LONDON'S GLOBAL UNIVERSITY

Review of the Inverse $\mathcal{Z}$-Transform and Its Use in the Pricing of Discrete Path-Dependent Options

Final Year Project Report

Candidate Number: KCPD9[1]

Dr Carolyn Phelan

Department of Computer Science
University College London

Submission date: May 23, 2024

[1]**Disclaimer:** This report is submitted as part requirement for the MEng degree in Mathematical Computation at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Acknowledgements

"And that there is not for man except that for which he strives."
(Surah An-Najm, No. 53, Ayat 39)

I want to extend my heartfelt gratitude to my beloved mother and her support through my journey. Her sacrifices, encouragement and constant duas (prayers) have brought me to where I am today. May Allah shower my mother with endless happiness and contentment in every aspect of her life.

I'd also like to extend my thanks to my dear brother and sister, outer family and friends for their continuous encouragement, support and motivation. I am truly grateful for your presence in my life, and for the countless ways in which you have contributed to my success and enjoyment.

To Carolyn Phelan - I am profoundly grateful for the continuous support and guidance provided during my final year of study. Your support was not limited to academic matters and provided a source of inspiration. I would also like to extend my thanks to Guido Germano and Benjamin Loveless for their indirect support through their academic publications.

Finally, I owe it to myself for putting my head down and persevering to complete this degree. Whilst there were many ups and downs along this journey and moments of self-doubt, I think it's safe to say I've made it, Alhamdulillah.

Abstract

This study explores the literature on the inverse z-transform and its application in pricing discrete path-dependent options, a financial derivative whose payoffs depend on the path taken by the underlying asset price. Such options are labelled as exotic options, including lookback and barrier options. By examining various numerical approximation methods for the inverse z-transform, the research aims to evaluate their efficiency and accuracy. The focus is on the methods proposed by Abate and Whitt (1992a,b) and Cavers (1978), as well as the new approach by Boyarchenko and Levendorskiĭ (2022) involving a sinh deformation of the contour.

We propose to back-track the deformation into a contour resembling the commonly used circular path through the use of optimisation algorithms, such as gradient descent and its variant, adaptive moment estimation. Extensive tests were carried out to analyse the effects of the parameters for the different numerical methods in terms of their efficiency and efficacy, and to determine the optimal setup.

Through the employment of the optimisation techniques, we find parameters for the conformal mapping, involving the sinh deformation, that resemble the unit circle to an accuracy greater than 1E-16. We provide a successful outlook on the trade-offs between accuracy and speed for the approximation methods, discuss the setups needed to achieve such results and offer efficient implementations of the methods.

Contents

Chapter 1

Introduction

1.1 Motivation

Standard (vanilla) options are the most basic and commonly traded options that allow the holder to buy or sell the asset at a later time. We attribute the first recorded option contract to that of the Greek philosopher Thales of Miletus (Thompson, 1994). Believing that the olive harvest would be plentiful, Thales secured the right to use the olive presses at a low price. This agreement is considered a call option as it gives the holder the right, but not the obligation, to execute the contract at a predetermined price.

The formal study of pricing options began much later with Bachelier (1900)'s work in modelling stock prices as a Brownian motion and his derivation of an option pricing formula. However, we deem the seminal work of Black and Scholes (1973) in laying out the foundation for modern option pricing theory. The introduction of the Black-Scholes model coincided with the establishment of the Chicago Board Options Exchange (CBOE) in 1973, standardising option contracts and providing a trading platform. The rapid growth (Fig 1.1) in the options market sparked a demand and increased interest in option pricing research.

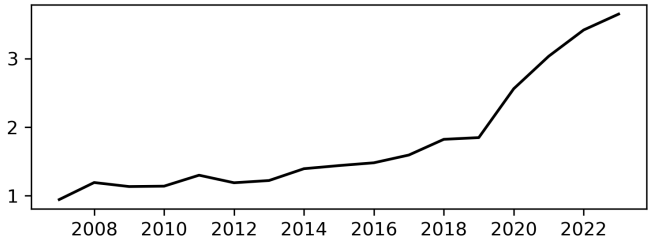


Figure 1.1: Visualisation of the volume of options (in billions) traded over the past 17 years on exchanges CBOE, BATS, C2 and EDGX.

However, the assumptions of the Black-Scholes model do not lend themselves to an accurate representation of real-world asset prices. Furthermore, exotic options differ in terms of their structure, payoff, and underlying assets to meet specific investor needs or market conditions. For instance, path-dependent options have payoffs determined by the path of the underlying asset price over the life of the option rather than just the price at expiration. The price of the underlying asset is monitored at discrete points in time. One could say that in practice, most, if not all, path-dependent options in markets are discrete path-dependent options (Kou, 2007).

The pricing of discrete path-dependent options has been a longstanding area of research in the financial mathematics literature. The importance and value of these options results in a vast array of research including around geometric Brownian motions (Lu and Jin, 2017; Guardasoni et al., 2020), general Lévy processes (Fang and Oosterlee, 2009a; Fusai et al., 2016; Phelan, 2018; Chen et al., 2021; Boyarchenko and Levendorskiĭ, 2022), general Markov processes (Cui and Taylor, 2021; Zhang and Li, 2023) and stochastic volatility (Soleymani and Barfeie, 2019; Kirkby and Nguyen, 2020).

A common problem faced by those in the literature is the linear dependency on the monitoring dates of the underlying asset price. Fusai et al. (2016) demonstrated that the iteration on the monitoring dates can be avoided by working in the z-domain, extending on work by that of Carr and Madan (1999). Applying the Spitzer (1957) identities and solving the resulting equations requires an inverse operation to obtain the option price. Reverting from the z-domain to the time domain requires a numerical approximation of the inverse z-transform. Despite its relation to the Fourier transform, the literature lacks in comparison; the performance of existing approximations, in terms of accuracy and efficiency, necessitates a specific setup, choice of parameters and desired trade-off.

1.2 Aims and Objectives

Given the importance of the numerical inverse z-transform (NIZT), the focus of this project will be **(i)** studying the effect of the parameters on the varying methods, and **(ii)** testing on a group of well-defined transform pairs. Thus, we aim to try and discover which methods provide the best fit in terms of efficiency and accuracy and what the best setup to achieve this is. The measurable objectives (O1-O5) associated with this aim are as follows:

- (O1) Research into the pricing of options
- (O2) Research Boyarchenko and Levendorskii's new NIZT method (Sinh-Deformation)
- (O3) Analyse and implement the existing NIZT methods in literature
- (O4) Test the implementations against well-defined transform pairs
- (O5) Compute metrics on parameter setups for each method

1.3 Overview

The upcoming chapters of this report provide a detailed description of the different stages taken through this project. Chapter 2 provides a self-contained technical background on the key concepts and methods used in this project. Given the breadth of the topic, references are included for those looking to delve deeper into the subject. In Chapter 3, we outline the experiments conducted in Chapter 4 to evaluate the performance of the numerical approximation methods. We detail the implementation of the methods and the experimental setup, including parameter selection and the benchmarking process. We then provide the results on a list of transform pairs and discuss the implications of the findings. Finally, Chapter 5 summarises the steps taken and the findings of this project with suggested work for further continuation.

Chapter 2

Background

In Chapter 2, we establish a foundational understanding of the topic in hand. This section is designed to be self-contained, providing essential background for all readers, while references are included for those seeking a deeper exploration.

We conduct an analysis of the mathematical concepts used in option pricing, with a focus on Fourier-based methods. While a large body of theoretical work exists in this area, the practical implementation of these methods is often computationally expensive, especially when evaluating high-dimensional integrals involved in the pricing formulas. We study literature on numerical approximation methods and their application in the pricing of exotic options, specifically lookback and barrier options. Subsequently, we explore the use-case of machine learning techniques to optimise parameters during our experimentation in Chapter 3.

2.1 The $\mathcal{Z}$-Transform

The z-transform is a transformation of a real or complex time function $f(n)$, often used for analysing discrete-time signals and systems. It is a generalisation of the discrete-time Fourier transform (DTFT) that extends the analysis to the complex plane. The bilateral Z-transform is formally defined as:

$$\tilde{f}(z) = \mathcal{Z}_{n\to z}[f(n)] = \sum_{n=-\infty}^{\infty} f(n)z^{-n} \tag{2.1}$$

"By definition, a complex number z is an ordered pair (x, y) of real numbers x and y, written $z = (x, y)$" (Kreyszig, 2010). In practice, complex numbers are written in the form $z = x + iy$, where x and y are real numbers and i is the imaginary unit. We may find it easier to represent complex numbers in their polar form, $z = re^{i\theta}$, where r represents the magnitude of z and θ represents the angle of z with respect to the positive real axis.

In the analysis of causal systems - systems for which a time origin is defined and it is illogical to consider signal values for negative time - the unilateral z-transform is used. Unlike the bilateral z-transform in Eq. (2.1), we sum from zero to positive infinity, yielding:

$$\tilde{f}(z) = \mathcal{Z}_{n\to z}[f(n)] = \sum_{n=0}^{\infty} f(n)z^{-n} \tag{2.2}$$

The region within the complex z-plane where the z-transform converges is known as the Region of Convergence (ROC). The ROC is defined for the set of values of z for which the z-transform is

absolutely summable:

$$\mathbf{ROC} = \left\{ z : \sum_{n=0}^{\infty} |f(n)z^{-n}| < \infty \right\} \tag{2.3}$$

For causal sequences, the ROC is typically the exterior of the outermost pole in the Z-plane, denoted as $|z| > a$. If we say that z_1 converges, then z_1 exists within the ROC. Thus, all z such that $|z| \geq |z_1|$ also converge. This region excludes the poles, as the transform does not converge at those points. For the system to be *stable*, the ROC must include the unit circle, $|z| = 1$, implying that all poles must lie within the unit circle (Loveless and Germano, 2021).

Example 1 *Consider the function given by*

$$H(z) = \frac{(z-i)(z+i)}{\left(z - \left(-\frac{1}{4} - \frac{1}{2}i\right)\right)\left(z - \left(\frac{1}{2} + \frac{1}{2}i\right)\right)} \tag{2.4}$$

The ROC of $H(z)$ must exclude the poles at $z = -\frac{1}{4} - \frac{1}{2}i$ and $z = \frac{1}{2} + \frac{1}{2}i$, thus the ROC is $|z| > \frac{1}{2}$. The system is stable as the ROC includes the unit circle.

2.1.1 Relation to the Fourier Transform

It is useful to note the relationship between the z-transform and the Fourier transform. Taking the Fourier transform of a sampled function $f(t)$ results in:

$$\mathcal{F}\left[f(t) \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t)\right] = \int_{-\infty}^{\infty} f(t) \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t) e^{-i\omega t} dt \tag{2.5}$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t)\delta(t - n\Delta t) e^{-i\omega t} dt \tag{2.6}$$

$$= \sum_{n=-\infty}^{\infty} f(n\Delta t) e^{-i\omega n t} \tag{2.7}$$

where we make use of the sifting property of the delta function. If we normalise the sampling interval to 1, we get:

$$\sum_{n=-\infty}^{\infty} f(n) e^{-in\omega} \tag{2.8}$$

This is the discrete-time Fourier transform (DTFT) of the sequence $f(n)$. The sequence $f(n)$ is sampled at discrete-time intervals $t_n = n\triangle t$, where the sampling interval $\triangle t$ is the time between consecutive samples and the time index n numbers the samples. The DTFT is a periodic function of ω with period 2π, and its existence relies on the absolute summability of the sequence $f(n)$:

$$\sum_{n=-\infty}^{\infty} |f(n)| < \infty \tag{2.9}$$

The Z-transform generalises Eq. (2.8) to the complex plane, not just the unit circle where $r = 1$ (Schafer and Oppenheim, 1989).

2.1.2 Relation to Probability Distribution Functions

Random events and signals refer to situations where the outcome is not deterministic, but can be described by probability. Understanding these concepts involves using Probability Distribution

Functions; the Probability Mass Function (PMF) for discrete random variables and the Probability Density Function (PDF) for continuous random variables. Given the nature of this project, we'll be focusing on the PMF.

The PMF is defined for a discrete random variable X taking on values x_i with probabilities p_i, as $P(X = x_i) = p_i$. The PMF satisfies the following properties:

$$\sum_{i=0}^{n} p_i = 1 \qquad \text{and} \qquad 0 \leq p_i \leq 1 \ \ \forall i \tag{2.10}$$

We may find it useful to provide a concise representation of the entire distribution such that we expand upon the PMF, $p(x)$, to obtain the Probability Generating Function (PGF), $G_X(q)$, defined as:

$$G_X(q) = E[q^X] = \sum_{x=0}^{\infty} p(x)q^x, \tag{2.11}$$

where $E[\cdot]$ denotes the expectation operator, and q is a complex number. We deliberately use q to distinguish the PGF from the z-transform (Eq. 2.2).

Example 2 *Consider a fair six-sided dice. The PMF for the dice roll is given by*

$$p(x) = \begin{cases} \frac{1}{6} & \text{if } x = 1, 2, 3, 4, 5, 6 \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

where $p(x)$ is the probability of rolling a number x. The PGF for the dice roll is then

$$G_X(q) = \sum_{x=0}^{\infty} p(x)q^x = \frac{1}{6}\sum_{x=1}^{6} q^x = \frac{q}{6} \cdot \frac{1 - q^6}{1 - q}, \tag{2.13}$$

where we use the formula for the sum of a geometric series. The PGF encapsulates the entire distribution of the dice roll into a single function.

The concept of summarising information is not unique to probability theory. In the analysis of signals, we aim to encapsulate the behaviour of a sequence into a single function. This is akin to the PGF, where the z-transform is used to analyse discrete-time signals and systems. Drawing on the principles outlined by Ross (2014), we can bridge the gap between probability theory and signal processing, leveraging the z-transform to analyse the behaviour of signals in the complex plane.

2.2 The Inverse $\mathcal{Z}$-Transform

The inverse Z-transform aims to retrieve the sequence $f(n)$ from its Z-transform $\tilde{f}(z)$. While this transform is typically defined for finding individual values of n, it can also be extended to calculate the entire sequence. This is commonly defined by the Cauchy integral around a contour C in the complex plane. The chosen contour C is a counter-clockwise closed path that encloses the region of convergence (ROC) of $\tilde{f}(z)$. Formally, the inverse Z-transform is given by

$$f(n) = \mathcal{Z}^{-1}_{z \to n}[\tilde{f}(z)] = \frac{1}{2\pi i} \oint_C \tilde{f}(z) z^{n-1} dz \tag{2.14}$$

The integral approach enables the extraction of the complete sequence $f(n)$ for all integer values

n, provided the ROC and the properties of $\tilde{f}(z)$ allow. In practical settings, numerical approximations are often used due to the computational challenges of evaluating the Cauchy integral. Whilst there are many ways to go about this (Merrikh-Bayat, 2014; Rajković et al., 2004; Horváth et al., 2020), most methods are done on a circular contour. Aligning with the focus of our project, we shift our attention to contour integrals.

2.2.1 Abate and Whitt 1992

The numerical approximation formula offered by Abate and Whitt (1992a,b) is based on a Fourier series catering to the inversion of probability generating functions as elucidated in Section 2.1.2. The format is conducive to queuing theory and other probabilistic models where the Z-transform is defined as $q = 1/z$. The authors approximate the inversion using a trapezoidal rule for numerical integration over a complex contour given by

$$f(n) \approx \frac{1}{2nr^n}\left(\tilde{f}(r) + 2\sum_{k=1}^{n-1}(-1)^k \mathrm{Re}\left(\tilde{f}(re^{\frac{ik\pi}{n}})\right) + (-1)^n \tilde{f}(-r)\right) \tag{2.15}$$

The parameter r is used to control the error; setting $r = 10^{-\lambda/2n}$ yields an accuracy bound of $10^{-\lambda}$. The authors leverage the inherent symmetry within the complex plane to enhance computational efficiency by exploiting the complex conjugate symmetry of $\tilde{f}(z)$; each term $\tilde{f}(re^{\frac{ik\pi}{n}})$ in the upper half has a mirror image in the lower half. The computational load is thus halved by *folding* the problem in this manner.

Given the nature of this project, we may find it easier to use the following definition, where we set the input of $\tilde{f}(z)$ to that of $z = 1/q$, to approximate Eq. (2.14).

$$f(n) \approx \frac{1}{2nr^n}\left(\tilde{f}(\frac{1}{r}) + 2\sum_{k=1}^{n-1}(-1)^k \mathrm{Re}\left(\tilde{f}(\frac{1}{re^{\frac{ik\pi}{n}}})\right) + (-1)^n \tilde{f}(-\frac{1}{r})\right) \tag{2.16}$$

The Nyquist-Shannon sampling theorem states that a signal must be sampled at a rate of at least twice the highest frequency present in the signal to avoid aliasing (Shannon, 1949; Nyquist, 1928). The number of points, n, used in Eq. (2.16) must be sufficient to capture the information properly. If n is too small, the approximation may lead to inaccuracies - akin to aliasing in signal processing.

2.2.2 Cavers 1978

Building on our analysis in Section 2.1.1, Cavers (1978) proposes sampling the z-transform of a function on a circular contour at equally spaced points and then applying the inverse fast Fourier transform (IFFT) to these sampled points to approximate the original time-domain signal. The DTFT suffers from a high computational cost with a time complexity of $\mathrm{O(N^2)}$ given the need to perform O(N) operations for each of the N different points. Thus, the author employ a more efficient algorithm by Cooley and Tukey (1965) to what is known as the fast Fourier transform (FFT), with a time complexity of O(NlogN). We can formulate this as:

$$f(n) = r^n \mathrm{IFFT}[\tilde{f}(re^{2\pi ik/N})], \tag{2.17}$$

which can be rewritten into the following:

$$f(n) = \frac{r^n}{N}\mathrm{FFT}[\tilde{f}(re^{2\pi ik/N})], \tag{2.18}$$

where r denotes the radius of the contour, n is the time index, and N is the number of points within the integration grid. The factor r^n scales the result appropriately based on the radius of the contour. Recent studies, such as those by Loveless and Germano (2021), show that increasing N results in an error tolerance approaching E-16, *i.e. machine zero*, with double precision; note that this requires some optimisation of the chosen parameters.

Loveless et al. (2023) explores replacing the IFFT of Eq. (2.17) with a numerical summation over the grind points on half of the unit circle, similar to that of Abate and Whitt (1992a,b)'s implementation. This is given as

$$f(n) = r^n \sum_{k=1}^{N} \tilde{f}(re^{-\pi ik/N}) \tag{2.19}$$

The authors conclude that both implementations achieve machine-level accuracy of E-16 with some slight deviations and requiring an optimal hyperparameter setup.

2.3 Optimisation Techniques

In the context of computational mathematics, optimisations techniques are used to identify the optimal or a sufficiently effective solution to a problem within a given set of constraints. The goal is to minimise or maximise a specific objective function by systematically choosing the values of the variables. The objective function is often referred to as the *cost function* or *loss function* and the variables are referred to as *parameters*. The optimisation problem can be formulated as

$$\text{minimise } f(x) \text{ subject to } x \in \Omega, \tag{2.20}$$

where $f(x)$ is the objective function and Ω is the feasible region defined by the constraints of the problem.

Gradient descent is one of the most popular algorithms for parameter optimisation with success in Deep Learning and Neural Networks employing variants of the algorithm (Lu and Jin, 2017; Zhang, 2019; Zeebaree et al., 2019). The adaptability to diverse problem domains (Tian et al., 2023) parallels our use case, where gradient descent is applied outside traditional deep learning to optimise parameters of a mathematical function (Persson et al., 2022).

2.3.1 Gradient Descent

Gradient descent iteratively converges to a local minimum of a function by moving in the direction of the steepest descent, as defined by the negative gradient. This method is expressed mathematically as

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{2.21}$$

where x_k is the parameter vector at iteration k, α_k is the learning rate, and $\nabla f(k)$ represents the gradient of the function at x_k. The selection of α_k determines the size of the step taken towards the minimum; too large can overshoot the minimum, too small can result in a long convergence time. The process repeats until a predetermined termination criterion is met, typically when the change in the value of $f(k)$ falls below a threshold. This iterative process is showcased in the pseudocode below:

Algorithm 1 Gradient Descent

Initialise x_0, set $k = 0$
while termination conditions not met **do**
    $g_k = \nabla f(x_k)$ ▷ Compute Gradient
    $\alpha_k \to \mathbb{R}^+$
    $x_{k+1} = x_k - \alpha_k g_k$
    $k = k + 1$
end while

Stochastic Gradient Descent

However, classic Gradient Descent faces limitations, including susceptibility to local minima. Stochastic Gradient Descent (SGD) addresses this by introducing variability in the optimisation process. It modifies Eq. (2.21) to use a randomly selected subset of data to compute the gradient, leveraging noise to escape local minima. We define the update rule to

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \tag{2.22}$$

where $\nabla f_{i_k}(x_k)$ is the gradient of the cost function with respect to a random subset i_k.

ADAM Optimiser

ADAM (Adaptive Moment Estimation) is a popular optimisation algorithm introduced by Kingma and Ba (2014) that combines the advantages of two extensions to the stochastic gradient descent method: AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman and Hinton, 2012). ADAM computes adaptive learning rates for each parameter by estimating the first and second moments of the gradients. This makes the method particularly well-suited for large-scale and complex optimisation problems, such as those countered in deep learning and computational optimisation. It's ability to adjust learning rates based on gradient information allows ADAM to efficiently navigate the parameter space and converge to extremely precise solutions (Reddi et al., 2019). The update rule is given by:

$$x_{k+1} = x_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}, \tag{2.23}$$

where first moment estimate m_k and the second moment estimate v_k are computed as exponential moving averages of the gradient and the squared gradient, respectively. The hyperparameters β_1 and β_2 control the decay rates of these moving averages. To correct for initialisation bias, the estimates of the first and second moments ($\hat{m}_k$and $\hat{v}_k$) are bias-corrected. The parameter update is performed using the adapted learning rate α and the bias-corrected estimates of the moments. We add ϵ to the denominator to prevent division be zero; typically set to 10^{-8}. This process is shown in the pseudocode below:

Algorithm 2 ADAM Optimiser

Initialise x_0, $m_0 = 0$, $v_0 = 0$, set $k = 0$
Hyperparameters: α, β_1, β_2, ϵ
while termination conditions not met **do**
    $k = k + 1$
    $g_k = \nabla f(x_k)$ ▷ Compute Gradient
    $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
    $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
    $\hat{m}_k = \frac{m_k}{1-\beta_1^k}$ ▷ Bias-Corrected First Moment
    $\hat{v}_k = \frac{v_k}{1-\beta_2^k}$ ▷ Bias-Corrected Second Moment
    $x_{k+1} = x_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$
end while

2.4 Option Pricing

The concept involves determining the value of options, which are financial contracts that give the holder the right, *but not the obligation*, to buy or sell an asset at a set price, K (strike price), within a specified time-frame, T (expiration date). The value of an option, V, is derived from the underlying asset, denoted as S, which can be a stock, bond, or commodity. This relationship can be mathematically expressed as:

$$V = f(S, K, T, \sigma, r), \tag{2.24}$$

where σ represents the volatility of the underlying asset[1] and r is the risk-free interest rate[2].

A pivotal point in option pricing was the introduction of Black and Scholes (1973)'s model in estimating the price of European-style options, which can only be exercised at the expiration date. The Black-Scholes model is based on the assumption that the price of the underlying asset follows a geometric Brownian motion in idealistic conditions. However, many options traded in real markets are American-style, allowing the holder to exercise the option at any time before the expiration date. This complicates the pricing process as it involves solving an *optimum stopping problem*. Merton (1973) extends the Black-Scholes model to American options by expressing the price as the solution to a free boundary problem. While Merton's work provided a theoretical foundation, solving the free boundary problem analytically is challenging. Instead, numerical methods such as binomial trees (Cox et al., 1979) and simulation-based methods (Longstaff and Schwartz, 2001) have been developed to price American options.

2.4.1 Path-Dependent Options

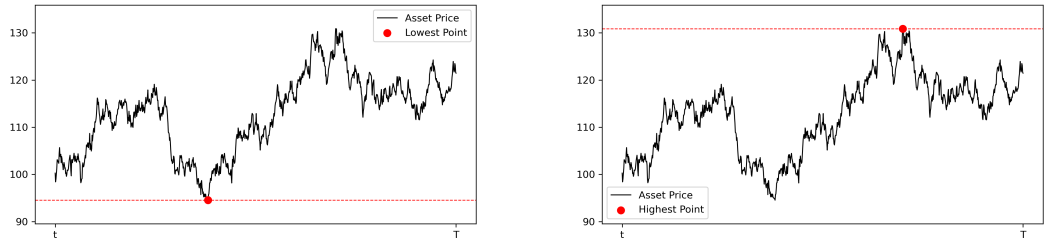
We regard American and European options as standard (*vanilla*) options. In contrast, exotic options offer more complex features and conditions that can be tailored to meet specific needs and/or requirements. Such differences are the reason as to why exotic options are traded over-the-counter (OTC)[3], whereas standard options are typically traded on organised exchanges[4].

In most cases, the payoff of an option depends on the price of the underlying asset at discrete points in time rather than continuously. This is known as **discrete monitoring.** Two widely

[1] A measure of how much the price of the asset is expected to fluctuate over a certain period of time.
[2] The interest rate at which an invest can lend money with virtually no risk of loss.
[3] Over-The-Counter refers to the process of trading of financial instruments directly between two parties without the oversight of a formal exchange.
[4] Organised exchanges refer to formal marketplaces that facilitate the buying and selling of financial instruments.

traded types of discretely monitored options are lookback and barrier options (Dadachanji, 2015). These options are classified as *path-dependent options*, within the category of exotic options, where the payoff is based on the path of the underlying asset price rather than just the price at expiration.

Lookback Options



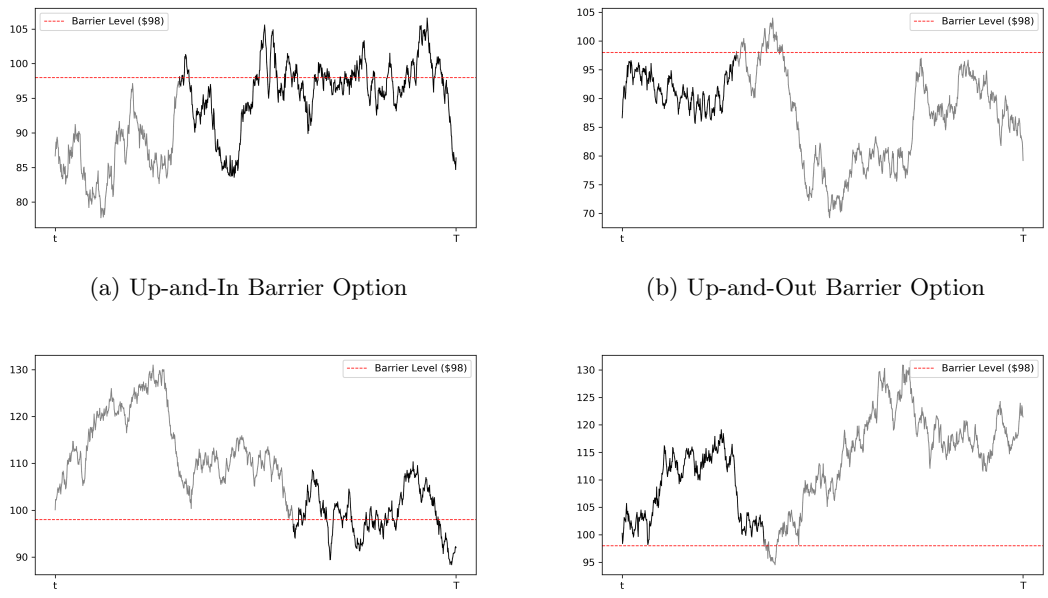
(a) Lookback Call Option

(b) Lookback Put Option

Figure 2.1: Illustration of Lookback Options: Displaying the mechanics of (a) Call Option and (b) Put Option. The red line represents the barrier, while the red dot marks the strike price.

The payoff depends on the maximum and minimum asset price over the life of the option. A *lookback call option* (Figure 2.1a) gives the holder the right to buy the asset at the lowest price during the option period, while a *lookback put option* (Figure 2.1b) allows the holder to sell the asset at the highest price during the option period.

Barrier Options



(a) Up-and-In Barrier Option

(b) Up-and-Out Barrier Option

(c) Down-and-In Barrier Option

(d) Down-and-Out Barrier Option

Figure 2.2: Illustration of Barrier Options: Displaying the active (black) and inactive (grey) phases of each option type. (a) Up-and-In Option, (b) Up-and-Out Option (c) Down-and-In Option, and (d) Down-and-Out Option. The red line represents the barrier.

The payoff depends on whether the price of the underlying asset reaches a certain level (the barrier) during the life of the option. A *knock-in* barrier option only comes into existence if the barrier has been touched, while a *knock-out* barrier option ceases to exist instead. For example, a *down-and-out* barrier option (Figure 2.2d) is a type of knock-out option that becomes worthless if the price of the underlying asset falls below the barrier level. On the other hand, a *down-and-in* barrier option (Figure 2.2c) is a type of knock-in option that only becomes active if the asset price falls below the barrier level.

Double Barrier Options

Double barrier option involve two price levels: an upper and lower barrier that define the boundaries for the option's activity. These options can either be *knock-in* or *knock-out*. A double knock-in option becomes active only if the asset price hits either barrier, similar to a combination of Figure 2.2a and Figure 2.2c. Conversely, a double knock-out option ceases to exist if the asset price touches either barrier combining the characteristics of Figure 2.2b and Figure 2.2d.

2.4.2 Modelling Asset Prices

Accurate models allow traders, investors and risk managers to assess the fair value of financial products and make informed decisions. The Black-Scholes model assumes that the price of the underlying asset follows a geometric Brownian motion (GBM). Under this assumption, the asset price $S(t)$ evolves according to the stochastic differential equation:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t), \tag{2.25}$$

where μ represents the expected return, and $W(t)$ is a standard Wiener process. The solution to this equation gives the asset price at time t as:

$$S(t) = S(0)e^{(\mu - 0.5\sigma^2) + \sigma W(t)} \tag{2.26}$$

The model assumes that the volatility is constant, which is rarely the case as market volatility tends to fluctuate over time due to economics conditions or market sentiment. Additionally, the model doesn't account for jumps or sudden large movements in asset prices, usually seen at times of significant news. These limitations have led to the development of more sophisticated representations; given the scope and background of this project, we primarily focus on exponential Lévy processes.

Exponential Lévy Processes

We extend Eq. (2.25) by incorporating jump processes to account for sudden changes in asset prices. A stochastic process $X(t)$ is termed a Lévy process if **(i)** the changes in the process are independent of each other, **(ii)** the distribution of the changes in the process depends only on the length of the time interval, not on the starting point, and **(iii)** it starts at zero, *i.e.*, $X(0) = 0$. The dynamics of a Lévy process can be described by the following stochastic differential equation:

$$dX(t) = \mu dt + \sigma dW(t) + dJ(t), \tag{2.27}$$

where $J(t)$ represents a jump process (typically a compound Poisson process) to account for sudden, significant changes. An exponential Lévy process describes the asset price process as:

$$S(t) = S(0)e^{X(t)} \tag{2.28}$$

As the characteristic function of exponential Lévy models are often available in closed form, which facilitates the application of Fourier-based methods, they have been widely used in the pricing of path-dependent options as shown by the works of Fusai et al. (2016); Kwok et al. (2011); Feng and Linetsky (2008); Phelan et al. (2019).

2.4.3 Fourier-Based Pricing

The pricing of these options can be computationally expensive due to the high-dimensional integrals involved in the pricing formulas. Whilst Heston (1993) explored the idea of modelling asset prices in the Fourier domain, Carr and Madan (1999) were the first to price European options expressing both the characteristic function and the payoff in the Fourier domain. It involved transforming the pricing problem from the time domain to the frequency domain, where the pricing formula simplifies to a one-dimensional integral. The Fourier-based methods have been widely used for the pricing of options discussed in Section 2.4.1 (Eberlein et al., 2010).

Fang and Oosterlee (2009a) introduced a pricing technique based on the Fourier-cosine expansion approximating the characteristic function of the underlying asset price, which was then expanded to a broader class of exotic options (Fang and Oosterlee, 2009b, 2011) and general lévy processes (Lord et al., 2008). Whilst the method shows high levels of accuracy with exponential error convergence on the number of terms, this is only the case when the governing probability density function is sufficiently smooth. When modelling the asset price by the variance gamma process (Madan et al., 1998), we achieve only algebraic error convergence due to the discontinuities in the process. Ruijter et al. (2015) showed that this can be remedied using spectral filters however the issue remains in that the computational time is linearly dependent on the number of monitoring dates for discretely monitored options. This seems to also be the case for Feng and Linetsky (2008)'s employment of the Hilbert transform using backward induction to price barrier options.

In an effort to eliminate the computational inefficiency tied to the number of monitoring dates, Fusai et al. (2006) looked into shifting the pricing problems from the time domain into the z-domain, where convolution operations become straightforward multiplications. This helped set the stage for Fusai et al. (2016)'s method incorporating the Hilbert and z-transform to calculate the Wiener-Hopf factors, which decompose the characteristic function of the price process into two parts; positive and negative jumps. The authors demonstrated that the computational cost is independent of the number of monitoring dates, and the error decays exponentially with the number of grid points. Whilst extensions to the method have been made such as those by Phelan (2018)'s new scheme using spectral filters to improve convergence (Phelan et al., 2019), we shift our focus to the use-case of the inverse z-transform in the pricing of options.

2.4.4 NIZT in Option Pricing

We consider the case that of a double barrier option (Section 2.4.1), which are characterised by their dependency on the asset's price staying within a specified range between an upper boundary u and a lower boundary l. The probability of the asset's price being at x at the nth monitoring

without hitting these barriers is represented by $f(x, n)$. This forms the basis of our recursive relationship for these options:

$$\int_l^u f(x', n-1)p(x - x', \triangle t)\ dx', \tag{2.29}$$

where $p(x - x', \triangle t)$ is the transition density between the states from x' to x over a small time increment $\triangle t$. The integral expression represents the transition probabilities convolved with the state probabilities from the previous time step, under the condition that the asset remains within the barriers u and l. We expand this relationship across all monitoring periods as follows:

$$\sum_{n=1}^{\infty} q^n f(x, n) = \sum_{n=1}^{\infty} q^n \int_l^u f(x', n-1)p(x - x', \triangle t)\ dx' \tag{2.30}$$

We notice the similarity to that of Eq. (2.2), however we set $z^{-1} = q$ to remain consistent with that in literature. By shifting the index $m = n - 1$, the expression simplifies to

$$= q \int_l^u \sum_{m=0}^{\infty} q^m f(x', m)p(x - x', \triangle t)\ dx' \tag{2.31}$$

This manipulation restructures the series into a form amenable to the use of transitioning to the z-domain. We define the z-transform of $f(x', n)$ as

$$\tilde{f}(x', q) = \mathcal{Z}_{n\to q}[f(x', n)] = \sum_{n=0}^{\infty} q^n f(x', n) \tag{2.32}$$

This encapsulation into $\tilde{f}(x', q)$ allows us to write the relationship as

$$\sum_{n=0}^{\infty} q^n f(x, n) - q^0 f(x, 0) = q \int_l^u \sum_{m=0}^{\infty} q^m f(x', m)p(x - x', \triangle t)\ dx' \tag{2.33}$$

to which we can see how the z-transform helps in handling the summation over all n:

$$\tilde{f}(x', q) = q \int_l^u \tilde{f}(x', q)p(x - x', \triangle t)\ dx' + f(x, 0) \tag{2.34}$$

Such a form enables the use of fourier-based methods; Fusai et al. (2016) demonstrated that the iteration over all monitoring dates can be bypassed using the Spitzer identities, originally detailed by Spitzer (1957) and expanded upon by Kemperman (1963). The identities require the Wiener-Hopf factorisations and decompositions, which were computed with the Plemelj-Sokhotsky relations and the Hilbert transform. Whilst Fusai et al. (2016) provide a comprehensive list of references, we ought to also mention the continuations of Phelan et al. (2018, 2019); Loveless et al. (2023). After such calculations, the inverse z-transform (Section 2.2) of $\tilde{f}(x, q)$ is needed to translate the representation back to the time-domain probabilities. Thus, the focus of this work on the inverse z-transform not only underscores its mathematical importance but also highlights its role in bridging theoretical constructs with real-world financial applications.

Chapter 3

Experiment

Having established the theoretical foundation, we now turn our attention to the practical implementation of methods proposed by Abate and Whitt (1992a,b) and Cavers (1978). Whilst the former is based on a circular contour, we also explore the idea laid out by Boyarchenko and Levendorskiĭ (2022) in sampling the z-transform on a sinh-deformation.

This chapter outlines the experimental setup, implementation and evaluation of the methods. We measure our results in terms of accuracy and computational efficiency against well-known transformation pairs (Table 3.1) in an attempt to provide a thorough comparison of the methods.

All experiments were designed to be used on Visual Studio Code 1.89.1 using Python 3.11.7, NumPy 1.26.3 and MatPlotLib 3.8.0; we use such an environment given its robustness and extensive support for scientific computing and optimisation tasks. Whilst we outline all experiments/methods as pseudocode to allow for easy adaptations to other computing environments, the python implementation of the key functions are provided in Appendix A.

3.1 Transform Pairs

To measure the *success* of our implementations, we find it convenient to use analytical transform pairs for the unilateral z-transform. Whilst a wide array of transform pairs exist, we focus on those listed in Table 3.1 as they are "simple to compute and provide an excellent benchmark" for our use case (Loveless and Germano, 2021). For each function $f(t)$, we derive the corresponding transform pair in the subsequent sections.

Function	$f(t)$	$\tilde{f}(z)$
Heaviside Step	1	$\frac{z}{z-1}$
Polynomial	t	$\frac{z}{(z-1)^2}$
Decaying Exp	e^{-at}	$\frac{z}{z-e^{-a\triangle t}}$
Sinusoidal	$\sin(\omega t)$	$\frac{z^{-1}\sin(\omega\triangle t)}{1-2\cos(\omega\triangle t)z^{-1}+z^{-2}}$

Table 3.1: List of Transform Pairs

3.1.1 Heaviside Step

The Heaviside step function, $f(t)$, is an important piecewise function in signal processing and control theory (Proakis, 2007). This is commonly defined as

$$f(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 & \text{if } t \geq 0 \end{cases} \tag{3.1}$$

The unilateral z-transform of the Heaviside step function can be calculated using Eq. (2.2):

$$\tilde{f}(z) = \mathcal{Z}_{t\to z}[f(t)] = \sum_{t=0}^{\infty} f(t) z^{-t} = \sum_{t=0}^{\infty} z^{-t} \tag{3.2}$$

We identify this as a geometric series where each term has the form r^n with $r = z^{-1}$ and $n = t$. The sum of an infinite geometric series with first term a and a common ratio r, where $|r| < 1$ is given by

$$S_\infty = \frac{a}{1-r} \tag{3.3}$$

For the sum to converge, we require $|z > 1|$ given the pole at $z = 1$, and it follows that

$$\tilde{f}(z) = \frac{1}{1 - z^{-1}} = \frac{z}{z-1} \tag{3.4}$$

3.1.2 Polynomial

The polynomial function, often referred to as the linear ramp function, is defined as $f(t) = t$ for $t \geq 0$. Following a similar approach to that of the previous section, we have

$$\tilde{f}(z) = \mathcal{Z}_{t\to z}[f(t)] = \sum_{t=0}^{\infty} f(t) z^{-t} = \sum_{t=0}^{\infty} t z^{-t}, \tag{3.5}$$

in which we take note of the increasing integer multiplier t and the kernel z^{-t}. To solve this series, we use the derivative property of the z-transform given by

$$\mathcal{Z}_{t\to z}[tf(t)] = -z \frac{d\tilde{f}(z)}{dz}, \tag{3.6}$$

which shows that multiplying the sequence $f(t)$ by t in the time domain corresponds to multiplying the z-transform $\tilde{f}(z)$ by $-z$ and taking the derivative with respect to z in the z-domain. Thus, if we consider the Heaviside step function, it leads on that

$$\tilde{f}(z) = -z \frac{d}{dz}\left(\frac{z}{z-1}\right) = -z\left(\frac{z-1-z}{(z-1)^2}\right) = \frac{z}{(z-1)^2} \tag{3.7}$$

3.1.3 Decaying Exponential

The decaying exponential function, commonly expressed as $f(t) = e^{-at}$ for $t \geq 0$ and $a > 0$, is very useful in the analysis of systems with exponential decay or growth characteristics. However, such a signal is sampled at discrete time intervals; if $\triangle t$ is the sampling interval and n represents the discrete time index, then the continuous function $f(t)$ becomes $f[n] = e^{-an\triangle t}$ when sampled

at intervals of $\triangle t$. The unilateral z-transform of $f[n]$ is given by

$$\tilde{f}(z) = \mathcal{Z}_{n\to z}[f[n]] = \sum_{n=0}^{\infty} f[n]z^{-n} = \sum_{n=0}^{\infty} e^{-an\triangle t}z^{-n} \tag{3.8}$$

This is a geometric series where each term is $(e^{-a\triangle t}z^{-1})^n$. Using Eq. (3.3), we can express $\tilde{f}(z)$ as:

$$\tilde{f}(z) = \frac{1}{1-(e^{-a\triangle t}z^{-1})} = \frac{z}{(z-e^{-a\triangle t})} \tag{3.9}$$

3.1.4 Sinusoidal

The sinusoidal function $f(t) = \sin(\omega t)$ is a key waveform given its periodic nature. Similar to that of the decaying exponential transform pair, we discretely sample the function at intervals $\triangle t$, which results in the discrete time function $x[n] = \sin(\omega n\triangle t)$. Thus, the unilateral z-transform is computed as

$$\tilde{f}(z) = \mathcal{Z}_{n\to z}[f[n]] = \sum_{n=0}^{\infty} f[n]z^{-n} = \sum_{n=0}^{\infty} \sin(\omega n\triangle t)z^{-n} \tag{3.10}$$

We make use of Euler's formula, defined as $\sin(x) = \frac{e^{ix}-e^{-ix}}{2i}$, to express our sine function and split the term into two separate geometric series:

$$\tilde{f}(z) = \sum_{n=0}^{\infty}\left(\frac{e^{i\omega n\triangle t} - e^{-i\omega n\triangle t}}{2i}\right)z^{-n} = \frac{1}{2i}\cdot\left(\sum_{n=0}^{\infty} e^{i\omega n\triangle t}z^{-n} - \sum_{n=0}^{\infty} e^{-i\omega n\triangle t}z^{-n}\right) \tag{3.11}$$

We then refer back to Eq. (3.3) to express the summations as functions of z. It follows on that

$$\tilde{f}(z) = \frac{1}{2i}\cdot\left(\frac{1}{1-e^{i\omega\triangle t}z^{-1}} - \frac{1}{1-e^{-i\omega\triangle t}z^{-1}}\right) \tag{3.12}$$

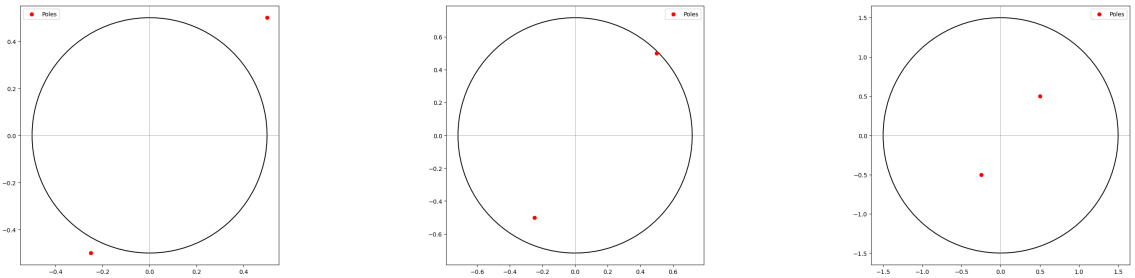
where we simplify it into the form:

$$\tilde{f}(z) = \frac{1}{2i}\cdot\frac{-z^{-1}(e^{i\omega\triangle t} - e^{i\omega\triangle t})}{(1-e^{i\omega\triangle t}z^{-1})(1-e^{-i\omega\triangle t}z^{-1})} = \frac{z^{-1}\sin(\omega\triangle t)}{1-2\cos(\omega\triangle t)z^{-1}+z^{-2}} \tag{3.13}$$

3.2 Circular Contour

"The inverse at point T can be obtained from the contour integral where C is a counter-clockwise closed path encircling the origin and entirely in the region of convergence" (Horváth et al., 2020). A circular contour is commonly used to solve Eq. (2.14), defined as $z = re^{i\theta}$, however the setting of r is important as shown for Example 1 in Figure 3.1.

We thus formulate an algorithmic approach to the implementations of the NIZT outlined in Section 2.2.1 and Section 2.2.2. For each of the methods, we outline a series of experiments and validation tests to carry out.



(a) r = 0.5 (b) r = 0.72 (2 d.p.) (c) r = 1.5

Figure 3.1: Displaying the effect of the radius r on a circular contour ($n = 100$). The red dots represent the poles of $H(z)$ in Example 1 with (a) including the poles within the ROC, (b) excluding the poles but inclusive of the unit circle, and (c) excluding the poles but not stable.

3.2.1 Abate and Whitt 1992

Given a sufficient hyperparameter setup, we will require, as input, just the function to be inverted, $\tilde{f} : \mathbb{C} \to \mathbb{C}$, and the point $n_{\in}\mathbb{R}$. The implementation of such a function follows very easily from Eq. (2.16) as shown in the pseudocode below:

Algorithm 3 Implementation of Equation 2.16

procedure AbateWhitt($\tilde{f}, n$)
  $\lambda \leftarrow \mathbb{R}^+$
  $r \leftarrow 10^{-\lambda/(2 \cdot n)}$
  summation $\leftarrow 0$
  for $k \leftarrow 1$ **to** n **do**
    $z \leftarrow \frac{1}{r \cdot \exp(\mathrm{i} \cdot k \cdot \pi / n)}$
    sum $\leftarrow$ sum $+ (-1)^k \cdot \mathrm{Re}(\tilde{f}(z))$
  end for
  sum $\leftarrow \tilde{f}(\frac{1}{r}) + 2 \cdot$ sum $+ (-1)^n \cdot \tilde{f}(-\frac{1}{r})$
  return $\frac{\text{sum}}{2 \cdot n \cdot r^n}$
end procedure

Computational Accuracy

Abate and Whitt (1992a,b) showed that the absolute error, $|\epsilon|$, of their implementation is bounded by

$$|\epsilon| = \frac{1}{r^N - 1} \approx \frac{1}{r^N}, \tag{3.14}$$

where the approximation holds for large r^N. As discussed in Section 2.2.1, setting $r = 10^{-\lambda/2n}$ yields an accuracy of $10^{-\lambda}$, *i.e.*, λ significant digits. However, we have that if the machine accuracy is $\epsilon_m = 10^{-\lambda_m}$, the bound falls apart around $\lambda > \frac{2}{3}\lambda_m$ (Abate and Whitt, 1992a, Remark 5.8). Loveless et al. (2023) states that "with IEEE 754 double precision representation of floating point numbers, $\epsilon_m = 2^{-52} \approx 10^{-15.65}$, and thus it makes little sense to set $\lambda > 10.5$, which yields the IZT up to about 10 or 11 significant digits".

Thus, we investigate the effect of different values of λ on the accuracy of the Abate-Whitt method when applied to the Heaviside step and polynomial transform pairs (Table 3.1). We expect to observe an increase in accuracy as λ increases, up to a certain point where the limitations of machine precision and rounding errors become dominant.

Computational Speed

In addition to accuracy, computational efficiency is another important factor to consider. The Abate-Whitt algorithm described in Section 2.2.1 has a time complexity of $\mathrm{O}(n)$, where n is the number of terms used in the approximation. This is evident in Algorithm 3 for the inclusion of the single for loop. With each iteration, the dominant operation is the evaluation of the input function $\tilde{f}$. Assuming the function evaluation time is bounded by some constant, the overall time complex remains $\mathrm{O}(n)$.

We thus run tests to empirically verify this linear complexity. Plotting the computation time against n should show an approximately linear relationship if the $\mathrm{O}(n)$ analysis holds in practice. Practically, it is not feasible to eliminate any interference from external background processes. However, we make efforts to minimise such effects by executing multiple runs of each test and taking the average.

3.2.2 Cavers 1978

As discussed in Section 2.2.2, Cavers (1978) introduces a secondary grid defined by the parameter $J_{\in}\mathbb{R}^+$ and it follows on that our algorithm will require this as input. We also remind ourselves that the methods differs in regard to the type of n; the data type is a list of integers compared to the previously mentioned where we passed an integer. The implementation is presented as:

Algorithm 4 Implementation of Equation 2.17

```
1: procedure CAVERS(f̃, n, J)
2:     γ ← ℝ⁺
3:     r ← 10^(γ/J)
4:     z ← Array[J]
5:     for k ← 0 to J − 1 do
6:         z[k] ← r · exp(i ∗ 2π ∗ k/J)
7:     end for
8:     F ← IFFT(f̃(z))
9:     f ← r^n · F[n]
10:    return f
11: end procedure
```

Whilst the latter makes use of the IFFT, we look into analysing the comparative metrics with the definition including that of the numerical approximation given by Eq (2.19). We denote the number of points in the secondary grid as J and let N be the last value / length of n.

Algorithm 5 Implementation of Equation 2.19

procedure CAVERS_SUM($\tilde{f}, n, J$)
  $\gamma \leftarrow \mathbb{R}^+$
  $N = n[-1]$
  $r \leftarrow 10^{\gamma/J}$
  $z \leftarrow \text{Array}[J]$
  for $j \leftarrow 0$ **to** J **do**
    $z[j] \leftarrow r \cdot \exp(i * 2\pi * j/N)$
  end for
  $f \leftarrow \text{Array}[N]$
  for $i \leftarrow 0$ **to** N **do**
    $f[i] \leftarrow \frac{1}{J} \cdot \sum(z^{n[i]} \cdot \tilde{f}(z))$
  end for
  return f
end procedure

Computational Accuracy

As discussed in Section 2.2.2, achieving machine-level accuracy is possible but requires some optimisation of the hyper-parameters. We aim to view the accuracy in regard to changes of the hyper-parameter n, J and the setting of γ. Our main focus is to try and achieve machine zero accuracy, E-16, with double precision. However, we also find it interesting to see the changes in accuracy making use of the numerical summation method outlined in Algorithm 5.

Computational Speed

Similar to the approach in Section 3.2.1, we plot the results against different N values, where we set J = 2N, for the different implementations. Given that IFFT has a time complexity of O(NLogN), we expect this to be the dominating factor in Algorithm 4. Similar to the approach in Section 3.2.1, we take the average over multiple runs to minimise the interference of external processes.

3.2.3 NumPy Vectorisation

NumPy vectorisation is used to perform array operations more efficiently by eliminating the need for explicit loops in Python. By leveraging NumPy's underlying C implementations (Harris et al., 2020), we can maximise the performance of the algorithms used in our project.

In the Abate and Whitt method, the computation involves a series of complex exponentials and summations. These computations can be handled in a single step rather than iterating through each element. This approach involves using vectorised indexing with `np.arange` to generate index arrays, and `np.sum` to efficiently compute the sum of array elements.

The Cavers IFFT method relies heavily on the FFT package, which is already highly optimised in NumPy. However, the DFT summation can benefit from the use of NumPy's broadcasting to perform element-wise operations across arrays of different shapes without explicit loops, and using vectorised matrix multiplication to compute the powers. Summation is once again done with `np.sum` across specified axes, allowing for efficient aggregation of the results.

Whilst we have outlined the general algorithm for the NIZT methods in the previous sections, we present a highly efficient Python implementations in Appendix A.

3.3 Sinh Deformation

A circular contour integral is commonly used for approximating the inverse z-transform as we've seen in Section 2.2.1 and 2.2.2. However, this does not mean it may be the best option. Boyarchenko and Levendorskiĭ (2022) propose a new method for a numerical evaluation of Eq. (2.14) by deforming the commonly used circular contour $\{z = re^{i\theta} | -\pi < \theta < \pi\}$ through the conformal mapping:

$$\xi(y) = \sigma + ib\sinh(i\omega + y), \tag{3.15}$$

where parameters are chosen such that $\sigma, y_{\in}\mathbb{R}$, $b_{\in}\mathbb{R}^{+}$, and ω is restricted to the interval $(-\pi/2, \pi/2)$. The unique selection of parameters allows the contour to be tailored specifically to the characteristics of $\tilde{f}(z)$. The authors state that the conformal mapping alleviates problems faced in literature (Boyarchenko and Levendorskiĭ, 2014, 2019; Schmelzer and Trefethen, 2007) in regard to the degree of accuracy and computational time. Applying a change of variables to Eq. (2.14) yields

$$f(n) = \int_{\mathbb{R}} \frac{b}{2\pi}\xi(y)^{-n-1}\cosh(i\omega + y)\tilde{f}(\xi(y))dy. \tag{3.16}$$

By denoting the integrand as $x_n(y)$ and approximating the application of the infinite trapezoidal, we have

$$f(n) \approx 2\epsilon \ \mathrm{Re}\left(\sum_{j\in\mathbb{Z}} x_n(j\epsilon)(1 - \delta_{j0}/2)\right) \tag{3.17}$$

While the authors provide numerical examples, they omit the parameters needed to recreate the experiment and lack an explanation to the sufficient conditions for their numerical examples. However, Boyarchenko and Levendorskii (2023) provide a more in-depth discussion into the sufficient conditions for the applicability of their NIZT, albeit "rich" in complexity, and impose conditions for the use on certain processes.

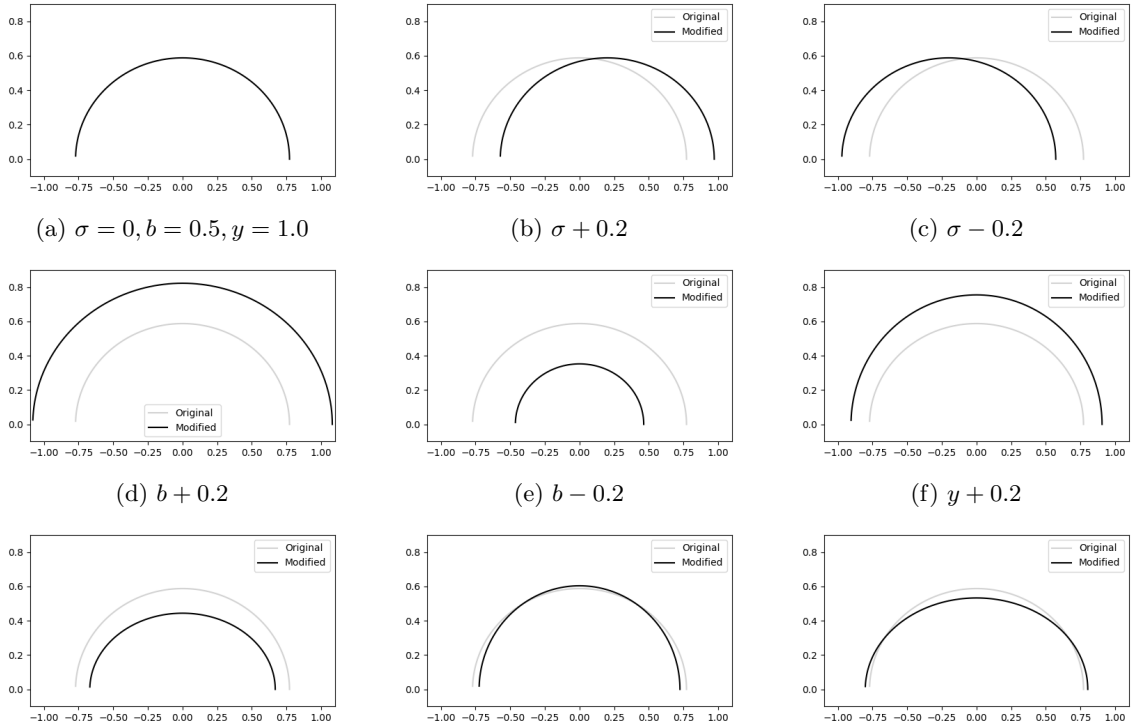
Considering the details, numerous factors and references ($\approx 53.85\%$ self-citations), we instead propose to learn of the effects of the parameters and see if we can back-track the sinh deformation to that of the unit circle. By finding parameters that achieve a circular contour, we aim to provide a setup in creating a heuristic implementation to the previously mentioned circular contour methods.

3.3.1 Deforming the Contour

In an attempt to recreate the unit circle from the conformal mapping (Equation 3.15), we first try to understand the effects of the parameters on the contour. We plot the contour for different values of σ, b and y to observe the deformation.

Algorithm 6 Implementation of Equation 3.15

```
procedure HYPERBOLIC_SINE(σ, b, y, N)
    z ← Array[N]                                   ▷ z ∈ ℂ^N
    for k ← 0 to N − 1 do
        ω ← i · (−π/2 + kπ/N)
        z[k] ← σ + i · b · sinh(ω + y)
    end for
    return z
end procedure
```



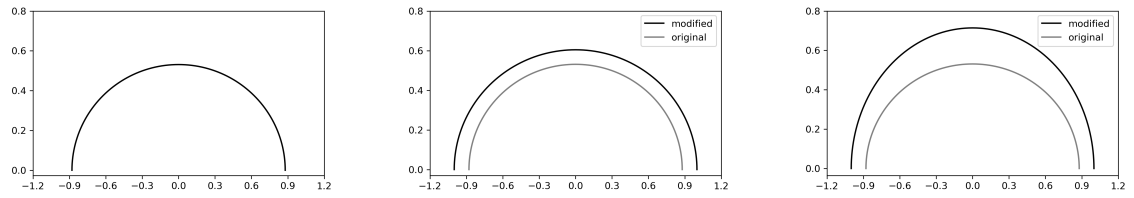
(a) $\sigma = 0, b = 0.5, y = 1.0$ (b) $\sigma + 0.2$ (c) $\sigma - 0.2$

(d) $b + 0.2$ (e) $b - 0.2$ (f) $y + 0.2$

(g) $y - 0.2$ (h) $b - 0.1, y + 0.2$ (i) $b + 0.1, y - 0.2$

Figure 3.2: Displaying the effect of the parameters on the contour for Algorithm 6 with $\{-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}\}$ and $n = 100$ points. Plots (b, c, d, e, f, g, h, i) highlights the changes in parameter the effects on the contour in comparison to the base contour (a). The black line represents the created contour, with the grey contour representing the base contour with parameters $\sigma = 0, b = 0.5, y = 1.0$.

In relation to the base contour (Figure 3.2a), we observe that increasing σ shifts the contour to the right (Figure 3.2b), while decreasing σ shifts the contour to the left (Figure 3.2c). The parameter b acts as a scale factor on the sinh deformation as shown in Figure 3.2d and 3.2e. Similarly, y acts as a scale factor but with a more pronounced vertical effect as seen in Figure 3.2f and 3.2g. Mixing the parameters b and y results in a more complex deformation where the counter-opposing effects changes the contour in a non-linear manner (Figure 3.2h and 3.2i).



(a) $\max(\text{Re}(z)) = 0.88$ (2 d.p.) (b) $\max(\text{Re}(z)) = 1.0$ (c) $\max(\text{Re}(z)) = 1.0$

Figure 3.3: Displaying the scaling effect of parameters b and y. The initial deformation is created with **(a)** $\sigma = 0.0, b = 0.7, y = 0.7$ and the scaled results are constructed with **(b)** setting only $b = 0.7967054596179172$ and **(c)** setting only $y = 0.895588098866514$. The parameters were constructed with N = 100 points and chosen with an error tolerance of E-9.

To achieve an extent on the real axis of 1.0, we scale our original deformation (Figure 3.3a) by the individual parameters. We see the differences in transformation in Figure 3.3b and Figure 3.3c for parameters b and y, respectively.

3.3.2 Parameter Selection for the Sinh Deformation

Given the complex changes resulting from the parameters b and y, we utilise optimisation techniques (Section 2.3) to find values that yield a contour resembling the unit circle. To achieve this, we first formulate a loss function, L_1, to set out our criteria. We let

$$z = [z_0, z_1, .., z_{n-1}]$$

be the representation of a contour consisting of $|z| = n$ points, where $z_k = x + iy$ for $0 \leq k < n$. If we consider the unit circle, each point z_k has a distance of 1 from the origin (0,0). Thus, to compare the similarity for a given contour, we can define the loss function $L_1 : \mathbb{C}^n \to \mathbb{R}$ as:

$$L_1 = \sum_{i=0}^{n-1} \left(|z_i| - 1\right)^2, \tag{3.18}$$

where we square the deviation of each point z_i from that of the unit radius to account for both positive and negative differences. The summation of all the squared deviations gives us our total loss. As a comparative measure, we compute the loss for the upper half of the unit circle for $n = 100$ points and $\theta_{\in}[0, \pi]$: $L_1 \approx$ 7E-31.

To minimise L_1 for a given contour, we initially use the gradient descent algorithm outlined in Algorithm 1. We numerically estimate the partial derivative of L_1, with respect to x, using finite difference approximation with $\epsilon = 10^{-8}$; see Burden and Faires (1997):

$$\frac{\partial L_1}{\partial x} \approx \frac{L_1(x + \epsilon) - L_1(x)}{\epsilon} \tag{3.19}$$

From our understanding of the parameter σ, seen in Section 3.3.1, we set it to 0 and try to learn the parameters b and y. This is because σ controls the shift in the x-axis as shown in Figure 3.2c and Figure 3.2b. The general steps is given by:

Algorithm 7 GD for Minimising L_1

procedure GD(α, ℓ)
  Initialise b and y ▷ with given constraints (Section 3.3)
  $\sigma \leftarrow 0.0$
  $n \leftarrow 100$
  $L_1 \leftarrow \infty$
  while $L_1 < \ell$ **do** ▷ until loss within acceptable tolerance
    $z \leftarrow$ `hyperbolic_sine`(σ, b, y, n) ▷ outlined in Algorithm 6
    $L_1 \leftarrow \sum_{i=0}^{|z|}(|z_i| - 1)^2$
    Compute gradients $\frac{\partial L_1}{\partial b}$ and $\frac{\partial L_1}{\partial y}$
    $b \leftarrow b - \alpha \frac{\partial L_1}{\partial b}$
    $y \leftarrow y - \alpha \frac{\partial L_1}{\partial y}$
  end while
  return b, y
end procedure

For the learning of these parameters, we stick to $n = 100$ points. Within our first attempts of the gradient descent algorithm, we opt to use initial parameter values of $b = 0.7$ and $y = 0.895588$ with a learning rate of $\alpha =$ 1E-3. Our choice of the initial parameters stem from the closest

resemblance to that of the unit circle seen in Figure 3.3c, which has a L_1 loss of 2.89 (2 d.p.).

However, achieving such a high-level degree of accuracy may become challenging if the learning rate in gradient descent requires constant adjustments and proves to be ineffective. In such cases, we can switch to using the ADAM optimiser (Section 2.3) to handle the situation more effectively. We structure our approach in the following pseudocode:

Algorithm 8 ADAM for Minimising L_1

procedure ADAM$(n, \alpha, \beta_1, \beta_2, \epsilon, \ell)$
  Initialise b and y
  $m_b, v_b, m_y, v_y \leftarrow 0, 0, 0, 0$
  $t \leftarrow 0$
  $L_1 \leftarrow \infty$
  while $L_1 > \ell$ **do** ▷ ℓ denotes the acceptable tolerance
    $z \leftarrow$ `hyperbolic_sine`(σ, b, y, n) ▷ outlined in Algorithm 6
    $L_1 \leftarrow \sum_{i=0}^{|z|}(|z_i| - 1)^2$
    Compute gradients $\frac{\partial L_1}{\partial b}$ and $\frac{\partial L_1}{\partial y}$ ▷ using Eq. (3.19)
    $m_b \leftarrow \beta_1 m_b + (1 - \beta_1)\frac{\partial L_1}{\partial b}$
    $m_y \leftarrow \beta_1 m_y + (1 - \beta_1)\frac{\partial L_1}{\partial y}$
    $v_b \leftarrow \beta_2 v_b + (1 - \beta_2)(\frac{\partial L_1}{\partial b})^2$
    $v_y \leftarrow \beta_2 v_y + (1 - \beta_2)(\frac{\partial L_1}{\partial y})^2$
    $\hat{m}_b \leftarrow \frac{m_b}{1-\beta_1^t}$
    $\hat{m}_y \leftarrow \frac{m_y}{1-\beta_1^t}$
    $\hat{v}_b \leftarrow \frac{v_b}{1-\beta_2^t}$
    $\hat{v}_y \leftarrow \frac{v_y}{1-\beta_2^t}$
    $b \leftarrow b - \alpha\frac{\hat{m}_b}{\sqrt{\hat{v}_b}+\epsilon}$
    $y \leftarrow y - \alpha\frac{\hat{m}_y}{\sqrt{\hat{v}_y}+\epsilon}$
  end while
  return b, y
end procedure

To achieve a sufficient set of parameters, we aim to exploit as much similarity with the unit circle as possible with our values of b and y; we let $\ell =$ 1E-30. We set our initial parameter values based on the results of our gradient descent algorithm, with a learning rate of $\alpha =$ 1E-5. Following the recommendations of Kingma and Ba (2014) and Goodfellow et al. (2016), we set $\beta_1 = 0.9$ and $\beta_2 = 0.999$, based on their extensive experiments and theoretical considerations.

Distance between each points

Redo calculations and double check - feels wrong

In Section 3.2, we see that the methods construct the circular contour with n points spaced out evenly between $0 \leq \theta \leq \pi$. However, it is clearly shown that this is not always the case when constructing a contour using the conformal mapping, defined in Eq. (3.15), and illustrated in Figure 3.2.

Example 3 *Consider the upper-half of a circular contour $z_\in\mathbb{C}^n$ numerically constructed with $n =$ 100 points of the form $z_k = re^{i(k(\pi+\frac{\pi}{n})/n)}$, where $0 \leq k < n$ and $0 \leq \frac{k(\pi+\frac{\pi}{n})}{n} \leq \pi$. This enables the first point to be situated at e^{i0} and the last point at to be situated at $e^{i\pi}$.*

To determine the distance between each consecutive point, we note that the angle between each point is $\Delta\theta = \frac{\pi}{n-1}$*. For* $n = 100$*, this simplifies to* $\Delta\theta = \frac{\pi}{99}$*. The distance* d *between two consecutive points on the unit circle can be calculated using the formula* $d = 2\sin\left(\frac{\Delta\theta}{2}\right)$*. Substituting* $\Delta\theta = \frac{\pi}{99}$ *gives:*

$$d = 2\sin\left(\frac{\pi}{198}\right) \approx 2 \cdot \frac{\pi}{198} = \frac{\pi}{99}$$

Numerically, this distance is approximately 0.0317*. Therefore, for a circular contour with* n *points, the distance between each consecutive point can be approximated as* $\frac{\pi}{n-1}$ *given the correct start and end points.*

Thus, if we consider our sinh deformation, we can formulate a mini-experiment into finding the average distance between each point, constructed with the parameters found in the previous section, and see how this differs to the unit circle.

Chapter 4

Results

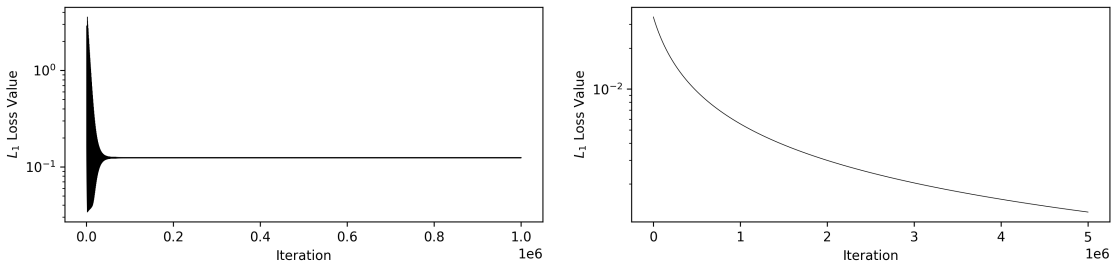
We delve into the experimental results of our study outlined in the previous chapter. The primary focus is on evaluating the efficiency and accuracy of various NIZT techniques when applied to well-defined transform pairs. Having conducted an analysis of the sinh deformation laid out by Boyarchenko and Levendorskiĭ (2022), we present our results in resembling the deformed contour to that of a circular contour. We then provide a comparative analysis of the successful methods against well-defined transform pairs.

The use of optimisation techniques were conducted on a windows 11 system with an i9-13900k, 32GB 6000MHz RAM and a GeForce Nvidia RTX 4090. The numerical benchmarking of the NIZT methods were tested on a Sonoma 14.4.1 macOS (2020 MacBook Pro) system with a 1.4GHz Intel Core i5 and 8GB 2133MHz RAM.

4.1 Parameter Selection for the Sinh Deformation

Our first experiment is to find parameters b and y that result in a contour resembling the upper-half of the unit circle. We employ an adaptation of the gradient descent method, outlined in Algorithm 7, and then try to improve on this using the ADAM optimiser, outlined in Algorithm 8. Each implementation returns parameters achieving the best loss during its execution.

4.1.1 Gradient Descent



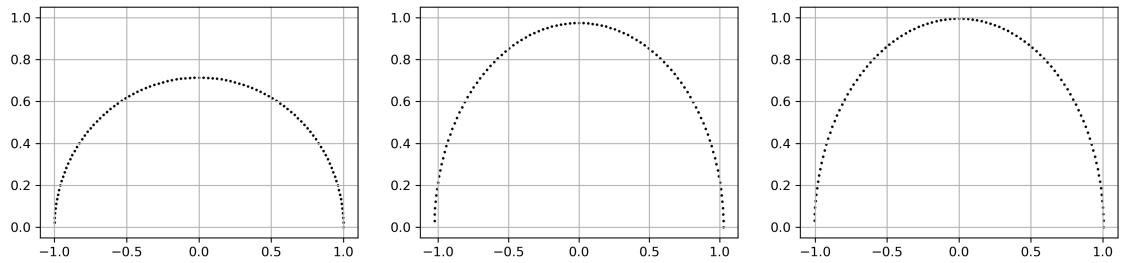
(a) α = 1E-3

(b) α = 1E-5

Figure 4.1: Plots of the L_1 loss against each of the (a) 1,000,000 and (b) 5,000,000 iterations. Each contour consists of $n = 100$ points. The initial parameters used are (a) $b = 0.7$ and $y = 0.895588$ (b) $b = 0.3227341040398728$ and $y = 1.8244690757568378$. Figure (a)'s lowest score was $L_1 \approx$ 3E-2 with the initial parameters of (b). Figure (b)'s lowest score was $L_1 \approx$ 1E-3 with parameters $b = 0.14124692711511863$ and $y = 2.6503746377415633$.

From our first attempt at the gradient descent (GD) algorithm (Figure 4.1a), we see that the L_1 loss converges to a higher value than the best loss of $L_1 = 0.03390312294797621 \approx$ 3E-2 with parameters $b = 0.3227341040398728$ and $y = 1.8244690757568378$. This indicates that the GD algorithm did not effectively reduce the loss suggesting that the learning rate α might be too high, causing the algorithm to overshoot the optimal parameter values.

Given the sub-optimal convergence observed in Figure 4.1a, we ran a second set of experiments with a reduced learning rate of $\alpha = 1 \times 10^{-5}$ over 1,000,000 iterations initially. We used the parameters that achieved the best loss from our first run and we see a much better convergence rate in Figure 4.1b for the first 1,000,000 iterations. We then extend this over till we see a notable decrease in the convergence rate at around iteration 5×10^6. By lowering the learning rate, the GD algorithm was able to achieve a lower L_1 loss of $0.0012438982603344348 \approx$ 1E-3.

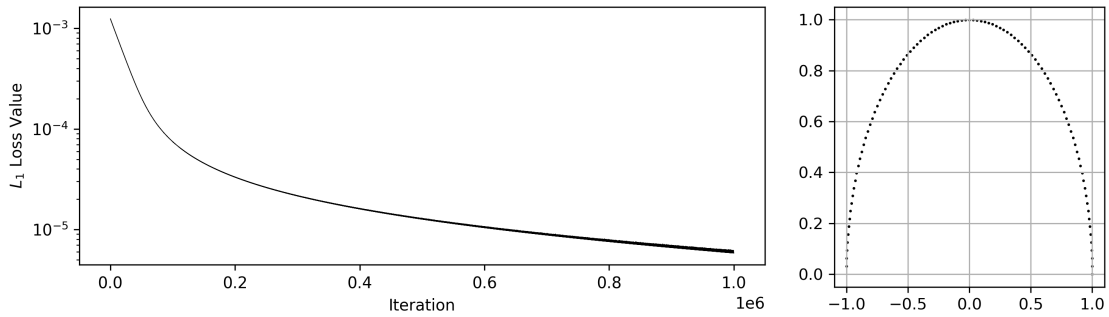


(a) $L_1 = 2.89186578074851$ (b) $L_1 = 0.03390312294797621$ (c) $L_1 = 0.0012438982603344348$

Figure 4.2: Visualisation of the contours achieved by the gradient descent algorithm in comparison to our initial parameters with their respective L_1 loss values.

Whilst we can see the gradual improvements (Figure 4.2), this is still very far from that of the unit circle with $L_1 \approx 7E-31$. And whilst reducing the learning rate will achieve a better loss, continuous adjustments of learning rate α is rather tedious practically. We thus employ ADAM optimiser in attempt to achieve a higher level of accuracy.

4.1.2 ADAM Optimiser



(a) 1,000,000 iterations with $\alpha =$ 1E-5 and initial loss of $L_1 \approx$ 1.2E-3 (b) $L_1 \approx$ 5.8E-6

Figure 4.3: L_1 loss plot over 1,000,000 iterations initialised with $n = 100$ points, $b = 0.14124692711511863$ and $y = 2.6503746377415633$. The optimiser achieves Figure 4.3b after its first execution constructed with parameters $b = 0.0369625708464355$ and $y = 3.9909964437511105$.

The Adaptive Moment Estimation (ADAM) optimiser provides a more sophisticated approach to parameter tuning as discussed in Section 2.3.1. We conducted our experiments initially with

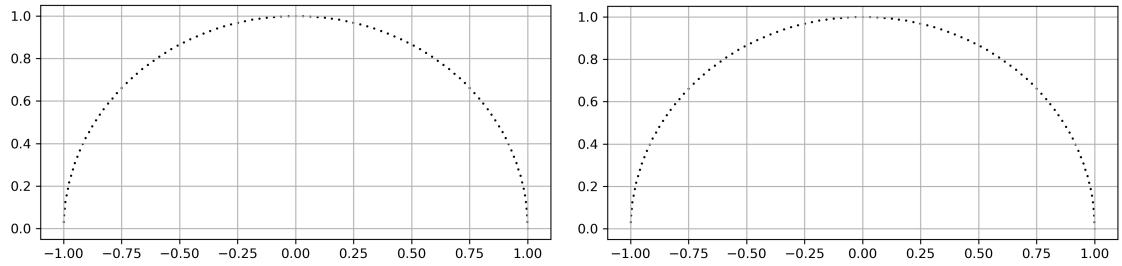
parameters set to $b = 0.14124692711511863$ and $y = 2.6503746377415633$, and the learning rate α was set to 1×10^5. At first glance, we see a much a better convergence and L_1 loss in Figure 4.3b, plotted over 1,000,000 iterations, compared to Figure 4.1b, plotted over 5,000,000 iterations. We thus reiterate the experiment but change the initial parameters in hopes of achieving a better L_1 score.

Here, we document the results of various runs with re-feeding the parameters back-to-back for each run. The table below lists the best L_1 losses during different stages of these runs.

b	y	L_1 Loss
0.023601865718395856	4.4395753229571016	9.888987368769754E-07
0.012128186643020126	5.105370236695753	6.761351664008986E-08
0.006856851878987493	5.675654087958846	6.90825126014524E-09
0.0016260199411660154	7.114767183400269	2.1845205202267918E-11
0.0010793477830991617	7.5245455065711715	4.241412093544462E-12
1.2712609608154088E-05	11.966063364769047	9.33039337162676E-15
1.2786721545826545E-05	11.960250483055397	4.955043180518183E-16
6.515415674882209E-05	10.331901633745023	8.120200210832519E-17
1.4883598328756484E-05	11.80839791524078	4.682445773102879E-19
1.48835270677098E-05	11.808402703087143	1.5334589506224306E-19

Table 4.1: Best L_1 loss, with corresponding parameters b and y, achieved during several stages of the runs using ADAM optimiser with varying initial learning rates of 1E-3 $\leq \alpha \leq$ 1E-7.

After multiple runs and adjustments to the initial learning rate, we achieve a loss of $L_1 \approx$ 1.5E-19 with parameters $b = 1.48835270677098$E-05 and $y = 11.808402703087143$. Whilst we were unable to find better parameters given the time constraints of this project, we deem this sufficient enough given the similarity to that of the unit circle (up to 18 decimal places). This is visually shown in the following figure:



(a) Circle Contour: $z = re^{i\theta}$

(b) Sinh Deformation: $\sigma + ib\sinh(\omega + y)$

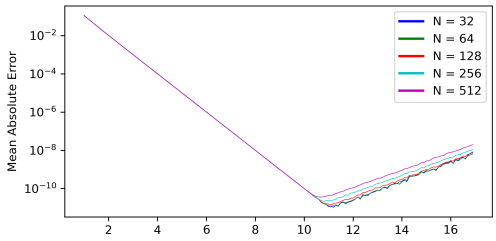
Figure 4.4: Visual representation for $n = 100$ points of the (a) circle contour with $r = 1$ and $0 \leq \theta < \pi$, and (b) sinh deformation with $\sigma = 0.0$, $b = 1.48835270677098$E-05, $y = 11.808402703087143$ and $-\frac{\pi}{2} \leq \omega < \frac{\pi}{2}$.

As discussed in Section 3.3.2, we check to see the average distance between each point of both the circular contour and the sinh deformation. In an idealistic setting, we expect the difference between each point to be $\frac{\pi}{99} \approx 0.03173325912716963$ as shown in Example 3. Figure 4.4a) has an average distance of 0.03172875474309711 whilst Figure 4.4b has an average distance of 0.031728754743097394. The numerical implementations of the contours are accurate up to 5 significant figures. However, the similarity of both contours to each other goes up to 17 significant figures. With both this affirmation and that of the L_1 loss, we can say the parameters provide a good level of success in regard to constructing the circular contour with the conformal mapping.

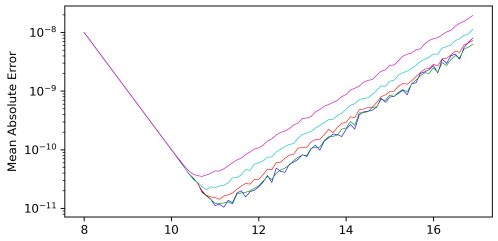
4.2 Numerical Benchmarking

Our next experiment is to evaluate the performance of the different NIZT techniques. The focus is on the efficiency and accuracy of the methods applied to various transform pairs (Section 3.1). The results are discussed in terms of the mean absolute error (MAE) and computational time.

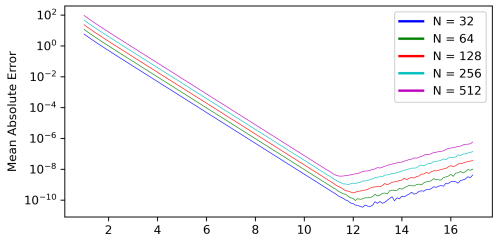
4.2.1 Abate and Whitt 1992



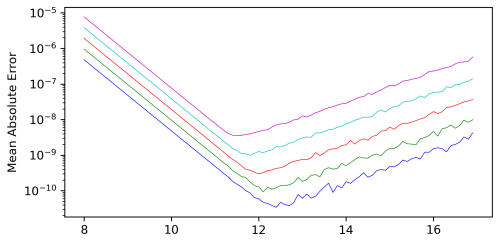
(a) Heaviside Step ($1 \leq \lambda < 17$)



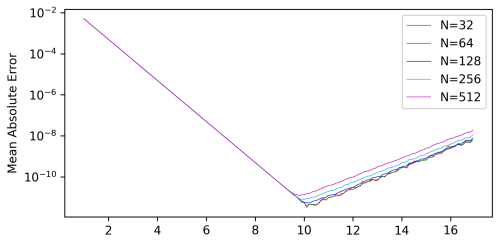
(b) Heaviside Step ($8 \leq \lambda < 17$)



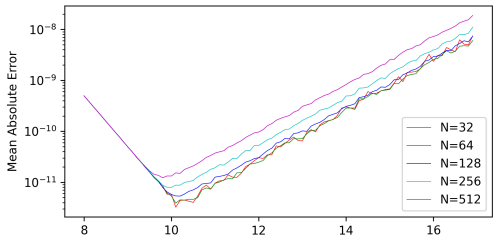
(c) Polynomial ($1 \leq \lambda < 17$)



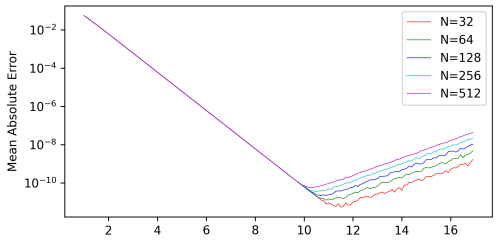
(d) Polynomial ($8 \leq \lambda < 17$)



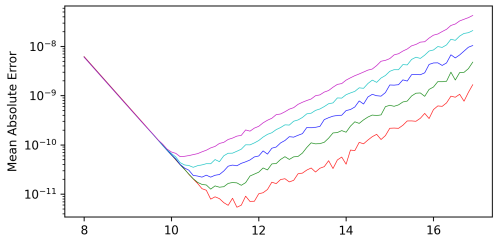
(e) Decaying Exponential ($1 \leq \lambda < 17$)



(f) Decaying Exponential ($8 \leq \lambda < 17$)

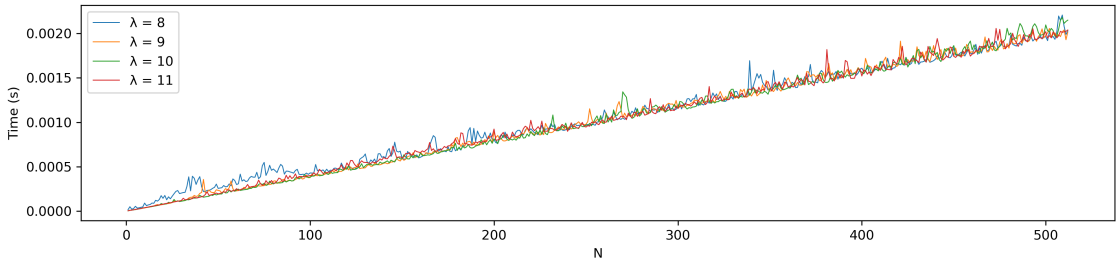


(g) Sinusoidal ($1 \leq \lambda < 17$)

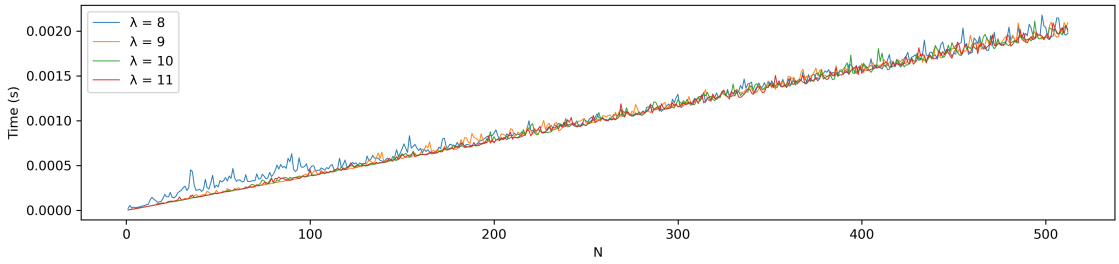


(h) Sinusoidal ($8 \leq \lambda < 17$)

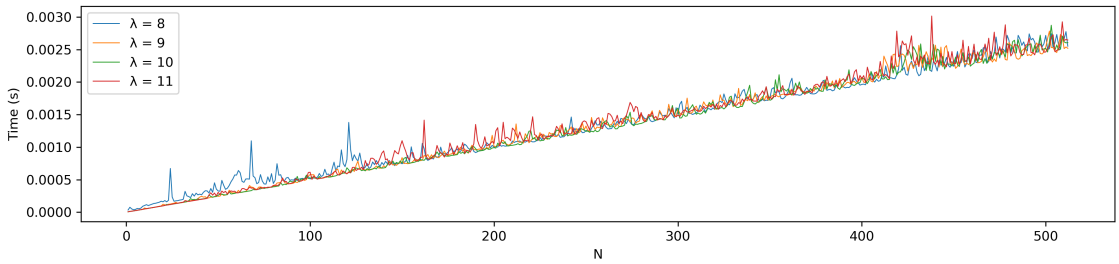
Figure 4.5: Mean absolute error (MAE) of the Abate-Whitt method for different values of λ when applied to the transform pairs: (a, b) Heaviside Step, (c, d) Polynomial, (e, f) Decaying Exponential with $a = 0.1$ and $t = 10$, and (g, h) Sinusoidal with $\omega = \frac{\pi}{4}$ and $t = 10$. The MAE is plotted on a logarithmic scale against λ values, where $\triangle\lambda = 0.1$. The different lines represent the MAE for values of N: 32, 64, 128, 256, 512.



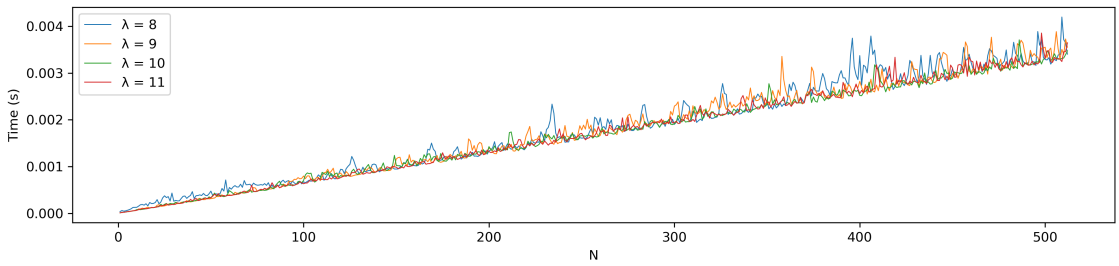
(a) Heaviside Step ($1 \leq N \leq 512$)



(b) Polynomial ($1 \leq N \leq 512$)



(c) Decaying Exponential ($1 \leq N \leq 512$)



(d) Sinusoidal ($1 \leq N \leq 512$)

Figure 4.6: Timings, in seconds, of the Abate-Whitt method for different values of N when applied to the transform pairs: (a) Heaviside Step, (b) Polynomial, (c) Decaying Exponential and (d) Sinusoidal. The different lines represent the chosen λ for each run. Each benchmark is taken as the average over 5 experiments taking the 3rd run of each experiment after restarting the kernel.

Our results (Figure 4.5) show that the mean absolute error for the different transform pairs (Heaviside Step, Polynomial, Decaying Exponential and Sinusoidal) varies significantly with the parameter λ for all $N = 32, 64, 128, 256$ and 512. According to our analysis in Section 3.2.1, the choice of the contour radius r influences the accuracy of the numerical inverse z-transform. Our results align with that of Loveless and Germano (2021) as we see our mean absolute error achieving around 11 significant figures until the accuracy decreases for $\lambda \geq 12$; this is the general case for all transform

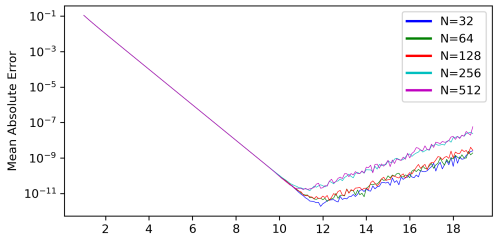
pairs explored in the experiment.

For each benchmark of the transform pairs, we restarted the Jupyter kernel and ran 2 successive runs to avoid any additional computational overhead and setup. The timings of the 3rd run were recorded and this was repeated 5 different times. Figure 4.6 shows the average of these 5 runs for each of the transform pairs (Heaviside Step, Polynomial, Exponential and Sinusoidal).

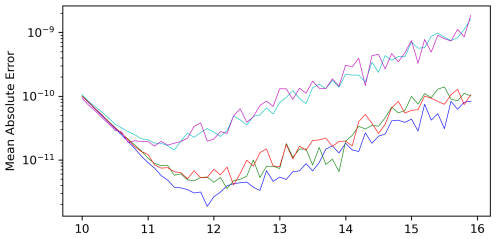
The results directly correlates with the discussion in Section 3.2.1 on the computational complexity of the method as we see a strong linear relationship on the number of points, N, for all transform pairs.

While the general case, in terms of efficiency and accuracy, are similar for each transform pair, we do mention some of the differences we have observed. For the decaying exponential and sinusoidal transform pairs, the mean absolute error remains relatively low across a range λ values, suggesting that the NIZT method handles these functions more effectively; we do note a significant time in computation for the Sinusoidal function. This is most likely due to the oscillatory nature of the sinusoidal function and the additional overhead from using NumPy's sine function in these evaluations (Harris et al., 2020).

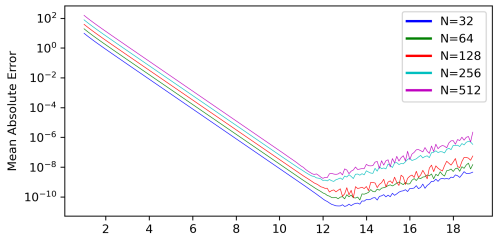
4.2.2 Cavers 1972



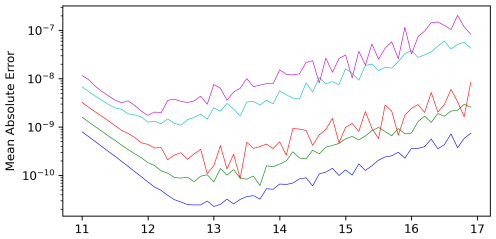
(a) Heaviside Step ($1 \leq \gamma < 19$)



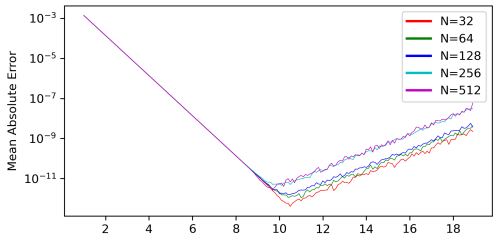
(b) Heaviside Step ($10 \leq \gamma < 16$)



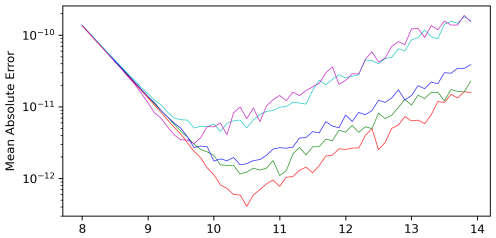
(c) Polynomial ($1 \leq \gamma < 19$)



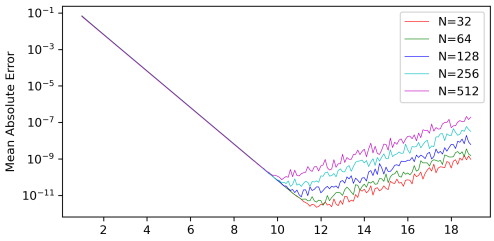
(d) Polynomial ($11 \leq \gamma < 17$)



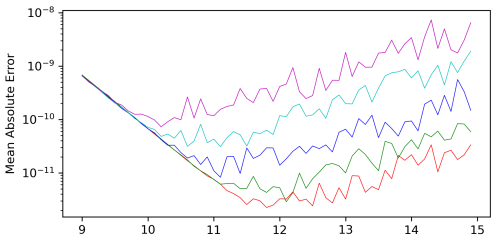
(e) Decaying Exponential ($1 \leq \gamma < 19$)



(f) Decaying Exponential ($8 \leq \gamma < 14$)

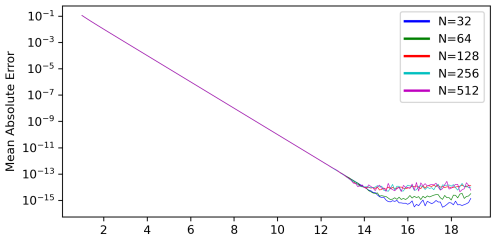


(g) Sinusoidal ($1 \leq \gamma < 19$)

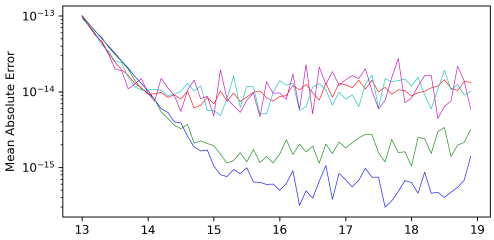


(h) Sinusoidal ($9 \leq \gamma < 15$)

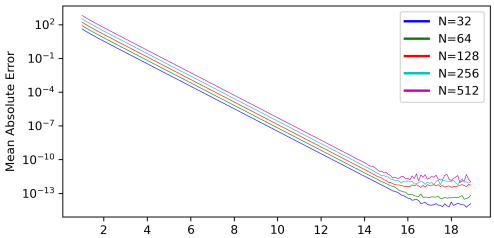
Figure 4.7: Mean absolute error (MAE) of the Cavers IFFT method for different values of λ when applied to the transform pairs: (a, b) Heaviside Step, (c, d) Polynomial, (e, f) Decaying Exponential with $a = 0.1$ and $t = 10$, and (g, h) Sinusoidal with $\omega = \frac{\pi}{4}$ and $t = 10$. The MAE is plotted on a logarithmic scale against λ values, where $\triangle\lambda = 0.1$. The different lines represent the MAE values of N, presented in the legend, with $J = 2$N.



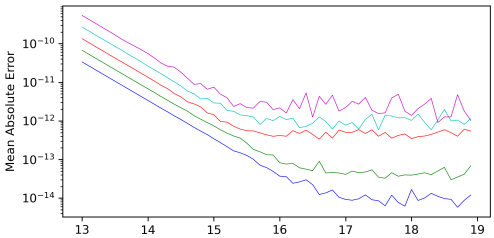
(a) Heaviside Step ($1 \leq \gamma < 19$)



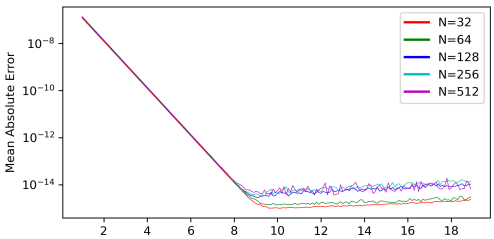
(b) Heaviside Step ($13 \leq \gamma < 19$)



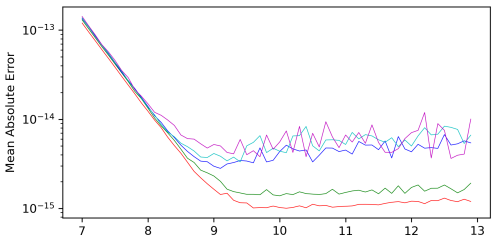
(c) Polynomial ($1 \leq \gamma < 19$)



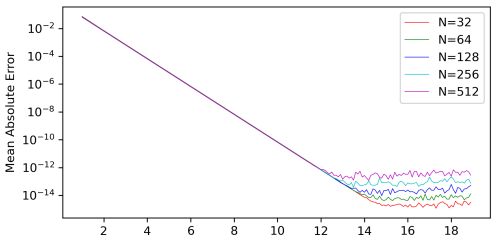
(d) Polynomial ($13 \leq \gamma < 19$)



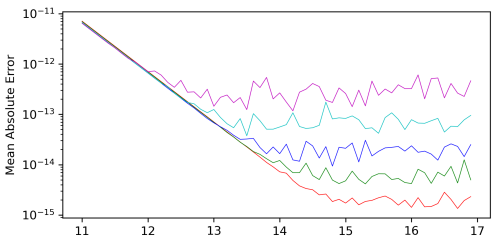
(e) Decaying Exponential ($1 \leq \gamma < 19$)



(f) Decaying Exponential ($7 \leq \lambda < 13$)

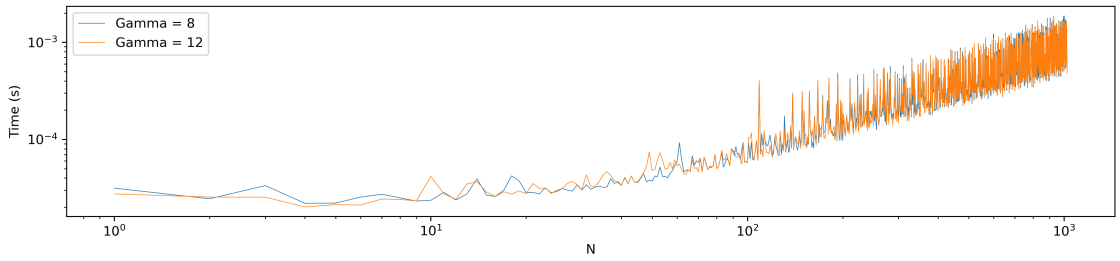


(g) Sinusoidal ($1 \leq \gamma < 19$)

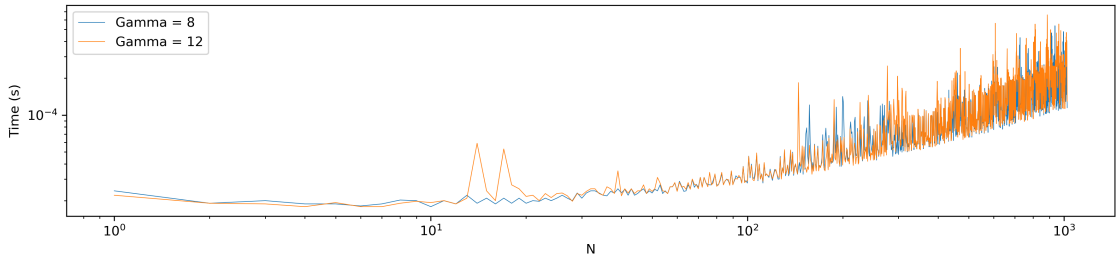


(h) Sinusoidal ($11 \leq \gamma < 17$)

Figure 4.8: Mean absolute error (MAE) of the Cavers IFFT method for different values of λ when applied to the transform pairs: (a, b) Heaviside Step, (c, d) Polynomial, (e, f) Decaying Exponential with $a = 0.1$ and $t = 10$, and (g, h) Sinusoidal with $\omega = \frac{\pi}{4}$ and $t = 10$. The MAE is plotted on a logarithmic scale against λ values, where $\triangle\lambda = 0.1$. The different lines represent the MAE values of N, presented in the legend, with $J = 8$N.

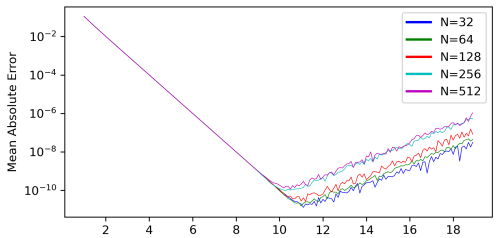


(a) Timings of Heaviside Step function

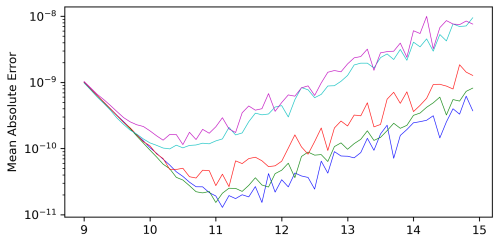


(b) Timings of Polynomial function

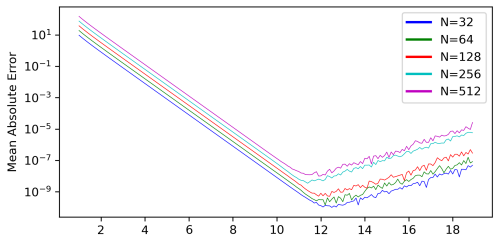
Figure 4.9: Timings, in seconds, of the Cavers IFFT method for different values of N when applied to the transform pairs: (a, c) Heaviside Step, and (b, d) Polynomial. The different lines represent the chosen γ for each run. Each benchmark is taken as the average over 5 experiments taking the 3rd run of each experiment after restarting the kernel. The timings are plotted on a logarithmic scaled and are computed with J=2N points.



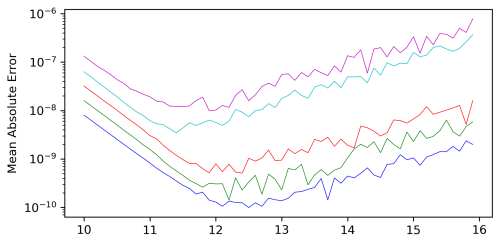
(a) Heaviside Step ($1 \leq \gamma < 19$)



(b) Heaviside Step ($9 \leq \gamma < 15$)



(c) Polynomial ($1 \leq \gamma < 19$)



(d) Polynomial ($10 \leq \gamma < 16$)

Figure 4.10: Mean absolute error (MAE) of the Cavers DFT method for different values of λ when applied to the transform pairs: (a, b) Heaviside Step, and (c, d) Polynomial. The different lines represent the MAE values of N, presented in the legend, with $J = 2N$.

In Figure 4.7, our lowest mean absolute error tends around the level of 1×10^{-11}. Whilst this is similar to the Abate and Whitt method, the Cavers IFFT method exhibits a non-linear time complexity, as shown in Figure 4.9. The key advantage of the Cavers method is its ability to reach 1×10^{-16} with some adjustments to the parameters. Whilst the latter was performed with J=2N, Figure 4.8 shows great success closing in on 1E-16 tolerance using J=8N. By incrasing the number of points within the secondary grid, the method can achieve machine-zero tolerance with double precision, but this enhancement comes at the cost of the increased time computational due to the added points with a time complexity of O(NLogN).

When comparing the Cavers DFT implementation (Algorithm 4) with the Caver's IFFT implementation (Algorithm 5), we notice differences in the mean absolute error. The DFT implementation generally shows a decrease in performance across all transform pairs (Figure 4.10). For instance, in the case of the Heaviside Step function, the IFFT implementation (Figure 4.7b) achieves a mean absolute error surpassing 1×10^{-11}, whereas the DFT implementation (Figure 4.10b) approaches but does not quite reach a mean absolute error of 1×10^{-11}.

4.2.3 Comparative Analysis

Whilst we have explored the general changes for a wide range of values in the previous section, we present the results for the best error achieved across all 3 approximations methods for the each of the transform pairs (Table 3.1).

Function	**Method**	**Parameters**	**Error**	**Time (s)**
Heaviside Step	Abate and Whitt	$\lambda = 10.9$	6.32E-12	0.0000420
	Cavers IFFT	$\gamma = 10.9$, J = 2N	4.27E-11	0.0000484
		$\gamma = 13.1$, J = 4N	3.28E-13	0.0000487
		$\gamma = 15.2$, J = 10N	6.65E-15	0.0000838
		$\gamma = 18.4$, J = 20N	4.53E-16	0.0001410
	Cavers DFT	$\gamma = 10.2$, J = 2N	1.83E-10	0.0004690
		$\gamma = 11.7$, J = 4N	8.64E-13	0.0009620
		$\gamma = 13.4$, J = 10N	2.86E-14	0.0024700
		$\gamma = 15.1$, J = 20N	2.18E-15	0.0045500
Polynomial	Abate and Whitt	$\lambda = 12.9$	1.51E-11	0.0000381
	Cavers IFFT	$\gamma = 11.8$, J = 2N	1.35E-09	0.0000401
		$\gamma = 13.4$, J = 4N	1.02E-11	0.0000424
		$\gamma = 17.9$, J = 10N	3.35E-13	0.0000627
		$\gamma = 15.9$, J = 20N	5.69E-14	0.0001130
	Cavers DFT	$\gamma = 11.0$, J = 2N	1.58E-09	0.0004870
		$\gamma = 13.4$, J = 4N	8.66E-12	0.0008990
		$\gamma = 16.6$, J = 10N	4.57E-13	0.0024100
		$\gamma = 16.2$, J = 20N	1.48E-14	0.0051300
Decaying Exponential	Abate and Whitt	$\lambda = 10.6$	7.75E-13	0.0000291
	Cavers IFFT	$\gamma = 10.2$, J = 2N	6.09E-13	0.0000303
		$\gamma = 10.9$, J = 4N	5.17E-15	0.0000358

Table 4.2 – *Continued on next page*

Continued from previous page

Function	Method	Parameters	Error	Time (s)
	Cavers DFT	$\gamma = 11.2$, J = 10N	4.60E-17	0.0000560
		$\gamma = 10.2$, J = 20N	3.20E-18	0.0000908
		$\gamma = 11.1$, J = 2N	3.36E-11	0.0004270
		$\gamma = 10.4$, J = 4N	2.31E-14	0.0008290
		$\gamma = 10.8$, J = 10N	1.77E-15	0.0021700
		$\gamma = 16.1$, J = 20N	2.67E-16	0.0048500
Sinusoidal	Abate and Whitt	$\lambda = 10.8$	6.25E-13	0.0000439
	Cavers IFFT	$\gamma = 10.7$, J = 2N	3.82E-12	0.0000412
		$\gamma = 10.1$, J = 4N	2.70E-14	0.0000520
		$\gamma = 13.9$, J = 10N	7.31E-16	0.0000877
		$\gamma = 11.0$, J = 20N	9.29E-16	0.0001440
	Cavers DFT	$\gamma = 10.2$, J = 2N	2.06E-10	0.0004340
		$\gamma = 12.2$, J = 4N	2.38E-12	0.0008500
		$\gamma = 13.5$, J = 10N	2.64E-14	0.0021900
		$\gamma = 17.0$, J = 20N	2.89E-17	0.0044700

Table 4.2: NIZT comparison for $n = 100$ on the transform pairs (i) Heaviside Step, (ii) Polynomial, (iii) Decaying Exponential with $a = 0.5$ and $t = 80$, and (iv) Sinusoidal with $\omega = \frac{\pi}{4}$ and $t = 80$. The selected parameters for λ and γ yield the best results for the range [8, 20) in increments of 0.1. The corresponding time taken is the lowest achieved across 10 runs.

The Cavers IFFT method generally demonstrates superior accuracy across the different transform pairs. In contrast, the Cavers DFT summation method performs subpar in all cases given the singular exception of the Sinusoidal function for $J = 20N$, where we achieve an error of 2.89E-17 and the latter achieves 9.29E-16. It can be clearly seen that the approximation formula by Cavers performs best with the Decaying Exponential and Sinusoidal transform pairs. We see an instance of surpassing machine-zero, with double precision, tolerance for the Decaying Exponential function with $\gamma = 11.2$ and J=10N; 4.60E-17. Given the lack of customisability for the Abate and Whitt method, our lowest error is displayed with the Decaying Exponential and Sinusoidal function at around 6E-13.

Given our findings, we can indeed conclude that, in most cases, the Abate and Whitt method achieves its highest accuracy at around $\lambda \approx 11$. Whilst we do achieve ≈1E-13 accuracy, our extensive experimentation (Figure 4.5) shows that the MAE maxes out at around 11/12 significant figures. From both Table 4.2 and Figure 4.8, Cavers can achieve a much higher degree of accuracy with some changes to the hyper-parameters.

While more accurate, Cavers incurs higher computational costs, especially as γ and J increase. The Heaviside Step function with J=20N takes 0.000141 seconds for the IFFT implementation and the DFT summation implementation takes 0.0045500 seconds. We ought to also mention that the computational cost is higher for Cavers DFT than the other methods in all cases. On the other hand, the method by Abate and Whitt seems to be the fastest or one of the fastest for all transform pairs. This can be seen to provide a more balanced approach with moderate accuracy and computational efficiency.

Chapter 5

Conclusion

5.1 Summary

This study successfully explores the relevancy of the inverse z-transform in the pricing of discrete path-dependent options. A greater outlook is given to Lookback and Barrier options and the necessitation for the use of numerical Fourier transform methods. The employment of numerical methods is discussed and outlined for both the general implementation and the Python implementation, making use of NumPy's vectorisation capabilities.

Whilst the latter involves a circular contour, we explore the idea of deforming the contour via a sinh function. The use optimisation techniques, such as gradient descent and its variant, the ADAM optimiser, is successfully employed to find parameters for the sinh deformation that achieves a contour resembling that of the unit circle; an accuracy greater than 1E-16, *machine-zero*, with double precision. This sets the stage for heuristic implementations for the Abate and Whitt method as well as Caver's.

In an attempt to compare the NIZT methods, we provide a comparison on the methods and a look into the sensitivity of the parameters to the output of the approximation formulas. The results seen validate the theoretical considerations demonstrating that the choice of parameters and the nature of the transform pair impact the accuracy and efficiency of the inverse z-transform.

The method introduced by Cavers (1978)'s is shown to achieve an accuracy surpassing 1E-16 but suffers from a high-computational cost. The method introduced by Abate and Whitt (1992a,b) offers an accuracy of around 1E-11 but with a computational cost lower than that of Cavers; a middle ground showcasing the trade-off between accuracy and efficiency.

5.2 Critical Evaluation

5.3 Future Work

5.3.1 Series Acceleration

5.3.2 Parameter Selection

During the course of the project, there was no set method for selecting the parameters for Equation 3.15, hence, our effort of employing numerous techniques in an attempt to find the optimal parameters. However, as of recently at the time of writing, Boyarchenko and Levendorskiĭ (2024)

have released a paper detailing parameter selection for the conformal mapping (Equation 3.15).

Bibliography

Abate, J. and Whitt, W. (1992a). The fourier-series method for inverting transforms of probability distributions. *Queueing systems*, 10:5–87.

Abate, J. and Whitt, W. (1992b). Numerical inversion of probability generating functions. *Operations Research Letters*, 12(4):245–251.

Bachelier, L. (1900). Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86.

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654.

Boyarchenko, S. and Levendorskiĭ, S. (2014). Efficient variations of the fourier transform in applications to option pricing. *Journal of Computational Finance*, 18(2).

Boyarchenko, S. and Levendorskiĭ, S. (2019). Sinh-acceleration: Efficient evaluation of probability distributions, option pricing, and monte carlo simulations. *International Journal of Theoretical and Applied Finance*, 22(03):1950011.

Boyarchenko, S. and Levendorskiĭ, S. (2022). Efficient inverse z-transform and pricing barrier and lookback options with discrete monitoring. *arXiv preprint arXiv:2207.02858*.

Boyarchenko, S. and Levendorskii, S. (2023). Efficient inverse z-transform: sufficient conditions. *arXiv preprint arXiv:2305.10725*.

Boyarchenko, S. and Levendorskiĭ, S. (2024). Efficient inverse z-transform and wiener-hopf factorization. *arXiv preprint arXiv:2404.19290*.

Burden, R. L. and Faires, J. D. (1997). *Numerical analysis*. Brooks Cole.

Carr, P. and Madan, D. (1999). Option valuation using the fast fourier transform. *Journal of computational finance*, 2(4):61–73.

Cavers, J. (1978). On the fast fourier transform inversion of probability generating functions. *IMA Journal of Applied Mathematics*, 22(3):275–282.

Chen, J., Fan, L., Li, L., and Zhang, G. (2021). Sinc approximation of multidimensional hilbert transform and its applications. *Available at SSRN 3091664*.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

Cox, J. C., Ross, S. A., and Rubinstein, M. (1979). Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263.

Cui, Z. and Taylor, S. (2021). Pricing discretely monitored barrier options under markov processes through markov chain approximation. *Journal of Derivatives*, 28(3):8–33.

Dadachanji, Z. (2015). *FX Barrier Options: A comprehensive guide for industry quants*. Springer.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Eberlein, E., Glau, K., and Papapantoleon, A. (2010). Analysis of fourier transform valuation formulas and applications. *Applied Mathematical Finance*, 17(3):211–240.

Fang, F. and Oosterlee, C. W. (2009a). A novel pricing method for european options based on fourier-cosine series expansions. *SIAM Journal on Scientific Computing*, 31(2):826–848.

Fang, F. and Oosterlee, C. W. (2009b). Pricing early-exercise and discrete barrier options by fourier-cosine series expansions. *Numerische Mathematik*, 114(1):27–62.

Fang, F. and Oosterlee, C. W. (2011). A fourier-based valuation method for bermudan and barrier options under heston's model. *SIAM Journal on Financial Mathematics*, 2(1):439–463.

Feng, L. and Linetsky, V. (2008). Pricing discretely monitored barrier options and defaultable bonds in lévy process models: a fast hilbert transform approach. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 18(3):337–384.

Fusai, G., Abrahams, I. D., and Sgarra, C. (2006). An exact analytical solution for discrete barrier options. *Finance and Stochastics*, 10:1–26.

Fusai, G., Germano, G., and Marazzina, D. (2016). Spitzer identity, wiener-hopf factorization and pricing of discretely monitored exotic options. *European Journal of Operational Research*, 251(1):124–134.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Guardasoni, C., Rodrigo, M. R., and Sanfelici, S. (2020). A mellin transform approach to barrier option pricing. *IMA Journal of Management Mathematics*, 31(1):49–67.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Ro, J. F., Wiebe, M., Peterson, P., Grard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). *Array programming with NumPy*.

Heston, S. L. (1993). 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.

Horváth, I., Mészáros, A., and Telek, M. (2020). Numerical inverse transformation methods for z-transform. *Mathematics*, 8(4):556.

Kemperman, J. (1963). A wiener-hopf type method for a general random walk with a two-sided boundary. *The Annals of Mathematical Statistics*, 34(4):1168–1193.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkby, J. L. and Nguyen, D. (2020). Efficient asian option pricing under regime switching jump diffusions and stochastic volatility models. *Annals of Finance*, 16(3):307–351.

Kou, S. (2007). Discrete barrier and lookback options. *Handbooks in operations research and management science*, 15:343–373.

Kreyszig, E. (2010). *Advanced Engineering Mathematics*. John Wiley & Sons.

Kwok, Y. K., Leung, K. S., and Wong, H. Y. (2011). Efficient options pricing using the fast fourier transform. In *Handbook of computational finance*, pages 579–604. Springer.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing american options by simulation: a simple least-squares approach. *The review of financial studies*, 14(1):113–147.

Lord, R., Fang, F., Bervoets, F., and Oosterlee, C. W. (2008). A fast and accurate fft-based method for pricing early-exercise options under lévy processes. *SIAM Journal on Scientific Computing*, 30(4):1678–1705.

Loveless, B. and Germano, G. (2021). Review of numerical inversion techniques of the z-transform. *Working Paper*.

Loveless, B., Phelan, C. E., and Germano, G. (2023). Accurate numerical inverse z-transform and its use in the fourier-z pricing of discretely monitored path-dependent options. *Preprint submitted to Elsevier*. Preprint submitted January 12, 2023.

Lu, S. and Jin, Z. (2017). Improved stochastic gradient descent algorithm for svm. *International Journal of Recent Engineering Science (IJRES)*, 4(4):28–31.

Madan, D. B., Carr, P. P., and Chang, E. C. (1998). The variance gamma process and option pricing. *Review of Finance*, 2(1):79–105.

Merrikh-Bayat, F. (2014). Two methods for numerical inversion of the z-transform. *arXiv preprint arXiv:1409.1727*.

Merton, R. C. (1973). Theory of rational option pricing. *The Bell Journal of economics and management science*, pages 141–183.

Nyquist, H. (1928). Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644.

Persson, P.-O., Franco, M., and Sweeney Blanco, R. (2022). Gradient-based optimization. Accessed: [10-03-2024].

Phelan, C. E. (2018). *Fourier transform methods for the pricing of barrier options and other exotic derivatives*. PhD thesis, UCL (University College London).

Phelan, C. E., Marazzina, D., Fusai, G., and Germano, G. (2018). Fluctuation identities with continuous monitoring and their application to the pricing of barrier options. *European Journal of Operational Research*, 271(1):210–223.

Phelan, C. E., Marazzina, D., Fusai, G., and Germano, G. (2019). Hilbert transform, spectral filters and option pricing. *Annals of Operations Research*, 282(1):273–298.

Proakis, J. G. (2007). *Digital signal processing: principles, algorithms, and applications, 4/E*. Pearson Education India.

Rajković, P. M., Stanković, M. S., and Marinković, S. D. (2004). A method for numerical evaluating of inverse z-transform. *Facta universitatis-series: Mechanics, Automatic Control and Robotics*, 4(16):133–139.

Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Ross, S. M. (2014). *Introduction to probability models*. Academic press.

Ruijter, M., Versteegh, M., and Oosterlee, C. W. (2015). On the application of spectral filters in a fourier option pricing technique. *Journal of Computational Finance*, 19(1):75–106.

Schafer, R. W. and Oppenheim, A. V. (1989). *Discrete-time signal processing*, volume 5. Prentice Hall Englewood Cliffs, NJ.

Schmelzer, T. and Trefethen, L. N. (2007). Computing the gamma function using contour integrals and rational approximations. *SIAM Journal on Numerical Analysis*, 45(2):558–571.

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

Soleymani, F. and Barfeie, M. (2019). Pricing options under stochastic volatility jump model: A stable adaptive scheme. *Applied Numerical Mathematics*, 145:69–89.

Spitzer, F. (1957). The wiener-hopf equation whose kernel is a probability density.

Thompson, W. J. (1994). Aristotle: Philosophy and politics, theory and practice. In *Proceedings of the American Catholic Philosophical Association*, volume 68, pages 109–124.

Tian, Y., Zhang, Y., and Zhang, H. (2023). Recent advances in stochastic gradient descent in deep learning. *Mathematics*, 11(3).

Tieleman, T. and Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 17.

Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., and Zebari, D. A. (2019). Trainable model based on new uniform lbp feature to identify the risk of the breast cancer. In *2019 international conference on advanced science and engineering (ICOASE)*, pages 106–111. IEEE.

Zhang, G. and Li, L. (2023). A general approach for lookback option pricing under markov models. *Quantitative Finance*, 23(9):1305–1324.

Zhang, J. (2019). Gradient descent based optimization algorithms for deep learning models training. *arXiv preprint arXiv:1903.03614*.

Appendices

Appendix A

Code Listings

A.1 Transform Pairs

These transform pairs serve as benchmarks for testing the accuracy and efficiency of numerical inversion methods. Each pair consists of a time-domain function and its corresponding Z-transform, providing a practical basis for analysing and validating the implemented algorithms.

A.1.1 Heaviside Step

```
def f(t):
    return 1

def ftilde(z):
    return z / (z-1)
```

Listing A.1: Implementation of the Heaviside Step function and its $\mathcal{Z}$-transform (Section 3.1.1)

A.1.2 Polynomial

```
def f(t):
    return t

def ftilde(z):
    return z / (z-1)**2
```

Listing A.2: Implementation of the Polynomial function and its $\mathcal{Z}$-transform (Section 3.1.2)

A.1.3 Decaying Exponential

```
def f(a, t):
    return np.exp(-a * t)

def ftilde(z, a, t, N):
    delta_t = t / N
    denominator = 1 - np.exp(-a * delta_t) / z
    return 1 / denominator
```

Listing A.3: Implementation of the Decaying Exp function and its $\mathcal{Z}$-transform (Section 3.1.3)

A.1.4 Sinusiodal

```
def f(omega, t):
    return np.sin(omega * t)

def ftilde(z, omega, t, N):
    delta_t = t / N
    numerator = z**(-1) * np.sin(omega * delta_t)
    denominator = 1 - 2 * np.cos(omega * delta_t) * z**(-1) + z**(-2)
    return numerator / denominator
```

Listing A.4: Implementation of the Sinusoidal function and its $\mathcal{Z}$-transform (Section 3.1.4)

A.2 Contours

A.2.1 Circular

```
def circle(r, n):
    z = []
    for k in range(0, n):
        theta = np.pi * k / n
        point = r * np.exp(1j * theta)
        z.append(point)
    return z
```

Listing A.5: Implementation of a circular contour from $0 \leq \theta < \pi$.

```
def circle(r, n):
    z = []
    for k in range(0, n):
        theta = (np.pi + np.pi / n) * k / n
        point = r * np.exp(1j * theta)
        z.append(point)
    return z
```

Listing A.6: Implementation of a circular contour from $0 \leq \theta \leq \pi$.

A.2.2 Sinh Deformation

```
def hyperbolic_sine(sigma, b, y, n):
    z = np.zeros(n, dtype=complex)
    for k in range(n):
        omega = 1j * (-np.pi/2 + k * np.pi / n)
        z[k] = sigma + 1j * b * np.sinh(omega + y)
    return z
```

Listing A.7: Implementation of the conformal mapping (Equation 3.15) from $-\frac{\pi}{2} \leq \omega < \frac{\pi}{2}$.

```
def hyperbolic_sine(sigma, b, y, n):
    z = np.zeros(n, dtype=complex)
    for k in range(n):
        omega = 1j * (np.pi) * (k - (n - 1) / 2) / (n - 1)
        z[k] = sigma + 1j * b * np.sinh(omega + y)
    return z
```

Listing A.8: Implementation of the conformal mapping (Equation 3.15) from $-\frac{\pi}{2} \leq \omega \leq \frac{\pi}{2}$.

A.3 Numerical Inverse $\mathcal{Z}$-Transform

We provide the python implementation of the numerical inverse z-transform for the methods discussed in Section 2.2. The implementations are structured such that we make use of NumPy's vectorised operations in an attempt of achieving the best performance possible.

A.3.1 Abate and Whitt 1992

```
def abate_whitt(ftilde, n, lmbda):
    rho = 10 ** (-lmbda / (2 * n))

    k = np.arange(1, n)
    z = 1 / (rho * np.exp(1j * k * np.pi / n))
    summation = 2 * np.sum(((-1) ** k) * np.real(ftilde(z)))

    result = (1 / (2 * n * rho ** n)) * (ftilde(1 / rho) + summation + ((-1) ** n)
    * ftilde(-1 / rho))
    return result
```

Listing A.9: Implementation of Abate and Whitt (1992a,b)'s NIZT (Equation 2.16).

A.3.2 Cavers 1978

```
def cavers(ftilde, n, N, gamma):
    r = 10 ** (gamma / (N))
    k = np.arange(N)
    z = r * np.exp(1j * 2 * np.pi * k / (N))
    F = np.fft.ifft(ftilde(z))
    f = r ** n * F[n]
    return f
```

Listing A.10: Implementation of Cavers (1978)'s FFT NIZT (Equation 2.17)

```
def cavers_dft(ftilde, n, J, gamma):
    r = 10 ** (gamma / J)

    j = np.arange(J)
    z = r * np.exp(1j * 2 * np.pi * j / J)

    z_ftilde = ftilde(z)
    z_power = z[:, None] ** n  # Compute z^n for all n and z

    f = np.sum(z_power * z_ftilde[:, None], axis=0) / J
    return f
```

Listing A.11: Implementation of Cavers (1978)'s DFT NIZT (Equation 2.19)

A.4 Optimisation Techniques

A.4.1 Loss Function

```
def loss(z):
    L1 = np.sum((np.abs(z) - 1) ** 2)
    return L1
```

Listing A.12: null

A.4.2 Gradient Descent

```
def gradient_descent(N, learning_rate=1e-3, num_iterations=1_000_000):
  # initial values
    b = 0.7
    y = 0.895588

    # initialising parameters
    best_loss = float('inf')
    best_b = b
    best_y = y

    for i in range(num_iterations):
        # compute loss
        L1 = loss_function(b, y, N)

        if L1 < best_loss:
            best_loss = L1
            best_b = b
            best_y = y

        # compute gradients
        epsilon = 1e-8
        grad_b = (loss_function(b + epsilon, y, N) - L1) / epsilon
        grad_y = (loss_function(b, y + epsilon, N) - L1) / epsilon

        # update parameters
        b -= learning_rate * grad_b
        y -= learning_rate * grad_y

    return best_b, best_y, best_loss
```

Listing A.13: null

A.4.3 ADAM Optimiser

```
def adam_optimiser(N, initial_learning_rate=1e-3, num_iterations=1_000_000, beta1
    =0.9, beta2=0.999, epsilon=1e-5):
  # initial values
    b = 0.14124692711511863
    y = 2.6503746377415633

  # initialising parameters
    best_loss = float('inf')
    best_b = b
    best_y = y
    m_b, v_b = 0, 0
    m_y, v_y = 0, 0
```

```
    t = 0

    for i in range(num_iterations):
        t += 1

        # compute loss
        L1 = loss_function(b, y, N)

        if L1 < best_loss:
            best_loss = L1
            best_b = b
            best_y = y

        # compute gradients (estimate)
        grad_b = (loss_function(b + epsilon, y, N) - L1) / epsilon
        grad_y = (loss_function(b, y + epsilon, N) - L1) / epsilon

        # update biased first moment
        m_b = beta1 * m_b + (1 - beta1) * grad_b
        m_y = beta1 * m_y + (1 - beta1) * grad_y

        # update biased second moment
        v_b = beta2 * v_b + (1 - beta2) * (grad_b ** 2)
        v_y = beta2 * v_y + (1 - beta2) * (grad_y ** 2)

        # compute bias-corrected first moment
        m_b_hat = m_b / (1 - beta1 ** t)
        m_y_hat = m_y / (1 - beta1 ** t)

        # compute bias-corrected second moment
        v_b_hat = v_b / (1 - beta2 ** t)
        v_y_hat = v_y / (1 - beta2 ** t)

        # update parameters
        b -= initial_learning_rate * m_b_hat / (np.sqrt(v_b_hat) + epsilon)
        y -= initial_learning_rate * m_y_hat / (np.sqrt(v_y_hat) + epsilon)

    return best_b, best_y, best_loss
```

Listing A.14: null