InFoMM
Industrially Focused
Mathematical Modelling

Engineering and
Physical Sciences
Research Council

arm nag®

Nested multilevel Monte Carlo methods and a modified Euler-Maruyama scheme utilising approximate Gaussian random variables suitable for vectorised hardware and low-precisions

Oliver Sheridan-Methven
oliver.sheridan-methven@hotmail.co.uk
Mansfield College

Thursday 21st January 2021

Supervisors:

Prof. Michael Giles	Oxford
Dr Christopher Goodyer	Arm

A thesis submitted for the degree of
Doctor of Philosophy
as part of the centre for doctoral training in
Industrially Focused Mathematical Modelling.

Contents

Acknowledgements

There are several people and groups I would like to acknowledge and thank for their assistance and contributions towards the completion of this thesis, of which only some are explicitly named here.

The first acknowledgement that should be made is to those who have financially sponsored this work, as without their funding and financial support this project would not have been possible at all. This includes the Engineering and Physical Sciences Research Council (EPSRC) and Oxford University's centre for doctoral training in Industrially Focused Mathematical Modelling (InFoMM), with the EP/L015803/1 funding grant. Furthermore, this project (and another smaller project) was jointly funded by Arm and NAG, with Arm being the primary sponsor for this thesis. Additionally, funding for conferences to present this work and technical training courses was also provided by the Inference, Computation and Numerics for Insights into Cities (ICONIC) project, and the programme grant EP/P020720/1, and also by Mansfield College Oxford and their graduate travel (and book) grants.

Several people have dedicated their time and resources to developing and shaping the content of this thesis. The most notable of these whom I am most indebted to is my academic supervisor Prof. Mike Giles, who has spearheaded this work from its inception, and read through many drafts of this thesis in various stages of completion. Similarly, my industrial supervisor, Dr Christopher Goodyer, coordinated Arm's involvement and also read several draft chapters of the thesis. There is my numeric analysis research group, the InFoMM staff and students, and also the stochastic simulation group, who all received numerous presentations on this project, and whose questions (and scrutiny) helped steer this work. Similarly, both Arm and NAG have hosted various presentations of this work and provided valuable feedback from an industrial and practitioners' perspective, which proved equally valuable and gave important insights pertinent to bridging the gap between theory and implementation.

This research has had access to various items of software and hardware, which would not have been possible without the generosity of various groups. This includes NAG providing free access to their C library, Arm for providing access to their libraries, compilers, ThunderX2 machines, emulators, and SVE capable hardware. Also, both Arm and NAG provided assistance from their technical team members, without whom several of these experiments would likely not have been achievable. Additionally, Wes Armour from the Oxford e-Research Centre assisted in the setup and hosting of a Nvidia Jetson Xavier machine for running our half-precision experiments.

This thesis has not gone unchecked, and the thesis' final assessors, Prof. Klaus Ritter and Dr Yuji Nakatsukasa, provided many valuable and acute discussions, criticisms, corrections, and explanations. Furthermore, I would like to mention the assessors for the thesis' transfer and confirmation of status: Prof. Patrick Farrell, Dr Abdul-Lateef Haji-Ali, Dr Yuji Nakatsukasa, and Prof. Christoph Reisinger. Their input, criticisms, and comments were all extremely helpful in sharpening the focus of the thesis and pointing out which topics and results required highlighting or more careful and detailed discussion.

Not only have the aforementioned individuals, teams, research groups and

departments provided their technical assistance to this project, they have provided their time, attention, patience, and enthusiasm to my research project. This immediately fostered and sustained a research environment which was both exciting and encouraging to work in. Similarly, I would like to thank my family and friends. This thesis has proved to be a considerable undertaking, and having their encouragement and support during this project has helped to keep me focused and highly motivated.

Abstract

We present a modified Euler-Maruyama scheme using approximate random variables, produced by the inverse transform method, using cheap approximations to the inverse Gaussian cumulative distribution function. We analyse the error for two approximations: a piecewise constant approximation on equally spaced intervals, and a piecewise linear approximation using geometric intervals dense at the singularities. High speed implementations faster than Intel's MKL are provided, suitable for modern vector hardware. The error between the approximations from the exact and modified Euler-Maruyama schemes is bounded by the error from the approximate random variables.

We incorporate this scheme into a multilevel Monte Carlo framework producing a nested scheme, and show that the discretisation error couples to the random variables' approximation error. The result directly extends to Lipschitz and differentiable payoff functions. For Lipschitz and non-differentiable payoff functions simulated using a time step δ, there is a transition from a variance decay of $\mathcal{O}(\delta)$ to $\mathcal{O}(\delta^{1/2})$ as the discretisation error becomes dominant. These variance bounds are demonstrated numerically for geometric Brownian motion and a variety of payoff functions of varying smoothness.

For approximate random variables computed in low-precision, a model for the accumulated rounding error is developed and assessed. Half-precision is viable for a range of coarse path simulations, and can be extended further by incorporating a Kahan compensated summation. We empirically demonstrate these ideas are transferable to the Milstein scheme, and the more difficult Cox-Ingersoll-Ross process and its non-central χ^2-distribution. We estimate that under the Black-Scholes model, options can be priced using path simulations with approximate Gaussian random variables, obtaining a five-times or more speed improvement without losing accuracy.

Chapter 1

Introduction

The remit of this thesis is quite wide in nature, covering both the practicalities of producing high-performance code, and the mathematical analysis associated with using vectorised and reduced-precision capable hardware. This thesis is intended for a reader with a high level of mathematical knowledge, and so we assume a reasonable degree of familiarity with topics including stochastic calculus, Monte Carlo methods, and similar areas. However, we do not expect the reader to necessarily have much knowledge of the lower level details underlying the execution of computer programs at both a programming and hardware level. Consequently, we will introduce these topics more gradually and with less familiarity assumed.

In this chapter we provide background material introducing multilevel Monte Carlo, vectorised arithmetic, and reduced-precision calculations. We then highlight some motivating applications driving this research and present relevant literature surrounding the topic.

1.1 Objectives of the thesis

This thesis is the culmination of a research project from the centre for doctoral training in industrially focused mathematical modelling. As such, this project originated from industrial considerations and subsequently progressed along a research trajectory which was both aligned with, and influenced by, the interests of the industrial partners who sponsored this project. The primary industrial sponsor was Arm, although NAG also had a smaller share in the evolution of this project. These two represent a vast industrial body spanning the design of instruction sets, hardware, libraries, and multiple other technologies. Consequently, the origin for the research contained within this thesis is very applied and computational, and throughout this work we tried to keep the results and analysis applicable to industry practitioners alongside mathematicians and scientists in academia.

The industrial objective for this project was quite straightforward: can any branches of mathematics be adapted to benefit from hardware which is capable of performing vectorised arithmetic and working in low-precisions? The mathematical result of this has been to isolate the branch of mathematics surrounding the simulation of stochastic systems, and develop mathematical frameworks which can fully utilise this modern hardware to perform faster computations whilst retaining the same level of accuracy. We have pursued this field of stochastic simulation as it represents a huge cross section of large scale industrial and scientific computing, with examples including weather forecasting and financial modelling to name just two. Additionally, from a mathematical perspective, we have developed approximations and numerical schemes demonstrating how to realise these benefits, and rigorous numerical analysis to ensure convergent results whilst no accuracy is lost. This has of course built on and extended the work of others, but also provided new results and insights into the numerical schemes, such as understanding the variance of a four-way multilevel Monte Carlo correction.

One objective we have satisfied is to take a very prevalent and ubiquitous simulation setting and analyse this in great detail. We have analysed stochastic path simulations for Monte Carlo estimation and have broken this down into its component procedures. We have sped up the computational bottlenecks, modified the numerical scheme to incorporate our modifications, analysed the resultant error introduced, and presented a multilevel Monte Carlo framework which can compensate for this, ensuring the speed improvements are maintained without losing accuracy. The industrial achievements in this framework were constructing approximate inverse cumulative distribution functions and providing extremely fast implementations across a range of modern hardwares. These implementations are empirically verified to supersede the exact functions provided from scientific and commercial library vendors, and have mathematical bounds on the approximation error. The specific distribution we have considered is the Gaussian distribution, because of its ubiquitous nature in science and prevalence in real world stochastic simulations. Looking at this distribution we have been able to go into a greater depth of analysis than could likely be achieved for most other distributions. While we anticipate other distributions may offer greater computational savings, we also anticipate their analysis may be harder, if at all possible.

Furthermore, we have taken these mathematical approximations and fully incorporated these into equally common stochastic simulation schemes, complete with implementation details and any new considerations arising from our approximations. The scheme we consider is the Euler-Maruyama scheme, as it is both a tool frequently used due to its applicability for a wide class of problems where other schemes might fail, and also because it allows the analysis to go to a greater depth than could likely be achieved with other schemes. This modified scheme has been fully incorporated into a nested multilevel Monte Carlo framework which has provided us with the means of realising the speed improvements of the approximations and still retaining the original accuracy.

In all of the stages outlined, we have been able to satisfy both our industrial and academic objectives. Our industrial objectives of fully utilising vectorised and low-precision capable hardware are satisfied from our approximations of the inverse cumulative distribution function of the Gaussian distribution. Secondly, for our industrial partners we show how one can utilise these in a stochastic simulation framework by modifying the Euler-Maruyama scheme. Lastly, we can reassure practitioners of the veracity of their answers both by our empirical demonstrations and by our accompanying body of numerical analysis. Throughout this we have been able to also meet our academic objectives with the development of high speed approximations and a thorough analysis of our modified numerical scheme for stochastic simulation with a nested multilevel Monte Carlo framework.

1.2 Monte Carlo and stochastic simulation

It is a frequent task in mathematics and science to seek to describe systems whose behaviour and evolution is subject to random and probabilistic influences. When a mathematically closed form solution is either impossible or intractable then it is not uncommon to either approximate or estimate this behaviour. For a system which is subject to random behaviour it is not uncommon to seek statistics which can quantify the behaviour, such as the mean, variance, etc. Many of these are expressible as an expectation of some function of the system under consideration. The Monte Carlo method is a procedure of estimating these quantities by means of simulations, where increasing the number of simulations which are used will act to improve the estimate, as should increasing the fidelity of the simulations. For a comprehensive detailing of stochastic simulation we recommend Asmussen and Glynn [13], while for a thorough exposition of Monte Carlo methods for financial applications we recommend Glasserman [92].

Whilst originating from and prevalent in probabilistic settings, the Monte Carlo method has a variety of applications outside of this domain. It is straightforward to cast the Monte Carlo method as a numerical integration technique [56, 92, 162] ideal for high dimensional problems, and through the Feynman-Kac [127] theorem it has become a highly useful tool for the solution of some classes of partial differential equations. Lastly, Monte Carlo methods have several applications in applied statistics and Bayesian inference [75, 197], especially with the development of Markov chain Monte Carlo [73].

1.2.1 Multilevel Monte Carlo

Since the development of multilevel Monte Carlo for parametric integration by Heinrich [104] and approximating solutions to stochastic differential equations (SDEs) by Kebaier [130] and Giles [76], the multilevel methodology has been extended to a large class of stochastic modelling. These cover partial differential equations (PDEs) with stochastic coefficients [17, 48], stochastic PDEs [84], continuous-time Markov process models of biochemical reactions [9, 10], Markov chain Monte Carlo [115, 187], nested simulation [37, 81], probability density estimation [23, 88], and several other areas. A comprehensive review of multilevel Monte Carlo research is provided by Giles [79].

In this work, we focus on a branch of multilevel Monte Carlo which utilises approximate random variables. Typical to most classes of Monte Carlo simulation, random variables need to be drawn from well known and understood distributions, which typically include the uniform, Gaussian, Poisson, (non-central) χ^2, etc. Using the inverse transform method to map uniform random variables to the desired distribution, we will analyse the convergence of the Euler-Maruyama scheme when the exact inverse cumulative distribution function (CDF) is replaced by an approximation, producing what we call approximate random variables. The motivation for this is that several of these near exact implementations (with respect to machine precision) are computationally very expensive, and typically their implementations are ill-suited to modern vectorised processors or graphical processing units. The cost of generating the exact random variables can frequently be the computational bottleneck, and so our approximate random variables are designed to be much faster to compute. We will show that a nested multilevel Monte Carlo framework which utilises these approximate random variables can be adopted for several common estimation problems requiring the simulation of SDEs. We will present the variance analysis of the multilevel Monte Carlo correction that is introduced when compensating for the inverse Gaussian CDF's approximation's error.

Naturally there is much work from the mathematical community surrounding the convergence of numerical schemes for stochastic differential equations and Monte Carlo methods [107, 108, 117, 123, 133, 172, 173]. The most relevant to this research though is an interesting cost model for reduced-bit algorithms presented by Giles et al. [89, 90]. While this research is under the setting of information based complexity, their work produces a complexity and cost model for generating random bits. This work naturally leads to the use of quantised Gaussian random variables, which is the first approximation we present, and an approximation we use in our numerical simulations. In this work some of our analysis slightly extends parts of theirs, although our focus of determining the overall multilevel Monte Carlo cost is quite different.

Another work related to the topics we present is by Müller et al. [160]. In their work they use three and four-point approximations for the Gaussian distribution, equivalent to a piecewise constant approximation using three to four intervals of differing sizes. The partitioning of the unit interval is chosen so the first few moments of the approximation and the exact distribution match. However, there are two limitations to their work. The first is that they present the analysis for the weak-error for the modified payoff estimates rather than the strong-error [160, Section 6], where the latter is often necessary for the analysis and bounding of the

small variance between fine and coarse simulations for multilevel Monte Carlo [78, page 17] [76]. The second pitfall is that their work fails to strictly respect the telescoping summation central to the formulation of multilevel Monte Carlo. For the fine level's exact Gaussian increments the combined and rescaled coarse increment follows the same distribution. However, for approximate random variables, while the rescaled summation producing the coarse path's random increment still forms an approximate Gaussian random variable, it does not necessarily follow the exact same distribution as its constituents fine path's random increments. In our work we recognise and account for this subtlety in our analysis, and comment that this forces our framework to be interpreted as a nested multilevel Monte Carlo and not a multi-index Monte Carlo [99].

1.3 Vectorised arithmetic

The typical computational model usually assumed when performing a series of tasks is that of *serial processing*. When a program has been compiled for serial processing, the executable produced will issue several instructions to the processor, which in most computing tasks will typically take the form: read in a datum, perform work on this datum, and write a result. This is repeated on each data element one after the other.

However, in several applications the work that needs to be performed on the data is identical for each datum and independent from other data. If this is the case then there are specialised instructions the compiler can issue, which the processor can then execute on its *vector processing unit* (VPU). The advantage of these units is that they can perform the work on several data at once, in approximately the same amount of time the serial processing takes for an individual datum. The greater the number of items the vector can hold then the greater the degree of parallelisation and the greater the performance improvement. Performing computations using this vector hardware is known as *vectorised arithmetic*.

Unfortunately there are several pitfalls and common use cases which can inhibit code from being vectorised, or result in badly vectorised code. Such issues range from memory access patterns, loop dependencies, nested parallelisation, etc. One of the most prolific factors is conditional execution, where the different vector elements require different work, which can dominate the performance of vectorised code, as we will see and discuss later in this thesis.

1.4 Reduced-precision

Over the decades there has been a trend in computing to spend ever larger amounts of time to obtain more accurate answers. As part of this, the degree of precision which computers use to store numbers, be they integers or floating-point, has also increased, moving from using 32-bit representations up to 64-bit (and even higher in some cases). As the level of precision has increased we have required evermore complicated algorithms to achieve such levels of precision, including high order polynomial evaluations, computation of Newton-Raphson iterations, higher order series or rational expansions, etc. In recent years though, several branches of mainstream and scientific computing, such as graphics, machine learning, and numerical simulation, have been drifting in the reverse direction. There are several applications which are either time critical, or process data with a large degree of noise, where such levels of precision are superfluous. This has led to the development of *reduced-precision* data types, and corresponding algorithms, which perform the computations at a lower precision level (e.g. *half-precision* requires 16-bit floats) in faster times.

This should complement vectorised arithmetic quite naturally, as the smaller the data's size, the more can fit inside the vectors, and the greater the parallelisation and speed-up. However, as we will highlight in this thesis, working with reduced-precision is not without its

difficulties and pitfalls, with practical issues such as *vector divergence* becoming problematic, and numerical error being introduced and accumulated at much more significant levels in many numerical algorithms.

1.5 Motivating applications

This section mentions some areas of computing and computational science particularly relevant to high-performance low-precision vectorised arithmetic. We believe these contain either a rich source of interesting problems for the context of vectorised arithmetic, or possibly also for reduced-precision considerations. While this thesis of course tackles some of these areas, this does include topics outside the contents of this thesis.

1.5.1 Intrinsic functions

There are several core intrinsic operations and functions which typically operate at the hardware level. The most fundamental instructions include $\{+, -, \times\}$. At a slightly higher level we have several core functions such as $\div$, $\exp(\cdot)$, $\sin(\cdot)$, $\sqrt{\cdot}$, $\log(\cdot)$, and several others, such as those defined in the ubiquitous standard mathematical library `math.h`.

Given that the hardware can ultimately only add and multiply numbers together, such elementary functions need to be decomposed into addition and multiplication operations. Where machines have only a finite level of precision, many of these functions are approximated, typically by using polynomials or rational approximations, derived from associated Taylor series expansions or Padé approximations. A requested precision level determines both the number of terms to include, and possibly the appropriate expansion. There are even differing optimised functions for different domains, such as those detailed in Intel's maths library `mathimf.h`, e.g. `exp` : $x \to \exp(x)$ and `expm1` : $x \to \exp(x) - 1$, and `log` : $x \to \log(x)$ and `logp1` : $x \to \log(1 + x)$.

Which function approximation is used may be determined at compile time or dynamically at run time, and this can differ for precision levels and input values. As this thesis considers half-precision floating-point inputs, we anticipate that due to the limited range of possible inputs and outputs that half-precision can represent, certain intrinsic functions can be evaluated faster by different routines which are specialised for half-precision. Furthermore, if these are packed into vector hardware, depending on the use case, the optimal routine may be dependent on the vector size and the risk of conditional branching. Overall this presents many situations where a simple function call can be optimally chosen for the use case.

1.5.2 Random number generation

The generation of random numbers is a corner stone of many applications, varying from cryptography to scientific simulations. Abstracting this away from its applications for a moment, this field can be revisited from the two different perspectives of vector hardware and reduced-precision.

Several random number generators work by basic integer and bit manipulations which a scalar processor will work through to produce a stream of random numbers. However, given the availability of vector hardware and the possibly considerable speed improvements this can facilitate, there is a huge attraction to trying to parallelise the generation across the vector lanes. If the routine for generating the random numbers is simple enough, there is scope for removing external function calls and directly hard coding the generation to use the vector units. Depending on the level of resources this consumes this can produce random numbers either for further use on the vector processing units, or for consumption by the scalar processing units.

There has been work focusing on many of these issues for the parallelisation of random number generators for GPUs by Bradley et al. [31], and much of this should carry over naturally to vectorised generation on CPUs.

A second consideration is the cost associated with generating random numbers, or more specifically random bits. There is an associated cost to producing a stream of random bits, measured both in computational time and in the physical energy required for the processing. Foik [70] discuss the physical energy requirements per random bit (with a focus on true random number generators using a physical entropy source). Furthermore, Giles et al. [89, 90] present a theoretical analysis of the performance of random number generators with an associated cost per bit, differentiating between random number generators and random bit generators. Aggregating these two considerations, we see that when using very low precision such as half-precision numbers we may have differing cost metrics for producing random numbers at differing precision levels. Conversely, if very low energy consumption is preferred then the use of reduced-precision variables may be a necessary practical consideration.

1.5.3 Multilevel Monte Carlo

Regular Monte Carlo methods require a large throughput of random numbers. The more random numbers used, the more accurate the answer. Given that Monte Carlo simulations are usually *embarrassingly parallel*, the benefits from the additional parallelisation by effectively using the vectorised hardware are immediately obvious.

An extension of regular Monte Carlo is *multilevel Monte Carlo* [76], which makes the assumption that there are differing levels of accuracy to which the underlying simulation can be approximated. The premise is that there is access to a crude and less accurate simulation which is computationally inexpensive, and that there is a more accurate version which is correspondingly more expensive, (and possibly several intermediate levels). For a given computational budget, the multilevel Monte Carlo framework achieves the highest accuracy by a combination of simulations at the different levels, with very many of the cheap and inaccurate simulations, and relatively few of the costly high accuracy simulations.

In the framework of reduced-precision variables, it is possible to divide the levels by evaluating the simulation at differing precision levels, such as simulations at full double-precision, down to single and half-precision (an idea first proposed by Giles [78]). It can be presumed the lower precision simulations are cheaper to compute. Furthermore, at the reduced-precision, we can consider introducing further divisions by considering different function implementations for the various precision tiers, each possibly cruder and faster than the last. Having such crude and low accuracy simulations is ideal for multilevel Monte Carlo simulations, provided the variance and accuracy of these levels can be understood, and the performance gains are sufficient to outweigh the loss in accuracy.

1.5.4 Machine learning

The increase in machine learning and the demand for computing resources to handle ever larger big-data tasks is unprecedented. This has been one of the primary driving forces behind the development of software and hardware which is capable of half-precision operations. Given the large levels of noise typical to many real-world data sources there is little sense in using needlessly high precision levels. Furthermore, data-sets can become so large that simply processing through them in a reasonable time frame is a challenge. One such example is the development of the *Half* C++ library. As of version 1.12, this stores variables in half-precision, and for performance reasons does calculations in single-precision. The low-precision storage is designed for processing large data-sets typical to machine learning.

Having mentioned the motivation for reduced-precision during the training stage of machine learning, therein lies a second motivation. Typically a long time is spent training a model once, and then the trained model is used multiple times, and frequently the model should also be fast at making predictions. Examples of this include high frequency trading, facial recognition, text processing, speech-to-text dictation, etc. If the application is required to be very fast, then the trained model could also be implemented in reduced-precision (using either floating-point or perhaps fixed-point).

Much of the underlying mathematics in many machine learning techniques centre around common linear algebra calculations. Performing these calculations at higher speeds by using reduced precision is quickly becoming a very active area of research [2, 3, 26, 44, 98, 111–113, 149].

1.6 Literature

Given the interdisciplinary nature of this research there is a wide body of pre-existing literature surrounding several aspects of this project. As such we will introduce supporting literature as we progress through this thesis. Nonetheless, we provide some points of reference surrounding the main topics of this thesis within this section.

Beginning with the high-performance computing aspect of this topic, there has been significant literature by Loshin [147, 148] on efficient high-performance computing. Effective treatment of data alignment and informed compiler use in C [66, 96, 125] are well discussed, and benchmarking methods and hardware capabilities are well documented [65, 169]. With respect to vectorised computing, there is a large pool of research into effective vectorisation using OpenMP [132], and parallel computing more generally [19, 189, 191]. However, while there has been work by Bradley et al. [31] into effective parallelisation of random numbers, this has largely been in the context of GPUs. While much of this can be carried over to vectorised arithmetic on a CPU, much cannot. As such, much work on the transformation of random numbers has been revisited [5, 18, 34, 77, 80, 150, 158, 205, 213]. Furthermore, while we have been exploring the vectorisation of such schemes, adaptations to the upcoming Arm scalable vector extension (SVE) vector length agnostic (VLA) hardware [170, 194] is unexplored, unlike the existing Neon architecture [157] and the recent ThunderX2 [153–156]. Documentation surrounding vectorisation on Intel hardware is widespread [62, 132, 185]. Applications using specialised vectorisation focused compilers (e.g. `ispc` [171]) have been explored by Lee et al. [142] and claims gains close to 85% of code written in intrinsics.

Naturally there is much work from the mathematical community surrounding the convergence of numerical schemes for stochastic differential equations and Monte Carlo methods [107, 108, 117, 123, 133, 172, 173]. Similarly, extensions involving low discrepancy numbers are numerous [45, 54, 126]. However, while floating-point arithmetic is well understood [168], there is a dearth of work surrounding reduced-precision arithmetic within numerical schemes, although an interesting cost model for reduced-bit algorithms is presented by Giles et al. [90].

Inspecting the "Half" library, half-precision is currently largely used for storage reasons, and the use of half-precision in numerical algorithms and mathematical analysis is in relatively early stages. Nonetheless, preliminary studies have been undertaken by Higham and several others [26, 44, 111–113]. This is not too surprising, as the release of half-precision hardware is very recent and still ongoing, and not yet standardised. There has been some previous work by Langou et al. [139] on recovering double-precision accuracy by exploiting the fast performance of single-precision calculations for applications in linear systems, and recently this has been extended to the context of half-precision notably by Higham et al. [111, 113] and Dongarra et al. [2, 3, 98, 149]. Furthermore, work has been conducted by Omland et al. [167], Omland [166], and Brugger et al. [36] on mixed precision multilevel Monte Carlo methods,

although this has focused on non-standard precision levels using *reconfigurable field programmable gate array* (FPGA) hardware.

1.7 Contributions of the thesis

The contributions of this thesis are fourfold. The first contribution is the development and analysis of two approximations to the inverse Gaussian cumulative distribution function which are extremely fast and scale well on modern hardware. These are a piecewise constant approximation and a piecewise linear approximation, where the former uses equally partitioned intervals, and the latter uses geometrically decaying intervals. For these approximations we have been able to bound the error moments between the distribution of exact Gaussian random variables and that arising from using the approximation.

Secondly, we present a modification of the Euler-Maruyama scheme which utilises these approximate distributions to produce associated path simulations. These simulations can be run much faster than those using the exact distributions, and we have been able to bound all the moments of the strong error introduced by our approximations. We have found that the overall error introduced is the product of a temporal discretisation error and the expected error in our function approximation.

Thirdly, we have incorporated our modified path simulations into a nested multilevel Monte Carlo framework. This has allowed us to retain almost all of the speed improvements from our approximations, but through the multilevel Monte Carlo correction, we can retain the original accuracy prior to introducing the approximation. Incorporating our savings into a temporally discretised multilevel Monte Carlo construction, we find that the multilevel correction is a four-way difference between a temporal difference and a difference between exact and approximate random variables. Our analysis has been able to bound this four-way difference rigorously for both the case of expectations of differentiable and non-differentiable Lipschitz functionals. The latter case of non-differentiable Lipschitz functionals presents an appreciably different behaviour unique to the introduction of approximate random variables, where there is a transition between two different convergence rates, dependent on the relative size of the approximation error compared to the discretisation error.

Lastly, we have also added to key stages of our analysis by considering the effects of finite-precision arithmetic and the high degree of rounding error arising from working with low-precision floating-point representations, such as 16-bit half-precision. We have incorporated the frameworks established by Arciniega and Allen [11] and Omland [166] to cover approximate random variables and by modelling the rounding errors in a probabilistic way have produced a comprehensive and rigorous description of the simulation's accrued rounding error. We find this has a root mean squared error proportional to the unit round off and the square root of the number of simulation steps.

In addition to these theoretical contributions, we have empirically demonstrated that the ideas we have developed and provided a thorough mathematical underpinning for, appear applicable in an even wider setting of stochastic simulation. Firstly, approximate random variables appear to be directly transferable to the Milstein scheme. Secondly, the Cox-Ingersoll-Ross process, which is a more difficult stochastic process that those encompassed by our analysis, also appears to benefit from approximate random variables. The Cox-Ingersoll-Ross process requires the non-central χ^2-distribution, which using the same approximations developed for the Gaussian, shows great potential for the successful incorporation of approximate random variables.

1.8 Outline of the thesis

In the remainder of this thesis we will introduce motivating hardware details, and progress this through to the development of nested multilevel Monte Carlo schemes utilising approximate random variables. Here we overview the contents of the remainder of this thesis chapter by chapter.

Chapter 2 details hardware relevant to vectorised arithmetic and high-performance computing more generally. We will discuss modern hardware's capabilities and features which we will seek to exploit in our function approximations. This will include the cache hierarchy, multi-threading, SIMD operations, vector registers, and precision levels.

Chapter 3 gives two constructions of approximations to the inverse Gaussian cumulative distribution function. These are a piecewise constant approximation on equally partitioned intervals, and a piecewise linear function on geometrically decaying intervals. The errors for both these approximations are presented and bounds are produced.

Chapter 4 presents all the numerics associated with implementations of the approximations from Chapter 3. This will include empirical evidence supporting the bounds that were previously produced, alongside compiler reports and high performance implementations for each of the major hardware providers such as Arm and Intel.

Chapter 5 introduces the Euler-Maruyama scheme for producing path simulations for stochastic differential equations. We present the modification of the scheme which incorporates approximate random variables and then bound the strong error, with empirical evidence to support the bounds shown.

Chapter 6 takes our modified Euler-Maruyama scheme and consolidates this into a nested multilevel Monte Carlo framework, ensuring that the error introduced from switching from exact to approximate random variables can be corrected for. This produces bounds for the variance of a four-way difference between temporal differences between fine and coarse paths and differences between exact and approximate random variables. This covers the case of expectations of Lipschitz functionals which are differentiable and non-differentiable, obtaining significantly different behaviour between these cases.

Chapter 7 gives the empirical behaviour for the errors predicted in Chapter 6, and provides an overview detailing the computational savings which could be expected from the introduction of approximate random variables.

Chapter 8 models the behaviour of the cumulative rounding error resulting from having to perform arithmetic in finite-precision and bounds the subsequent error. We bound the error arising in an Euler-Maruyama path simulation and see that the growth in this resultant error is consistent with the central limit theorem and a probabilistic treatment.

Chapter 9 gives the empirical behaviour observed for rounding error and contrasts this to the predictions of Chapter 8. We also present the incorporation of Kahan compensated summation [128] as a means of delaying the onset of significant rounding error to further encourage the use and scope of IEEE 16-bit half-precision.

Chapter 10 extends the ideas we have developed in this thesis, namely approximate random variables and numeric schemes adapted to use these, to distributions other than the Gaussian and schemes other then the Euler-Maruyama method. These include the Milstein scheme, the CIR-process, and the non-central χ^2-distribution.

Chapter 11 presents the conclusions derived from this work, and offers commentary on interesting avenues for further work.

Appendix A presents a compilation of standard results from real analysis and probability theory which we make use of in the thesis.

Chapter 2

Modern hardware and vectorising programs

2.1 Hardware overview

Throughout this thesis we will frequently be exploiting and referring to various features common in modern hardware. We expect the typical reader to likely be familiar with some of these concepts, but probably not all, and so we present an introduction to some of the concepts here. We do not intend for this to be an extensive detailing, and so for more details we would refer the reader to Loshin [147, 148], Hennessy and Patterson [105], and Garside [74].

2.1.1 The cache hierarchy

One of the most crucial features of modern hardware which can easily enhance or cripple a program's performance is the *cache hierarchy* [148, Chapter 2], or more generally called the *memory hierarchy*. The basis for a cache hierarchy stems from the fact that in modern manufacturing, it is relatively cheap to produce storage. A consequence of this is that the cheaper the storage, the more is produced and fitted onto a chip/disk. Unfortunately, the cheaper and larger the storage, the longer it takes to find, read, or write to this storage. Conversely, it is possible to manufacture higher speed storage, but at a greater cost.

Balancing the performance benefits against the financial costs, and reviewing the typical access patterns that computer programs demonstrate, the current industry *de facto* is to incorporate several tiers of these memories onto a chip. This consists of a large amount of cheap and slow storage, and then decreasing sizes of more expensive and power consuming memory structures in the form of caches. For a typical computer processor (ignoring clusters for simplicity) there are typically three cache levels in the cache hierarchy, as shown in Figure 2.1.1. For an m-core processor with cores $\{C_i\}_{i=0}^{m-1}$, each core can be expected to have its own *private* L_1 and L_2 caches, while a larger L_3 cache is *shared* across all the cores. The L_3 cache will be the largest, slowest, and cheapest, while the lower caches (L_1 and L_2) will be faster to access, but at a greater manufacturing cost. Additionally the L_1 cache is also split into two halves, the L_1^{D} and L_1^{I} caches, which hold data and processing instructions respectively.

When a piece of data is read for the first time, it is read in from the *(dynamic) random access memory* (D)RAM. Typically it is expected that nearby data elements will likely be needed shortly thereafter, so these are also read in as one large segment called a *cache line*. A copy of this proliferates down into the lower caches. Then when a new data element is requested, the processor will first query the L_1 cache to see if the data is there. If it is not, then higher caches, and eventually the RAM, will be searched until the data is found. This structure

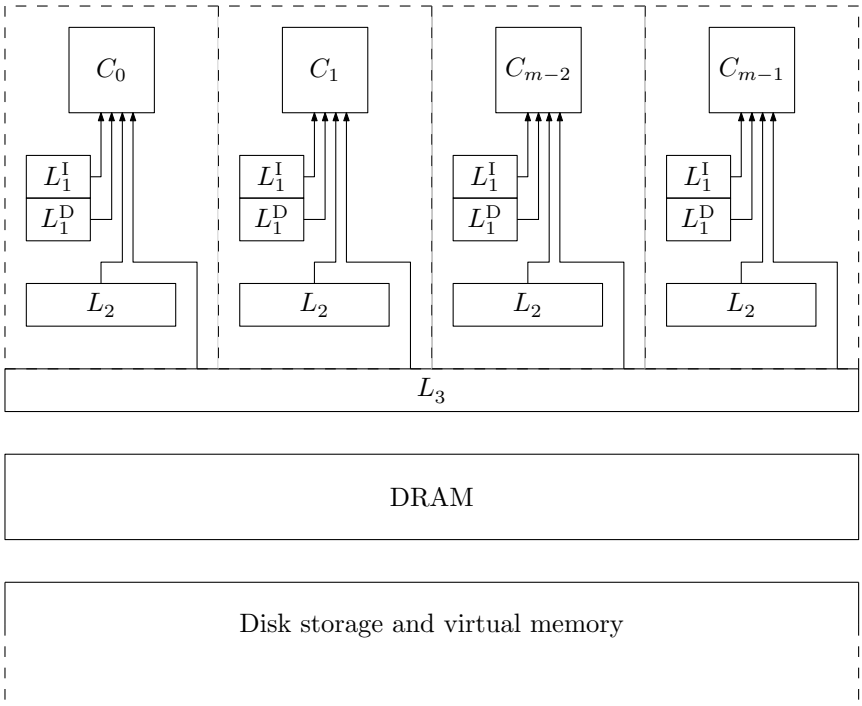


Figure 2.1.1 – The typical cache hierarchy for a modern processors.

promotes the re-use of data, and operations acting on this data ideally need either recently used data (likely still in the cache) or nearby data (hopefully nearby in the cache line).

Ensuring the data is worked on repeatedly while in the cache is known as *temporal locality*. Having good temporal locality in the code promotes good use of the cache and fast access to data, reducing input/output (I/O) bottlenecks. Similarly, ensuring the data is clustered together in memory is known as *spatial locality*. Having good spatial locality increases the likelihood that if several data items are required, they have probably already been brought down into the cache when the cache line was read previously. When a cache is queried for data and it is in the cache, this is known as a *cache hit*. Conversely when the query is unsuccessful, (and subsequent queries in the slower higher caches are required), this is known as a *cache miss* and carries an associated loss in performance resulting from the processor having to wait a longer time to retrieve data. Good locality should reduce the frequency of cache misses.

To give an idea of scale in Figure 2.1.1, for one of our department's Intel Xeon Gold 6140 CPUs, there are 18 cores, and 200 GB of RAM. The L_1^{I}, L_1^{D}, L_2, and L_3 caches are 32 kB, 32 kB, 1024 kB, and 25.4 MB respectively. Finding the exact latencies for these caches for each specific chip can be difficult, so as a proxy for the cache latencies we use the specifications from a previous model, as summarised in Table 2.1.1. Inspecting the scales of these latencies, it is clear that if the operations have an irregular usage or access pattern and do not utilise good spatial or temporal locality, then there will be a huge performance loss from the high latencies required from accessing main memory.

2.1.2 Levels of parallelism

There are various levels of granularity to which a program can be parallelised to improve performance [147, 6.1]. These range from the coarsest level of parallelisation over a network, intermediate levels using multi-threading, to fine grain parallelism involving vectorisation and pipelining.

Memory	Size	Latency (clock cycles)
Registers	$\lesssim 10\,\text{kB}$	≤ 1
L_1	32 kB	3–4
L_2	1024 kB	10–14
L_3	25.4 MB	40–80
DRAM	200 GB	200–300

Table 2.1.1 – Some characteristic specifications about the cache hierarchy. The sizes are for the Intel Xeon Gold 6140, while the latencies are proxy values aggregated from the older Intel Core i7 and Intel Xeon 5500 processors [143, pages 8, 9, 22], and those provided by Drepper [63, pages 16 and 20–23].

2.1.2.1 Parallelism hierarchy

For many modern super-computer systems, and for an increasing number of commercial and domestic computers, there are several opportunities for parallelism. The highest degree of parallelism is to distribute a task across several computer nodes which are connected through a network, and is usually achieved by using the *message passing interface* (MPI) library. This distributes work to several processors.

Most processors now exploit multi-core technology, and will contain several cores. These may have a shared memory structure or a multi-socketed *non-uniform memory access* (NUMA) structure. The work can then be further subdivided across these cores by spawning threads on each of these cores which communicate with each other through a shared memory interface, typically using OpenMP directives.

Lastly, the tasks for a single threaded process can be further parallelised. One method is to overlap instructions in a process known as *pipelining*. A second is to identify instructions which act on several data elements, and if possible bundle these into larger group operations, in a process known as *vectorisation*. These are the finest degrees of parallelism which a programmer can achieve, and achieving this is a task split between the programmer, the compiler, and the hardware. A programmer can usually provide hints to the compiler (possibly very explicitly) that regions of code can be vectorised, whereas whether tasks are effectively pipelined is largely at the mercy of the compiler and ultimately the hardware. This degree of parallelism is known as *instruction level parallelism* (ILP).

2.1.3 Methods for vectorisation

There are three methods to achieve vectorisation in your code, ranging from the easiest to the most difficult. The first (and easiest) is implicit auto-vectorisation by the compiler. The second is by explicitly declaring sections as suitable for vectorisation using the OpenMP `simd` directive. The last (and most difficult) approach is to write code using specialised vector intrinsics or assembly. For those unfamiliar with vectorisation we recommend Sabahi et al. [185], van der Pas et al. [202, Chapter 5], and Douglas and Elliot [62].

2.1.3.1 Auto-vectorisation

This is the easiest method of vectorising code, as it simply leaves the job entirely to the compiler. Most modern compilers are capable of vectorising code segments. The simpler the task, the more likely the compiler will spot if it can be vectorised. Whether auto-vectorisation is enabled is dependent on the default optimisation level of the compiler. For the Intel `icc` compiler

this is enabled by default unde the `-O2` optimisation level, and for the GNU `gcc` compiler under `-O3`.

2.1.3.2 SIMD directives

While the compiler is usually very good at spotting opportunities to vectorise code, if the programmer relies on auto-vectorisation, then they have no control over which parts are vectorised, nor how they are vectorised. The programmer can regain control and assist the compiler by providing hints and flags using `simd` directives. These directives are supported in the OpenMP standard, and also many compilers (such as Intel) provided some extended functionality. The short paper by Klemm et al. [132] showcases that in many situations where auto-vectorisation might fail, using the OpenMP `simd` extension gives the compiler enough information to successfully vectorise, achieving speed improvements of between 2–5 on their hardware.

The advantage of this coding method is that it allows the programmer to navigate issues such as data alignment, control flow divergences, non-trivial memory access patterns, differing data types and lengths, nested loops, etc. Any one of these can easy cause a compiler to fail to vectorise a section of code, or achieve a sub-optimal vectorisation. By allowing the programmer to relay this sort of information to the compiler, and also possibly make suggestions about suitable vectorisation lengths, the compiler is able to produce code which can achieve much higher performances.

2.1.3.3 Vector intrinsics

When writing code which relies on auto-vectorisation or uses OpenMP `simd` directives, the programmer needs to make very few assumptions about the underlying hardware. This leads to increased portability. However, for even higher degrees of performance, there is the option of writing your code using vector intrinsics.

The lighter version of this might be to use a library such as SLEEF (`sleef.h`), which provides SIMD versions of many elementary functions and introduces vector data types. The benefit of this is that it is portable, although it does require possibly quite significant code refactoring. At this level you are beginning to write code functionality explicitly in terms of vectorised operations.

The next level beyond this is to use a vendor specific library of intrinsics, such as Intel's `immintrin.h`. These are very low level functions which are nearly a one-to-one correspondence with Intel assembly instructions. While these can be extremely fast, they cannot be expected to be portable beyond Intel hardware. Similarly, these are sufficiently esoteric to a specific vectorisation hardware (SSE, SSE2, AVX, AVX2, AVX-512, etc.) that they may not scale to possibly different future vector architectures.

Furthermore, these require a large degree of familiarity and are not immediately readable. These are best reserved for people developing high performance mathematical libraries or similarly demanding software. It does not appear too much of a leap to say that programming with these intrinsics is effectively writing decorated assembly, which is best reserved as a last resort for advanced and competent users.

2.2 Existing hardware

Machines have been capable of vectorised arithmetic for a long time, and at least since the 1970's with machines such as TI-ASC and CRAY-1 [147, pages 64–66]. In early generations

there were also notably different hardware architectures for *array processing*, an early form of SIMD parallelisation, as discussed by Garside [74, pages 307–310]. What has changed over the decades has been the demand for vector capabilities, the size of the vectors, and the range of operations which can be vectorised.

In recent years there has been an appreciable plateau in the increase of clock frequencies, and so it has become economically favourable to promote the use of vectorised computing as a means of improving performance. We present two of the main offerings in vector hardware, which are the Intel AVX-512 and Arm SVE architectures. Intel's AVX-512 hardware is widely documented [65], while for more information about Arm's SVE architecture we recommend Petrogalli [170] and Stephens et al. [194].

2.2.1 Intel AVX-512

Intel have offered instructions specialised for vector hardware for many years. Their earliest instruction set specialised for SIMD instructions was the *streaming SIMD extension* (SSE) instruction set, which was 128-bit wide, and could operate on four single-precision 32-bit floating-point numbers. New generations of this were released up to SSE4, and was then redesigned into the *advanced vector extension* (AVX) instruction set. This again evolved and is currently at AVX-512, with vectors 512-bit wide, having 32 vector registers, and 16 masking registers which are 64-bit wide.

These vector registers can be partitioned to hold a small number of large data items (e.g. extended double-precision floating-point numbers), or a large number of smaller items (e.g. short integers). Figure 2.2.1 shows an example of an AVX-512 vector which has been partitioned into n elements, holding the first n element of an array `a`. For Intel the AVX-512 vector registers are named using the `zmm` prefix in Intel assembly. The key feature to note from the Intel hardware is that the vector is specified to be 512-bit wide, and the programmer can be assured of this.

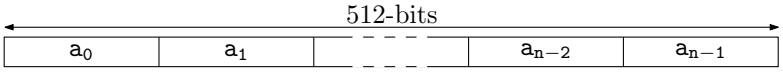


Figure 2.2.1 – A schematic of an AVX-512 vector register holding n elements of an array `a`.

The role of the masking registers (and similarly for predicate registers on Arm hardware) is to create a logical flag for each vector lane. This flag is set as the result of assessing some conditional statement (either explicitly in the code or implicitly by the compiler). Subsequent operations, possibly either arithmetic operations or memory operations, can then be selected/filtered based on the corresponding values within these masking registers. This enables the execution of conditional code on vector hardware, and also protects against vector read and write operations triggering errors which their serial counterparts would not.

2.2.2 Arm SVE

Arm also offer vector capable hardware architectures, and currently this is in their Neon vectors, which are 128-bit wide. However, Arm are soon to release a substantially different offering in vector design with their upcoming *scalable vector extension* (SVE) architecture in the "Post-K" supercomputer "Fugaku" by Fujitsu and RIKEN which will utilise the Fujitsu A64FX Arm-based chip which is SVE capable. Whereas Intel instructions assume a predetermined vector length, the vector instructions in SVE will be *vector length agnostic* (VLA). This means that the vector registers in Arm's SVE offering can be any multiple of 128-bit, and anywhere between 128–2048-bit [170]. Predication registers will also be an equal multiple of 16-bit. Hence the vector registers in Arm's design can vary from very small vectors to very large, the vector

instructions will be indifferent to the length, and the manufacturer is free to choose whichever vector length they think most suitable for their specific product and market.

At the time of writing, hardware is only now becoming available which provides an implementation of the SVE instruction set. At the RIKEN centre for computational science in Japan, the A64FX chips are 512-bit wide. Furthermore, for all SVE hardware, there will be 32 vector registers, 16 predication registers (akin to Intel's masking registers), and a single *first faulting register* (`FFR`). Figure 2.2.2 depicts the various vector and predicate registers offered with Arm's SVE architecture.

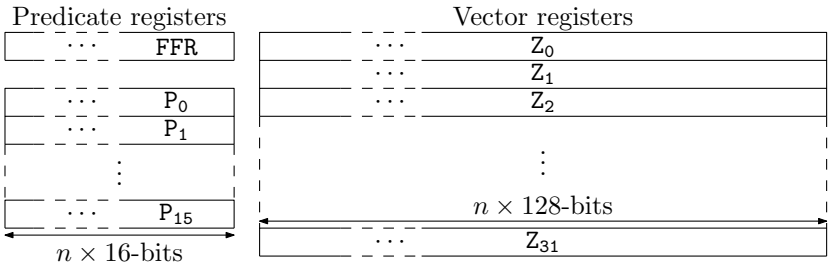


Figure 2.2.2 – A schematic of the hardware offered within the Arm SVE architecture.

2.3 Vectorising programs

Given that vector hardware is almost certainly present on any processor, and that the *vector processing unit* (VPU) can be run concurrently with the *floating-point unit* (FPU) and *arithmetic logical unit* (ALU), then not having vectorised code will be needlessly wasting the hardware's computing capabilities. Furthermore, given the potential speed-up could be several factors (possibly an order of magnitude), code should look to utilise the hardware's vector capabilities for the ultimate performance.

2.3.1 SIMD vectorisation

To appreciate when code can utilise vector instructions, we need to review the differences between *scalar* and *vector processing*. Classically, when writing several tasks for a computer to perform, we may naïvely expect that our tasks are translated into a sequence of hardware instructions, which the hardware will then execute one after the other in a well ordered sequence. By extension, if we have several data to process, then once an operation is done on one of the data, it is repeated on the next, and when this completes the processor operates on the next, and so on. This sequential processing of instructions on a set of data is known as *scalar processing*.

Frequently though we find that we have some unchanging set of instructions which we would like to repeatedly use to process several data in an identical fashion. If this is the case then we can bundle these into *single instruction multiple data* (SIMD) operations, and it is these SIMD operations which naturally extend onto the vector hardware.

Typically the most frequent scenario when a programmer will encounter these repeating instructions which exhibit SIMD operations are in `for` loops (or similarly `do` loops in languages such as Fortran). Code 2.3.1 demonstrates a simple `for` loop acting on an array `a`. The array `a` is assumed to have 1024 or more elements. In this case we have a simple operation which acts in an identical manner to each element of the array, which is an example of a single instruction multiple data (SIMD) operation. Hence this operation is suitable for vectorisation.

To begin appreciating the benefits of vectorisation, consider the slightly more involved operation representing $a + b = c$ with arrays `a`, `b`, and `c` as depicted by the more

```
1 for (unsigned int i = 0; i < 1024; i++)
2 {
3     a[i] *= 2;
4 }
```

Code 2.3.1 – A simple `for` loop in C which doubles an array `a`. This operation naturally presents itself for SIMD vectorisation.

general C function in Code 2.3.2. Here `a`, `b`, and `c` are all *pointers* to integers. Because pointers in C can overlap in memory, we use the `restrict` keyword to indicate that these map to distinct and non-overlapping memory locations. There are a few notable operations happening here. Firstly, `a` and `b` need to be loaded in from memory. Next, each of the elements need to be added together. Lastly, once the results have been computed, these need to be moved out of the registers and stored into `c`.

```
1  void add_arrays(int * restrict a,
2                  int * restrict b,
3                  int * restrict c,
4                  unsigned int n)
5  {
6      for (unsigned int i = 0; i < n; i++)
7      {
8          c[i] = a[i] + b[i];
9      }
10 }
```

Code 2.3.2 – A function which writes the sum of `a` and `b` into an array `c`.

In Figure 2.3.1 we depict how Code 2.3.2 can be processed in either a scalar fashion or a vectorised fashion. In the scalar processing the operations (`+`, `=`) act on each element one by one. If we suppose we have a vector width that holds a modest 4 integers, then we can replicate the functionality using vectorised SIMD operations. On modern hardware, vector operations can be expected to be almost equal in cost to their scalar counter parts. So considering our example, we see that there are only a quarter of the original number of scalar operations, so such a simple task could be expected to be completed in a quarter of the time that the scalar processing would take.

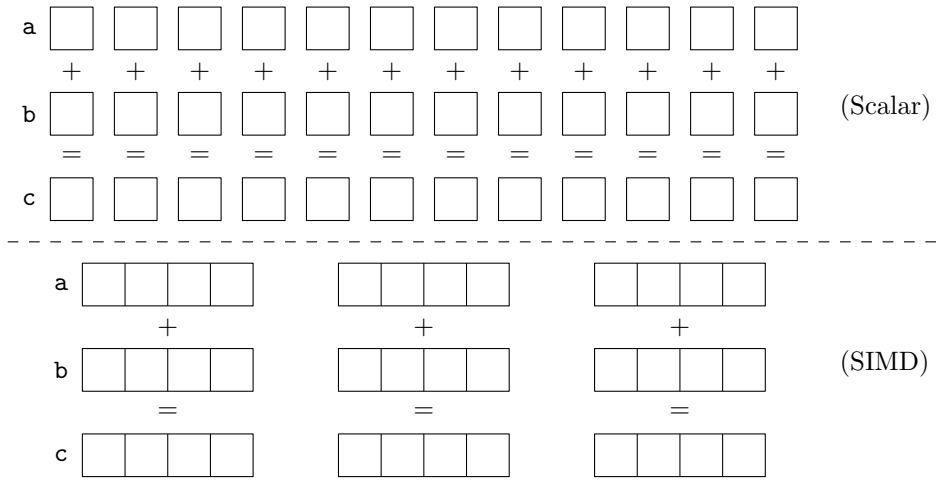


Figure 2.3.1 – The processing of Code 2.3.2 with a vector width holding a modest 4 integers. **(Above)** Scalar. **(Below)** SIMD.

2.3.2 Inhibiting vectorisation

On the surface it would appear that once we identify segments of SIMD operations in our program, then it should be vectorisable, and we might want to request the compiler produces associated vector instructions. However, there are numerous pitfalls which can hinder or prevent the compiler, and here we mention a few common pitfalls:

Implied dependence In C/C++ pointers can alias the same memory block. This means that if memory must be read and written to, then the compiler cannot guarantee that the write from one vector operation will not erroneously interfere with subsequent operations. This usually requires declaring the memory structure more explicitly, such as by using the `restrict` keyword.

Unknown iteration counts Ideally the compiler should know how many iterations are required. For OpenMP this means that the number of loop iterations must be known when the loop is encountered. This excludes `while` loops, and usually requires that `for` loop counts be known at compile time.

Nested `for` loops If there are several nested `for` loops, it is not clear if the outer or inner loop should be vectorised, or if they should be collapsed. By default the compiler will only try to auto-vectorise the innermost `for` loops.

Loop dependencies If the output from one iteration of the loop depends on the result from a previous iterate in the loop. This typically occurs in read-after-write, such as when the result of operating on one element updates its neighbours (typical in some finite-difference procedures and matrix factorisation methods).

Indirect memory access If the index of a write location is the result of an intermediate operation, such as an intermediate array look up (e.g. `c[d[i]]`), then the compiler cannot determine if two writes might coincide.

Conditional operations If there is any conditionality in how to work on the data between differing elements in the vector, then this begins to violate the single instruction multiple data pre-requisite.

2.3.2.1 Conditional branching

One of the most frequent and tricky obstacles to vectorising code can be conditional operations within a `for` loop, which can have implicit or explicit conditionality. Code 2.3.3 gives an example of a rather explicit conditional operation. In this simple example we either copy `b[i]` to `c[i]` or compute the negative square of `b[i]` and write that to `c[i]`. The operations in Code 2.3.3 are moderately arbitrary, but more generally we would expect to perform either some computation A or B depending on the result on the condition.

```
1  for (unsigned int i = 0; i < 1024; i++)
2  {
3      if (a[i] > 0)
4      {
5          c[i] = b[i];                /* Computation A. */
6      }
7      else
8      {
9          c[i] = - b[i] * b[i];       /* Computation B. */
10     }
11 }
```

Code 2.3.3 – Conditional branching within a `for` loop. This conditionality can be subtly compacted in ternary operations.

A compiler could fail to vectorise conditional code execution because of any combination of the following:

- The compiler could not create a logical mask/predicate for the operation.
- The two branches have an interdependency.
- One or more branches leave the `for` loop (e.g. `break` conditions).
- A branch calls an external function (e.g. linked library functions).

We mentioned that the compiler will try to create a logical mask to handle the operation. We should highlight at this point though a key difference between how scalar code processes a conditional computation, and the SIMD framework for conditional processing. For simplicity suppose we only have a single condition C which is either true or false, is evaluated at run-time, and has conditional computations A and B corresponding to the true and false outcomes respectively. The scalar handling would be to evaluate the condition, and then perform either A or B, but not both, and then possibly write the result somewhere. The SIMD framework though cannot perform A for some of the vector lanes and B for others both at the same time. The resolution adopted then is to compute both A and B for all the vector elements, and then compute the results of the conditional statement for each entry, and then write the results accordingly. The key difference here is that while the scalar processing computed A **or** B (but not both), the vector processing computed both A **and** B. This different processing is depicted in Figure 2.3.2. The programmer should not assume the compiler or hardware will evaluate the condition first and then handle the computations dependent on the results.

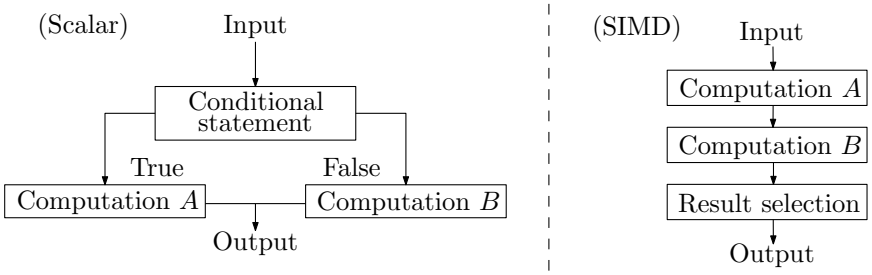


Figure 2.3.2 – **(Left)** Scalar handling of conditional branching. **(Right)** SIMD conditional branching.

The fact that different lanes may need to perform differing tasks is *divergent* behaviour (on Nvidia GPU's this is akin to *warp divergence*). There are several pitfalls surrounding this. Suppose the hardware partitioning fits n elements into the vector width, (where the case $n = 1$ corresponds to scalar processing). Consider a single vector instruction where without loss of generality suppose more of the entries require computation A. In the extreme case where all elements only require A, then we may be computing B needlessly, wasting vector registers, and delaying proceeding with the program. Furthermore, if we have several conditions and branches, then it may be quicker to process the array in a scalar fashion than in a vector one.

A particularly common scenario is when one branch B is less frequently required, but is typically more computationally expensive. Suppose branches A and B are expected to take k_A and k_B clock cycles respectively, and evaluating the condition C requires k_C. Let computation B occur with probability $p_B \leq 0.5$, and the likelihood of one vector element requiring B be independent of the other vector elements. The probability then that the array requires a mix of computations A and B is then $1 - (1 - p_B)^n - p_B^n$. If N elements need processing, where N is some large integer multiple of n, then we can define the speed-up efficiency η as

$$\eta = \frac{\text{scalar processing time}}{\text{vector processing time}}. \tag{2.3.1}$$

Accounting for the computation of both branches we can calculate the number of vector operations N_A which only required branch A to be computed, similarly N_B for only B, and

$N_{A\cap B}$ for both. We make the idealisation that if all the vector lanes require the same computation, then only that computation is computed, and not both.

$$\eta = \frac{N(k_A(1-p_B)+k_Bp_B+k_C)}{k_AN_A+k_BN_B+(k_A+k_B)N_{A\cap B}+\frac{N}{n}k_C} \tag{2.3.2}$$

$$= \frac{n(k_A(1-p_B)+k_Bp_B+k_C)}{k_A(1-p_B)^n+k_Bp_B^n+(k_A+k_B)(1-(1-p_B)^n-p_B^n)+k_C} \tag{2.3.3}$$

We can evaluate (2.3.3) for several toy values, and an example is provided in Table 2.3.1.

p_B	n						
	2	4	8	16	32	64	128
2^{-1}	1.34	2.15	4.05	8.08	16.2	32.3	64.6
2^{-2}	1.2	1.6	2.48	4.53	8.98	18.0	35.9
2^{-3}	1.22	1.51	2.0	3.04	5.47	10.8	21.6
2^{-4}	1.32	1.66	2.06	2.71	4.09	7.32	14.4
2^{-5}	1.48	2.0	2.52	3.13	4.11	6.18	11.0
2^{-6}	1.65	2.47	3.36	4.24	5.25	6.89	10.3
2^{-7}	1.78	2.97	4.46	6.06	7.65	9.49	12.4
2^{-8}	1.88	3.37	5.61	8.45	11.5	14.5	18.0
2^{-9}	1.94	3.65	6.55	10.9	16.4	22.3	28.1
2^{-10}	1.97	3.81	7.19	12.9	21.5	32.3	43.9

Table 2.3.1 – The vectorisation speed-up efficiency η from (2.3.3) for differing vector sizes n and divergence probabilities p_B. Values are generated for $k_A = 5$, $k_B = 100$, and $k_C = 1$.

For the example speed-up efficiencies in Table 2.3.1 there are several regions of interest. Let us begin by inspecting the first row where $p_B = 0.5$. As the number of elements which fit inside a vector increases we notice that the speed-ups increase nearly linearly with the vector length. Given such a high divergence probability, as the vectors become ever larger the likelihood of divergence in a vector becomes a certainty. This means that while the scalar code would be dominated by the expensive calculation of B, the vector code must perform both A and B for each vector, which is only slightly more than B with our example values $k_A = 5$, $k_B = 100$, and $k_C = 1$. However, because these expensive B computations are nearly evenly spaced throughout we can only expect our vector to ever be approximately half filled with these expensive computations. Hence if we approximate B as the dominant computation, then our vector is only half filled with these values, and hence the speed-up we might expect from this vectorisation almost follows *Amdahl's law*, but with the speed-up coefficient being just over half the number of vector elements.

Conversely, we can consider the other extreme where the probability of divergence is very small, such as in the bottom row with $p_B = 2^{-10}$. Consider the first three vector lengths which can hold 2, 4, and 8 elements respectively, and notice the associated speed-ups of 1.97, 3.81, and 7.19, which are approximately equal to the vector lengths. For such small vectors there is a negligible chance of divergence, and so these coefficients exactly match the speed-up associated with Amdahl's law.

An informative intermediate regime is when there becomes an appreciable chance of having divergent behaviour only in the larger vectors. Consider for example the row corresponding to $p_B = 2^{-6}$. For a vector width where $n = 64$ then we expect there to be one B calculation required, and there is a 63.5% chance a vector requires a mix of computations A and B (this increases to 86.7% for $n = 128$). The key feature to notice here is that while the bulk of the calculations are the relatively inexpensive A computation, as the vector widths

double, the speed-ups do not double. Notice that by the time we could fit a massive 128 elements in our vector, the speed-up is only a factor of 10.3. We can see that by the time we are reaching the very wide and finely partitioned vectors the performance gains are sub-linear. This is because as the number of elements increases the probability of at least a single expensive calculation becomes appreciable, and this contaminates the rest of the vector. To further appreciate the subtlety of this point we recommend considering the minimum in performance down the $n = 128$ column.

It is quickly worth commenting on the number of elements we chose to present in Table 2.3.1, and whether such large values of n can be expected. For Intel's AVX-512 there are other supplementary modules which can extend the core/fundamental functionality, and one such module which is available in Skylake is the *byte and word support* module AVX-512 BW which offers arithmetic operations and masking capabilities for 512-bit vectors containing down to 8-bit or 16-bit integer elements, allowing operations on vectors containing up to 64 elements [65, 3.4]. Furthermore, when we consider the very large 2048-bit vectors the Arm SVE architecture could produce, then if we partitioned this into 16-bit data sizes, then these vectors could fit 128 elements. Stephens et al. [194, page 27] state that each predicate comprises 8-bit per 64-bit vector element, which can allow predication down to the granularity of a single byte, and that the smallest data element can be a single byte in size. With these designs it is important then to keep in mind that there is the scope for highly partitioned vectors which become increasingly susceptible to vector lane divergence.

The lesson to be learnt from this worked through enumeration of possible speed-up efficiencies is that conditional branching causes the actual speed-up realised from vectorising code to be only a marginal amount of that anticipated by Amdahl's law. We will see later in this thesis that producing approximations for the inverse Gaussian cumulative distribution function which circumvent or avoid conditional branching are crucial to achieving random numbers faster than those available from commercial libraries. This scenario of encountering conditional branching with one expensive but unlikely case, and one cheap and likely case, is exactly the problem we will encounter later.

Chapter 3

Constructing approximate Gaussian random variables

3.1 The inverse transform method

A key stage in a Monte Carlo implementation is transforming a uniform random variable into a random variable which follows some specified distribution. One such method is the *inverse transform method* [92, 2.2.1]. For a *probability density function* (PDF) $f(\cdot)$ which we want to sample from, then given its *cumulative distribution function* (CDF) $F(\cdot)$ we can form its *inverse cumulative distribution function* $F^{-1}(\cdot)$. If we then apply the inverse cumulative distribution function to uniformly drawn random variables $u \sim \mathcal{U}([0,1])$, then the distribution of the random variables $F^{-1}(u)$ is the desired distribution. A numerical demonstration of this is shown in Figure 3.1.1.

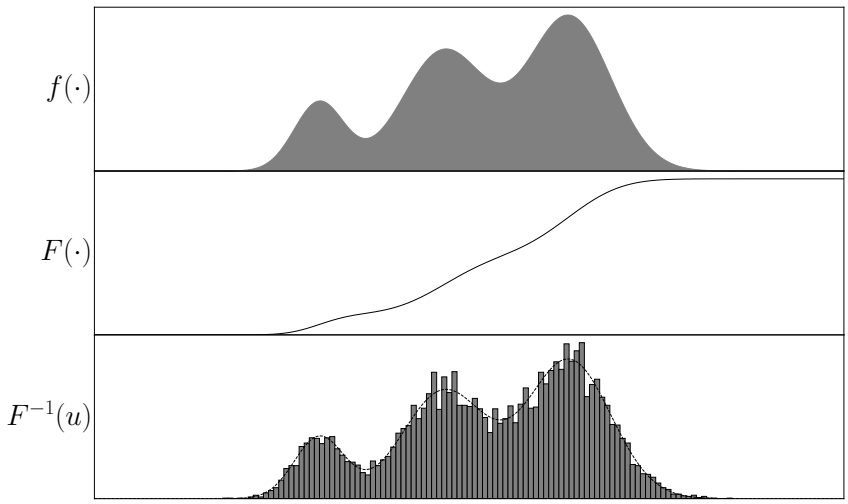


Figure 3.1.1 – A numerical demonstration of the convergence in distribution of the inverse transform method.

There are several benefits of using such an approach, but one of the most considerable is that this method ensures only one uniform random variable is required to produce a Gaussian random variable. Furthermore, no samples are rejected, which is essential for quasi-Monte Carlo, as rejecting samples would remove the low discrepancy property of the sequence of quasi-random numbers, and so the inverse transform method is appropriate in this setting. Lastly, the inverse transform method can be applied to a wide range of possible distributions, and hence we hope and anticipate that any insight gained in our analysis, constructions, or implementations might be applicable to distributions beyond the Gaussian distribution.

3.1.1 The inverse Gaussian cumulative distribution function

Countless applications (especially in finance) require Gaussian random variables in their Monte Carlo simulations, and so it is particularly important to consider the inverse transform method for the Gaussian distribution. Hence we must evaluate the inverse Gaussian cumulative distribution function $\Phi^{-1}(\cdot)$. (Outside of Monte Carlo applications, evaluating the probability of rare events involving $\Phi^{-1}(\cdot)$ is common in statistics, such as when computing p-values in *hypothesis testing*). A plot of $\Phi^{-1}(\cdot)$ is presented in Figure 3.1.2, and the corresponding Gaussian distribution's probability density function $\phi(\cdot)$, where $\phi(x) \coloneqq \frac{1}{\sqrt{2\pi}}\exp(-\frac{x^2}{2})$ for $x \in \mathbb{R}$, is shown in Figure 3.1.3.

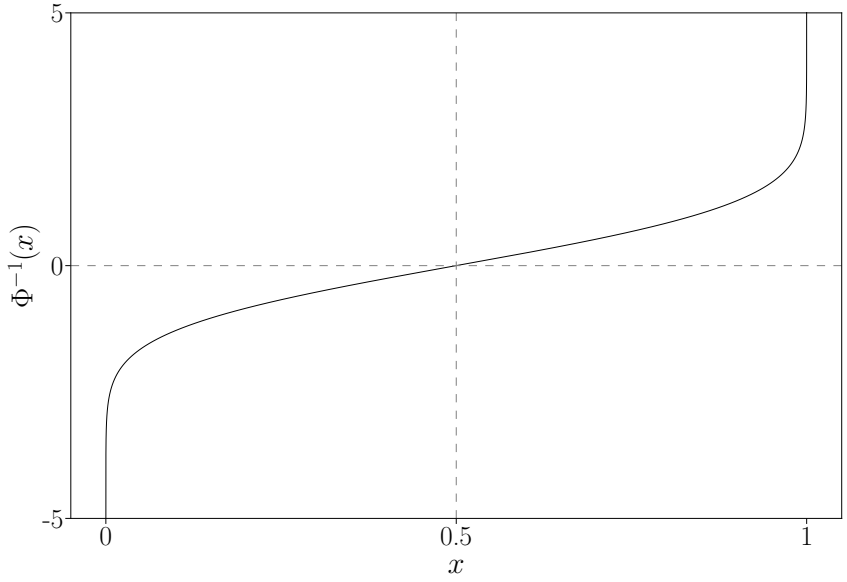


Figure 3.1.2 – The inverse Gaussian CDF function. The function is rotationally symmetric about $x = 0.5$. The range $[-5, 5]$ approximately corresponds to the domain $[10^{-7}, 1 - 10^{-7}]$.

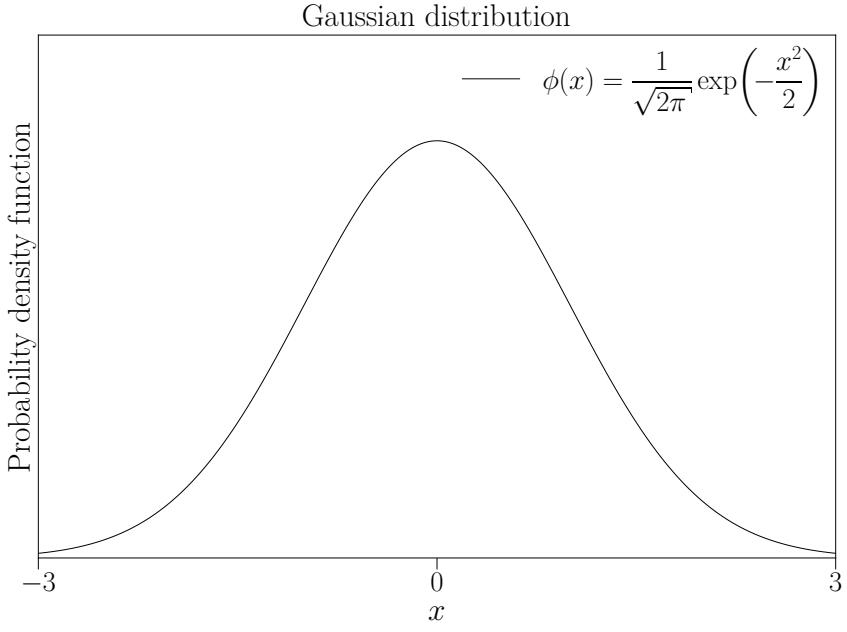


Figure 3.1.3 – The probability density function of the Gaussian distribution.

3.1.1.1 Approximate random variables

Our goal is to produce approximations $\tilde{\Phi}^{-1}(\cdot)$ which approximate the inverse Gaussian CDF $\Phi^{-1}(\cdot)$ such that $\tilde{\Phi}^{-1} \approx \Phi^{-1}$. Using the inverse transform method on a uniform random variable u with $\Phi^{-1}(\cdot)$ produces a Gaussian random variable Z and with $\tilde{\Phi}^{-1}(\cdot)$ produces a random variable $\tilde{Z}$. This means that the distribution of random variables $\tilde{Z}$ we will produce from our approximation $\tilde{\Phi}^{-1}(\cdot)$ will come from a distribution which is approximately Gaussian in some sense. As such we introduce the term *approximate random variables* to describe random variables which come from an approximation of a distribution's inverse CDF function, in contrast to *regular or exact random variables* which come from the exact distributions.

Definition 3.1.1 (approximate random variables). For a distribution $\mathcal{D}$ with the inverse cumulative distribution function D^{-1}, then the random variables produced by the inverse transform method using some approximation $\tilde{D}^{-1} \approx D^{-1}$ are called *approximate random variables* which are said to approximately follow the distribution $\mathcal{D}$.

Related work by Xu and Berger [217] investigates the optimal approximations of measures through a finite number of atoms, where either the locations or the weights of the atoms can be pre-specified. As we will see, this is related to the work presented here and that also presented by Giles et al. [90].

3.2 Equipartitioned piecewise constant approximations

Inspecting $\Phi^{-1}(\cdot)$ in Figure 3.1.2, a mathematical approximation which immediately suggests itself is a piecewise constant approximation. Furthermore, the simplest case of this is to have the intervals equipartitioned. Knowing that our ultimate ambition will be to construct approximations which are computationally very fast, rather than being discouraged by what could be considered a rudimentary, banausic, or crude approximation, we are actually encouraged by such a simple construction, hoping that mathematical simplicity will correspond to computational speed.

Before constructing a piecewise constant approximation over equipartitioned intervals, we will need to specify how we choose the constant for each interval. A few natural candidates which present themselves are the expected value within the interval, the mid-point, or the inner boundary point (choosing the inner boundaries to avoid the singularities at the outer boundaries). The following results show that random variables using these constructions have uniformly bounded moments, which will be useful in later analysis.

Lemma 3.2.1. *[27, C.2] For a standard Gaussian random variable* $Z \sim \mathcal{N}(0,1)$*, all its moments are bounded such that* $\mathbb{E}(|Z|^p) < \infty$ *for any* $1 \leq p < \infty$*.*

Lemma 3.2.2. *Using an equipartitioned discretisation of* $\Phi^{-1}(\cdot)$ *into* $2K$ *intervals* $\{I_k\}_{k=1}^{2K}$*, with the* k*-th interval denoted by* I_k*, where* $1 \leq K < \infty$*, then if the approximate Gaussians* $\tilde{Z}$ *are constructed as*

$$\tilde{Z}_k \coloneqq \mathbb{E}(Z \,|\, \Phi(Z) \in I_k), \tag{3.2.1}$$

where $\Phi(\cdot)$ *is the Gaussian cumulative distribution function, then all finite moments of* $|\tilde{Z}|$ *are uniformly bounded with respect to* K *such that for any* $1 \leq p < \infty$ *we have* $\sup_K \mathbb{E}(|\tilde{Z}|^p) < \infty$*.*

Proof. As our approximation will be rotationally symmetric, we only need to consider the approximation of $\Phi^{-1}(\cdot)$ by K intervals in the domain $[0.5, 1]$, where we know both Z and $\tilde{Z}$ are always positive. Considering then the k-th interval in this domain, we can apply Jensen's inequality for expectations (Lemma A.2.2) for the p-th power for some finite integer $1 \leq p < \infty$

$$(\mathbb{E}(Z \,|\, \Phi(Z) \in I_k))^p \leq \mathbb{E}(Z^p \,|\, \Phi(Z) \in I_k) \tag{3.2.2}$$

using the definition of $\tilde{Z}_k$

$$\tilde{Z}_k^p \leq \mathbb{E}(Z^p \,|\, \Phi(Z) \in I_k) \tag{3.2.3}$$

averaging over the K intervals (of which there are only finitely many)

$$\mathbb{E}_K\Big(\tilde{Z}_k^p\Big) \leq \mathbb{E}_K(\mathbb{E}(Z^p \,|\, \Phi(Z) \in I_k)) \tag{3.2.4}$$

gives

$$0 < \mathbb{E}\Big(\tilde{Z}^p\Big) \leq \mathbb{E}(Z^p) < \infty, \tag{3.2.5}$$

where in the final inequality we used Lemma 3.2.1. **QED**

Corollary 3.2.2.1. *If instead the approximate Gaussian random variables are constructed as either the central values*

$$\tilde{Z}_k^{\mathrm{C}} \coloneqq \Phi^{-1}\bigg(\frac{\max I_k + \min I_k}{2}\bigg) \tag{3.2.6}$$

or the inner values

$$\tilde{Z}_k^{\mathrm{I}} \coloneqq \begin{cases} \Phi^{-1}(\min I_k) & I_k \geq 0.5 \\ \Phi^{-1}(\max I_k) & I_k < 0.5, \end{cases} \tag{3.2.7}$$

then these also have all their moments uniformly bounded.

Proof. As $\Phi^{-1}(\cdot)$ is a convex function in the domain $[0.5, 1)$, then the constructed approximate Gaussians $\tilde{Z}_k$ from (3.2.1) is an upper bound on $\tilde{Z}_k^{\mathrm{C}}$ by the Hermite-Hadamard inequality (Lemma A.1.1) and subsequently also $\tilde{Z}_k^{\mathrm{I}}$, and as Lemma 3.2.2 shows $\tilde{Z}_k$ has finite moments, then so too must the other two constructions. **QED**

For the remainder of this work, unless otherwise stated, if an equipartitioned piecewise constant approximation of the Gaussian distribution is used, we will assume the constant's value is the expectation within the interval as in (3.2.1).

An example equipartitioned piecewise constant approximation using 16 intervals is shown in Figure 3.2.1. The corresponding probability density function which results from this is shown in Figure 3.2.2, and is a collection of Dirac-δ functions.

As this approximation takes a distribution with a continuous output and approximates this by a distribution with only a finite set of discrete outputs, we say that we have *quantised* the distribution. We will henceforth often refer to the output of this approximation as quantised, and that this produces quantised random variables. Furthermore, schemes which employ these quantised random variables may be prefixed as quantised too, to indicate the use of this approximation scheme.

The work by Giles et al. [90] considers a Gaussian random variable arising from a uniform random variable $U \sim \mathcal{U}([0,1))$ which has been truncated to q-bits [90, Theorem 1, Section 2.2], and hence corresponds to the approximation by a piecewise constant approximation with 2^q equipartitioned intervals. They too showed that all the moments of the approximate Gaussians were uniformly bounded and the mean of the approximate Gaussians was zero. Furthermore, for an approximate Gaussian with only q-bits, which we denote as $\tilde{Z}^{(q)}$ where

$$\tilde{Z}^{(q)} \coloneqq \Phi^{-1}\bigg(\frac{\lfloor 2^q U \rfloor}{2^q} + 2^{-(q+1)}\bigg), \tag{3.2.8}$$

they showed

$$\mathbb{E}\Big(\Big|Z - \tilde{Z}^{(q)}\Big|^2\Big)^{\frac{1}{2}} \preceq 2^{-\frac{q}{2}} q^{-\frac{1}{2}}, \tag{3.2.9}$$

where $f \preceq g$ denotes that there exists a strictly positive finite constant c such that $f \leq cg$.

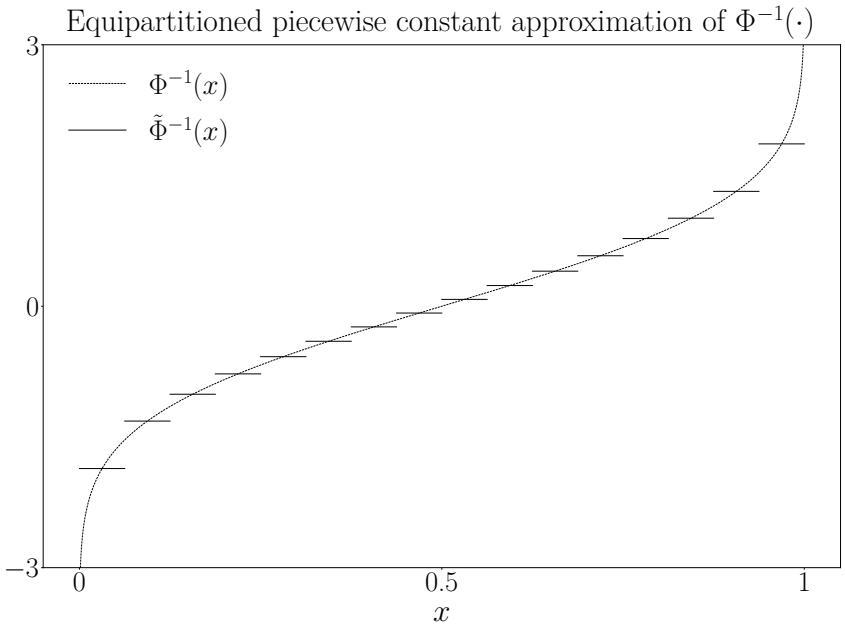


Figure 3.2.1 – An example equipartitioned piecewise constant approximation to $\Phi^{-1}(\cdot)$ using 16 intervals.

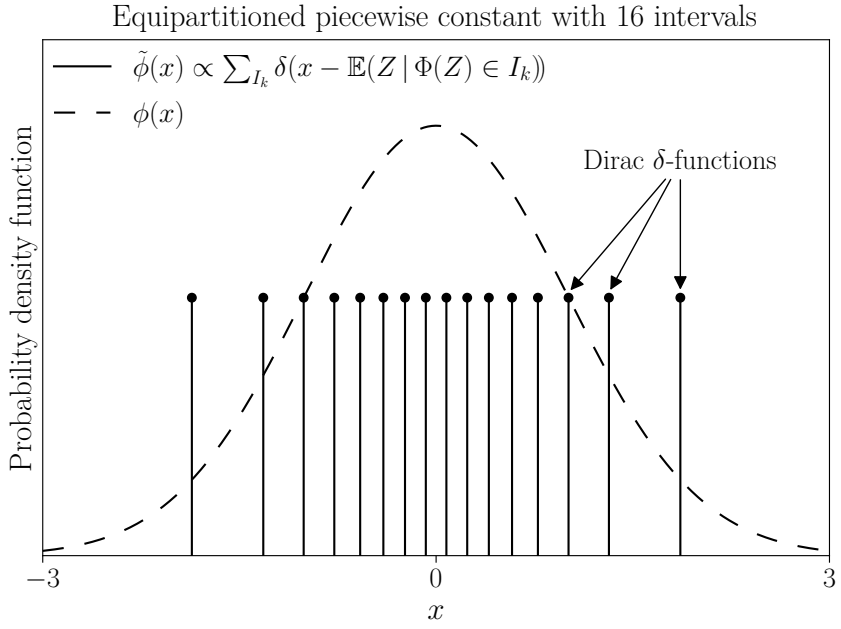


Figure 3.2.2 – An example probability density function resulting from an equipartitioned piecewise constant approximation. This example is produced from the approximation shown in Figure 3.2.1.

We can compare the bound from (3.2.9) from Giles et al. [90] to the actual decay in the L^2-error, which is more commonly known as the *root mean squared error* (RMSE), which is shown in Figure 3.2.3. We can see from this that the bound obtained by Giles et al. [90] bounds the error well, but that a simple power law model produced from the regression fit is also reasonable[1].

Taking the insight from Giles et al. [90] that these quantised distributions can be interpreted as coming from truncated uniforms, we denote the available number of bits as the *quantisation*, and use this to parametrise the fidelity of the approximation.

[1]This may be desirable for simplified modelling purposes in any subsequent analytic calculations or heuristics.

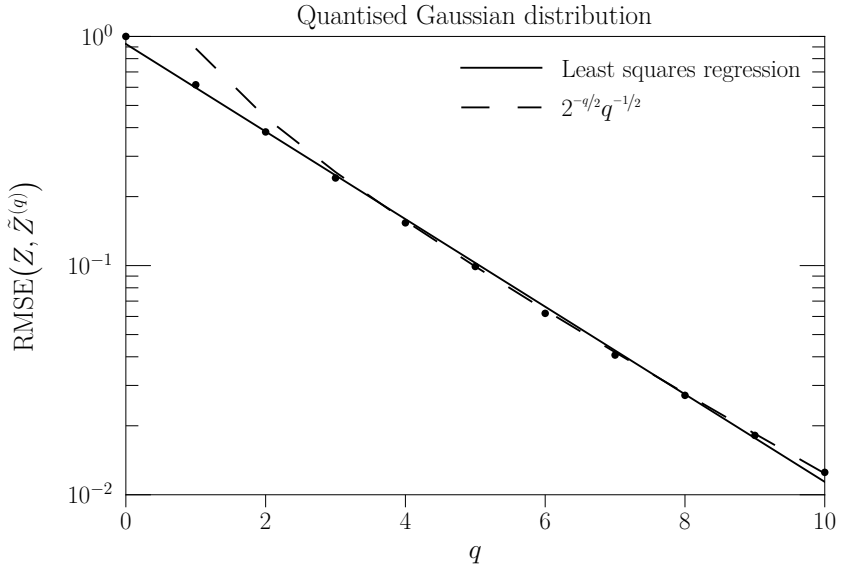


Figure 3.2.3 – The RMSE between the exact and approximate Gaussians for 2^q intervals. We show the bound from Giles et al. [90] and also the empirical least squares regression fit $\mathrm{RMSE}(Z, \tilde{Z}^{(q)}) \approx \exp(-0.075 - 0.44q) \equiv 0.93 \times 2^{-0.63q}$.

3.2.0.1 High-order moments of quantised Gaussian random variables

At this point it is useful to extend (3.2.9) (from Giles et al. [90]) to higher order moments, as this will be useful later when we consider the L^p-convergence (in Section 5.2.1 and again in Section 6.2.1). We use the convention that $f(z) \approx g(z)$ denotes $\lim_{z\to\infty} \frac{f(z)}{g(z)} = 1$. The main result of this section will be Lemma 3.2.4, and the preceding lemmas and corollaries are extensions of those developed by Giles et al. [90].

Lemma 3.2.3. *Defining* $z_q \coloneqq \Phi^{-1}(1 - 2^{-q})$*, then for* $q \gg 1$ *we have* $2^{-q} \approx \frac{\phi(z_q)}{z_q}$*, and hence* $z_q \lessapprox \sqrt{q \log(4)}$.

Corollary 3.2.3.1. *To second order in* z_q *we have* $2^{-q} \approx \phi(z_q)(\frac{1}{z_q} - \frac{1}{z_q^3})$.

Proof. We trivially have that $1 - \Phi(z_q) = 2^{-q}$, and so it remains to evaluate the integral $1 - \Phi(z_q) = \int_{z_q}^\infty \phi(s)\,\mathrm{d}s$. We can do this using integration by parts using $\frac{\mathrm{d}}{\mathrm{d}z}\phi(z) = -z\phi(z)$, giving $\int_{z_q}^\infty \phi(s)\,\mathrm{d}s = [\frac{-\phi(s)}{s}|_{z_q}^\infty - \int_{z_q}^\infty \frac{1}{s^2}\phi(s)\,\mathrm{d}s$. Repeating this several times gives $\int_{z_q}^\infty \phi(s)\,\mathrm{d}s = \phi(z_q)(\frac{1}{z_q} - \frac{1}{z_q^3} + \frac{3}{z_q^5} - \cdots) \lessapprox \frac{\phi(z_q)}{z_q}$, and hence to first order we have $2^{-q} \approx \frac{\phi(z_q)}{z_q}$ and to second order $2^{-q} \approx \phi(z_q)(\frac{1}{z_q} - \frac{1}{z_q^3})$. Now substituting in $\phi(z) \coloneqq \frac{1}{\sqrt{2\pi}}\exp(-\frac{z^2}{2})$ into $2^{-q} \lessapprox \frac{\phi(z_q)}{z_q}$, this re-arranges to $z_q \approx \sqrt{q\log(4) - 2\log(z_q) - \log(2\pi)}$ which in the limit of $q \gg 1$ approximates to $z_q \lessapprox \sqrt{q \log(4)}$. **QED**

Corollary 3.2.3.2. $\sqrt{q \log(4) - \log(q\pi \log(16))} \lessapprox z_q \lessapprox \sqrt{q \log(4)}$ *for* $q \gg 1$ *and* $1 \lessapprox \frac{2^q \phi(z_q)}{z_q} \lessapprox (1 - z_q^{-2})^{-1}$.

Proof. The approximation $\sqrt{q \log(4) - \log(q\pi \log(16))}$ is obtained as the first correction from the iterative method approximation for z_q, and hence acts as a lower bound. For the second bound we recall $2^{-q} = \phi(z_q)(\frac{1}{z_q} - \frac{1}{z_q^3} + \frac{3}{z_q^5} - \cdots)$ which if we bound using the second term we obtain $1 \gtrapprox \frac{2^q\phi(z_q)}{z_q}(1 - z_q^{-2})$ from which the desired upper bound follows. **QED**

It is possible to numerically evaluate these integrals to see how good these approximations are and how large q needs to be before their quality of approximation is

reasonable. Results from numerical evaluations are shown in Figures 3.2.4 and 3.2.5, from which we can see these approximations are of a reasonable quality for even quite modest values, down to quantisations $q \approx 10$.

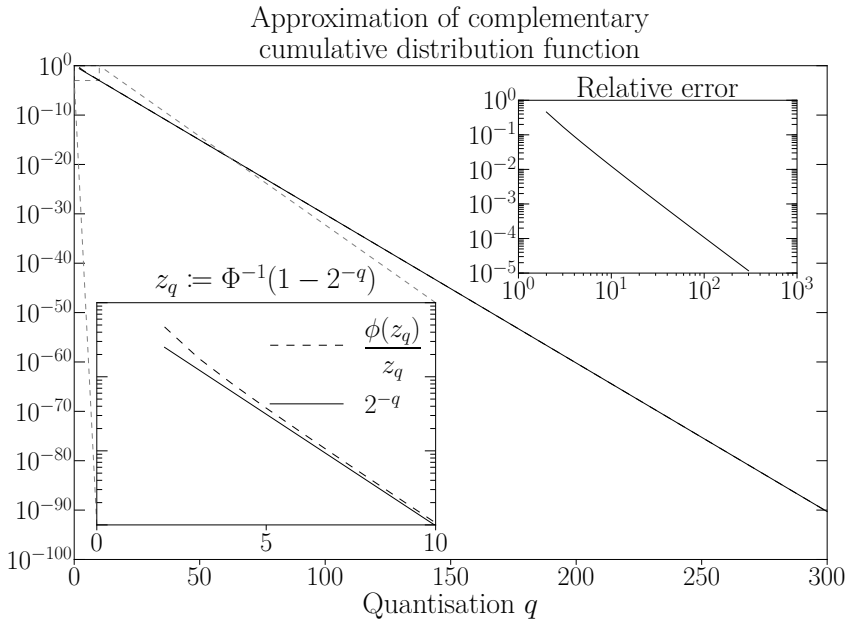


Figure 3.2.4 – Approximation of the Gaussian complementary CDF $1-\Phi$ for increasing quantisations.

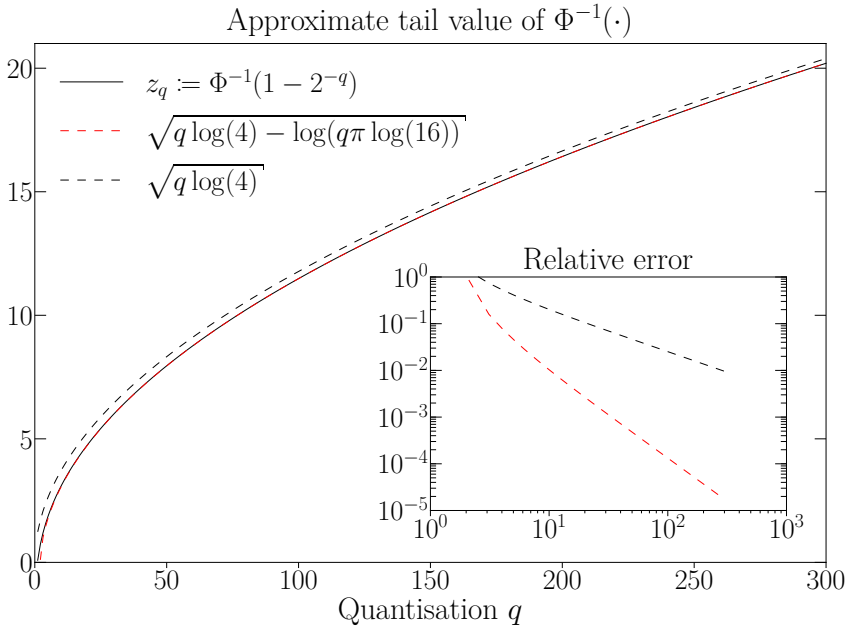


Figure 3.2.5 – The approximate tail values of $\Phi^{-1}(\cdot)$. Numerically we find Corollary 3.2.3.2 is satisfied for $q \geq 1$.

Corollary 3.2.3.3. $\int_0^z \phi(s)^{1-p}\,\mathrm{d}s \approx \frac{\phi(z)^{1-p}}{(p-1)z}$ *and* $\int_z^\infty (s-z)^p\phi(s)\,\mathrm{d}s \approx \frac{p!\phi(z)}{z^{p+1}}$ *for integers* $p \geq 2$.

Proof. The proof $\int_0^z \phi(s)^{1-p}\,\mathrm{d}s \approx \frac{\phi(z)^{1-p}}{(p-1)z}$ is motivated from integration by parts, and the relation is demonstrated [90, Lemma 8] by applying L'Hôpital's rule

$$\lim_{z\to\infty} \frac{\int_0^z \phi(s)^{1-p}\,\mathrm{d}s}{\left(\frac{\phi(z)^{1-p}}{(p-1)z}\right)} = \lim_{z\to\infty} \frac{\phi(z)^{1-p}}{\phi(z)^{1-p}\left(1-\frac{1}{(p-1)z^2}\right)} = \lim_{z\to\infty} \frac{1}{\left(1-\frac{1}{(p-1)z^2}\right)} = 1. \tag{3.2.10}$$

However, the second integral $\int_z^\infty (s-z)^p \phi(s)\,\mathrm{d}s$ is a little more involved, but again proceeds by integrating by parts. We differentiate $(s-z)^p$ to give $\frac{\mathrm{d}}{\mathrm{d}s}(s-z)^p = p(s-z)^{p-1}$ and integrate $\phi(s)$, where we recall from Lemma 3.2.3 that $\int_z^\infty \phi(s)\,\mathrm{d}s = \phi(z)(\frac{1}{z} - \frac{1}{z^3} + \frac{3}{z^5} - \cdots)$ and hence $\int_{-\infty}^z \phi(s)\,\mathrm{d}s = 1 - \phi(z)(\frac{1}{z} - \frac{1}{z^3} + \frac{3}{z^5} - \cdots)$, hence giving $\frac{\mathrm{d}}{\mathrm{d}s}(\phi(s)(\frac{1}{s} - \frac{1}{s^3} + \frac{3}{s^5} - \cdots)) = -\phi(s)$. Performing the integration by parts gives

$$\int_z^\infty (s-z)^p \phi(s)\,\mathrm{d}s \lessapprox \underbrace{\left[p(s-z)^{p-1}\phi(s)\left(-\frac{1}{s} + \frac{1}{s^3} - \frac{3}{s^5} + \cdots\right)\right|_z^\infty}_{=0} \tag{3.2.11}$$

$$+ p\int_z^\infty (s-z)^{p-1}\phi(s)\left(\frac{1}{s} - \frac{1}{s^3} + \frac{3}{s^5} - \cdots\right)\mathrm{d}s$$

upper bounding $\frac{1}{s} - \frac{1}{s^3} + \frac{3}{s^5} - \cdots$ by $\frac{1}{s}$

$$\lessapprox p\int_z^\infty (s-z)^{p-1}\frac{\phi(s)}{s}\,\mathrm{d}s \tag{3.2.12}$$

upper bounding $\frac{1}{s}$ by $\frac{1}{z}$ in the integrand, as $\phi(s)$ will decay much faster, gives

$$\lessapprox \frac{p}{z}\int_z^\infty (s-z)^{p-1}\phi(s)\,\mathrm{d}s \tag{3.2.13}$$

iteratively evaluating this integral $p-1$ more times

$$\lessapprox \frac{p!}{z^p}\int_z^\infty \phi(s)\,\mathrm{d}s \tag{3.2.14}$$

$$\lessapprox \frac{p!}{z^p}\left(\frac{\phi(z)}{z}\right) \tag{3.2.15}$$

$$\lessapprox \frac{p!\phi(z)}{z^{p+1}} \tag{3.2.16}$$

where the last approximate inequality is tight. **QED**

For numerical vindication of these bounds we show their convergence in Figures 3.2.6 and 3.2.7, and again find that these are reasonable for even modest Gaussian values. (We can notice from Figure 3.2.6 that the factor of z in the denominator in the approximation from Corollary 3.2.3.3 causes the approximation to diverge as $z \to 0$, although we notice that Corollary 3.2.3.3 assumes $z \gg 1$, and thus is not problematic for our purposes).

Lemma 3.2.4. *Suppose we have a* q-bit *piecewise constant approximation* $Q(u) \coloneqq \tilde{\Phi}^{-1}(u) \approx \Phi^{-1}(u)$ *from an equipartitioned discretisation of* $\Phi^{-1}(\cdot)$*, such as those constructed in Lemma 3.2.2 and Corollary 3.2.2.1. Denote the interval* $I_k \coloneqq (k2^{-q}, (k+1)2^{-q}) \equiv (u_k, u_{k+1})$ *for* $k \in \{0, 1, 2, \ldots, 2^q - 1\}$ *where* $K \equiv 2^q - 1$ *such that on this interval* $Q_k \coloneqq Q(u)$ *for* $u \in I_k$*. We assume there exists a constant* C *which is independent of* q *such that the following conditions are satisfied:*

1. $Q_k = -Q_{K-k}$ *for* $k \in \{0, 1, 2, \ldots, K\}$.
2. $\Phi^{-1}(u_k) \leq Q_k \leq \Phi^{-1}(u_{k+1})$ *for* $k \in \{2^{q-1}, 2^{q-1}+1, \ldots, K-1\}$.
3. $\Phi^{-1}(u_K) \leq Q_K \leq \Phi^{-1}(u_K) + Cq^{-1/2}$.

Then for any integer $p \geq 2$ *we have* $\mathbb{E}(|\Phi^{-1}(U) - Q(U)|^p) = \mathcal{O}(2^{-q}q^{-p/2})$ *for* $q \gg 1$.

Proof. We can consider $\mathbb{E}(|\Phi^{-1}(U) - Q(U)|^p)$ evaluated across the various partitioned intervals, where $\mathbb{E}(|\Phi^{-1}(U) - Q(U)|^p) = \sum_{k=0}^K A_k = 2\sum_{k=2^{q-1}}^K A_k$ and $A_k \coloneqq \int_{u_k}^{u_{k+1}} |\Phi^{-1}(U) - Q_k|^p\,\mathrm{d}U$ for $k \in \{2^{q-1}, 2^{q-1}+1, \ldots, K\}$ with $u_{K+1} \coloneqq 1$. As $\Phi^{-1}(\cdot)$ is convex, then for an interval I_k with $k \in \{2^{q-1}, 2^{q-1}+1, \ldots, K-1\}$ we can use the intermediate value theorem (Theorem A.1.4) to show that there exists a point $\xi_k \in I_k$ such that $Q_k = \Phi^{-1}(\xi_k)$. Furthermore, by use of the mean value theorem (Theorem A.1.5) and noting that $\frac{\mathrm{d}}{\mathrm{d}x}\Phi^{-1}(x) = \frac{1}{\phi(\Phi^{-1}(x))}$ there exists a position $\eta_k \in [u_k, \xi_k)$ such that

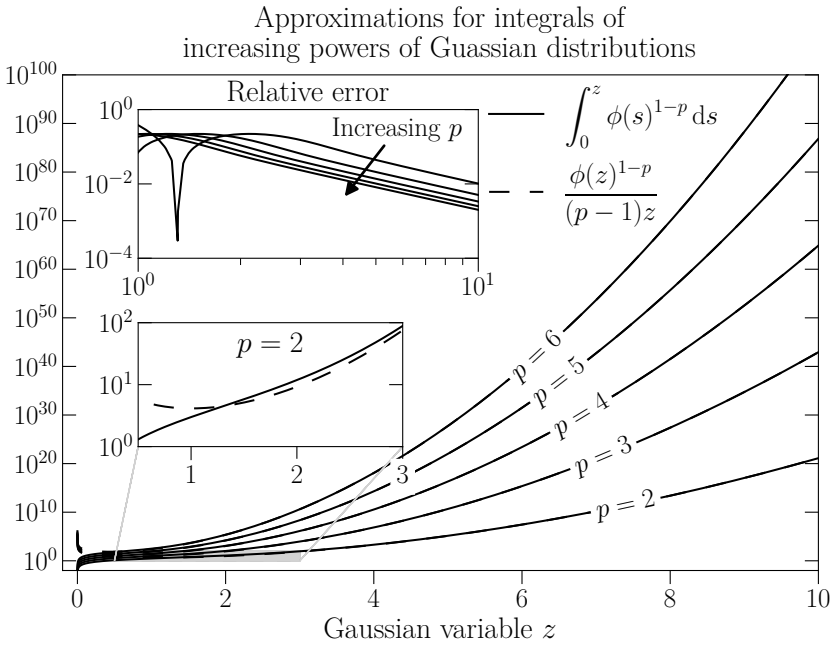


Figure 3.2.6 – Approximation of the integral $\int_0^z \phi(s)^{1-p}\,\mathrm{d}s$ for increasing powers of the Gaussian probability density function.

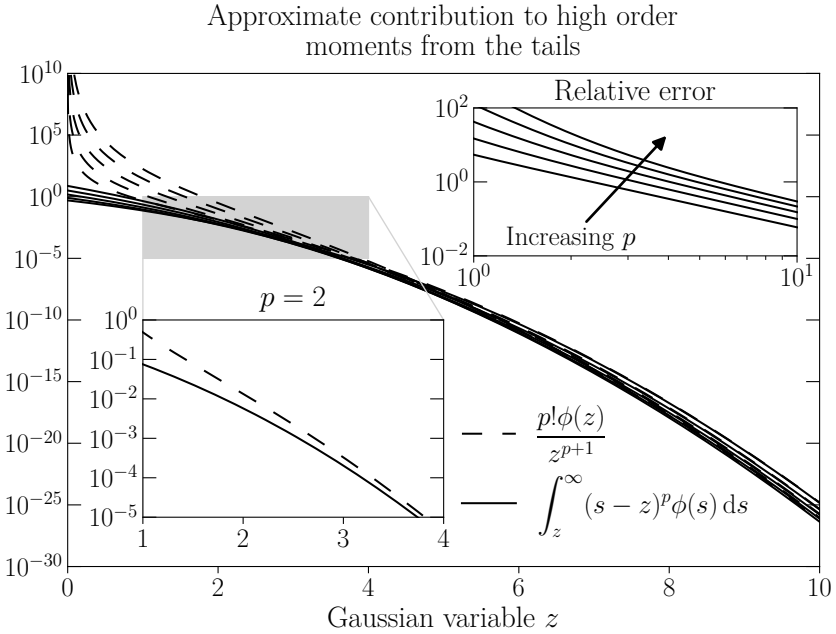


Figure 3.2.7 – Approximation of the contribution to high order Gaussian moments from the tails.

$\Phi^{-1}(u) - \Phi^{-1}(\xi_k) = \frac{u-\xi_k}{\phi(\Phi^{-1}(\eta_k))}$. Furthermore, as $\phi(\cdot)$ is monotonically decreasing on $(\frac{1}{2}, 1)$, then for $u \in (u_k, u_{k+1})$ we have $\left|\Phi^{-1}(u) - \Phi^{-1}(\xi_k)\right| \lessapprox \frac{2^{-q}}{\phi(Z_{k+1})}$. Substituting this result in our expression for $\mathbb{E}(\left|\Phi^{-1}(U) - Q(U)\right|^p)$ gives

$$\mathbb{E}(\left|\Phi^{-1}(U) - Q(U)\right|^p) = 2 \sum_{k=2^{q-1}}^{K-1} A_k + 2A_K \tag{3.2.17}$$

$$= 2 \sum_{k=2^{q-1}}^{K-1} \int_{u_k}^{u_{k+1}} \left|\Phi^{-1}(U) - Q_k\right|^p \mathrm{d}U + 2A_K \tag{3.2.18}$$

using $|\Phi^{-1}(u) - Q_k| \lessapprox \frac{2^{-q}}{\phi(Z_{k+1})}$ for $u \in (u_k, u_{k+1})$, and $u_{k+1} - u_k = 2^{-q}$

$$\lessapprox 2 \sum_{k=2^{q-1}}^{K-1} 2^{-q} \left(\frac{2^{-q}}{\phi(Z_{k+1})} \right)^p + 2A_K \tag{3.2.19}$$

$$\lessapprox 2^{-qp-q+1} \sum_{k=2^{q-1}}^{K-1} \phi(Z_{k+1})^{-p} + 2A_K \tag{3.2.20}$$

we bound all but the last term in the summation by a translated integral

$$\lessapprox 2^{-pq+1} \left(\int_{\frac{1}{2}}^{1-2^{-q}} \phi(\Phi^{-1}(U))^{-p} \, \mathrm{d}U + 2^{-q} \phi(Z_K)^{-p} \right) + 2A_K \tag{3.2.21}$$

$$\lessapprox 2^{-pq+1} \int_0^{Z_K} \phi(Z)^{1-p} \, \mathrm{d}Z + 2^{-q+1} Z_K^{-p} \left(\frac{2^q \phi(Z_K)}{Z_K} \right)^{-p} + 2A_K \tag{3.2.22}$$

using Corollary 3.2.3.3

$$\lessapprox 2^{-pq+1} \frac{\phi(Z_K)^{1-p}}{(p-1)Z_K} + 2^{-q+1} Z_K^{-p} \left(\frac{2^q \phi(Z_K)}{Z_K} \right)^{-p} + 2A_K \tag{3.2.23}$$

$$\lessapprox \frac{2^{-q+1} Z_K^{-p}}{p-1} \left(\frac{2^q \phi(Z_K)}{Z_K} \right)^{1-p} + 2^{-q+1} Z_K^{-p} \left(\frac{2^q \phi(Z_K)}{Z_K} \right)^{-p} + 2A_K \tag{3.2.24}$$

using Corollary 3.2.3.2

$$\lessapprox \frac{2^{-q+1} Z_K^{-p}}{p-1} + 2^{-q+1} Z_K^{-p} + 2A_K \tag{3.2.25}$$

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2A_K \tag{3.2.26}$$

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2 \int_{U_K}^1 \left| \Phi^{-1}(U) - \tilde{Z}_K \right|^p \mathrm{d}U \tag{3.2.27}$$

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2 \int_{U_K}^1 \left| \Phi^{-1}(U) - Z_K + Z_K - \tilde{Z}_K \right|^p \mathrm{d}U \tag{3.2.28}$$

using Jensen's inequality for summations (Lemma A.1.7) and condition 3

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2^p \int_{Z_K}^\infty |Z - Z_K|^p \phi(Z) \, \mathrm{d}Z + 2^{p-q} C^p q^{\frac{-p}{2}} \tag{3.2.29}$$

using Corollary 3.2.3.3

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2^p p! \frac{\phi(Z_K)}{Z_K^{p+1}} + 2^{p-q} C^p q^{\frac{-p}{2}} \tag{3.2.30}$$

using Corollary 3.2.3.2

$$\lessapprox 2^{-q+1} \left(\frac{p}{p-1} \right) Z_K^{-p} + 2^{p-q} p! Z_K^{-p} + 2^{p-q} C^p q^{\frac{-p}{2}} \tag{3.2.31}$$

using Corollary 3.2.3.2

$$\lessapprox \mathcal{O}\left(2^{-q} q^{-p/2} \right), \tag{3.2.32}$$

where the coefficient for $\mathcal{O}(2^{-q} q^{-p/2})$ is only a function of p and not of q. **QED**

Corollary 3.2.4.1. *The approximations constructed with Lemma 3.2.2 and Corollary 3.2.2.1 satisfy conditions 1–3 in Lemma 3.2.4.*

Proof. For approximations constructed with Lemma 3.2.2 and Corollary 3.2.2.1, these immediately satisfy conditions 1 and 2. As the construction from Lemma 3.2.2 acts as an upper bound on those from Corollary 3.2.2.1, it suffices to show the former satisfies condition 3. We recall the notation from Lemma 3.2.2 that $\tilde{Z}_k \coloneqq \mathbb{E}(Z \mid \Phi(Z) \in I_k)$ and

introduce $Z_k \coloneqq \Phi^{-1}(u_k)$. We can then consider the difference

$$\tilde{Z}_K - Z_K = 2^q \int_{U_K}^{1} \Big(\Phi^{-1}(U) - \Phi^{-1}(U_K)\Big)\,\mathrm{d}U \tag{3.2.33}$$

$$= 2^q \int_{Z_K}^{\infty} (Z - Z_K)\phi(Z)\,\mathrm{d}Z \tag{3.2.34}$$

using Corollary 3.2.3.3

$$\lessapprox 2^q \frac{\phi(Z_K)}{Z_K^2} \tag{3.2.35}$$

$$\lessapprox 2^q \frac{1}{Z_K}\left(\frac{\phi(Z_K)}{Z_K}\right) \tag{3.2.36}$$

from Corollary 3.2.3.2 we have $\frac{1}{Z_K} \lessapprox \frac{1}{\sqrt{q\log(4)}}(1 + \frac{\log(q\pi\log(16))}{q\log(16)}) \leq \frac{1}{\sqrt{q\log(4)}}(1 + \frac{\pi}{\mathrm{e}}) < \frac{1.831}{\sqrt{q}}$ for $q \geq 3$

$$< \frac{1.831}{\sqrt{q}} 2^q \left(\frac{\phi(Z_K)}{Z_K}\right) \tag{3.2.37}$$

using Corollary 3.2.3.2

$$< \frac{1.831}{\sqrt{q}} \left(1 - \frac{1}{Z_K^2}\right)^{-1} \tag{3.2.38}$$

as $(1 - Z_K^{-2})^{-1} \leq 4.1$ for $q \geq 3$

$$< \frac{7.5}{\sqrt{q}} \tag{3.2.39}$$

for $q \geq 3$. By choosing an appropriate constant C to replace the 7.5 under the assumption $q \gg 1$ this satisfies condition 3. **QED**

Taking $C = 7.5$ in condition 3 from Lemma 3.2.4, we can evaluate the bound from Lemma 3.2.4 using (3.2.31). Taking $p = 2$ and $q = 10$ we obtain the bound $\mathbb{E}(|\Phi^{-1}(U) - Q(U)|^p) < 0.023$, which would correspondingly bound the RMSE by 0.15. We can see from Figure 3.2.3 that the value is approximately 0.015, and thus our bound, whilst not especially tight, is within an order of magnitude and thus still of practical use. Interestingly, lower bounds may be possible using related work by Xu and Berger [217], although for our purposes upper bounds are our primary concern.

3.3 Geometrically partitioned piecewise linear approximations

There are two natural improvements to the equipartitioned piecewise constant approximation we have previously developed. The first will be to increase the order of the piecewise polynomial from a constant to a linear function, (or possibly even higher orders). The second would be to cluster the intervals more densely around the singularities. Within this section we present a generalised analysis to a geometric partitioning, where for simplicity in the analysis we consider using piecewise linear polynomial approximations within each partitioning interval. Although we consider a rather generalised geometric partitioning, a case of particular interest will be when this partitioning is on dyadic intervals, hence we will also often refer to this as a *dyadic* approximation.

We saw for the piecewise constant approximation there were several choices available for assigning each intervals' constant value. Similarly, there are numerous possibilities for assigning a gradient and intercept for a piecewise linear approximation within an interval. For analytic tractability in our proof of Lemma 3.3.1, will use the L^2 optimal linear function. (This choice will ensure the functionals determining the gradient and intercept are linear, crucially allowing us to apply the Peano kernel theorem (Theorem A.1.6) in our proof).

Lemma 3.3.1. *We construct a symmetric approximation $D(\cdot) \approx \Phi^{-1}(\cdot)$, with K intervals in the range $[0, \frac{1}{2}]$, where we define the k-th interval*

$$I_k \coloneqq \begin{cases} [\frac{r^k}{2}, \frac{r^{k-1}}{2}] & \text{for } k \in \{1, 2, \dots, K-1\} \\ [0, \frac{r^{K-1}}{2}] & k = K \end{cases} \tag{3.3.1}$$

for some decay rate $0 < r < 1$. Within each of the intervals we have a piecewise approximation $D_k(u_k) \equiv D(u_k)$ for any $u_k \in I_k$, where we take $D_k(\cdot)$ to be a linear function, making $D(\cdot)$ a piecewise linear function. The gradient and intercept of each $D_k(\cdot)$ is set by performing the L^2-error minimisation $D_k \coloneqq \operatorname{argmin}_{D' \in \mathcal{P}_1} \mathbb{E}(|\Phi^{-1}(u) - D'(u)|^2 \mathbb{1}_{\{u \in I_k\}})$ where $\mathcal{P}_1$ is the set of all 1^{st} order polynomials. Then for a given uniform random variable $U \sim \mathcal{U}([0,1])$ and corresponding Gaussian random variable $Z \coloneqq \Phi^{-1}(U)$ and approximate Gaussian random variable $\tilde{Z} \coloneqq D(U)$, we have

$$\mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^p\Big) = \mathcal{O}\Big((1-r)^{2p}\Big) + \mathcal{O}\left(r^{K-1} \log^{-\frac{p}{2}}\left(r^{1-K}\sqrt{\frac{2}{\pi}}\right)\right) \tag{3.3.2}$$

for any $1 \leq p < \infty$, which is equivalent to

$$= \mathcal{O}\Big((1-r)^{2p}\Big) + o\Big(r^{K-1}\Big). \tag{3.3.3}$$

Proof. As in the proof of Lemma 3.2.4 we begin by breaking down the expectation into the contribution from all the intervals without the singularity, and the final interval with the singularity, where

$$\mathbb{E}\Big(\Big|\Phi^{-1}(U) - D(U)\Big|^p\Big) = 2\sum_{k=1}^{K-1} \int_{I_k} \Big|\Phi^{-1}(u) - D_k(u)\Big|^p \,\mathrm{d}u + 2\int_{I_K} \Big|\Phi^{-1}(u) - D_K(u)\Big|^p \,\mathrm{d}u\,. \tag{3.3.4}$$

We first consider the contribution coming from the non-singular intervals. To achieve this we will express the point-wise error in each interval using the Peano kernel theorem (Theorem A.1.6), and then bound this. To begin we note that for a function $f\colon [a,b] \to \mathbb{R}$ with $0 \leq a \leq b \leq \frac{1}{2}$ we can construct its L^2-optimal linear approximation $\alpha(f) + \beta(f)u$ for $u \in [a,b]$ where α and β are functionals acting on f to give the intercept and gradient of the linear approximation respectively. If we define the point-wise error at a point $u \in [a,b]$ as $f(u) - \alpha(f) - \beta(f)u$ then we note that this is a linear functional $L_u(\cdot)$ acting on f where $L_u(f)\colon [a,b] \to \mathbb{R}$ with $L_u(f)(u) \coloneqq f(u) - \alpha(f) - \beta(f)u$. Noticing that by construction this operator must annihilate linear functions, we can then construct the Peano kernel $k(\xi; u)$ for $\xi \in [a,b]$ where $k(\xi;u) \coloneqq (u-\xi)^+ - \alpha((\cdot - \xi)^+) - \beta((\cdot-\xi)^+)u \equiv (u-\xi)^+ - \bar{\alpha}(\xi) - \bar{\beta}(\xi)u$, with $\bar{\alpha}(\xi) \coloneqq \alpha((\cdot - \xi)^+)$ and similarly for $\bar{\beta}(\xi)$. Substituting this into the Peano kernel theorem (Theorem A.1.6) gives the point-wise error $\varepsilon(u)$ at $u \in [a,b]$ as $\varepsilon(u) \coloneqq L_u(f)(u) = \int_a^b k(\xi;u) f''(\xi)\,\mathrm{d}\xi$. (In our application, we have $f \equiv \Phi^{-1}$, but we will continue with the more general notation using f until the specific results concerning $\Phi^{-1}(\cdot)$ are required).

The first step to using the Peano kernel theorem (Theorem A.1.6) will be to determine the L^2-optimal intercept and coefficient required for the kernel. Notice that for f its L^2-optimal linear approximation must have its functional derivative with respect to α and β be zero such that $\frac{\delta}{\delta\alpha}\int_a^b (f(u) - \alpha(f) - \beta(f)u)^2\,\mathrm{d}u = 0$ and $\frac{\delta}{\delta\beta}\int_a^b (f(u) - \alpha(f) - \beta(f)u)^2\,\mathrm{d}u = 0$. These simplify to the simultaneous equations

$$\alpha(f)(b-a) + \beta(f)\left(\frac{b^2-a^2}{2}\right) = \int_a^b f(u)\,\mathrm{d}u \tag{3.3.5}$$

and

$$\alpha(f)\left(\frac{b^2-a^2}{2}\right) + \beta(f)\left(\frac{b^3-a^3}{3}\right) = \int_a^b u f(u)\,\mathrm{d}u\,. \tag{3.3.6}$$

When we consider the appropriate Peano kernel function $f \to (\cdot - \xi)^+$ then we have $\alpha(f) \to \bar{\alpha}(\xi)$ and similarly for $\bar{\beta}$, the integrals are readily evaluated, and the simultaneous equations give

$$\bar{\alpha}(\xi) = -\frac{(b-\xi)^2((b+a)\xi - 2a^2)}{(b-a)^3} \qquad \text{and} \qquad \bar{\beta}(\xi) = \frac{(b-\xi)^2(2\xi + b - 3a)}{(b-a)^3}. \tag{3.3.7}$$

Now having an expression for the Peano kernel function, we can write the point-wise error in full as $\varepsilon(u) = \int_a^b ((u-\xi)^+ - \bar{\alpha}(\xi) - \bar{\beta}(\xi)u) f''(\xi)\,\mathrm{d}\xi$. If we linearly rescale our variables from the interval $[a,b] \to [0,1]$ under $\eta \to \tilde{\eta}$ where $\tilde{\eta} \coloneqq \frac{\eta - a}{b-a}$ then after simplifying our expressions for $\bar{\alpha}$ and $\bar{\beta}$ this becomes $\varepsilon(u) = (b-a)^2 \int_0^1 ((\tilde{u} - \tilde{\xi})^+ - (1 - \tilde{\xi}^2)(\tilde{\xi} + \tilde{u})) f''((b-a)\tilde{\xi} + a)\,\mathrm{d}\tilde{\xi}$. Taking the absolute value, using Jensen's inequality for expectations (Lemma A.2.2) and Jensen's inequality for summations (Lemma A.1.7), and knowing that the maximum value for f'' is obtained at the lower boundary when $f \equiv \Phi^{-1}$, then we obtain the bound $|\varepsilon(u)| \leq -6(b-a)^2 f''(a)$. Here we can note that this is a similar (but not as tight) result to that obtained from using a linear interpolation approximation [24, Theorem 16.1], where for a function f and its linear interpolant $\tilde{f}$ over an interval $[a,b]$ that $|f(x) - \tilde{f}(x)| \leq \frac{1}{2}(b-a)^2 \sup_{\xi \in [a,b]} |f''(\xi)|$. We will see that in subsequent analysis that the tightness of our bound is not critical.

We can readily rearrange this error to show $|\varepsilon(u)| \leq -6a^2 f''(a)(\frac{1-r}{r})^2$. Using the chain rule and noting that $\frac{\mathrm{d}\Phi^{-1}(u)}{\mathrm{d}u} = \frac{1}{\phi(\Phi^{-1}(u))}$ and $\frac{\mathrm{d}^2\Phi^{-1}(u)}{\mathrm{d}u^2} = \frac{\Phi^{-1}(u)}{\phi^2(\Phi^{-1}(u))}$ the error can be differentiated with respect to a and bounded by its maximum ε^*. Taking logarithms of the integrand and differentiating we can show that the maximum is at $a = u'$ and must satisfy $0 = \frac{2}{\Phi(\Phi^{-1}(u'))} + (2\Phi^{-1}(u') + \frac{1}{\Phi^{-1}(u')})\frac{1}{\phi(\Phi^{-1}(u'))}$. The solution of this can numerically be found as $u' \approx 0.177$, and hence the error can be bounded by approximately 2.59 for each non-singular interval.

Inspecting then the L^p-error from the non-singular intervals we obtain

$$\sum_{k=1}^{K-1} \int_{I_k} \left|\Phi^{-1}(u) - D_k(u)\right|^p \mathrm{d}u < \sum_{k=1}^{K-1} \int_{I_k} |\varepsilon^*(I_k)|^p \,\mathrm{d}u \tag{3.3.8}$$

using our bound for ε^*

$$< 2.59^p \left(\frac{1-r}{r}\right)^{2p} \sum_{k=1}^{K-1} \int_{I_k} \mathrm{d}u \tag{3.3.9}$$

taking the limit $r \to 1$ and $K \to \infty$ such that $\frac{r^K}{2} = C$ for some constant $0 < C < \frac{1}{2}$, then the summation becomes

$$< 2.59^p \left(\frac{1-r}{r}\right)^{2p} \int_C^{\frac{1}{2}} \mathrm{d}u \tag{3.3.10}$$

bounding the integral by $\frac{1}{2}$ gives

$$< \frac{2.59^p}{2} \left(\frac{1-r}{r}\right)^{2p} \tag{3.3.11}$$

which as $r \to 1$ is $\mathcal{O}((1-r)^{2p})$.

It only remains to consider the interval I_K. We begin by finding the coefficients of the optimal fit. For the I_K interval, identical to what we had previously in (3.3.5) to (3.3.6), we obtain $\alpha = \frac{1}{b}\int_0^b \Phi^{-1}(u)\,\mathrm{d}u - \frac{b\beta}{2}$ and $\beta = \frac{12}{b^3}\int_0^b (u - \frac{b}{2})\Phi^{-1}(u)\,\mathrm{d}u$. We first evaluate β

$$\beta = \frac{12}{b^3} \int_0^b \left(u - \frac{b}{2}\right) \Phi^{-1}(u)\,\mathrm{d}u \tag{3.3.12}$$

letting $Z_b \coloneqq \Phi^{-1}(b)$

$$= \frac{12}{b^3} \int_{-\infty}^{Z_b} \left(\Phi(z) - \frac{b}{2}\right) z\phi(z)\,\mathrm{d}z \tag{3.3.13}$$

integrating by parts

$$= \frac{12}{b^3}\left(\left[-\left(\Phi(z)-\frac{b}{2}\right)\phi(z)\right|_{-\infty}^{Z_b} + \int_{-\infty}^{Z_b}\phi^2(z)\,\mathrm{d}z\right) \tag{3.3.14}$$

$$= \frac{12}{b^3}\left(-\frac{b\phi(Z_b)}{2} + \frac{\Phi(\sqrt{2}Z_b)}{\sqrt{4\pi}}\right) \tag{3.3.15}$$

$$= \frac{6}{b^3}\left(\frac{\Phi(\sqrt{2}Z_b)}{\sqrt{\pi}} - b\phi(Z_b)\right). \tag{3.3.16}$$

Next we evaluate α

$$\alpha = \frac{1}{b}\int_0^b \Phi^{-1}(u)\,\mathrm{d}u - \frac{b\beta}{2} \tag{3.3.17}$$

$$= \frac{1}{b}\int_{-\infty}^{Z_b} z\phi(z)\,\mathrm{d}z - \frac{b\beta}{2} \tag{3.3.18}$$

$$= -\frac{\phi(Z_b)}{b} - \frac{b\beta}{2} \tag{3.3.19}$$

$$= \frac{2\phi(Z_b)}{b} - \frac{3\Phi(\sqrt{2}Z_b)}{b^2\sqrt{\pi}}. \tag{3.3.20}$$

A useful result which we will require will be to evaluate $\beta\phi(Z_b)$, for which we have

$$\beta\phi(Z_b) = \frac{6}{b^3}\left(\frac{\Phi(\sqrt{2}Z_b)}{\sqrt{\pi}} - b\phi(Z_b)\right)\phi(Z_b) \tag{3.3.21}$$

Corollary 3.2.3.1 implies $\Phi(\sqrt{2}Z_b) \approx -\frac{\phi(\sqrt{2}Z_b)}{\sqrt{2}}(\frac{1}{Z_b} - \frac{1}{2Z_b^3})$ and $\phi(\sqrt{2}Z_b) \equiv \sqrt{2\pi}\phi^2(Z_b)$ gives

$$\approx -\frac{6\phi(Z_b)}{b^3}\left(\phi^2(Z_b)\left(\frac{1}{Z_b} - \frac{1}{2Z_b^3}\right) + b\phi(Z_b)\right) \tag{3.3.22}$$

$$\approx -\frac{6\phi(Z_b)}{b}\left(\frac{\phi^2(Z_b)}{b^2}\left(\frac{1}{Z_b}\right)\left(1 - \frac{1}{2Z_b^2}\right) + \frac{\phi(Z_b)}{b}\right) \tag{3.3.23}$$

using Corollary 3.2.3.1 implies $\frac{\phi(Z_b)}{b} \approx -Z_b(1 + \frac{1}{Z_b^2})$ giving

$$\approx -\frac{6\phi(Z_b)}{b}\left(\frac{Z_b^2}{Z_b}\left(1 + \frac{2}{Z_b^2}\right)\left(1 - \frac{1}{2Z_b^2}\right) - Z_b\left(1 + \frac{1}{Z_b^2}\right)\right) \tag{3.3.24}$$

$$\approx -\frac{6Z_b\phi(Z_b)}{b}\left(\frac{1}{2Z_b^2} + \mathcal{O}\left(Z_b^{-4}\right)\right) \tag{3.3.25}$$

$$\approx \frac{-3\phi(Z_b)}{bZ_b} \tag{3.3.26}$$

using Lemma 3.2.3

$$\approx 3. \tag{3.3.27}$$

It is worth briefly remarking on an interesting consequence of this result. We can rearrange our expression to show $\beta b \approx \frac{-3}{Z_b}$, where βb is the range of values our approximation spans in this singular interval. Hence, as our singular interval becomes ever tighter around the singularity, the range of values our approximation takes is decreasing, and our approximation flattens relative to the interval $[0, b]$ as b decreases.

With the gradient and intercept in the singular interval known exactly, we can directly write the L^p-error in the singular interval

$$\int_{I_K}\left|\Phi^{-1}(u) - D_K(u)\right|^p \mathrm{d}u = \int_{-\infty}^{Z_b}|z - \alpha - \beta\Phi(z)|^p\phi(z)\,\mathrm{d}z\,. \tag{3.3.28}$$

We know D_K intercepts Φ^{-1} at two locations (as Φ^{-1} is convex). Denoting the smaller and larger of these as u_- and u_+ respectively where $u_- < u_+$ with the corresponding Gaussian values $Z_- \coloneqq \Phi^{-1}(u_-)$ and $Z_+ \coloneqq \Phi^{-1}(u_+)$, we can split our integration domain into

$$= \int_{-\infty}^{Z_-} |z - \alpha - \beta\Phi(z)|^p \phi(z)\,\mathrm{d}z + \int_{Z_-}^{Z_b} |z - \alpha - \beta\Phi(z)|^p \phi(z)\,\mathrm{d}z\,. \tag{3.3.29}$$

Within these domains we have the bounds

$$\leq \int_{-\infty}^{Z_-} |z - Z_-|^p \phi(z)\,\mathrm{d}z + \int_{Z_-}^{Z_b} |Z_+ - \alpha - \beta\Phi(z)|^p \phi(z)\,\mathrm{d}z \tag{3.3.30}$$

$$\leq \int_{-\infty}^{Z_-} |z - Z_b|^p \phi(z)\,\mathrm{d}z + \int_{-\infty}^{Z_b} |Z_+ - \alpha - \beta\Phi(z)|^p \phi(z)\,\mathrm{d}z \tag{3.3.31}$$

$$\leq \int_{-\infty}^{Z_b} |z - Z_b|^p \phi(z)\,\mathrm{d}z + |\beta b|^p \int_0^b \mathrm{d}u \tag{3.3.32}$$

$$\leq \int_{-\infty}^{Z_b} |z - Z_b|^p \phi(z)\,\mathrm{d}z + |\beta b|^p b \tag{3.3.33}$$

Corollary 3.2.3.3 gives

$$\leq \frac{p!\phi(Z_b)}{|Z_b|^{p+1}} + \left|\frac{3}{Z_b}\right|^p b \tag{3.3.34}$$

using Lemma 3.2.3 to give $\Phi(Z_b) \approx -\frac{\phi(Z_b)}{Z_b}$ and $\Phi(Z_b) = b$ gives

$$\leq \mathcal{O}\left(\frac{b}{|Z_b|^p}\right) \tag{3.3.35}$$

from the proof of Lemma 3.2.3 we have $\frac{Z_b^2}{2} \approx -\log\left(\sqrt{2\pi}b\right)$ which gives

$$\leq \mathcal{O}\left(b \log^{-\frac{p}{2}}\left(\frac{1}{\sqrt{2\pi}b}\right)\right) \tag{3.3.36}$$

$$\leq o(b) \tag{3.3.37}$$

recalling $b = \frac{r^{K-1}}{2}$

$$\leq o\left(r^{K-1}\right). \tag{3.3.38}$$

Combining this result with that from the non-singular intervals produces the desired result, and thus completes the proof. **QED**

The decay rate predicted by Lemma 3.3.1 is observed numerically, as shown in Figure 3.3.1. The bound can also be shown to remain effective for higher values of p.

To visualise the approximation which results from the construction in Lemma 3.3.1, we consider when our intervals are dyadic, corresponding to a decay rate $r = \frac{1}{2}$. The resultant approximation and the corresponding probability density function are shown in Figures 3.3.2 and 3.3.3. We have presented a rather coarse partitioning using only 8 intervals so that the approximation in the singular interval is apparent. The approximations in the non-singular intervals attain a reasonable fidelity, so we have had to zoom in once to show the discrepancy between the two. Furthermore, if we zoom in again close to an interval boundary, we see that the approximations do not overlap, and hence there are some inaccessible regions for our approximation. We have highlighted these intervals in Figure 3.3.3 where there are miniscule regions where the probability density is zero.

It is natural to inspect Figure 3.3.2 and think that a continuous piecewise linear approximation (without the discontinuities) might be preferable. While this is undoubtedly more aesthetically pleasing, in the L^2-norm such an approximation would by definition have an inferior error to the discontinuous piecewise linear approximation. Similarly, imposing continuity as a constraint makes the intercept and gradient within each interval coupled to those from the

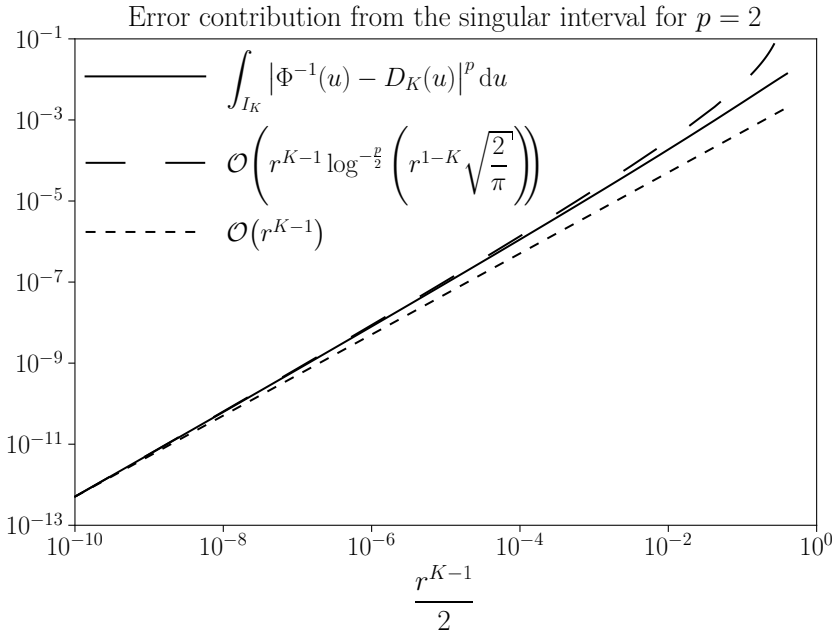


Figure 3.3.1 – The contribution to the L^p-error from the singular interval from Lemma 3.3.1. The bound remains tight also for higher values of p, which for clarity are not shown.

adjacent intervals. Hence determining these becomes a difficult optimisation problem. Thus, aesthetics aside, the discontinuous piecewise linear function is superior to a continuous version.

It is tempting to ask if there is an optimal relation between r and K, or similarly what the optimal choice for r is such that K is minimised subject to achieving some prescribed error? Similar questions arise for the piecewise constant approximation's optimal values. As mentioned, the work by Xu and Berger [217] works towards addressing this as do regular sequence designs by Ritter [181]. However, while these are very interesting questions and active research areas, such questions may be somewhat misleading in our context. The ultimate aim of our approximations is computational speed, and as we will see later in Chapters 4 and 7, the optimal choices of r and K are determined by a combination of speed and accuracy considerations. Specifically, we will see that the available hardware determines which approximation to use and its configuration. Thus, while having such bounds will be crucial for our later nested multilevel Monte Carlo applications in Chapter 7, their computational optimality is usually of more practical importance than their asymptotic optimality.

3.4 Polynomial approximations over the entire domain

So far we have been considering approximations of $\Phi^{-1}(\cdot)$ by various piecewise constant or linear functions. However, we can of course consider a polynomial approximation over the entire domain $[0, 1]$.

Lemma 3.4.1. *Any finite order polynomial approximation $\tilde{Z}^{\mathrm{P}}$ of $\Phi^{-1}(\cdot)$ has uniformly bounded moments.*

Proof. All finite order polynomial approximations $\tilde{Z}^{\mathrm{P}}$ in the domain $[0, 1]$ are bounded, and thus the moments are uniformly bounded. **QED**

To construct an approximation, the procedure would be to choose a polynomial of a given order and choose the coefficients such that the error is minimal. The error could be measured in any L^p-norms, where it is a well known result that the optimal fit in the L^2-norm is

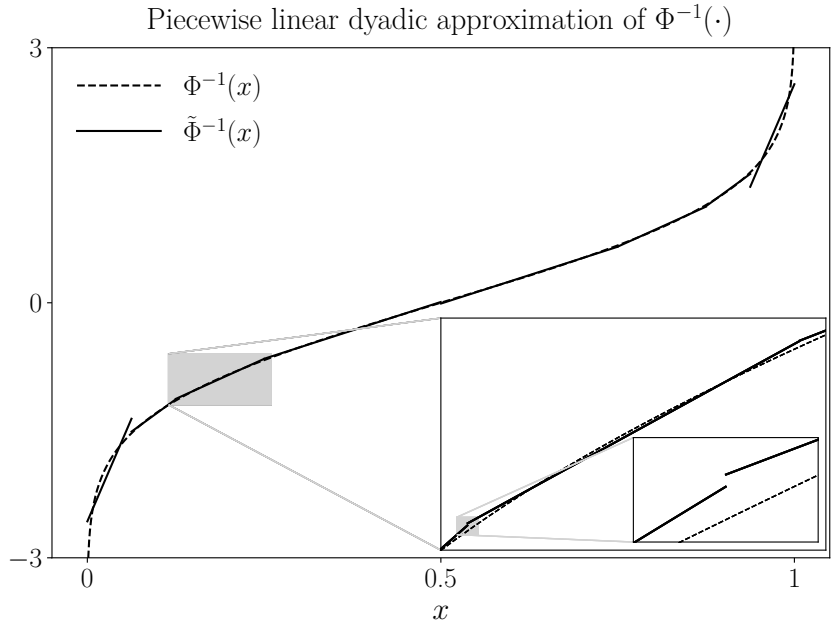


Figure 3.3.2 – An example piecewise linear approximation using 8 dyadic intervals.

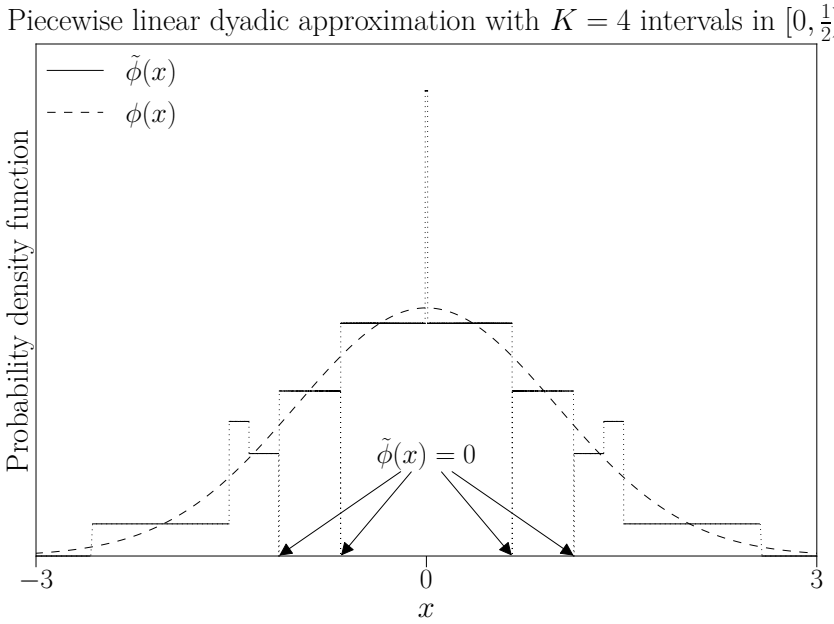


Figure 3.3.3 – An example probability density function resulting from a piecewise linear approximation using dyadic intervals. This example is produced from the approximation shown in Figure 3.3.2.

achieved by a truncated Legendre series approximation [195, 198] [206, 5.5, pages 123–127] [71, Theorem 3.8 and Corollary 3.1, pages 84–85], which of course uses the Legendre polynomials as an orthogonal basis [71, 6.2]. Nonetheless, while this approximation may be optimal in the L^2-norm, it is not obvious that the L^p-error should also be vanishing with increasing polynomial orders. Similarly, while a truncated Legendre series may be optimal in the L^2-norm, it is possible that a truncated Chebyshev series may by competitive in the L^2-norm and perhaps better in other higher L^p-norms. We did investigate these approximations, but ultimately were unable to show any analytic convergence results. Consequently, we only briefly mention this type of approximation in this chapter for the sake of completeness. While numeric experiments did suggest that truncated Legendre series approximations were convergent both in the L^2-norm and higher norms, in lieu of any supporting analysis we chose to omit these results. Pursuing polynomial approximations any further remains a possible avenue for further research.

For anyone considering re-investigating such polynomial approximations, we would like to highlight some of the literature we found useful. For an early review of orthogonal polynomials and their behaviour in the L^p-norm then there is substantial work by Pollard [174–177], which was extended by Newman and Rudin [163], where the most relevant sources are from Pollard [175, Theorem 8.1] [177, page 442]. Concerning the convergence of Legendre polynomials there is the work by Wang and Xiang [209], Xiang and Bornemann [216], Wang [208], and Webb et al. [210]. For Chebyshev approximations of differentiable functions there is the work by Trefethen [198, Theorem 7.1] and Canuto et al. [43, (5.5.10), page 293], and for some singular functions Boyd [30] and Trefethen et al. [200]. Regarding the asymptotic behaviour of Legendre polynomials there is the standard text Szegő [195, 8.21, Theorems 8.21.1 and 8.21.2, pages 192–193], and several bounds by Lohöfer [144, page 227, (2)], Bernstein [21], and Martin [151, page 1432, (5)].

3.4.1 Rational approximations

A powerful extension to polynomial approximations are rational approximations (and associated Padé approximations). From an analytic standpoint, the ability of rational approximations to be singular suggests them as a natural candidate for our purposes. Indeed, Blair et al. [25] uses them to approximate the inverse Gaussian cumulative distribution function, although their analysis is very limited and does not cover convergence behaviour. As demonstrated by Blair et al. [25], these approximations readily achieve machine precision accuracy, and hence form the primary technique used in most library implementations. Chapter 4 discusses rational approximations more thoroughly and highlights some of the computational shortcomings of this approach.

Chapter 4

Implementing approximate Gaussian random variables

In this chapter we introduce the common numerical schemes for computing the inverse Gaussian cumulative distribution function $\Phi^{-1}(\cdot)$. We present the evolution of the method from single to double-precision, and the transformations necessary to maintain precision in the asymptotic tail regions. The performance from various commercial and free numerical libraries are compared along with versions containing vectorised modifications. The performances are compared on Intel and Arm-based machines compiled using Intel's `icc` and Arm's `armclang`. These investigations will elucidate, motivate, and justify the development of our approximations to $\Phi^{-1}(\cdot)$, and explain the origins of their superior speeds.

4.1 Approximation schemes

Evaluating the inverse Gaussian cumulative distribution function $\Phi^{-1}(\cdot)$ has long been under the attention of scientific computing. As with numerically computing many classically analytic functions, it is only necessary to construct an approximation that is accurate to machine precision. By considering the functional form from Figure 3.1.2, we see that this can be split into a *central region* and two *tail regions*, as shown in Figure 4.1.1. Inspecting Figure 4.1.1 it is clear that the central region around $x \approx 0.5$ can easily be approximated by a polynomial or rational approximation (cf. Padé approximations [179, 5.12]). However, these approximations will lose accuracy in the asymptotic limits. To remedy this, what is typically done is the function's evaluation is decomposed into a central region, and the two asymptotes are gathered into a tail region. It is known that the tails diverge very slowly, and that the square root of the logarithm of the tails is approximately linear, which we saw earlier in Lemma 3.2.3. It is typical then to form a polynomial in terms of the square-root of the logarithm of the tails [18, 101, 165, 213]. (Moro [158] forms a good functional approximation in terms of the logarithm of the logarithm!).

4.1.1 Algorithm development

As machines have developed capabilities for storing increasingly precise numbers, the degree of precision required from approximations to $\Phi^{-1}(\cdot)$ has also increased. Typically this has involved simply increasing the degree of either the polynomial or rational approximation used. Additionally, alternate function approximations have been proposed, such as the double logarithm approach by Moro [158]. From the earliest methods presented by Hastings Jr et al. [101] to more recent works by Giles [77] and Acklam [5], a brief chronology of the schemes are presented in Table 4.1.1, including rational Padé approximations [179, 5.12], Newton-Raphson

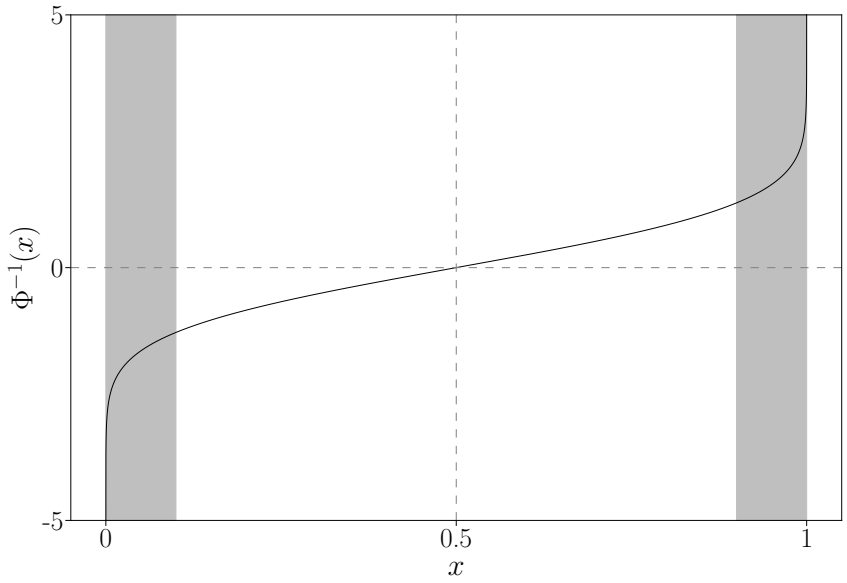


Figure 4.1.1 – The inverse Gaussian CDF function's asymptotic tail regions (shaded).

iterations and Halley's method [179, 9.4–9.4.2], and lookup tables.

Year	Author	Method	Operations
1955	Hastings Jr et al. [101]	$R_{2,3}$	$\surd, \log, \div$
1974	Odeh and Evans [165]	$R_{4,4}$	$\surd, \log, \div$
1977	Beasley and Springer [18]	$R_{3,4}$	$\surd, \log, \div$
1988	Wichura [213]	$R_{7,7}$	$\surd, \log, \div$
1994	Marsaglia et al. [150]	P_3, NR, LT	exp
2011	Giles [77]	P_{26}	$\surd, \log$
2015	Acklam [5]	$R_{6,5}$, NR	$\surd, \log, \div, \exp$

Table 4.1.1 – A brief chronology of the methods to compute $\Phi^{-1}(\cdot)$. Methods denoted $R_{a,b}$ are rational approximations with numerator and denominator orders of a and b respectively. A regular polynomial of order k is denoted by P_k. Iterations using the Newton-Raphson method are denoted NR, and use of lookup tables by LT.

4.1.1.1 Rational approximations

It is informative to briefly overview one of the most prolific methods for computing $\Phi^{-1}(\cdot)$, which is the method presented by Beasley and Springer [18] (which is in turn based on Odeh and Evans [165]), which involves forming a rational approximation. By inspecting Figure 3.1.2 we can see that by symmetry it suffices to build a good approximation for $\Phi^{-1}(\cdot)$ in the domain $x \in (0, 0.5]$. To achieve this Beasley and Springer [18] form two intermediate variables $q \coloneqq x - 0.5$, and $r \coloneqq \sqrt{\log(0.5 - |q|)}$ and compute the rational approximations

$$\Phi^{-1}(x) \approx \begin{cases} q\dfrac{A(q^2)}{B(q^2)} & |q| \leq 0.42 \\ \operatorname{sign}(q)\dfrac{C(r)}{D(r)} & \text{otherwise} \end{cases} \qquad \text{for } x \in (0, 1). \tag{4.1.1}$$

In the original work by Beasley and Springer [18] the polynomials A, B, C, and D were of orders 3, 4, 3, and 2 respectively. However, to achieve answers appropriate for double-precision these have been increased up to seventh order by Wichura [213]. For a brief but comprehensive overview of several approximation methods we recommend Brophy [34].

It is the algorithm by Wichura [213] which is used in GNU's GSL and in the Cephes library. Furthermore Python's SciPy library uses the Cephes library for its evaluation of $\Phi^{-1}(\cdot)$ (the corresponding function in Cephes being `ndtri`). The NAG equivalent (`nag_deviates_normal`) is also an extension of that by Wichura [213]. Lastly, Burkardt [39] provides ASA241 [213] as a MATLAB library for computing $\Phi^{-1}(\cdot)$ also based on the algorithm by Wichura [213], which supersedes the earlier ASA111 [18] routine based on the algorithm by Beasley and Springer [18].

4.1.1.2 High order polynomials

The appeal of rational approximations is that they typically achieve the minimal approximation error for the fewest number of coefficients. However, it is important to identify that they do require a division operation. While such division operations are regarded as relatively cheap, they are several times more expensive than *fused multiply add* (FMA) operations. A representative (but inexact) comparison would be that a single division operation is comparable to 8–40 FMA operations and with latencies 5–10 times longer [215]. For Intel's Skylake, division is approximately a factor of 6 (and a latency of 20–80 cycles) [69, page 233], and almost a factor of 10–16 for AVX vector instructions (albeit not AVX-512) [69, pages 245–246]. While these values may likely continue to improve with future hardware, it is not unreasonable to consider a division operation as being comparable in cost to 10–15 FMA operations, and with a considerably larger latency.

Having seen such larger costs associated with the division operation involved with the rational approximations, we see that the division operation can be an overlooked cost in the computation. Taking the method by Wichura [213] as an example, this requires rational approximations of seventh order, and hence requires approximately 14 FMA operations (7 each for the numerator and denominator). However, if we include the division with a relative cost of approximately 15 cycles, then (overlooking latencies for simplicity) such a method is comparable to approximately 30 cycles.

Such an observation of increased clock cycles arising from division is aligned with the motivation for the high order polynomial approach presented by Giles [77]. Here a 22–26^{th} order polynomial is used and no division is required. The benefit from using such a high order polynomial was that it allowed the central region to be widened, and the probability of hitting either of the tails was 0.2%. Comparing this to the method by Beasley and Springer [18] which has a tail probability of 16%, we see that the approach by Giles [77] makes the likelihood of random input requiring the tail calculation considerably smaller. The original motivation for the method was to have such small tails, as this approach was originally developed for implementation on *graphical processing units* (GPUs) to avoid an effect known as *warp divergence*, which is akin to conditional branching. Hence such an approach should also be considered appropriate for vectorised hardware.

4.1.2 Numerical performance

Having seen the myriad of approaches which have been developed for computing $\Phi^{-1}(\cdot)$ at ever increasing precision levels, it remains to see how well these can be adapted to perform on vectorised hardware. For these experiments we compare implementations from Intel's *vector statistics library 18.0.0* (VSL) and the NAG C library (CLL6I261DL, mark 26). It should be noted that the NAG library has two offerings, one general purpose library, and a second which is optimised for Intel hardware. Unless otherwise stated, we will use NAG's general purpose library, as this represents an out of the box commercial offering which does not make hardware assumptions. For freely available library implementations we used the *GNU scientific library 2.1* (GSL) [72], the *Cephes mathematical library 2.8* [159], and a modified version of

the `erfinv` implementation by Giles [77]. Apart from the pre-compiled libraries from Intel and NAG, the other libraries were compiled using the Intel C compiler `icc` 18.0.0.

To conduct our experiments we formed a dynamically allocated array of 50 000 input values and called the function implementations on this array 1000 times, totalling 5×10^7 calculations of $\Phi^{-1}(\cdot)$. The array size of 50 000 was chosen as 50 000 doubles, each 8 bytes in size, should occupy approximately 40% of the L_2 cache (cf. Table 2.1.1). We chose two non-overlapping arrays for the input and output locations (using in total 80% of the L_2 cache). Hence the intention of this was to keep I/O costs to a minimum, but ensure we had a large enough workload so the program ran long enough to give reliable timing results.

We considered three scenarios for the input arrays: constant input where each element of the array was the same, random inputs which were uniformly drawn, and an array of NaN values. These represent two typical use cases where the function is expected to be called on the same (or very similar) input and when the function is applied to random input, and the third showcases the typical time required for error handling. To try and ensure a fair comparison between different implementation standards, functions were modified where necessary to correctly handle edge cases such as $\{\pm 0, 1\}$, and to produce NaN values with either invalid input (e.g. $x \notin [0, 1]$) or NaN inputs. No distinction was made between regular NaN values and signalling NaN values [192, pages 175 and 350].

We can compare the performance on two different machines, an Intel machine and an Arm-based machine. The Intel machine (Intel Xeon Gold 6140) is from the Skylake series, is hyper-threaded, operates at 2.3 GHz, and has two vector processing units with 512-bit vectors. The Arm machine is a Cavium ThunderX2, operating at 2.0 GHz, not hyper-threaded, with two vector processing units with 128-bit vectors. (Newer sever units are produced by Marvell).

4.1.2.1 Skylake

The results from the different computations are shown in Figure 4.1.2. There is a lot of information contained within Figure 4.1.2, so we will deconstruct it and comment on it piece by piece.

The first observation to make is the performance of Intel and their times for *high accuracy* (HA) and *low accuracy* (LA). These are hovering at around 8–10 clock cycles, with the lower accuracy mode operating the higher speeds. At a first glance this performance appears very constant even into the tails, which it is. We only notice a tail behaviour at $x \approx 2.9 \times 10^{-11}$, when the computation time spikes to thousands of cycles, and the edge cases are at around twenty cycles. As this is a pre-compiled library it can be seen as operating by some black-box method. Based on the presentations by Cornea [49, 50] we have some suspicions about how Intel might be operating, but this is ultimately speculative, so we postpone discussing this until later.

On the other end of the spectrum we can consider the scalar implementation provided by GNU's GSL. This uses the algorithm by Wichura [213] and has a lower tail region for $x \leq 0.075$. We can see from this there is a small margin between the central and tail cases, where the central section is taking in the region of 100 cycles. Identically, the general purpose library from NAG also implements the same algorithm, and we can see this provides marginal improvements. We can lastly inspect how Cephes fairs in this category, and we see that while it has a considerably faster central region, and to a lesser degree a faster tail region, the tails in this algorithm are much wider for $x \leq 0.135$. We can see that these wide tails really diminish the gains from the faster central region, and that for random input NAG and GSL are approximately equal, while Cephes has a small lead on them.

There are three GNU GSL implementations listed, with two differentiated by the "SIMD" and "ACCUM" labels (which we produced), where ACCUM indicates an

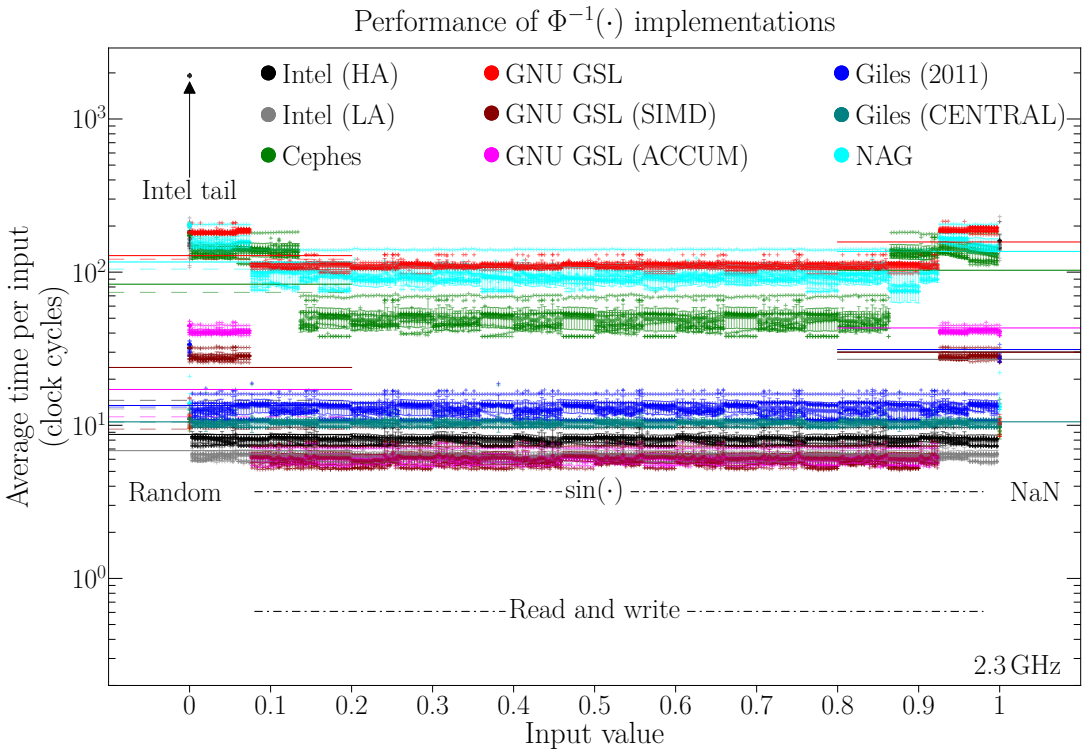


Figure 4.1.2 – A comparison of double-precision $\Phi^{-1}(\cdot)$ implementations by Intel, NAG, GNU, and Giles [77]. Intel implementations for low and high-accuracy modes are shown. Averages for NaN input are shown as solid lines on the right. Averages for random input are shown as solid lines on the left. The empirically expected averages are shown as dashed lines on the left. These data were gathered using a 64-bit x86 Intel Xeon Gold 6140 CPU operating at 2.3 GHz under GNU/Linux 4.13.0-25-generic under the Ubuntu 16.04.3 LTS distribution. C code was compiled using the Intel C compiler `icc` 18.0.0 with the flags `mkl`, `std=c11`, `qopenmp`, `qopt-report`, `O2`, `march=skylake-avx512`, `xCORE-AVX512`, and `qopt-zmm-usage=high`.

accumulating implementation (explained shortly). The "GNU GSL" data represent the original scalar implementation, while the two others demonstrate two different approaches to vectorise the algorithm. For simplicity we should first consider the "GNU GSL (SIMD)" results, and we can first notice the considerable performance improvement in the central region, dropping from 100 cycles to approximately 6, giving a speed-up by a factor of approximately 16. Considering that the hardware can fit 8 doubles into the AVX-512 vector and there are 2 vector processing units in the chip, the factor of 16 can be attributed to the SIMD vectorisation. This considerable speed-up though is limited to only the central region, and we see that the tail regions only improved by about half as much. This is because all of the edge cases and error handling are handled in this branch by SIMD operations, and hence these do more work than their scalar counterparts.

It now remains to explain the difference between the GNU GSL (SIMD/ACCUM) implementations. The original SIMD implementation proceeded as sketched in Code 4.1.1, where if an edge case was encountered within a vector operation (flagged using `tails_encountered`), then the entire vector calculation would be repeated but with the extended functionality required to handle the central, tail, and edge cases. Having considered this, we see that if a single vector element requires the tail calculation, then the vector lane diverges and all the vector elements undergo the more costly calculation. This effect of vector contamination/divergence can be readily seen when comparing the average times for random input. The solid lines on the left of Figure 4.1.2 show the average time for random input. The dashed lines on the left are the empirically determined averages, which are the ℓ_1-norm of the data, and are what we might have naïvely expected having seen the times for the continuous input. We consistently find that the actual realised times are significantly slower than the empirically expected times, demonstrating

the hindrances caused from conditional branching on vector units.

```
#define VECTOR_ELEMENTS  // Number of elements in a vector.

unsigned int n_simd;     // Number of SIMD operations required.

for (unsigned int b = 0; b < n_simd; b++)
{
    register bool tails_encountered = false;

    #pragma omp simd reduction(||:tails_encountered)
    for (unsigned int i = 0; i < VECTOR_ELEMENTS; i++)
    {
        /* Perform central region calculation. */
        bool is_tail_value = ... ;
        tails_encountered = is_tail_value;
    }

    if (tails_encountered)
    {
        #pragma omp simd
        for (unsigned int i = 0; i < VECTOR_ELEMENTS; i++)
        {
            /* Handle tail or edge cases if required. */
        }
    }
}
```

Code 4.1.1 – A simplified code segment sketching how edge cases were handled in the GNU GSL (SIMD) implementation.

The alternative ACCUM implementation to this requires a more involved approach which looks to accumulate these divergent cases, keeping track of the input value and the output location. When enough tail values have been encountered that these can be computed in a vectorised fashion, then the necessary computation is performed. This was the "ACCUM" method.

We should lastly comment on the result from the adaptation of the `erfinv` function produced by Giles [77] (2011), originally designed for GPU hardware. The average performance measured for this routine is 13 clock cycles. We can see then this is the most competitive freely available out of the box method, and its initial GPU design is well adapted to the wide Intel AVX-512 vector hardware. We also include a second variant of this which only performs the central calculation, and bypasses checks on the validity of the input, and does not compute the tail regions, averaging 10.5 clock cycles. While this latter variant is not up to the same robustness level as the other variants, it does demonstrate the overhead derived from having to determine if error handling or tail calculations are required.

4.1.2.2 ThunderX2

Similar to the results for the Intel machine, we can asses the performance for the same code when run on an Arm-based ThunderX2. For these results the code required a small amount of refactoring. Notably, any Intel and NAG functionality was removed and replaced with GNU GSL where required. Furthermore, to achieve vectorisation it was necessary to remove the `isnan` function from the code bases. This meant that these implementations did not perform this step and so not only had slightly reduced functionality, they also have been written using possibly less instructions.

To make a comparison between the Arm and Intel results, we considered two cases

when compiling the implementations. The first was when we maintained the assertion at compile time that the SIMD vector lengths were 512-bit long[1], and the second was having this modified to 128-bit, where these represent the natural vector lengths of the Intel AVX-512 architecture and the Arm Neon architectures respectively. Similarly, we tried where possible to replicate the compilation flags between the two compilers, in this case using `armclang` 18.2. The results for the 512-bit and 128-bit compilations are shown in Figures 4.1.3 and 4.1.4 respectively.

Comparing the performances between the two $\Phi^{-1}(\cdot)$ implementations across the Intel and Arm hardware, as shown in Figures 4.1.2 and 4.1.3, we can make several observations. Our first comment is that the performance of the scalar implementations from GNU and Cephes remain at a very similar level, both averaging at around 100 clock cycles. However, notably between these two we see that which has the better average time is hardware dependent. Importantly though we do see an improvement in the central region when vectorising the GNU implementations with `armclang`. The improvement is a factor of approximately 4.5. We would expect a factor of 2 improvement from using the 128-bit Neon vectors. However, the scalar and vector units are the same on the ThunderX2, and so we suspect this extra improvement may possibly be due to either of two reasons, or a combination thereof. The first could be the manual masking operations introduced to the vector implementations for handling edge cases. The second could be the overhead of the function call, with the scalar version requiring multiple function calls, whereas the vectorised version is only called once.

Between the two different GNU implementations, the tail regions initially could not be vectorised as these contained the `isnan` checks and edge cases which the compiler could not vectorise. However, we can see the difference between the basic SIMD implementation and the accumulating variant, with the later performing noticeably much closer to its empirically expected value. This again demonstrates the significance of divergent behaviour, even for small degrees of vectorisation.

We can see that the performance of the higher order polynomial method introduced by Giles [77] is not performing particularly well on the Arm Neon hardware. This should not be too surprising, as it always takes the moderately expensive logarithm operation, and is designed to recover good performance from high degrees of parallel execution. However, given only 2 elements fit inside the Neon vectors (for 64-bit double-precision), it is not surprising that this is not as competitive against the scalar algorithms.

It remains to compare the performance when the implementations were compiled for their native 128-bit vector lengths, which are shown in Figure 4.1.4. The difference between this and the 512-bit compilation is that for an Arm Neon vector the former vector length would require no unrolling factor. We can see from Figure 4.1.4 that the scalar operations have remained the same, as expected. As for the central region in the vectorised versions of the GNU implementations, we see a lesser benefit from the vectorisation, suggesting that the compiler or hardware was able to produce a better execution when the loop was unrolled. A similar remark holds between the two performances of the Giles [77] implementations. The original version "Giles (2011)" in this case has a worse performance, while the "Giles (Central)" variant, which has only an unconditional execution of the central approximation for all input values, remains the same. This again suggests that performance was being recovered from a degree of unrolling when compiled for 512-bit vectors.

Having seen the results from Intel and Arm hardware for a range of commercial and open source library vendors, we can see that the core hindrance to recovering the full speed improvements that vectorisation has to offer stems from the conditional branching when having to handle the central and tail regions separately. Hence, moving forward, we anticipate that approximations which have a homogenous calculation over the whole domain will circumvent this issue, and be the key to achieving the best performance. As we will see, our earlier piecewise

[1]This information was passed to the compiler by the OpenMP `simdlen` clause, explained later in Section 4.2.1.

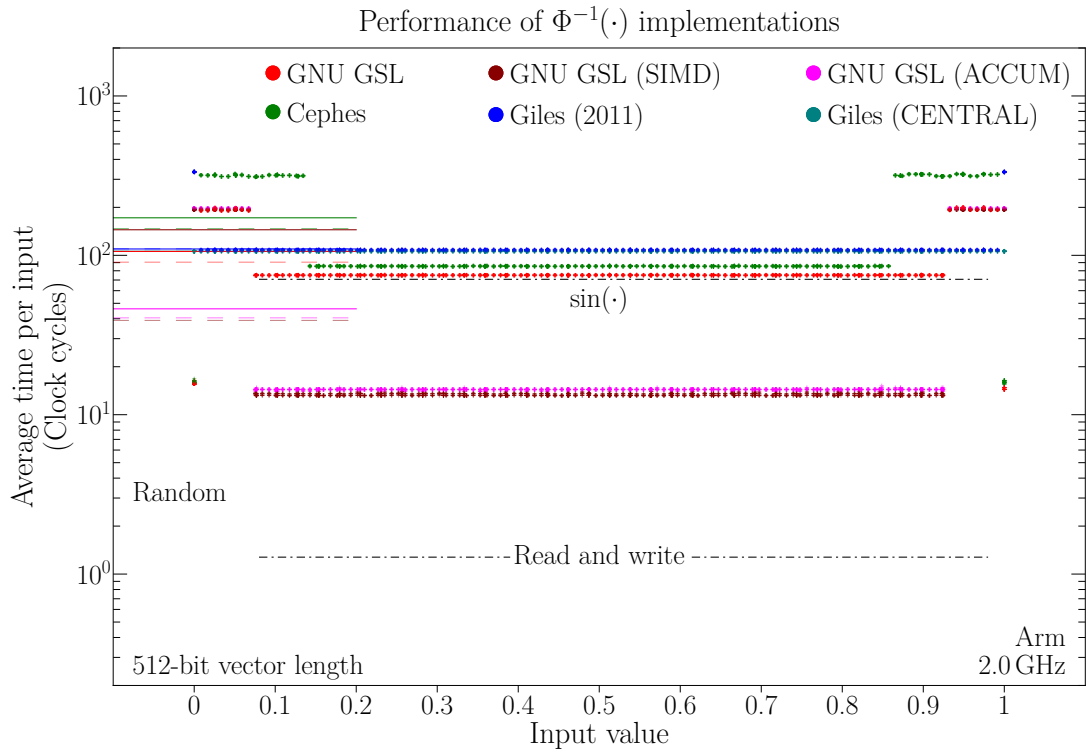


Figure 4.1.3 – A comparison of double-precision $\Phi^{-1}(\cdot)$ implementations on a ThunderX2 machine with 128-bit Neon vector units. These data were gathered using an AArch64 ThunderX2 operating at 2.0 GHz under GNU/Linux. C code was compiled using the Arm C compiler `armclang` 18.2 with the flags `std=c11`, `O2`, `fopenmp`, `fsimdmath`, `mcpu=native`, `funsafe-math-optimizations`, `ffp-contract=fast`, `target=aarch64-arm-none-eabi`, `insight`, and `g`. SIMD pragmas were compiled assuming 512-bit vector lengths.

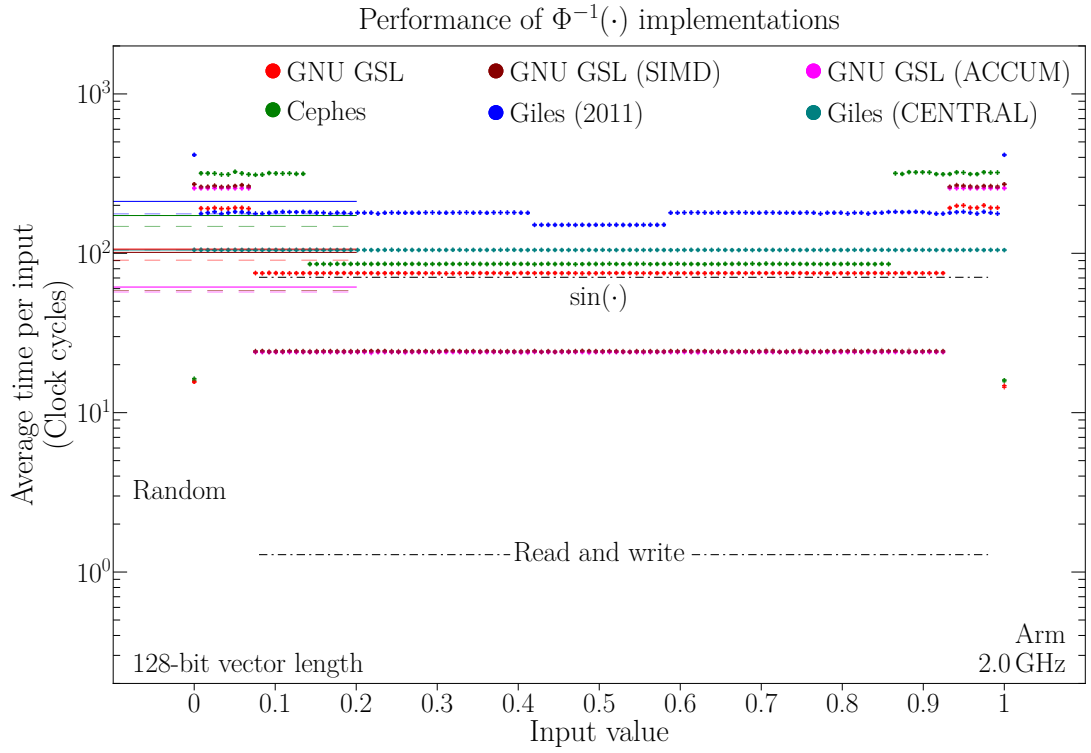


Figure 4.1.4 – The same as Figure 4.1.3, except SIMD pragmas were compiled assuming 128-bit vector lengths.

approximation schemes can both be implemented in such a homogenous manner.

4.2 Lookup tables

Lookup tables have been a long established method in science for facilitating fast calculations of complicated functions. These date back to trigonometric and logarithmic tables [42, 182], the earliest of which being a base-10 logarithm table from Briggs [32] (1617). Nonetheless, their use in computational settings is common, and one method for evaluating $\Phi^{-1}(\cdot)$ would be to use such a lookup table [134, 6.2.1]. The simple approach of such a method is to determine the nearest entry in the table that the input value corresponds to, and output the corresponding table entry. Use of such tables will be familiar to many scientists [212]. Furthermore, for those who may be sceptical about implementations using a lookup table, this is part of the approach adopted by Marsaglia et al. [150] as part of their routine for rapidly evaluating $\Phi^{-1}(\cdot)$.

Pursuing a lookup table for $\Phi^{-1}(\cdot)$, there are two typical ways we might want to construct the table with n entries (a.k.a. bins), and this is determined by how we choose to discretise the domain $[0, 1)$. There are two fundamental ways to partition the interval $[0, 1)$, the first being to have equally spaced partitions, and the second is to have some *user-specified* partitioning. We denote these partitionings "*equipartitioned*" and "*user-defined*" respectively. We would like to stress the equipartitioned case, and this is because if we construct a sorted lookup table, then searching the equipartitioned lookup table is an $\mathcal{O}(1)$ operation, whereas a more general user-specified partitioning is $\mathcal{O}(\log(n))$. One such method to search a sorted user-defined lookup table could be a bisection/binary search [134, page 407]. To map a uniform value $u \in [0, 1)$ to its entry in a equipartitioned table we simply multiply u by the number of bins and take the integer part (this requires *zero indexing*), as demonstrated in Code 4.2.1.

```
1 int n, idx;
2 double u, rv, lookup_table[n];
3 idx = (int) (n * u);  /* Decimal part is truncated away. */
4 rv = lookup_table[idx];
```

Code 4.2.1 – A simple example showing how to query an equipartitioned lookup table with `n` entries to evaluate the random variable `rv` $\approx \Phi^{-1}$(`u`).

We know from the body of work relating to approximations of $\Phi^{-1}(\cdot)$ that the most difficult regions to capture are those near the asymptotes at $\{0, 1\}$. With this in mind, if we only allow ourselves a modest number of bins, then it is not unreasonable to expect that a user-defined partitioning might like to place a higher density of bins in the neighbourhood of the asymptotes. As such, we considered 3 possible partitionings:

Equipartitioned Uniformly spaced bin intervals which can be queried in $\mathcal{O}(1)$ operations.
Uniform The same as equipartitioned bins, except because these are user-defined they require $\mathcal{O}(\log(n))$ operations to determine the value.
Geometric The widths of the intervals decay by powers of two approaching the asymptotes.

For clarity these are depicted in Figure 4.2.1. Although there are numerous ways a user can specify a partitioning, we anticipated that these two heuristics should be reasonably representative of a broad range of typical partitionings.

At this point we can start to map our earlier approximations to these implementations. Namely, the piecewise constant approximation corresponds to the equipartitioned lookup table. Similarly, the geometrically partitioned piecewise linear "resembles" the geometric partitioning. The exact implementation of the geometrically partitioned piecewise linear construction though will be developed later.

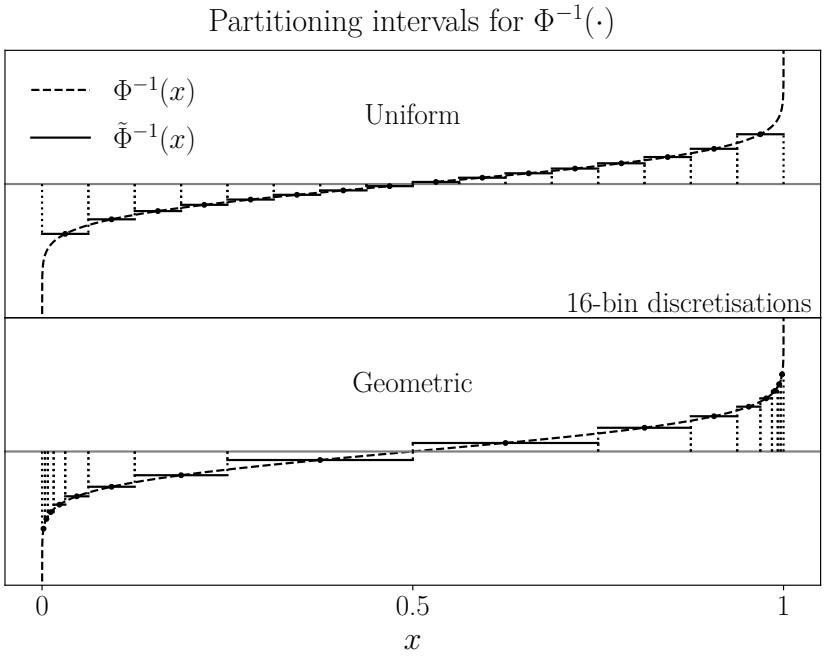


Figure 4.2.1 – Two possible different types of discretisation of $\Phi^{-1}(\cdot)$ for a sorted lookup table. In both cases we have set each bin's value to be the central value for the interval. **(Above)** Uniform partitioning of the domain. **(Below)** The geometric heuristic partitioning where the bins cluster around the asymptotes.

4.2.1 Vector length and loop unrolling

Ahead of presenting the performance times for lookup table implementations, we will very briefly need to mention vector lengths and `for` loop unrolling. Recalling Code 4.2.1, we might typically try to vectorise such a section of code with the OpenMP directive `#pragma omp simd`, as shown in Code 4.2.2. However, while the programmer might assume that the vector length `VEC_LEN` will be the length of the vector and the `for` loop will be executed in a single iteration, this may not be the case. Unless the vector length is explicitly stated using the `simdlen` directive as in Code 4.2.3 then "*an implementation-defined default value is assumed for the vector length*" [202, 5.2.2]. This effectively leaves the compiler free to choose what vector length it thinks is best, which need not be equal to `VEC_LEN`.

While it is not possible to hard code the vector length without the use of specific vector intrinsics, using the `simdlen` clause acts as a guide to the compiler. This is useful when the programmer may know *a priori* that a certain vector length should give optimal performance. An incorrect choice will not lead to incorrect results, but may impact performance [202, 5.2.2].

For the results we will shortly present (in Figure 4.2.4), we can compile the lookup table for either the equipartitioned tables or the user-defined partitionings with `simdlen` unspecified or specified. This is demonstrated for the equipartitioned lookup tables in Codes 4.2.2 and 4.2.3 respectively. Using the Intel C compiler `icc` 18.0.0 we can enable the generation of the optimisation reports using the `-qopt-report=5` (level 5 being the most detailed). We include the optimisation reports produced for the Intel Xeon Gold 6140 Skylake processor which is AVX-512 capable. Using 64-bit double-precision variables for the lookup table, this implies that the 512-bit vectors can hold up to 8 doubles. Inspecting the optimisation reports (reformatted for improved readability and key information emphasised) in Codes 4.2.4 and 4.2.5, corresponding to a `simdlen` being unspecified and specified respectively, we can see that the former leaves the compiler free to chose a vector length different from `VEC_LEN`. Notice that in Code 4.2.4 the compiler chose a shorter vector length and to unroll the `for` loop by a factor of 4, whereas in Code 4.2.5 the `for` loop was not unrolled (an unroll

```
#define VEC_LEN  /* User specified vector length. */

int n_simd_batches, n_bins;
double * restrict input;
double * restrict output;
double * restrict lookup_table;

for (int b = 0; b < n_simd_batches; b++)
{
    #pragma omp simd /* Implemenation defined default assumed. */
    for (int i = 0; i < VEC_LEN; i++)
    {
        output[i] = lookup_table[(int) (n_bins * input[i])];
    }
}
```

Code 4.2.2 – An example of decomposing a series of operations into SIMD operations. This leaves the compiler free to determine the appropriate vector length.

```
#define VEC_LEN  /* User specified vector length. */

int n_simd_batches, n_bins;
double * restrict input;
double * restrict output;
double * restrict lookup_table;

for (int b = 0; b < n_simd_batches; b++)
{
    #pragma omp simd simdlen(VEC_LEN) /* Explicitly guide compiler. */
    for (int i = 0; i < VEC_LEN; i++)
    {
        output[i] = lookup_table[(int) (n_bins * input[i])];
    }
}
```

Code 4.2.3 – An example of decomposing a series of operations into SIMD operations. This guides the compiler to use a vector length of `VEC_LEN`.

factor of 1) and was performed in a single iteration. The compiler estimates that the latter will give a speed up of 0.54, and hence will run slower, whereas the unrolled loop it estimates will have a speed up of 0.97.

Having seen the pessimistic speed up estimates provided by the compiler, it's natural to ask whether the compiler would choose to optimise this operation at all had it not been requested with the OpenMP directive? Comparing the two codes presented in Code 4.2.6 we can see that we can either rely on automatic vectorisation from the compiler, or explicitly declare the `for` loop as suitable for vectorisation using the OpenMP `simd` directive. In this case `icc` chooses to vectorise both of these, where the automatically vectorised case expects a speed up of 1.14 using a vector length of 2 and unroll factor of 4 (identical to Code 4.2.4).

In the results presented we provide the measured performance for the compiler's choice of vectorisation versus an explicitly requested vector length (in our examples this being `VEC_LEN` which is equal to 8).

It is also clear from Code 4.2.4 that the compiler is also concerned about the memory alignment. This level of implementation detail is excessive for our purposes and will ultimately only affect loop peeling instructions. For some introductory material to memory alignment we recommend Haase [96], and for loop peeling Larsen et al. [141, Section 4.3] [140, Section 3.2].

```
1  LOOP BEGIN at /path/to/lookup_table.c
2      remark: (vectorisation support)
3          reference output[i] has unaligned access
4          reference input[i] has unaligned access
5          unaligned access used inside loop body
6          conversion from float to int will be emulated
7          irregularly indexed load was emulated for <lookup_table[:]>
8              part of index is read from memory
9      remark:
10         vector length 2
11         loop was completely unrolled
12         unroll factor set to 4
13         normalized vectorization overhead 0.070
14     remark:
15         OpenMP SIMD LOOP WAS VECTORIZED
16     remark:
17         unmasked unaligned unit stride loads: 1
18         unmasked unaligned unit stride stores: 1
19         unmasked indexed (or gather) loads: 1
20     --- begin vector cost summary ---
21         scalar cost: 13
22         vector cost: 12.500
23         estimated potential speedup: 0.970
24         type converts: 1
25     --- end vector cost summary ---
26 LOOP END
```

Code 4.2.4 – The Intel compiler report for Code 4.2.2.

```
1  LOOP BEGIN at /path/to/lookup_table.c
2      remark:
3          ...
4          vector length 8
5          loop was completely unrolled
6          normalized vectorization overhead 0.038
7          ...
8      --- begin vector cost summary ---
9          scalar cost: 13
10         vector cost: 23.120
11         estimated potential speedup: 0.540
12         type converts: 1
13     --- end vector cost summary ---
14 LOOP END
```

Code 4.2.5 – The Intel compiler report for Code 4.2.3.

4.2.2 Competing for shared L_3 cache and DRAM memory

The performance times for the lookup tables are best shown in several stages to highlight some key performance features observed. The first of these which we should highlight is the performance degradation seen when the lookup table spills into shared memory. The performance times showing this degradation are in Figure 4.2.2.

Looking at Figure 4.2.2 there is one key feature we would like to draw the reader's attention to, which is the linear increase in the execution time once the lookup table needs to exit the L_2 cache. For the user-defined uniformly distributed partitioning, when random data is queried on each iteration of the `for` loop, then any entry in the lookup table is as likely as any other. Hence we can expect to explore the entire table, in which case upon each search we cannot expect the entry to be in the private L_2 cache, and hence we must search the shared

```
int n_items;  // Much larger than vector length.

for (int i = 0; i < n_items; i++)
{
    output[i] = lookup_table[(int) (n_bins * input[i])];
}
```

```
int n_items;  // Much larger than vector length.

#pragma omp simd
for (int i = 0; i < n_items; i++)
{
    output[i] = lookup_table[(int) (n_bins * input[i])];
}
```

Code 4.2.6 – Two possible versions for using a lookup table. **(Above)** Automatic vectorisation by the compiler. **(Below)** Explicit request for vectorisation using the OpenMP `simd` directive.

memory (either the L_3 cache or RAM). Searching the shared memory is much more expensive, and this is made clear by the much larger average times when compared to the other partitioning methods.

Furthermore, because the memory is shared, each core will be querying the shared memory. We can see that as we increase the number of cores which are concurrently executing a lookup table search, then the more cores which are active, the more traffic the shared memory encounters, and the slower a query is. We demonstrate this by running the lookup table searches with either a single core running on the processor, up to 10 or 30 cores concurrently running. While these cores are running we ensured the processor was otherwise idle with no other processes running. We can see then that as we increase the number of concurrent processes having to compete to access the shared memory, the lookup time increases in a related fashion. Additionally, the variability in the performance also increases, shown by the increasing variation seen in the average times.

The overarching result we can take from Figure 4.2.2 is that for a lookup table to be a feasible alternative to a function evaluation, the table must be small enough to fit within the private L_1 or L_2 caches.

4.2.3 User-specified partitionings

Provided we restrict our considerations to lookup tables which we can fit in private memory, it remains then to inspect the performance of having to perform a search through a user-specified partitioning against an equipartitioned table. If we zoom in on Figure 4.2.2 then we obtain Figure 4.2.3.

From Figure 4.2.3 we can see the performance difference between the $\mathcal{O}(\log(n))$ time required for the bisection search of a user-defined partitioning and the $\mathcal{O}(1)$ time for an equipartitioned table. Strikingly we can see that the scaling of the equipartitioned table is exceptionally good. Meanwhile, if we wanted to use some user-specified partitioning then these would only be competitive methods to function evaluations which took an average of approximately 100 clock cycles or more. Hence, given that we have been measuring the evaluation of $\Phi^{-1}(\cdot)$ to be in the region of 10 clock cycles, in this circumstance user-defined bins are not competitive. However, this procedure should be considered for possibly more involved calculations or distributions other than $\Phi^{-1}(\cdot)$.

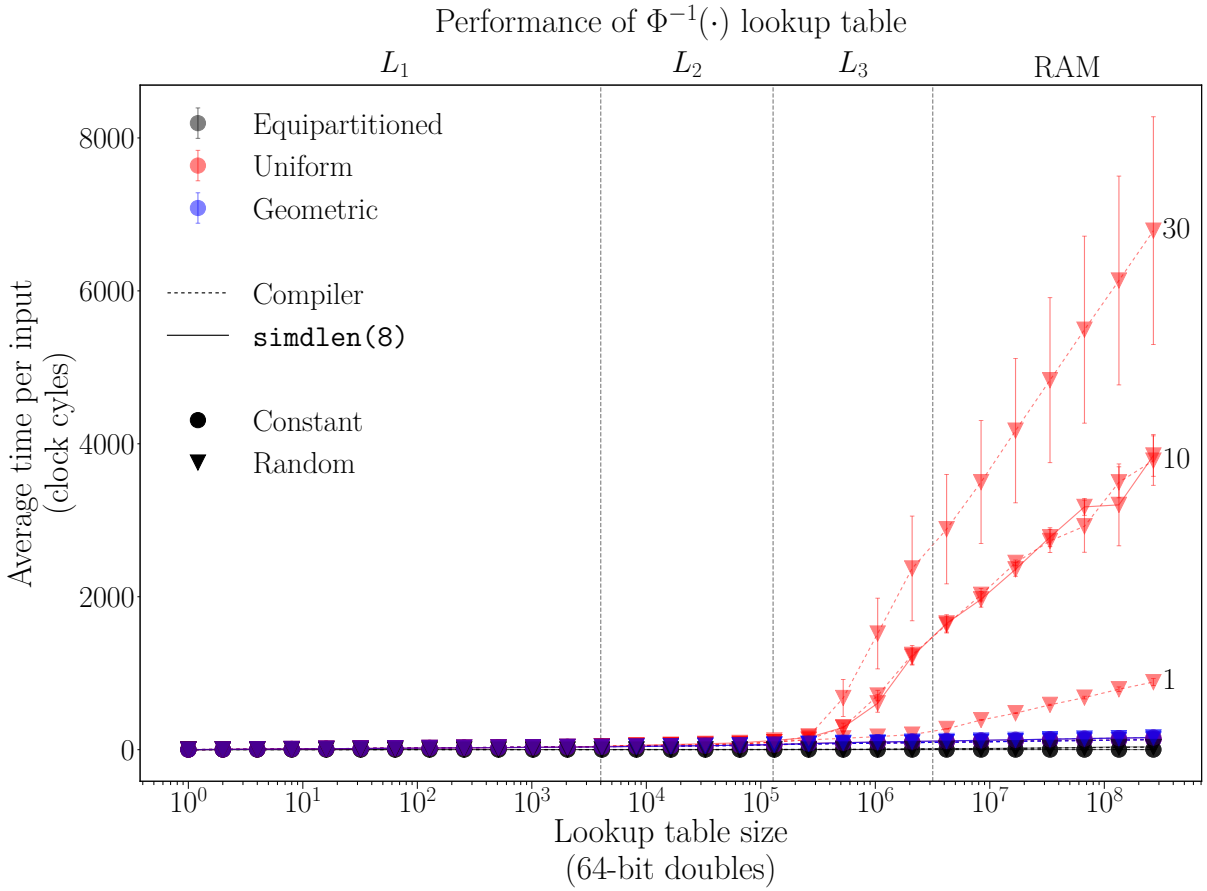


Figure 4.2.2 – The performance of a dynamically allocated lookup table. The different colours represent the different type of bin partitionings. The line styles represent whether the vector length was chosen by the compiler or explicitly declared using the `simdlen` clause. The triangular markers represent when the input values queried were randomly distributed, and the circular markers when a constant value was queried repeatedly. We indicate the different cache boundaries for various lookup table sizes when working with 64-bit double-precision entries. Lastly, we also label the results with the number of concurrent cores running.

Interestingly, we can see that the equipartitioned table transitions from $\mathcal{O}(1)$ to $\mathcal{O}(\log(n))$ once it spills into shared memory. This demonstrates the hardware requiring an $\mathcal{O}(\log(n))$ routine to find and access the memory address in RAM.

4.2.4 Equipartitioned tables

To appreciate the performance of the equipartitioned table in comparison to the user-specified partitionings we can present the timings on a log-log scale, as in Figure 4.2.4. We can lastly see from Figure 4.2.4 that the equipartitioned lookup table takes between 1–2 clock cycles on average when the table can be held in the private memory of the L_1 or L_2 caches. Furthermore, the lookup times when held in the L_1 cache appear constant. When inside the L_1 cache there is a performance difference between the tables compiled with the compiler's choice of vector length, and then when the vector length is explicitly requested, where the latter is the faster. It is interesting to note that explicitly specifying `simdlen` gives a superior performance, contrary to what the compiler previously anticipated. This of course demonstrates that compiler reports are not infallible, and that ultimately any conclusions concerning performance should only be made by assessing real experimental data.

Considering the size of the L_2 cache is 1024 kB (cf. Table 2.1.1), then if we were to allocate around half of this space for a lookup table then this could fit 64 000 (64-bit) double-

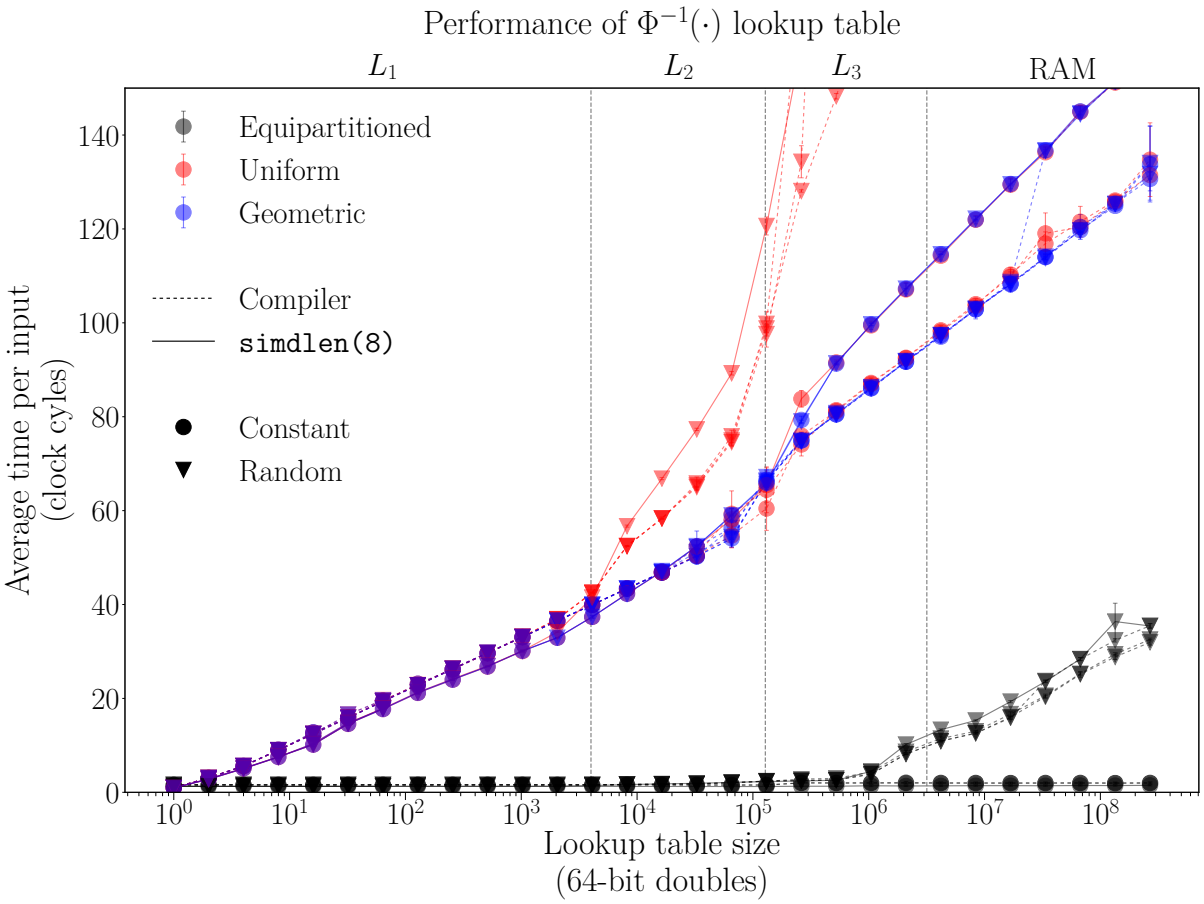


Figure 4.2.3 – The performance difference between having to perform a bisection search through a sorted user-specified partitioning compared to an equipartitioned table. (cf. Figure 4.2.2).

precision values. If we then map this into a table corresponding to a selection of random bits, then this could store all possible 16-bit random numbers ($2^{16} \approx 65\,500$). With a view to the possibility of half-precision random numbers, we can see such a small table may be sufficient. (We recall that reduced size uniform random numbers truncated to a limited number of bits has been modelled previously by Giles et al. [90]).

As for the $\Phi^{-1}(\cdot)$ implementations, we can compare the performance when using an Arm-based ThunderX2 machine, with the performance as shown in Figure 4.2.5. Comparing the performance to that achieved using Intel hardware and compilation, we can see that overall this tells a very similar story. The key message is again that an equipartitioned lookup table remains the only realistically feasible option. In this case the average lookup time being approximately 4 clock cycles, which, while competitive, is slower than the speeds measured from the Intel hardware.

4.3 Piecewise polynomial approximations

A natural extension to using a piecewise constant approximation is a piecewise polynomial approximation, and to cluster intervals near the singularities.

4.3.1 Dyadic partitioning

Based on the empirical timings from Sections 4.2.3 to 4.2.4, we saw that equipartitioned tables displayed a considerable performance advantage over generalised

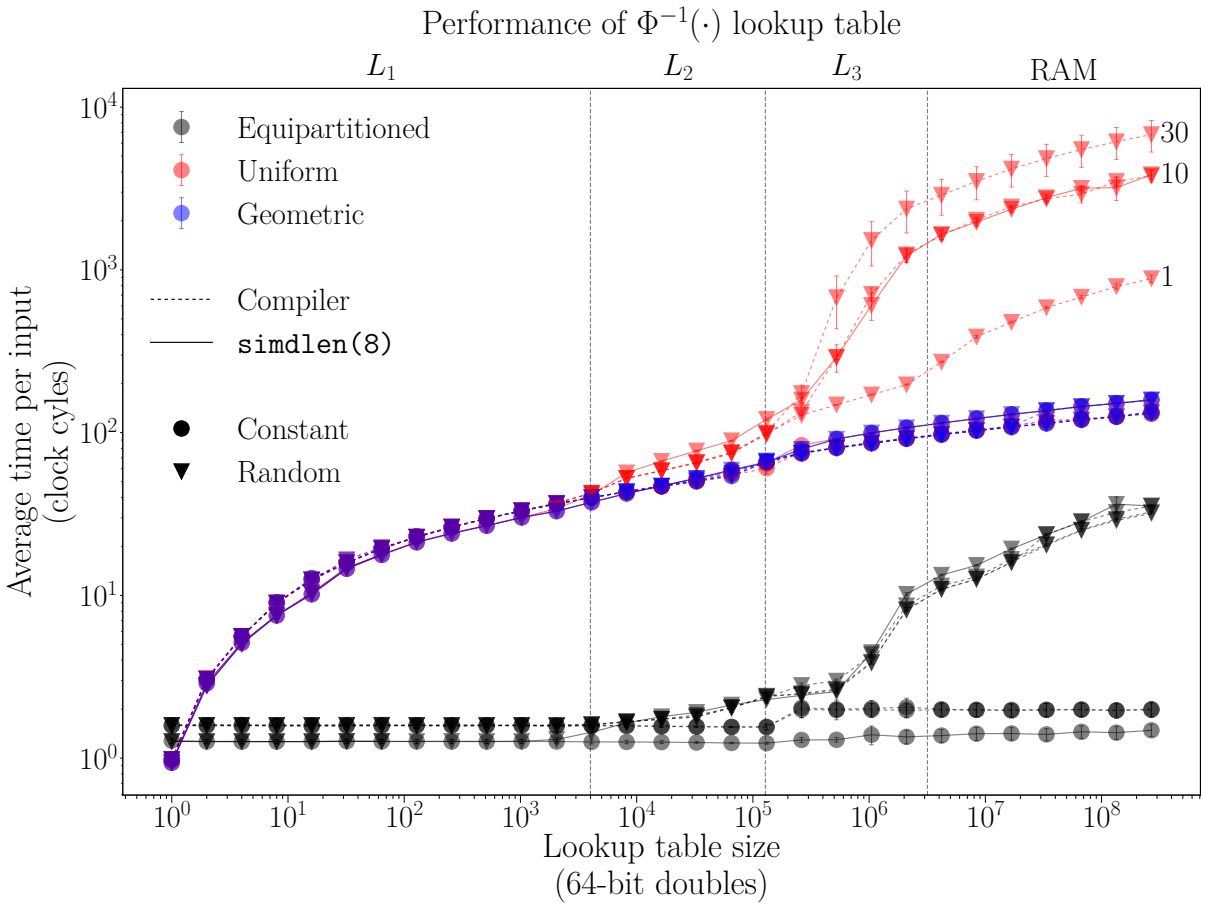


Figure 4.2.4 – The performance times for the lookup tables on a log-log scale.

user-specified partitions. Alas, if we recall the approximation schemes from Section 4.1, we noted that most of $\Phi^{-1}(\cdot)$ was easily approximated by polynomials and rational approximations, but that these struggled near the asymptotes. Motivated by these considerations, it seems desirable to see if it is possible to construct a partitioning which can have a reasonably fast $\mathcal{O}(1)$ lookup time, but where the partitions are denser at the asymptotes.

One solution to this problem is to use dyadic partitions. The explanation is that the floating-point representation stores the base 2 exponent of a number, which can readily be mapped to a table index in $\mathcal{O}(1)$ time. To make this clear, in floating-point representation a value x is represented as $x \equiv \pm(1.a_1a_2\cdots a_m\cdots)_2 \times 2^{(b_1b_2\cdots b_e)_2}$. We can read off the exponent's value, correct for the exponent bias appropriately, and map the dyadic intervals to array indices. Consider a 32-bit single-precision float, which has 1 sign bit, 23 bits for the mantissa, and 8 bits for the exponent. First we treat the float in memory as if it were an unsigned integer by pointer aliasing[1] (a.k.a. type-punning). We can then perform a bitwise-AND to keep only the exponent bits, and then bit-shift to the right by 23 bits. The exponent though is biased by 127 ($= 2^{8-1}-1$), so if we subtract away the exponent from 126 ($= 127-1$) we produce an array index. The array indices produced by this procedure for the various dyadic intervals are enumerated in Table 4.3.1. Given the symmetry of $\Phi^{-1}(\cdot)$ we only need to produce an approximation in the range $(0, \frac{1}{2}]$. An overview of this dyadic piecewise polynomial approximation is given in Algorithm 4.3.1.

A benefit to this approach is that it allows us to ensure the uncovered region is an exponentially small fraction of the domain as we increase the maximum permissible array index. This means that for any reasonable portion of $(0, 1)$ we only need a small size of array, which is

[1] C and C++ have a strict aliasing rule concerning type-punning [120, 6.5.2.3] [121] [192, pages 163–164] [137] [53], so pointer aliasing in this fashion is not strictly language compliant and can result in undefined behaviour.

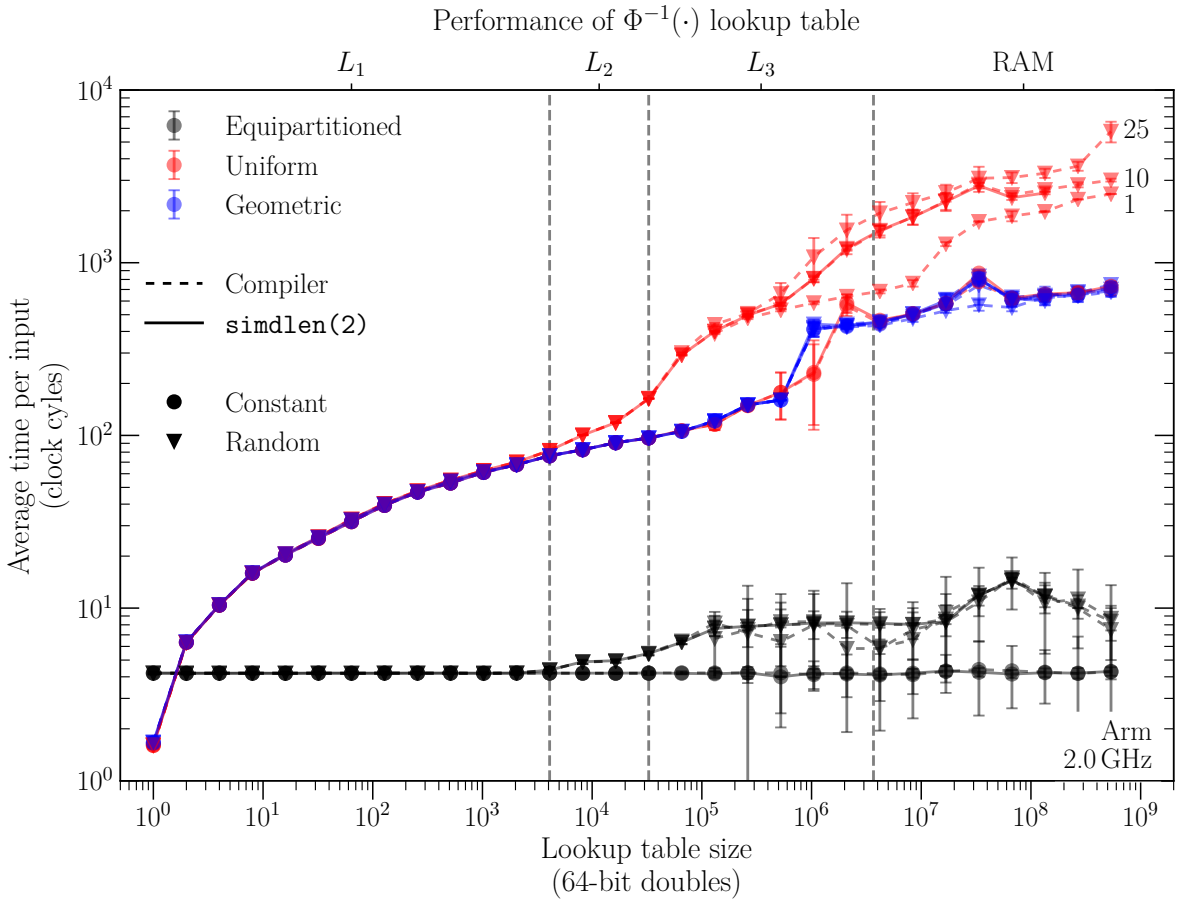


Figure 4.2.5 – The performance times for the lookup tables on a ThunderX2 machine.

logarithmically proportional to the size of the uncovered region. Whereas before we had aimed to store the lookup tables in the L_1 or L_2 caches, we will see in Section 4.3.3.2 that these dyadic lookup tables can be small enough such that they fit within vector registers, and consequently reap the increased speeds as highlighted in Table 2.1.1.

To illustrate this construction, we show the dyadic intervals and the corresponding piecewise linear approximation to $\Phi^{-1}(\cdot)$ in Figure 4.3.1. Notice the similarity between this and the geometric partitioning proposed in Section 4.2 and sketched in Figure 4.2.1.

As we will be making extensive use of this geometrically partitioned piecewise polynomial approximation specifically using dyadic intervals, for brevity we will regularly denote this just as the *dyadic approximation*.

We mentioned earlier that Intel had a very small tail region for its implementation

Dyadic interval	Array index
$[\frac{1}{2}, 1)$	0
$[\frac{1}{4}, \frac{1}{2})$	1
$[\frac{1}{8}, \frac{1}{4})$	2
$\vdots$	$\vdots$
$[\frac{1}{2^{n+1}}, \frac{1}{2^n})$	n

Table 4.3.1 – The array indices (using zero indexing) produced for various dyadic intervals appropriate for approximating $\Phi^{-1}(\cdot)$.

Input: Floating-point uniform random number $U \in [0, 1)$.
Output: Floating-point approximate Gaussian random number $\tilde{Z}$.
1 Form predicate based on whether $U > \frac{1}{2}$.
2 Reflect about $\frac{1}{2}$ to obtain $U \in [0, \frac{1}{2}]$.
3 Alias U as an unsigned integer.
4 Read the exponent bits using bitwise-AND.
5 Right shift away the mantissa's bits.
6 Obtain an array index by correcting for exponent bias.
7 Cap the array index to avoid overflow.
8 Read the polynomial coefficients $\{a_i\}_{i=0}^{n}$ using the array index.
9 Re-interpret U as a float.
10 Form the polynomial approximation $\tilde{Z} = \sum_{i=0}^{n} a_i U^i$.
11 Correct sign of approximation based on predicate.

Algorithm 4.3.1 – Piecewise polynomial approximation using dyadic intervals.

of $\Phi^{-1}(\cdot)$. At this point it is possible to speculate about what Intel is doing. Given the presentations by Cornea [49, 50] and the provided example implementation of the reciprocal `RCP14S` in `RECIP14.c` [50], this uses a 64-bin lookup table in a similar fashion to what we have suggested, using a piecewise linear construction. Furthermore, they too compute the size of the exponent (albeit they suggest doing so using a `while` loop). Lastly, Cornea [49, page 8] comment that vector permute instructions are possible for small enough lookup tables. Combining all of this information, and the very small tail region that Intel demonstrated, it is entirely possible to speculate that their implementation might be based on a similar design to ours.

4.3.2 Fitting the polynomial coefficients

One of the questions which may trouble someone wishing to implement this method or a similar extension, is how to find the approximating polynomial's coefficients? Specifically, we wish to find the optimal coefficients in the L^2 sense, whereas it is more common to focus on the L^∞-error in many numerical library routines. To find the coefficients, we write the polynomial approximation as $\sum_{i=0}^{n} a_i u^i \approx \Phi^{-1}(u)$, and form the L^2-error $\int_a^b (\Phi^{-1}(u) - \sum_{i=0}^{n} a_i u^i)^2 \,\mathrm{d}u$. Taking the derivative of this error with respect to a_j and setting this to zero, the optimal coefficients are the solution to

$$\sum_{i=0}^{n} a_i \left(\frac{b^{i+j+1} - a^{i+j+1}}{i+j+1} \right) = \int_a^b \Phi^{-1}(u) u^j \,\mathrm{d}u \equiv \int_{\Phi^{-1}(a)}^{\Phi^{-1}(b)} z \Phi^j(z) \phi(z) \,\mathrm{d}z \tag{4.3.1}$$

for $j \in \{0, 1, \ldots, n\}$. Either of the integral expressions are readily evaluated by most software, with the latter sometimes being preferable in the singular interval where $a = 0$. This forms a set of simultaneous linear equations which are readily solved for each interval.

4.3.3 Piecewise polynomials of varying orders

With the optimal coefficients found, the question then remains as to what size polynomial we require? For a comparison to the previous lookup tables using 1024 entries, we note from Figure 3.2.3 that these have an L^2-error (a.k.a. the RMSE) of approximately 10^{-2}. Hence for comparison we will aim for a RMSE that is at least similar in size or smaller.

Ahead of choosing the polynomial order, the domain of the approximation needs to be addressed, and we need to set the size of the lookup table. If we are targeting table sizes that will fit inside vector registers, then it is appropriate to move to 32-bit single-precision floats

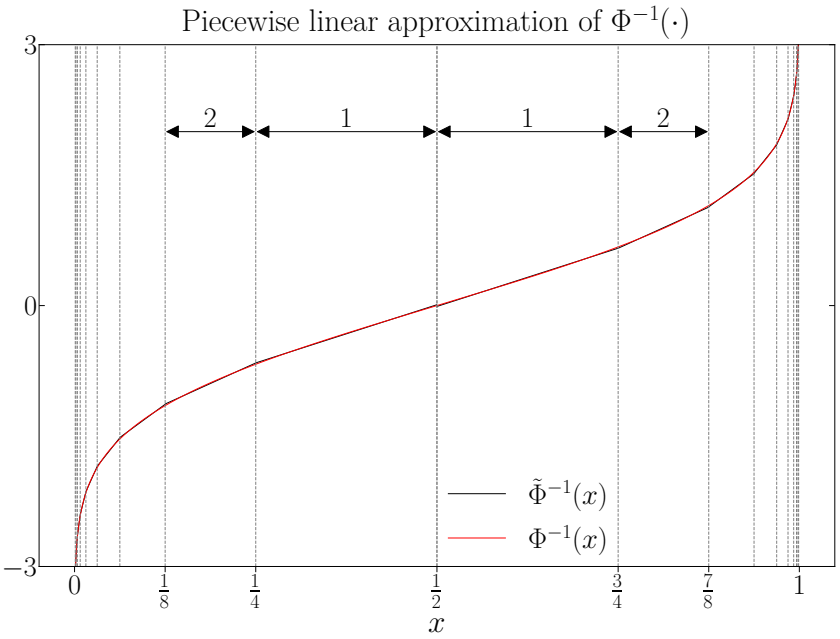


Figure 4.3.1 – Piecewise linear approximation of $\Phi^{-1}(\cdot)$ using dyadic intervals. The dyadic intervals and the corresponding array indices (labelled) correspond to those listed in Table 4.3.1. By symmetry the value 0.5 is reserved for the zeroth index of the array.

so twice the number of entries can fit inside the table. Furthermore, for comparisons with the latest Intel hardware, we will use the working assumption that our vector registers are 512-bit vectors. We will indicate when the vectors differ from 512-bit.

Proceeding to use a table which holds 16 entries for each of the coefficients of a polynomial, we can inspect the values for the RMSE that different sized polynomials produce. The RMSE values obtained are shown in Figure 4.3.2, where we also show results for tables with fewer than 16 entries.

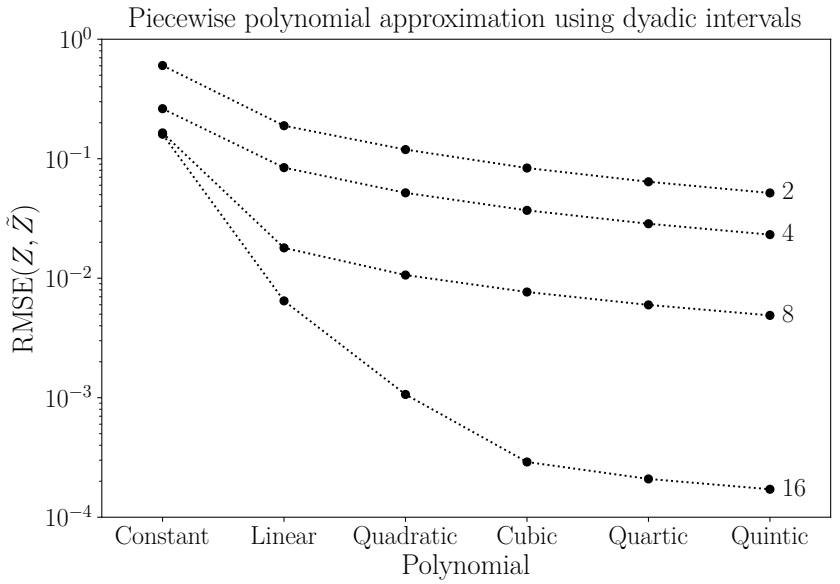


Figure 4.3.2 – The RMSE for piecewise polynomial approximations of $\Phi^{-1}(\cdot)$ using dyadic intervals, with the maximum number of intervals indicated.

Given we have a target RMSE of 10^{-2} we can see from Figure 4.3.2 that for a table with 16 entries we require at least a piecewise linear approximation, and that going to a quadratic or cubic would increase this by approximately one or two orders of magnitude respectively. For

the smaller table with 8 entries we see that there is not a huge difference between linear and cubic polynomials. This indicates that for tables with 8 or fewer entries, the dominant contribution to the RMSE is coming from the final singular interval for these modest polynomial sizes.

It is informative to remark on the appreciable reduction in the RMSE between the piecewise constant and linear approximations, especially for tables with 8 or more entries. This can be explained by observing the vastly improved fidelity of a linear approximation in the large central regions by comparing Figures 3.2.1 and 3.3.2, and noting that evidently the major contribution to the RMSE is from the central regions.

Given the sketch of the procedure outlined in Algorithm 4.3.1, we anticipate the bulk of the instructions and execution will likely be in the reading and writing to memory, and the intermediate integer manipulations. As such, the difference of incorporating in 2 additional FMAs to go from a linear piecewise polynomial to a cubic is likely minor compared to the gain in the RMSE. As such we will implement and profile both. The implementation of the slightly higher order cubic approximation can be readily simplified to a linear approximation, as we will see, and hence we will typically present the implementations of the piecewise cubic.

4.3.3.1 OpenMP SIMD implementation

It is possible to implement Algorithm 4.3.1 in a relatively short amount of C code, where we demonstrate one implementation in Code 4.3.1 by the function `vs_inverse_gaussian_cdf_logarithmic_polynomial_approx`[2].

4.3.3.2 Register implementation

As we mentioned, the appeal of having such small arrays of coefficients is that they should be able to fit inside vector registers, where arrays of 16 coefficients storing 32-bit single-precision floats could each fit inside a 512-bit vector register.

Having such knowledge is ideal if we were targeting a known vector length, such as if we were optimising for an Intel AVX-512 core. However, knowledge of the vector lengths is contrary to any vector length agnostic (VLA) style of programming, which is the natural style the Arm scalable vector extension (SVE) promotes, as does a more general and simplistic SIMD style of programming. As such, it seems the notion of utilising this information of vector lengths is at odds with VLA code design.

4.3.3.3 Arm SVE implementation

We can take the implementation from Code 4.3.1 and see how well the compiler performs auto-vectorisation when asked to issue VLA code and SVE instructions. For this we use the `armclang` 19.1 compiler which produced the assembly presented in Code 4.3.2, and deciphered in Table 4.3.2. We have indicated the general SVE load instructions `ldw1` which load in the coefficients from memory, and hence a general VLA implementation is unable to place the arrays of coefficients in vector registers, but instead reads them repeatedly from the L_1 cache.

Having the compiler issue SVE instructions while trying to enforce a minimum vector size appear to be two mutually exclusive goals, and we were unable to influence the compiler into issuing any table lookup operations from the vector registers. This meant that to achieve this we had to turn to using inline assembly to issue the instructions we wanted. In our case this was the `tbl` instruction which performs a gather operation from a vector register. The

[2]The name of this function is largely self-explanatory, albeit the `vs` denotes vectorised single-precision, and the `logarithmic` denotes that the intervals are equally spaced on a logarithmic scale.

```c
#include <stdlib.h>
#include <stdbool.h>

#define N_MANTISSA_32 23 // For IEEE-754 format
#define FLOAT32_EXPONENT_BIAS_TABLE_OFFSET (127 - 1)
#define TABLE_MAX_INDEX (16 - 1) // Zero indexing
#define FLOAT32_AS_UINT32(x) (*((unsigned int *) &x)) // For pointer aliasing

const float a_0[16] = {...}; // The polynomial coefficients
const float a_1[16] = {...};
const float a_2[16] = {...};
const float a_3[16] = {...};

#pragma omp declare simd
static inline float polynomial_approximation(float u, unsigned int index)
{   /* Polynomial evaluation using Horner's rule. */
    return a_0[index] + u * (a_1[index] + u * (a_2[index] + u * a_3[index]));
}

#pragma omp declare simd
static inline unsigned int get_table_index_from_float(float u)
{   /* Obtaining the array index from the uniform float. */
    unsigned int index = FLOAT32_AS_UINT32(u) >> N_MANTISSA_32;
    index = FLOAT32_EXPONENT_BIAS_TABLE_OFFSET - index;
    return index > TABLE_MAX_INDEX ? TABLE_MAX_INDEX : index;
}

void vs_inverse_gaussian_cdf_logarithmic_polynomial_approx(
    unsigned int N,
    const float * restrict input,
    float * restrict output)
{   /* Converting the uniform to the approximate Gaussian random variable. */
    #pragma omp simd
    for (unsigned int i = 0; i < N; i++)
    {
        float u, z;  // Uniform and Gaussian random variables.
        u = input[i];
        bool predicate = u < 0.5f;
        u = predicate ? u : 1.0f - u;
        unsigned int index = get_table_index_from_float(u);
        z = polynomial_approximation(u, index);
        z = predicate ? z : -z;
        output[i] = z;
    }
}
```

Code 4.3.1 – A simple OpenMP implementation of a piecewise cubic approximation to $\Phi^{-1}(\cdot)$ using dyadic intervals, following Algorithm 4.3.1.

implementation of this using SVE instructions is shown in Code 4.3.3, where we have highlighted the `tbl` instructions, while much of the body of the implementation originates from Code 4.3.2.

At the time of writing, Code 4.3.3 was profiled on prototype Arm SVE capable hardware. However, the results are confidential and cannot be disclosed in this thesis. Consequently, we will shortly show similar implementations for Intel hardware, but will be unable to provide real experimental timings for Arm SVE hardware.

```
<vs_inverse_gaussian_cdf_logarithmic_polynomial_approx>:
 0: cbz w0, dc      <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0xdc>
 4: mov w9, w0
 8: adrp    x10, 0  <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx>
 c: adrp    x11, 40 <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x40>
10: adrp    x12, 20 <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x20>
14: adrp    x13, 60 <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x60>
18: mov     x8, xzr
1c: fmov    z0.s, #5.000000000000000000e-01
20: ptrue   p0.s
24: fmov    z1.s, #1.000000000000000000e+00
28: whilelo p1.s, xzr, x9
2c: mov     z2.s, #126
30: mov     z3.s, #7
34: add     x10, x10, #0x0
38: add     x11, x11, #0x0
3c: add     x12, x12, #0x0
40: add     x13, x13, #0x0
44: b       80      <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x80>
48: nop
...
7c: nop
80: ld1w    {z4.s}, p1/z, [x1, x8, lsl #2]
84: fsub    z5.s, z1.s, z4.s
88: fcmgt   p2.s, p0/z, z0.s, z4.s
8c: sel     z4.s, p2, z4.s, z5.s
90: lsr     z5.s, z4.s, #23
94: sub     z5.s, z2.s, z5.s
98: cmplo   p3.s, p0/z, z5.s, #7
9c: fmul    z6.s, z4.s, z4.s
a0: sel     z5.s, p3, z5.s, z3.s
a4: lsl     z5.s, z5.s, #2
a8: ld1w    {z7.s}, p1/z, [x10, z5.s, uxtw]
ac: ld1w    {z16.s}, p1/z, [x11, z5.s, uxtw]
b0: fmla    z7.s, p0/m, z16.s, z6.s
b4: ld1w    {z16.s}, p1/z, [x12, z5.s, uxtw]
b8: ld1w    {z5.s}, p1/z, [x13, z5.s, uxtw]
bc: fmad    z5.s, p0/m, z6.s, z16.s
c0: fmad    z4.s, p0/m, z5.s, z7.s
c4: fneg    z5.s, p0/m, z4.s
c8: sel     z4.s, p2, z4.s, z5.s
cc: st1w    {z4.s}, p1, [x2, x8, lsl #2]
d0: incw    x8
d4: whilelo p1.s, x8, x9
d8: b.mi    80      <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x80>
dc: ret
```

Code 4.3.2 – The assembly mnemonics produced from compiling Code 4.3.1 using `armclang` (19.1) with SVE enabled using the compiler flags `std=c11`, `O3`, `Wall`, `Wvector-operation-performance`, `Wopenmp-simd`, `armpl`, `fopenmp`, `fsimdmath`, `fvectorize`, `ffp-contract=fast`, `ftree-vectorize`, `funsafe-math-optimizations`, `ffinite-math-only`, `fno-trapping-math`, `fno-signaling-nans`, `march=armv8.4-a+sve`, and `mcpu=thunderx2t99`.

4.3.3.4 Intel intrinsics implementation

It remains to see if our envisioned benefits can be realised on actual hardware, and also compare the difficulty of using registers as lookup tables on hardware which has known vector lengths with the capacity to store the tables.

The first stage is to see how autovectorisation performs on the OpenMP implementation relying on judicious use of the OpenMP SIMD `pragma`. While the Intel

Line(s)	Description
`0`	If no iterations are required (`N` $= 0$), then skip to function return.
`4`	Move `N` to appropriate location for the `while` loop comparison.
`8-14`	Obtain address of coefficient arrays.
`18`	Initialise the loop counter to zero.
`1c`	Initialise constant of $\frac{1}{2}$.
`20`	Initialise an all true predication.
`24`	Initialise constant of 1.
`28`	Assess if the `while` loop's increment counter meets the condition.
`2c-30`	Initialise constants of 126 and 15.
`34-40`	Redundant.
`44`	Skipping forward to the main loop body which is padded with null operations (`nop`) to ensure the loop instructions are nicely aligned in the instruction cache's memory.
`48-7c`	Null operations.
`80`	Reading input values.
`84`	Computing $1 - U$ for possible use mapping U to a point in $[0, \frac{1}{2})$.
`88`	Compute predicate based on $U < \frac{1}{2}$.
`8c`	Perform the mapping of U such that $U \in [0, \frac{1}{2})$.
`90`	Remove the mantissa.
`94`	Correct for exponent bias.
`98`	Assess if the array index exceeds the array's length.
`9c`	Compute U^2.
`a0`	Truncate array indices larger than the array's length, avoiding overflow.
`a4`	Left bit shift the array index by 2 for correct memory offsets. Currently the array indices are single byte (8-bit) offsets, so bit shifting by two multiples these by 4 giving 32-bit offsets, corresponding to the length of a 32-bit single-precision float.
`a8-ac`	Load appropriate coefficients from memory (which will be in the L_1 cache).
`b0`	Compute even polynomial's contribution.
`b4-b8`	Load appropriate coefficients from memory.
`bc`	Compute odd polynomial's contribution.
`c0`	Combine the even and odd polynomial contributions to form $\tilde{Z}$.
`c4`	Compute $-\tilde{Z}$ in case it will be required.
`c8`	Negate $\tilde{Z}$ if required.
`cc`	Write the output values.
`d0`	Increment loop counter.
`d4`	Assess if the `while` loop's increment counter meets the condition.
`d8`	Checks if the condition state's first active element is true, such that there is at least one or more elements which require computing, and if so jump to the main body of the `while` loop at line `80`.
`dc`	Return from the function.

Table 4.3.2 – A deciphering of the assembly in Code 4.3.2.

compiler preferred shorter vector lengths, by adding in the `simdlen(16)` flag to the SIMD `pragma` we were able to have the compiler generate vectors holding 16 elements (each being a 32-bit single-precision `float`). Despite putting the constant arrays of coefficients in either a header file or the function's local variable scope, the compiler generated instructions where the arrays were loaded in from memory each time, (retrieved from the L_1 cache). An example of the assembly produced (using Intel assembly syntax) is shown in Code 4.3.4.

Fortunately, we can see from Code 4.3.4 that the Intel `icc` compiler issued the irregular memory load using a `vpermps` permutation instruction, as desired. However, the

```
void vs_inverse_gaussian_cdf_logarithmic_polynomial_approx(
    unsigned int N,
    const float * restrict input,
    float * restrict output)
{   /* Converting uniforms to approximate Gaussian random variables,
     * assuming vector lengths sufficient for coefficient arrays. */
    unsigned int i_dummy;  // A dummy variable.
    asm("\t"
        "cbz     %[N], 2f                                    \n\t"
        "mov     %[i], xzr                                   \n\t"
        "fmov    z0.s, #5.00000000000000000e-01              \n\t"
        "ptrue   p0.s                                        \n\t"
        "fmov    z1.s, #1.00000000000000000e+00              \n\t"
        "ld1w    {z2.s}, p0/z, [%[a_0], %[i], lsl #2]        \n\t"
        "ld1w    {z3.s}, p0/z, [%[a_1], %[i], lsl #2]        \n\t"
        "ld1w    {z4.s}, p0/z, [%[a_2], %[i], lsl #2]        \n\t"
        "ld1w    {z5.s}, p0/z, [%[a_3], %[i], lsl #2]        \n\t"
        "mov     z6.s, #126                                  \n\t"
        "mov     z7.s, %[table_max_index]                    \n\t"
        "whilelo p1.s, xzr, %[N]                               \n"
        "1:                                                  \n\t"
        "ld1w    {z8.s}, p1/z, [%[input], %[i], lsl #2]      \n\t"
        "fsub    z9.s, z1.s, z8.s                            \n\t"
        "fcmgt   p2.s, p0/z, z0.s, z8.s                      \n\t"
        "sel     z8.s, p2, z8.s, z9.s                        \n\t"
        "lsr     z9.s, z8.s, #23                             \n\t"
        "sub     z9.s, z6.s, z9.s                            \n\t"
        "cmplo   p3.s, p0/z, z9.s, %[table_max_index]        \n\t"
        "sel     z9.s, p3, z9.s, z7.s                        \n\t"
        "fmul    z10.s, z8.s, z8.s                           \n\t"
        "tbl     z11.s, {z2.s}, z9.s                         \n\t"
        "tbl     z12.s, {z4.s}, z9.s                         \n\t"
        "fmad    z12.s, p0/m, z10.s, z11.s                   \n\t"
        "tbl     z11.s, {z3.s}, z9.s                         \n\t"
        "tbl     z13.s, {z5.s}, z9.s                         \n\t"
        "fmad    z13.s, p0/m, z10.s, z11.s                   \n\t"
        "fmad    z13.s, p0/m, z8.s, z12.s                    \n\t"
        "fneg    z10.s, p0/m, z13.s                          \n\t"
        "sel     z10.s, p2, z13.s, z10.s                     \n\t"
        "st1w    {z10.s}, p1, [%[output], %[i], lsl #2]      \n\t"
        "incw    %[i]                                        \n\t"
        "whilelo p1.s, %[i], %[N]                            \n\t"
        "b.any   1b                                            \n"
        "2:                                                  \n\t"
        : [i] "=&r" (i_dummy)
        : "[i]" (0), [input] "r" (input), [output] "r" (output), [N] "r" (N),
        [a_0] "r" (a_0), [a_1] "r" (a_1), [a_2] "r" (a_2), [a_3] "r" (a_3),
        [table_max_index] "i" (TABLE_MAX_INDEX)
        : "memory", "cc", "p0", "p1", "p2", "p3", "z0", "z1", "z2", "z3",
        "z4", "z5", "z6", "z7", "z8", "z9", "z10", "z11", "z12", "z13"
        );
}
```

Code 4.3.3 – An adaptation of Codes 4.3.1 and 4.3.2 using SVE inline assembly to ensure the coefficients are held in vector registers throughout the entire function call (achieved using the table lookup command `tbl`). The variable `i_dummy` is to ensure the variable `i` has an output and the `=&` indicates this needs its own register to the other inputs. Combining this with the `"memory"` side effect ensures that the compiler cannot remove the `i_dummy` variable, even though it is not used [12]. This is compiled using the same flags as Code 4.3.2.

reference to the location in memory using the "`ZMMWORD PTR [rip+0x0]`" was not desired. Although it is not shown in Code 4.3.4, the tail loop is handled using masked gather

```
<vs_inverse_gaussian_cdf_logarithmic_polynomial_approx>:
...
 76:   vmovups zmm13,ZMMWORD PTR [r13+rdx*4+0x0]
 7e:   vcmpltps k1,zmm13,zmm5
 85:   vmovdqu8 xmm6{k1}{z},xmm4
 8b:   vpcmpnequb k3,xmm2,xmm6
 92:   knotw  k2,k3
 96:   vsubps zmm13{k2},zmm3,zmm13
 9c:   vmulps zmm15,zmm13,zmm13
 a2:   vpsrld zmm7,zmm13,0x17
 a9:   vpsubd zmm8,zmm1,zmm7
 af:   vpminud zmm9,zmm8,zmm0
 b5:   vpermps zmm11,zmm9,ZMMWORD PTR [rip+0x0]        # coefficients
 bf:   vpermps zmm10,zmm9,ZMMWORD PTR [rip+0x0]        # coefficients
 c9:   vpermps zmm12,zmm9,ZMMWORD PTR [rip+0x0]        # coefficients
 d3:   vpermps zmm14,zmm9,ZMMWORD PTR [rip+0x0]        # coefficients
 dd:   vfmadd213ps zmm15,zmm10,zmm11
 e3:   vfmadd231ps zmm15,zmm12,zmm13
 e9:   vfmadd213ps zmm15,zmm13,zmm14
 ef:   vxorps zmm16,zmm15,DWORD PTR [rip+0x0]{1to16}
 f9:   vmovaps zmm16{k3},zmm15
 ff:   vmovups ZMMWORD PTR [r14+rdx*4],zmm16
106:   add    edx,0x10
109:   cmp    edx,eax
10b:   jb     76 <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x76>
...
24d:   ret
...
```

Code 4.3.4 – The assembly mnemonics produced from compiling Code 4.3.1 using `icc` (18.0.0) with `std=c11`, `O3`, `simd`, `p`, `mkl`, `qopenmp`, `xCore-AVX512`, `march=skylake-avx512`, and `qopt-zmm-usage=high`. The assembly uses the Intel assembly syntax. (The assembly has been slightly abbreviated and simplified in places for improved readbility).

operations. This highlights that for the main loop the compiler recognised the applicability of the permute instruction. In order to have arrays of coefficients held in constant registers throughout the duration of the main loop body, we were forced to turn to using Intel vector intrinsics. While the use of vector intrinsics was considerably more work than the simple OpenMP implementation, and of course resulted in non-portable code, it was quicker to build and debug than inline assembly. The implementation using the Intel vector intrinsics is shown in Code 4.3.5.

We can then inspect the instructions produced by Code 4.3.5, where the assembly produced is shown in Code 4.3.6. We can notice that the coefficients are stored in vector registers outside of the main loop body, and then accessed again using the `vpermd` permutation instruction.

Having turned to vector intrinsics to issue table lookups (which use underlying permutation instructions) acting on constant arrays/registers of coefficients, we can compare the speed improvements that such a scheme provides. The performance data were gathered on a 64-bit x86 Intel Xeon Gold 6140 CPU operating at 2.3 GHz under GNU/Linux 4.13.0-25-generic under the Ubuntu 16.04.3 LTS distribution. The measurements for uniformly drawn inputs are shown in Table 4.3.3.

We can see from Table 4.3.3 that even the OpenMP approximation of $\Phi^{-1}(\cdot)$ produces significant time savings compared to the exact single-precision implementation of $\Phi^{-1}(\cdot)$, including those from Intel's MKL. For comparison, we have also included the open-source implementations from Burkardt [39] [213] (ASA241) and Giles [77]. These three were also compiled using the Intel compiler, but unlike earlier we made no attempt to vectorise

```
void vs_inverse_gaussian_cdf_logarithmic_polynomial_approx(
    unsigned int N,
    const float * restrict input,
    float * restrict output)
{
    int n_iterations = N / VECTOR_LENGTH;  // Assuming no loop tail.
    /* Loading in constants. */
    __m512 half_float = _mm512_set1_ps(0.5f);
    __m512 one_float = _mm512_set1_ps(1.0f);
    __m512 minus_zero_float = _mm512_set1_ps(-0.0f);  // For negation using XOR.
    __m512i exponent_bias_table_offset = _mm512_set1_epi32(126);
    __m512i table_max_index = _mm512_set1_epi32(TABLE_MAX_INDEX);
    __m512 coef_0 = _mm512_load_ps(poly_coef_0);
    __m512 coef_1 = _mm512_load_ps(poly_coef_1);
    __m512 coef_2 = _mm512_load_ps(poly_coef_2);
    __m512 coef_3 = _mm512_load_ps(poly_coef_3);
    __m512 c_0, c_1, c_2, c_3;  // The required coefficients.
    __m512i c;  // A useful temporary variable for aliasing.
    /* Loop variables. */
    __m512 u, z, z_even, z_odd, u_squared;
    __m512i idx;
    __mmask16 predicate;
    for (int i = 0, offset = 0; i < n_iterations; i++, offset += VECTOR_LENGTH)
    {
        u = _mm512_loadu_ps(input + offset);
        predicate = _mm512_cmplt_ps_mask(half_float, u);
        u = _mm512_mask_sub_ps(u, predicate, one_float, u);
        /* Building the index. */
        idx = _mm512_castps_si512(u);  // Aliasing (no instruction issued).
        idx = _mm512_srli_epi32(idx, N_MANTISSA_32);
        idx = _mm512_sub_epi32(exponent_bias_table_offset, b);
        idx = _mm512_min_epi32(idx, table_max_index);
        /* Loading the required polynomial coefficients. */
        c = _mm512_castps_si512(coef_0);
        c = _mm512_permutexvar_epi32(idx, c); // Shuffles use 32-bit integers.
        c_0 = _mm512_castsi512_ps(c);
        c = _mm512_castps_si512(coef_1);
        c = _mm512_permutexvar_epi32(idx, c);
        c_1 = _mm512_castsi512_ps(c);
        c = _mm512_castps_si512(coef_2);
        c = _mm512_permutexvar_epi32(idx, c);
        c_2 = _mm512_castsi512_ps(c);
        c = _mm512_castps_si512(coef_3);
        c = _mm512_permutexvar_epi32(idx, c);
        c_3 = _mm512_castsi512_ps(c);
        /* Constructing the polynomial approximation. */
        u_squared = _mm512_mul_ps(u, u);
        z_even = _mm512_fmadd_ps(c_2, u_squared, c_0);
        z_odd = _mm512_fmadd_ps(c_3, u_squared, c_1);
        z = _mm512_fmadd_ps(z_odd, u, z_even);
        z = _mm512_mask_xor_ps(z, predicate, minus_zero_float, z); // Negation.
        _mm512_storeu_ps(output + offset, z);
    }
}
```

Code 4.3.5 – An adaptation of Code 4.3.1 using Intel intrisics to remove redundant loads. This is compiled with `icc` (18.0.0) using `std=c11`, `O3`, `simd`, `p`, `mkl`, `qopenmp`, `xCore-AVX512`, `march=skylake-avx512`, and `qopt-zmm-usage=high`.

these implementations. As we saw from the earlier double-precision experiments from Section 4.1.2.1 we were unable to produce faster implementations than Intel, so they are purely presented for a relative comparison, where the performance increase of the dyadic

```
<vs_inverse_gaussian_cdf_logarithmic_polynomial_approx>:
...
 5e:   vmovups zmm3,ZMMWORD PTR [rip+0x0]        # coefficients
 68:   vmovups zmm2,ZMMWORD PTR [rip+0x0]        # coefficients
 72:   vmovups zmm1,ZMMWORD PTR [rip+0x0]        # coefficients
 7c:   vmovups zmm0,ZMMWORD PTR [rip+0x0]        # coefficients
...
 90:   vmovups zmm15,ZMMWORD PTR [r14+rax*4]
 97:   inc    edx
 99:   vcmpltps k1,zmm8,zmm15
 a0:   vsubps zmm15{k1},zmm7,zmm15
 a6:   vmulps zmm13,zmm15,zmm15
 ac:   vpsrld zmm9,zmm15,0x17
 b3:   vpsubd zmm10,zmm5,zmm9
 b9:   vpminsd zmm11,zmm10,zmm4
 bf:   vpermd zmm12,zmm11,zmm3
 c5:   vpermd zmm14,zmm11,zmm2
 cb:   vpermd zmm16,zmm11,zmm1
 d1:   vpermd zmm17,zmm11,zmm0
 d7:   vfmadd213ps zmm16,zmm13,zmm12
 dd:   vfmadd213ps zmm17,zmm13,zmm14
 e3:   vfmadd213ps zmm17,zmm15,zmm16
 e9:   vxorps zmm17{k1},zmm6,zmm17
 ef:   vmovups ZMMWORD PTR [r15+rax*4],zmm17
 f6:   add    eax,0x10
 f9:   cmp    edx,ebx
 fb:   jb     90 <vs_inverse_gaussian_cdf_logarithmic_polynomial_approx+0x90>
...
10d:   ret
...
```

Code 4.3.6 – The assembly mnemonics produced from compiling Code 4.3.5 using `icc` (18.0.0) with `std=c11`, `O3`, `simd`, `p`, `mkl`, `qopenmp`, `xCore-AVX512`, `march=skylake-avx512`, and `qopt-zmm-usage=high`. The assembly uses the Intel assembly syntax. (The assembly has been slightly abbreviated and simplified in places for improved readbility).

implementations should primarily be compared to Intel's library functions. Of course, if a developer is required to use freely available implementations, then the performance improvement will only be more pronounced. Similarly, we have included $\sin(\cdot)$ for comparison, and a simple read and write, where the latter indicates the maximum throughput speed which any implementation of $\Phi^{-1}(\cdot)$ can hope to achieve. We can see that our linear dyadic implementation using intrinsics is very near this maximum speed.

We can see that the transition from the OpenMP implementation to the vector intrinsics version produces a speedup by approximately 25%. Comparing this to Intel (HA), the dyadic implementation for the piecewise cubic polynomials presents a speed increase by a factor of approximately 4–5 , and from Figure 4.3.2 we recall this gives an RMSE of 3×10^{-4}. If we are prepared to use the slightly less refined linear approximation with an RMSE of 6×10^{-3} then the speed of such an approximation becomes considerably faster, and quite near the maximum function speed for any function that requires the reading and writing of data.

We can conclude from these investigations that while an equipartitioned lookup table is performant on a range of hardwares, if we can ensure the vector length is sufficiently large then the dyadic approximation typically has the best speed and RMSE. The dyadic approximation using cubic polynomials has the superior RMSE, but for the fastest speeds the linear dyadic approximation is the best option. However, the dyadic approximation does require certain hardware assumptions about the vector length and typically requires more involved coding which is not strictly language compliant. Consequently, for most of the numerics going forward we will be using the equipartitioned lookup table. Nonetheless, we would advise practitioners

Implementation	Time (Clock cycles)	Precision
ASA241 [213]	46.6 ± 0.3	Single
Giles (2011) [77]	45.5 ± 0.7	Single
Intel (HA)	3.42 ± 0.04	Single
Intel (LA)	2.6 ± 0.01	Single
$\sin(\cdot)$	1.2 ± 0.1	Single
Dyadic (OpenMP) Cubic	0.928 ± 0.01	Single
Dyadic (Intrinsics) Cubic	0.715 ± 0.01	Single
Dyadic (Intrinsics) Linear	0.507 ± 0.02	Single
Read and write	0.377 ± 0.03	Single

Table 4.3.3 – Average times for uniformly drawn input as measured on Intel hardware for some single-precision implementations of $\Phi^{-1}(\cdot)$ and the dyadic approximation. For the dyadic approximation we consider the simple autovectorisation achieved by using OpenMP SIMD pragmas and more involved implementations using Intel vector intrinsics, where we consider piecewise cubic and linear approximations. For comparative purposes we also include a simple $\sin(\cdot)$ operation and a read and write operation.

who want to maximally utilise our proposals to consider using our dyadic approximations (if possible).

Chapter 5

Euler-Maruyama schemes

In this chapter we present the usual Euler-Maruyama scheme for approximating solutions to stochastic differential equations. We review the strong convergence of such a scheme and present a revised analysis of an approximate Euler-Maruyama scheme (which uses approximate random variables).

5.1 The Euler-Maruyama scheme

We will be considering the general nonautonomous scalar *stochastic differential equation* (SDE)

$$\mathrm{d}X_t = a(t, X_t)\,\mathrm{d}t + b(t, X_t)\,\mathrm{d}W_t\,, \tag{5.1.1}$$

where W_t is a standard Wiener process for $t \in [t_0, T]$ under some filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}, \mathcal{F}_t)$, where for brevity $W_t \equiv W_t^{\mathbb{P}}$ unless otherwise stated. The solution to (5.1.1) is correctly interpreted as the solution to the integral equation $X_T = X_{t_0} + \int_{t_0}^T a(s, X_s)\,\mathrm{d}s + \int_{t_0}^T b(s, X_s)\,\mathrm{d}W_s$, where the first integral is a Lebesgue integral and the second an Itô integral. Specifically we have the following definition of a *strong solution*.

Definition 5.1.1 (strong solution). [131, Definition 5.1, page 126] [133, pages 105 and 128] [46, Definition 1, pages 255–256] A continuous and $\mathcal{F}_t$-adapted process X_t is a *strong solution* to the SDE (5.1.1) if for all $t_0 \leq t \leq T$ the integrals $\int_{t_0}^t a(s, X_s)\,\mathrm{d}s$ and $\int_{t_0}^t b(s, X_s)\,\mathrm{d}W_s$ exist, (the second being an Itô integral), and $X_t = X_{t_0} + \int_{t_0}^t a(s, X_s)\,\mathrm{d}s + \int_{t_0}^t b(s, X_s)\,\mathrm{d}W_s$ for any given Wiener process W_t.

For such a stochastic differential equation, and the Euler-Maruyama scheme with approximate solution $\hat{X}_T$ and uniform time increment δ, we will frequently make use of a few key assumptions [133, 4.5]:

Assumption 5.1.1 (measurability). The functions a and b are jointly (Lebesgue) $\mathcal{L}^2$-measurable in $(t, x) \in [t_0, T] \times \mathbb{R}$.

Assumption 5.1.2 (Lipschitz continuity). There exists a constant $K > 0$ such that $|a(t, x) - a(t, y)| \leq K|x - y|$ for all $t \in [t_0, T]$ and $x, y \in \mathbb{R}$, and identically for b.

Assumption 5.1.3 (linear growth). There exists a constant $K > 0$ such that $|a(t, x)| \leq K(1 + |x|)$ for all $t \in [t_0, T]$ and $x \in \mathbb{R}$, and identically for b.

Assumption 5.1.4 (measurable initial condition). X_{t_0} is $\mathcal{F}_{t_0}$-measurable with $\mathbb{E}(X_{t_0}^2) < \infty$.

Assumption 5.1.5 (convergent initial value). There exists a constant $K > 0$ such that $\mathbb{E}(|X_{t_0} - \hat{X}_{t_0}|^2) \leq K\delta$.

Assumption 5.1.6 (temporal Hölder continuity with linear growth). There exists a constant $K > 0$ such that $|a(t,x) - a(s,x)| \leq K(1+|x|)\sqrt{|t-s|}$ for all $s,t \in [t_0,T]$ and $x \in \mathbb{R}$, and identically for b.

Assumptions 5.1.1 to 5.1.2 are sufficient to show the following result about the existence of a strong solution to (5.1.1):

Theorem 5.1.1. *[133, Lemma 4.5.2] [145, Theorem 8.18] Under Assumptions 5.1.1 to 5.1.2, then if strong solutions to (5.1.1) exists, then for the same initial value and Wiener process these form a pathwise unique solution.*

In addition to this, the following corollary stems from Assumption 5.1.2, which is a result we will later need to extend.

Corollary 5.1.1.1. *The partial derivatives $\frac{\partial a}{\partial x}$ and $\frac{\partial b}{\partial x}$ exist almost everywhere and are bounded.*

Proof. This is Rademacher's theorem (Theorem A.1.2) applied to Assumption 5.1.2. **QED**

For the purposes of our scheme we suppose that we have discretised our time interval $[t_0,T]$ into N increments $\{\tau_i\}_{i=0}^N$ where $\tau_0 = t_0$ and $\tau_N = T$ with $\tau_{i+1} - \tau_i = \delta$ for all $i \in \{0,1,\dots,N-1\}$, where $\delta N = T - t_0$. For convenience we denote $\hat{X}_i \equiv \hat{X}_{\tau_i}$ for all $i \in \{0,1,\dots,N\}$. With these notations we have the usual Euler-Maruyama scheme

$$\hat{X}_{n+1} = \hat{X}_n + a(\tau_n, \hat{X}_n)\delta + b(\tau_n, \hat{X}_n)\sqrt{\delta}\, Z_n, \tag{5.1.2}$$

where $\{Z_i\}_{i=0}^{N-1}$ are *independently and identically distributed* (i.i.d.) $\mathcal{F}_{\tau_{i+1}}$-measurable random variables from a standard Gaussian distribution $Z_i \sim \mathcal{N}(0,1)$ where $Z_i \coloneqq \sqrt{\frac{N}{T-t_0}}(W_{\tau_{i+1}} - W_{\tau_i})$. Having introduced this scheme, we have the following extension to Theorem 5.1.1.

Theorem 5.1.2. *[133, Theorem 4.5.3] Under Assumptions 5.1.1 to 5.1.4 the stochastic differential equation (5.1.1) has a pathwise unique strong solution X_t on $[t_0,T]$ with $\sup_{t\in[t_0,T]} \mathbb{E}(X_t^2) < \infty$. Furthermore, under Assumptions 5.1.1 to 5.1.6 the same bound also applies for the Euler-Maruyama approximation $\hat{X}$ [133, 10.2, (2.15)].*

So now given an approximation scheme, we can define the weak and strong convergence between X and $\hat{X}$, and note the main convergence theorem for the Euler-Maruyama scheme in our context, for which the strong error will be more relevant.

Definition 5.1.2 (weak convergence order). [133, 9.7] A general time discretisation $\hat{X}$ with maximal time increment δ *converges weakly* with order $\gamma > 0$ to X at time T if there exists a constant $K > 0$, where K is independent of δ, and a $0 < \delta_0 < \infty$, such that for any $g \in C_{\mathrm{P}}^{2(\gamma+1)}$ we have $|\mathbb{E}(g(\hat{X}_T)) - \mathbb{E}(g(X_T))| \leq K\delta^\gamma$ for any $\delta \in (0,\delta_0)$. Here C_{P}^l is the set of test functions which are l-times continuously differentiable with the derivatives up to and including the l-th derivative having polynomial growth.

Definition 5.1.3 (strong convergence order). [133, 9.6] A general time discretisation $\hat{X}$ with maximal time increment δ *converges strongly* with order $\gamma > 0$ to X at time T if there exists a constant $K > 0$, where K is independent of δ, and a $0 < \delta_0 < \infty$, such that $\mathbb{E}(|\hat{X}_T - X_T|) \leq K\delta^\gamma$ for any $\delta \in (0,\delta_0)$.

Theorem 5.1.3. *[133, Theorem 10.2.2] Under Assumptions 5.1.1 to 5.1.6, where the constants do not depend on δ, then there exists a constant $K > 0$, independent of δ, and a $0 < \delta_0 < \infty$, such that $\mathbb{E}(\sup_{n\leq N} |\hat{X}_n - X_{\tau_n}|) \leq K\delta^{1/2}$, for any $\delta \in (0,\delta_0)$. Furthermore for any $1 \leq p < \infty$ we also have the bound $\mathbb{E}(\sup_{n\leq N} |\hat{X}_n - X_{\tau_n}|^p) \leq K_p\delta^{p/2}$.*

Our primary result from this chapter will be to extend Theorem 5.1.3 for a variant of (5.1.2) which employs approximate random variables.

5.1.1 Systems of SDEs

As presented thus far, we have limited the scope of our considerations to a single stochastic process, rather than a more generalised multidimensional system of stochastic differential equations. While our assumptions can be readily generalised to multidimensional counterparts, the key requirement for extending the results of this chapter would rely on a multidimensional analogue of Lemma 5.2.3, which we will later present as the key tool underpinning our analytic results. Lemma 5.2.3 in turn utilises Jensen's inequality for expectations (Lemma A.2.2), Doob's maximal inequalities (Lemma A.2.6), the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7), and Grönwall's discrete inequalities (Lemma A.1.9). As noted by Harter and Richou [100, 1.1], multidimensional versions of these do exist in a similar form, albeit with possibly different coefficients [100, e.g. page 3]. Thus, we anticipate that it may be possible to extend the results presented in this chapter to multidimensional stochastic processes.

However, the later results in Chapter 6 surrounding nested multilevel Monte Carlo frameworks use the mean value theorem (theorem A.1.5) and a variant thereof (the four-point mean value theorem (Lemma 6.2.1)). These do not necessarily facilitate multidimensional processes nor vector valued functions, and thus extending the results of Chapter 6 to systems of stochastic processes may indeed be more difficult (or even impossible).

5.2 The approximate Euler-Maruyama scheme

We propose an *approximate Euler-Maruyama scheme*, producing the approximate solution $\tilde{X}_T$, which substitutes the Gaussian random variables Z_n with approximate Gaussian random variables $\tilde{Z}_n$, giving

$$\tilde{X}_{n+1} = \tilde{X}_n + a(\tau_n, \tilde{X}_n)\delta + b(\tau_n, \tilde{X}_n)\sqrt{\delta}\,\tilde{Z}_n, \tag{5.2.1}$$

where $\tilde{Z} \approx \mathcal{N}(0,1)$. We will suppose that there is some parameter $q \in [0,\infty]$ which describes the fidelity of the approximation such that $\lim_{q\to\infty} \tilde{Z} \sim \mathcal{N}(0,1)$. More specifically, for some truly Gaussian random variable Z_n we suppose that the corresponding approximating random variable $\tilde{Z}_n$ converges to Z_n in some appropriate sense. This convergence could be either in probability, expectation, cumulative distribution functions, etc.

Notice that the weak Euler-Maruyama scheme [133, page XXXII, (27)–(28)] using Rademacher random variables ($\tilde{Z} = \pm 1$ with equal probability) is an example of such an approximate Euler-Maruyama scheme.

5.2.1 L^p strong convergence

The key result which we mirror is Theorem 5.1.3, the strong convergence of the Euler-Maruyama scheme, as presented rigorously by Kloeden and Platen [133, Theorem 10.2.2]. (A less rigorous sketch-proof for autonomous stochastic differential equations is presented by Jahnke [123, 5.2.3]).

Assumption 5.2.1 (L^p-measurable initial condition). X_{t_0} is $\mathcal{F}_{t_0}$-measurable with $\mathbb{E}(|X_{t_0}|^p) < \infty$ for a finite integer $p \geq 1$

Assumption 5.2.2 (bounded approximate normals). $\mathbb{E}(|\tilde{Z}|^p) < \infty$ for any integer $p \geq 1$.

Assumption 5.2.3. $\mathbb{E}(\tilde{Z}) = 0$ and $\mathbb{E}(|\tilde{Z} - Z|^p) \leq \mathbb{E}(|Z|^p) < \infty$ for any finite p.

Corollary 5.2.0.1. $\mathbb{E}(|\tilde{Z}|^p) \leq 2^p\mathbb{E}(|Z|^p) < \infty$ *for any finite* p.

Proof. From the triangle inequality we have $|\tilde{Z}| \leq |Z| + |\tilde{Z} - Z|$. Raising this to the p-th power and applying Jensen's inequality for summations (Lemma A.1.7) we have $|\tilde{Z}|^p \leq 2^{p-1}(|Z|^p + |\tilde{Z} - Z|^p)$, and from Assumption 5.2.3 and Lemma 3.2.1 the result immediately follows. **QED**

We are almost ready to present our result for the L^p strong error. However, in later proofs we will require the existence and finiteness of several moments for the application of Doob's maximal inequalities (Lemma A.2.6), the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7), and Grönwall's discrete inequalities (Lemma A.1.9).

Lemma 5.2.1. *Under Assumptions 5.1.1 to 5.1.4 and Assumption 5.2.1, for an approximate Euler-Maruyama scheme approximation $\tilde{X}$, for any $1 \leq p < \infty$, if $\mathbb{E}(|\tilde{Z}|^p) < \infty$ then we have $\mathbb{E}(\sup_{n \leq N} |\tilde{X}_n|^p) < \infty$, and similarly for the regular Euler-Maruyama approximation $\hat{X}$. Furthermore, this bound is uniform with respect to N such that for any $1 \leq p < \infty$ we have $\sup_N \mathbb{E}(\sup_{n \leq N} |\tilde{X}_n|^p) < \infty$.*

Proof. Begin by defining $\mathscr{Z}_N := \mathbb{E}(\sup_{n \leq N} |\hat{X}|^p)$ and then use Jensen's inequality for summations (Lemma A.1.7) to give

$$\mathscr{Z}_N \leq 3^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\hat{X}_0\right|^p\right) + 3^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\sum_{m=0}^{n-1} a_m \delta\right|^p\right) + 3^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\sum_{m=0}^{n-1} b_m \sqrt{\delta} Z_m\right|^p\right) \tag{5.2.2}$$

using Assumption 5.1.4, Jensen's inequality for summations (Lemma A.1.7), Doob's maximal inequalities (Lemma A.2.6), and letting C_p represent an arbitrary constant dependent only on p (where C_p changes from line to line and term to term), we have

$$\leq C_p + C_p \delta^p N^{p-1} \sum_{m=0}^{N-1} \mathbb{E}\left(\sup_{n \leq m} |a_n|^p\right) + C_p \delta^{\frac{p}{2}} \mathbb{E}\left(\left|\sum_{m=0}^{N-1} b_m Z_m\right|^p\right) \tag{5.2.3}$$

using the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7)

$$\leq C_p + C_p \delta \sum_{m=0}^{N-1} \mathbb{E}\left(\sup_{n \leq m} |a_n|^p\right) + C_p \delta \sum_{m=0}^{N-1} \mathbb{E}(|b_m|^p)\mathbb{E}(|Z_m|^p) \tag{5.2.4}$$

using $\mathbb{E}(|Z|^p) < \infty$, (and similarly for $\tilde{Z}$), we can then use Assumption 5.1.3

$$\leq C_p + C_p \delta \sum_{m=0}^{N-1} \mathscr{Z}_m \tag{5.2.5}$$

using Grönwall's discrete inequalities (Lemma A.1.9)

$$\leq C_p. \tag{5.2.6}$$

An identical procedure follows for $\tilde{X}$ for bounding $\mathbb{E}(\sup_{n \leq N} |\tilde{X}|^p)$, and hence this completes the proof. **QED**

Corollary 5.2.1.1. $\mathbb{E}(\sup_{n \leq N} |\hat{X}_n - \tilde{X}_n|^p) < \infty$.

Proof.

$$\mathbb{E}\left(\sup_{n \leq N} \left|\hat{X}_n - \tilde{X}_n\right|^p\right) \leq \mathbb{E}\left(\sup_{n \leq N} \left|\left|\hat{X}_n\right| + \left|\tilde{X}_n\right|\right|^p\right) \tag{5.2.7}$$

$$\leq 2^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left\{\left|\hat{X}_n\right|^p + \left|\tilde{X}_n\right|^p\right\}\right) \tag{5.2.8}$$

$$\leq 2^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\hat{X}_n\right|^p + \sup_{n \leq N} \left|\tilde{X}_n\right|^p\right) \tag{5.2.9}$$

$$\leq 2^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\hat{X}_n\right|^p\right) + 2^{p-1}\mathbb{E}\left(\sup_{n \leq N} \left|\tilde{X}_n\right|^p\right) \tag{5.2.10}$$

using Lemma 5.2.1

$$< \infty. \tag{5.2.11}$$

QED

Lemma 5.2.2. *For any integer $p \geq 1$ we have $\mathbb{E}(|\hat{X}_{n+1} - \hat{X}_n|^p) \leq C\delta^{p/2}$ for some constant C, and identically for $\tilde{X}$.*

Proof. For the Euler-Maruyama updating step we need only take $|\cdot|^p$ and apply Jensen's inequality for summations (Lemma A.1.7), and then note that for a suitably scaled interval that $\delta^p \leq \delta^{p/2}$, and the result immediately follows, and identically for $\tilde{X}$. **QED**

We will frequently find that much of the analysis bounding various types of strong errors follows a very similar procedure to these preceding results. To avoid having to repeatedly go through all the steps involved in applying Grönwall's discrete inequalities (Lemma A.1.9) and the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7) and the associated minutia, we will present Lemma 5.2.3 which bounds the error arising from a numerical scheme with martingale and non-martingale terms contributing.

Lemma 5.2.3. *Suppose we have the following updating scheme for the process $\mathcal{E}_n$ with a discretisation interval δ*

$$\mathcal{E}_{n+1} = \mathcal{E}_n + \delta\mathcal{A}_n + \sqrt{\delta}\, Z_n\mathcal{B}_n + \Xi_n + \Theta_n, \tag{5.2.12}$$

where $\mathcal{E}_0 = 0$ almost surely, Z_n are i.i.d. zero mean random variables with all finite moments bounded, and $\mathcal{A}_n$ and $\mathcal{B}_n$ are $\mathcal{F}_{\tau_n}$-adapted with $|\mathcal{A}_n| \leq L_A|\mathcal{E}_n|$ and $|\mathcal{B}_n| \leq L_B|\mathcal{E}_n|$ for some strictly positive and finite constants L_A and L_B. The process Ξ is a martingale where $\mathbb{E}(\Xi_n \,|\, \mathcal{F}_{\tau_n}) = 0$, and for integers $p \geq 2$ and a constant $s \in \mathbb{R}$ there are finite and strictly positive constants c_1 and c_2 such that $\mathbb{E}(|\Xi_n|^p) \leq c_1\delta^{p(s+1/2)}$, and similarly $\mathbb{E}(|\Theta_n|^p) \leq c_2\delta^{p(s+1)}$ for Θ. Then there exists constants c_3 and c_4 which depends only on L_A, L_B, p, and $(T - t_0)$ such that

$$\mathbb{E}\left(\sup_{n\leq N}|\mathcal{E}_n|^p\right) \leq c_3\Big(c_4c_1 + (T-t_0)^{\frac{p}{2}}c_2\Big)\delta^{ps}, \tag{5.2.13}$$

where $c_4 = 18p^{3/2}(p-1)^{-3/2}$.

Proof. We begin by defining $\mathscr{Z}_N \coloneqq \mathbb{E}(\sup_{n\leq N}|\mathcal{E}_n|^p)$, and by applying Jensen's inequality for summations (Lemma A.1.7) to (5.2.12) we obtain

$$\begin{aligned}\mathscr{Z}_N \leq\ & 4^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\sum_{m=0}^{n-1}\delta\mathcal{A}_m\right|^p\right) + 4^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\sum_{m=0}^{n-1}\sqrt{\delta}\, Z_m\mathcal{B}_m\right|^p\right) \\ &+ 4^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\sum_{m=0}^{n-1}\Xi_m\right|^p\right) + 4^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\sum_{m=0}^{n-1}\Theta_m\right|^p\right).\end{aligned} \tag{5.2.14}$$

We can apply the triangle inequality along with $|\mathcal{A}_n| \leq L_A|\mathcal{E}_n|$ to the first term to bound this by $(T-t_0)^{p-1}L_A^p\delta\sum_{m=0}^{N-1}\mathscr{Z}_m$. For the second term, we notice that as the Z_m are i.i.d. and zero mean that the summation within the supremum forms a martingale. So by defining $\mathcal{M}_n \coloneqq \sum_{m=0}^{n-1}\mathcal{B}_mZ_m$ and applying the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7) we can bound this by $C_p\delta^{p/2}\mathbb{E}(\langle\mathcal{M}\rangle_N^{p/2})$, where in this instance C_p is the specific constant $C_p = 18p^{3/2}(p-1)^{-3/2}$ coming from the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7). By applying Jensen's inequality for summations (Lemma A.1.7), $|\mathcal{B}_n| \leq L_B|\mathcal{E}_n|$, and denoting

$\Omega_p \coloneqq \mathbb{E}(|Z|^p) < \infty$ we can bound this all by $C_p\Omega_p L_B^p (T-t_0)^{p/2-1}\delta\sum_{m=0}^{N-1}\mathscr{Z}_m$, giving

$$\mathscr{Z}_N \leq 4^{p-1}\Big((T-t_0)^{p-1}L_A^p + C_p\Omega_p L_B^p (T-t_0)^{\frac{p}{2}-1}\Big)\delta\sum_{m=0}^{N-1}\mathscr{Z}_m + 4^{p-1}\mathbb{E}\bigg(\sup_{n\leq N}\bigg|\sum_{m=0}^{n-1}\Xi_m\bigg|^p\bigg) + 4^{p-1}\mathbb{E}\bigg(\sup_{n\leq N}\bigg|\sum_{m=0}^{n-1}\Theta_m\bigg|^p\bigg). \tag{5.2.15}$$

For the martingale term involving Ξ we again use Doob's maximal inequalities (Lemma A.2.6), the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7), and Jensen's inequality for summations (Lemma A.1.7) which bounds this contribution by $C_p N^{p/2-1}\sum_{m=0}^{N-1}\mathbb{E}(|\Xi_m|^p)$. Recalling that $\mathbb{E}(|\Xi_n|^p) \leq c_1\delta^{p(s+1/2)}$ this bound becomes $C_p c_1 (T-t_0)^{p/2}\delta^{ps}$. Similarly the term involving Θ is tackled by applying Jensen's inequality for summations (Lemma A.1.7) and is bounded by $c_2(T-t_0)^p\delta^{ps}$. Incorporating these two bounds gives

$$\mathscr{Z}_N \leq 4^{p-1}\Big((T-t_0)^{p-1}L_A + C_p\Omega_p L_B^p (T-t_0)^{\frac{p}{2}-1}\Big)\delta\sum_{m=0}^{N-1}\mathscr{Z}_m + 4^{p-1}\Big(C_p c_1 (T-t_0)^{\frac{p}{2}}\delta^{ps} + c_2(T-t_0)^p\delta^{ps}\Big). \tag{5.2.16}$$

By applying Grönwall's discrete inequalities (Lemma A.1.9) and after a short amount of simplification we obtain

$$\leq c_3\Big(C_p c_1 + (T-t_0)^{\frac{p}{2}}c_2\Big)\delta^{ps}. \tag{5.2.17}$$

We know from the discrete Burkholder-Davis-Gundy inequality (Lemma A.2.7) that in this case the final $C_p = 18p^{3/2}(p-1)^{-3/2}$, which completes the proof. **QED**

Theorem 5.2.4. *Under Assumptions 5.1.1 to 5.1.6 and Assumption 5.2.1, where the constants do not depend on δ, we assume there exists a symmetric approximate Gaussian distribution $\tilde{Z}$ satisfying Assumption 5.2.2. Then there exists a finite constant $0 < K_p < \infty$, independent of N, δ, and q, but dependent on p, such that $\mathbb{E}(\sup_{n\leq N}|\hat{X}_n - \tilde{X}_n|^p) \leq K_p\mathbb{E}(|\tilde{Z}-Z|^p)$ for any $p \geq 2$.*

Proof. Using the abbreviated notation $\hat{a}_n \equiv a(\tau_n, \hat{X}_n)$ and $\tilde{a}_n \equiv a(\tau_n, \tilde{X}_n)$, and similarly for b, and defining $\mathcal{E}_n \coloneqq \hat{X}_n - \tilde{X}_n$, then by differencing (5.1.2) and (5.2.1) we obtain

$$\mathcal{E}_{n+1} = \mathcal{E}_n + \delta(\hat{a}_n - \tilde{a}_n) + \sqrt{\delta}\,Z_n(\hat{b}_n - \tilde{b}_n) + \sqrt{\delta}\,\tilde{b}_n(Z_n - \tilde{Z}_n). \tag{5.2.18}$$

This is immediately in the form suitable for applying Lemma 5.2.3. Using Assumption 5.1.2 we immediately have $|\hat{a}_n - \tilde{a}_n| \leq K|\mathcal{E}_n|$, and similarly $|\hat{b}_n - \tilde{b}_n| \leq K|\mathcal{E}_n|$. The only term to consider is the final martingale term $\sqrt{\delta}\,\tilde{b}_n(Z_n - \tilde{Z}_n)$. For this we can utilise that $Z_n - \tilde{Z}_n$ is independent to $\tilde{b}_n$ and hence write

$$\mathbb{E}\Big(\Big|\sqrt{\delta}\,\tilde{b}_n(Z_n - \tilde{Z}_n)\Big|^p\Big) = \delta^{\frac{p}{2}}\mathbb{E}\Big(\Big|\tilde{b}_n\Big|^p\Big)\mathbb{E}\Big(\Big|Z-\tilde{Z}\Big|^p\Big) \tag{5.2.19}$$

using Assumption 5.1.3 and Lemma 5.2.1 we bound $\mathbb{E}(|\tilde{b}_n|^p) < \infty$ to give

$$\leq K_p\mathbb{E}\Big(\Big|Z-\tilde{Z}\Big|^p\Big)\delta^{\frac{p}{2}}, \tag{5.2.20}$$

which corresponds to $s = 0$ in Lemma 5.2.3. Hence by applying Lemma 5.2.3 we obtain

$$\mathbb{E}\bigg(\sup_{n\leq N}\Big|\hat{X}_n - \tilde{X}_n\Big|^p\bigg) \leq K_p\mathbb{E}\Big(\Big|Z-\tilde{Z}\Big|^p\Big), \tag{5.2.21}$$

which completes the proof. **QED**

5.3 Empirical performance of the Euler-Maruyama scheme

From our previous analysis of the Euler-Maruyama scheme, there is only one item which requires empirical demonstration, which is the strong convergence for our approximate Euler-Maruyama scheme (Theorem 5.2.4). The usual the strong convergence of the Euler-Maruyama scheme (Theorem 5.1.3) and the size of the Euler-Maruyama update (Lemma 5.2.2) are already well documented by Kloeden and Platen [133, Theorem 10.2.2].

For our demonstrations we take as our underlying process in (5.1.1) a *geometric Brownian motion* such that $a(t, X_t) = \mu X_t$ and $b(t, X_t) = \sigma X_t$, where μ and σ are strictly positive constants such that (5.1.1) becomes

$$\frac{\mathrm{d}X_t}{X_t} = \mu \,\mathrm{d}t + \sigma \,\mathrm{d}W_t \,. \tag{5.3.1}$$

In our simulations we take $\mu = 0.05$, $\sigma = 0.2$, $t_0 = 0$, $T = 1$, and $X_0 = 1$.

The strong convergence for our approximate Euler-Maruyama scheme is predicted in Theorem 5.2.4. For this we require approximate random variables, and use the piecewise constant approximation from Lemma 3.2.2 in our demonstrations. For this we have already seen the behaviour of the RMSE of the approximate random variables earlier in Figure 3.2.3. Hence for Theorem 5.2.4, we take $p = 2$ and anticipate $\mathbb{E}(|\hat{X}_n - \tilde{X}_n|^2) = \mathcal{O}(\mathbb{E}(|\tilde{Z} - Z|^2))$. Taking the square root of this to obtain the RMSE we see this corresponds to $\mathrm{RMSE}(\hat{X}, \tilde{X}) = \mathcal{O}(\mathrm{RMSE}(Z, \tilde{Z}))$. Taking the approximation to have 2^q intervals, this gives rise to the approximate random variables $\tilde{Z}^{(q)}$ and approximate SDE solution $\tilde{X}^{(q)}$. We see in Figure 5.3.1 the predicted relations between the errors is as expected.

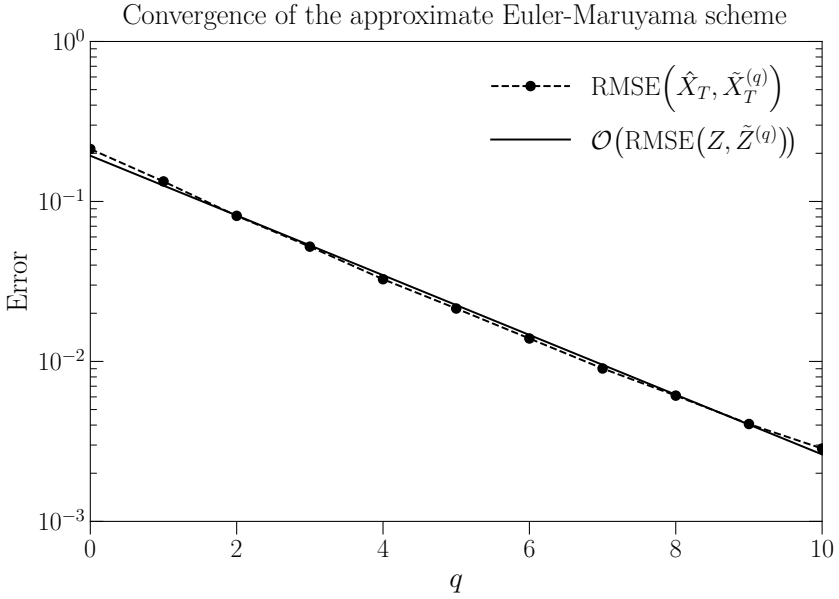


Figure 5.3.1 – The strong convergence of the approximate Euler-Maruyama scheme using a piecewise constant approximation with 2^q intervals. Recalling the power law relation from Figure 3.2.3 we show the least squares fit $\mathrm{RMSE}(\hat{X}, \tilde{X}) \approx \exp(-1.6 - 0.44q)$.

Chapter 6

Nested multilevel Monte Carlo

We begin this chapter by briefly reviewing multilevel Monte Carlo, and mention introducing reduced-precision levels into this framework, giving another route to construct multilevel frameworks. We discuss the variance structure between the different levels and how these can be coupled by varying the fidelity of the underlying approximate random variables. After highlighting the key difference between our proposed nested multilevel framework and a multi-index Monte Carlo, we present analysis for a nested multilevel Monte Carlo, giving particular attention to the multilevel correction term when introducing approximate random variables.

6.1 Multilevel Monte Carlo

The basis of *multilevel Monte Carlo* (MLMC) works on the premise that there exists some high accuracy approximation which is expensive to compute, and a lower accuracy approximation which is cheaper to compute, possibly with several intermediate strata of approximations between these two extremes. Notably, Giles [76] shows that in a finance setting, if a desired accuracy is required for a Lipschitz payoff using an Euler-Maruyama discretisation, the computational cost of achieving an accuracy $\mathcal{O}(\epsilon)$ reduces from $\mathcal{O}(\epsilon^{-3})$ for regular Monte Carlo, possibly down to $\mathcal{O}(\epsilon^{-2}\log^2(\epsilon))$ for multilevel Monte Carlo. (For a comprehensive treatment deriving these results and discussing implementations, we also recommend Lord et al. [145, 8.6, page 353] in addition to Giles [76]).

Suppose X is the solution to the stochastic differential equation (5.1.1), and that we would like to estimate the value of some function $P \equiv P(X_T)$ of the solution at some terminal time T. We will refer to P as the *payoff* function for continuity with the mathematical finance literature. Let us assume that we cannot solve (5.1.1) exactly and instead must rely on some numerical approximation $\hat{X}$, such as one obtained by the Euler-Maruyama scheme (or possibly $\tilde{X}$ for our approximate Euler-Maruyama scheme presented in Chapter 5).

Now we suppose that there are $L+1$ levels $\{0, 1, \ldots, L\}$ of approximation we can use to generate our approximate solutions, and denote the approximate payoffs as $\{\hat{P}_l\}_{l=0}^{L}$, where the expectation $\mathbb{E}(\hat{P}_L)$ is the best estimate of the true expectation $\mathbb{E}(P)$, which is the quantity of interest. We introduce multilevel Monte Carlo by presenting the telescopic sum

$$\mathbb{E}(P) \approx \mathbb{E}\Big(\hat{P}_L\Big) = \mathbb{E}\Big(\hat{P}_0\Big) + \sum_{l=1}^{L} \mathbb{E}\Big(\hat{P}_l - \hat{P}_{l-1}\Big). \tag{6.1.1}$$

For convenience let's now assume we only have one discretisation level $l = 0$, which can be computed under two different degrees of fidelity. We have already discussed altering the fidelity of the random numbers, but we also mention now the possibility the altering the fidelity

of the arithmetic precision which the computer uses for its calculations. (The analysis from altering the precision will be handled separately in Chapter 8). Outlining these in more detail:

Exact and approximate Gaussian increments Chapter 5 presents an approximation scheme using approximate Gaussian random variables $\tilde{Z}$. The appeal of this is that if these approximations come from some k-bin lookup table LT_k, or a p-th order polynomial P_p, then these can be computationally much cheaper than exact Gaussian increments Z. Hence an example could be a 1024-bin lookup table where we have $\mathbb{E}(\hat{P}_0) \rightarrow \mathbb{E}(\hat{P}_{\mathrm{LT}_{1024}}) + \mathbb{E}(\hat{P}_Z - \hat{P}_{\mathrm{LT}_{1024}})$, or even a very crude cubic polynomial approximation where $\mathbb{E}(\hat{P}_0) \rightarrow \mathbb{E}(\hat{P}_{\mathrm{cubic}}) + \mathbb{E}(\hat{P}_Z - \hat{P}_{\mathrm{cubic}})$.

Reduced floating-point precisions It can be assumed that if the simulation can be done in 32-bit single or 64-bit double-precision, then the single-precision calculation should be faster, as the single-precision calculations can be more vectorised, and typically require less intensive numerical algorithms for only 32-bit accuracy. This gives the two levels as $\mathbb{E}(\hat{P}_0) \rightarrow \mathbb{E}(\hat{P}_{32\text{-bit}}) + \mathbb{E}(\hat{P}_{64\text{-bit}} - \hat{P}_{32\text{-bit}})$. Similarly, we can extend this further to 16-bit half-precision calculations where $\mathbb{E}(\hat{P}_0) \rightarrow \mathbb{E}(\hat{P}_{16\text{-bit}}) + \mathbb{E}(\hat{P}_{32\text{-bit}} - \hat{P}_{16\text{-bit}})$. The appeal of such methods is that the speed-up in going down to half-precision should be quite considerable. The coupling between the single and half-precision should be sufficiently close that the computational cost of estimating the second expectation should be negligible compared to the first, because it will require very few samples.

Although we have introduced these types of reduced-fidelity at the lowest $l = 0$ tier of a single-level discretised multilevel framework, there is no reason to suppose that this cannot be adopted at all of the levels. Denoting the reduced-fidelity payoff $\tilde{P}$ (either coming from reduced floating-point precision, approximate random variables, or possibly a combination of both), and defining $P_{-1} \coloneqq 0$, we see that we can nest these new levels within our original multilevel framework

$$\mathbb{E}(P) \approx \sum_{l=0}^{L} \mathbb{E}\Big(\hat{P}_l - \hat{P}_{l-1}\Big) = \sum_{l=0}^{L} \mathbb{E}\Big(\tilde{P}_l - \tilde{P}_{l-1}\Big) + \sum_{l=0}^{L} \mathbb{E}\Big(\Big(\hat{P}_l - \hat{P}_{l-1}\Big) - \Big(\tilde{P}_l - \tilde{P}_{l-1}\Big)\Big). \tag{6.1.2}$$

Provided then that the variance within the second expectation is much smaller than the first, then to balance the variances from these two expectations we would require fewer samples to evaluate the second expectation, and then many more samples for the first. The first should be much cheaper to evaluate computationally if this is able to avoid the costly high-fidelity calculations of the exact inverse Gaussian cumulative distribution function $\Phi^{-1}(\cdot)$. Lastly, we should note that this framework can extend to expectations under different distributions other than the Gaussian distribution.

6.1.1 Different types of multilevel constructions

We have just mentioned in passing the various ways we can combine the speed-ups from using reduced-precision into a multilevel Monte Carlo framework, so it is helpful to explicitly list the three means by which a multilevel framework can be constructed. We are able to produce a multilevel framework by varying between the levels the discretisation, the precision, or the quantisation/approximation.

Varying the temporal discretisation and producing fine and coarse paths is the traditional means of constructing a multilevel scheme. Alternatively, we can vary the precision level the floating-point computations are performed in, as originally suggested by Giles [78]. Lastly, the distribution of the random numbers can be approximated to differing qualities, such as using $\Phi^{-1}(\cdot)$, low-order polynomials, lookup tables, etc.

Each of these three factors can be adjusted between levels to improve the computational performance, whilst contributing to the error of the multilevel estimate. Intuitively, we might expect these three forms of error to be independent and orthogonal to each other. However, we will see later in this chapter that some of these errors couple, making such an approach particularly interesting and computationally attractive.

6.1.2 Multi-index Monte Carlo

Shortly in Section 6.1.3, we will mention a comparison between our proposed nested multilevel Monte Carlo construction, and a multi-index Monte Carlo interpretation. We will find that in our proposed framework the two are not equivalent and our construction cannot be interpreted as a multi-index Monte Carlo. To this end, we will very briefly explain multi-index Monte Carlo.

Introduced by Haji-Ali et al. [99], multi-index Monte Carlo is an extension of multi-level Monte Carlo, introducing a family of possible combinations of multilevel estimator decompositions into different telescoping expressions. Haji-Ali et al. [99] introduce a set of integer multi-indices $\boldsymbol{\alpha} \in \mathbb{N}^d$ for a d-dimensional problem (counting time as a dimension just as spatial dimensions are), for which we are interested in the expectation of some functional P, where P is a solution to a partial differential equation. Discrete approximations of P are denoted $P_{\boldsymbol{\alpha}}$. The first order differencing operator Δ_i is defined as

$$\Delta_i P_{\boldsymbol{\alpha}} \coloneqq \begin{cases} P_{\boldsymbol{\alpha}} - P_{\boldsymbol{\alpha}-\boldsymbol{e}_i} & \text{if } \alpha_i > 0 \\ P_{\boldsymbol{\alpha}} & \text{if } \alpha_i = 0, \end{cases} \tag{6.1.3}$$

where $\boldsymbol{e}_i$ are canonical vectors in $\mathbb{R}^d$ such that $e_{ij} = \delta_{ij}$. Furthermore, they define the first-order mixed difference operator $\Delta \coloneqq \otimes_{i=1}^d \Delta_i \equiv \Delta_1(\otimes_{i=2}^d \Delta_i) \equiv \cdots \equiv \Delta_d(\otimes_{i=1}^{d-1} \Delta_i)$. An important feature to notice with the multi-index construction is that it assumes these individual difference operators in the definition of Δ are commutative. (In our nested multilevel construction with approximate random variables, we will give an example showing this is no longer the case).

Haji-Ali et al. [99] let $\Delta\mathscr{P}_{\boldsymbol{\alpha}}$ be an unbiased estimator for $\Delta P_{\boldsymbol{\alpha}}$. (A trivial example is $\Delta\mathscr{P}_{\boldsymbol{\alpha}} \coloneqq \Delta P_{\boldsymbol{\alpha}}$, and a less trivial example is an antithetic estimator such as that proposed by Giles and Szpruch [85]). Their multi-index Monte Carlo estimator is then $\sum_{\boldsymbol{\alpha}\in\mathcal{I}} \frac{1}{M_{\boldsymbol{\alpha}}} \sum^{M_{\boldsymbol{\alpha}}} \Delta\mathscr{P}_{\alpha}$ for some set of indices $\mathcal{I} \subset \mathbb{N}^d$, where $M_{\boldsymbol{\alpha}}$ simulations are performed for a given multi-index $\boldsymbol{\alpha}$.

6.1.3 Nested multilevel and not multi-index

Those familiar with multi-index Monte Carlo [99] might be thinking that having all these possible means of producing levels corresponds to different indices in a multi-index Monte Carlo? While this is tempting, there is a subtle difference which is elucidated in Remark 6.2.3.1, meaning that our proposed framework is instead a nested multilevel Monte Carlo set-up, and not a multi-index Monte Carlo.

6.2 A nested multilevel analysis

The goal of this section will be to analyse the expectation, variance, and higher moments of the multilevel Monte Carlo correction term and the quantisation term arising in the nested multilevel framework from (6.1.2). We will consider an underlying process following the SDE from (5.1.1) and again use Assumptions 5.1.1 to 5.1.6. Towards the end of this section we will extend this analysis to the expectations of functionals of the underlying process by considering various classes of payoff functions.

For the multilevel analysis we will often be considering the difference between two sets of levels, producing a difference of differences, which we call a four-way difference. To make progress here we will require a four-point adaptation of the mean value theorem (Theorem A.1.5).

Definition 6.2.1 (Lipschitz continuity with polynomial growth). A function $f\colon \mathbb{R} \to \mathbb{R}$ is *Lipschitz continuous with polynomial growth* of order r if there exists an exponent $r \geq 0$ and constant $K > 0$ such that $|f(x) - f(y)| \leq K(1 + |x|^r + |y|^r)|x - y|$ for all $x, y \in \mathbb{R}$.

Lemma 6.2.1 (the four-point mean value theorem). *For a function $f\colon \mathbb{R} \to \mathbb{R}$ which is $C^1(\mathbb{R})$, and where f' is Lipschitz continuous with Lipschitz constant L', then for the positions x_1, x_2, x_3, and x_4, there exists a positively weighted average $\xi \in [\min\{x_i\}_{i=1}^4, \max\{x_i\}_{i=1}^4]$ such that $f(x_1) - f(x_2) - f(x_3) + f(x_4) = (x_1 - x_2 - x_3 + x_4)f'(\xi) + R$, where R can be bound by $|R| \leq \frac{1}{2}L'(|x_1 - x_2| + |x_3 - x_4|)(|x_1 - x_3| + |x_2 - x_4|)$.*

Proof. Without loss of generality we can assume that under an appropriate labelling of the points x_2 and x_3 that $|x_1 - x_2| + |x_3 - x_4| \leq |x_1 - x_3| + |x_2 - x_4|$. From the mean value theorem (Theorem A.1.5) we have $f(x_1) - f(x_2) = (x_1 - x_2)f'(\xi_1)$ and $f(x_3) - f(x_4) = (x_3 - x_4)f'(\xi_2)$, where ξ_1 and ξ_2 are positive weighted averages of $\{x_1, x_2\}$ and $\{x_3, x_4\}$ respectively. Taking the difference of these we obtain the four-way difference

$$f(x_1) - f(x_2) - f(x_3) + f(x_4) = (x_1 - x_2)f'(\xi_1) - (x_3 - x_4)f'(\xi_2) \tag{6.2.1}$$

$$= \frac{1}{2}(x_1 - x_2 - x_3 + x_4)(f'(\xi_1) + f'(\xi_2)) + R \tag{6.2.2}$$

where $R = \frac{1}{2}(x_1 - x_2 + x_3 - x_4)(f'(\xi_1) - f'(\xi_2))$. As f' is continuous then by the intermediate value theorem (Theorem A.1.4) there exists a ξ which is a positively weighted average of ξ_1 and ξ_2, (and hence of x_1, x_2, x_3, and x_4), such that $\frac{1}{2}(f'(\xi_1) + f'(\xi_2)) = f'(\xi)$.

We now look to apply the Lipschitz condition to f' in our expression for R. For this we note that $\xi_1 - \xi_2$ can be expressed in terms of the differences from the arithmetic means of $\{x_1, x_2\}$ and $\{x_3, x_4\}$, giving

$$\xi_1 - \xi_2 = \left(\xi_1 - \frac{x_1 + x_2}{2}\right) - \left(\xi_2 - \frac{x_3 + x_4}{2}\right) + \frac{x_1 - x_3 + x_2 - x_4}{2} \tag{6.2.3}$$

taking the absolute values and our initial assumption about the differences we obtain

$$|\xi_1 - \xi_2| \leq \left|\xi_1 - \frac{x_1 + x_2}{2}\right| + \left|\xi_2 - \frac{x_3 + x_4}{2}\right| + \frac{|x_1 - x_3|}{2} + \frac{|x_2 - x_4|}{2} \tag{6.2.4}$$

$$\leq \tfrac{1}{2}(|x_1 - x_2| + |x_3 - x_4| + |x_1 - x_3| + |x_2 - x_4|) \tag{6.2.5}$$

using $|x_1 - x_2| + |x_3 - x_4| \leq |x_1 - x_3| + |x_2 - x_4|$

$$\leq |x_1 - x_3| + |x_2 - x_4|. \tag{6.2.6}$$

So applying the Lipschitz condition to f' in $|R|$ immediately gives $|R| \leq \frac{1}{2}L'(|x_1 - x_2| + |x_3 - x_4|)(|x_1 - x_3| + |x_2 - x_4|)$, completing the proof. **QED**

Corollary 6.2.1.1 (the four-point mean value theorem with polynomial growth). *For a function $f\colon \mathbb{R} \to \mathbb{R}$ which is $C^1(\mathbb{R})$, where f' is Lipschitz continuous with polynomial growth of order r and Lipschitz coefficient L', then for the four positions x_1, x_2, x_3, and x_4, there exists a positively weighted average $\xi \in [\min\{x_i\}_{i=1}^4, \max\{x_i\}_{i=1}^4]$ such that the four-way difference $f(x_1) - f(x_2) - f(x_3) + f(x_4) = (x_1 - x_2 - x_3 + x_4)f'(\xi) + R$, where R is bound by $|R| \leq \frac{1}{2}L'(1 + |x_1|^r + |x_2|^r + |x_3|^r + |x_4|^r)(|x_1 - x_2| + |x_3 - x_4|)(|x_1 - x_3| + |x_2 - x_4|)$.*

Proof. This follows identically as that for the four-point mean value theorem (Lemma 6.2.1) with the final inclusion of the polynomial growth terms when applying the Lipschitz continuity condition. **QED**

6.2.1 Fine and coarse paths

We can recall from Section 5.1 that we were able to write down the Euler-Maruyama scheme as (5.1.2) in terms of Gaussian random variables Z. However, when we have to construct both a fine and a coarse path from the same underlying Brownian motion, we will see that it is helpful to recast (5.1.2) in terms of the Brownian increments ΔW, giving

$$\hat{X}_{n+1} = \hat{X}_n + a(\tau_n, \hat{X}_n)\delta + b(\tau_n, \hat{X}_n)\Delta W_n, \tag{6.2.7}$$

where $\Delta W_n \coloneqq \sqrt{\delta}Z_n$. Now if we wish to construct fine and coarse path approximations, we need these both to be driven by the same underlying Brownian motion. For the sake of simplicity we will consider a fine path which has a discretisation which is twice as fine as the coarse path. Following the half-index notation used by Giles and Szpruch [85, Section 3], we define the Euler-Maruyama schemes for the coarse and fine path approximations $\hat{X}^{\mathrm{c}}$ and $\hat{X}^{\mathrm{f}}$ as

$$\hat{X}^{\mathrm{f}}_{n+\frac{1}{2}} = \hat{X}^{\mathrm{f}}_n + a(\tau_n, \hat{X}^{\mathrm{f}}_n)\delta_{\mathrm{f}} + b(\tau_n, \hat{X}^{\mathrm{f}}_n)\Delta W^{\mathrm{f}}_n \tag{6.2.8}$$

and

$$\hat{X}^{\mathrm{c}}_{n+1} = \hat{X}^{\mathrm{c}}_n + a(\tau_n, \hat{X}^{\mathrm{c}}_n)\delta_{\mathrm{c}} + b(\tau_n, \hat{X}^{\mathrm{c}}_n)\Delta W^{\mathrm{c}}_n, \tag{6.2.9}$$

where $\delta_{\mathrm{c}} \coloneqq 2\delta_{\mathrm{f}}$, $\Delta W^{\mathrm{f}}_n \coloneqq \sqrt{\delta_{\mathrm{f}}}Z_n$, and the coarse path is coupled to the fine path by the coarse path Brownian construction $\Delta W^{\mathrm{c}}_n \coloneqq \Delta W^{\mathrm{f}}_n + \Delta W^{\mathrm{f}}_{n+\frac{1}{2}}$. As ΔW^{f} is constructed using exact Gaussian increments, we have that ΔW^{f} and ΔW^{c} follow the same type of distribution and $\Delta W^{\mathrm{c}} \overset{\mathrm{d}}{\sim} \sqrt{2}\Delta W^{\mathrm{f}}$.

Lemma 6.2.2. *For the coarse path approximation $\hat{X}^{\mathrm{c}}$ we have $\mathbb{E}(|\hat{X}^{\mathrm{c}}|^p) < \infty$ and $\mathbb{E}(\sup_{n\leq N}|\hat{X}^{\mathrm{c}}_n - X_n|^p) < K\delta^{p/2}$.*

Proof. From the definition of ΔW^{c}_n we know $\Delta W^{\mathrm{c}} \overset{\mathrm{d}}{\sim} \sqrt{2}\Delta W^{\mathrm{f}}$, and hence we can define the coarse increment $Z^{\mathrm{c}}_n \coloneqq \frac{Z_n + Z_{n+1/2}}{\sqrt{2}}$, where $\Delta W^{\mathrm{c}} \equiv \sqrt{\delta_{\mathrm{c}}}Z^{\mathrm{c}}_n$. Immediately we have $Z^{\mathrm{c}} \sim \mathcal{N}(0,1)$, and hence the coarse path can be recognised as updating by an Euler-Maruyama scheme. Hence our previous results from Theorem 5.1.3 and Lemma 5.2.1 hold true also for the coarse path, completing the proof. **QED**

With this construction, we can see how closely coupled the fine path is to the coarse path in Lemma 6.2.3.

Lemma 6.2.3. *Under Assumptions 5.1.1 to 5.1.6, then for any integer $p \geq 2$, the strong error between the coarse and fine paths constructed using (6.2.8) and (6.2.9) scales as $\mathbb{E}(\sup_{n\leq N}|\hat{X}^{\mathrm{f}}_n - \hat{X}^{\mathrm{c}}_n|^p) \leq C\delta_{\mathrm{f}}^{p/2}$, for some constant C.*

Proof.

$$\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}^{\mathrm{f}}_n - \hat{X}^{\mathrm{c}}_n\right|^p\right) = \mathbb{E}\left(\sup_{n\leq N}\left|\left(\hat{X}^{\mathrm{f}}_n - X_n\right) - \left(\hat{X}^{\mathrm{c}}_n - X_n\right)\right|^p\right) \tag{6.2.10}$$

$$\leq 2^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}^{\mathrm{f}}_n - X_n\right|^p\right) + 2^{p-1}\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}^{\mathrm{c}}_n - X_n\right|^p\right) \tag{6.2.11}$$

using Theorem 5.1.3 and Lemma 6.2.2

$$\leq C_1\delta_{\mathrm{f}}^{p/2} + C_2\delta_{\mathrm{c}}^{p/2} \tag{6.2.12}$$

$$\leq C\delta_{\mathrm{f}}^{p/2}. \tag{6.2.13}$$

QED

The next stage is to adapt (6.2.8) and (6.2.9) to use approximate Gaussian increments $\tilde{Z}$ and ensure we can recover Lemma 6.2.3 and similar results associated with the exact schemes. To this end we define the approximate multilevel Euler-Maruyama schemes

$$\tilde{X}^{\mathrm{f}}_{n+\frac{1}{2}} = \tilde{X}^{\mathrm{f}}_n + a(\tau_n, \tilde{X}^{\mathrm{f}}_n)\delta_{\mathrm{f}} + b(\tau_n, \tilde{X}^{\mathrm{f}}_n)\Delta\tilde{W}^{\mathrm{f}}_n \tag{6.2.14}$$

and

$$\tilde{X}^{\mathrm{c}}_{n+1} = \tilde{X}^{\mathrm{c}}_n + a(\tau_n, \tilde{X}^{\mathrm{c}}_n)\delta_{\mathrm{c}} + b(\tau_n, \tilde{X}^{\mathrm{c}}_n)\Delta\tilde{W}^{\mathrm{c}}_n, \tag{6.2.15}$$

where $\Delta\tilde{W}^{\mathrm{f}}_n \coloneqq \sqrt{\delta_{\mathrm{f}}}\tilde{Z}_n$, and the coarse path is coupled to the fine path by the coarse path Brownian construction $\Delta\tilde{W}^{\mathrm{c}}_n \coloneqq \Delta\tilde{W}^{\mathrm{f}}_n + \Delta\tilde{W}^{\mathrm{f}}_{n+1/2}$.

Remark 6.2.3.1. A key distinction which arises here is that because $\Delta\tilde{W}^{\mathrm{f}} \overset{\mathrm{d}}{\sim} \mathcal{N}(0, \delta_{\mathrm{f}})$, then when we combine the two fine increments to produce the coarse increment, these do not follow the same distribution and $\Delta\tilde{W}^{\mathrm{c}} \overset{\mathrm{d}}{\sim} \sqrt{2}\Delta\tilde{W}^{\mathrm{f}}$. This is a rather subtle and very important point which the reader should appreciate, as this is what will stop this framework from being formulated as a multi-index Monte Carlo.

To highlight the practical implications of this consider the quantised multilevel framework from (6.1.2), where to make the example more concrete we specifically consider without loss of generality the $l = 2$ discretisation level. The $l = 2$ level will be the coarse level when coupled with the $l = 3$ level, and conversely will be the fine level when coupled with the $l = 1$ level. This means that the quantised multilevel Monte Carlo framework will compute $\{\tilde{P}^{\mathrm{f}}_2 - \tilde{P}^{\mathrm{c}}_1, (\hat{P}^{\mathrm{f}}_2 - \hat{P}^{\mathrm{c}}_1) - (\tilde{P}^{\mathrm{f}}_2 - \tilde{P}^{\mathrm{c}}_1)\}$ and $\{\tilde{P}^{\mathrm{f}}_3 - \tilde{P}^{\mathrm{c}}_2, (\hat{P}^{\mathrm{f}}_3 - \hat{P}^{\mathrm{c}}_2) - (\tilde{P}^{\mathrm{f}}_3 - \tilde{P}^{\mathrm{c}}_2)\}$, where we have made explicitly clear which are the fine and coarse levels. Multi-index Monte Carlo would assume (and require) that for an underlying Brownian motion path sample ω that the associated payoff $P|_\omega$ and its approximate Euler-Maruyama approximation $\tilde{P}_2|_\omega$ on the $l = 2$ level must be equal when the $l = 2$ is either the fine or coarse path and that $\tilde{P}^{\mathrm{f}}_2|_\omega = \tilde{P}^{\mathrm{c}}_2|_\omega$. Our quantised multilevel Monte Carlo framework, which is a nested multilevel Monte Carlo, does not require this.

Lemma 6.2.4. *Under the same assumptions as Theorem 5.2.4, where in Assumptions 5.1.1 to 5.1.6 the constants do not depend on δ_{f}, with the approximate coarse paths constructed in (6.2.15) using the approximate Brownian increments $\Delta\tilde{W}^{\mathrm{c}}_n$, then there exists a finite constant $0 < K_p < \infty$, independent of N, δ_{f}, and q, but dependent on p, such that*

$$\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}^{\mathrm{c}}_n - \tilde{X}^{\mathrm{c}}_n\right|^p\right) \leq K_p\mathbb{E}\left(\left|\tilde{Z} - Z\right|^p\right) \tag{6.2.16}$$

for any integer $p \geq 2$.

Proof. For this it will be sufficient to show that the implicit approximate Gaussian random variable corresponding to $\Delta\tilde{W}^{\mathrm{c}}_n$ satisfies the requirements of Theorem 5.2.4. Hence consider the n-th approximate time increment where

$$\begin{aligned}
\Delta\tilde{W}^{\mathrm{c}}_n &\equiv \Delta\tilde{W}^{\mathrm{f}}_n + \Delta\tilde{W}^{\mathrm{f}}_{n+\frac{1}{2}} && (6.2.17)\\
&\equiv \sqrt{\delta_{\mathrm{f}}}\left(\tilde{Z}_n + \tilde{Z}_{n+\frac{1}{2}}\right) && (6.2.18)\\
&\equiv \sqrt{\delta_{\mathrm{c}}}\left(\frac{\tilde{Z}_n + \tilde{Z}_{n+\frac{1}{2}}}{\sqrt{2}}\right) && (6.2.19)\\
&\equiv \sqrt{\delta_{\mathrm{c}}}\tilde{Z}^{\mathrm{c}}_n, && (6.2.20)
\end{aligned}$$

where we have defined the random variable $\tilde{Z}^{\mathrm{c}}_n$ such that $\tilde{Z}^{\mathrm{c}}_n \coloneqq \frac{\tilde{Z}_n + \tilde{Z}_{n+1/2}}{\sqrt{2}}$. We take care to note that as $\tilde{Z}_n$ and $\tilde{Z}_{n+1/2}$ are approximate Gaussian random variables, then $\tilde{Z}^{\mathrm{c}} \overset{\mathrm{d}}{\sim} \tilde{Z}$.

At this point we can readily show from Jensen's inequality for summations (Lemma A.1.7) that under Assumption 5.2.3 that $\tilde{Z}^{\mathrm{c}}$ has zero mean and its p-th moment is finite for all $1 \leq p < \infty$. This then satisfies the requirements for Theorem 5.2.4, from which (6.2.16) immediately follows, completing the proof. **QED**

Lemma 6.2.5. *Under Assumptions 5.1.1 to 5.1.6, then for any integer $p \geq 2$, the strong error between coarse and fine paths constructed using (6.2.14) and (6.2.15) scales as $\mathbb{E}(\sup_{n \leq N} |\tilde{X}_n^{\mathrm{f}} - \tilde{X}_n^{\mathrm{c}}|^p) \leq C\delta_{\mathrm{f}}^{p/2}$, for some constant C.*

Proof. By defining the difference process $\mathcal{E}_n \coloneqq \tilde{X}_n^{\mathrm{c}} - \tilde{X}_n^{\mathrm{f}}$ and by taking the difference between (6.2.14) and (6.2.15) we obtain

$$\begin{aligned}
\mathcal{E}_{m+1} = \mathcal{E}_m + \delta_{\mathrm{c}} \underbrace{\left(a\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - a\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right)\right)}_{\mathcal{A}_m} + \sqrt{\delta_{\mathrm{c}}} \underbrace{\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right)\right)}_{\mathcal{B}_m} \tilde{Z}_m^{\mathrm{c}} \qquad (6.2.21) \\
\left.\begin{aligned}
&+ \sqrt{\delta_{\mathrm{f}}}\left(b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right)\tilde{Z}_{m+\frac{1}{2}} \\
&+ \sqrt{\delta_{\mathrm{f}}}\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right)\tilde{Z}_{m+\frac{1}{2}}
\end{aligned}\right\} \Xi_m \\
\left.\begin{aligned}
&+ \delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right) \\
&+ \delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right),
\end{aligned}\right\} \Theta_m
\end{aligned}$$

where we have indicated the correspondence to the equivalent terms in Lemma 5.2.3. The $\mathcal{A}_m$ and $\mathcal{B}_m$ terms are immediately bound in a suitable manner by using Assumption 5.1.2.

It only remains to bound the terms constituting Ξ_m and Θ_m, which because of Jensen's inequality for summations (Lemma A.1.7) it suffices to bound each individually. Considering the first non-martingale term contributing to Θ_m, we can see from Assumption 5.1.6 that we obtain

$$\mathbb{E}\left(\left|\delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right)\right|^p\right) \leq K\delta_{\mathrm{f}}^p \mathbb{E}\left(1 + \left|\tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right|^p\right)\delta_{\mathrm{f}}^{\frac{p}{2}} \qquad (6.2.22)$$

using Lemma 5.2.1

$$\leq K_p \delta_{\mathrm{f}}^{\frac{3p}{2}}, \qquad (6.2.23)$$

which corresponds to $s = \frac{1}{2}$ in Lemma 5.2.3. Similarly for the second term contributing to Θ_m, a combination of Assumption 5.1.2 and Lemma 5.2.2 yields

$$\mathbb{E}\left(\left|\delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right)\right|^p\right) \leq K\delta_{\mathrm{f}}^{\frac{3p}{2}}. \qquad (6.2.24)$$

The terms contributing to Ξ_m are equivalent to those forming Θ_m, only with $a \to b$, $\delta_{\mathrm{f}} \to \sqrt{\delta_{\mathrm{f}}}$, and the introduction of a $\tilde{Z}$ term. As $\tilde{Z}_{m+1/2}$ is an independent random variable we can use Assumption 5.2.2 to give similar bounds to those obtained for Θ_m, but where both are instead bounded by δ_{f}^p (rather than $\delta_{\mathrm{f}}^{3p/2}$), which for the martingale term again corresponds to $s = \frac{1}{2}$. Using our bounds with $s = \frac{1}{2}$, we can apply Lemma 5.2.3, from which the result immediately follows. **QED**

With this result now at hand, we can turn our attention to the variance of the multilevel correction term, for which we will require a few additional assumptions on the smoothness of the drift and volatility functions.

Assumption 6.2.1 (existence of spatial partial derivatives). The partial derivatives $\frac{\partial a}{\partial X}$ and $\frac{\partial b}{\partial X}$ exist everywhere and are continuous.

Assumption 6.2.2 (Lipschitz continuity of spatial partial derivatives). There exists a constant $K > 0$ such that $|\frac{\partial a}{\partial X}(t,x) - \frac{\partial a}{\partial X}(t,y)| \leq K|x-y|$ for all $t \in [t_0, T]$ and $x, y \in \mathbb{R}$, and identically for $\frac{\partial b}{\partial X}$.

Assumption 6.2.3 (temporal Hölder continuity with linear growth for spatial partial derivatives). The partial derivatives $\frac{\partial a}{\partial X}$ and $\frac{\partial b}{\partial X}$ also follow the Hölder continuity condition with bounded growth, where there exists a constant $K > 0$ such that $|\frac{\partial a}{\partial X}(t,x) - \frac{\partial a}{\partial X}(s,x)| \leq K(1+|x|)\sqrt{|t-s|}$ for all $s, t \in [t_0, T]$ and $x \in \mathbb{R}$, and identically for $\frac{\partial b}{\partial X}$.

Remark 6.2.5.1. Assumption 6.2.1 nearly follows from applying Rademacher's theorem (Theorem A.1.2) to Assumption 5.1.2, but would only imply the existence of $\frac{\partial a}{\partial X}$ and $\frac{\partial b}{\partial X}$ *almost everywhere*. However, as we will be requiring further convenient properties from the partial derivatives $\frac{\partial a}{\partial x}$ and $\frac{\partial b}{\partial x}$ in Assumption 6.2.3 it will be necessary to assume their existence and continuity everywhere.

Theorem 6.2.6. *Under Assumptions 5.1.1 to 5.1.6 and Assumptions 5.2.1, 5.2.2, and 6.2.1 to 6.2.3, then the coarse, fine, exact, and approximate Euler-Maruyama approximations from (6.2.8) to (6.2.9) and (6.2.14) to (6.2.15) gives*

$$\mathbb{E}\left(\sup_{n \leq N} \left|\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \tilde{X}_n^{\mathrm{f}} + \tilde{X}_n^{\mathrm{c}}\right|^p\right) \leq C\delta_{\mathrm{c}}^{\frac{p}{2}} \mathbb{E}\left(\left|Z - \tilde{Z}\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}} \tag{6.2.25}$$

for any integer $p \geq 2$ and real number $\epsilon \in (0, \infty)$, where C is some finite constant which depends only on p and ϵ.

Proof. Defining the four-way difference $\mathcal{E}_n \coloneqq \hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \tilde{X}_n^{\mathrm{f}} + \tilde{X}_n^{\mathrm{c}}$, then taking the difference between (6.2.8), (6.2.9), (6.2.14), and (6.2.15) we obtain

$$\begin{aligned}
\mathcal{E}_{m+1} = \mathcal{E}_m &+ \delta_{\mathrm{c}}\left(a\left(\tau_m, \hat{X}_m^{\mathrm{f}}\right) - a\left(\tau_m, \hat{X}_m^{\mathrm{c}}\right) - a\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) + a\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right)\right) \\
&+ \delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - a\left(\tau_m, \hat{X}_m^{\mathrm{f}}\right) - a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + a\left(\tau_m, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right) \\
&+ \delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_m, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + a\left(\tau_{m+\frac{1}{2}}, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right) \\
&+ \sqrt{\delta_{\mathrm{f}}}\left(Z_m + Z_{m+\frac{1}{2}}\right)\left(b\left(\tau_m, \hat{X}_m^{\mathrm{f}}\right) - b\left(\tau_m, \hat{X}_m^{\mathrm{c}}\right) - b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) + b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right)\right) \\
&+ \sqrt{\delta_{\mathrm{f}}}\left(\tilde{Z}_m - Z_m\right)\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right)\right) \\
&+ \sqrt{\delta_{\mathrm{f}}}\left(\tilde{Z}_{m+\frac{1}{2}} - Z_{m+\frac{1}{2}}\right)\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right) \\
&+ \sqrt{\delta_{\mathrm{f}}} Z_{m+\frac{1}{2}}\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - b\left(\tau_m, \hat{X}_m^{\mathrm{f}}\right) - b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + b\left(\tau_m, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right) \\
&+ \sqrt{\delta_{\mathrm{f}}} Z_{m+\frac{1}{2}}\left(b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - b\left(\tau_m, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + b\left(\tau_{m+\frac{1}{2}}, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right).
\end{aligned} \tag{6.2.26}$$

If we then apply the four-point mean value theorem (Lemma 6.2.1) for all the four-way differences, where the difference is at the same time point, then we can begin to write this in a form suitable for Lemma 5.2.3, obtaining

$$= \mathcal{E}_m + \delta_{\mathrm{c}} \underbrace{\frac{\partial a}{\partial X}(\tau_m, \xi_{1,m})\mathcal{E}_m}_{\mathcal{A}_m} + \sqrt{\delta_{\mathrm{f}}}\left(Z_m + Z_{m+\frac{1}{2}}\right) \underbrace{\frac{\partial b}{\partial X}(\tau_m, \xi_{3,m})\mathcal{E}_m}_{\mathcal{B}_m} + \Xi_m + \Theta_m, \tag{6.2.27}$$

where

$$\begin{aligned}
\Theta_m = \delta_{\mathrm{c}} R_{1,m} &+ \delta_{\mathrm{f}} \frac{\partial a}{\partial X}(\tau_m, \xi_{2,m})\left(\tilde{X}_m^{\mathrm{f}} - \hat{X}_m^{\mathrm{f}} - \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}} + \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + \delta_{\mathrm{f}} R_{2,m} \\
&+ \delta_{\mathrm{f}}\left(a\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - a\left(\tau_m, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) + a\left(\tau_{m+\frac{1}{2}}, \hat{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right),
\end{aligned} \tag{6.2.28}$$

and

$$\begin{aligned}\Xi_m = {} & \sqrt{\delta_{\mathrm{f}}}\Big(Z_m + Z_{m+\frac{1}{2}}\Big)R_{3,m} + \sqrt{\delta_{\mathrm{f}}}\Big(\tilde{Z}_m - Z_m\Big)\Big(b\Big(\tau_m, \tilde{X}^{\mathrm{c}}_m\Big) - b\Big(\tau_m, \tilde{X}^{\mathrm{f}}_m\Big)\Big) \\ & + \sqrt{\delta_{\mathrm{f}}}\Big(\tilde{Z}_{m+\frac{1}{2}} - Z_{m+\frac{1}{2}}\Big)\Big(b\Big(\tau_m, \tilde{X}^{\mathrm{c}}_m\Big) - b\Big(\tau_{m+\frac{1}{2}}, \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big) \\ & + \sqrt{\delta_{\mathrm{f}}}Z_{m+\frac{1}{2}}\frac{\partial b}{\partial X}(\tau_m, \xi_{4,m})\Big(\tilde{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}} + \hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big) + \sqrt{\delta_{\mathrm{f}}}Z_{m+\frac{1}{2}}R_{4,m} \\ & + \sqrt{\delta_{\mathrm{f}}}Z_{m+\frac{1}{2}}\Big(b\Big(\tau_m, \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big) - b\Big(\tau_{m+\frac{1}{2}}, \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big) - b\Big(\tau_m, \hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big) + b\Big(\tau_{m+\frac{1}{2}}, \hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big),\end{aligned} \tag{6.2.29}$$

for suitably defined $\{\xi_{i,m}\}_{i=1}^4$, and where the remainder terms are bounded by

$$|R_{1,m}| \leq K_1\Big(\Big|\hat{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{c}}_m\Big| + \Big|\tilde{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{c}}_m\Big|\Big)\Big(\Big|\hat{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{f}}_m\Big| - \Big|\hat{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{c}}_m\Big|\Big), \tag{6.2.30}$$

$$|R_{2,m}| \leq K_2\Big(\Big|\tilde{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{f}}_m\Big| + \Big|\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}} - \hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|\Big)\Big(\Big|\tilde{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big| + \Big|\hat{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|\Big), \tag{6.2.31}$$

$$|R_{3,m}| \leq K_3\Big(\Big|\hat{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{c}}_m\Big| + \Big|\tilde{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{c}}_m\Big|\Big)\Big(\Big|\hat{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{f}}_m\Big| + \Big|\hat{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{c}}_m\Big|\Big), \tag{6.2.32}$$

and

$$|R_{4,m}| \leq K_4\Big(\Big|\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}} - \hat{X}^{\mathrm{f}}_m\Big| + \Big|\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}} - \tilde{X}^{\mathrm{f}}_m\Big|\Big)\Big(\Big|\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}} - \tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big| + \Big|\hat{X}^{\mathrm{f}}_m - \tilde{X}^{\mathrm{f}}_m\Big|\Big). \tag{6.2.33}$$

The criteria in Lemma 5.2.3 required for the $\mathcal{A}_m$ and $\mathcal{B}_m$ terms immediately follows from Assumption 6.2.2. Hence, by making use of Jensen's inequality for summations (Lemma A.1.7), it will be sufficient to individually bound each term constituting Ξ_m and Θ_m. Beginning with the easier of these, namely Θ_m, we begin by bounding the $R_{1,m}$ term, where by Jensen's inequality for summations (Lemma A.1.7) we obtain

$$\begin{aligned}\mathbb{E}(|R_{1,m}|^p) \leq {} & C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^p\Big|\tilde{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{f}}_m\Big|^p\Big) + C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{f}}_m\Big|^p\Big|\tilde{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{f}}_m\Big|^p\Big) \\ & + C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^p\Big|\hat{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{f}}_m\Big|^p\Big) + C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{f}}_m - \hat{X}^{\mathrm{f}}_m\Big|^p\Big|\hat{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{f}}_m\Big|^p\Big).\end{aligned} \tag{6.2.34}$$

Using Hölder's inequality for expectations (Lemma A.2.3) allows us to split these expectations into products of discretisation and quantisation terms with Hölder coefficients $r_1, r_2 \in (1, \infty)$ (with $r_1^{-1} + r_2^{-1} = 1$). Considering the first of these expectations

$$\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^p\Big|\tilde{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{f}}_m\Big|^p\Big) \leq \mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^{pr_1}\Big)^{\frac{1}{r_1}}\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \tilde{X}^{\mathrm{f}}_m\Big|^{pr_2}\Big)^{\frac{1}{r_2}} \tag{6.2.35}$$

using Lemma 6.2.5

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^{pr_1}\Big)^{\frac{1}{r_1}}\Big(\delta_{\mathrm{c}}^{\frac{pr_2}{2}}\Big)^{\frac{1}{r_2}} \tag{6.2.36}$$

letting $r_1 \coloneqq 1 + \epsilon$ where $0 < \epsilon \ll 1$

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{c}}_m - \hat{X}^{\mathrm{c}}_m\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}\delta_{\mathrm{c}}^{\frac{p}{2}} \tag{6.2.37}$$

using Lemma 6.2.4

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}\delta_{\mathrm{c}}^{\frac{p}{2}}. \tag{6.2.38}$$

By appropriately interchanging the use of Lemma 6.2.4 with Theorem 5.2.4, and Lemma 6.2.5 with Lemma 6.2.3, identical bounds for the remaining expectations constituting $R_{1,m}$ follow identically. Furthermore, all of the remainder terms can be similarly split into the expectations of discretisation and quantisation terms. Thus each $\{R_{i,m}\}_{i=1}^4$ follows the same bound as we found for $R_{1,m}$.

Consider then the second term in Θ_m, which from Assumption 5.1.2 we know $|\frac{\partial a}{\partial X}|$ is bounded, and hence we obtain

$$\mathbb{E}\Big(\Big|\delta_{\mathrm{f}}\frac{\partial a}{\partial X}(\tau_m,\xi_{2,m})\Big(\tilde{X}^{\mathrm{f}}_m-\hat{X}^{\mathrm{f}}_m-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big|^p\Big)$$
$$\leq C_p\delta_{\mathrm{f}}^p\mathbb{E}\Big(\Big|\tilde{X}^{\mathrm{f}}_m-\hat{X}^{\mathrm{f}}_m-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|^p\Big) \tag{6.2.39}$$

using (6.2.8) and (6.2.14)

$$\leq C_p\delta_{\mathrm{f}}^{2p}\mathbb{E}\Big(\Big|a\Big(\tau_m,\hat{X}^{\mathrm{f}}_m\Big)-a\Big(\tau_m,\tilde{X}^{\mathrm{f}}_m\Big)\Big|^p\Big) \tag{6.2.40}$$
$$+C_p\delta_{\mathrm{f}}^{\frac{3p}{2}}\mathbb{E}\Big(\Big|b\Big(\tau_m,\hat{X}^{\mathrm{f}}_m\Big)Z_m-b\Big(\tau_m,\tilde{X}^{\mathrm{f}}_m\Big)\tilde{Z}_m\Big|^p\Big)$$

using Assumption 5.1.2 and Theorem 5.2.4, both the expectations of the drift and volatility terms are bounded by $\mathbb{E}(|Z-\tilde{Z}|^{p})$, giving

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^p\Big)\Big(\delta_{\mathrm{c}}^{2p}+\delta_{\mathrm{c}}^{\frac{3p}{2}}\Big) \tag{6.2.41}$$

using Lyapunov's inequality (Lemma A.2.4) we can bound the lower order moment $\mathbb{E}(|\tilde{Z}-Z|^{p})\leq\mathbb{E}(|\tilde{Z}-Z|^{p(1+\epsilon)})^{(1+\epsilon)^{-1}}$, giving

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}\Big(\delta_{\mathrm{c}}^{2p}+\delta_{\mathrm{c}}^{\frac{3p}{2}}\Big) \tag{6.2.42}$$

and for a suitably scaled interval we have $\delta_{\mathrm{c}}^{\alpha}\geq\delta_{\mathrm{c}}^{\beta}$ for $0<\alpha<\beta$, and so

$$\leq C_p\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}\delta_{\mathrm{c}}^{\frac{3p}{2}}. \tag{6.2.43}$$

Moving to the final term appearing in Θ_m we use the fundamental theorem of calculus for Lebesgue integrals (Theorem A.1.3) to obtain

$$\mathbb{E}\Big(\Big|\delta_{\mathrm{f}}\Big(a\Big(\tau_m,\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)-a\Big(\tau_{m+\frac{1}{2}},\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)-a\Big(\tau_m,\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)+a\Big(\tau_{m+\frac{1}{2}},\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big)\Big|^p\Big)$$
$$=\delta_{\mathrm{f}}^p\mathbb{E}\Bigg(\Bigg|\int_0^1\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\frac{\partial a}{\partial X}\Big(\tau_m,\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\lambda\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big)\,\mathrm{d}\lambda$$
$$-\int_0^1\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\frac{\partial a}{\partial X}\Big(\tau_{m+\frac{1}{2}},\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\lambda\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big)\,\mathrm{d}\lambda\Bigg|^p\Bigg) \tag{6.2.44}$$

interchanging the integration and the absolute function we can then use Assumption 6.2.3

$$\leq C_pK\delta_{\mathrm{f}}^{\frac{3p}{2}}\mathbb{E}\Bigg(\Big|\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|^p\int_0^1\Big(1+\Big|\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\lambda\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big|\Big)^p\,\mathrm{d}\lambda\Bigg) \tag{6.2.45}$$

using Hölder's inequality for expectations (Lemma A.2.3)

$$\leq C_pK\delta_{\mathrm{f}}^{\frac{3p}{2}}\mathbb{E}\Bigg(\Big|\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|^{p(1+\epsilon)}\Bigg)^{\frac{1}{1+\epsilon}} \tag{6.2.46}$$
$$\times\mathbb{E}\Bigg(\int_0^1\Big(1+\Big|\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}+\lambda\Big(\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big)\Big|\Big)^{p\left(1+\frac{1}{\epsilon}\right)}\,\mathrm{d}\lambda\Bigg)^{\frac{\epsilon}{1+\epsilon}}$$

for a given value of ϵ and p, then using Lemma 5.2.1 the latter expectation is bounded by a constant which depends only on p and ϵ, and can be absorbed into C_p, leaving

$$\leq C_pK\delta_{\mathrm{f}}^{\frac{3p}{2}}\mathbb{E}\Bigg(\Big|\hat{X}^{\mathrm{f}}_{m+\frac{1}{2}}-\tilde{X}^{\mathrm{f}}_{m+\frac{1}{2}}\Big|^{p(1+\epsilon)}\Bigg)^{\frac{1}{1+\epsilon}} \tag{6.2.47}$$

using Theorem 5.2.4

$$\leq C_pK\delta_{\mathrm{f}}^{\frac{3p}{2}}\mathbb{E}\Bigg(\Big|\tilde{Z}-Z\Big|^{p(1+\epsilon)}\Bigg)^{\frac{1}{1+\epsilon}}. \tag{6.2.48}$$

Combining all of the results obtained thus far for the constituents of Θ_m, we can bound Θ_m by

$$\mathbb{E}(|\Theta_m|^p) \leq C_p \delta_{\mathrm{f}}^{\frac{3p}{2}} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}}, \tag{6.2.49}$$

which corresponds to $s = \frac{1}{2}$ in Lemma 5.2.3.

All but two of the terms which constitute Ξ_m are bound in identical manners to those in Θ_m, where the exceptions are the second and third terms in Ξ_m. Inspecting the third term we have

$$\begin{aligned}\mathbb{E}\left(\left|\sqrt{\delta_{\mathrm{f}}}\left(\tilde{Z}_{m+\frac{1}{2}} - Z_{m+\frac{1}{2}}\right)\left(b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right)\right|^p\right) \\ = \delta_{\mathrm{f}}^{\frac{p}{2}} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^p\right) \mathbb{E}\left(\left|b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right|^p\right)\end{aligned} \tag{6.2.50}$$

using Jensen's inequality for summations (Lemma A.1.7)

$$\begin{aligned}\leq C_p \delta_{\mathrm{f}}^{\frac{p}{2}} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^p\right) \Bigg(& \mathbb{E}\left(\left|b\left(\tau_m, \tilde{X}_m^{\mathrm{c}}\right) - b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right)\right|^p\right) \\ & + \mathbb{E}\left(\left|b\left(\tau_m, \tilde{X}_m^{\mathrm{f}}\right) - b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right|^p\right) \\ & + \mathbb{E}\left(\left|b\left(\tau_m, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right) - b\left(\tau_{m+\frac{1}{2}}, \tilde{X}_{m+\frac{1}{2}}^{\mathrm{f}}\right)\right|^p\right)\Bigg).\end{aligned} \tag{6.2.51}$$

For the first of these parenthesised expectations we use Lemma 6.2.5 to bound this by $\mathcal{O}(\delta_{\mathrm{f}}^{p/2})$, and similarly for the second term using Lemma 5.2.2. For the third expectation we obtain the same bound by utilising Assumption 5.1.6 and Lemma 5.2.1. These three bounds combine to give

$$\leq C_p \delta_{\mathrm{f}}^{p} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^p\right) \tag{6.2.52}$$

using Lyapunov's inequality (Lemma A.2.4)

$$\leq C_p \delta_{\mathrm{f}}^{p} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}}. \tag{6.2.53}$$

The remaining second term in Ξ_m is bound identically. Thus, we find for Ξ_m a very similar bound to Θ_m, namely

$$\mathbb{E}(|\Xi_m|^p) \leq C_p \delta_{\mathrm{f}}^{p} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}}, \tag{6.2.54}$$

which again has the same dependence on the quantisation error and has $s = \frac{1}{2}$ for Lemma 5.2.3. Combining our bounds for Θ_m and Ξ_m and using $s = \frac{1}{2}$ in Lemma 5.2.3, the desired result immediately follows. **QED**

Remark 6.2.6.1. In the proof of Theorem 6.2.6 we used the fundamental theorem of calculus for Lebesgue integrals (Theorem A.1.3) for $a(\tau_m, \cdot)$ when arriving at (6.2.44), (and similarly for b). However, at a first glance, given the second argument to a is stochastic, we might ask why we haven't had to use Itô's lemma, which would involve a second partial derivative. Comparing the fundamental theorem of calculus for Lebesgue integrals (Theorem A.1.3) $f(t) - f(s) = \int_s^t f'(u)\,\mathrm{d}u$ to Itô's lemma $f(X_t) - f(X_s) = \int_{X_s}^{X_t} f'(X_u)\,\mathrm{d}X_u + \int_s^t \frac{1}{2} f''(X_u)\,\mathrm{d}\langle X\rangle_u$, and it might seem we have missed off the Itô correction from the quadratic variation when we wrote the difference $a(\tau_m, \tilde{X}_m^{\mathrm{f}}) - a(\tau_m, \hat{X}_m^{\mathrm{f}}) = \int_0^1 (\hat{X}_m^{\mathrm{f}} - \tilde{X}_m^{\mathrm{f}}) \frac{\partial a}{\partial X}(\tau_m, \tilde{X}_m^{\mathrm{f}} + \lambda(\hat{X}_m^{\mathrm{f}} - \tilde{X}_m^{\mathrm{f}}))\,\mathrm{d}\lambda$. However, the resolution in this is realising that although the path along X_t is rough and would require Itô's lemma to evolve the value temporally, the integration path between $\hat{X}_m^{\mathrm{f}}$ and $\tilde{X}_m^{\mathrm{f}}$ is purely in the spatial direction, and is not being evolved at all in the temporal direction. Instead the integration path is smooth, and under Assumption 6.2.1 the derivative along this path exists, and hence we only require the fundamental theorem of calculus for Lebesgue integrals (Theorem A.1.3) and not Itô's lemma.

Corollary 6.2.6.1. *Under the same assumptions as Theorem 6.2.6, then for* q-bit *approximate Gaussian random variables constructed as in Lemma 3.2.2 or Corollary 3.2.2.1, then the bound from Theorem 6.2.6 becomes* $\mathbb{E}(|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}|^p) \lessapprox C\delta_{\mathrm{c}}^{p/2} q^{-p/2} 2^{-q/(1+\epsilon)}$.

Proof. The proof follows immediately from Lemma 3.2.4. **QED**

Corollary 6.2.6.2. *For a payoff function* $P\colon \mathbb{R} \to \mathbb{R}$ *which is* $C^1(\mathbb{R})$ *where* P' *is Lipschitz continuous with polynomial growth of order* r *and Lipschitz coefficient* L'*, then there exists a* $p \geq 2$*,* $\epsilon \in (0, \infty)$*, and a constant* $C > 0$ *depending on* p *and* ϵ *and neither on* δ_{c} *nor* $\mathbb{E}(|\tilde{Z} - Z|^{p(1+\epsilon)})$ *such that* $\mathbb{E}(|P(\hat{X}_N^{\mathrm{f}}) - P(\hat{X}_N^{\mathrm{c}}) - P(\tilde{X}_N^{\mathrm{f}}) + P(\tilde{X}_N^{\mathrm{c}})|^p) \leq C\delta_{\mathrm{c}}^{p/2}\mathbb{E}(|\tilde{Z} - Z|^{p(1+\epsilon)})^{1/(1+\epsilon)}$.

Remark 6.2.6.2. We have neglected to include the usual supremum term over the time steps. The reason for this is that in the vast majority of applications the payoff function only makes physical sense at the terminal time, or may only be well defined at termination. This holds true also when considering applications outside of finance.

Proof. For brevity we introduce the abbreviation $\hat{P}^{\mathrm{f}} \equiv P(\hat{X}_N^{\mathrm{f}})$, and introduce $\hat{P}^{\mathrm{c}}$, $\tilde{P}^{\mathrm{f}}$, and $\tilde{P}^{\mathrm{c}}$, each similarly defined. In which case we can use the four-point mean value theorem with polynomial growth (Corollary 6.2.1.1) and Jensen's inequality for summations (Lemma A.1.7) to give

$$\mathbb{E}\Big(\Big|\hat{P}^{\mathrm{f}} - \hat{P}^{\mathrm{c}} - \tilde{P}^{\mathrm{f}} + \tilde{P}^{\mathrm{c}}\Big|^p\Big) \leq C\mathbb{E}\Big(|P'(\xi)|^p\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}\Big|^p\Big) + C\mathbb{E}(|R|^p) \tag{6.2.55}$$

for a ξ and an R constructed in accordance with the four-point mean value theorem with polynomial growth (Corollary 6.2.1.1).

If we consider the first expectation, then we can bound this using Definition 6.2.1 (absorbing $(L')^p$ into a constant C) to give

$$\begin{aligned}
&\mathbb{E}\Big(|P'(\xi)|^p\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}\Big|^p\Big) \\
&\qquad \leq C\mathbb{E}\Big(\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}\Big|^p\Big(1 + \Big|\hat{X}_N^{\mathrm{f}}\Big|^r + \Big|\hat{X}_N^{\mathrm{c}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{f}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{c}}\Big|^r\Big)^p\Big) \qquad (6.2.56)
\end{aligned}$$

using Hölder's inequality for expectations (Lemma A.2.3) for some $0 < \epsilon_1 \ll 1$

$$\begin{aligned}
&\leq C\mathbb{E}\Big(\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}\Big|^{p(1+\epsilon_1)}\Big)^{\frac{1}{1+\epsilon_1}} \qquad (6.2.57) \\
&\qquad \times \mathbb{E}\left(\left(1 + \Big|\hat{X}_N^{\mathrm{f}}\Big|^r + \Big|\hat{X}_N^{\mathrm{c}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{f}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{c}}\Big|^r\right)^{\frac{p(1+\epsilon_1)}{\epsilon_1}}\right)^{\frac{\epsilon_1}{1+\epsilon_1}}
\end{aligned}$$

by Lemma 5.2.1 the second expectation is a finite constant independent of both δ_{c} and all moments of $\tilde{Z} - Z$ and hence can be absorbed into C leaving

$$\leq C\mathbb{E}\Big(\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{f}} + \tilde{X}_N^{\mathrm{c}}\Big|^{p(1+\epsilon_1)}\Big)^{\frac{1}{1+\epsilon_1}} \tag{6.2.58}$$

using Theorem 6.2.6

$$\leq C\delta_{\mathrm{f}}^{\frac{p(1+\epsilon_1)}{2}}\mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^{p(1+\epsilon_1)(1+\epsilon_2)}\Big)^{\frac{1}{(1+\epsilon_1)(1+\epsilon_2)}} \tag{6.2.59}$$

letting $1 + \epsilon \equiv (1+\epsilon_1)(1+\epsilon_2)$ and noting that for a sufficiently small δ_{f} that $\delta_{\mathrm{f}}^a < \delta_{\mathrm{f}}^b$ for any $0 < b < a$ then

$$\leq C\delta_{\mathrm{f}}^{\frac{p}{2}}\mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}. \tag{6.2.60}$$

It now remains to bound the contribution from the $\mathbb{E}(|R|^p)$ term. We know from the four-point mean value theorem with polynomial growth (Corollary 6.2.1.1) that

$$\mathbb{E}(|R|^p) \leq C\mathbb{E}\Big(\Big(1 + \Big|\hat{X}_N^{\mathrm{f}}\Big|^r + \Big|\hat{X}_N^{\mathrm{c}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{f}}\Big|^r + \Big|\tilde{X}_N^{\mathrm{c}}\Big|^r\Big)^p\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}}\Big|^p\Big|\hat{X}_N^{\mathrm{f}} - \tilde{X}_N^{\mathrm{f}}\Big|^p\Big) + \cdots \tag{6.2.61}$$

where for brevity we have omitted the three other cross terms in our expression for $|R|$ which are bound in an identical manner to the term shown. Applying Hölder's inequality for expectations (Lemma A.2.3) twice gives

$$\leq C\mathbb{E}\Big(\Big|\hat{X}_N^{\mathrm{f}} - \tilde{X}_N^{\mathrm{f}}\Big|^{p(1+\epsilon)}\Big)\mathbb{E}\Big(\Big|\hat{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{c}}\Big|^{2p\left(\frac{1+\epsilon}{\epsilon}\right)}\Big)^{\frac{\epsilon}{2(1+\epsilon)}} \tag{6.2.62}$$
$$\times\,\mathbb{E}\left(\left(1+\Big|\hat{X}_N^{\mathrm{f}}\Big|^r+\Big|\hat{X}_N^{\mathrm{c}}\Big|^r+\Big|\tilde{X}_N^{\mathrm{f}}\Big|^r+\Big|\tilde{X}_N^{\mathrm{c}}\Big|^r\right)^{2p\left(\frac{1+\epsilon}{\epsilon}\right)}\right)^{\frac{\epsilon}{2(1+\epsilon)}}+\cdots$$

using Theorem 5.2.4 and Lemma 6.2.3 to bound the first two expectations, and lastly by Lemma 5.2.1 the final expectation is a finite constant independent of both δ_{c} and all moments of $\tilde{Z}-Z$, these give

$$\leq C\delta_{\mathrm{f}}^{\frac{p}{2}}\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}+\cdots \tag{6.2.63}$$

the other terms which have been omitted are identically bound by substituting Theorem 5.2.4 or Lemma 6.2.3 with Lemma 6.2.4 or Lemma 6.2.5 where required respectively, and doing so gives

$$\leq C\delta_{\mathrm{f}}^{\frac{p}{2}}\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{p(1+\epsilon)}\Big)^{\frac{1}{1+\epsilon}}. \tag{6.2.64}$$

If we now combine this with the previous identical bounds into our expression for $\mathbb{E}(|\hat{P}^{\mathrm{f}}-\hat{P}^{\mathrm{c}}-\tilde{P}^{\mathrm{f}}+\tilde{P}^{\mathrm{c}}|^{\,p})$ the desired result follows, completing the proof. **QED**

Corollary 6.2.6.3. *Let a continuous payoff function $P\colon\mathbb{R}\to\mathbb{R}$ have a single position K where P is not differentiable, such that P is $C^1(\mathbb{R}\setminus\{K\})$ where P and P' are Lipschitz continuous, both with polynomial growth of order r and Lipschitz coefficients L and L' respectively except at K, such that*

$$|P(x)-P(y)|\leq L(1+|x|^r+|y|^r)|x-y| \qquad \text{for all } x,y\in\mathbb{R} \tag{6.2.65}$$
$$\big|P'(x)-P'(y)\big|\leq L'(1+|x|^r+|y|^r)|x-y| \qquad \text{if either } x,y>K \text{ or } x,y<K. \tag{6.2.66}$$

Furthermore, let X_T have a bounded probability density in the neighbourhood of K such that there exists a constant $c>0$ such that for any $D>0$ we have the bound $\mathbb{P}(|X_T-K|<D)\leq cD$. Then there exists an $m\geq 2$, $q>m$, $0<\epsilon<1-\frac{m}{q}$, and a constant $C>0$ which depends only on q and ϵ and neither on δ_{f} nor $\mathbb{E}(|\tilde{Z}-Z|^{\,q})$ such that

$$\mathbb{E}\Big(\Big|P(\hat{X}_N^{\mathrm{f}})-P(\hat{X}_N^{\mathrm{c}})-P(\tilde{X}_N^{\mathrm{f}})+P(\tilde{X}_N^{\mathrm{c}})\Big|^m\Big)\leq C\min\left\{\delta_{\mathrm{f}}^{\frac{m}{2}}\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}},\right. \tag{6.2.67}$$
$$\left.\delta_{\mathrm{f}}^{\frac{1-\epsilon}{2}-\frac{m}{2q}}\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^q\Big)^{\frac{m}{q}}\right\}.$$

Remark 6.2.6.3. The proof of Corollary 6.2.6.3 follows in a similar style to an antithetic multilevel Monte Carlo analysis by Giles and Szpruch [85, Assumption 5.1, pages 18–20].

Remark 6.2.6.4. Within the statement of Corollary 6.2.6.3 we assume the process admits a locally bounded probability density. The earlier assumptions from Chapter 5 are not sufficient on their own to produce such a probability density, and thus we assume it's existence in Corollary 6.2.6.3. Showing the existence and properties of such densities is an active area of research [57, 58, 102, 136, 138, 164, 196] (often utilising Malliavin calculus), outside the remit of this thesis.

Proof. For some neighbourhood within a distance D either side of K we can deconstruct the possible scenarios which contribute to the expectation. The first will be when all four approximate solution paths are on the same side of the K, in which case we will readily be able to apply Corollary 6.2.6.2 as before. The other scenario is when some of the paths

straddle either side of K, in which case we will need a revised bound. To consider these two scenarios we introduce the two event sets A and B where $A \coloneqq \{|X_T - K| \leq D\}$ and $B \coloneqq \{\max\{|\hat{X}_N^{\mathrm{f}} - X_T|, |\hat{X}_N^{\mathrm{c}} - X_T|, |\tilde{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{f}}|, |\tilde{X}_N^{\mathrm{c}} - \hat{X}_N^{\mathrm{c}}|\} \geq \frac{D}{2}\}$.

For future brevity within the proof we will define a four-way differencing operator $\square$ by $\square P \equiv P(\hat{X}_N^{\mathrm{f}}) - P(\hat{X}_N^{\mathrm{c}}) - P(\tilde{X}_N^{\mathrm{f}}) + P(\tilde{X}_N^{\mathrm{c}})$. Furthermore, we will introduce the relational operator $\prec$ such that for any two strictly positive functions g and h that $g(\delta_{\mathrm{f}}, \mathbb{E}(|\tilde{Z} - Z|^q)) \prec h(\delta_{\mathrm{f}}, \mathbb{E}(|\tilde{Z} - Z|^q))$ implies that there exists a constant $c > 0$ which is independent of both δ_{f} and $\mathbb{E}(|\tilde{Z} - Z|^q)$ such that $g(\delta_{\mathrm{f}}, \mathbb{E}(|\tilde{Z} - Z|^q)) \leq c\, h(\delta_{\mathrm{f}}, \mathbb{E}(|\tilde{Z} - Z|^q))$.

Looking to bound $\mathbb{E}(|\square P|^m)$ we begin by splitting the expectation over the mutually exclusive and collectively exhaustive sets $A \cup B$ and $A^c \cap B^c$

$$\mathbb{E}(|\square P|^m) = \mathbb{E}\Big(|\square P|^m \mathbb{1}_{\{A^c \cap B^c\}}\Big) + \mathbb{E}\Big(|\square P|^m \mathbb{1}_{\{A \cup B\}}\Big). \tag{6.2.68}$$

For the first expectation all four paths are on the same side of the K and hence we can use Corollary 6.2.6.2 to give

$$\mathbb{E}\Big(|\square P|^m \mathbb{1}_{\{A^c \cap B^c\}}\Big) \prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^{m(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}} \tag{6.2.69}$$

as $q > m$ then we if we choose our ϵ such that $q > m(1+\epsilon)$ and hence $\epsilon < \frac{q}{m} - 1$ then we can use Lyapunov's inequality (Lemma A.2.4) to bound $\mathbb{E}(|\tilde{Z} - Z|^{m(1+\epsilon)})^{\frac{1}{1+\epsilon}} \leq \mathbb{E}(|\tilde{Z} - Z|^q)^{\frac{m}{q}}$ giving

$$\prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\left(\left|\tilde{Z} - Z\right|^q\right)^{\frac{m}{q}}. \tag{6.2.70}$$

Now we can turn our attention to bounding $\mathbb{E}(|\square P|^m \mathbb{1}_{\{A \cup B\}})$, where we begin by using Hölder's inequality for expectations (Lemma A.2.3)

$$\mathbb{E}\Big(|\square P|^m \mathbb{1}_{\{A \cup B\}}\Big) \leq \mathbb{E}\Big(|\square P|^{\frac{m}{\sigma}}\Big)^{\sigma} (\mathbb{P}(A) + \mathbb{P}(B))^{1-\sigma} \tag{6.2.71}$$

for some $0 < \sigma < 1$. By construction, where X_T has a bounded density in the neighbourhood of K we always have $\mathbb{P}(A) \prec D$. Hence it remains to bound $\mathbb{P}(B)$ where we would like to construct our choice of D and its dependence on δ_{f} and $\mathbb{E}(|\tilde{Z} - Z|^q)$ such that we match the decay rate of $\mathbb{P}(A)$ and hence also achieve $\mathbb{P}(B) \prec D$.

To bound $\mathbb{P}(B)$ we can bound the probabilities of either of the four distances appearing in the definition of B of exceeding $\frac{D}{2}$, and hence

$$\mathbb{P}(B) = \mathbb{P}\left(\max\left\{\left|\hat{X}_N^{\mathrm{f}} - X_T\right|, \left|\hat{X}_N^{\mathrm{c}} - X_T\right|, \left|\tilde{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{f}}\right|, \left|\tilde{X}_N^{\mathrm{c}} - \hat{X}_N^{\mathrm{c}}\right|\right\} \geq \frac{D}{2}\right) \tag{6.2.72}$$

$$\leq \mathbb{P}\left(\left|\hat{X}_N^{\mathrm{f}} - X_T\right| \geq \frac{D}{2}\right) + \mathbb{P}\left(\left|\hat{X}_N^{\mathrm{c}} - X_T\right| \geq \frac{D}{2}\right) \tag{6.2.73}$$

$$+ \mathbb{P}\left(\left|\tilde{X}_N^{\mathrm{f}} - \hat{X}_N^{\mathrm{f}}\right| \geq \frac{D}{2}\right) + \mathbb{P}\left(\left|\tilde{X}_N^{\mathrm{c}} - \hat{X}_N^{\mathrm{c}}\right| \geq \frac{D}{2}\right),$$

where we see we need to bound discretisation and quantisation errors. When performing the subsequent bounds, we will see that there are two scenarios which we will need to treat separately:

Scenario I $\quad \delta_{\mathrm{f}}^{1/2} < \mathbb{E}(|\tilde{Z} - Z|^q)^{\frac{1}{q+1}}$.
Scenario II $\quad \delta_{\mathrm{f}}^{1/2} > \mathbb{E}(|\tilde{Z} - Z|^q)^{\frac{1}{q+1}}$.

We will later show that both scenarios coincide and give the same bound in the event $\delta_{\mathrm{f}}^{1/2} = \mathbb{E}(|\tilde{Z} - Z|^q)^{\frac{1}{q+1}}$. We will begin by considering first scenario I.

Scenario I: We begin by bounding the fine quantisation term, which will determine a dependence of D on $\mathbb{E}(|\tilde{Z}-Z|^q)$. So bounding $\mathbb{P}(|\hat{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{f}}|\geq\frac{D}{2})$ using Markov's inequality (Lemma A.2.5) gives

$$\mathbb{P}\left(\left|\hat{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{f}}\right|\geq\frac{D}{2}\right)\leq\frac{\mathbb{E}\left(\left|\hat{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{f}}\right|^q\right)}{\left(\frac{D}{2}\right)^q} \tag{6.2.74}$$

using Theorem 5.2.4

$$\prec\frac{\mathbb{E}\left(\left|\tilde{Z}-Z\right|^q\right)}{D^q}. \tag{6.2.75}$$

To enforce that this is $\prec D$ we can set D to have some power law dependence on $\mathbb{E}(|\tilde{Z}-Z|^q)$. Hence if we set $D\coloneqq\mathbb{E}(|\tilde{Z}-Z|^q)^\alpha$ for some α then for this to ensure the bound is $\prec D$ we require $\mathbb{E}(|\tilde{Z}-Z|^q)\prec\mathbb{E}(|\tilde{Z}-Z|^q)^{\alpha(1+q)}$. We now note that for a sufficiently small time increment such that $0<\delta_{\mathrm{f}}<\delta_{\mathrm{c}}\ll 1$ we have for any $0<\alpha<\beta<\infty$ that $\delta_{\mathrm{c}}^\beta\prec\delta_{\mathrm{c}}^\alpha$ and similarly for δ_{f}. Identically from $\mathbb{E}(|\tilde{Z}-Z|^q)^\beta\equiv\mathbb{E}(|\tilde{Z}-Z|^q)^\alpha\mathbb{E}(|\tilde{Z}-Z|^q)^{\beta-\alpha}$ if we apply Assumption 5.2.3 to $\mathbb{E}(|\tilde{Z}-Z|^q)^{\beta-\alpha}$ we obtain $\mathbb{E}(|\tilde{Z}-Z|^q)^\beta\prec\mathbb{E}(|\tilde{Z}-Z|^q)^\alpha$. Hence enforcing $\mathbb{E}(|\tilde{Z}-Z|^q)\prec\mathbb{E}(|\tilde{Z}-Z|^q)^{\alpha(1+q)}$ requires $\alpha<\frac{1}{q+1}$ and hence is satisfied by choosing $\alpha\coloneqq\frac{1-\rho}{q+1}$ for some $0<\rho<1$, and hence we obtain $D\coloneqq\mathbb{E}(|\tilde{Z}-Z|^q)^{\frac{1-\rho}{q+1}}$. By use of Lemma 6.2.4 an identical bound for the coarse quantisation term similarly follows.

If we now tackle the discretisation term, again by using Markov's inequality (Lemma A.2.5), we obtain

$$\mathbb{P}\left(\left|\hat{X}_N^{\mathrm{f}}-X_T\right|\geq\frac{D}{2}\right)\leq\frac{\mathbb{E}\left(\left|\hat{X}_N^{\mathrm{f}}-X_T\right|^p\right)}{\left(\frac{D}{2}\right)^p} \tag{6.2.76}$$

for some $p\geq 1$. Using Theorem 5.1.3

$$\prec\frac{\delta_{\mathrm{f}}^{\frac{p}{2}}}{D^p}. \tag{6.2.77}$$

Enforcing that this is $\prec D$ where $D\coloneqq\mathbb{E}(|\tilde{Z}-Z|^q)^{\frac{1-\rho}{q+1}}$ requires $\delta_{\mathrm{f}}^{1/2}\prec\mathbb{E}(|\tilde{Z}-Z|^q)^{\frac{1-\rho}{q+1}\left(1+\frac{1}{p}\right)}$. We can satisfy this even in the limit as p becomes arbitrarily large by requiring $\delta_{\mathrm{f}}^{1/2}\prec\mathbb{E}(|\tilde{Z}-Z|^q)^{\frac{1-\rho}{q+1}}$. We now observe that this requirement is satisfied if $\delta_{\mathrm{f}}^{1/2}<\mathbb{E}(|\tilde{Z}-Z|^q)^{\frac{1}{q+1}}$, and hence is always satisfied in scenario I, hence the requirement to divide our considerations into the two scenarios. An identical bound for the coarse path similarly follows.

Having bounded $\mathbb{P}(A)$ and $\mathbb{P}(B)$ it remains to bound $\mathbb{E}(|\square P|^{m/\sigma})^\sigma$ where we do so by considering the difference of the two quantisation terms (for scenario I these can be considered typically smaller than the discretisation differences), giving

$$\mathbb{E}\left(|\square P|^{\frac{m}{\sigma}}\right)^\sigma=\mathbb{E}\left(\left|P(\hat{X}_N^{\mathrm{f}})-P(\hat{X}_N^{\mathrm{c}})-P(\tilde{X}_N^{\mathrm{f}})+P(\tilde{X}_N^{\mathrm{c}})\right|^{\frac{m}{\sigma}}\right)^\sigma \tag{6.2.78}$$

using Jensen's inequality for summations (Lemma A.1.7)

$$\leq 2^{m-\sigma}\left(\mathbb{E}\left(\left|P(\hat{X}_N^{\mathrm{f}})-P(\hat{X}_N^{\mathrm{c}})\right|^{\frac{m}{\sigma}}\right)+\mathbb{E}\left(\left|P(\tilde{X}_N^{\mathrm{f}})+P(\tilde{X}_N^{\mathrm{c}})\right|^{\frac{m}{\sigma}}\right)\right)^\sigma \tag{6.2.79}$$

using (6.2.65)

$$\begin{aligned}\prec\Bigg(&\mathbb{E}\left(\left(1+\left|\hat{X}_N^{\mathrm{f}}\right|+\left|\hat{X}_N^{\mathrm{c}}\right|\right)^{\frac{mr}{\sigma}}\left|\hat{X}_N^{\mathrm{f}}-\hat{X}_N^{\mathrm{c}}\right|^{\frac{m}{\sigma}}\right)\\&+\mathbb{E}\left(\left(1+\left|\tilde{X}_N^{\mathrm{f}}\right|+\left|\tilde{X}_N^{\mathrm{c}}\right|\right)^{\frac{mr}{\sigma}}\left|\tilde{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{c}}\right|^{\frac{m}{\sigma}}\right)\Bigg)^\sigma\end{aligned} \tag{6.2.80}$$

using Hölder's inequality for expectations (Lemma A.2.3)

$$\prec \left(\mathbb{E}\left(\left(1+\left|\hat{X}_N^{\mathrm{f}}\right|+\left|\hat{X}_N^{\mathrm{c}}\right|\right)^{\frac{2mr}{\sigma}}\right)^{\frac{1}{2}}\mathbb{E}\left(\left|\hat{X}_N^{\mathrm{f}}-\hat{X}_N^{\mathrm{c}}\right|^{\frac{2m}{\sigma}}\right)^{\frac{1}{2}}\right. \tag{6.2.81}$$

$$\left.+\,\mathbb{E}\left(\left(1+\left|\tilde{X}_N^{\mathrm{f}}\right|+\left|\tilde{X}_N^{\mathrm{c}}\right|\right)^{\frac{2mr}{\sigma}}\right)^{\frac{1}{2}}\mathbb{E}\left(\left|\tilde{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{c}}\right|^{\frac{2m}{\sigma}}\right)^{\frac{1}{2}}\right)^{\sigma}$$

the first expectations in each of the two terms are bound by a finite constant by using Lemma 5.2.1, and the second expectations are bound by $\delta_{\mathrm{f}}^{m/2\sigma}$ using Lemmas 6.2.3 and 6.2.5, giving

$$\prec \left(\delta_{\mathrm{f}}^{\frac{m}{2\sigma}}\right)^{\sigma} \tag{6.2.82}$$

$$\prec \delta_{\mathrm{f}}^{\frac{m}{2}}. \tag{6.2.83}$$

Combining both the bounds into our expression for $\mathbb{E}(|\square P|^m \mathbb{1}_{\{A\cup B\}})$ gives

$$\mathbb{E}\left(|\square P|^m \mathbb{1}_{\{A\cup B\}}\right) \prec \delta_{\mathrm{f}}^{\frac{m}{2}}\mathbb{E}\left(\left|\tilde{Z}-Z\right|^q\right)^{\frac{(1-\rho)(1-\sigma)}{q+1}} \tag{6.2.84}$$

letting $(1-\epsilon)\equiv(1-\rho)(1-\sigma)$ we obtain

$$\prec \delta_{\mathrm{f}}^{\frac{m}{2}}\mathbb{E}\left(\left|\tilde{Z}-Z\right|^q\right)^{\frac{1-\epsilon}{q+1}}, \tag{6.2.85}$$

which is in the form as desired in Corollary 6.2.6.3. Hence we now switch our attention to scenario II.

Scenario II: We begin here by first bounding the probability for a large discretisation error and see what consequences this produces when we later come to bound the probability of a large quantisation error. We begin by using Markov's inequality (Lemma A.2.5)

$$\mathbb{P}\left(\left|\hat{X}_N^{\mathrm{f}}-X_T\right|\geq\frac{D}{2}\right)\leq\frac{\mathbb{E}\left(\left|\hat{X}_N^{\mathrm{f}}-X_T\right|^p\right)}{\left(\frac{D}{2}\right)^p} \tag{6.2.86}$$

for some $1\leq p<\infty$. Using Theorem 5.1.3

$$\prec \frac{\delta_{\mathrm{f}}^{\frac{p}{2}}}{D^p}. \tag{6.2.87}$$

If we enforce that this is $\prec D$ this requires $\delta_{\mathrm{f}}\prec D^{\frac{2(1+p)}{p}}$. Again we define a power law relation such that $D\coloneqq\delta_{\mathrm{f}}^{\mu}$ for some $\mu>0$ and the requirement on μ becomes $\mu<\frac{p}{2(1+p)}$. We can see that for the range $p\in(1,\infty)$ that $\mu\in(\frac{1}{4},\frac{1}{2})$. Hence as we go to smaller moments it makes sense to introduce some $0<\eta<\frac{1}{2}$ where $\frac{1-\eta}{2}=\mu$, and hence our requirement becomes $\delta_{\mathrm{f}}\prec\delta_{\mathrm{f}}^{(1-\eta)(1+p)/p}$. This is satisfied provided we consider the p-th moment such that $\infty>p>\frac{1}{\eta}-1>1$, giving rise to $D\coloneqq\delta_{\mathrm{f}}^{(1-\eta)/2}$. An identical bound follows for the coarse path.

The bound for the quantisation term again starts by using Markov's inequality (Lemma A.2.5) and Theorem 5.2.4 to give

$$\mathbb{P}\left(\left|\hat{X}_N^{\mathrm{f}}-\tilde{X}_N^{\mathrm{f}}\right|\geq\frac{D}{2}\right)\prec\frac{\mathbb{E}\left(\left|\tilde{Z}-Z\right|^q\right)}{D^q} \tag{6.2.88}$$

using the definition of scenario II to bound the expectation and recalling $D\coloneqq\delta_{\mathrm{f}}^{(1-\eta)/2}$ gives

$$\prec\frac{\delta_{\mathrm{f}}^{\frac{q+1}{2}}}{\delta_{\mathrm{f}}^{\frac{q(1-\eta)}{2}}} \tag{6.2.89}$$

$$\prec\delta_{\mathrm{f}}^{\frac{1+q\eta}{2}} \tag{6.2.90}$$

as $D \coloneqq \delta_{\mathrm{f}}^{(1-\eta)/2}$

$$\prec D. \tag{6.2.91}$$

As before an identical results follows similarly for the coarse paths by instead employing Lemma 6.2.4.

As before, we next need to bound the $\mathbb{E}(|\square P|^{m/\sigma})^{\sigma}$ term

$$\mathbb{E}\Big(|\square P|^{\frac{m}{\sigma}}\Big)^{\sigma} \equiv \mathbb{E}\Big(|\square P|^{q(1-\rho)}\Big)^{\frac{m}{q(1-\rho)}} \tag{6.2.92}$$

for some choice of σ and q there exists a $0 < \rho < 1$ such that $\rho \coloneqq 1 - \frac{m}{\sigma q}$, and hence $\sigma = \frac{m}{q(1-\rho)}$, which gives

$$\equiv \mathbb{E}\left(\left|P(\hat{X}_N^{\mathrm{f}}) - P(\hat{X}_N^{\mathrm{c}}) - P(\tilde{X}_N^{\mathrm{f}}) + P(\tilde{X}_N^{\mathrm{c}})\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} \tag{6.2.93}$$

expressing this as the difference of two quantisation errors, which in the setting of scenario II we expect to be typically of a smaller size than the discretisation errors, gives

$$\equiv \mathbb{E}\left(\left|\Big(P(\hat{X}_N^{\mathrm{f}}) - P(\tilde{X}_N^{\mathrm{f}})\Big) - \Big(P(\hat{X}_N^{\mathrm{c}}) - P(\tilde{X}_N^{\mathrm{c}})\Big)\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} \tag{6.2.94}$$

using Jensen's inequality for summations (Lemma A.1.7) gives

$$\prec \mathbb{E}\left(\left|P(\hat{X}_N^{\mathrm{f}}) - P(\tilde{X}_N^{\mathrm{f}})\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} + \mathbb{E}\left(\left|P(\hat{X}_N^{\mathrm{c}}) - P(\tilde{X}_N^{\mathrm{c}})\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} \tag{6.2.95}$$

using (6.2.65)

$$\begin{aligned} &\prec \mathbb{E}\left(\left(1 + \left|\hat{X}_N^{\mathrm{f}}\right| + \left|\tilde{X}_N^{\mathrm{f}}\right|\right)^{rq(1-\rho)}\left|\hat{X}_N^{\mathrm{f}} - \tilde{X}_N^{\mathrm{f}}\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} \\ &\quad + \mathbb{E}\left(\left(1 + \left|\hat{X}_N^{\mathrm{c}}\right| + \left|\tilde{X}_N^{\mathrm{c}}\right|\right)^{rq(1-\rho)}\left|\hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{c}}\right|^{q(1-\rho)}\right)^{\frac{m}{q(1-\rho)}} \end{aligned} \tag{6.2.96}$$

using Hölder's inequality for expectations (Lemma A.2.3) where we ensure the exponent of the difference between the exact and approximate paths is increased to the q-th power

$$\begin{aligned} &\prec \mathbb{E}\left(\left(1 + \left|\hat{X}_N^{\mathrm{f}}\right| + \left|\tilde{X}_N^{\mathrm{f}}\right|\right)^{\frac{rq(1-\rho)}{\rho}}\right)^{\frac{m\rho}{q(1-\rho)}} \mathbb{E}\Big(\left|\hat{X}_N^{\mathrm{f}} - \tilde{X}_N^{\mathrm{f}}\right|^{q}\Big)^{\frac{m}{q}} \\ &\quad + \mathbb{E}\left(\left(1 + \left|\hat{X}_N^{\mathrm{c}}\right| + \left|\tilde{X}_N^{\mathrm{c}}\right|\right)^{\frac{rq(1-\rho)}{\rho}}\right)^{\frac{m\rho}{q(1-\rho)}} \mathbb{E}\Big(\left|\hat{X}_N^{\mathrm{c}} - \tilde{X}_N^{\mathrm{c}}\right|^{q}\Big)^{\frac{m}{q}} \end{aligned} \tag{6.2.97}$$

noting that the first expectation in each pair can be bounded by a finite constant using Lemma 5.2.1, and that the second terms are bounded using Theorem 5.2.4 and Lemma 6.2.4, giving

$$\prec \mathbb{E}\Big(\left|\tilde{Z} - Z\right|^{q}\Big)^{\frac{m}{q}}. \tag{6.2.98}$$

Combining our bounds for the two probabilities with the expectation of the four-way difference allows us to now bound $\mathbb{E}(|\square P|^{m}\mathbb{1}_{\{A\cup B\}})$ by

$$\mathbb{E}(|\square P|^{m}\mathbb{1}_{\{A\cup B\}}) \prec \mathbb{E}\Big(\left|\tilde{Z} - Z\right|^{q}\Big)^{\frac{m}{q}}\delta_{\mathrm{f}}^{(\frac{1-\eta}{2})(1-\frac{m}{q(1-\rho)})}. \tag{6.2.99}$$

If we now inspect the exponent $(\frac{1-\eta}{2})(1-\frac{m}{q(1-\rho)})$, then for a sufficiently small ρ we have $(\frac{1-\eta}{2})(1-\frac{m}{q(1-\rho)}) \equiv (1-\eta)(\frac{1}{2} - \frac{m}{2q}(1-\rho'))$ for some $0 < \rho' \ll 1$. Expanding this out and collecting together the perturbation terms, it is possible to choose for any $q > m$ a ρ' sufficiently small such that the first order perturbation in η is greater than the first order ρ' term. This leads to an overall slightly negative perturbation, and hence it is possible to find some $0 < \epsilon'$ such that $(\frac{1-\eta}{2})(1-\frac{m}{q(1-\rho)}) \equiv \frac{1-\epsilon'}{2} - \frac{m}{2q}$, giving

$$\prec \mathbb{E}\Big(\left|\tilde{Z} - Z\right|^{q}\Big)^{\frac{m}{q}}\delta_{\mathrm{f}}^{\frac{1-\epsilon'}{2}-\frac{m}{2q}}. \tag{6.2.100}$$

Furthermore, as we know $0 < \eta < \frac{1}{2}$, and $0 < \sigma < 1$ with $\rho := 1 - \frac{m}{q\sigma}$, we have $0 < \rho < 1 - \frac{m}{q} < 1$. So given $(\frac{1-\eta}{2})(1 - \frac{m}{q(1-\rho)}) \equiv \frac{1-\epsilon'}{2} - \frac{m}{2q}$, it is always possible to satisfy this equivalence and under the domain constraints for ρ and η for any q provided we chose $0 < \epsilon' < 1 - \frac{m}{q}$.

Scenarios I or II: We are now in a position to combine both scenarios, and hence we obtain the result

$$\mathbb{E}(|\square P|^m) \prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} + \begin{cases} \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}} & \text{if } \delta_{\mathrm{f}}^{\frac{1}{2}} < \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q+1}} \\ \delta_{\mathrm{f}}^{\frac{1-\epsilon'}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} & \text{if } \delta_{\mathrm{f}}^{\frac{1}{2}} > \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q+1}}. \end{cases} \tag{6.2.101}$$

However, this does not yet cover the case when $\delta_{\mathrm{f}}^{1/2} = \mathbb{E}(|\tilde{Z} - Z|^{\,q})^{1/(q+1)}$. To address this we consider the scenario I bound

$$\delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}} \equiv \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{-\frac{m}{q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \tag{6.2.102}$$

substituting in $\delta_{\mathrm{f}}^{1/2} = \mathbb{E}(|\tilde{Z} - Z|^{\,q})^{1/q+1}$ for the first two expectations

$$\equiv \delta_{\mathrm{f}}^{\frac{m}{2}} \left(\delta_{\mathrm{f}}^{\frac{q+1}{2}}\right)^{\frac{1-\epsilon}{q+1}} \left(\delta_{\mathrm{f}}^{\frac{q+1}{2}}\right)^{-\frac{m}{q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \tag{6.2.103}$$

which simplifies to

$$\equiv \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}}. \tag{6.2.104}$$

Hence if we set $\epsilon = \epsilon'$ we see that the two bounds for the two different scenarios coincide.

Furthermore, we can show that in the limit of decreasing δ_{f} and $\mathbb{E}(|\tilde{Z} - Z|^{\,q})$ that the $\delta_{\mathrm{f}}^{m/2}\mathbb{E}(|\tilde{Z} - Z|^{\,q})^{m/q}$ term is negligible compared to the terms coming from the two scenarios. Comparing first to the scenario I term we have

$$\delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q}} \tag{6.2.105}$$

$$\prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q+1}} \tag{6.2.106}$$

$$\prec \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}}. \tag{6.2.107}$$

Similarly for the scenario II term we have

$$\delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \prec \delta_{\mathrm{f}}^{\frac{1}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \tag{6.2.108}$$

$$\prec \delta_{\mathrm{f}}^{\frac{1}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \tag{6.2.109}$$

$$\prec \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} \tag{6.2.110}$$

where we must enforce $\frac{1-\epsilon}{2} - \frac{m}{2q} > 0$ which requires $\epsilon < 1 - \frac{m}{q}$. (We notice that this bound has already previously been required). With these two results we can write

$$\mathbb{E}(|\square P|^m) \prec \begin{cases} \delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}} & \text{if } \delta_{\mathrm{f}}^{\frac{1}{2}} \leq \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q+1}} \\ \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}} & \text{if } \delta_{\mathrm{f}}^{\frac{1}{2}} > \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1}{q+1}} \end{cases} \tag{6.2.111}$$

which is more conveniently expressed as

$$\prec \min\left\{\delta_{\mathrm{f}}^{\frac{m}{2}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{1-\epsilon}{q+1}}, \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2} - \frac{m}{2q}} \mathbb{E}\Big(\Big|\tilde{Z} - Z\Big|^q\Big)^{\frac{m}{q}}\right\}. \tag{6.2.112}$$

To show the equivalence of the two scenarios to the representation using the minimum we need to show that if the first term is the minimum this requires scenario I to be the case, and vice versa for scenario II. Hence we begin by considering when the first term is the smaller of the two and hence

$$\delta_{\mathrm{f}}^{\frac{m}{2}}\,\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{q}\Big)^{\frac{1-\epsilon}{q+1}} \leq \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2}-\frac{m}{2q}}\,\mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{q}\Big)^{\frac{m}{q}} \tag{6.2.113}$$

$$\iff \qquad \delta_{\mathrm{f}}^{\frac{1}{2}} \leq \mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{q}\Big)^{\left(\frac{m}{q}-\frac{1-\epsilon}{q+1}\right)\left(m-1+\epsilon+\frac{m}{q}\right)^{-1}} \tag{6.2.114}$$

the exponent readily simplifies to $\frac{1}{q+1}$ giving

$$\iff \qquad \delta_{\mathrm{f}}^{\frac{1}{2}} \leq \mathbb{E}\Big(\Big|\tilde{Z}-Z\Big|^{q}\Big)^{\frac{1}{q+1}} \tag{6.2.115}$$

which we recognise as the condition for scenario I. Conversely we see that had the other term been the minimum of the two then this must correspond to scenario II.

The final item remaining is to recall our bounds for how small ϵ must be as a function of q. In the previous analysis we have required $\epsilon < \frac{q}{m} - 1$ and $\epsilon < 1 - \frac{m}{q}$, where we notice that for $q > m$ that the latter inequality is strictly tighter, and hence we only require $\epsilon < 1 - \frac{m}{q}$. This completes the proof. **QED**

Chapter 7

Implementing a nested multilevel Monte Carlo

In this chapter we discuss some of the details involved in successfully implementing the nested multilevel Monte Carlo analysed thus far. We begin by revisiting the multilevel Monte Carlo construction and deriving expressions for the time savings which can be expected from varying discretisations and utilising approximate random variables. We proceed to show that the required tight couplings of the multilevel corrections can be seen empirically. Lastly, given the numeric results seen thus far, we can estimate the computational savings one can expect to achieve from utilising our proposed nested multilevel Monte Carlo framework.

7.1 The multilevel Monte Carlo construction

For regular multilevel Monte Carlo, which uses different temporal discretisations as the basis for differing simulations, it uses the estimator $\hat{\theta} \coloneqq \sum_{l=0}^{L} \frac{1}{m_l} \sum^{m_l} \Delta_l$, where $\Delta_l \coloneqq \hat{P}_l - \hat{P}_{l-1}$ with $\hat{P}_L$ being the approximation of the property P which we wish to estimate, and m_l instances of Δ_l are realised for each level. For simplicity we will restrict our attention to estimators of this form, but there are possible other unbiased estimators, such as e.g. an antithetic construction [85]. We use the definition that $\hat{P}_{-1} \coloneqq 0$ for simplicity of notation.

We are able to work out the total time required for simulation in order to achieve an estimate with a prescribed error. The error between an estimator and the exact quantity incurs a trade-off between the bias of an estimator and the estimator's variance. In our setting this is usually the bias introduced by having to use a given approximation scheme to approximate the SDE, which in our case we will take to be the Euler-Maruyama scheme unless otherwise stated (or a variant thereof). We can reduce this bias by taking paths with a greater fidelity and smaller time increments, but this makes each path simulation more costly. Additionally, the path simulations have a variance, and this is reduced by simulating more paths. The relation between the *mean squared error* (MSE) of an estimator and the variance and bias is given by the well known formula $\mathrm{MSE}\left(\hat{\theta}\right) = \mathbb{V}(\hat{\theta}) + \mathrm{Bias}^2(\hat{\theta})$ [92, page 16].

In our setting, for the random variable Δ_l with a computational cost c_l and variance v_l, we can compute the total time required to achieve a given error [76]. If we set the MSE to be ε^2, then given we choose a maximum simulation accuracy such that the bias satisfies $\mathrm{Bias}^2(\hat{\theta}) \lessapprox \frac{\varepsilon^2}{2}$, then we can write down a total computational time cost $T = \sum_{l=0}^{L} m_l c_l$ and minimise this subject to the constraint on the MSE. Performing this constrained minimisation over an objective function $\mathscr{F}$, using a Lagrange multiplier μ to enforce the constraint $\mathbb{V}(\hat{\theta}) = \frac{\varepsilon^2}{2}$, then we have the objective function $\mathscr{F} \coloneqq \sum_{l=0}^{L} m_l c_l + \mu(\sum_{l=0}^{L} \frac{v_l}{m_l} - \frac{\varepsilon^2}{2})$ where $\partial_\mu \mathscr{F} = \partial_{m_l} \mathscr{F} = 0$.

The resulting equations can be solved to give $m_l = \sqrt{\mu \frac{v_l}{c_l}}$ and $\mu = \frac{2}{\varepsilon^2}\sum_{l=0}^{L} m_l c_l$. Re-arranging and eliminating μ gives $m_l = \varepsilon^{-1}\sqrt{\frac{2Tv_l}{c_l}}$ and $T = 2\varepsilon^{-2}(\sum_{l=0}^{L}\sqrt{v_l c_l})^2$.

It is worth commenting on the case when we only have one level and $L = 0$, in which case this collapses to regular Monte Carlo with a total time $T = 2\varepsilon^{-2}v_0 c_0$. If using the Euler-Maruyama scheme, this has a weak error, (and thus bias), of order 1 [133, 14.1, page 457], and hence Bias $= \mathcal{O}(\delta)$. Combining this with $|\text{Bias}(\hat{\theta})| \approx \frac{\varepsilon}{\sqrt{2}}$ we see that $\delta = \mathcal{O}(\varepsilon)$, and thus a simulation with N increments has a cost $c_0 = \mathcal{O}(N) = \mathcal{O}(\delta^{-1}) = \mathcal{O}(\varepsilon^{-1})$. This means the total computational time is $\mathcal{O}(\varepsilon^{-3})$. Notably, it was Giles [76] who showed that the expression for the total time of a non-trivial multilevel, under certain assumptions, can be brought down to $\mathcal{O}(\varepsilon^{-2}\log^2(\varepsilon))$ or even $\mathcal{O}(\varepsilon^{-2})$.

7.1.1 Using approximate random variables

With the introduction of approximate random variables we have the means to cheaply produce approximate path simulations. When using the approximate random variables, we consider utilising a single approximation level. With the introduction of the approximate random variables[1] we have the new estimator $\tilde{\theta} \coloneqq \sum_{l=0}^{L}(\frac{1}{\tilde{m}_l}\sum^{\tilde{m}_l}\tilde{\Delta}_l + \frac{1}{\tilde{M}_l}\sum^{\tilde{M}_l}\square_l)$ where $\tilde{\Delta}_l \coloneqq \tilde{P}_l - \tilde{P}_{l-1}$ and $\square_l \coloneqq \hat{P}_l - \hat{P}_{l-1} - \tilde{P}_l + \tilde{P}_{l-1}$, where as before we use the convention $\tilde{P}_{-1} \coloneqq 0$. As before we define $\tilde{\Delta}_l$ as having a computation cost $\tilde{c}_l$ and variance $\tilde{v}_l$ and similarly $\tilde{C}_l$ and $\tilde{V}_l$ for $\square_l$.

Again we can write the expression for the new total cost $\tilde{T}$ and minimise this under a constraint on the MSE. Performing this optimisation gives a total computational cost of

$$\tilde{T} = 2\varepsilon^{-2}\left(\sum_{l=0}^{L}\sqrt{\tilde{v}_l\tilde{c}_l} + \sqrt{\tilde{V}_l\tilde{C}_l}\right)^2 \tag{7.1.1}$$

with $\tilde{m}_l = \varepsilon^{-1}\sqrt{\frac{2\tilde{T}\tilde{v}_l}{\tilde{c}_l}}$ and $\tilde{M}_l = \varepsilon^{-1}\sqrt{\frac{2\tilde{T}\tilde{V}_l}{\tilde{C}_l}}$.

Using these values we can compute the additional savings achieved by using the approximate random variables (without any loss in accuracy). We begin by considering our expression for the required total time in (7.1.1), where we use the assumption that our approximation is of a sufficiently high fidelity such that $\tilde{v}_l \approx v_l$ to give

$$\tilde{T} \approx 2\varepsilon^{-2}\left(\sum_{l=0}^{L}\sqrt{v_l c_l}\sqrt{\frac{\tilde{c}_l}{c_l}}\left(1 + \sqrt{\frac{\tilde{V}_l\tilde{C}_l}{v_l\tilde{c}_l}}\right)\right)^2 \tag{7.1.2}$$

using a conservative bound valid over all levels

$$\lessapprox T\max_{l\leq L}\left\{\frac{\tilde{c}_l}{c_l}\left(1 + \sqrt{\frac{\tilde{V}_l\tilde{C}_l}{v_l\tilde{c}_l}}\right)^2\right\}. \tag{7.1.3}$$

The key ingredients to realising a strong computational saving are twofold. The first requirement should be a strong savings in the cost of path simulations such that $\frac{\tilde{c}_l}{c_l} \ll 1$. The second requirement will be that the reduction in the variance of the correction, as compared to the original, outweighs the increased cost such that $\frac{\tilde{V}_l\tilde{C}_l}{v_l\tilde{c}_l} \ll 1$. We will shortly show that these two requirements are satisfied in our framework, and quantify the savings which can be expected.

[1] We assume here that we introduce approximate random variables at all of the available levels, and that we force $\Delta_l \to \tilde{\Delta}_l + \square_l$ for all $l \in \{0, 1, 2, \ldots L\}$.

7.2 Multilevel couplings

From our discussion of the multilevel Monte Carlo framework we have seen that we need to couple our levels such that the multilevel correction has a much reduced variance compared to the original quantity's. In order to see this we will need to simulate an SDE, and for this we use the same geometric Brownian motion from (5.3.1) that we used earlier in Section 5.3. Furthermore, our default payoff function P we will assume to be a call option unless otherwise stated, where a call option is defined as $P_{\text{call}}(X) \coloneqq (X - K)^+$, for K being some positive constant referred to as the strike.

7.2.1 Discretisation levels

To make clear the coupling mechanism at play in multilevel Monte Carlo, we begin by demonstrating the coupling between levels where only the degree of discretisation between the two changes. So for an Euler-Maruyama estimate with N increments we can see the quality of the path estimates for increasingly fine and coarse paths, as shown in Figure 7.2.1.

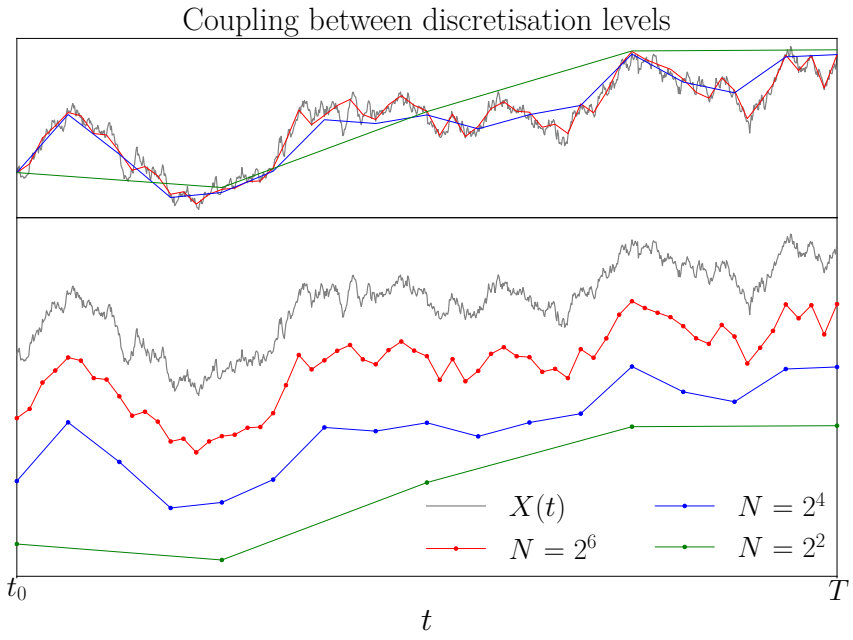


Figure 7.2.1 – The coupling between different discretisation levels. **(Above)** Coarse and fine paths in relation to an exact path realisation. **(Below)** The same paths with a vertical offset and markers denoting the sampled time points.

Given these coupled paths, we can assess the error between the two, either looking at the weak error, or the strong error, both defined in Definitions 5.1.2 and 5.1.3. For multilevel Monte Carlo analysis it is the strong error which we need to concern ourselves with. For the usual Euler-Maruyama scheme, it has strong convergence order of $\frac{1}{2}$, which is well known, and is discussed by Glasserman [92, 6.1.2, page 345] and Kloeden and Platen [133, Theorem 14.5.2]. While the total mean squared error of an estimator can be expressed as the sum of the variance and squared bias [92, 1.1.3, page 16], Glasserman [92, 6.3.3] highlights the role the weak error has in the trade off between variance and bias. However, the core concern for multilevel Monte Carlo frameworks is the strong error, which we will be focusing on.

7.2.2 Quantisation levels

We can also couple the levels between two simulations by varying the quality of the inverse transformation mapping. We can use the exact evaluation of $\Phi^{-1}(\cdot)$ to obtain random variables whose distribution perfectly matches the Gaussian distribution, or we can use a cruder approximation, such as the quantised piecewise constant approximation, producing a *quantised-multilevel Monte Carlo* (QMLMC). We give examples of the paths using quantised Gaussian increments in Figure 7.2.2, which demonstrates the coupling between the different quantisation levels.

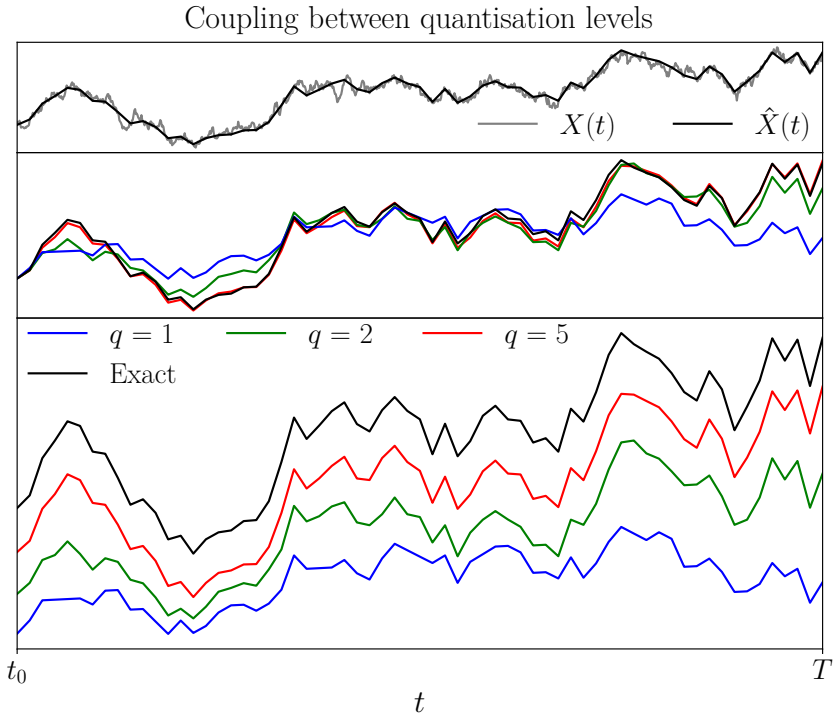


Figure 7.2.2 – The coupling between different quantisation levels. **(Top)** The exact path realisation and the Euler-Maruyama estimate using the exact evaluation of $\Phi^{-1}(\cdot)$. **(Centre)** Crude and accurate paths using the approximate Euler-Maruyama scheme in relation to the exact Euler-Maruyama approximation. **(Bottom)** The same paths with a vertical offset, for an equipartitioned approximation of $\Phi^{-1}(\cdot)$ using 2^q bins, as in Section 4.2 and Figure 4.2.1.

One of the observations we can make from Figure 7.2.2 is that the quantisation with only 2^1 bins does not fluctuate to the same extent as the higher fidelity quantisations. This is because the very crude quantisation does not facilitate the large values at the asymptotes of $\Phi^{-1}(\cdot)$.

Having qualitatively demonstrated the coupling between the different quantisation levels, we can quantitatively show how this ties in with the nested quantised multilevel Monte Carlo framework in (6.1.2). By considering the multilevel discretisation correction between a fine and coarse path $\hat{P}_1 - \hat{P}_0$, we can see how the variance of the quantisation correction $\hat{P}_1 - \hat{P}_0 - (\tilde{P}_1 - \tilde{P}_0)$ changes between using $\Phi^{-1}(\cdot)$ and an equipartitioned lookup table with 2^q bins. Simulating 10^5 paths for each quantisation, and letting the coarse path use a single increment ($\delta = 1$) and the fine path use four increments ($\delta = \frac{1}{4}$) [36, 76], then taking $K = X_0$, the results are shown in Figure 7.2.3.

We can discuss several of the features of Figure 7.2.3. The discretisation correction $\hat{P}_1 - \hat{P}_0$ does not vary with the quantisation, as expected. As the quantisation increases, the quantised discretisation correction $\tilde{P}_1 - \tilde{P}_0$ converges to the un-quantised correction. This convergence is from below, which is a manifestation of the reduced path variance that was observed in Figure 7.2.2. Lastly, the quantisation correction's variance is decaying steadily

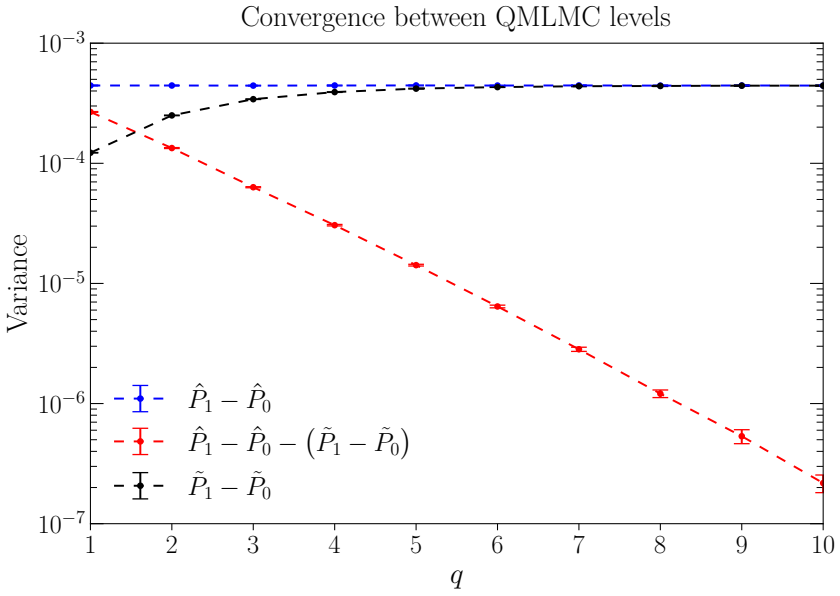


Figure 7.2.3 – The convergence between the discretised MLMC estimator and the nested QMLMC estimator.

with the degree of quantisation, and for a lookup table with 2^{10} bins the variance of the correction is reduced by approximately 10^3. Hence the cheaper quantised term will dominate the number of samples in the nested multilevel Monte Carlo simulation, and the correction terms will be few, which is a key and encouraging result for the quantised-multilevel Monte Carlo construction going forward.

7.2.3 Convergence for different payoff functions

Based on our previous results, namely Corollaries 6.2.6.2 and 6.2.6.3, we expect differing types of convergence behaviour for the multilevel Monte Carlo correction term arising from the introduction of approximate random variables for different types of payoff functions. For these multilevel Monte Carlo simulations we use the fine and coarse coupling suggested by Giles [76] and used by Brugger et al. [36], where the coarsest level $l = 0$ uses a single time step, and each higher level increases this by a power of four, such that $N = 4^l$, and hence $\delta = N^{-1} = 2^{-2l}$. Furthermore, for the approximations, we are using the quantised uniform piecewise constant approximation, implemented using a lookup table, with the constant value being the expected value within each respective interval, where there are 2^q intervals within $[0, 1)$, corresponding to a q-bit uniform. For this case we recall from Lemma 3.2.4 that $\mathbb{V}(Z - \tilde{Z}) \propto 2^{-q}q^{-1}$ which numerically will be dominated by the exponential term and appear largely similar to $\mathcal{O}(2^{-q})$ as q becomes much greater than 1.

The first result we can assess is the convergence without the payoff function, where from Theorem 6.2.6 we expect $\mathbb{V}(\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \tilde{X}_n^{\mathrm{f}} + \tilde{X}_n^{\mathrm{c}}) \approx \mathcal{O}(2^{-2l}2^{-q})$ where we have taken the limit $\epsilon \to 0$ in Theorem 6.2.6 (which is what we expect to observe numerically). The convergence results are shown in Figure 7.2.4.

Inspecting Figure 7.2.4 we can make several remarks. The first is that the decay rate with the discretisation level, as demonstrated by the least squares fit, tightly matches the 2^{-2l} predicted, and we see this holds as q is increased. Secondly, the spacing between the results for an increment in q is nearly a factor of two each time. We see a decay rate of approximately $2^{-1.2q}$, which is not dissimilar to the $\mathcal{O}(2^{-q})$ anticipated. Lastly, the variance for the $q = 10$ data is approximately 10^3–10^4 smaller than the variance which uses the exact Gaussian random variables, showing that the correction is tightly coupled and very few samples of this should be

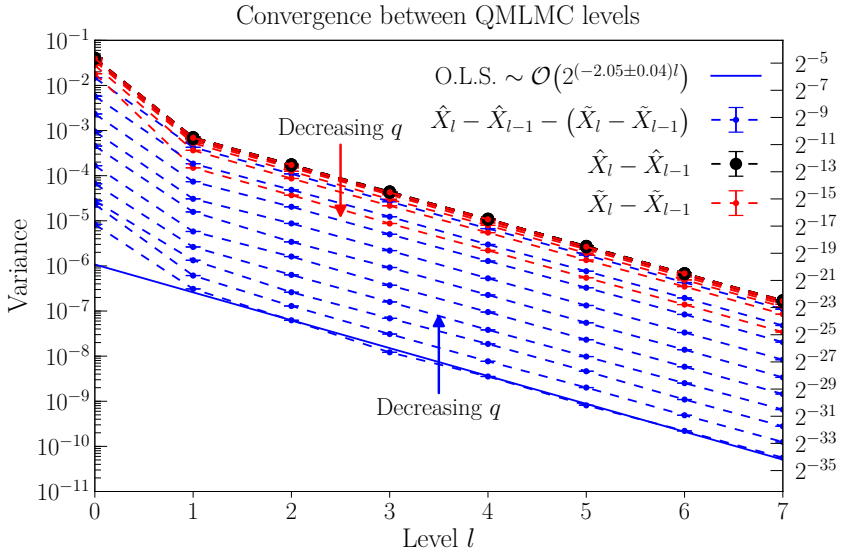


Figure 7.2.4 – The convergence of the quantised-multilevel Monte Carlo correction term for the different discretisation levels and for a varying number of q-bit uniforms transformed using a equipartitioned piecewise constant approximation up to $q = 10$. We show the ordinary least squares (O.L.S.) fit for levels $l \geq 1$ and $q = 10$.

required in a multilevel Monte Carlo implementation.

Encouraged by this result, we can inspect the prediction from Corollary 6.2.6.3 where we include a call option payoff which contains a non-differentiable point at the strike $K = X_0$. Under the same assumptions we expect

$$\mathbb{V}\left(\hat{P}^{\mathrm{f}} - \hat{P}^{\mathrm{c}} - \tilde{P}^{\mathrm{f}} + \tilde{P}^{\mathrm{c}}\right) \approx \mathcal{O}\left(\min\left\{\delta_{\mathrm{f}}\mathbb{E}\left(\left|\tilde{Z} - Z\right|^{q'}\right)^{\frac{1-\epsilon}{q'+1}}, \delta_{\mathrm{f}}^{\frac{1-\epsilon}{2}-\frac{1}{q'}}\mathbb{E}\left(\left|\tilde{Z} - Z\right|^{q'}\right)^{\frac{2}{q'}}\right\}\right) \tag{7.2.1}$$

for some choice of $q' > 2$ and $0 < \epsilon < 1 - \frac{2}{q'}$. Taking q' to be some very large number, then we see that the variance will transition between $\mathcal{O}(2^{-2l})$ for very fine discretisations and coarse quantisations, to $\mathcal{O}(2^{-l})$ for very coarse discretisations and fine quantisations. The convergence results found are shown in Figure 7.2.5.

Inspecting Figure 7.2.5 we can see that for the very crudest quantisations we approach the $\mathcal{O}(2^{-2l})$ behaviour expected. Inspecting the local fits in between the discretisation levels 1–3, we see that as we increase q, the decay rate is transitioning to 2^{-l} which we also anticipate. To showcase the transition between the two rates we again consider the $q = 10$ quantisation. We see that for this quantisation if we were to fit the decay rate over levels $l \geq 1$, we see a decay rate less than the 2^{-2l} expected. Rather we see that for $q = 10$, as we increase the simulations from the coarser to the finer discretisation levels, our decay rate transitions as we had hoped.

Of course, there is an endless degree of niceness, smoothness, continuity, etc. with which we can construct payoff functions. Some of these are purely academic, others trivial, and several in between of interest in both academic and industrial settings. While our analysis covers only continuous payoff functions with or without a point of discontinuity, which are Lipschitz, we can show the convergence behaviour for a wide variety of payoff functions, where some of which are outside the scope of our analysis. A variety of different example payoff functions are listed in Table 7.2.1, and their convergence rates shown in Figure 7.2.6.

Looking at Figure 7.2.6 we can make several remarks about the convergence behaviours exhibited. The first and most striking of these is the behaviour for the discontinuous payoff functions (a)–(d). We can notice from these that as the levels increase and become finer, there is no variance decrease in the multilevel correction term to capitalise on. Furthermore, there is only a variance decrease at the coarsest one or two levels. This

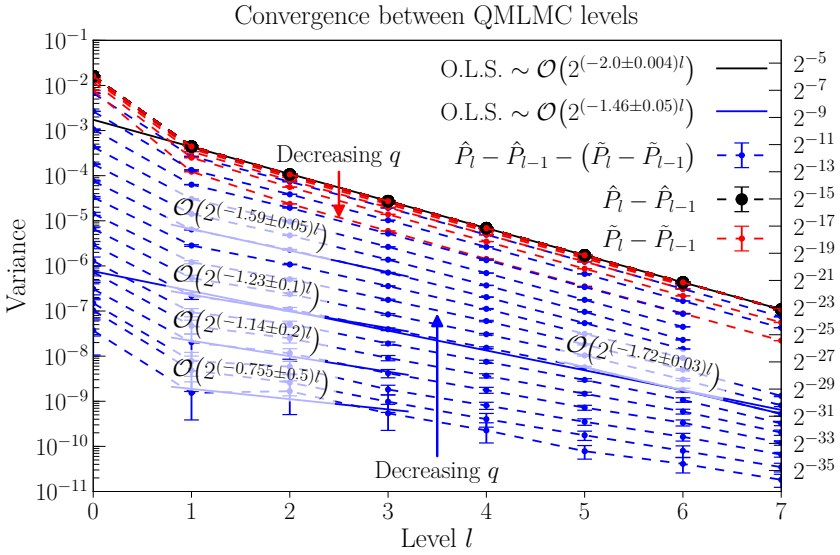


Figure 7.2.5 – The convergence of the quantised-multilevel Monte Carlo correction term for a non-differentiable call option payoff function, considering quantisations up to $q = 16$. We show the ordinary least squares (O.L.S.) fit for levels $l \geq 1$ and $q = 10$, and then several more local fits for smaller clusters of adjacent discretisation levels.

Name	Continuous	Differentiable	Lipschitz	Flat	Example
a	✗	✗	✗	✗	$\left(\sqrt{(X^2-K^2)^+}+K\right)\mathbb{1}_{\{X\geq K\}} + \sqrt{(K^2-X^2)^+}$
b	✗	✗	✗	✓	$\sqrt{(X^2-K^2)^+}+K\mathbb{1}_{\{X\geq K\}}$
c	✗	✗	✗	✗	$\lvert X-K\rvert + K\mathbb{1}_{\{X\geq K\}}$
d	✗	✗	✗	✓	$K\mathbb{1}_{\{X\geq K\}}$
e	✓	✗	✗	✗	$\sqrt{KX}$
f	✓	✗	✗	✓	$\sqrt{K(X-K)^+}$
g	✓	✗	✓	✗	$\lvert X-K\rvert$
h	✓	✗	✓	✓	$(X-K)^+$
i	✓	✓	✓	✗	X^2/K
j	✓	✓	✓	✓	$((X-K)^3/K^2)^+$
k	✓	✓	✓	✗	X

Table 7.2.1 – A selection of example payoff functions with varying degrees of smoothness defined on $X \in (0, \infty)$ parametrised by a strike value K. We indicate whether the function is continuous, differentiable, Lipschitz (locally), and if the function is constant (flat) in a neighbourhood to one or more sides of the strike.

might suggest our proposed framework utilising approximate random variables can be applied for only the coarsest level or two. Nonetheless, it is clear that the behaviour these payoff functions exhibit falls outside of the current scope of our analysis, and remains an avenue for further research. The most tractable and applicable of these would be (d), which is also known as a digital or binary option, which has already been under the investigation of several studies [13, 14, 79, 86, 87].

A similar remark holds true for the payoff function (f). The key difference between (f) and (e), is that while they are both not locally Lipschitz, the kink for (e) is at the origin and for (f) is at the strike K. For a geometric Brownian motion, its solution is a log-normal distribution [133, 4.4, page 119, (4.6)], and hence for (e) the point of non-differentiability and non Lipschitz

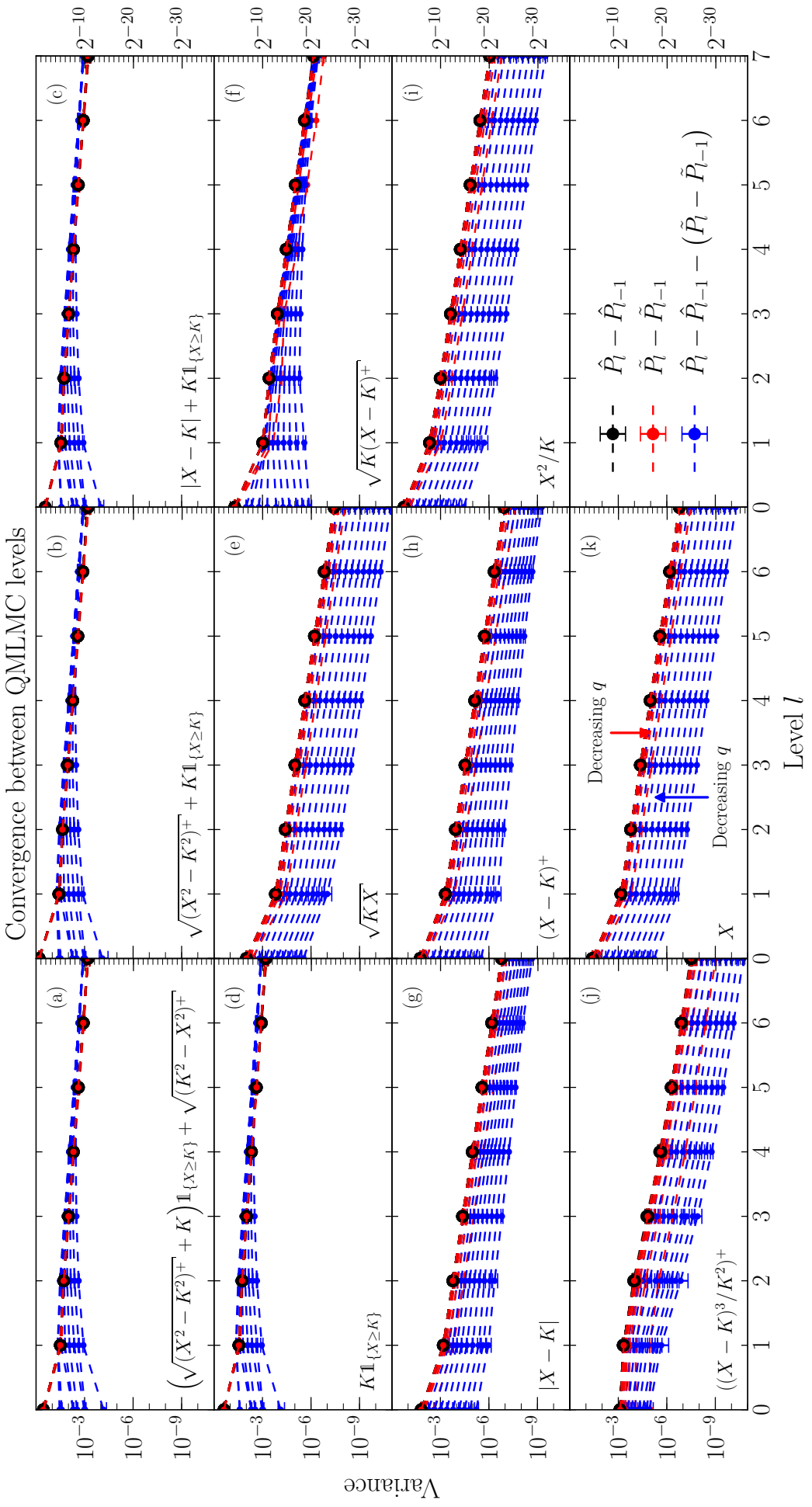


Figure 7.2.6 – The variance structure of a single quantised nested multilevel Monte Carlo scheme, driven by a geometric Brownian Motion. The properties of the payoff functions are listed in Table 7.2.1.

continuity is inaccessible, whereas for (f) it is accessible. This gives (f) a neighbourhood around the strike where the payoff is "near discontinuous". It is for this reason that (f) exhibits similar behaviour to (a)–(d).

For the payoff functions (i)–(k), in the domain $(0, \infty)$, these are all differentiable with polynomial growth in the derivatives and locally Lipschitz. This means they fall under the scope of Theorem 6.2.6, and demonstrate the expected convergence behaviour that we saw earlier in Figure 7.2.4.

The payoff function (e) is very similar to (i)–(k) in so far as that away from the origin (which is inaccessible), it too is differentiable. Furthermore, for any finite distance $\epsilon > 0$ away from the origin, then in the domain $[\epsilon, \infty)$ it too is differentiable with polynomial growth in the derivatives and locally Lipschitz. Thus, although (e) falls outside of the scope of Theorem 6.2.6, for the reasons just mentioned we can anticipate that Theorem 6.2.6 should provide a good description nonetheless, which it seems to.

The last payoff functions to discuss are (g) and (h), where although they are Lipschitz, they both have a point of non-differentiability. As such they fall under the scope of Corollary 6.2.6.3, and exhibit the anticipated behaviour we saw earlier in Figure 7.2.5.

7.3 Expected savings

As an example to showcase some of the computational savings that can be expected using approximate random variables we will consider two example payoff functions, where the first will be the well behaved Lipschitz and differentiable payoff $P(X_T) \coloneqq X_T$, and the second will be the Lipschitz and non-differentiable call option payoff $P(X_T) \coloneqq (X_T - K)^+$ with strike K.

For our simulations we will assume the cost is dominated entirely by the amount of random numbers which are required. If we decide to use our dyadic approximation, then we know that this will achieve a good performance for the random number generation. If we assume our simulations are performed in 32 bit single-precision, then we know the time required to generate Gaussian random numbers on Intel and Arm hardware. Taking Intel AVX-512 hardware as an example then we have the following cost benefits shown in Table 7.3.1. For the Intel hardware we have timing results for the equipartitioned lookup table in double-precision, which coincidentally also appears to demonstrate a time difference of a factor of $\frac{1}{7}$.

	Description	Precision	Time (Clock cycles)
Z	Exact	Single	3.42 ± 0.04
		Double	8.5 ± 0.5
$\tilde{Z}$	Dyadic	Single	0.51 ± 0.02
$\tilde{Z}$	Lookup table	Double	1.2 ± 0.1
$\tilde{c}_l$	Crude		$\frac{1}{7}c_l$
$\tilde{C}_l$	MLMC correction		$\frac{8}{7}c_l$

Table 7.3.1 – Approximate time estimates for single precision simulations using dyadic approximations on Intel AVX-512 hardware.

Given we are using the dyadic approximation, this is quite high fidelity, and we find the drop in variance is typically quite considerable, where the changes in variance are shown in Table 7.3.2. (The values used in Table 7.3.2 have been inferred from Figure 7.3.1). For the call option payoff, as we expect different variance drops as we vary the levels, we use a conservative

and sub-optimal bound obtained at a high level and use this to bound all levels. As an example of this, we notice the for the call option in Figure 7.3.1, that for level $l = 1$ the relative drop in variance of the multilevel correction is much more than at level $l = 7$, and hence we use the drop's value at $l = 7$ as a conservative bound for use in (7.1.3).

	Description	Variance	
		X_T	$(X_T - K)^+$
$\tilde{v}_l$		$\approx v_l$	$\approx v_l$
$\tilde{V}_l$	Dyadics	$< 2^{-14} v_l$	$< 2^{-9} v_l$
$\tilde{V}_l$	Lookup table	$< 2^{-12} v_l$	$< 2^{-8} v_l$

Table 7.3.2 – Approximate variance relations that are observed for our example simulation. (cf. Figure 7.3.1).

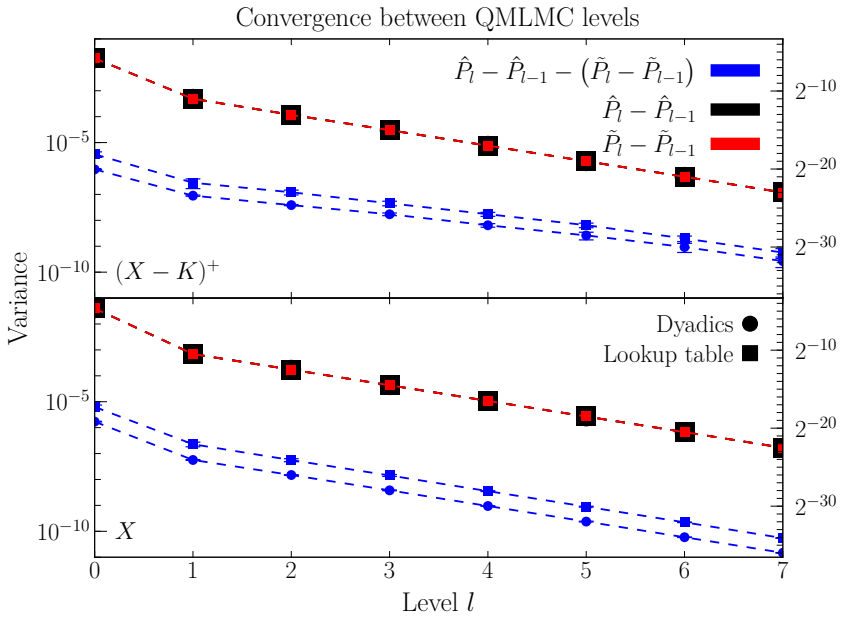


Figure 7.3.1 – The variance structure of a quantised nested multilevel Monte Carlo scheme, (similar to Figure 7.2.6), for different approximations of $\Phi^{-1}(\cdot)$. We show the behaviour for an equipartitioned piecewise constant approximation implemented using a lookup table with 1024 intervals, and a piecewise linear approximation with 16 dyadic intervals. **(Above)** A call option. **(Below)** The underlying stochastic process.

Using these values in (7.1.3) we can compute the additional savings which can be achieved, which for both our two payoffs evaluates to approximately $\tilde{T} \leq 0.18\,T$. This is an approximate time saving by a factor of 5.5, and in both situations the drop in variance is more than enough to realise almost all of the savings coming from using the cheaper approximate random variables. In this calculation, the squared term within the maximum in (7.1.3), which contains the variance saving, evaluates to $\frac{1}{0.79}$, and hence at worst approximately 80% of the performance savings from switching to the approximate random variables can be realised (at best this becomes 96%). The results for both the dyadic and lookup table approximations are summarised in Table 7.3.3.

To give an idea of how many more simulations we must perform and how often we need to perform a multilevel correction we can compare m_l, $\tilde{m}_l$, and $\tilde{M}_l$. These evaluate to

$$\tilde{m}_l \approx m_l \qquad \text{and} \qquad \tilde{M}_l = \tilde{m}_l \sqrt{\frac{\tilde{V}_l \tilde{c}_l}{\tilde{v}_l \tilde{C}_l}} \approx \begin{cases} \frac{1}{362} \tilde{m}_l & \text{if } P(X_T) \coloneqq X_T \\ \frac{1}{64} \tilde{m}_l & \text{if } P(X_T) \coloneqq (X_T - K)^+. \end{cases} \tag{7.3.1}$$

Description	Speedup (efficiency)	
	X_T	$(X_T - K)^+$
Dyadics	6.7 (96%)	5.5 (79%)
Lookup table	6.4 (92%)	5.0 (72%)

Table 7.3.3 – Approximate savings which could be expected for our example simulation.

We can see from this that we need to do a negligible number of additional simulations, and we very infrequently need to evaluate one of the expensive multilevel corrections.

Hence, for either payoff, using a dyadic approximation, a factor of 5.5 speed improvement is readily achievable by switching to approximate random variables, and this can be achieved with a simple implementation of using a fixed ratio of crude simulations to corrections, which could be set to e.g. 60–1.

Chapter 8

Finite-precision effects

In this chapter we introduce two models for finite-precision rounding errors. We will first review a short *ad hoc* model proposed by Arciniega and Allen [11] which presents an additive noise model for the roundoff errors, which is intended to represent an average-case error bound. For finite-precision representations with r bits of precision, they find the mean squared error decreases as $\mathcal{O}(N2^{-2r})$. Similarly we will present a more thorough treatment of roundoff errors using the methodology established by Omland [166] which obtains a worst case mean squared error bound of $\mathcal{O}(N^2 2^{-2r})$. Taking the model presented by Omland [166], we will show that the error process introduced by this model approximately recovers the model by Arciniega and Allen [11], but with the introduction of second order terms whose cumulative effect is non-negligible.

8.1 An ad hoc additive error model

Arciniega and Allen [11] provide a statistical analysis for the roundoff error the arises when accounting for floating-point arithmetic with the regular Euler-Maruyama scheme. By considering the usual Euler-Maruyama scheme in (5.1.2), they propose that the effects from the various floating-point arithmetic operations in each iteration can be modelled by an additive error ϵ_n. Arciniega and Allen [11] assume these errors are i.i.d. and follow a zero mean Gaussian distribution with variance $\mathbb{V}(\epsilon) \leq C\varrho^2$, where C is some finite and strictly positive constant and $\varrho \propto 2^{-r}$, (and hence ϱ is proportional to the unit roundoff error). This gives rise to an approximation $\check{X}$, produced by the modified Euler-Maruyama scheme

$$\check{X}_{n+1} = \check{X}_n + a(\tau_n, \check{X}_n)\delta + b(\tau_n, \check{X}_n)\sqrt{\delta}Z_n + \epsilon_n. \tag{8.1.1}$$

Using this scheme Arciniega and Allen [11] show the error bound in Theorem 8.1.1.

Theorem 8.1.1. *[11, Theorem 2.2] Under Assumptions 5.1.1 to 5.1.6 where the constants do not depend on δ, we use the adapted Euler-Maruyama scheme in (8.1.1). With i.i.d. errors $\epsilon_n \sim \mathcal{N}$ with $\mathbb{E}(\epsilon) = 0$ and $\mathbb{V}(\epsilon) \leq C\varrho^2$, where C is a finite and strictly positive constant and $\varrho \propto 2^{-r}$, then there exists a constant $K > 0$, independent of N and r, such that we have the bound $\mathbb{E}(\sup_{n\leq N} |\hat{X}_n - \check{X}_n|^2) \leq KN2^{-2r}$.*

Remark 8.1.1.1. The original theorem as stated by Arciniega and Allen [11, Theorem 2.2] did not include the supremum, and the proof followed by a recursive construction for the accumulated error. Their proof is easily extended to include the supremum and make use of Grönwall's discrete inequalities (Lemma A.1.9), and then proceeds identically to the proof of Lemma 5.2.3.

Remark 8.1.1.2. The result from Theorem 8.1.1 gives an L^2-error proportional to $N^{1/2}$, which is the standard deviation we might anticipate from adding i.i.d. Gaussian distributed errors from the central limit theorem.

8.2 Finite-precision roundoff error

In this section we will need to introduce several items of notation for handling finite-precision representations of values. We largely follow the set-up established by Omland [166], and consider a base-2 finite-precision representation where we will assume that we have r bits to represent the mantissa, and that all numbers are stored using standard floating-point notation. For an infinite-precision variable x, we denote its finite-precision representation as $\bar{x}^{(r)}$, where for brevity we will usually omit the dependence on r and simply use $\bar{x}$, where the precision used should be clear from the context. We will assume that we have a suitable number of exponent bits in the floating-point representation that no operations involving finite-precision variables will result in overflow or underflow, and nor will we require denormalised representations. For more detail regarding floating-point arithmetic we recommend Higham [110, Chapter 2] and Trefethen and Bau [199, Lecture 13].

To transform an infinite-precision variable x into its finite-precision representation $\bar{x}$ we will need some mapping between the real numbers $\mathbb{R}$, and those which can be represented in finite-precision $\bar{\mathbb{R}}$. To achieve this we will introduce the roundoff operator $\mathscr{R}(\cdot; r) : \mathbb{R} \to \bar{\mathbb{R}}$, which we will typically abbreviate as $\mathscr{R}(\cdot)$, where $\bar{x} \coloneqq \mathscr{R}(x)$. Possible rounding operators include the rounding modes: round to nearest even, round to zero, round to positive infinity, and round to negative infinity. These rounding modes are largely self-explanatory, with the exception of round to nearest even. For this rounding mode, if we are exactly half way between two adjacent floating point numbers, then we round to the number whose final digit in their mantissa is even (zero) [110, 2.1, page 38]. Without loss of generality, we will presume (unless otherwise stated) that we will be rounding to the nearest even, as this is an unbiased rounding mode and is typically the default rounding mode implemented on most hardware. Notice that for any rounding mode $\mathscr{R}(\cdot)$ has no effect when acting on members of $\bar{\mathbb{R}}$, where $\mathscr{R}(\bar{x}) = \bar{x}$ for all $\bar{x} \in \bar{\mathbb{R}}$.

Theorem 8.2.1. *[110, Theorem 2.2] For $x \in \mathbb{R}$ there exists an ε such that $\mathscr{R}(x) = x(1+\varepsilon)$, where $|\varepsilon| < \varrho$, where ϱ is the unit roundoff.*

8.2.1 Arithmetic roundoff

We can perform various binary operations $\circledast$ in finite-precision, where $\circledast \in \{\oplus, \ominus, \otimes, \oslash\}$, (corresponding to the infinite-precision arithmetic operations $* \in \{+, -, \times, \div\}$ respectively). We assume the operation is carried out in infinite precision, and the resulting answer is then rounded, e.g. $x \circledast y \coloneqq \mathscr{R}(x * y)$. For these binary operations, we have the *standard rounding model* [110, 2.2, (2.4)] [199, page 99, (13.7)]

$$\mathscr{R}(x \circledast y) = (x * y)(1 + \varepsilon), \tag{8.2.1}$$

where $|\varepsilon| \leq \varrho$ for $\circledast \in \{\oplus, \ominus, \otimes, \oslash\}$, and similarly for $*$, (where $y \neq 0$ when we are performing the division operation). Higham [110] presents Lemma 8.2.2 for bounding the roundoff error from composite operations, and in recent work by Higham and Mary [111], they propose Assumption 8.2.1 as a probabilistic model for rounding errors.

Lemma 8.2.2. *[110, Lemma 3.1] If $|\varepsilon_i| \leq \varrho$ and $s_i \in \{\pm 1\}$ for $i \in \{1, 2, \ldots, n\}$, and $n\varrho < 1$, then $\prod_{i=1}^{n}(1+\varepsilon_i)^{s_i} = 1 + \theta_n$, where $|\theta_n| \leq \gamma_n$ and $\gamma_n \coloneqq \frac{n\varrho}{1-n\varrho}$.*

Assumption 8.2.1 (probabilistic relative errors). [111, Model 2.1, page 3] In model (8.2.1), the quantities ε associated with every pair of operands can be modelled as independent zero mean random variables.

Given the deterministic nature of the IEEE round to nearest even rounding mode, Higham and Mary [111] remark: "*[Assumption 8.2.1] is clearly not always realistic*". In the

material we will later present, we will discuss the relation and applicability of this model to our setting with the Euler-Maruyama scheme.

We also require finite-precision function approximations which account for the error introduced from a function's finite-precision representation and the subsequent floating-point operations involved. To this end, for an infinite-precision function $f(\cdot)$, we introduce the floating-point approximation $\bar{f}(\cdot)$, where we bound the error that the floating-point approximation introduces by $|\bar{f}(x) - f(x)| \leq C2^{-r}(1 + |f(x)|)$, for some finite positive constant C. We notice that this bounds both the absolute and the relative error. We assume this holds for the non-autonomous functions for the drift and volatility, such that we have the bound $|\bar{a}(t,x) - a(t,x)| \leq C2^{-r}(1 + |a(t,x)|)$, and similarly for $\bar{b}$.

Remark 8.2.2.1. For a function to satisfy this it implicitly requires a Lipschitz (or differentiability) condition, both with respect to time and space. However, the drift and volatility functions are only $\frac{1}{2}$-Hölder continuous with respect to time. Hence we will strengthen assumptions of temporal $\frac{1}{2}$-Hölder continuity to Lipschitz continuity.

Assumption 8.2.2 (Lipschitz continuity). There functions a and b are Lipschitz continuous with respect to both their spatial and temporal dependence.

It is also possible to extend these bounds, such that the definition and construction of a floating-point function approximation takes into account the rounding of its arguments. As such we introduce a modified version of (8.2.2) which reads

$$\left|\bar{a}(\bar{t},\bar{x}) - a(t,x)\right| \leq C2^{-r}(1 + |a(t,x)|), \tag{8.2.2}$$

and similarly for $\bar{b}$.

It is useful to note that the IEEE standard [118, 5.2] requires the basic operations $\{+, -, \times, \div, \sqrt{}\}$ return the floating-point number to the nearest exact result with respect to the appropriate machine rounding mode [201, page 15]. This implies that for $\overline{\text{sqrt}}(\cdot)$ we only have a relative error of the form (8.2.1), rather than the more general error model from (8.2.2), and hence $\overline{\text{sqrt}}(x) = \sqrt{x}(1+\varepsilon)$.

Lemma 8.2.3. *If we assume the error model in (8.2.2) and Assumption 5.1.3, then $\bar{a}$ and $\bar{b}$ also have linear growth.*

Proof. The proof follows identically to that given by Omland [166, Lemma 4.1], although requires a trivial extension for nonautonomous functions. **QED**

Lastly, we will recognise that where appropriate most modern computer hardware is capable of performing *fused multiple add* (FMA) instructions, where these are done exactly and then rounded, and hence we can utilise the result $\bar{a} \oplus \bar{b} \otimes \bar{c} = \mathscr{R}(\bar{a} + \bar{b} \times \bar{c})$ [110, 2.6, page 46].

8.3 Finite-precision random number generators

When we deal with ideal random number generators, such as the uniform distribution or the Gaussian distribution, these produce random numbers in the range $[0, 1)$ or $\mathbb{R}$ respectively. However, the output from a finite-precision random number generator must be in $\bar{\mathbb{R}}$. Hence, ahead of addressing the convergence of any numeric schemes, we will require some means of quantifying the convergence of the random numbers produced by a finite-precision random number generator to those from an ideal one, including our approximate distributions. To achieve this we will use the Wasserstein metric, and some results from measure theory and probability theory. For readers less familiar with these topics, we recommend the reader to Kloeden and Platen [133, Chapter 2], Klebaner [131, Chapter 2], and Villani [203, Chapter 6].

We begin by letting $\mathcal{P}(\mathbb{R})$ denote the set of all probability measures μ on the measurable space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$, where $\mathcal{B}(\mathbb{R})$ is the *Borel σ-algebra* generated by $\mathbb{R}$ [131, page 29], and similarly $\mathcal{P}(\mathbb{R}^2)$ on the space $(\mathbb{R}^2, \mathcal{B}(\mathbb{R}^2))$. We can then define the *Wasserstein space* $\mathcal{P}_p(\mathbb{R}) \coloneqq \{\mu \in \mathcal{P}(\mathbb{R})\colon \int_{\mathbb{R}} |x|^p \,\mathrm{d}\mu(x) < \infty\}$, and for $\mu_1, \mu_2 \in \mathcal{P}_p(\mathbb{R})$ we can also define $\mathcal{P}(\mathbb{R}^2; \mu_1, \mu_2) \coloneqq \{\mu \in \mathcal{P}(\mathbb{R}^2)\colon \mu(e_i^{-1}(\cdot)) = \mu_i(\cdot) \text{ for } i \in \{1, 2\}\}$, where $e_1, e_2 \colon \mathbb{R}^2 \to \mathbb{R}$ are the projections of the first and second components respectively. Here $\mathcal{P}(\mathbb{R}^2; \mu_1, \mu_2)$ is the set of all probability measures on $(\mathbb{R}^2, \mathcal{B}(\mathbb{R}^2))$ which have the marginals μ_1 and μ_2 respectively. To measure the similarity of two distributions, we define the p-th order *Wasserstein metric* $W_p(\mu_1, \mu_2) \coloneqq (\inf\{\int_{\mathbb{R}^2} |x - y|^p \,\mathrm{d}\mu(x, y) : \mu \in \mathcal{P}(\mathbb{R}^2; \mu_1, \mu_2)\})^{1/p}$ for $\mu_1, \mu_2 \in \mathcal{P}_p(\mathbb{R})$ [203, Definition 6.1, page 93].

Definition 8.3.1 (Weak convergence in $\mathcal{P}_p$). A sequence of measures $\{\mu_k\}_{k\in\mathbb{N}} \in \mathcal{P}_p(\mathbb{R})$ *weakly converges in* $\mathcal{P}_p(\mathbb{R})$ to $\mu \in \mathcal{P}_p(\mathbb{R})$ if for any bounded and continuous function φ we have $\lim_{k\to\infty} \int \varphi \,\mathrm{d}\mu_k = \int \varphi \,\mathrm{d}\mu$.

Theorem 8.3.1. *[203, page 108] Let $p \in [1, \infty)$, and let there be a sequence of measures $\{\mu_k\}_{k\in\mathbb{N}} \in \mathcal{P}_p(\mathbb{R})$ and $\mu \in \mathcal{P}_p(\mathbb{R})$. If μ_k weakly converges in $\mathcal{P}_p(\mathbb{R})$ to μ then this implies $W_p(\mu_k, \mu) \to 0$, and the reverse also holds true.*

We are now in a position to define probability measures using floating-point approximations and quantify the rate of convergence to the exact measure.

Definition 8.3.2. [47, pages 10 and 32] [166, page 22] The support $\mathrm{supp}(\cdot)$ of a measure μ is $\mathrm{supp}(\mu) = \mathbb{R}\backslash\mathcal{V}$, where $\mathcal{V}$ is the union of all open subsets of $\mathbb{R}$ which have zero μ-measure.

Definition 8.3.3. [166, Definition 3.15] A sequence of probability measures $\{\bar{\mu}^{(r)}\}_{r\in\mathbb{N}} \in \mathcal{P}_2(\mathbb{R})$ is called a *floating-point approximation* of $\mu \in \mathcal{P}_2(\mathbb{R})$ if there exists a finite constant $K > 0$ such that $\mathrm{supp}(\bar{\mu}^{(r)}) \subseteq \bar{\mathbb{R}}$ and $W_2(\bar{\mu}^{(r)}, \mu) \leq K2^{-r}$ for all $r \in \mathbb{N}$.

An important result which we will make use off is Lemma 8.3.2 by Omland [166].

Lemma 8.3.2. *[166, Lemma 3.16] Given a measure $\mu \in \mathcal{P}_2(\mathbb{R})$ and a floating-point approximation $\bar{\mu}^{(r)}$, there exists a probability space with a real valued random variable Y with distribution μ, and a finite constant $K > 0$, such that for all precisions $r \in \mathbb{N}$ and random variables $\bar{Y}^{(r)} \in \bar{\mathbb{R}}$ with distribution $\bar{\mu}^{(r)}$, that $\mathbb{E}(|Y - \bar{Y}^{(r)}|^2) \leq K2^{-2r}$.*

In the work by Omland [166, 3.3], they show the existence of floating-point approximations to the uniform, exponential, and Gaussian distribution. The floating-point approximation $\bar{\mu}_{\mathcal{U}}^{(r)}$ of the uniform distribution $\mu_{\mathcal{U}}$ is given by the discrete uniform distribution on the set $\{n2^{-r}\}_{n=0}^{2^r-1}$, and the approximation $\bar{\mu}_{\mathcal{N}}$ to the Gaussian distribution $\mu_{\mathcal{N}}$ if formed by a construction following the Box-Muller scheme [188, 2.3.1] [92, 2.3.2]. Denoting the Gaussian distribution as $\mu_{\mathcal{N}}$, and the approximate Gaussian distribution as $\mu_{\tilde{\mathcal{N}}}$, then in the following we postulate the existence of a floating-point approximation $\bar{\mu}_{\tilde{\mathcal{N}}}$ of $\mu_{\tilde{\mathcal{N}}}$.

Assumption 8.3.1. For a symmetric approximate Gaussian distribution $\mu_{\tilde{\mathcal{N}}}$, there exists a floating-point approximation $\bar{\mu}_{\tilde{\mathcal{N}}}^{(r)}$ to $\mu_{\tilde{\mathcal{N}}}$ having a Wasserstein convergence rate $W_2(\bar{\mu}_{\tilde{\mathcal{N}}}^{(r)}, \mu_{\tilde{\mathcal{N}}}) \leq C2^{-r}$ for any precision $r \geq r^*$ for some precision level r^* and constant C which is independent of r and r^*.

Corollary 8.3.2.1. $\mathbb{E}(|\tilde{Z} - \bar{Z}|^2) \leq K2^{-2r}$, *where $\tilde{Z} \sim \mu_{\tilde{\mathcal{N}}}$ and $\bar{Z} \sim \bar{\mu}_{\tilde{\mathcal{N}}}$.*

Proof. The result immediately follows from Lemma 8.3.2. **QED**

8.4 The finite-precision Euler-Maruyama scheme

With the finite-precision framework established, we can define the approximate Euler-Maruyama scheme which uses finite-precision arithmetic as

$$\bar{X}_{n+1} = \bar{X}_n \oplus \Big(\Big(\bar{a}\Big(\bar{\tau}_n, \bar{X}_n \Big) \otimes \bar{\delta} \Big) \oplus \Big(\bar{b}\Big(\bar{\tau}_n, \bar{X}_n \Big) \otimes \overline{\mathrm{sqrt}}\Big(\bar{\delta} \Big) \Big) \otimes \bar{Z}_n \Big), \tag{8.4.1}$$

where $\bar{X}_0 = \mathscr{R}(X_0)$, and $\bar{Z}_n \sim \bar{\mu}_{\tilde{\mathcal{N}}}$, where $\bar{\mu}_{\tilde{\mathcal{N}}}$ is the finite-precision approximate Gaussian random variable. It is important to note that $\bar{Z}_n$ is not the rounded version of the Gaussian increment and $\bar{Z}_n \neq \mathscr{R}(\tilde{Z}_n)$. (This model with $\bar{Z}_n \sim \bar{\mu}_{\mathcal{N}}$ was presented by Omland [166]).

Compared to the strong error bound $\mathcal{O}(N^{1/2}2^{-r})$ shown by Arciniega and Allen [11] in Theorem 8.1.1, Omland [166] showed a more rigorous but conservative bound $\mathcal{O}(N2^{-r})$ in Theorem 8.4.1.

Theorem 8.4.1. *[166, Theorem 4.8] Using Assumptions 5.1.2 and 5.1.3 and (8.2.2), and assuming (5.1.1) is an autonomous SDE, then there exists a finite constant $C > 0$, such that for all $N \in \mathbb{N}$, integers $n \in \{0, 1, \ldots, N\}$, and precisions $r \geq 2 + \lceil \log_2(n+1) \rceil$, that (8.4.1) using $\bar{Z} \sim \bar{\mu}_{\mathcal{N}}$ has the strong error bound $\mathbb{E}(|\bar{X}_n - \hat{X}_n|^2)^{1/2} \leq Cn2^{-r}$.*

Comparing Theorem 8.4.1 from Omland [166] with Theorem 8.1.1 from Arciniega and Allen [11], we can see that Theorem 8.4.1 results in a linear growth in the accumulated rounding error, which is a worst case bound. Our goal in the proceeding analysis will be twofold. Firstly, we will seek to re-introduce approximate Gaussian random variables where $\bar{Z} \sim \bar{\mu}_{\tilde{\mathcal{N}}}$. Secondly, and more importantly, we will seek to incorporate a model for the roundoff error, similar to Assumption 8.2.1 by Higham and Mary [111], into the finite-precision Euler-Maruyama setup established by Omland [166]. This will allow us to recover and more rigorously justify the model from Arciniega and Allen [11], but with an extension to incorporate possibly systematic rounding effects, suitable for analysis using our previous framework from Lemma 5.2.3.

At this point it is useful to collect together the various assumptions and results that we will take forward which relate to our finite-precision framework (in addition to our usual assumptions):

Assumption 8.4.1 (IEEE). The IEEE standard [119] for floating-point arithmetic is used.

Assumption 8.4.2 (unit roundoff). The rounding error is limited by the unit roundoff ϱ such that Theorem 8.2.1 holds.

Assumption 8.4.3 (round to nearest even). We are using the round to nearest even rounding mode such that $\varrho = 2^{-r}$.

Assumption 8.4.4 (floating-point functions). We are using floating-point function approximations for all functions such that (8.2.2) holds.

Assumption 8.4.5. The finite-precision implementation of the approximate Gaussian distribution is rotationally symmetric, and thus $\mathbb{E}(\bar{Z} - \tilde{Z}) = 0$.

Lemma 8.4.2. $\mathbb{E}(|\bar{X}|^2) < \infty$.

Lemma 8.4.3. $\mathbb{E}(|\bar{Z} - \tilde{Z}|^p) \leq \mathbb{E}(|\tilde{Z}|^p)$ *for any finite integer* $p \geq 1$.

8.4.1 Mirroring the Arciniega and Allen model

It is anticipated that the model by Arciniega and Allen [11] can be recovered from that proposed by Omland [166], or a very similar model under appropriate assumptions. To this end we look to expand the model from Omland [166] and see the effects of arithmetic roundoff

within the Euler-Maruyama update. We begin by re-writing (8.4.1) as $\bar{X}_{n+1} = \bar{X}_n \bigoplus (A_n \bigoplus B_n)$, where $A_n \coloneqq \bar{a}(\bar{\tau}_n, \bar{X}_n) \otimes \bar{\delta}$ and $B_n \coloneqq (\bar{b}(\bar{\tau}_n, \bar{X}_n) \otimes \overline{\text{sqrt}}(\bar{\delta})) \otimes \bar{Z}_n$.

The first addition will produce an absolute error η'_n from $A_n \bigoplus B_n = A_n + B_n + \eta'_n$, where η'_n will be of a size comparable with the unit roundoff for the larger of A_n and B_n, which is B_n. Thus, from using (8.2.2), Assumption 5.1.3, and Lemma 8.2.3, we obtain $|\eta'_n| \sim |B_n \varrho| = \mathcal{O}(\sqrt{\delta}\,\varrho(1 + |\bar{X}_n|))$, and after performing this first floating-point addition we will be left with

$$\bar{X}_{n+1} = \bar{X}_n \bigoplus (B_n + A_n + \eta'_n), \tag{8.4.2}$$

where we have written $(B_n + A_n + \eta'_n)$ in order of decreasing magnitudes.

For the remaining addition operation, we expect our SDE to be scaled such that $X_n = \mathcal{O}(1)$, and thus we expect $|X_n| \gg |B_n|$. The nett result from the remaining floating-point addition then is that this will produce a second absolute arithmetic error η_n where $|\eta_n| \sim |X_n \varrho| = \mathcal{O}(\varrho(1 + |\bar{X}_n|))$. Our finite-precision Euler-Maruyama update will then become $\bar{X}_{n+1} \approx \bar{X}_n + B_n + A_n + \eta_n + \eta'_n$, where again we have written the contributions in order of decreasing magnitudes. We can identify this as the usual Euler-Maruyama update with two dominant sources of error. The first is η'_n, arising from the addition of the drift term to the volatility term. The second is η_n, arising from the addition of this sum to the underlying process. We expect $|\eta'_n| = \mathcal{O}(\varrho\sqrt{\delta}(1 + |\bar{X}_n|))$ and $|\eta_n| = \mathcal{O}(\varrho(1 + |\bar{X}_n|))$, and thus $|\eta_n| \gg |\eta'_n|$. If we then allow for the inclusion of other higher order contributions, we can then see that we expect to obtain

$$\bar{X}_{n+1} = \bar{X}_n + B_n + A_n + \eta_n + \eta'_n + \eta''_n. \tag{8.4.3}$$

In a similar spirit to Assumption 8.2.1 by Higham and Mary [111], we propose Assumption 8.4.6 (which we justify with Lemma 8.4.4), where we make the modelling assumption that the composite effects of rounding errors can be well described by (8.4.3) where η_n is an unbiased zero mean (martingale) increment with $\mathbb{E}(|\eta_n|) \leq C\varrho(1 + \mathbb{E}(|\bar{X}_n|))$, η'_n is a possibly biased/systematic non-zero mean (non-martingale) increment with $\mathbb{E}(|\eta'_n|) \leq C'\varrho\sqrt{\delta}(1 + \mathbb{E}(|\bar{X}_n|))$, and $\eta''_n = 0$, where C and C' are constants.

Assumption 8.4.6 (rounding errors in the Euler-Maruyama scheme). For the Euler-Maruyama scheme using exact (or approximate) random variables Z (or $\tilde{Z}$), the composite effects of rounding error introduce two dominant sources of error, η and η', where at each step we have

$$\bar{X}_{n+1} = \bar{X}_n + a(\tau_n, \bar{X}_n)\delta + b(\tau_n, \bar{X}_n)\sqrt{\delta}\underbrace{\tilde{Z}_n}_{(\text{or } Z_n)} + \eta_n + \eta'_n, \tag{8.4.4}$$

where the larger of these is $\eta_n = \mathcal{O}(\varrho(1 + |\bar{X}_n|))$, which is a martingale increment, and the smaller of these is $\eta'_n = \mathcal{O}(\varrho\sqrt{\delta}(1 + |\bar{X}_n|))$, which is a possibly non-martingale increment.

8.4.2 The leading order additive rounding error

Inspecting Assumption 8.4.6, the key modelling assumption requiring justification is the martingale nature of η_n. It is straightforward to reason that $\mathbb{E}(|\eta_n|) = \mathcal{O}(\varrho(1 + \mathbb{E}(|\bar{X}_n|)))$, which if we take $\mathbb{E}(|\bar{X}_n|) = \mathcal{O}(1)$ simplifies to $\mathbb{E}(|\eta_n|) = \mathcal{O}(\varrho)$. Thus, to justify η_n being a martingale increment it is sufficient to reason that $|\mathbb{E}(\eta_n)| \ll \mathbb{E}(|\eta_n|)$, which we achieve through Lemma 8.4.4.

Lemma 8.4.4. *Assuming $\mathbb{E}(|\bar{X}_n|) = \mathcal{O}(1)$, then the leading order absolute roundoff error η_n in (8.4.4) from Assumption 8.4.6 resulting from the floating-point addition in (8.4.2) has an expectation $\mathbb{E}(\eta_n) = \mathcal{O}(\varrho^2)$ when using exact random variables.*

Proof. Let us represent the floating-point addition operation in (8.4.2) by $\bar{\alpha} \bigoplus \bar{\beta}$, where $\bar{\alpha} \equiv \bar{X}_n = \mathcal{O}(1)$, $\beta \equiv A_n + B_n = \mathcal{O}(\sqrt{\delta})$, and $\bar{\beta} = \mathscr{R}(\beta)$. The absolute rounding error $\eta \equiv \eta_n$

(dropping the subscript for brevity) is given by

$$\eta \coloneqq (\bar{\alpha} \bigoplus \bar{\beta}) - (\bar{\alpha} + \bar{\beta}) \equiv \Big(\mathscr{R}(\bar{\alpha} + \beta) - (\bar{\alpha} + \beta)\Big) - \Big(\mathscr{R}(\beta) - \beta\Big) + \Big(\mathscr{R}\Big(\bar{\alpha} + \bar{\beta}\Big) - \mathscr{R}(\bar{\alpha} + \beta)\Big). \quad (8.4.5)$$

The three parenthesised terms we will bound in turn. The preference for this representation is that we can readily assume β to have a smooth probability density function ρ (resembling a Gaussian distribution), which will facilitate taking expectations.

Inspecting the first term in (8.4.5), we can see this is the absolute error resulting from rounding the quantity $\bar{\alpha} + \beta$. Without much loss of generality, as we have assumed $\bar{\alpha} = \mathcal{O}(1)$, let us suppose $\bar{\alpha} \in (1, 2)$. Using IEEE floating-point representation, the set of representable numbers $(1, 2) \cap \bar{\mathbb{R}}$ will all be equally spaced. The quantity $z \coloneqq \bar{\alpha} + \beta$ will fall inside some interval $I_y \coloneqq [y - \varsigma, y + \varsigma]$, where $y \pm \varsigma$ are adjacent floating-point numbers. We can then evaluate the expectation $\mathbb{E}((\bar{z} - z)\mathbb{1}_{\{z \in I_y\}})$, where we either have $z < y$ and we round down, $z > y$ and we round up, or $z = y$ and we are in a tie-break rounding situation. Hence this becomes

$$\mathbb{E}\Big((\bar{z} - z)\mathbb{1}_{\{z \in I_y\}}\Big) = \mathbb{E}\Big((\bar{z} - z)\mathbb{1}_{\{z \in [y-\varsigma, y)\}}\Big) + \mathbb{E}\Big((\bar{z} - z)\mathbb{1}_{\{z \in (y, y+\varsigma]\}}\Big) + \mathbb{E}\Big((\bar{z} - z)\mathbb{1}_{\{z = y\}}\Big) \quad (8.4.6)$$

as $\{z = y\}$ has zero measure this leaves

$$= \int (\bar{z} - z)\mathbb{1}_{\{z \in [y-\varsigma, y)\}} \mathbb{P}(\mathrm{d}z) + \int (\bar{z} - z)\mathbb{1}_{\{z \in (y, y+\varsigma]\}} \mathbb{P}(\mathrm{d}z) \quad (8.4.7)$$

$$= \int_{y-\varsigma}^{y} ((y - \varsigma) - z)\rho(z - \bar{\alpha})\,\mathrm{d}z + \int_{y}^{y+\varsigma} ((y + \varsigma) - z)\rho(z - \bar{\alpha})\,\mathrm{d}z \quad (8.4.8)$$

$$= \int_{-\varsigma}^{0} (-\varsigma - z)\rho(y + z - \bar{\alpha})\,\mathrm{d}z + \int_{0}^{\varsigma} (\varsigma - z)\rho(y + z - \bar{\alpha})\,\mathrm{d}z \quad (8.4.9)$$

$$= \int_{0}^{\varsigma} (\varsigma - z)(\rho(y + z - \bar{\alpha}) - \rho(y - z - \bar{\alpha}))\,\mathrm{d}z \quad (8.4.10)$$

as ρ is smooth we use a Taylor expansion

$$= \int_{0}^{\varsigma} (\varsigma - z)(2z\rho'(y - \bar{\alpha}) + \mathcal{O}(z^3))\,\mathrm{d}z \quad (8.4.11)$$

$$= \int_{0}^{\varsigma} (\varsigma - z)(2z\rho'(y - \bar{\alpha}) + \mathcal{O}(z^3))\,\mathrm{d}z \quad (8.4.12)$$

$$= \frac{\varsigma^3 \rho'(y - \bar{\alpha})}{3} + \mathcal{O}\Big(\varsigma^5\Big) \quad (8.4.13)$$

$$\approx \frac{\varsigma^3 \rho'(y - \bar{\alpha})}{3}. \quad (8.4.14)$$

Using

$$\mathbb{P}(z \in I_y) \approx 2\varsigma\rho(y - \bar{\alpha}) \quad (8.4.15)$$

and the definition of conditional expectation (Definition A.2.1), then it follows that

$$\mathbb{E}(\bar{z} - z \,|\, z \in I_y) \approx \frac{\varsigma^2 \rho'(y - \bar{\alpha})}{6\rho(y - \bar{\alpha})}. \quad (8.4.16)$$

Combining this with the law of total expectation (Theorem A.2.1), then in the approximate limit $I_y \to \mathrm{d}I_y$ we obtain

$$\mathbb{E}(\bar{z} - z) \approx \int \mathbb{E}(\bar{z} - z \,|\, z \in \mathrm{d}I_y)\mathbb{P}(\mathrm{d}I_y) \quad (8.4.17)$$

$$\approx \frac{1}{6}\int_{-\infty}^{\infty} \varsigma^2 \rho'(y - \bar{\alpha})\,\mathrm{d}y \quad (8.4.18)$$

the dominant contribution will be for $y = \mathcal{O}(1)$, and hence if we were to make the approximation that ς is a constant with respect to y, and thus $\varsigma(y) \approx \varrho$, this would give $\int_{-\infty}^{\infty} \varsigma^2 \rho'(y - \bar{\alpha})\,\mathrm{d}y \approx \varrho^2 \int_{-\infty}^{\infty} \rho'(y-\bar{\alpha})\,\mathrm{d}y = \varrho^2\,[\rho(y-\bar{\alpha})]_{-\infty}^{\infty} = 0$. Thus we see that ς completely being a constant would result in the expectation being zero, which we see is slightly too aggressive an approximation for our desired bound. However, for IEEE floating-point numbers, ς is a constant between dyadic intervals, and thus only changes when y is a power of two. Approximating the integration domain with only the positive real numbers we obtain

$$\approx \frac{1}{6}\int_0^{\infty} \varsigma^2 \rho'(y - \bar{\alpha})\,\mathrm{d}y \tag{8.4.19}$$

$$\approx \frac{1}{6}\sum_{k=-\infty}^{\infty} \int_{2^k}^{2^{k+1}} \varsigma^2 \rho'(y - \bar{\alpha})\,\mathrm{d}y \tag{8.4.20}$$

ς is a constant in these integration domains, where $2\varsigma \approx \varrho 2^k$, giving

$$\approx \frac{1}{6}\sum_{k=-\infty}^{\infty} \varrho^2 2^{2k-2} \int_{2^k}^{2^{k+1}} \rho'(y - \bar{\alpha})\,\mathrm{d}y \tag{8.4.21}$$

$$\approx \frac{1}{6}\sum_{k=-\infty}^{\infty} \varrho^2 2^{2k-2} \Big[\rho(y - \bar{\alpha})\Big|_{2^k}^{2^{k+1}} \tag{8.4.22}$$

$$\approx \frac{1}{6}\varrho^2 \sum_{k=-\infty}^{\infty} 2^{2k-2} \Big[\rho(y - \bar{\alpha})\Big|_{2^k}^{2^{k+1}} \tag{8.4.23}$$

as ρ is approximately a Gaussian this summation will converge to a constant $\mathcal{O}(1)$, giving

$$\approx \mathcal{O}\Big(\varrho^2\Big). \tag{8.4.24}$$

We can see from this result that the first term in (8.4.5) has an expectation $\mathcal{O}(\varrho^2)$. By an identical line of reasoning we can arrive at the same result for the second term in (8.4.5) (which can be achieved by taking $\bar{\alpha} \to 0$ in our previous analysis). This then leaves the final term in (8.4.5).

For the final term in (8.4.5), in most scenarios $\bar{\alpha}+\bar{\beta}$ and $\bar{\alpha}+\beta$ will round to the same number, giving zero error. These will only possibly round to different numbers if $\bar{\alpha}+\bar{\beta}$ requires a tie-break rounding condition. Introducing the slightly larger interval $I'_y \coloneqq [y - 2\varsigma, y + 2\varsigma]$, where $y - 2\varsigma$, y, and $y + 2\varsigma$ are all representable, without loss of generality we assume y is even and $y \pm 2\varsigma$ are odd. Given $\beta = \mathcal{O}(\sqrt{\delta})$ and $\bar{\alpha} = \mathcal{O}(1)$, we know that $|\beta - \bar{\beta}| \leq \varsigma'$ where $\varsigma' \ll \varsigma$. Keeping our definition $z \coloneqq \bar{\alpha} + \beta$ and introducing $\zeta \coloneqq \bar{\alpha} + \bar{\beta}$, then the discrete set of values ζ can take has a much finer granularity than the three representable numbers in I'_y, namely $\zeta \in \{y \pm n\varsigma'\} \cap I'_y$ for integers $n \in \mathbb{N}$. We display the set of values ζ can take in I'_y in Figure 8.4.1, where we demonstrate several possible rounding scenarios.

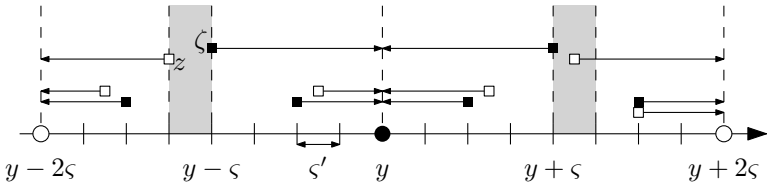


Figure 8.4.1 – The set of possible values for ζ in I'_y, demonstrating when the round to nearest even tie-break causes rounding error in the shaded regions. We denote even representable values using a solid circular marker (●), odd values with a hollow circular marker (○), ζ with a solid square (■), and z with a hollow square (□). Arrows show the values these are rounded to with the round to nearest even rounding mode.

Inspecting Figure 8.4.1, we can see that in most situations ζ and z round to the same number. These only round to different numbers when ζ lies on a tie-break value and z takes a different value and would be rounded to an odd number, as indicated in Figure 8.4.1.

Thus, for the final term in (8.4.5) we have the expectation

$$\mathbb{E}\Big(\big(\bar{\zeta}-\bar{z}\big)\mathbb{1}_{\{z\in I'_y\}}\Big)=\mathbb{E}\Big(\big(\bar{\zeta}-\bar{z}\big)\mathbb{1}_{\{\zeta=y-\varsigma\}}\Big)+\mathbb{E}\Big(\big(\bar{\zeta}-\bar{z}\big)\mathbb{1}_{\{\zeta=y+\varsigma\}}\Big) \tag{8.4.25}$$

z will be closer to the odd values

$$\begin{aligned}&=\mathbb{E}\Big(\big(\bar{\zeta}-\bar{z}\big)\mathbb{1}_{\{z\in[y-\varsigma-\varsigma',y-\varsigma)\}}\mathbb{1}_{\{\zeta=y-\varsigma\}}\Big) \\ &\quad+\mathbb{E}\Big(\big(\bar{\zeta}-\bar{z}\big)\mathbb{1}_{\{z\in(y+\varsigma,y+\varsigma+\varsigma']\}}\mathbb{1}_{\{\zeta=y+\varsigma\}}\Big)\end{aligned} \tag{8.4.26}$$

the first expectation will round $\zeta\to y$ and $z\to y-2\varsigma$, giving a nett rounding error of 2ς, and the second expectation will round $\zeta\to y$ and $z\to y+2\varsigma$ giving a rounding error of -2ς, and thus these expectations give rise to the integrals

$$=\int_{y-\varsigma-\varsigma'}^{y-\varsigma}2\varsigma\mathbb{P}(\mathrm{d}z)+\int_{y+\varsigma}^{y+\varsigma+\varsigma'}-2\varsigma\mathbb{P}(\mathrm{d}z) \tag{8.4.27}$$

$$=\int_{y-\varsigma-\varsigma'}^{y-\varsigma}2\varsigma\rho(z-\bar{\alpha})\,\mathrm{d}z+\int_{y+\varsigma}^{y+\varsigma+\varsigma'}-2\varsigma\rho(z-\bar{\alpha})\,\mathrm{d}z \tag{8.4.28}$$

$$=2\varsigma\int_0^{\varsigma'}\big(\rho(z+y-\varsigma-\varsigma'-\bar{\alpha})-\rho(z+y+\varsigma-\bar{\alpha})\big)\,\mathrm{d}z \tag{8.4.29}$$

performing a Taylor series approximation and keeping the leading order contribution gives

$$\approx 2\varsigma\int_0^{\varsigma'}(-2\varsigma-\varsigma')\rho'(y-\bar{\alpha})\,\mathrm{d}z \tag{8.4.30}$$

$$\approx 2\varsigma\Big(-2\varsigma\varsigma'-(\varsigma')^2\Big)\rho'(y-\bar{\alpha}). \tag{8.4.31}$$

If we again use Definition A.2.1 obtain $\mathbb{E}(\bar{\zeta}-\bar{z}\mid z\in I'_y)$, and combined with the law of total expectation (Theorem A.2.1) and the bound $\varsigma'\ll\varsigma$, this can be bound identically to the first two terms (contributions where $\varsigma'\geq\varsigma$ are negligible). Hence we have $\mathbb{E}(\eta)=\mathcal{O}(\varrho^2)$, which completes the proof. **QED**

Corollary 8.4.4.1. *Lemma 8.4.4 holds when using approximate Gaussian random variables.*

Proof. The proof follows similarly to that of Lemma 8.4.4, with the exception that ρ exists almost everywhere, is bounded, has finitely many discontinuities, and vanishes sufficiently fast. **QED**

8.4.3 The accumulated rounding error

Using Assumption 8.4.6 we can directly see from Lemma 5.2.3 that this will corresponds to $s=-\frac{1}{2}$ from both the martingale increment term η_n and the non-martingale increment term η'_n. Furthermore, the coefficients c_1 and c_2 in Lemma 5.2.3 are both proportional to $\varrho(1+|\bar{X}_n|)$, and thus from Lemma 5.2.3 we can directly obtain $\mathbb{E}(\sup_{n\leq N}|\tilde{X}_n-\bar{X}_n|^2)\leq C\varrho^2(1+\mathbb{E}(|\bar{X}_N|^2))\delta^{-1}$. Using the relation that $N\delta=T-t_0$ (and hence $\delta^{-1}=\mathcal{O}(N)$) we can see that this produces the result in Theorem 8.4.5.

Theorem 8.4.5. *Under Assumption 8.4.6, when $\bar{Z}_n\sim\bar{\mu}_{\tilde{\mathcal{N}}}$ we have*

$$\mathbb{E}\left(\sup_{n\leq N}\left|\tilde{X}_n-\bar{X}_n\right|^2\right)\leq CN\varrho^2\left(1+\mathbb{E}\left(\left|\bar{X}_N\right|^2\right)\right), \tag{8.4.32}$$

and when $\bar{Z}_n\sim\bar{\mu}_{\mathcal{N}}$ we have

$$\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}_n-\bar{X}_n\right|^2\right)\leq CN\varrho^2\left(1+\mathbb{E}\left(\left|\bar{X}_N\right|^2\right)\right). \tag{8.4.33}$$

Proof. The result follows from combining Assumption 8.4.6 with Lemma 5.2.3. **QED**

We can notice that Theorem 8.4.5 produces a near identical result to that obtained by Arciniega and Allen [11] in Theorem 8.1.1, except where we have allowed for the use of approximate random variables, and scaled our bound by the size of the underlying process. Furthermore, based on Lemma 8.4.4, we have provided a stronger justification for the leading order zero mean nature of the absolute rounding error than the ad hoc basis by Arciniega and Allen [11].

8.5 Multilevel Monte Carlo in finite-precision

The next natural extension to this analysis is to incorporate this into a multilevel Monte Carlo framework. We will have our normal high-precision temporal discretisation (possibly using exact random variables), and our lower precision calculation (likely using approximate random variables). Hence we have multiple possible four-way differences we can consider, where the most noteworthy are shown in Table 8.5.1.

Four-way difference	Change
$\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}$	Approximate random variables and finite-precision
$\tilde{X}_n^{\mathrm{f}} - \tilde{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}$	Finite-precision
$\bar{X}_n^{\mathrm{f},(r)} - \bar{X}_n^{\mathrm{c},(r)} - \bar{X}_n^{\mathrm{f},(r')} + \bar{X}_n^{\mathrm{c},(r')}$	High-precision to low-precision ($r > r'$)

Table 8.5.1 – Possible four-way differences which incorporate finite-precision arithmetic effects.

For our applications we will assume that the differences in the precision level are substantial enough (e.g. halving the precision) that compared to the lower precision level, the higher precision level can be regarded as near-infinite precision. Hence the appropriate model we will investigate is the first from Table 8.5.1, namely $\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}$.

Taking our finite-precision model in Assumption 8.4.6 and differencing this with the approximate Euler-Maruyama scheme (5.2.1) for fine and coarse levels, and using our equivalent four-point difference operator $\bar{\square} X \equiv \hat{X}^{\mathrm{f}} - \hat{X}^{\mathrm{c}} - \bar{X}^{\mathrm{f}} + \bar{X}^{\mathrm{c}}$ we obtain

$$\bar{\square} X_{n+1} = \bar{\square} X_n + \delta_{\mathrm{f}} \bar{\square} a_n + \sqrt{\delta_{\mathrm{f}}}\, \bar{\square}(b_n Z_n) + \bar{\square}(\eta_n + \eta_n'). \tag{8.5.1}$$

For the four-way differences of the drift and the volatility terms we obtain equivalent expressions to those we obtained when we first introduced the four-way difference in going from the exact to the approximate Gaussian random variables in Theorem 6.2.6. The difference in this case will be that instances of $\mathbb{E}(|Z - \tilde{Z}|)$ will now become $\mathbb{E}(|Z - \bar{Z}|)$, where because our approximations are not approaching machine precision we expect $\mathbb{E}(|Z - \tilde{Z}|) \approx \mathbb{E}(|Z - \bar{Z}|)$.

The new term of interest is the final four-way difference of the roundoff error, where $\bar{\square}(\eta_n + \eta_n') = \bar{\square}\eta_n + \bar{\square}\eta_n'$. To see why this is of particular interest, consider the first of these four-way differences $\bar{\square}\eta_n = \eta_n^{\mathrm{c}} - \eta_n^{\mathrm{f}} - \eta_{n+\frac{1}{2}}^{\mathrm{f}}$. A few observations should be made here. The first is that there are no contributions from the exact arithmetic operations. The second is that the fine path produces two contributions, which are the roundoff errors in the two fine increments. Furthermore, between the fine and the coarse levels the roundoff errors are independent ($\perp\!\!\!\perp$), and we have $\eta_n^{\mathrm{c}} \perp\!\!\!\perp \eta_n^{\mathrm{f}}$ and $\eta_n^{\mathrm{c}} \perp\!\!\!\perp \eta_{n+1/2}^{\mathrm{f}}$. For the two roundoff errors on the fine level, whilst not independent ($\eta_n^{\mathrm{f}} \not\perp\!\!\!\perp \eta_{n+1/2}^{\mathrm{f}}$), they still have zero conditional mean and $\mathbb{E}(\eta_{n+1/2}^{\mathrm{f}} \mid \eta_n^{\mathrm{f}}) = 0$. Thus we expect no cancellation with these terms and $\mathbb{E}(|\bar{\square}\eta_n|) = \mathcal{O}(\mathbb{E}(|\eta_n|))$. By the same line of reasoning we also expect an equivalent result for $\bar{\square}\eta_n'$.

With these observations at hand, to bound the L^p-norm of the multilevel Monte Carlo correction $\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}$ we will need to follow an identical procedure to the proof of Lemma 5.2.3. We will not be directly using Lemma 5.2.3, because we can see that there will be two different rates with which the terms decay or increase. There will be the usual $s = \frac{1}{2}$

decaying rate from introducing approximate random variables, with a coefficient proportional to the error from approximating the Gaussian distribution. In addition to this, there will be a new $s = -\frac{1}{2}$ rate from the rounding error, with a coefficient proportional to $\varrho(1 + \mathbb{E}(|\bar{X}_N^{\mathrm{f}}|))$.

Theorem 8.5.1. *Under Assumption 8.4.6 with $\bar{Z}_n \sim \bar{\mu}_{\tilde{\mathcal{N}}}$ we have*

$$\mathbb{E}\left(\sup_{n\leq N}\left|\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}\right|^p\right) \preceq \delta^{\frac{p}{2}}\mathbb{E}\left(\left|Z - \bar{Z}\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}} + \varrho^p\left(1 + \mathbb{E}\left(\left|\bar{X}_N^{\mathrm{f}}\right|^p\right)\right)\delta^{-\frac{p}{2}}. \quad (8.5.2)$$

Proof. In an identical manner to our proof of Lemma 5.2.3, if we define $\mathcal{E}_n \coloneqq \hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}$ and $\mathscr{Z}_N \coloneqq \mathbb{E}(\sup_{n\leq N}|\mathcal{E}_n|)$, and follow the same initial steps as in the proof of Lemma 5.2.3, then we can obtain

$$Z_n \leq C_p\delta\sum_{m=0}^{N-1}\mathscr{Z}_m + C_p\left(c_1\delta^{\frac{p}{2}} + c_2\delta^{\frac{p}{2}} + c_3\delta^{-\frac{p}{2}} + c_4\delta^{-\frac{p}{2}}\right), \quad (8.5.3)$$

where the coefficients c_1 and c_2 are near identical to those in on our analysis of $\mathbb{E}(\sup_{n\leq N}|\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \bar{X}_n^{\mathrm{f}} + \bar{X}_n^{\mathrm{c}}|^p)$, and namely c_1 and c_2 are both proportional to $\mathbb{E}(|Z - \bar{Z}|^{p(1+\epsilon)})^{1/(1+\epsilon)}$. The c_3 and c_4 coefficients arise from bounds for $\mathbb{E}(|\eta_n|^p)$ and $\mathbb{E}(|\eta_n'|^p)$, and are proportional to $\varrho^p(1 + \mathbb{E}(|\bar{X}_N^{\mathrm{f}}|^p))$, and as η_n and η_n' both correspond to $s = -\frac{1}{2}$ in Lemma 5.2.3, these give rise to the $\delta^{-p/2}$ terms. Thus we obtain

$$\leq C_p\delta\sum_{m=0}^{N-1}\mathscr{Z}_m + C_p\left(\mathbb{E}\left(\left|Z - \bar{Z}\right|^{p(1+\epsilon)}\right)^{\frac{1}{1+\epsilon}}\delta^{\frac{p}{2}} + \varrho^p\left(1 + \mathbb{E}\left(\left|\bar{X}_N^{\mathrm{f}}\right|^p\right)\right)\delta^{-\frac{p}{2}}\right). \quad (8.5.4)$$

By an application of Grönwall's discrete inequalities (Lemma A.1.9) the desired result immediately follows. **QED**

Inspecting Theorem 8.5.1, we can see that we anticipate two competing behaviours. The first is when we are simulating coarse (or medium resolution) paths, in which case the impact of rounding error is negligible, and we see the same convergence behaviour as from Theorem 6.2.6 dominating. However, as our path simulations become sufficiently fine, δ becomes very small, and the number of path increments N becomes very large, then the effects of rounding error will no longer be negligible. Surpassing some critical temporal resolution and decreasing δ any further, one can expect the variance of the multilevel Monte Carlo correction to increase at a rate $\mathcal{O}(N)$ as a result of the accumulated rounding error incurred from each update of the Euler-Maruyama scheme. The onset of the rounding error becoming significant will be sooner and at a coarser level the better the fidelity of the approximate random number distribution or similarly for lower precision calculations.

An interesting and novel feature exposed by Theorem 8.5.1, is that the Euler-Maruyama scheme is allowed to incur systematic rounding effects (η') in addition to the zero mean effects (η). Provided these systematic effects have a discretisation dependence $\mathcal{O}(\sqrt{\delta})$, and thus an overall size up to $\mathcal{O}(\varrho\sqrt{\delta})$, then their nett effect is the same order as the nett effect from the zero mean terms.

Overall then, Theorem 8.5.1 provides two core insights. The first is that for the effects of rounding error to be negligible, low-precisions can be used on coarse and medium resolution simulations, but that for increasingly finer path simulations the floating-point precision will eventually need to be increased. Additionally, Euler-Maruyama implementations which produce systematic rounding errors can be permissible provided the size of such an error is $\mathcal{O}(\varrho\sqrt{\delta})$ at each update, in which case their nett effect should not be distinguishable from the other non-systematic rounding errors that would otherwise have been accumulated.

Chapter 9

Implementing numeric schemes in finite-precision

In this chapter we provide numeric demonstrations showing the cumulative effect of floating-point arithmetic rounding error for the Euler-Maruyama scheme. We verify the key modelling assumptions we made which are specific to the Euler-Maruyama scheme, and present the resultant behaviour in our nested multilevel Monte Carlo framework. Furthermore, we showcase the Kahan compensated summation method as a means of delaying the onset of rounding error.

For these numeric experiments, implementations were in C and run on a Nvidia Jetson AGX Xavier machine. This comprises of a Nvidia 512-core Volta GPU and an 8-core Arm v8.2 64-bit CPU. Both the GPU and CPU support half-precision for both data storage and floating-point arithmetic processing (without intermediate typecasting). The half-precision data-type is supported on the CPU by `gcc` via the `_Float16` data-type (suffixed using `f16`) (cf. `arm_fp16.h` [192, 6.13, page 485]). As the CPU was capable of half-precision, it was sufficient to only run the experiments on the CPU. For convenience, we aliased the 16-bit half-precision data-type as `half` using a `typedef`.

For our simulations, we again take our underlying stochastic process from (5.3.1) to be a standard geometric Brownian motion setup, with the same parameters as we previously used and outlined in Section 5.3. As we will be using IEEE floating-point values, it is useful to recall some of their attributes, which are summarised in Table 9.0.1.

Name	Type	Size (bit)	Range	Mantissa (bit)	Precision
Double	`double`	64	$[10^{-323}, 10^{308}]$	52	$\varrho_{64} \approx 2^{-53} \approx 10^{-16}$
Single	`float`	32	$[10^{-45}, 10^{38}]$	23	$\varrho_{32} \approx 2^{-24} \approx 10^{-7}$
Half	`_Float16/half`	16	$[10^{-7}, 10^{5}]$	10	$\varrho_{16} \approx 2^{-11} \approx 10^{-3}$

Table 9.0.1 – Properties of IEEE floating-point formats.

In our simulations, unlike before, we now want to discourage any and all compiler optimisations, and keep the computations performed by the hardware as close to the original C code as possible. To achieve this, when compiled using `gcc`, we used the compiler flags `-O0`, `-Wall`, `-march=armv8.2-a+fp16`, `-lgsl`, `-lm`, and `-lgslcblas`, where notably `-O0` disables all compiler optimisations.

9.1 Introducing floating-point calculations

The first item to demonstrate is the accumulation of rounding error in the Euler-Maruyama scheme which results solely from the introduction of floating-point arithmetic. We know from Theorem 8.4.5 that we expect the strong error between the infinite-precision and finite-precision Euler-Maruyama estimates to be $\mathcal{O}(\varrho\sqrt{N})$, (regardless of whether we are using exact or approximate random variables). As we never have access to infinite-precision calculations, but only ever finite-precision calculations, we take 64-bit calculations as a proxy for infinite precision where applicable. For added clarity we use the subscript to denote the floating-point precision level. From Theorem 8.4.5 we expect

$$\mathbb{E}\Big(\Big|\hat{X}_{64} - \hat{X}_{32}\Big|\Big) = \mathcal{O}\Big(\varrho_{32}\sqrt{N}\Big), \tag{9.1.1}$$

and an equivalent expression for half-precision. The empirical behaviour for single and half precision is shown in Figure 9.1.1.

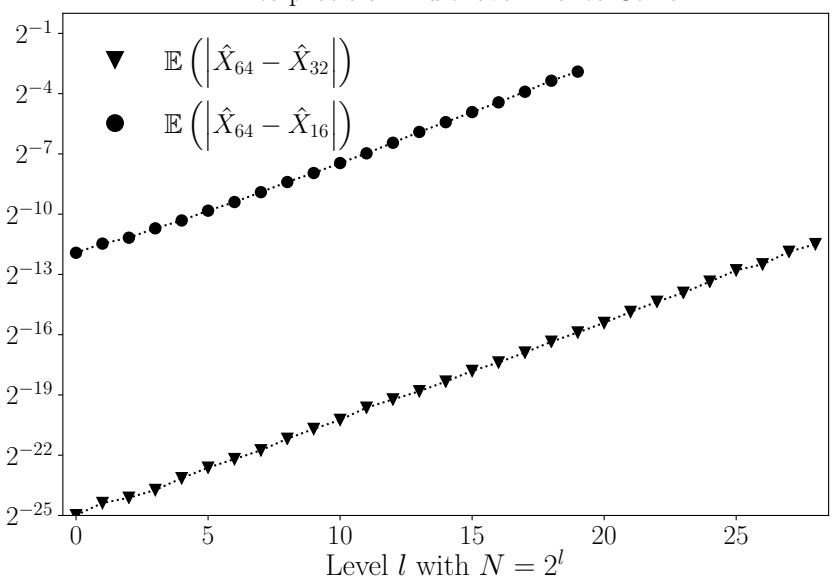


Figure 9.1.1 – The accumulation of rounding error using exact Euler-Maruyama simulations in varying precisions.

From Figure 9.1.1 we can make a few remarks. The first is that the expected $\mathcal{O}(\sqrt{N})$ behaviour can be readily seen for both single and half-precision. The next item, is that as we transition from single to double-precision, we should anticipate an increase of order $\frac{\varrho_{16}}{\varrho_{32}} \approx 2^{13}$, which again can be observed. Lastly, when $N = 1$ the error should be approximately ϱ, which given the values from Table 9.0.1 we can see is the case. Thus, all these features corroborate Theorem 8.4.5.

9.2 Rounding error in the nested multilevel Monte Carlo scheme

Our next major result to verify will be Theorem 8.5.1, for which if we use approximate random variables, then we expect to observe $\mathbb{V}(\hat{X}^{\mathrm{f}}_{64} - \hat{X}^{\mathrm{c}}_{64} - \tilde{X}^{\mathrm{f}}_{32} + \tilde{X}^{\mathrm{c}}_{32}) = \mathcal{O}(\delta) + \mathcal{O}(\varrho^2_{32}\delta^{-1})$ as we vary either the precision or the time increment, with a similar expression holding for half-precision. The result for various precisions and discretisations are shown in Figure 9.2.1.

There are several observations to be made about Figure 9.2.1, and the easiest starting point is to consider when we use approximate random variables in 32-bit single-precision. We

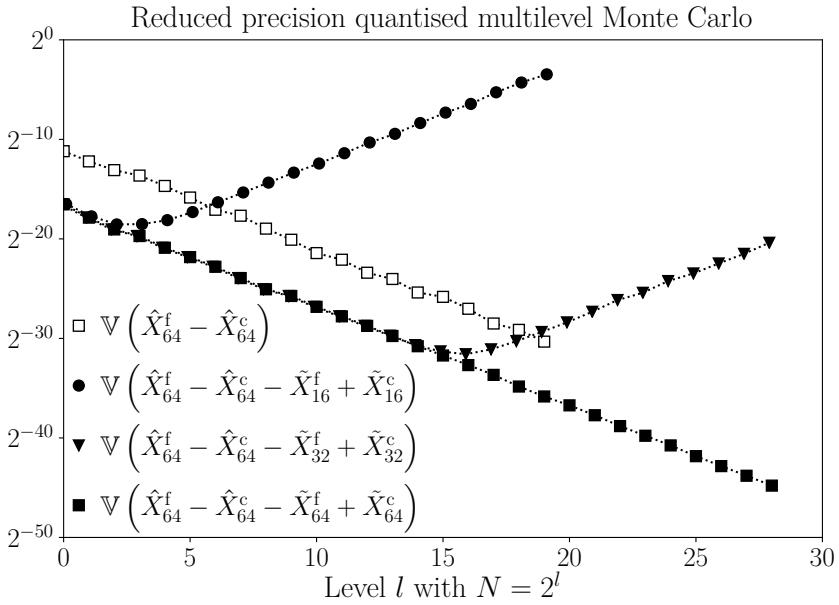


Figure 9.2.1 – The variance of the multilevel Monte Carlo correction when introducing approximate random variables and reduced-precision.

can see that as we start from the coarsest levels and increase the temporal resolution, that the variance is dominated by the leading order $\mathcal{O}(\delta)$ term. This demonstrates that for the coarsest levels, the most significant source of error is the discretisation, as expected. The variance continues to drop until around level $l = 15$, at which point the accumulated rounding error becomes significant, and then as the levels become increasingly finer the cumulative rounding error increases at the anticipated $\mathcal{O}(\delta^{-1})$ rate. The accumulated rounding error when exceeding level $l = 15$ quickly closes the gap between the variance of the two-way difference and the four-way difference. This indicates that once rounding error just starts to become significant, there is scope to increase the levels only once or twice more before the variance reduction from the four-way difference is lost. Furthermore, the discretisation corresponding to $l = 15$ is incredibly fine, and it is quite unlikely that any simulation will perform such fine path simulations. This goes some way to vindicating the position that single-precision is usually sufficient for most simulations.

For double-precision, the story is even more straight forward. The paths were never fine enough that the effect of rounding error were significant compared to the error from introducing an approximate random number distribution.

For half-precision, we can primarily see the effect of rounding error, and that it becomes the dominant and significant source of error for discretisations corresponding to $l \geq 3$. Thus, for any moderate or fine resolution path simulations, half-precision is too imprecise for any meaningful calculations. However, the converse to this is that half-precision is tolerable for the coarsest few levels. Importantly, we should not be discouraged by this result. An important consequence of the multilevel Monte Carlo analysis by Giles [76] is that under appropriate situations, where the variance has a temporal decay rate $\mathcal{O}(\delta)$ or $o(\delta)$, then the savings from the multilevel Monte Carlo framework are at their best. In these optimal $o(\delta)$ regimes, the majority of the computational work is performed at the coarsest levels. For our Euler-Maruyama scheme, we are in the intermediate $\mathcal{O}(\delta)$ regime where the work load is approximately evenly spread over the levels. However, an improved temporal dependence of order $o(\delta)$ can possibly be achieved from different numeric schemes, (e.g. the Milstein scheme). This would mean almost all the work will be on the coarsest levels, for which we would be able to benefit from using half-precision. (Incorporating approximate random variables with the Milstein scheme is explored later in Chapter 10).

For completeness, we should also mention that we expect the difference in the impact of rounding error to differ by $\frac{\varrho_{32}^2}{\varrho_{16}^2} \approx 2^{23}$, which again we see. From this relation, we can estimate the level when rounding error might become significant for double-precision, which we would expect to be $l \approx 45$. Furthermore, this simple relation can be used for predicting the onset of rounding errors for non-standard data-types. This could possibly predict the range of levels suitable for `bfloat16`, or possible user-defined precisions, such as those used by Omland et al. [167] on their FPGA applications. Interestingly, for `bfloat16` (with a mantissa of 7 bits), we might anticipate the onset of significant rounding error from level $l = 0$ onwards, and hence `bfloat16` might only be possible on the coarsest level, if even at all.

9.2.1 Kahan compensated summation

From the results in Figure 9.2.1, we have seen that the utility of half-precision is restricted to the coarsest few levels, corresponding to $l \leq 2$. Based on the multilevel Monte Carlo results from Giles [76] and our primary $\mathcal{O}(\delta)$ temporal rate of variance decay from Theorem 8.5.1 for the approximate Euler-Maruyama scheme, there is a clear and significant motivation to try and extend this domain of applicability, and delay the onset of significant rounding error.

The problem of summing a sequence of floating-point numbers and minimising the cumulative rounding error is well known, and there have been a variety of methods developed to overcome this, such as *pair-wise summation* or *compensated summation* [110, 4.1]. As the Euler-Maruyama scheme is a sequential and incremental procedure, a compensated summation is suitable, with *Kahan compensated summation* [129] being the least numerically intensive, and thus likely the most suitable for high-performance low-precision vectorised arithmetic. The Kahan compensated summation adds a given increment, and then subtracts away the computed summation prior to that increment. The difference of this inferred increment from the original provides an estimate for the incurred rounding error, which is then adjusted for when adding subsequent terms in the sequence. The Kahan compensated summation procedure is outlined in Algorithm 9.2.1. Interestingly, the related use of Kahan compensated summation in the numerical solution of ordinary differential equations was first proposed by Vitasek [204], and is demonstrated by Higham [109, pages 86–87].

Input: A sequence $\{x_1, x_2, \ldots, x_n\}$ of n floating point numbers.
Output: A high accuracy estimate of the summation $\sum_{i=1}^{n} x_i$.
1 Initialise both an accumulator a and compensation c to zero.
2 **for** $x_i \in \{x_1, x_2, \ldots, x_n\}$ **do**
3 Calculate a compensated increment $y \leftarrow x_i - c$.
4 Add the compensated increment $a_{\text{new}} \leftarrow a + y$ and keep the original $a_{\text{original}} \leftarrow a$.
5 Update the compensation $c \leftarrow (a_{\text{new}} - a_{\text{original}}) - y$.
6 Update the accumulator $a \leftarrow a_{\text{new}}$.
7 Use a to estimate $\sum_{i=1}^{n} x_i$.

Algorithm 9.2.1 – Kahan compensated summation.

An error analysis of Kahan compensated summation is provided by Higham [109, page 791, (3.11)], Knuth [135, Excercise 19, pages 229 and 571–573], and Goldberg [93], where Algorithm 9.2.1 has an overall absolute error of order

$$\Big(2\varrho + \mathcal{O}\Big(n\varrho^2\Big)\Big) \sum_{i=1}^{n} |x_i|, \tag{9.2.1}$$

and relative error

$$\Big(2\varrho + \mathcal{O}\Big(n\varrho^2\Big)\Big) \frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|}. \tag{9.2.2}$$

The ratio

$$\frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|} \tag{9.2.3}$$

is known as the condition number, representing the sensitivity of the summation to rounding error. For a series of zero mean random variables, the condition number can be expected to be $\mathcal{O}(\sqrt{n})$, but for our geometric Brownian motion, because of the drift term, the increments are not zero mean, and hence this is approximately a constant. Thus to leading order our error should have an $\mathcal{O}(1)$ dependence on n when using the Kahan compensated summation.

For the low-precision Euler-Maruyama scheme, the updates can be added using a Kahan compensated summation, where an implementation is demonstrated in Code 9.2.1. However, an aggressively optimising compiler may determine that the compensation should be zero, and optimise away the Kahan compensation. Avoiding this remains the responsibility of the programmer.

```
half euler_maruyama_with_kahan_compensation_16(half X, half dX, half * c)
{
    half y = dX - (*c);  // The compensated increment.
    half accumulated_sum = X + y;
    (*c) = (accumulated_sum - X) - y;
    return accumulated_sum;
}
```

Code 9.2.1 – The low-precision Euler-Maruyama scheme using Kahan compensated summation.

Incorporating in the Kahan compensated summation with our low-precision approximate Euler Maruyama scheme we can compute the variance of the multilevel Monte Carlo correction. Denoting paths which are constructed using the Kahan compensated summation with a subscript-"K", the results are shown in Figure 9.2.2. (The paths without Kahan summation are the same as they were in Figure 9.2.1, so require no discussion).

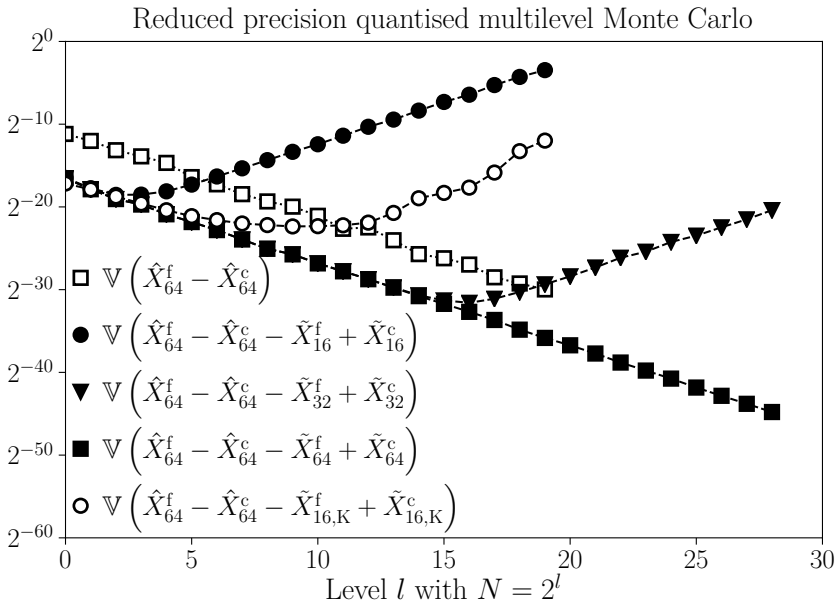


Figure 9.2.2 – The variance of the multilevel Monte Carlo correction when introducing approximate random variables and reduced precision.

Inspecting the half-precision simulations using Kahan compensated summation in Figure 9.2.1, we can notice that using the Kahan compensated summation achieves the desired effect. Namely, it delays the onset of significant rounding error, and extends the range of levels

suitable for half-precision to approximately $l \leq 5$, adding 3 more levels and approximately doubling the domain of applicability. Of course, our analysis has not factored in a Kahan compensated summation, and so this ultimately remains only a programming heuristic. Nonetheless, we speculate that it may provide a meaningful route to utilising low-precision simulations for intermediate path simulations. Furthermore, we notice the $\mathcal{O}(1)$ discretisation dependence for levels $l \leq 12$, as expected.

Chapter 10

Further applications

In this chapter we explore the applicability of the ideas we have developed thus far to wider families of applications related to those we have encountered, but which have not been covered in our analysis. Firstly, we will look to see if the approximate Gaussian random variables can be incorporated into the Milstein scheme (rather than the Euler-Maruyama scheme), and hence hopefully other SDE approximation schemes. Secondly, we will inspect the Cox-Ingersoll-Ross process, as the distribution is known to follow a non-central χ^2-distribution, and hence we will approximate the χ^2-distribution rather than the Gaussian distribution. This will demonstrate if approximations to distributions other than the Gaussian can be similarly utilised.

10.1 The Milstein scheme

There are alternative approximation schemes aside from the Euler-Maruyama. One of the most commonly used to approximate (5.1.1) is *the Milstein scheme* [133, 10.3, page 345, (3.1)]

$$\hat{X}_{n+1} = \hat{X}_n + a(\tau_n, \hat{X}_n)\delta + b(\tau_n, \hat{X}_n)\sqrt{\delta}\, Z_n + \frac{1}{2} b(\tau_n, \hat{X}_n)\frac{\partial b}{\partial X}(\tau_n, \hat{X}_n)\delta(Z_n^2 - 1), \qquad (10.1.1)$$

which can be viewed as a strong Taylor scheme [133, Chapter 10], which under modest differentiability assumptions has a strong convergence of order 1 [133, Theorem 10.6.3]. To incorporate approximate random variables, the natural extension to this is the *approximate Milstein scheme*

$$\tilde{X}_{n+1} = \tilde{X}_n + a(\tau_n, \tilde{X}_n)\delta + b(\tau_n, \tilde{X}_n)\sqrt{\delta}\, \tilde{Z}_n + \frac{1}{2} b(\tau_n, \tilde{X}_n)\frac{\partial b}{\partial X}(\tau_n, \tilde{X}_n)\delta(\tilde{Z}_n^2 - 1). \qquad (10.1.2)$$

Performing the same simulations as we did for Chapter 7, except substituting the approximate Euler-Maruyama scheme with the approximate Milstein scheme, we can again inspect the variances of the terms relevant to our multilevel Monte Carlo framework. The first is the convergence behaviour for the underlying process, shown in Figure 10.1.1.

Inspecting Figure 10.1.1 we can make several observations. Our first is that the variance $\mathbb{V}(\hat{X}_l - \hat{X}_{l-1})$ between the fine and the coarse levels for the exact random variables has a decay rate $\mathcal{O}(2^{-4l})$. Recalling that four fine increments sum to a coarse increment, this implies that the scheme has a strong convergence order of 1, which is a standard and well known result that is to be expected [133, Theorem 10.6.3]. Furthermore, the new multilevel Monte Carlo estimator using the approximate random variables converges to the same result. This is identical to our previous result when inspecting the Euler-Maruyama scheme.

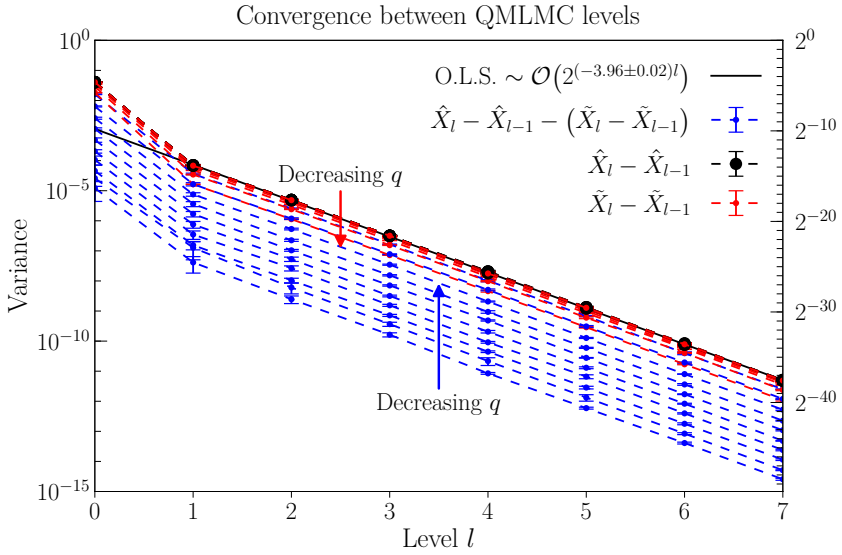


Figure 10.1.1 – The variance structure for a nested QMLMC scheme when using the approximate Milstein scheme. The approximate random variables are produced using the equipartitioned piecewise constant approximation with 2^q intervals.

The more significant result we can see is that there is a uniform convergence behaviour, independent of the level, for the multilevel correction term. We can see that this exhibits the same rate of strong convergence, namely $\mathcal{O}(\delta)$. Furthermore, the decrease in variance from incrementing q by 1 is approximately a factor of 2^{-1}. We recall that all these features were seen previously from the Euler-Maruyama scheme. This leads us to speculate that the corresponding adaptation of Theorem 6.2.6 for the approximate Milstein scheme would be $\mathbb{E}(\sup_{n\leq N}|\hat{X}_n^{\mathrm{f}} - \hat{X}_n^{\mathrm{c}} - \tilde{X}_n^{\mathrm{f}} + \tilde{X}_n^{\mathrm{c}}|^p) \leq C\delta_{\mathrm{c}}^p\mathbb{E}(|Z-\tilde{Z}|^{p(1+\epsilon)})^{1/(1+\epsilon)}$, where the exponent on the time step has been increased to match the strong convergence order of the usual Milstein scheme.

Performing the same simulations, but evaluating the call option payoff function, we obtain the results shown in Figure 10.1.2. Unlike our equivalent results for the Euler-Maruyama scheme for the non-differentiable call option payoff function, shown in Figure 7.2.5, we are unable to detect any transition in the convergence rate as we move from high quantisations and coarse time steps to low quantisations and fine time steps. There are two plausible explanations for this. The first is that we have not gone to random variable approximations of high enough fidelity. The second could be that the Milstein scheme is more robust to the use of approximate random variables than the Euler-Maruyama scheme and does not exhibit the same behaviour for this class of functions. Perhaps the effects we anticipated can be demonstrated for higher fidelity approximate random variables or for more difficult payoff functions, but there appears to be no evidence for this based on these preliminary experimental results.

10.2 The Cox-Ingersoll-Ross process

The Cox-Ingersoll-Ross (CIR) process [51] is a single-factor model commonly used to describe the dynamics of instantaneous interest rates, volatility processes in Heston models [106, page 329, (4)] for stochastic volatility modelling, and is generally common in fixed income modelling [161, 7.5, pages 214–220]. The CIR process [51, page 391, (17)] [161, 3.8.3, page 67, (3.16)] can be written as

$$\mathrm{d}X_t = \kappa(\theta - X_t)\,\mathrm{d}t + \sigma\sqrt{X_t}\,\mathrm{d}W_t\,, \tag{10.2.1}$$

where κ, θ, and σ are all strictly positive constants. This is understood to be a mean reverting model, and because of the $\sqrt{X_t}$ scaling for the volatility term, the process is non-negative. The

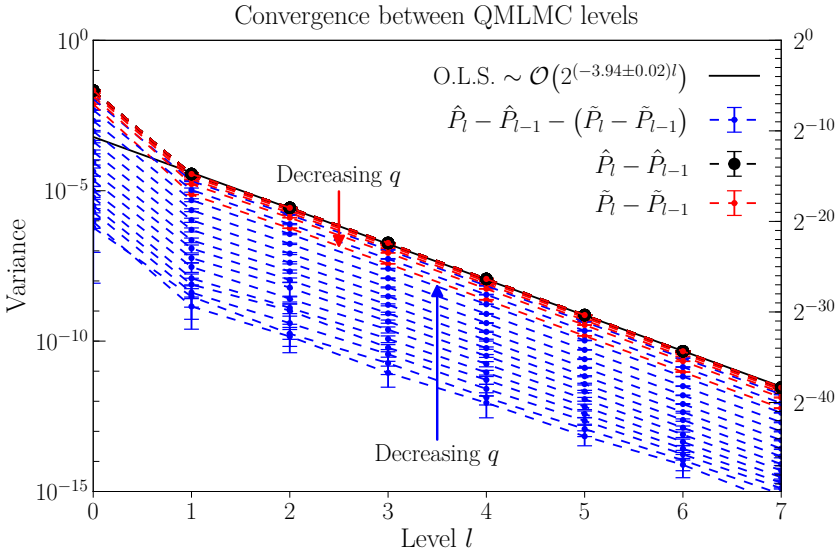


Figure 10.1.2 – The variance structure for a nested QMLMC scheme when using the approximate Milstein scheme. The approximate random variables are produced using the equipartitioned piecewise constant approximation. We show the variance results for a call option payoff function.

process is also strictly positive if $2\kappa\theta \geq \sigma^2$, which is known as the Feller condition [68].

If we are tasked with simulating this process and building a path approximation, we may turn to the usual Euler-Maruyama scheme. However, the process is known to be non-negative, and the Euler-Maruyama scheme does not guarantee this, and will fail when trying to evaluate the $\sqrt{X_t}$ term. There have been several attempts to remedy this shortcoming, which include the truncated scheme by Deelstra and Delbaen [59]

$$\hat{X}_{n+1} = \hat{X}_n + \kappa(\theta - \hat{X}_n)\delta + \sigma\sqrt{\hat{X}_n^+}\sqrt{\delta}Z_n, \tag{10.2.2}$$

the fully truncated version by Lord et al. [146, Theorem 4.2 and A.4]

$$\hat{X}_{n+1} = \hat{X}_n + \kappa(\theta - \hat{X}_n^+)\delta + \sigma\sqrt{\hat{X}_n^+}\sqrt{\delta}Z_n, \tag{10.2.3}$$

the reflected scheme by Berkaoui et al. [20, 2.1, (2)]

$$\hat{X}_{n+1} = \left|\hat{X}_n + \kappa(\theta - \hat{X}_n)\delta + \sigma\sqrt{\hat{X}_n}\sqrt{\delta}Z_n\right|, \tag{10.2.4}$$

or the related scheme by Higham et al. [108]

$$\hat{X}_{n+1} = \hat{X}_n + \kappa(\theta - \hat{X}_n)\delta + \sigma\sqrt{\left|\hat{X}_n\right|}\sqrt{\delta}Z_n. \tag{10.2.5}$$

Several other implicit and higher order schemes are studied by Alfonsi [6, 7, 8], and for a fuller discussion of Euler-Maruyama type methods we recommend Dereich et al. [60], Lord et al. [146], and Cozma and Reisinger [52].

As the volatility term is non-Lipschitz, our usual strong convergence analysis is not immediately applicable. The extension of the strong analysis to cover the volatility term being $\frac{1}{2}$-Hölder continuous is performed by Gyöngy and Rásonyi [95], and they obtain a strong error bound $\mathcal{O}(-\log^{-1}(\delta))$ for (10.2.2) and (10.2.5). However, if the Feller condition is satisfied, then Gyöngy [94] showed a strong convergence order of $\frac{1}{2}$ for (10.2.2), (10.2.4), and (10.2.5), as measured using the L^1-norm. Furthermore, for the reflected Euler-Maruyama scheme (10.2.4) from Berkaoui et al. [20, 2.1, (2)], to ensure that a strong convergence order of $\frac{1}{2}$ can be expected

when measured using an arbitrary L^{2p}-norm requires the assumption [60, page 1108] [20, page 3, Theorem 2.2, (10)] [146, page 185, (26)]

$$\frac{2\kappa\theta}{\sigma^2} > 1 + \sqrt{8}\max\left\{\sqrt{\frac{\kappa(16p-1)}{\sigma^2}}, 16p-2\right\}. \tag{10.2.6}$$

As remarked by Lord et al. [146, page 185]: "*One can easily check that, unfortunately, this condition [(10.2.6)] is hardly ever satisfied for any practical values of the parameters*". Inspecting this value for $p = 1$ we notice that this has the lower bound $\frac{2\kappa\theta}{\sigma^2} > 41$, which is so much higher and more restrictive than the Feller condition that by the central limit theorem the non-central χ^2-distribution will be very well approximated by a Gaussian in this regime [29, page 118].

10.2.1 The non-central χ^2-distribution

One of the appealing features of the CIR model is that, whilst an explicit solution to (10.2.1) in terms of the Weiner process W_t is unknown, the terminal distribution of X_T conditioned on X_t is known. The distribution is a non-central χ^2-distribution [51, page 392, (19)] [161, page 67]. Denoting the non-central χ^2-distribution as $\chi^2_\nu(\lambda)$, where ν denotes the *degrees of freedom* and λ the *non-centrality parameter*, then we have $\nu = \frac{4\kappa\theta}{\sigma^2}$ and $\lambda = \frac{4\kappa X_t}{\sigma^2}(\frac{\exp(-\kappa(T-t))}{1-\exp(-\kappa(T-t))})$.

The benefit of knowing the exact distribution is that it is possible to produce discretised path simulations (necessary for pricing exotic derivatives), where the increments exactly follow the correct distribution. This ensures that the Monte Carlo estimator is unbiased, and circumvents the difficulties associated with the Euler-Maruyama scheme maintaining positivity. The issue of bias for the Euler-Maruyama scheme can be quite considerable, and is explored by Broadie and Kaya [33], who demonstrate this can still be problematic for simulations using up to 3200 time steps and 10^7 path simulations [33, Section 4, page 222].

Unfortunately, simulating from the exact distribution is not without its drawbacks. Namely, simulating from the non-central χ^2-distribution is very expensive compared to the Gaussian distribution required for the Euler-Maruyama scheme. (This is discussed later and shown in Table 10.2.2).

In keeping with the intuition we have so far developed surrounding approximate random variables, we can see that the CIR process presents a natural candidate for their incorporation. The Euler-Maruyama scheme implies and requires that the non-central χ^2 random variable is approximated by a Gaussian distribution in the limit as the time increment δ vanishes to zero. Thus there are two natural points where we could introduce approximate random variables. The first would be to use the Euler-Maruyama scheme and approximate the Gaussian random variables with approximate random variables, as we have already done so far. The second would be to directly construct an approximate distribution $\tilde{\chi}^2_\nu(\lambda)$ of the non-central χ^2-distribution $\chi^2_\nu(\lambda)$. It is the latter of these two we are now particularly interested in investigating here. This is of particular and novel interest because this distribution is parametrised, and thus any approximation we will construct will too be parametrised, adding to any approximation's complexity. Various proposals for simulating the CIR process are shown in Table 10.2.1, where our particular focus of concern will be assessing the variance structure between what we call the approximate CIR process, and the Euler-Maruyama approximation.

Identical to before, we will construct our approximate distribution $\tilde{\chi}^2_\nu(\lambda)$ by approximating the inverse cumulative distribution function for the non-central χ^2-distribution. This will enable us to tightly couple the approximate distribution $\tilde{\chi}^2_\nu(\lambda)$ to the exact distribution $\chi^2_\nu(\lambda)$ by evaluating both inverse cumulative distribution functions for the same percentile $U \sim \mathcal{U}([0,1))$. We can notice that this is how we coupled the approximate and exact

Method	Estimator	Distribution
Exact CIR	$\mathbb{E}(\hat{P}(\hat{X}_T))$	$\hat{X}_T \sim \chi^2_\nu(\lambda)$
Approximate CIR	$\mathbb{E}(\hat{P}(\tilde{X}_T)) + \mathbb{E}(\hat{P}(\hat{X}_T) - \hat{P}(\tilde{X}_T))$	$\tilde{X}_T \sim \tilde{\chi}^2_\nu(\lambda)$
Euler-Maruyama	$\mathbb{E}(\hat{P}(\tilde{X}_T)) + \mathbb{E}(\hat{P}(\hat{X}_T) - \hat{P}(\tilde{X}_T))$	$\Delta\tilde{X}_t \sim \mathcal{N}$
Approximate Euler-Maruyama	$\mathbb{E}(\hat{P}(\tilde{X}_T)) + \mathbb{E}(\hat{P}(\hat{X}_T) - \hat{P}(\tilde{X}_T))$	$\Delta\tilde{X}_t \sim \tilde{\mathcal{N}}$

Table 10.2.1 – Simulation schemes for the CIR process, and some possible extensions which utilise approximate random variables.

Gaussian random variables for the Euler-Maruyama scheme. Furthermore, it is through sharing the same underlying percentile in each increment that we will be able to couple the exact CIR simulation to the Euler-Maruyama simulation. Recalling there is no explicit relation between the non-central χ^2-distribution and the underlying Weiner process, we rely on coupling the two through the percentile in this way.

For those looking to approximate the non-central χ^2-distribution and produce approximate random variables, then numerous approximations are possible [1, 22, 114, 124, 180, 186, 190, 214]. Additionally, it is possible to construct approximate random variables from Gaussian random variables, as shown by Abdel-Aty [1]. However, such schemes do not proceed by the inverse transform method, and hence do not give rise to any coupling mechanism appropriate for our multilevel Monte Carlo framework.

10.2.2 Approximating the non-central χ^2-distribution's inverse cumulative distribution function

As the time increment becomes vanishingly small, the distribution from the exact CIR process will converge to the distribution arising from the Euler-Maruyama scheme, and hence the non-central χ^2-distribution converges to the Gaussian distribution as $\delta \to 0$.

Letting $C_\nu^{-1}(\cdot;\lambda)$ denote the inverse cumulative distribution function of the non-central $\chi^2_\nu(\lambda)$-distribution, then the exact CIR simulation is

$$\hat{X}_{n+1} = \frac{\sigma^2(1-\exp(-\kappa\delta))}{4\kappa} C^{-1}_{\frac{4\kappa\theta}{\sigma^2}}\left(U_n; \hat{X}_n \frac{4\kappa\exp(-\kappa\delta)}{\sigma^2(1-\exp(-\kappa\delta))}\right), \tag{10.2.7}$$

where $U_n \sim \mathcal{U}([0,1))$. Furthermore, noticing that we can non-dimensionalise this by introducing the process $V_t \coloneqq \frac{X_t}{\theta}$, then we obtain $\hat{V}_{n+1} = \frac{f}{d} C_d^{-1}(U_n; \hat{V}_n \frac{(1-f)d}{f})$, where $\beta^2 \coloneqq \frac{\sigma^2}{\theta}$, $d \coloneqq \frac{4\kappa}{\beta^2}$, and $f \coloneqq 1-\exp(-\kappa\delta)$. Defining the function $Q_x(\cdot;y)$ through $Q_x(U;y) \coloneqq \frac{y}{x} C_x^{-1}(U; \frac{(1-y)x}{y})$, and the variable f' by $\frac{1-f'}{f'} \coloneqq \frac{(1-f)\hat{V}_n}{f}$ (and hence $f' \equiv \frac{f}{f+(1-f)\hat{V}_n}$), then we obtain the path simulation procedure

$$\hat{V}_{n+1} = \frac{f}{f'} Q_d(U_n; f'). \tag{10.2.8}$$

Notice that for a given CIR model that d is a strictly positive constant, which is constant for all discretisation levels. The only item which changes during a path simulation is f' as this has a dependence on $\hat{V}_n$. Notably, as $\delta \to 0$ (corresponding to $f' \to 0$), the update determined by $Q_d(U;f')$ converges to the Euler-Maruyama scheme's Gaussian increment, and we have $Q_d(U;f') \xrightarrow{\delta\to 0} 1 + \sqrt{\frac{4f'}{d}}\Phi^{-1}(U)$. We show $Q_d(\cdot;f')$ for several values of $f' \in (0,1)$ in Figure 10.2.1. If we extend this and define $P_d(U;f') \coloneqq \sqrt{\frac{d}{4f'}}(Q_d(U;f') - 1)$, and hence $Q_d(U;f') \equiv 1 + \sqrt{\frac{4f'}{d}} P_d(U;f')$, then we anticipate $P_d(U;f') \xrightarrow{\delta\to 0} \Phi^{-1}(U)$. This convergence is demonstrated in Figure 10.2.2, and the rate of convergence is shown in Figure 10.2.3. Inspecting Figure 10.2.3, this suggests that we may require very small time steps to reduce the bias of the

Euler-Maruyama scheme to a suitably small level for achieving a desirable MSE, a result which is consistent with the observations from Broadie and Kaya [33].

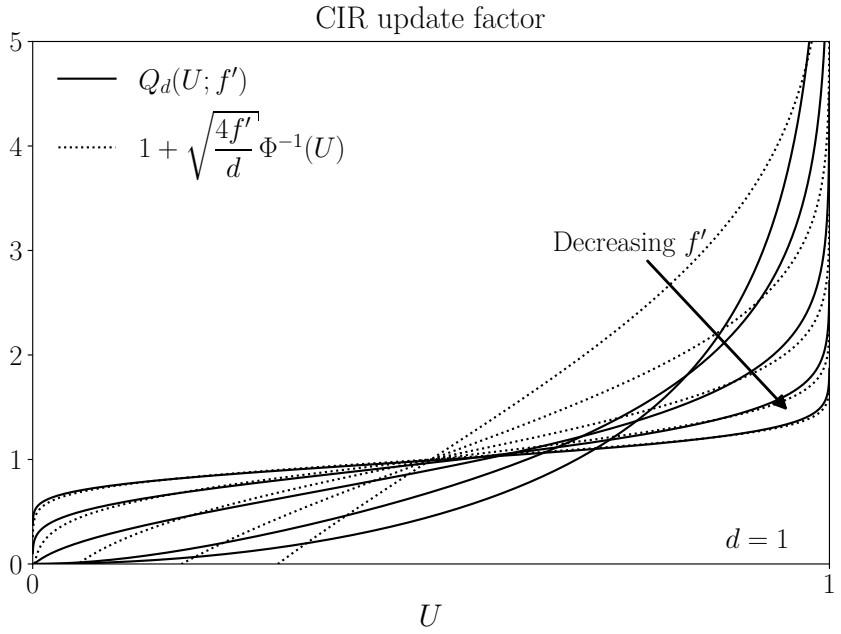


Figure 10.2.1 – The update to the CIR process given by (10.2.8) for $f' \in \{0.99, 0.32, 0.1, 0.03, 0.01\}$.

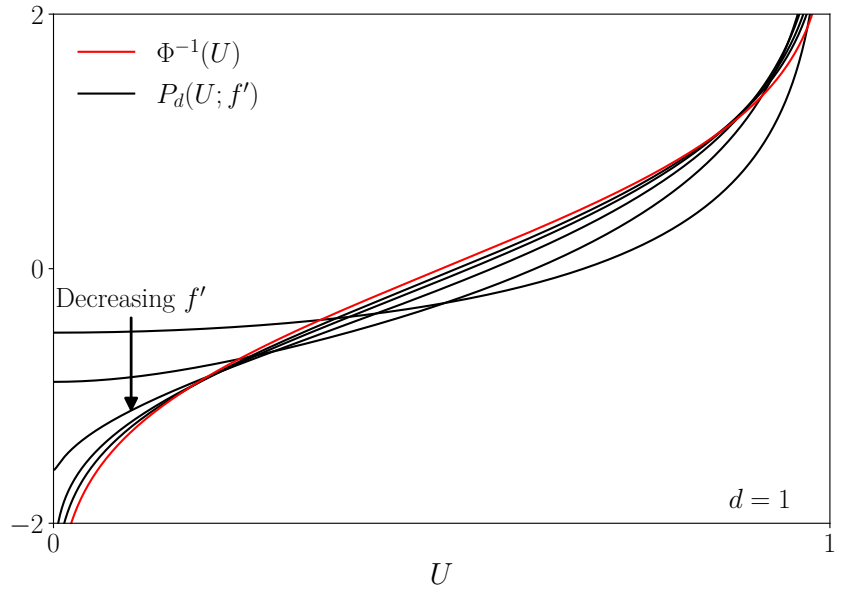


Figure 10.2.2 – The convergence of the CIR update to a Gaussian random variable, where the rate of decay is shown in Figure 10.2.3.

The natural adaptation of this is to incorporate approximate non-central χ^2 random variables, and to approximate $Q_d(\cdot; f')$ by some $\tilde{Q}_d(\cdot; f')$ where $\tilde{Q}_d(\cdot; f') \approx Q_d(\cdot; f')$. We notice that $f \xrightarrow{\delta \to 0} \kappa\delta$ and $f' \xrightarrow{\delta \to 0} \frac{f}{\hat{V}_n}$, and hence if we assume κ, β, and $\hat{V}_n$ are all $\mathcal{O}(1)$, then our approximation of $\tilde{Q}_d(\cdot; f')$, which is parametrised by f', will have its parametrisation vary approximately by the same degree that the discretisation's time step δ varies. Hence if $\delta = 4^{-l}$, then it is sensible to construct approximations for a discrete set of different and equally spaced values of $g(f')$ where $g(f')$ might feasibly be either $\log(f')$ or $\sqrt{f'}$, where the value for a specific value of f' is obtained from linear interpolation in the value of $g(f')$. For our approximations we choose to implement $g(f') \equiv \sqrt{f'}$ for convenience.

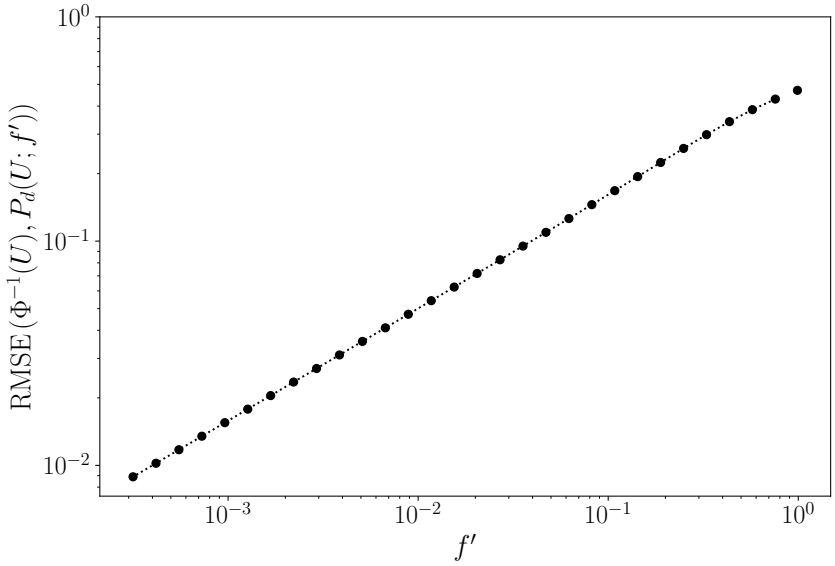


Figure 10.2.3 – The RMSE of the CIR update to a Gaussian update.

We form our approximation for $\tilde{Q}_d(\cdot; f')$ through a corresponding approximation $\tilde{P}_d(\cdot; f')$ to $P_d(\cdot; f')$, where we choose to directly approximate $P_d(\cdot; f')$ rather than $Q_d(\cdot; f')$ because we can see by a comparison between Figures 10.2.1 and 10.2.2 that $P_d(\cdot; f')$ is better scaled than $Q_d(\cdot; f')$, especially in the limit as $f' \to 0$. For each reference function which we need to approximate (recalling we have several reference functions for the different values of $g(f')$ which we will interpolate between), we perform a piecewise linear approximation over dyadic intervals, in almost exactly the same manner as we did for $\Phi^{-1}(\cdot)$. The only difference is that $P_d(\cdot; f')$ is not rotationally symmetric, and hence we construct an approximation in the interval $[0, \frac{1}{2})$, and another in $[\frac{1}{2}, 1]$. Choosing 16 dyadic intervals in each of the two halves of the domain, and 10 interpolation points for different values of f', we obtain the approximate CIR updating functions shown in Figure 10.2.4 (cf. Figure 10.2.1)

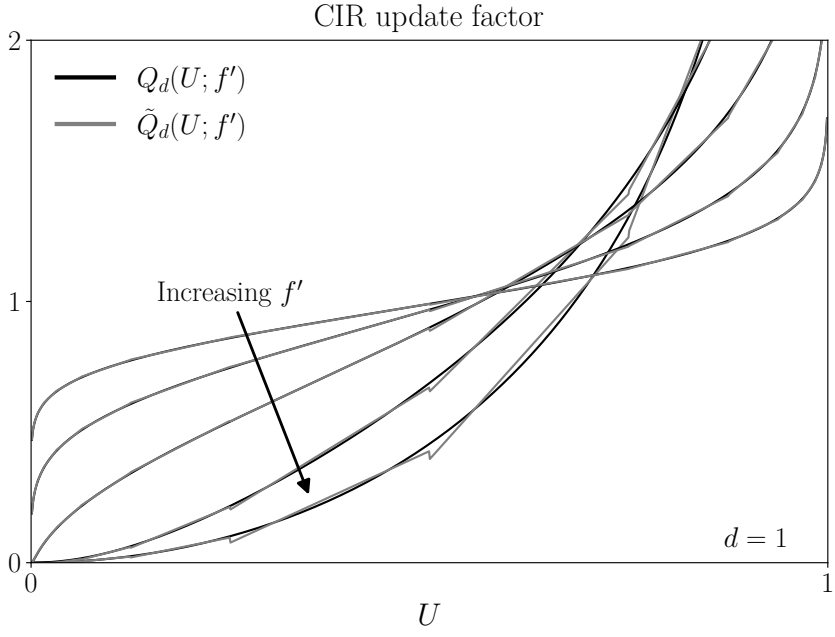


Figure 10.2.4 – The approximate update to the CIR process by an interpolated piecewise linear approximation over dyadic intervals for $f' \in \{0.99, 0.32, 0.1, 0.03, 0.01\}$. (cf. Figure 10.2.1).

10.2.3 Numeric simulations

For running our numeric simulations we will be simulating the dimensionless variance process V_t mentioned earlier, which follows $\mathrm{d}V_t = \kappa(1 - V_t)\,\mathrm{d}t + \beta\sqrt{V_t}\,\mathrm{d}W_t$ (similar to (10.2.1)). For this process, we will assume the Feller condition is satisfied, corresponding to $d \geq 2$. The simulation produced by the CIR model using an exact non-central χ^2-distribution is denoted $\hat{V}^{\chi^2}$ and follows (cf. (10.2.8))

$$\hat{V}_{n+1}^{\chi^2} = \frac{f}{f'} Q_d(U_n; f'), \tag{10.2.9}$$

and that using an approximate non-central χ^2-distribution is denoted $\tilde{V}^{\tilde{\chi}^2}$ and follows

$$\tilde{V}_{n+1}^{\tilde{\chi}^2} = \frac{f}{f'} \tilde{Q}_d(U_n; f'). \tag{10.2.10}$$

The two schemes are coupled by using the same percentile random variables U_n in both simulations.

If we wish to compare these simulations to those obtained from using an Euler-Maruyama scheme, then as previously discussed we need to settle on which variant of the Euler-Maruyama scheme we will be using to ensure the process remains well defined. For this purpose we choose the truncated scheme by Higham et al. [108] in (10.2.5) as this appears to most closely resemble the usual Euler-Maruyama scheme. Hence for the variance process we will produce an approximation $\hat{V}^{\mathcal{N}}$ which uses exact Gaussian random variables, and is constructed using (cf. (10.2.2))

$$\hat{V}_{n+1}^{\mathcal{N}} = \hat{V}_n^{\mathcal{N}} + \kappa\Big(1 - \hat{V}_n^{\mathcal{N}}\Big)\delta + \beta\sqrt{\Big|\hat{V}_n^{\mathcal{N}}\Big|}\,\sqrt{\delta}\,Z_n. \tag{10.2.11}$$

We couple this to the exact CIR simulations by producing the Gaussian random variables using $Z_n \coloneqq \Phi^{-1}(U_n)$, where the uniform random variable U_n is the same as that being used in the exact CIR simulation. Lastly, we can produce an approximate Euler-Maruyama estimate $\tilde{V}^{\tilde{\mathcal{N}}}$ using approximate Gaussian random variables by

$$\tilde{V}_{n+1}^{\tilde{\mathcal{N}}} = \tilde{V}_n^{\tilde{\mathcal{N}}} + \kappa\Big(1 - \tilde{V}_n^{\tilde{\mathcal{N}}}\Big)\delta + \beta\sqrt{\Big|\tilde{V}_n^{\tilde{\mathcal{N}}}\Big|}\,\sqrt{\delta}\,\tilde{Z}_n, \tag{10.2.12}$$

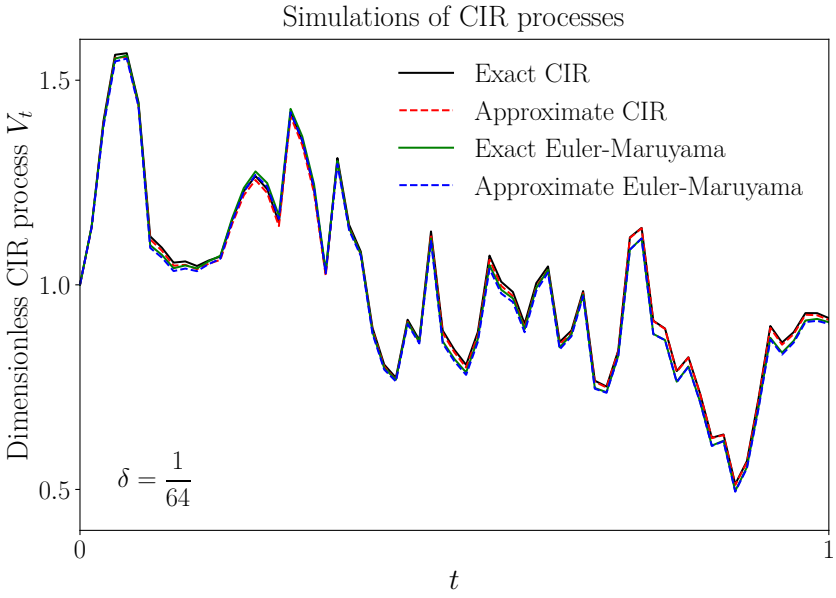
where $\tilde{Z}_n \coloneqq \tilde{\Phi}^{-1}(U_n)$.

Normalising our simulations over the interval time $[0, 1]$ such that κ and β are dimensionless quantities, we choose our parameters such that the Feller condition is just satisfied, and hence fix $d = 2$ and set $\kappa = \frac{\beta^2 d}{4}$. We then choose $\beta = 1$, so that most of the calculated quantities should remain $\mathcal{O}(1)$ during any simulation, and similarly we start our simulations with the initial condition $V_0 = 1$.

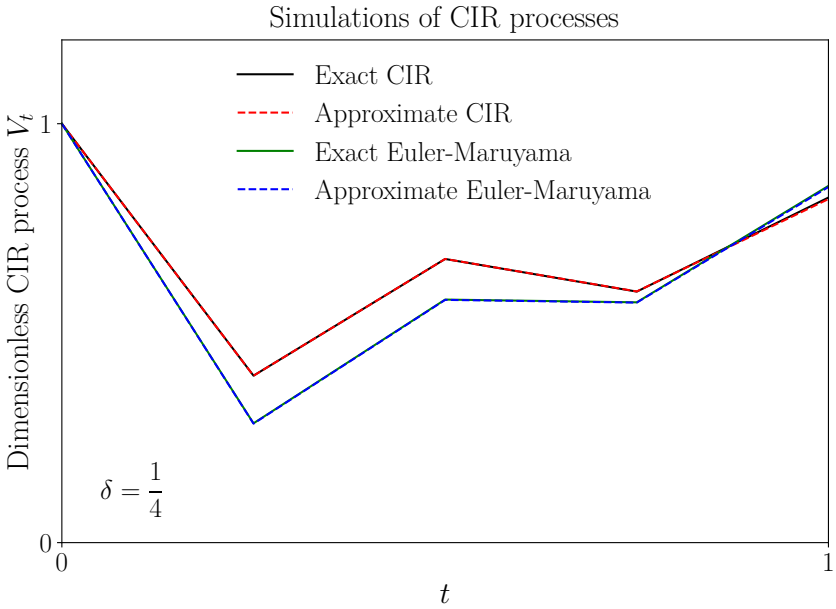
Proceeding with this configuration we can first confirm that all four schemes are closely coupled by inspecting Figure 10.2.5. We can first note that the coupling between the exact CIR paths and the Euler-Maruyama paths become more strongly coupled as the path simulations become finer, as can been seen by comparing their typical separation in Figures 10.2.5a and 10.2.5b. Furthermore, we can see that the paths produced using the exact and approximate random variables are extremely tightly coupled, resulting from the very high fidelity of the piecewise linear dyadic approximations for both these distributions.

Motivated by these results, we can inspect the behaviour of the possible multilevel Monte Carlo corrections for either switching to approximate non-central χ^2 random variables or to the Euler-Maruyama scheme. The results for couplings and behaviours of the multilevel terms are shown in Figure 10.2.6.

Inspecting Figure 10.2.6, there are a few observations we can make. The first is that the expected value obtained from the exact CIR process is approximately equal to that obtained



(a) Fine path simulation.



(b) Coarse path simulation.

Figure 10.2.5 – Simulations of CIR processes using fine and coarse path simulations by either utilising exact/approximate non-central χ^2 random variables or exact/approximate Gaussian random variables with the Euler-Maruyama scheme.

from the Euler-Maruyama scheme, and that this remains so after switching to approximate random variables from both distributions, and this appears the same for all discretisations. It seems largely coincidental that the Euler-Maruyama estimates match those from the exact CIR process for all discretisation levels, but this is largely to be expected and is not one of the particularly important features of Figure 10.2.6.

We can next remark that the multilevel correction term from switching to the Euler-Maruyama scheme decays as the simulations become finer with a strong convergence rate $\mathcal{O}(\delta^{1/2})$. This is as we expected, and it is unsurprising given the high fidelity of our approximate Gaussian random variables that the same decay rate is observed for the approximate Euler-Maruyama scheme.

The multilevel difference when switching from exact to approximate non-central χ^2

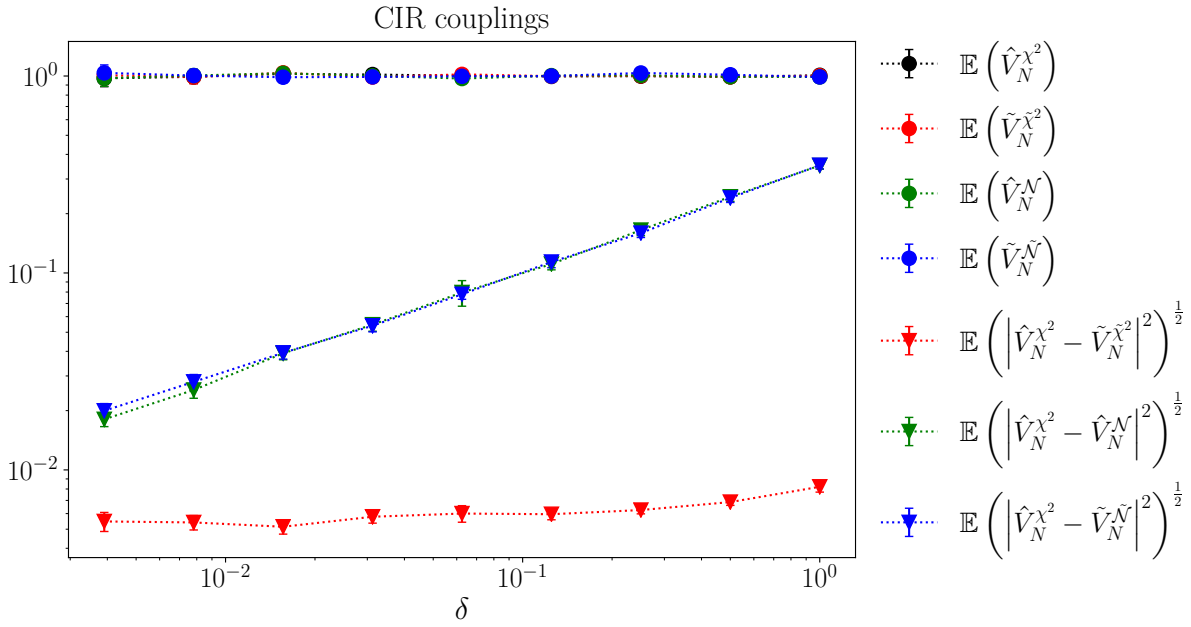


Figure 10.2.6 – The MLMC coupling and variance structure for a variable time step δ for the CIR process either utilising exact/approximate non-central χ^2 random variables or exact/approximate Gaussian random variables with the Euler-Maruyama scheme.

random variables has two notable features. The first is its largely unchanging value, and the second is its considerable variance reduction. That the value is largely unchanging is an exact parallel of the result from Theorem 5.2.4 that we found in our analysis of approximate Gaussian random variables within the Euler-Maruyama scheme. Hence seeing this similar behaviour here is encouraging. Similarly, the considerable drop in variance can be accredited to the high fidelity of the piecewise linear approximation on dyadic intervals.

Lastly, the reduction in variance from approximating the non-central χ^2-distribution is considerably more than from switching to the Euler-Maruyama scheme, with the two likely becoming comparable for $\delta \approx 10^{-4}$. The significance of this remark is that if the cost of producing approximate non-central χ^2 random variables can be brought down to times comparable to producing exact Gaussian random variables, then it would be computationally advantageous to incorporate approximate non-central χ^2 random variables rather than using the Euler-Maruyama scheme, even for very fine path simulations.

While we have seen that approximate Gaussian random variables are much cheaper to produce than exact Gaussian random variables, anticipating the speed of producing approximate non-central χ^2 random variables is more speculative. There are two factors contributing to this uncertainty, and the first is that because the inverse cumulative distribution function is no longer rotationally symmetric, we require twice as many coefficients to approximate the non-central χ^2-distribution for a certain non-centrality parameter, as we have different approximations in $[0, \frac{1}{2})$ and $[\frac{1}{2}, 1]$. The second contributing factor is that as the distribution is parametrised we need several approximations for different parametrisation values which are then interpolated. Both of these factors lead to a considerable increase in the number of coefficients (and hence vector registers) that would be required, and an increase in the approximation's complexity. For the approximation parameters that we used in our simulations, neither of these factors seem insurmountable, but implementing (and profiling) a high performance approximation for vector hardware remains an item for further research. Nonetheless, it is reasonable to suppose that there will not be enough vector registers to simultaneously hold all the necessary coefficients, but that these will still comfortably all fit within the L_1-cache. Thus, similar to our previous lookup table, we expect such an approximation to still be considerably fast, and likely taking of order a few clock cycles. Thus we anticipate such a scheme to be slightly better in cost to

sampling Gaussian random variables, or at least comparable.

10.2.3.1 Standard language implementations

To give an insight into how expensive it is to evaluate the inverse non-central χ^2 cumulative distribution function, we can take the associated function provided by Python's SciPy package (`ncx2.ppf` from `scipy.stats`). This calls the C routine `CDFCHN` from the `CDFLIB` library from Brown et al. [35] (implementations by Burkardt [38]). This computes the value by a root finding search routine [41, Section 4, page 337, Algorithm R] performed on the offset non-central χ^2 cumulative distribution function. This requires several evaluations of the cumulative distribution function, which for a non-central χ^2-distribution with ν degrees of freedom and non-centrality parameter λ we denote by $C_{\nu,\lambda}(\cdot)$. This is evaluated using the series expression $C_{\nu,\lambda}(\chi^2) = \sum_{j=0}^{\infty} \frac{1}{j!}(\frac{\lambda}{2})^j \exp(-\frac{\lambda}{2}) C_\nu(\chi^2)$ from Abramowitz and Stegun [4, page 942, (26.4.25)], where $C_\nu(\cdot)$ is the cumulative distribution function for a regular (central) χ^2-distribution.

We can see from this that as the non-centrality parameter increases this series requires more terms to converge to an accurate answer, resulting in a work load proportional to λ, as remarked by Burkardt [38, `cdflib.c`, `cdfchn`, lines 2734–2736]: "*Very large values of this parameter can consume immense computer resources*". The equivalent MATLAB function `ncx2inv` (from the "statistics and machine learning toolbox") is described as using "*Newton's method to converge to the solution*" [152, Chapter 32, page 4301], and hence appears to follow a similar approach to that used by Python's SciPy.

To assess how computationally intensive this routine is, we can measure the average time to produce 10^{3-5} non-central χ^2 random variable samples against Gaussian random variables. The ratio of these times is shown in Table 10.2.2 (comparing Python to MATLAB in Tables 10.2.2a and 10.2.2b respectively), where on a personal computer running at 2.5 GHz the average time per Gaussian sample was approximately 10^{-7} s for Python (and approximately the same for MATLAB's `norminv`), corresponding to approximately 250 clock-cycles per sample. Given this experiment was running in Python using unoptimised libraries, this is in keeping with our earlier times as measured for Cephes.

λ \ ν	1	5	10	50	100
1	37	36	40	54	73
5	40	46	48	62	85
10	54	56	63	69	97
50	101	103	103	144	143
100	191	190	192	189	185
200	243	246	240	233	221
500	465	474	465	446	416
1000	459	458	455	471	474

(a) Using Python and `ncx2.ppf` from Scipy's `scipy.stats`.

λ \ ν	1	5	10	50	100
1	168	214	259	456	294
5	651	782	840	1510	2046
10	935	1086	1050	1838	2496
50	3000	2969	2562	4118	5333
100	4929	3461	5039	6046	6299
200	9456	9603	10129	11524	12766
500	22691	22713	22702	23328	26273
1000	45872	43968	43807	44563	46780

(b) Using MATLAB and `ncx2inv` from MATLAB's statistics and machine learning toolbox.

Table 10.2.2 – The ratio of average computational times for evaluating the inverse non-central χ^2 cumulative distribution function against the inverse Gaussian cumulative distribution function for several values of the non-centrality parameter λ and degrees of freedom ν.

Looking at Table 10.2.2, we can see that even in the nicest regimes sampling from a non-central χ^2-distribution is approximately 40 times more expensive than a Gaussian, and can reach several hundred or thousand for difficult regimes. This alone demonstrates how expensive

it is to sample from the non-central χ^2-distribution. Furthermore, the increasing computational difficulty and its proportionality to the non-centrality parameter, as commented by Burkardt [38], is particularly clear for MATLAB's implementation.

Chapter 11

Conclusions

In the work presented, we produced a nested multilevel Monte Carlo framework using low-precision approximate random variables within the Euler-Maruyama scheme. To achieve this, we have constructed approximations to the Gaussian distribution, bound their errors, and provided high speed implementations designed for CPUs and GPUs. The framework and analysis is applicable to a variety of payoff functions, including when they are Lipschitz but not differentiable. Incorporating the effects of rounding error, an expected case error bound was developed, showing half-precision can be utilised on coarse levels. Lastly, we demonstrated these ideas are transferable to the Milstein scheme, and the more difficult Cox-Ingersoll-Ross process.

We began by discussing the inverse transform method and the Gaussian distribution, and introduced the general notion of approximate random variables, which are obtained by evaluating an approximation to a distribution's inverse cumulative distribution function. For the Gaussian distribution, due to its ubiquitous presence in scientific and financial computing, and its analytic tractability, we presented two possible approximation schemes. Both of these approximations were straightforward to motivate mathematically. The first was a piecewise constant function using equally spaced intervals over the function's domain, equivalent to an approximation presented by Giles et al. [90, 91]. The second was a novel piecewise linear function, spaced on geometric intervals dense at the singularities. For both these approximations, we analysed their convergence to the exact distribution with Lemmas 3.3.1 and 3.2.4 (extending the work by Giles et al. [90, 91]), where both could be made arbitrarily accurate by increasing the number of intervals. While both could achieve excellent fidelity, the piecewise linear approximation, for the same number of parameters, naturally achieved a much higher fidelity than the piecewise constant approximation.

For high speed implementations of these approximations, we assessed several library implementations of the exact function. Comparing the offerings of Intel, NAG, GNU, Cephes, and others, the critical obstacle for these implementations were the singularities, corresponding to the tails of the Gaussian distribution. For large vector widths, the parallel SIMD calculations would often diverge due to conditional branching, leading to poorer performance which only gets worse as the vectors become wider and their data-types smaller. Of all these libraries, the commercial offering of Intel's MKL was the best, averaging at approximately 4 or 8 clock-cycles per input for single and double-precision respectively. This was our target speed that required surpassing if approximate random variables were to be feasible.

Based on these findings, we developed two homogeneous (non-branching) routines, suitable for vector hardware, which implemented our proposed piecewise constant and piecewise linear approximations. The first of these was a lookup table implementation of the piecewise constant approximation. This was competitive if the table was small enough to fit in the private L_2 cache, reaching average times of 1–2 clock cycles in double-precision. The second was a vector register implementation of the piecewise linear approximation on

geometric intervals. For vectors which were 512-bit wide, the single-precision speed could be brought down to 0.5 clock cycles (with the I/O maximum being 0.4). Both of these approximations could achieve superior speeds to library implementations of the exact function. The two implementations had varying sophistication, where the lookup table was the simplest, most portable, and hardware agnostic. However, the piecewise linear approximation was the fastest, and we provided optimised implementations suitable for Arm and Intel hardware, written using OpenMP SIMD pragmas, vector intrinsics, and assembly instructions. These implementations also served to demonstrate the approximations could be constructed and tailored to specific hardware, aiming to achieve their computational speed by either exploiting memory and the cache-hierarchy or the floating-point units. As these approximations were homogenous, the greater the degree of vector parallelisation, the better the speed increase.

With the approximate random variables formulated, analysed, and implemented, we then introduced these into our numeric scheme, replacing exact Gaussian random variables with approximate ones in the Euler-Maruyama scheme. This is a natural generalisation to the weak Euler-Maruayama scheme using Rademacher random variables, and similarly the moment-matching scheme presented by Müller et al. [160]. The novel feature here was rather than altogether replacing the Euler-Maruyama scheme with the modified version using approximate random variables, we introduce it as a new estimator within a multilevel Monte Carlo framework. Repeating this within a multilevel Monte Carlo scheme using multiple discretisation levels, we introduced a nested multilevel Monte Carlo framework. The nested framework has the benefit of respecting the telescopic summation central to the multilevel Monte Carlo construction, which the version by Müller et al. [160] subtly violates. Furthermore, for the same reason, we demonstrated that our nested multilevel construction is not interpretable as a multi-index Monte Carlo [99].

Introducing approximate random variables into a single level Monte Carlo simulation, the strong error between the Euler-Maruyama estimates using exact and approximate random variables is bound only by the error of the approximate random variables, and is independent to the discretisation error. We then extended this to a multilevel Monte Carlo simulation. The usual two-way difference between fine and coarse paths was replaced with a version using approximate random variables, requiring a four-way difference for the multilevel correction. The variance of this four-way difference for the underlying process was bound in Theorem 6.2.6, and was shown to decay as the product (not the sum) of the discretisation error and the error from approximating the random number distribution. The two errors combining multiplicatively demonstrates a coupling between the two, where a multilevel Monte Carlo simulation would benefit from improving either. Our cornerstone results were extending Theorem 6.2.6 to payoff functions of the underlying process, considering Lipschitz and differentiable payoff functions in Corollary 6.2.6.2, and Lipschitz but non-differentiable payoffs in Corollary 6.2.6.3. For differentiable payoffs the core result from Theorem 6.2.6 immediately carries over into Corollary 6.2.6.2 without modification. However, for Lipschitz payoffs with a point of non-differentiability, we found a unique and novel transitioning behaviour. The temporal decay rate decreased as we transitioned from fine discretisations with low-fidelity approximate random variables to coarse discretisations with high-fidelity approximate random variables. Empirically we observed the behaviour predicted by Corollaries 6.2.6.2 and 6.2.6.3, and for payoffs with discontinuities there appeared little benefit from a direct incorporation of approximate random variables.

Taking our empirical findings for the variance reduction that results from incorporating approximate random variables and the computational speed increase, we are able to estimate the overall savings a practitioner could reasonably expect from our nested multilevel Monte Carlo framework. Taking a call option as a demonstrative example, we anticipate time savings of a factor of five or more, where the multilevel correction term could be computed once for every sixty simulations across all levels, all without losing accuracy.

For computing approximate random variables in low-precision, a model for the

accumulated rounding error incurred was developed. Taking the floating-point framework provided by Omland et al. [166, 167], we constructed a model (Assumption 8.4.6) for the rounding error sustained at each Euler-Maruyama update, paralleling the statistical model proposed by Arciniega and Allen [11]. However, our model described two primary sources of error: a leading order unbiased zero mean contribution, and a smaller second order systematic contribution. The cancellation of the leading order zero mean terms give an overall contribution equal to that arising from the smaller but systematic terms. Incorporating this into our nested multilevel Monte Carlo framework gives a separate error process, for which the expected average case rounding error grows at a rate proportional to the square-root of the number of Euler-Maruyama increments, as described by Theorem 8.5.1. The appeal of our model over the related model from Arciniega and Allen [11], aside from the wider scope of ours, is also the stronger basis for the zero mean nature of the leading order contribution. We justified this more rigorously in Lemma 8.4.4, using a detailed inspection of the composite floating-point arithmetic involved.

Performing simulations in double, single, and half-precision we were able to verify the separate effects of the four-way multilevel Monte Carlo correction and the collected rounding error. For double-precision the influence of rounding error was negligible, and similarly for single-precision it only became significant for extremely finely resolved simulations. However, for half-precision, rounding error became significant after the first few levels, restricting it to only the coarsest simulations. However, based on the multilevel Monte Carlo analysis by Giles [76], given the variance's decay rate for the Euler-Maruyama scheme, this was not catastrophic for half-precision. Rather, half-precision presented a real and viable candidate precision for coarse path simulations. Additionally, by incorporating a Kahan compensated summation to the Euler-Maruyama scheme, as Vitasek [204] did for ordinary differential equations, we were able to extend the utility of half-precision to a moderate range of discretisation levels.

The core machinery analysed in this thesis was the Gaussian distribution and the Euler-Maruyama scheme, primarily because of their analytic tractability, in addition to their wide spread and ubiquitous use. Nonetheless, we extended their use and empirically demonstrated the feasibility of approximate random variables beyond these. Firstly, we incorporated approximate Gaussian random variables into the Milstein scheme. The strong convergence rate of order 1 for the Milstein scheme was unaffected, and seemed to persist within our nested multilevel Monte Carlo framework. Furthermore, we were unable to detect any transitional behaviour for Lipschitz and non-differentiable payoff functions, making such an approach very attractive. Additionally, we considered the Cox-Ingersoll-Ross process, which required the construction of an approximate non-central χ^2-distribution. The variance was considerably better from switching to an approximate non-central χ^2-distribution rather than approximating the process using the Euler-Maruyama scheme. Alongside this, we demonstrated that the savings could be immense because exactly sampling from the non-central χ^2-distribution is extremely expensive, whereas it can be readily approximated using linear interpolation with our piecewise linear approximation over geometric intervals. Overall, this supports the hypothesis that approximate random variables are applicable to higher order schemes, harder classes of stochastic processes, and parametrised distributions.

11.1 Further work

Over the finite duration of this thesis we have identified several avenues for further research, explicitly performing some preliminary investigations in Chapter 10. The most immediately promising would be to incorporate approximate random variables into the Milstein scheme, and try to mirror Theorem 6.2.6. Our experimental investigations suggest that our previous results from Theorem 6.2.6 are paralleled, but with the improved order 1 rate of strong convergence. Aside from being of considerable academic interest, this would result in

the bulk of the computational work being focused on the coarsest levels [76]. This would have the added benefit of allowing half-precision calculations, compounding the speed improvements achievable. Furthermore, our results did not indicate any difficulty with Lipschitz and non-differentiable payoffs, which remains a slight drawback of the Euler-Maruyama scheme.

Similarly, if we could weaken our assumptions of Lipschitz continuity, possibly to $\frac{1}{2}$-Hölder continuity, then this could extend our scope to a wider class of stochastic processes, such as the Cox-Ingersoll-Ross process. This though is still likely limited in scope to processes where a strong convergence analysis can be achieved with the regular Euler-Maruyama scheme.

With a wider class of stochastic processes comes a wider class of distributions beyond the Gaussian, where we have already introduced the non-central χ^2-distribution. For various stochastic processes, including Lévy processes with jumps, the Poisson, Gamma, and central χ^2-distributions regularly occur [92, page 146]. We adapted one of our approximations of the Gaussian distribution to construct an approximation to the non-central χ^2-distribution, and similar constructions are likely possible for various other distributions. Furthermore, bounds for the approximation's error are likely attainable, such as for the Poisson or central χ^2-distributions.

Our approximations went up to piecewise linear in complexity, and we anticipate this can be extended to higher order polynomials. In Section 3.4 we also discussed higher order polynomials over the entire domain. Although we were unable to show analytic convergence results, we suspect they are achievable.

We found our Euler-Maruyama scheme with approximate random variables handled Lipschitz and differentiable payoff functions, and was still able to cope with Lipschitz and non-differentiable payoff functions, albeit with a transitional behaviour. However, payoffs with discontinuities, such as a digital option, currently remain out of reach. Using the Euler-Maruyama scheme with multilevel Monte Carlo to price digital options has been explored by Giles, Higham, and Mao [87] and Avikainen [14], and nested expectations requiring the heavy-side function recently by Giles and Haji-Ali [82, 83].[1] Using approximate random variables for pricing discontinuous payoff functions may be achievable, either analytically or only empirically, and would be an interesting direction to develop this work.

Applications beyond pricing are widespread, and commonly include sensitivity calculations (known as "Greeks") [79]. These methods include pathwise sensitivity, likelihood ratio methods, and finite difference methods. The likelihood ratio method has the advantage of not requiring differentiability of the payoff function, and hence fits nicely within the scope of Corollary 6.2.6.2, whereas the pathwise sensitivity method is well suited to the Euler-Maruyama scheme, but has more restrictive differentiability requirements. Seeing if approximate random variables can also be incorporated into these methods remains an interesting question warranting exploration.

Our analysis has as far as possible kept our assumptions of the underlying stochastic process as weak as possible, such as being non-autonomous, having Lipschitz spatial dependence, and $\frac{1}{2}$-Hölder temporal continuity. However, our examples have only used autonomous processes, and assessing our framework over a broader class of stochastic process still remains open. This will not only serve to assess our predictions, but also provide a larger body of evidence to the scientific and industrial community, working to convince them of the merits and applicability of approximate random variables and encourage its wider adoption and investigation.

[1] Giles and Haji-Ali also have recent work on efficient pricing methods for digital options, but there is currently no working title for this, and is based on personal communications between the authors.

Appendix A

Results from analysis and probability

Throughout this thesis we have referred to several standard and well known results, largely from analysis and probability. In this appendix we collect and list these result for completeness.

A.1 Real analysis

A.1.1 Convex analysis

Lemma A.1.1 (the Hermite-Hadamard inequality). *[97] [64, (1.1)] For a convex function* $f : [a, b] \to \mathbb{R}$ *we have the inequalities* $f(\frac{a+b}{2}) \leq \frac{1}{b-a} \int_a^b f(x)\,\mathrm{d}x \leq \frac{f(a)+f(b)}{2}$.

A.1.2 Differentiability

Theorem A.1.2 (Rademacher's theorem). *[103, Theorem 3.1] [67, Theorem 3.1.6, page 216] Let* $\Omega \subset \mathbb{R}^n$ *be an open set, and let* $f \colon \Omega \to \mathbb{R}^m$ *be a Lipschitz function, then* f *is differentiable almost everywhere in* Ω.

A.1.3 The fundamental theorem of calculus

Theorem A.1.3 (the fundamental theorem of calculus for Lebesgue integrals). *[16, page 82] If* $f(\cdot)$ *is absolutely continuous with Lebesgue measure* m *and is differentiable almost everywhere for* $x \in [u, v]$ *with* $f'(x) = g(x)$ *almost everywhere with respect to* m*, then* $f(x) - f(u) = \int_u^x g(s)\,\mathrm{d}s = \int_u^x f'(s)\,\mathrm{d}s$.

A.1.4 Intermediate values

Theorem A.1.4 (the intermediate value theorem). *[28, 184] For a continuous function* $f \colon [a, b] \to \mathbb{R}$ *where* $[a, b] \in \mathbb{R}$*, then for a number* u *within the range* $\min\{f(a), f(b)\} \leq u \leq \max\{f(a), f(b)\}$*, then there exists a number* $c \in (a, b)$ *such that* $u = f(c)$.

Theorem A.1.5 (the mean value theorem). *For a function* $f \colon \mathbb{R} \to \mathbb{R}$ *which is* $C^1(\mathbb{R})$*, then for the positions* x_1 *and* x_2 *there exists a positive weighting* $\xi = \lambda x_1 + (1 - \lambda)x_2$ *for some* $0 < \lambda < 1$ *such that* $f(x_1) - f(x_2) = (x_1 - x_2)f'(\xi)$.

A.1.5 The Peano kernel theorem

Theorem A.1.6 (the Peano kernel theorem). *[122, A.2.2.6, page 363] [178, Chapter 22, Theorem 22.1] [207] Let $L_x(\cdot)$ be a linear functional from the linear space $C^{\nu+1}([a,b])$ of functions defined on $[a,b]$ where their $(\nu+1)$-th derivative exists and is continuous in this interval. We suppose that for any bivariate function $f(\cdot,\cdot)$ that $L_x(\int_a^b f(x,\xi)\,\mathrm{d}\xi) = \int_a^b L_x(f(x,\xi))\,\mathrm{d}\xi$, where $f(\cdot,\xi), f(x,\cdot) \in C^{\nu+1}([a,b])$. Furthermore, we assume that L_x annihilates all polynomials $p(\cdot) \in \mathcal{P}_\nu$, where $\mathcal{P}_\nu$ is the set of all polynomial functions of order ν or less, such that $L_x(p) = 0$. We define the Peano kernel $k(\cdot)$ of L_x as $k(\xi) \coloneqq L_x((x-\xi)_+^\nu)$ for $\xi \in [a,b]$, where we use the notation $(\zeta)_+^m \coloneqq (\max\{0,\zeta\})^m$. If k is of bounded variation, then for any $g(\cdot) \in C^{\nu+1}([a,b])$ we have $L_x(g(x)) = \frac{1}{\nu!}\int_a^b k(\xi)\frac{\mathrm{d}^{\nu+1}g(\xi)}{\mathrm{d}\xi^{\nu+1}}\,\mathrm{d}\xi$.*

A.1.6 Inequalities

Lemma A.1.7 (Jensen's inequality for summations). *[173, pages 21, (1.2.24)] For the convex function $|\cdot|^p$ for any $p \geq 1$, we have $|\sum_{i=1}^n x_i|^p \leq n^{p-1}\sum_{i=1}^n |x_i|^p$.*

Lemma A.1.8 (Hölder's discrete inequality). *[173, page 21] Let $p,q \in (1,\infty)$ and $p^{-1} + q^{-1} = 1$, then for the sequences $\{a_i\}_{i=1}^n$ and $\{b_i\}_{i=1}^n$ we have $|\sum_{i=1}^n a_i b_i| \leq (\sum_{i=1}^n |a_i|^{\,p})^{1/p}(\sum_{i=1}^n |b_i|^{\,q})^{1/q}$.*

Lemma A.1.9 (Grönwall's discrete inequalities). *[61, 10.5.1.3] [116] If y_n, f_n, and g_n are non-negative sequences such that*

$$y_n \leq f_n + \sum_{0\leq k<n} g_k y_k \quad \forall\, n \geq 0, \tag{A.1.1}$$

then

$$y_n \leq f_n + \sum_{0\leq k<n} f_k g_k \prod_{k<j<n}(1+g_j) \leq f_n + \sum_{0\leq k<n} f_k g_k \exp\left(\sum_{k<j<n} g_j\right) \quad \forall\, n \geq 0. \tag{A.1.2}$$

Furthermore, if $f_n \leq \alpha$ for some non-negative constant α for all n, such that

$$y_n \leq \alpha + \sum_{0\leq k<n} g_k y_k \quad \forall\, n \geq 0, \tag{A.1.3}$$

then

$$y_n \leq \alpha \prod_{0\leq k<n}(1+g_k) \leq \alpha \exp\left(\sum_{0\leq k<n} g_k\right) \quad \forall\, n \geq 0. \tag{A.1.4}$$

Similarly, if $g_n \leq \beta$ for some non-negative constant β for all n, such that

$$y_n \leq \alpha + \sum_{0\leq k<n} \beta y_k \quad \forall\, n \geq 0, \tag{A.1.5}$$

then

$$y_n \leq \alpha(1+\beta)^n \leq \alpha\exp(n\beta) \quad \forall\, n \geq 0. \tag{A.1.6}$$

A.2 Probability theory

A.2.1 Conditional expectations

Definition A.2.1 (conditional expectation). [133, page 60, (2.17)] Let X be an integrable random variable on the probability space $(\Omega, \mathcal{A}, \mathbb{P})$, and let $\mathcal{S}$ be a sub-σ-algebra of $\mathcal{A}$. The

expectation of X conditioned on $\mathcal{S}$, denoted the *conditional expectation* $\mathbb{E}(X \,|\, \mathcal{S})$ is

$$\mathbb{E}(X \,|\, \mathcal{S})(\omega) = \begin{cases} \frac{1}{\mathbb{P}(A)} \int_A X \mathbb{P}(\mathrm{d}X) & \omega \in A \\ \frac{1}{\mathbb{P}(A^c)} \int_{A^c} X \mathbb{P}(\mathrm{d}X) & \omega \in A^c \end{cases} \tag{A.2.1}$$

for $\mathcal{S} = \{\emptyset, A, A^c, \Omega\}$ for some $A \in \mathcal{A}$ with $\mathbb{P}(A) \in (0,1)$. This can be more conveniently expressed in the simpler form $\mathbb{E}(X \mid Y) = \frac{\mathbb{E}(X \mathbb{1}_{\{Y\}})}{\mathbb{P}(Y)}$.

Theorem A.2.1 (the law of total expectation). *[183, 7.5.2, page 315, Proposition 5.1, (5.1)] [211, pages 380–383] [133, page 61, (2.18)] For two integrable random variables X and Y, defined on the same probability space $(\Omega, \mathcal{A}, \mathbb{P})$, then $\mathbb{E}_X(X) = \mathbb{E}_Y(\mathbb{E}_X(X \,|\, Y))$. Furthermore, for the nested σ-algebras $\mathcal{S}$ and $\mathcal{T}$ with $\mathcal{S} \subset \mathcal{T} \subset \mathcal{A}$ we have $\mathbb{E}(X \mid \mathcal{S}) = \mathbb{E}(\mathbb{E}(X \mid \mathcal{T}) \mid \mathcal{S})$.*

A.2.2 Inequalities

Lemma A.2.2 (Jensen's inequality for expectations). *[173, pages 21–22] For a random variable X and convex function $\phi(\cdot)$, if $\mathbb{E}(\phi(X))$ and $\phi(\mathbb{E}(X))$ are finite, then $\phi(\mathbb{E}(X)) \leq \mathbb{E}(\phi(X))$.*

Lemma A.2.3 (Hölder's inequality for expectations). *[173, page 22] [193] If $p, q \in (1, \infty)$ with $p^{-1} + q^{-1} = 1$, then for two random variables X and Y we have the following form of Hölder's inequality in expectation $\mathbb{E}(|XY|) \leq \mathbb{E}(|X|^p)^{1/p} \mathbb{E}(|Y|^q)^{1/q}$. The case when $p = q = 2$ is usually known as the Cauchy-Schwarz inequality.*

Lemma A.2.4 (Lyapunov's inequality). *[133, 1.4, page 17, (4.12)] [193, page 6] [173, 1.2] If a random variable X is not concentrated on a single point, and if $\mathbb{E}(|X|^s)$ exists for some $s > 0$, then $\mathbb{E}(|X - a|^r)^{1/r} \leq \mathbb{E}(|X - a|^s)^{1/s}$ for all $0 < r < s < \infty$ and $a \in \mathbb{R}$.*

Lemma A.2.5 (Markov's inequality). *[133, pages 17 and 60, (1.4.13) and (2.2.14)] For a random variable X and any constants $a, p > 0$ we have $\mathbb{P}(|X| \geq a) \leq \frac{\mathbb{E}(|X|^p)}{a^p}$.*

Lemma A.2.6 (Doob's maximal inequalities). *[131, Theorem 7.31] [15, Theorem 3] For a continuous time martingale $M(t)$ and a finite integer $1 < p < \infty$, then we have the bound $\mathbb{E}(\sup_{t_0 \leq s \leq T} |M(s)|^p) \leq (\frac{p}{p-1})^p \mathbb{E}(|M(T)|^p)$ provided $\mathbb{E}(|M(T)|^p) < \infty$. For a discrete time martingale $\{M_k\}_{k=0}^{\infty}$ we have $\mathbb{E}(\sup_{k \leq K} |M_k|^p) \leq (\frac{p}{p-1})^p \mathbb{E}(|M_K|^p)$ provided $\mathbb{E}(|M_K|^p) < \infty$.*

Lemma A.2.7 (the discrete Burkholder-Davis-Gundy inequality). *[40] [55, 14.3.2, Theorems 14.10–14.11, page 482] [131, 7.7] Let $\{X_n\}_{n \in \mathbb{N}}$ be a martingale with $X_0 = 0$, and define $X_n^+ \coloneqq \sup_{i \leq n} |X_i|$. For a convex function $\phi \colon \mathbb{R} \to \mathbb{R}$, there exists finite constants c_ϕ and C_ϕ where $0 < c_\phi < C_\phi < \infty$, such that $c_\phi \mathbb{E}(\phi(\langle X \rangle_n^{1/2})) \leq \mathbb{E}(\phi(X_n^+)) \leq C_\phi \mathbb{E}(\phi(\langle X \rangle_n^{1/2}))$. Furthermore, for the L^p-norm $\|\cdot\|_p$ with $p > 1$, which is a convex function, we have the bound $\frac{p-1}{18p^{3/2}} \|\langle X \rangle_n^{1/2}\|_p \leq \|X_n^+\|_p \leq \frac{18p^{3/2}}{(p-1)^{3/2}} \|\langle X \rangle_n^{1/2}\|_p$.*

References

[1] S.H. Abdel-Aty. Approximate formulae for the percentage points and the probability integral of the non-central χ^2 distribution. Biometrika, 41(3/4):538–540, 1954.

[2] Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra. Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs. In 2019 IEEE international parallel and distributed processing symposium (IPDPS), pages 111–122. IEEE, 2019.

[3] Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra. Towards half-precision computation for complex matrices: a case study for mixed precision solvers on GPUs. In 2019 IEEE/ACM 10th workshop on latest advances in scalable algorithms for large-scale systems (ScalA), pages 17–24. IEEE, 2019.

[4] Milton Abramowitz and Irene A. Stegun. Handbook of mathematical functions with formulas, graphs, and mathematical tables, volume 55. US government printing office, 1948. (6^{th} printing, November 1967).

[5] Peter J. Acklam. An algorithm for computing the inverse normal cumulative distribution function, October 2015. URL `https://web.archive.org/web/20151030212308/http://home.online.no/~pjacklam/notes/invnorm/index.html`. Accessed Tuesday 8^{th} September 2020.

[6] Aurélien Alfonsi. On the discretization schemes for the CIR (and Bessel squared) processes. Monte Carlo methods and applications, 11(4):355–384, 2005.

[7] Aurélien Alfonsi. A second-order discretization scheme for the CIR process: application to the Heston model. Preprint CERMICS hal-00143723, 14, 2008.

[8] Aurélien Alfonsi. High order discretization schemes for the CIR process: application to affine term structure and Heston models. Mathematics of computation, 79(269):209–237, 2010.

[9] David F. Anderson and Desmond J. Higham. Multilevel Monte Carlo for continuous time Markov chains, with applications in biochemical kinetics. Multiscale modeling & simulation, 10(1):146–179, 2012.

[10] David F. Anderson, Desmond J. Higham, and Yu Sun. Complexity of multilevel Monte Carlo tau-leaping. SIAM journal on numerical analysis, 52(6):3106–3127, 2014.

[11] Armando Arciniega and Edward Allen. Rounding error in numerical solution of stochastic differential equations. Stochastic analysis and applications, 21(2):281–300, 2003.

[12] Arm. Writing inline SVE assembly, May 2019. URL `https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/documentation/writing-inline-sve-assembly`.

[13] Søren Asmussen and Peter W. Glynn. Stochastic simulation: algorithms and analysis, volume 57. Springer science & business media, 2007.

[14] Rainer Avikainen. On irregular functionals of SDEs and the Euler scheme. Finance and stochastics, 13(3):381–401, 2009.

[15] Prakash Balachandran. Fundamental inequalities, convergence and the optional stopping theorem for continuous-time martingales, April 2008.

[16] Diómedes Bárcenas. The fundamental theorem of calculus for Lebesgue integral. Divulgaciones matemáticas, 8(1):75–85, 2000.

[17] Andrea Barth, Christoph Schwab, and Nathaniel Zollinger. Multi-level Monte Carlo finite element method for elliptic PDEs with stochastic coefficients. Numerische mathematik, 119(1):123–161, 2011.

[18] J.D. Beasley and S.G. Springer. Algorithm AS 111: the percentage points of the normal distribution. Journal of the royal statistical society. Series C (applied statistics), 26(1):118–121, March 1977.

[19] M.E.R. Berger. High performance histograms on SIMT and SIMD architectures, December 2015. Master's thesis, in computer science, Delft University of technology.

[20] Abdel Berkaoui, Mireille Bossy, and Awa Diop. Euler scheme for SDEs with non-Lipschitz diffusion coefficient: strong convergence. ESAIM: probability and statistics, 12:1–11, 2008.

[21] Serge Bernstein. Sur les polynomes orthogonaux relatifs à un segment fini (seconde partie). Journal de mathématiques pures et appliquées, 10:219–286, 1931.

[22] D.J. Best and D.E. Roberts. Algorithm AS 91: The percentage points of the χ^2 distribution. Journal of the royal statistical society. Series C (applied statistics), 24(3):385–388, 1975.

[23] Claudio Bierig and Alexey Chernov. Approximation of probability density functions by the multilevel Monte Carlo maximum entropy method. Journal of computational physics, 314:661–681, 2016.

[24] Birne Binegar. Errors in polynomial interpolation, 1998. URL `https://math.okstate.edu/people/binegar/4513-F98/4513-l16.pdf`. Lecture notes for Math 4513, Fall 1998.

[25] J. M. Blair, C. A. Edwards, and J. H. Johnson. Rational Chebyshev approximations for the inverse of the error function. Mathematics of computation, 30(136):827–830, 1976.

[26] Pierre Blanchard, Nicholas J. Higham, and Theo Mary. A class of fast and accurate summation algorithms. SIAM journal on scientific computing, 42(3):A1541–A1557, 2020.

[27] Stephen J. Blundell and Katherine M. Blundell. Concepts in thermal physics. Oxford University press, 2nd edition, 2014.

[28] Bernard Bolzano. Pure analytic proof of the doctrine: that between every two worthy of an opposite result, at least a real root of the equation, 1817. Original German title: rein analytischer beweis des lehrsatzes, daß zwischen je zwey werthen, die ein entgegengesetzes resultat gewähren, wenigstens eine reelle wurzel der gleichung liege.

[29] George E.P. Box, William H. Hunter, and Stuart Hunter. Statistics for experimenters, volume 664. John Wiley and sons, New York, 1978.

[30] John P. Boyd. The asymptotic Chebyshev coefficients for functions with logarithmic endpoint singularities: mappings and singular basis functions. Applied Mathematics and Computation, 29 (1):49 – 67, 1989.

[31] Thomas Bradley, Jacques du Toit, Robert Tong, Michael B. Giles, and Paul Woodhams. Parallelization techniques for random number generations. GPU computing gems emerald edition, 16:231–246, 2011.

[32] Henry Briggs. Logarithmorum chilias prima, 1617. London.

[33] Mark Broadie and Özgür Kaya. Exact simulation of stochastic volatility and other affine jump diffusion processes. Operations research, 54(2):217–231, 2006.

[34] Alfred L. Brophy. Approximation of the inverse normal distribution function. Behavior research methods, 17(3):415–417, 1985.

[35] Barry W. Brown, James Lovato, and Kathy Russell. CDFLIB: library of Fortran routines for cumulative distribution functions, inverses, and other parameters, February 1994.

[36] Christian Brugger, Christian de Schryver, Norbert Wehn, Steffen Omland, Mario Hefter, Klaus Ritter, Anton Kostiuk, and Ralf Korn. Mixed precision multilevel Monte Carlo on hybrid computing systems. In 2104 IEEE conference on computational intelligence for financial engineering & economics (CIFEr), pages 215–222. IEEE, 2014.

[37] Karolina Bujok, Ben M. Hambly, and Christoph Reisinger. Multilevel simulation of functionals of Bernoulli random variables with application to basket credit derivatives. Methodology and computing in applied probability, 17(3):579–604, 2015.

[38] John Burkardt. C source codes for CDFLIB, June 2019. URL `https://people.sc.fsu.edu/~jburkardt/c_src/cdflib/cdflib.html`. Accessed Wednesday 2nd September 2020.

[39] John Burkardt. MATLAB source codes for ASA241 and ASA111, April 2020. URL `https://people.sc.fsu.edu/~jburkardt/m_src/m_src.html`. Accessed Thursday 18th June 2020.

[40] Donald L. Burkholder, Burgess J. Davis, and Richard F. Gundy. Integral inequalities for convex functions of operators on martingales. In Proceedings of the sixth Berkeley symposium on mathematical statistics and probability, volume 2, pages 223–240, 1972.

[41] Jacques C.P. Bus and Theodorus Jozef Dekker. Two efficient algorithms with guaranteed convergence for finding a zero of a function. ACM transactions on mathematical software (TOMS), 1(4):330–345, 1975.

[42] Martin Campbell-Kelly. The history of mathematical tables: from Sumer to spreadsheets. Oxford University press, 2003.

[43] Claudio Canuto, Mohammed Yousuff Hussaini, Alfio Quarteroni, and Thomas A. Zang Jr. Spectral methods: fundamentals in single domains. Springer science & business media, 2007.

[44] Erin Carson, Nicholas J. Higham, and Srikara Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. MIMS eprint 2020.5, Manchester institute for mathematical sciences, the University of Manchester, UK, February 2020.

[45] Hongmei Chi, Peter Beerli, Deidre W. Evans, and Michael Mascagni. On the scrambled Sobol sequence. In International conference on computational science, pages 775–782. Springer, 2005.

[46] R.J. Chitashvili and N.L. Lazrieva. Strong solutions of stochastic differential equations with boundary conditions. Stochastics: an international journal of probability and stochastic processes, 5(4):255–309, 1981.
[47] Kai Lai Chung and Kailai Zhong. A course in probability theory. Academic press, 2001.
[48] Andrew K. Cliffe, Michael B. Giles, Robert Scheichl, and Aretha L. Teckentrup. Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients. Computing and visualization in science, 14(1):3, 2011.
[49] Marius Cornea. Intel AVX-512 instructions and their use in the implementation of math functions, June 2015. ARITH22: The 22nd IEEE symposium on computer arithmetic.
[50] Marius Cornea. Reference implementations for Intel architecture approximation instructions VRCP14, VRSQRT14, VRCP28, VRSQRT28, and VEXP2, December 2015. URL `https://software.intel.com/content/www/us/en/develop/articles/reference-implementations-for-ia-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2.html`. Accessed on Thursday 18th June 2020.
[51] John C. Cox, Jonathan E. Ingersoll Jr, and Stephen A. Ross. A theory of the term structure of interest rates. Econometrica, 53(2):385–408–164, March 1985.
[52] Andrei Cozma and Christoph Reisinger. Strong order 1/2 convergence of full truncation Euler approximations to the Cox–Ingersoll–Ross process. IMA journal of numerical analysis, 40(1):358–376, October 2018.
[53] Pascal Cuoq, Loïc Runarvot, and Alexander Cherepanov. Detecting strict aliasing violations in the wild. In International conference on verification, model checking, and abstract interpretation, pages 14–33. Springer, 2017.
[54] Ishaan L. Dalal, Deian Stefan, and Jared Harwayne-Gidansky. Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms. In International conference on application-specific systems, architectures and processors, ASAP 2008, pages 108–113. IEEE, 2008.
[55] Anirban DasGupta. Probability for statistics and machine learning: fundamentals and advanced topics. Springer science & business media, 2011.
[56] Philip J. Davis and Philip Rabinowitz. Methods of numerical integration. Courier corporation, 2007.
[57] Stefano De Marco. Smoothness and asymptotic estimates of densities for SDEs with locally smooth coefficients and applications to square root-type diffusions. The annals of applied probability, 21 (4):1282–1321, 2011.
[58] Arnaud Debussche and Nicolas Fournier. Existence of densities for stable-like driven SDE's with Holder continuous coefficients. Journal of functional analysis, 264:1757–1778, 04 2013.
[59] Griselda Deelstra and Freddy Delbaen. Convergence of discretized stochastic (interest rate) processes with stochastic drift term. Applied stochastic models and data analysis, 14(1):77–84, 1998.
[60] Steffen Dereich, Andreas Neuenkirch, and Lukasz Szpruch. An Euler-type method for the strong approximation of the Cox-Ingersoll-Ross process. Proceedings of the royal society A: mathematical, physical and engineering sciences, 468(2140):1105–1115, 2012.
[61] Jean Dieudonné. Foundations of modern analysis, volume 10 of Pure and applied mathematics. Academic press, enlarged and corrected printing edition, 1969.
[62] Rohan Douglas and Jamie Elliot. Vectorization: the rise of parallelism, June 2017. Quantifi.
[63] Ulrich Drepper. What every programmer should know about memory, November 2007. Red Hat, Inc.
[64] Abdallah el Farissi. Simple proof and refinement of Hermite-Hadamard inequality. Journal of mathematical inequalities, 4(3):365–369, 2010.
[65] Alaa Eltablawy and Andrey Vladimirov. Capabilities of Intel AVX-512 and Intel Xeon scalable processors, September 2017.
[66] Markus Fasselt. Restrict, static, & inline keywords in C, March 2014. Universität Hamburg.
[67] Herbert Federer. Geometric measure theory. Springer, 1969.
[68] William Feller. Two singular diffusion problems. Annals of mathematics, pages 173–182, 1951.
[69] Agner Fog. Instruction tables: lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs, 2018. Updated 9th April 2018.
[70] Conrad Georg Foik. Energy efficient true random number generator. Master's thesis, NTNU, 2015.
[71] Gerald B. Folland. Fourier analysis and its applications. Wadsworth & Brooks, 1992.
[72] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, and Rhys Ulerich. GNU scientific library 2.4, June 2017.
[73] Dani Gamerman and Hedibert F. Lopes. Markov chain Monte Carlo: stochastic simulation for Bayesian inference. CRC press, 2006.
[74] Roger Graham Garside. The architecture of digital computers. Oxford University press, 1980.

[75] John Geweke. Bayesian inference in econometric models using Monte Carlo integration. Econometrica, 57(6):1317–1339, 1989.

[76] Michael B. Giles. Multilevel Monte Carlo path simulation. Operations research, 56(3):607–617, 2008.

[77] Michael B. Giles. Approximating the `erfinv` function. In GPU computing gems, jade edition, volume 2, pages 109–116. Elsevier, 2011.

[78] Michael B. Giles. Multilevel Monte Carlo methods. In Monte Carlo and quasi-Monte Carlo methods 2012, pages 83–103. Springer Berlin Heidelberg, 2013.

[79] Michael B. Giles. Multilevel Monte Carlo methods. Acta numerica, 24:259–328, 2015.

[80] Michael B. Giles. Algorithm 955: approximation of the inverse Poisson cumulative distribution function. ACM transactions on mathematical software (TOMS), 42(1):7, 2016.

[81] Michael B. Giles and Takashi Goda. Decision-making under uncertainty: using MLMC for efficient estimation of EVPPI. Statistics and computing, 29(4):739–751, 2019.

[82] Michael B. Giles and Abdul-Lateef Haji-Ali. Multilevel nested simulation for efficient risk estimation. SIAM/ASA journal on uncertainty quantification, 7(2):497–525, 2019.

[83] Michael B. Giles and Abdul-Lateef Haji-Ali. Sub-sampling and other considerations for efficient risk estimation in large portfolios. arXiv preprint arXiv:1912.05484, 2019.

[84] Michael B. Giles and Christoph Reisinger. Stochastic finite differences and multilevel Monte Carlo for a class of SPDEs in finance. SIAM journal on financial mathematics, 3(1):572–592, 2012.

[85] Michael B. Giles and Lukasz Szpruch. Antithetic multilevel Monte Carlo estimation for multi-dimensional SDEs without Lévy area simulation. The annals of applied probability, 24(4):1585–1620, 2014.

[86] Michael B. Giles and Benjamin J. Waterhouse. Multilevel quasi-Monte Carlo path simulation. Advanced financial modelling, Radon series on computational and applied mathematics, 8:165–181, 2009.

[87] Michael B. Giles, Desmond J. Higham, and Xuerong Mao. Analysing multi-level Monte Carlo for options with non-globally Lipschitz payoff. Finance and stochastics, 13(3):403–413, 2009.

[88] Michael B. Giles, Tigran Nagapetyan, and Klaus Ritter. Multilevel Monte Carlo approximation of distribution functions and densities. SIAM/ASA journal on uncertainty quantification, 3(1): 267–295, 2015.

[89] Michael B. Giles, Mario Hefter, Lukas Mayer, and Klaus Ritter. Random bit multilevel algorithms for stochastic differential equations. Journal of complexity, 2019.

[90] Michael B. Giles, Mario Hefter, Lukas Mayer, and Klaus Ritter. Random bit quadrature and approximation of distributions on Hilbert spaces. Foundations of computational mathematics, 19 (1):205–238, 2019.

[91] Michael B. Giles, Mario Hefter, Lukas Mayer, and Klaus Ritter. An adaptive random bit multilevel algorithm for SDEs. Multivariate algorithms and information-based complexity, pages 15–32, May 2020.

[92] Paul Glasserman. Monte Carlo methods in financial engineering, volume 53 of Stochastic modelling and applied probability. Springer science & business media, 1st edition, 2013.

[93] David Goldberg. What every computer scientist should know about floating-point arithmetic. ACM computing surveys (CSUR), 23(1):5–48, 1991.

[94] István Gyöngy. A note on Euler's approximations. Potential analysis, 8(3):205–216, 1998.

[95] István Gyöngy and Miklós Rásonyi. A note on Euler approximations for SDEs with Hölder continuous diffusion coefficients. Stochastic processes and their applications, 121(10):2189–2200, 2011.

[96] Sven-Hendrik Haase. Alignment in C, January 2014. Universität Hamburg.

[97] Jacques Salomon Hadamard. Étude sur les propriétés des fonctions entiéres et en particulier d'une fonction considérée par Riemann. Journal de mathématiques pures et appliquées, 58:171–215, January 1893.

[98] Azzam Haidar, Panruo Wu, Stanimire Tomov, and Jack Dongarra. Investigating half precision arithmetic to accelerate dense linear system solvers. In Proceedings of the 8th workshop on latest advances in scalable algorithms for large-scale systems, pages 1–8, 2017.

[99] Abdul-Lateef Haji-Ali, Fabio Nobile, and Raúl Tempone. Multi-index Monte Carlo: when sparsity meets sampling. Numerische mathematik, 132(4):767–806, 2016.

[100] Jonathan Harter and Adrien Richou. A stability approach for solving multidimensional quadratic BSDEs. Electronic journal of probability, 24, 2019.

[101] Cecil Hastings Jr, Jeanne T. Wayward, and James P. Wong Jr. Approximations for digital computers. Princeton University press, 1955.

[102] Masafumi Hayashi, Arturo Kohatsu-Higa, and Gô Yûki. Local Hölder continuity property of the densities of solutions of SDEs with singular coefficients. Journal of theoretical probability, 26(4): 1117–1134, 2013.

[103] Juha Heinonen. Lectures on Lipschitz analysis, 2005. University of Jyväskylä.

[104] Stefan Heinrich. Monte Carlo complexity of global solution of integral equations. Journal of complexity, 14(2):151–175, 1998.

[105] John L. Hennessy and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

[106] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. The review of financial studies, 6(2):327–343, 1993.

[107] Desmond J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. SIAM review, 43(3):525–546, 2001.

[108] Desmond J. Higham, Xuerong Mao, and Andrew M. Stuart. Strong convergence of Euler-type methods for nonlinear stochastic differential equations. SIAM journal on numerical analysis, 40(3): 1041–1063, 2002.

[109] Nicholas J. Higham. The accuracy of floating point summation. SIAM journal on scientific computing, 14(4):783–799, 1993.

[110] Nicholas J. Higham. Accuracy and stability of numerical algorithms, volume 80. SIAM, 2$^{\text{nd}}$ edition, 2002.

[111] Nicholas J. Higham and Theo Mary. A new approach to probabilistic rounding error analysis. SIAM journal on scientific computing, 41(5):A2815–A2835, 2019.

[112] Nicholas J. Higham and Srikara Pranesh. Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems. MIMS preprint, 2019.

[113] Nicholas J. Higham, Srikara Pranesh, and Mawussi Zounon. Squeezing a matrix into half precision, with an application to solving linear systems. SIAM journal on scientific computing, 41(4):A2536–A2551, 2019.

[114] David C. Hoaglin. Direct approximations for chi-squared percentage points. Journal of the American statistical association, 72(359):508–515, 1977.

[115] Viet Ha Hoang, Christoph Schwab, and Andrew M. Stuart. Complexity analysis of accelerated MCMC methods for Bayesian inversion. Inverse problems, 29(8):085010, 2013.

[116] John M. Holte. Discrete Gronwall lemma and applications, October 2009. URL `http://homepages.gac.edu/~holte/publications/GronwallLemma.pdf`. Accessed Tuesday 8$^{\text{th}}$ September 2020.

[117] Stefano M. Iacus. Simulation and inference for stochastic differential equations: with R examples. Springer science & business media, 2009.

[118] IEEE. IEEE standard for binary floating-point arithmetic, 1985. Computer society standards committee. Working group of the microprocessor standards subcommittee.

[119] IEEE. IEEE standard for binary floating-point arithmetic, 2008. Computer society standards committee. Working group of the microprocessor standards subcommittee.

[120] International organization for standardization (ISO). ISO/IEC 9899:2011: programming languages–C. ISO working group, 14, 2012.

[121] International organization for standardization (ISO). ISO/IEC 9899:2018: programming languages–C. ISO working group, 2018.

[122] Arieh Iserles. A first course in the numerical analysis of differential equations. Cambridge University press, 1996.

[123] Tobias Jahnke. Numerical methods in mathematical finance, February 2017. Karlsruhe institute of technology.

[124] Norman L. Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. Continuous univariate distributions. John Wiley & sons, Ltd, 1995.

[125] Samuel Jones. Optimising cache use: `malloc()` revisited. Dissertation, University of Bath, May 2010.

[126] Corwin Joy, Phelim P. Boyle, and Ken Seng Tan. Quasi-Monte Carlo methods in numerical finance. Management science, 42(6):926–938, 1996.

[127] Mark Kac. On distributions of certain Wiener functionals. Transactions of the American mathematical society, 65(1):1–13, 1949.

[128] William Kahan. The improbability of probabilistic error analyses for numerical computations. In UCB statistics colloquium, Evans Hall edition, page 20, 1996.

[129] William Morton Kahan. Further remarks on reducing truncation errors. Communications of the association for computing machinery (ACM), 8:40, 1965.

[130] Ahmed Kebaier. Statistical Romberg extrapolation: a new variance reduction method and applications to option pricing. The annals of applied probability, 15(4):2681–2705, 2005.

[131] Fima C. Klebaner. Introduction to stochastic calculus with applications. Imperial College press, 3 edition, 2012.

[132] Michael Klemm, Alejandro Duran, Xinmin Tian, Hideki Saito, Diego Caballero, and Xavier Martorell. Extending OpenMP with vector constructs for modern multicore SIMD architectures. In International workshop on OpenMP, pages 59–72. Springer, 2012.

[133] Peter E. Kloeden and Eckhard Platen. Numerical solution of stochastic differential equations, volume 23 of Stochastic modelling and applied probability. Springer, 1999. Corrected 3rd printing.

[134] Donald E. Knuth. The art of computer programming: searching and sorting, volume 3. Reading, MA, Addison-Wesley, 1973.

[135] Donald E. Knuth. The art of computer programming: seminumerical algorithms, volume 2. Addison-Wesley Professional, 2014.

[136] Arturo Kohatsu-Higa and Azmi Makhlouf. Estimates for the density of functionals of SDEs with irregular drift. Stochastic processes and their applications, 123(5):1716 – 1728, 2013.

[137] Robbert Krebbers. Aliasing restrictions of C11 formalized in Coq. In International conference on certified programs and proofs, pages 50–65. Springer, 2013.

[138] Seiichiro Kusuoka. Existence of densities of solutions of stochastic differential equations by Malliavin calculus. Journal of functional analysis, 258(3):758–784, 2010.

[139] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In SC'06: proceedings of the 2006 ACM/IEEE conference on supercomputing, pages 50–50. IEEE, 2006.

[140] Samuel Larsen and Saman Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets, 2000.

[141] Samuel Larsen, Emmett Witchel, and Saman Amarasinghe. Increasing and detecting memory address congruence. In Proceedings. International conference on parallel architectures and compilation techniques, pages 18–29. IEEE, 2002.

[142] Mark Lee, Brian Green, Feng Xie, and Eric Tabellion. Vectorized production path tracing. In Proceedings of high performance graphics, page 10. ACM, 2017.

[143] David Levinthal. Performance analysis guide for Intel core i7 processor and Intel Xeon 5500 processors, 2008. URL `https://software.intel.com/content/www/us/en/develop/articles/processor-specific-performance-analysis-papers.html`. Accessed Friday 2nd October 2020.

[144] Georg Lohöfer. Inequalities for Legendre functions and Gegenbauer functions. Journal of approximation theory, 64(2):226–234, 1991.

[145] Gabriel J. Lord, Catherine E. Powell, and Tony Shardlow. An introduction to computational stochastic PDEs. Cambridge University press, 2014.

[146] Roger Lord, Remmert Koekkoek, and Dick van Dijk. A comparison of biased simulation schemes for stochastic volatility models. Quantitative finance, 10(2):177–194, 2010.

[147] David Loshin. High performance computing demystified. Academic press, 1994.

[148] David Loshin. Efficient memory programming. McGraw-Hill, Inc, 2001.

[149] Piotr Luszczek, Ichitaro Yamazaki, and Jack Dongarra. Increasing accuracy of iterative refinement in limited floating-point arithmetic on half-precision accelerators. In 2019 IEEE high performance extreme computing conference (HPEC), pages 1–6. IEEE, 2019.

[150] George Marsaglia, Arif Zaman, and John C.W. Marsaglia. Rapid evaluation of the inverse of the normal distribution function. Statistics & probability letters, 19(4):259–266, March 1994.

[151] Alan D. Martin. Unitarity and high-energy behavior of scattering amplitudes. Physical review, 129:1432–1436, February 1963.

[152] Matlab MathWorks. Statistics and machine learning toolbox: user's guide, 2018. R2018b.

[153] Simon McIntosh-Smith, James Price, Tom Deakin, and Andrei Poenaru. Comparative benchmarking of the first generation of HPC-optimised Arm processors on Isambard. Cray user group, 5, 2018.

[154] Simon McIntosh-Smith, James Price, Tom Deakin, and Andrei Poenaru. A performance analysis of the first generation of HPC-optimized Arm processors. Concurrency and computation: practice and experience, 31(16):e5110, 2019.

[155] Simon McIntosh-Smith, James Price, Andrei Poenaru, and Tom Deakin. Benchmarking the first generation of production quality Arm-based supercomputers. Concurrency and computation: practice and experience, page e5569, 2019.

[156] Simon McIntosh-Smith, James Price, Andrei Poenaru, and Tom Deakin. Scaling results from the first generation of Arm-based supercomputers. In Cray user group meeting, 2019.

[157] Gaurav Mitra, Beau Johnston, Alistair P. Rendell, Eric McCreath, and Jun Zhou. Use of SIMD vector operations to accelerate application code performance on low-powered ARM and Intel

platforms. In Parallel and distributed processing symposium workshops & PhD forum (IPDPSW), 2013 IEEE 27th international, pages 1107–1116. IEEE, 2013.

[158] Boris Moro. The full Monte. Risk, 8(2):57–58, 1995.

[159] Stephen L. Moshier. Cephes mathematical library, 1992. URL `http://www.netlib.org/cephes/`. Accessed Tuesday 8th September 2020.

[160] Eike H. Müller, Rob Scheichl, and Tony Shardlow. Improving multilevel Monte Carlo for stochastic differential equations with application to the Langevin equation. Proceedings of the royal society of London A: mathematical, physical and engineering sciences, 471(2176), 2015.

[161] Claus Munk. Fixed income modelling. Oxford University press, 2011.

[162] Yuji Nakatsukasa. Approximate and integrate: variance reduction in Monte Carlo integration via function approximation. arXiv preprint arXiv:1806.05492, 2018.

[163] Jerome Newman and Walter Rudin. Mean convergence of orthogonal series. Proceedings of the American mathematical society, 3(2):219–222, 1952.

[164] David Ba nos and Paul Krühner. Hölder continuous densities of solutions of SDEs with measurable and path dependent drift coefficients, 2016. URL `https://arxiv.org/abs/1604.08181`. arXiv preprint.

[165] Robert E. Odeh and J.O. Evans. Algorithm AS 70: the percentage points of the normal distribution. Journal of the royal statistical society. Series C (applied statistics), 23(1):96–97, 1974.

[166] Steffen Omland. Mixed precision multilevel Monte Carlo algorithms for reconfigurable computing systems, June 2016. PhD thesis/dissertation, D 386, Technische Universität Kaiserslautern (TUK).

[167] Steffen Omland, Mario Hefter, Klaus Ritter, Christian Brugger, Christian De Schryver, Norbert Wehn, and Anton Kostiuk. Exploiting mixed-precision arithmetics in a multilevel Monte Carlo approach on FPGAs. In FPGA based accelerators for financial applications, pages 191–220. Springer, 2015.

[168] Michael L. Overton. Floating point representation, 1996. URL `http://eaton.math.rpi.edu/CourseMaterials/Fall01/KU4800/Misc/overton.pdf`. An unpublished note, accessed Tuesday 8th September 2020.

[169] Gabriele Paoloni. How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures. Intel white paper, 2010. URL `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf`.

[170] Francesco Petrogalli. A sneak peek into SVE and VLA programming, November 2016. White paper.

[171] Matt Pharr and William R. Mark. `ispc`: A SPMD compiler for high-performance CPU programming. In Innovative parallel computing (InPar), 2012, pages 1–13. IEEE, 2012.

[172] Eckhard Platen. An introduction to numerical methods for stochastic differential equations. Acta numerica, 8:197–246, 1999.

[173] Eckhard Platen and Nicola Bruti-Liberati. Numerical solution of stochastic differential equations with jumps in finance, volume 64 of Stochastic modelling and applied probability. Springer, 2010.

[174] Harry Pollard. The mean convergence of orthogonal series of polynomials. Proceedings of the national academy of sciences of the United States of America, 32(1):8, 1946.

[175] Harry Pollard. The mean convergence of orthogonal series. I. Transactions of the American mathematical society, 62(3):387–403, 1947.

[176] Harry Pollard. The mean convergence of orthogonal series. II. Transactions of the American mathematical society, 63(2):355–367, 1948.

[177] Harry Pollard. The convergence almost everywhere of Legendre series. Proceedings of the American mathematical society, 35(2):442–444, 1972.

[178] Michael James David Powell. Approximation theory and methods. Cambridge University press, 1981.

[179] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical recipes in C. Cambridge University press, 3rd edition, 2007.

[180] Stephen O. Rice. Uniform asymptotic expansions for saddle point integrals—application to a probability distribution occurring in noise theory. Bell system technical journal, 47(9):1971–2013, 1968.

[181] Klaus Ritter. Asymptotic optimality of regular sequence designs. The annals of statistics, 24(5): 2081–2096, 1996.

[182] Denis Roegel. A reconstruction of Briggs' Logarithmorum chilias prima (1617), 2010.

[183] Sheldon Ross. A first course in probability. Pearson, 9th edition, 2014.

[184] Steve B. Russ. A translation of Bolzano's paper on the intermediate value theorem. Historia mathematica, 7(2):156–185, 1980.

[185] Mark Sabahi et al. A guide to vectorization with Intel C++ compilers, April 2017. Intel.

[186] Munuswamy Sankaran. On the non-central chi-square distribution. Biometrika, 46(1/2):235–237, 1959.

[187] Robert Scheichl, Andrew M. Stuart, and Aretha L. Teckentrup. Quasi-Monte Carlo and multilevel Monte Carlo methods for computing posterior expectations in elliptic inverse problems. SIAM/ASA journal on uncertainty quantification, 5(1):493–518, 2017.

[188] Rüdiger Seydel. Tools for computational finance, volume 3. Springer, 2006.

[189] William T. Shaw and Nick Brickman. Differential equations for Monte Carlo recycling and a GPU-optimized normal quantile, 2009.

[190] B.L. Shea. Algorithm AS R85: a remark on AS 91: the percentage points of the χ^2 distribution. Journal of the royal statistical society. Series C (applied statistics), 40(1):233–235, 1991.

[191] Ronald W. Shonkwiler and Lew Lefton. An introduction to parallel and vector scientific computation. Cambridge texts in applied mathematics. Cambridge University press, 2006.

[192] Richard M. Stallman and the GCC developer community. Using the GNU compiler collection, 2020. Version 10.1.0.

[193] David A. Stephens. 556: mathematical statistics I, 2014. McGill University.

[194] Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, Alastair Reid, Alejandro Rico, and Paul Walker. The ARM scalable vector extension. IEEE micro, 37(2):26–39, 2017.

[195] Gábor Szegő. Orthogonal polynomials, volume 23. American mathematical society, 1939.

[196] M. Tahmasebi and S. Zamani. Integration by parts formula and smoothness of densities of solutions to SDEs with locally Lipschitz coefficients, 2013. URL `https://arxiv.org/abs/1309.2781`. arXiv pre-print.

[197] Luke Tierney and Antonietta Mira. Some adaptive Monte Carlo methods for Bayesian inference. Statistics in medicine, 18(17-18):2507–2515, 1999.

[198] Lloyd N. Trefethen. Approximation theory and approximation practice, volume 128. SIAM, 2013.

[199] Lloyd N. Trefethen and David Bau. Numerical linear algebra, volume 50. SIAM, 3$^{\text{rd}}$ edition, 1997.

[200] Lloyd N. Trefethen, Yuji Nakatsukasa, and J.A.C. Weideman. Rational approximation of functions with log singularities, 2020. In preparation.

[201] Warwick Tucker. Validated numerics: a short introduction to rigorous computations. Princeton University press, 2011.

[202] Ruud van der Pas, Eric Stotzer, and Christian Terboven. Using OpenMP — the next step: affinity, accelerators, tasking, and SIMD. MIT press, 2017.

[203] Cédric Villani. Optimal transport: old and new, volume 338. Springer science & business media, 2008.

[204] Emil Vitasek. The numerical stability in solution of differential equations. In Conference on the numerical solution of differential equations, pages 87–111. Springer, 1969.

[205] Paul M. Voutier. A new approximation to the normal distribution quantile function. arXiv preprint arXiv:1002.0567, February 2010.

[206] Anders Vretblad. Fourier analysis and its applications, volume 223. Springer science & business media, 2003.

[207] Shayne Waldron. Refinements of the Peano kernel theorem. Numerical functional analysis and optimization, 20:147–161, 01 1999.

[208] Haiyong Wang. A new and sharper bound for Legendre expansion of differentiable functions. Applied mathematics letters, 85:95–102, 2018.

[209] Haiyong Wang and Shuhuang Xiang. On the convergence rates of Legendre approximation. Mathematics of computation, 81(278):861–877, 2012.

[210] Marcus Webb, Vincent Coppé, and Daan Huybrechs. Pointwise and uniform convergence of Fourier extensions. Constructive approximation, pages 1–37, 2019.

[211] Neil A. Weiss. A course in probability. Pearson Addison Wesley, 2006.

[212] John S. White. Tables of normal percentile points. Journal of the American statistical association, 65(330):635–638, 1970.

[213] Michael J. Wichura. Algorithm AS 241: the percentage points of the normal distribution. Journal of the royal statistical society. Series C (applied statistics), 37(3):477–484, April 1988.

[214] Edwin B. Wilson and Margaret M. Hilferty. The distribution of chi-square. proceedings of the national academy of sciences of the United States of America, 17(12):684, 1931.

[215] Markus Wittmann, Thomas Zeiser, Georg Hager, and Gerhard Wellein. Short note on costs of floating point operations on current x86-64 architectures: Denormals, overflow, underflow, and division by zero. arXiv preprint arXiv:1506.03997, 2015.

[216] Shuhuang Xiang and Folkmar Bornemann. On the convergence rates of Gauss and Clenshaw-Curtis quadrature for functions of limited regularity. SIAM journal on numerical analysis, 50(5): 2581–2587, 2012.

[217] Chuang Xu and Arno Berger. Best finite constrained approximations of one-dimensional probabilities. Journal of approximation theory, 244, 04 2017.