirical errors for the ball problem are plotted in Figure 24.

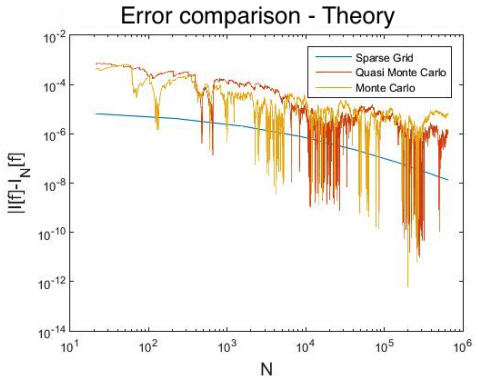

Figure 23: Comparison of the difference between the N point approximation of the test-integral and the true value computed by the different methods.

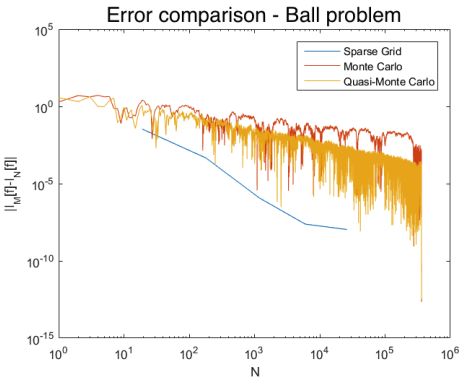

Figure 24: Comparison of the N point approximation of the impact coordinate subtracted from the $M = \sup N$ point approximation for each different method.

References

[1] Russel E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7:1–49, 1998.

[2] Paul Constantine. Experiences with sparse grids and Smolyak -type approximations. Lecture notes, 1 2012.

[3] T. A Driscoll, N. Hale, and L. N. Trefethen. *Chebfun Guide*. Pafnuty Publications, 2014.

[4] William Feller. *An introduction to probability theory and its applications*, volume 4 of *10*. The name of the publisher, The address, 3 edition, 7 1993.

[5] Michael Griebel and Hans-Joachim Bungartz. Sparse grids. *Acta Numerica*, 13:1–123, 2004.

[6] William J. Morokoff and Russel E. Caflisch. Quasi-random sequences and their discrepancies. *Journal of scientific computing*, 15(6):1251–1279, 11 1994.

[7] Olof Runborg. Notes on numerical integration, 2003.

[8] S.A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics Doklady*, 4:240–243, 7 1963.

[9] M. Griebel T. Gerstner. Sparse grids. In *Encyclopedia of Quantitative Finance*. Wiley, 2008.