A study into using differential evolution to optimise an audio signal processing problem

Master's thesis
University of the Arts Helsinki, Sibelius Academy
Centre for Music and Technology

Olli Erik Keskinen

31.3.2019

Abstract

Computational optimisation is a prominent paradigm for solving complex technical problems. This master's thesis is a study into applying optimisation to an audio signal processing problem. Differential evolution is used to approximate weights for a linear time-variant system to create a frequency-variant window function to be used with spectral analysis. Technical background for the frequency-variant window function is presented, difficulties arising from working with a time-frequency related audio problem are identified, and problem formulation for realising the optimisation scheme is derived. Convergence of the optimisation method and the obtained results are assessed. Finally, further work based on the findings is discussed.

Tiivistelmä

Laskennallinen optimointi on merkittävä paradigma monimutkaisten teknisten ongelmien ratkaisuun. Tämä maisterin opinnäytetyö tutkii optimoinnin soveltamista audiosignaalinkäsittelyyn. Työssä tutkitaan differentiaalievoluutiomenetelmän käyttämistä taajuusvariantin ikkunointifunktion muodostamiseen spektrianalyysin tarpeisiin. Työ esittelee teknisen taustan taajuusvarianteille ikkunointifunktiolle ja tarkastelee äänisignaalin prosessointiin liittyvän optimointiongelman haasteita. Työssä määritellään taajuusvariantin ikkunointifunktion optimoimisen edellyttämä optimointiongelma ja arvioidaan optimoitavan funktion suppenemista sekä saatuja tuloksia. Työn lopuksi käsitellään työn pohjalta esiin nousseita jatkotutkimuskohteita.

Contents

1 Introduction **5**

1.1 Background 5

1.2 Goal of the thesis 6

1.3 Structure of the thesis 7

2 Theory **8**

2.1 Gaussian function 8

2.2 Moving average 10

2.3 Linear time-variant filters 11

2.4 Differential evolution 11

2.5 Overfitting 16

3 Formulating the problem **18**

3.1 Cost function 18

3.2 Target function $T(\hat{\mathbf{x}})$ 19

3.3 Moving average system $\mathbf{H}(\hat{\mathbf{L}})$ 19

3.4 Test signal $\hat{\mathbf{x}}$ 21

3.5 Regularisation term $R(\hat{\mathbf{p}})$ 23

4 Optimising the problem with differential evolution **25**

4.1 Motivation for the use of differential evolution 25

4.2 Parameters 26

4.3 Adaptations to the differential evolution 27

4.4 Initialisation 28

4.5 Parallelisation 29

5 Results **31**

5.1 Studying the converging population 31

5.2 Investigating the best obtained result 36

5.3 Illustrating the error of the best obtained result 39

5.4 Understanding the error 41

5.5 Discussing the findings 42

6 Conclusions **44**

6.1 Further work 44

A Background to windowing and time-frequency plane **47**

B Frequency-dependent window function **50**

C Arbitrary-length moving average as a matrix operation **52**

C.1 Arbitrary-length moving average 53

C.2 Adapting the matrix 54

C.3 Treating the q for faster convergence 55

D Deriving the initial guesses for the population **57**

References **58**

1 Introduction

1.1 Background

Computational optimisation has become a ubiquitous approach for solving complex engineering problems. Optimisation is applied across fields to find solutions to problems that are difficult to solve with traditional means. Such problems might not have a well defined analytical solution, and the use of optimisation methods may be the only available approach for finding satisfactory solutions, even if no optimal solution can be obtained. [1, p. V–VI]

This emergence of optimisation as a paradigm has shifted the way problems may be approached [1, p. 1–2]. Previously, solving problems would typically have involved breaking the problem into subproblems, and then manually finding the optimal parameters for each subproblem. With the optimisation methods available to us, the problem as a whole may now be approached. The problem can be defined in terms of the desired results, and an optimisation method can then be used to construct a function that lets one arrive at such results. [2]

While this shift of paradigm does not reduce the need to understand the related specifics of the problem's domain, it lets us shift the focus when solving the problem. Manually finding the best set of parameters for a complex problem with multiple parameters can be very hard and laborious, if not impossible. With optimisation, we can define the desired output for a model or a parametrisable process, and then proceed to finding the parameters that produce the smallest error compared to the desired output [3, p. 1–3].

At the time of writing, applying optimisation to music and audio signal processing problems still appears to be relatively uncommon. The applications typically involve filter design [4], onset detection [5], source separation [6] and audio classification [7], to name a few. Noteworthy is that methods which are not reliant of computational optimisation are still routinely used in audio signal processing, and optimisation is often used to improve these existing methods instead of replacing them. One of the reasons behind this is that audio processing is inherently hard. Audio signals are both additive and oscillatory by nature [5], and extracting information from them the way human hearing does is very complicated (see for example [8] for a comprehensive introduction). It can therefore be argued that further research into optimisation of audio-related problems is needed to gain better understanding on how the domain can be approached.

1.2 Goal of the thesis

This master's thesis is a study into optimising a problem that is related to audio signal processing. The goal is to apply computational optimisation to create a linear-time variant window function for discrete-time signals. The window function is designed to have the shape of a Gaussian function and appear narrower for higher frequencies. See Fig. 1 for an illustration on how the desired window function affects different frequencies. For the background and motivation for such a window function, see Appendix A.

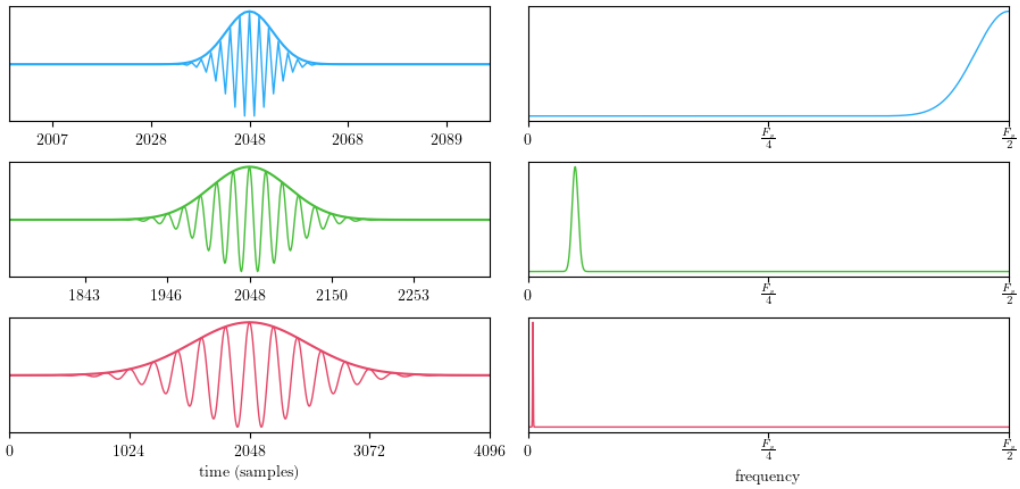

Figure 1
The desired window function for oscillations of different frequencies.

Plots on the left represent the time domain, and the plots on the right are their magnitude spectrum responses.

The window function appears narrower for higher frequencies and wider for lower frequencies. Conversely, the peaks towards the lower frequencies get more spectral resolution in the frequency domain.

The window function in question is implemented by filtering the signal with multiple runs of moving average filters, the length of which vary in time. Differential evolution method is used to approximate the lengths of the moving average filters for each point of the window.

The problem of approximating the moving average lengths is ill-posed by definition, meaning that a perfect solution cannot be expected to be found. This is dictated by the underlying mathematics and the formulation of the problem, where low computational cost is part of the problem formulation. The problem is also stochastic by nature due to

the time-frequency uncertainty principle, as discussed in the Appendix A. Because of this, one goal of this work is to study how the error that arises from the approximation can be spread evenly across the time-frequency plane.

This work comprises the findings of applying optimisation to an audio signal processing problem. Differential evolution is selected as the optimisation method due to readily available resources and documentation. The work seeks to find how the optimisation method should be adapted when working with an audio-related problem, and whether the selected approach is viable for the task. The approach, though tailored for the particular problem at hand, represents the challenges that can be encountered when working with problems that relate to audio processing and time-frequency representations. As such, the majority of this work is focused towards understanding the specifics of the problem, presenting the difficulties that arise from the domain of the problem, and how the difficulties were addressed when adapting the problem for optimisation.

1.3 Structure of the thesis

Section 2 provides theory to the concepts needed to realise the frequency-dependent window function. The optimisation problem is formulated in section 3. Section 4 describes the steps used to adapt the algorithm to the selected optimisation method. Section 5 examines the optimisation process and the findings from the optimisation. Section 6 is the conclusion and discusses the future work.

A part of this thesis is the source code for the developed optimisation program. The source code has been separately shared with the thesis examiners.

2 Theory

2.1 Gaussian function

The Gaussian function, often called just the Gaussian, and also known as normal distribution in the field of statistics, is a function of the form:

$$w(t) = e^{-\alpha^2(t-\tau)} \qquad \begin{array}{l}\text{where } \tau \text{ is the peak in time, and,} \\ \alpha \text{ controls the dilation}\end{array} \tag{1}$$

A Gaussian centred about the time 0 expressed as a function of the standard deviation can be written as:

$$w(t) = \exp\left(-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2\right) \qquad \text{where } \sigma \text{ is the standard deviation} \tag{2}$$

Gaussian function has a number of interesting properties. It is the eigenfunction of Fourier transform, meaning that FT of a Gaussian always transforms to another Gaussian [9]. This is illustrated in the Fig. 2.

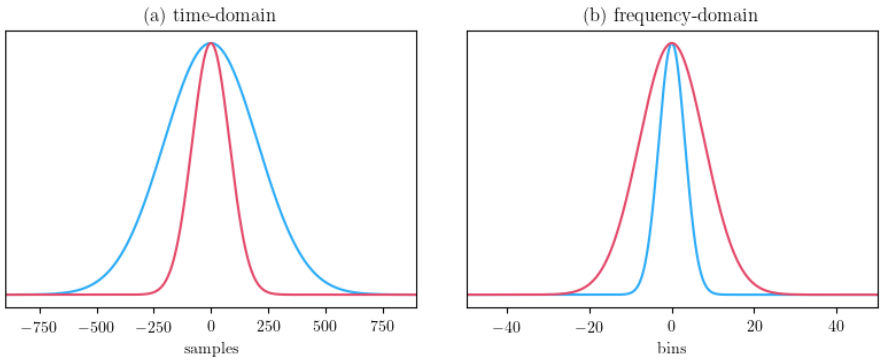

Figure 2
Gaussian functions in the time-domain (a) and their respective magnitude frequency responses (b).

The process of convolving a signal with a Gaussian distribution shaped filter kernel is referred to as Gaussian filtering. When used as a filter kernel, the Gaussian works as a lowpass filter with optimal time-domain properties [10]. It introduces no overshoot or ringing to the filtered signal, with the expense of rather poor frequency-domain slope [9]. A Gaussian lowpass filter can be characterised entirely by its standard deviation σ.

An often used technique in the image processing and machine vision is to approximate Gaussian filtering with multiple passes of a very simple lowpass filter. Based on the central limit theorem, as the number of lowpass filter passes on a signal approaches infinite, the resulting impulse response tends to a Gaussian shape [11]. In general, 4 moving average filter passes already give a satisfactory level of approximation [12].

Since the Gaussian has infinite support, meaning that it is non-zero for its entire range, to use Gaussian as a filter kernel or window function requires the truncation of the side-lobes. This can be done via multiplication with another window function that has finite support of desired length. [12] As the Gaussian tends close to zero quickly, empirically can be found that the window function used for truncation can be a rectangular window if the length is at least 6-8 σ.

Figure 3 illustrates the ringing that unsuccessful truncation of the side-lobes causes, known as the Gibbs phenomenon [12].

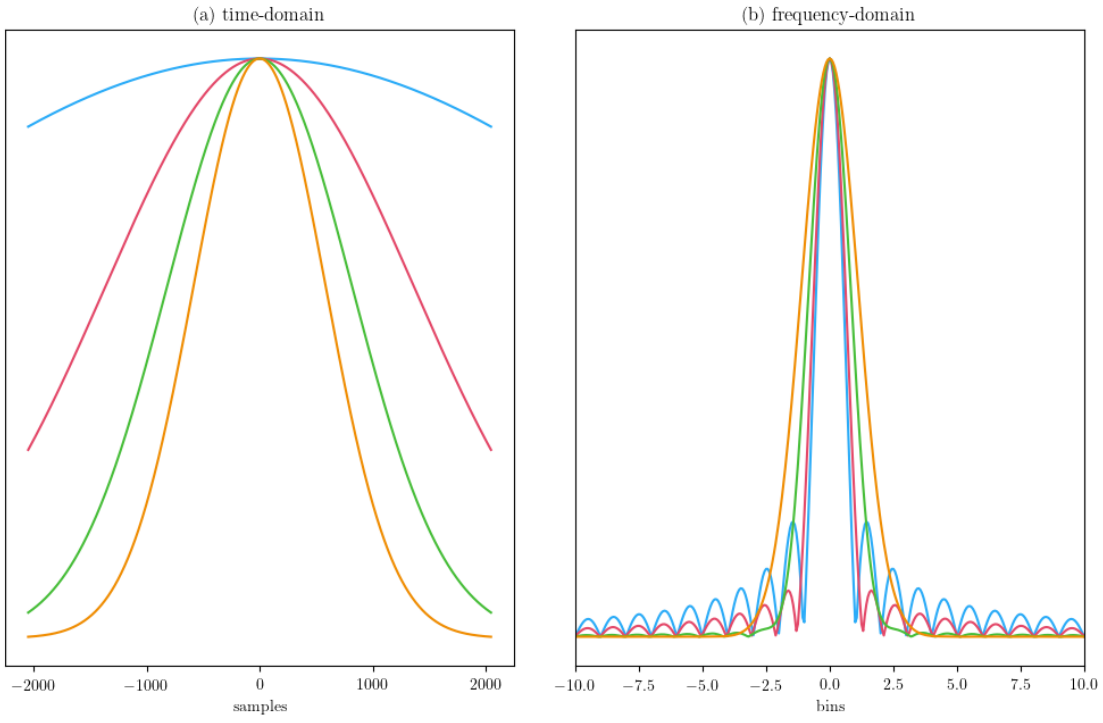

Figure 3
Gibbs phenomenon as the result of truncation. Gaussian impulse responses too wide for the length of the signal vector (a) and their respective magnitude frequency responses (b). The truncation of the Gaussian window causes ringing in the frequency-domain.

2.2 Moving average

Moving average filter is one of the simplest filters available. In continuous time, it corresponds to convolving a signal with a rectangular pulse. In discrete time, it can be expressed as an unweighted sum of the neighbouring samples. A moving average can be made linear-phase by making it symmetric about the centre, and by having the length of an odd integer. [9]

In discrete time the moving average of length L centred around the sample index i can be expressed as [9]:

$$y[i] = \frac{1}{L} \sum_{j=i-L/2}^{i+L/2} x[j], \qquad \text{where } L \text{ is an odd integer} \tag{3}$$

Unweighted moving average is a trade-off between good step response and poor frequency response. It functions as a lowpass filter with the frequency response the shape of a sinc-function. What makes the moving average appealing in the signal processing context is that it is computationally very cheap to implement. A moving average filter can be calculated as a recursive running sum, with each additional output sample costing only the calculation of one addition and one subtraction operation. [9]

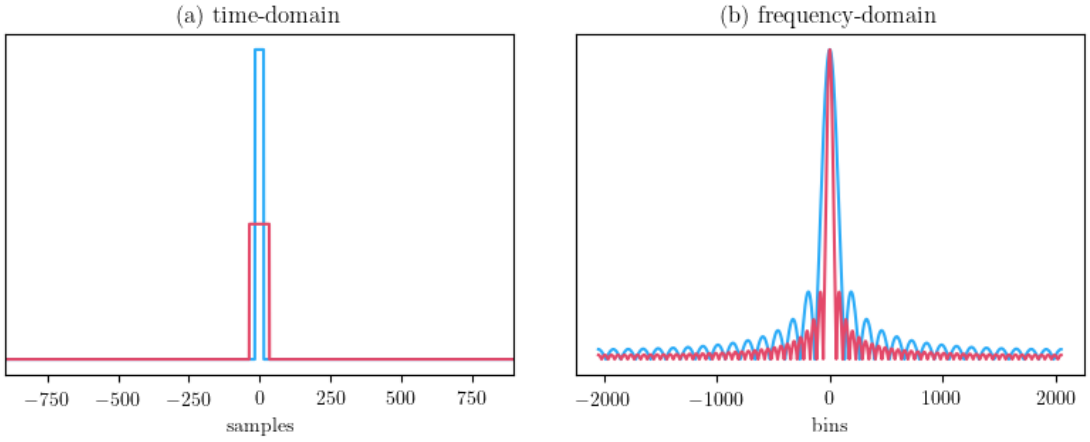

Figure 4
Moving average filter kernels in time-domain (a) and their respective magnitude frequency responses (b).

2.3 Linear time-variant filters

Most real-world filters used in signal processing are stationary, or linear time-invariant (LTI), systems. LTI filter theory forms a major body of work, and a lot of theory is dedicated to the analysis and understanding of such systems. A filter can also be non-stationary, that is, the frequency response can change as a function of time. Such a system is referred to as linear time-variant (LTV) system. The analysis of LTV systems is more complicated due to their non-stationary nature, as the methods typically used to study LTI systems, such as the impulse response and the response, require the system to be stationary [9].

A system is said to be linear if it has the following properties [13]:

Superposition property:

$$\mathcal{L}(\, x(t)\,) + \mathcal{L}(\, y(t)\,) = \mathcal{L}(\, x(t) + y(t)\,) \tag{4}$$

Scaling property:

$$g\mathcal{L}(\, x(t)\,) = \mathcal{L}(\, gx(t)\,) \tag{5}$$

In discrete time, systems can be expressed as a matrix multiplication between the signal vector and the system. If, for simplicity, the signal x is restricted to be of length N, then any linear operation on the signal can be represented as a $M \cdot x$, where M is a $N \times N$ matrix [13].

Filtering can be expressed as convolution between the signal and the filter [9]. A convolution of a signal with a stationary impulse response can be expressed as multiplication between a matrix and the signal vector. This corresponds to a sum of column vectors, where each column is the time-shifted version of the impulse function, multiplied by the corresponding input sample of the signal. To expand this into LTV filters, the time-shifted impulse response of each column can then be changed to be an impulse response with desired momentary properties. [14]

2.4 Differential evolution

Computational optimisation is a significant paradigm in modern-day engineering. As resources are often limited, optimisation becomes an important tool in achieving output and efficiency. [1] This also applies for audio processing, where optimisation can be used

to gain better results with faster computational times. In real-time sensitive applications, optimisation can be used as a trade-off to move computational burden from the runtime to the design of the algorithms.

Differential evolution as an optimisation method

Evolutionary algorithms are a branch of optimisation methods drawing inspiration from evolutionary concepts such as *recombination*, *mutation* and *survival of the fittest*. Due to working with similar concepts, differential evolution (DE) is often considered to belong to this family of optimisation methods. [3, p. 20]

More specifically, differential evolution is a is a metaheuristic minimisation method [15]. Metaheuristics are algorithm frameworks designed to solve complex optimisation problems, helpful in solving problems that include stochastic or incomplete information about the mathematics of the problem [16].

Optimisation problems typically try to find the optimal set of parameters to minimise the value of a so-called cost function, also referred to as objective function or loss function. When selecting the optimisation method for a given problem, one should consider the nature of the objective function, as elaborated in the following excerpt from [3, p. 1–2]:

> " [...]The objective function, $f(x) = f(x_0, x_1, ..., x_{D-1})$, has D parameters that influence the property being optimized. There is no unique way to classify objective functions, but some of the objective function attributes that affect an optimizer's performance are:
>
> - Parameter quantization. Are the objective function's variables continuous, discrete, or do they belong to a finite set? Additionally, are all variables of the same type?
> - Parameter dependence. Can the objective function's parameters be optimized independently (separable function), or does the minimum of one or more parameters depend on the value of one or more other parameters (parameter dependent function)?
> - Dimensionality, D. How many variables define the objective function?
> - Modality. Does the objective function have just one local minimum (uni-modal) or more than one (multi-modal)?
> - Time dependency. Is the location of optimum stationary (e.g., static), or non-stationary (dynamic)?

- Noise. Does evaluating the same vector give the same result every time (no noise), or does it fluctuate (noisy)?
- Constraints. Is the function unconstrained, or is it subject to additional equality and/or inequality constraints?
- Differentiability. Is the objective function differentiable at all points of interest?"

[3, p. 1–2]

Working principles of differential evolution

Differential evolution was initially developed and published by Price and Storn [17]. It imposes few requirements to the function being optimised: the function is not required to be linear or differentiable [18]. Essentially, DE treats the function to-be-optimised as a black box, and the use of the method revolves around designing a cost function that measures how well a given set of parameters, a parameter vector, solves the problem.

DE tries to achieve its results via a parallel direct search stochastic process [18]. Direct search refers to an approach where new solution candidates are tested as part of the optimisation process before they are accepted as new solutions. Direct search algorithms rely less on calculus than they do on heuristics and conditional branches. They are useful when attempting to optimise non-differentiable functions, such as functions with abrupt changes or other conditions, which make the differentiation unpractical. The testing in direct search methods such as DE is referred to as *selection*. The selection separates direct search algorithms from gradient-based methods. In gradient-based methods the generation process is based on an iterative function, and as such every generated solution will be accepted. Compared to this, direct search algorithms have to evaluate whether the new point actually improves the result. [3, p. 12–13]

DE works by maintaining a population of candidate solutions, known as individuals, that are used to probe the search space [18]. New trial vector is created by mutating the existing population. A mutated trial vector is then compared to an existing individual in the population via the cost function. If the new trial vector is an improvement over the existing individual, that is, if the cost or error of the trial vector is smaller, the trial vector replaces the existing individual.

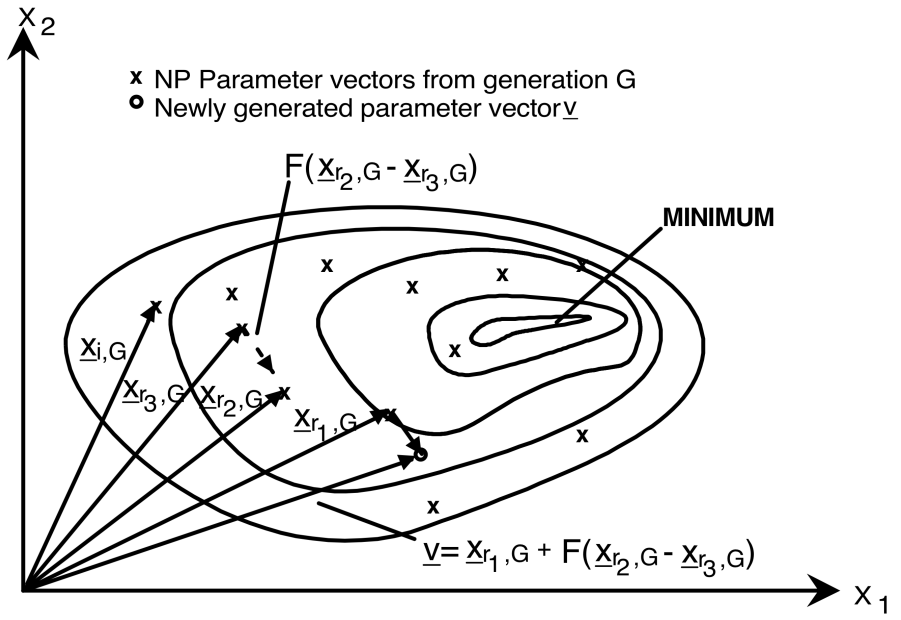

Figure 5
2-dimensional illustration of the generation of a new trial vector $\underline{v}$ by mutation.

The lines in the illustration represent the contour lines of the cost function, with the minimum marked. These can be visualised as a landscape, with the minimum representing the lowest point in the terrain. The optimal solution to the problem would the pair of values x_1 and x_2 that resides in the lowest point; in the illustration this would be represented as a vector pointing directly to the minimum.

In trying to find the optimal solution to the problem, DE mutates existing individuals of the population by recombining them with one-another.
Each individual is a set of parameters in the form of a vector. The new trial vector, here denoted $\underline{v}$, is created by adding the difference of the vectors $\underline{x}_{r_2,G}$ and $\underline{x}_{r_3,G}$ to the donor vector $\underline{x}_{r_1,G}$. Prior to adding, the difference is scaled by the *mutation factor* F.
Source: [17]

Mutation

There are several proposed mutation methods for forming the trial vector [18][4]. In this work the nominal formulation proposed in [17] by the original authors is used:

$v = x_{r_1,G} + F * (x_{r_2,G} - x_{r_3,G})$ where G stands for the current generation,
$r_1, r_2, r_3, \in [0, N_P - 1]$, integer and mutually different,
N_P is the number of individuals in the population, and,
$F > 0$

In this method, three random individuals, $x_{r_1,G}$, $x_{r_2,G}$ and $x_{r_3,G}$ in the population are selected as the parent individuals. The donor vector $x_{r_1,G}$ is mutated by the scaled difference of the two other vectors. As discussed in [3, p. 74–80], if each individual in the population is different, and if $F > 0$ and $F \neq 1$, this creates $(N_P)^2$ difference vectors that are used as the random sampling to ensure enough randomness in the mutation process. The parameter F, or the mutation factor, can thus be a constant. Figure 5 provides an illustration of the forming of the mutation vector.

Crossover

To increase the diversity of the trial vectors further, the DE also introduces another evolution-inspired source of mutation, the concept of *crossover*. In this context it means introducing the chance of mutating only a subgroup of the parameters instead of the whole parameter vector.

In [17], the crossover is formulated as:

$$u_j = \begin{cases} v_j & \text{for } j = \langle n \rangle_D, \langle n+1 \rangle_D, ..., \langle n+L \rangle_D \\ (x_{i,G})_j & \text{otherwise} \end{cases} \tag{6}$$

where D denotes the number of dimensions in the vector
$\langle\ \rangle_D$ denotes the modulo D, and,
the integer L is drawn from the interval $[0, D - 1]$.

Figure 6 provides an illustration of crossover. In [3, p. 92–94], other possible schemes for crossover are also described. In this work, the crossover was adapted to suit the problem formulation, as discussed further in the section 4.3.

As iterations are computed with DE, due to the mutation of the vectors, the population starts to converge around the local minima of the cost function, and finally focus around the global minimum and converge into it. The differential nature, meaning the approach that each new trial vector is constructed from the differences of the existing population

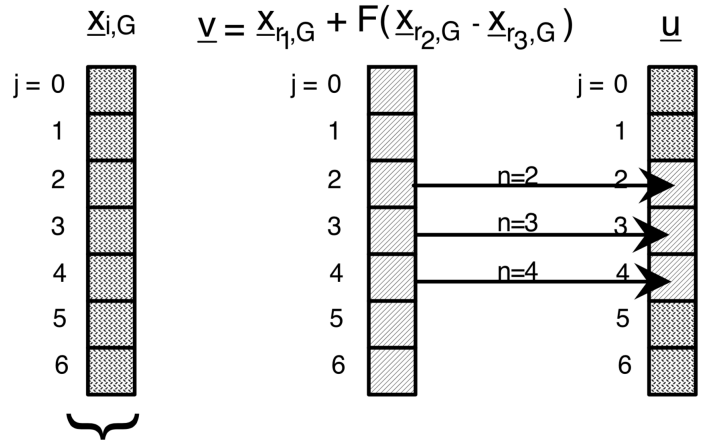

Figure 6
Illustration of crossover. When creating the final trial vector, some of the parameters of the final trial vector $\underline{u}$ may be transferred from the existing individual $\underline{x}_{i,G}$ instead of the mutated vector $\underline{u}$. The existing individual, represented by the vector $\underline{x}_{i,G}$, is the individual that the trial vector will be compared against. The parameters are transferred between the same corresponding position of the vectors, for example the parameter from index 3 of $\underline{x}_{i,G}$ to the index 3 of $\underline{u}$.

In this illustration, the mutated vector $\underline{v}$ only transfers parameters 2, 3 and 4 to the trial vector $\underline{u}$, and rest of the parameters are transferred from the individual that the trial vector will be compared against.
Source: [17]

---

vectors, leads to the situation where the trial vectors of the converging population become grouped. There is no need for additional parameters to control the step sizes the algorithm takes while converging, as the search vectors shorten automatically for a more fine-grained search. An excellent illustration of this is available in [3, p. 44–47].

2.5 Overfitting

Overfitting is an often encountered problem with computational optimisation and machine learning methods. If the system being optimised has too many options available to it compared to the test signal set, it will tend to overfit to the data in the test signal set. [19] In practice this means that the optimised data model starts to learn the individual features and "memorise" the test signals instead of finding a general solution [2]. Although overfitting is often discussed in the context of neural networks, many of the proposed techniques and principles can be applied to be used with differential evolution.

Several good practices have been proposed to address the problem of overfitting. Firstly, care should be taken that the test set is large enough compared to the complexity of the learning model. With too small a test set the learning model may provide good results with the tests, but fail to generalise to a broader range of signals. The test set should also include enough samples of all the possible variations of the problem. [2]

The performance of the model should be evaluated with different set of tests than the ones used to train it. This helps to ensure that the resulting solution provides a good general solution. [2][19]

Finally, an additional regularisation term may be used in the cost function. In the context of neural networks, regularisation is utilised to limit the growth of the network. This is done by imposing a cost to the operations in the network, essentially guiding the model to use fewer nodes or smaller weights in its operation. Large weights in neural networks can often result in more sensitivity to specific test signal features, at the expense of generalisation. [2] The details on how the regularisation is applied in the context of the this work are discussed in the section 3.5.

3 Formulating the problem

To find a solution to a computational problem, it is helpful first to analyse and divide the problem into subproblems. In this section, the cost function for the proposed differential evolution optimisation approach is outlined. Section 3.1 presents the outcome of the formulation, and in the subsections that follow the terms of the cost function are defined.

3.1 Cost function

At the core of using an optimisation method is designing a cost function. In general, a cost function is a function that may accept multiple parameters $\hat{\mathbf{L}}$, and should produce a single real value that represents the error with those parameters. In other words, the cost function $C(\hat{\mathbf{L}})$ represents the problem, and is then attempted to minimise. By doing this, we hope to arrive at the optimal set of parameters. Thus, care should be taken to make sure that the cost function represents the problem fully.

In this section, the cost function used in this work is derived. The final cost function is of the form:

$$C(\hat{\mathbf{L}}) = \left\| T(\hat{\mathbf{x}}) - \mathbf{H}(\hat{\mathbf{L}}) \cdot \hat{\mathbf{x}} \right\|^2 + R(\hat{\mathbf{L}}) \tag{7}$$

where $T(\hat{\mathbf{x}})$ is the known target that we're attempting to approximate,
$\mathbf{H}(\hat{\mathbf{L}})$ is the LTV system we're seeking to produce,
$\hat{\mathbf{x}}$ is the test signal, and,
$R(\hat{\mathbf{L}})$ is the regression error for the parameters.

The goal is to develop a fast computational approximation of a Gaussian window that appears relatively narrower for higher frequencies and wider for lower frequencies. The aim is to use the produced window function to determine the momentary spectrum of a signal.

The cost function produces the error by calculating the L^2-norm of the difference vector. This is also known as the least squares error, and is denoted as $\|\cdot\|^2$ in the Eq. 7. The L^2-norm is calculated by summing together the square of each element in the vector, and corresponds to the dot product of a vector with itself. For complex number vectors, the L^2-norm is defined as the sum of each element multiplied by its complex conjugate.

We want to measure the error as the spectrum of the difference of the known target and

the output produced by the LTV system. This could be achieved with the use of discrete-time Fourier transform. However, computation cost involved with Fourier transform is heavy compared to the rest of the cost function. Based on Parseval's Theorem [20], the L^2-norm of the spectrum equals to the L^2-norm of the time domain signal. Hence, to evaluate the L^2-error of the spectra of $T(\hat{\mathbf{x}})$ and $\mathbf{H}(\hat{\mathbf{L}})$, it is sufficient to evaluate the L^2-error of their time domain representations.

In the following subsections, the terms of the cost function are developed.

3.2 Target function $T(\hat{\mathbf{x}})$

The target function $T(\hat{\mathbf{x}})$ is the optimal solution of the frequency-variant window function. It can be defined as the sum of oscillations, each windowed individually to the desired width. The target function can be defined for the signal vector $\hat{\mathbf{x}}$ as:

$$T(\hat{\mathbf{x}}) = \sum_i G_i \exp\left(\frac{j2\pi t f_i}{F_s} + j\phi_i\right) \exp\left(-\frac{1}{2}\left(\frac{tm}{F_s f_i \sqrt{\ln 2}}\right)^2\right) \tag{8}$$

The function and its variables are derived in Appendix B.

The equation 16 represents a weighted sum of oscillations, each windowed with a Gaussian window of desired width to match the oscillation. The width of each Gaussian window is directly related to the wavelength of each frequency, and can be controlled with the variable m. The value of m used in this work is 2, as it leads to having $2m = 4$ wavelengths of the oscillation between the -3dB points of the window. Depending on the need, the value of m could be readily adapted to suit the use-case.

3.3 Moving average system $\mathbf{H}(\hat{\mathbf{L}})$

The moving average system $\mathbf{H}(\hat{\mathbf{L}})$ represents the algorithm that is to be optimised. The function $\mathbf{H}$ is a LTV system of the form:

$$\mathbf{H}(\hat{\mathbf{L}}) = H_1(\hat{\mathbf{L}}_1) \cdot H_2(\hat{\mathbf{L}}_2) \cdot \ldots \cdot H_M(\hat{\mathbf{L}}_M) \tag{9}$$

where M is the number of moving average filter passes. Each H_i is a matrix representing a single moving average pass over the signal when multiplied with the signal vector $\hat{\mathbf{x}}$. Each H_i has size $N \times N$, where N is the length of $\hat{\mathbf{x}}$.

As an input the system $\mathbf{H}(\hat{\mathbf{L}})$ takes a parameter vector, here denoted $\hat{\mathbf{L}}$, which holds the point-wise lengths for all the subsequent moving average runs. Each $L_{\text{moving average},i}$ in

$$\hat{\mathbf{L}} = [L_{1,1}, L_{1,2}, \ldots, L_{1,N-1}, L_{1,N}, L_{2,1}, \ldots, L_{2,N}, \ldots, L_{M,N}]$$

corresponds to the length of the moving average filter at index i.

The matrices H_i have the form:

$$H_i = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \cdots & w_{0,N-1} \\ w_{1,-1} & w_{1,0} & w_{1,1} & \cdots & w_{1,N-2} \\ w_{2,-2} & w_{2-1} & w_{2,0} & \cdots & w_{2,N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N-1,1-N} & w_{N-1,2-N} & w_{N-2,3-N} & \cdots & w_{N-1,0} \end{bmatrix} \tag{10}$$

$$\text{where:} \qquad w_{i,j} = \begin{cases} \frac{1}{L_i} & \text{if} \quad -\frac{\lambda_i - 1}{2} \leq j \leq \frac{\lambda_i - 1}{2} \\ q\frac{1}{L_i} & \text{if} \quad j = -(\frac{\lambda_i - 1}{2} + 1) \quad \text{or} \quad j = \frac{\lambda_i - 1}{2} + 1 \\ 0 & \text{otherwise} \end{cases}$$

This formulation is derived in the Appendix C.

As an example, the H_i for a signal vector with the length of 8 could be:

$$H_i = \begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{16} & 0 & 0 & 0 \\ \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{12} & 0 & 0 \\ 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 \\ 0 & 0 & \frac{1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{8} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Each column represents a moving average filter, and each moving average is weighted to avoid changing the energy of the filtered signal. The sum of each column is 1 – the weighting is not changed even if the moving average cannot fit the column to reduce anomalies at the edge regions of the filtered signal. The samples at the edges of the odd-length moving average kernels may be used to approximate the in-between odd lengths of the moving average, as discussed in Appendix C.

Based on the central limit theorem, as the number of moving averages M approaches infinity, the $\mathbf{H}\hat{\mathbf{x}}$ may become equal to the target function $T(\hat{\mathbf{x}})$. For the computational

approximation, M is restricted to a sensible number. The number of M represents the trade-off between the quality of the approximation and the computational cost of the algorithm. Based on empirical observations the M should be at least 4.

3.4 Test signal $\hat{\mathbf{x}}$

Optimisation methods have a tendency to overfit to the test signal set at hand if care is not taken to provide the optimisation process with a diverse enough test set.

When working with differential evolution, the preferred method would be to provide the optimisation process with a fixed input and target signals that all iterations would be compared against. However, a global, all encompassing test signal for this problem cannot be created. The perfect test signal would be one with all frequencies uniformly distributed at every position of time. The dual nature of oscillations prevents this: an oscillation will always take place in both time and frequency.

Because of this, the test signal should vary between iterations, and stochastically represent all the possible frequencies. Additionally, the desired properties of the system-to-be-optimised are only known for individual oscillations. As the system-to-be-optimised is linear, a test signal can be a sum of multiple individual oscillations.

Computational efficiency becomes a factor when iterating the process over millions of iterations. With a window function that is aimed towards audio processing applications, the preferred test signal set would consist of real segments of recorded audio that are processed to provide both the input signal $\hat{\mathbf{x}}$ and the known target $T(\hat{\mathbf{x}})$. Calculating such a test signal set would be computationally intensive, and also introduce the questions of how the sound sources should be selected. The signal set should also be of considerable size to avoid overfitting. As a result this would make the data set very cumbersome to operate on in terms of memory usage, or could potentially create a bottle neck to the performance of the optimisation process as each signal would have to be read from the hard drive.

To avoid the complications that the real-world test signal set would introduce, each test signal and its corresponding target are synthesised on the fly as a sum single oscillations.

Synthesizing the test signals

A number of considerations are taken into account when synthesising the test signals:

1. The test signal is changed periodically, and is formed so that the frequency distribution of the sum of all test signals has a desired property. In the final implementation virtually every iteration of the cost function is compared against a different test signal.

2. The average of all test signals should have a desired distribution in the frequency-domain. The frequency distribution determines how the error is spread across the frequency spectrum. In the final implementation, each frequency is determined by:

$$F_s \; 2^{r_f} \tag{12}$$

   where r_f is a uniformly distributed random number in the range $[\, \log_2(2/N),\ \log_2(F_s/2)\,]$.

   The range of the synthesised frequencies is selected to be between $2/N$ and $F_s/2$. The higher limit is the Nyquist frequency, which is the highest frequency that can be represented with any sampling rate. The choice of the lower limit is based on the length N of the signal vector. $N/2$ can fit two wavelengths of a frequency to the signal vector. As discussed in the Appendix B, this corresponds to a Gaussian window with the -3dB points at the edges of the signal vector.

   Selecting the frequencies based on equation 12 results in a fairly uniform distribution of frequencies in the target signal, as depicted in the figure 7. This lets the error to be uniformly distributed along all frequencies of the spectrum, which based on empirical studies is desired when using the L^2 error norm.

3. To produce signals with more resemblance to real-life audio signals, each oscillation is scaled with a random level uniformly distributed in the range of -90dB to 0dB, and each oscillation has uniformly distributed random phase.

4. Each test signal consist of at least N summed oscillations, where N is the length of the window function.

5. Each cost function computation consists of multiple cost functions with different test signals, with their errors averaged. This lessens the chance of an overfitted individual entering the population.

6. To keep the average error levels steady, each pair of input signal and target signal is normalised in respect to the RMS of the input signal.

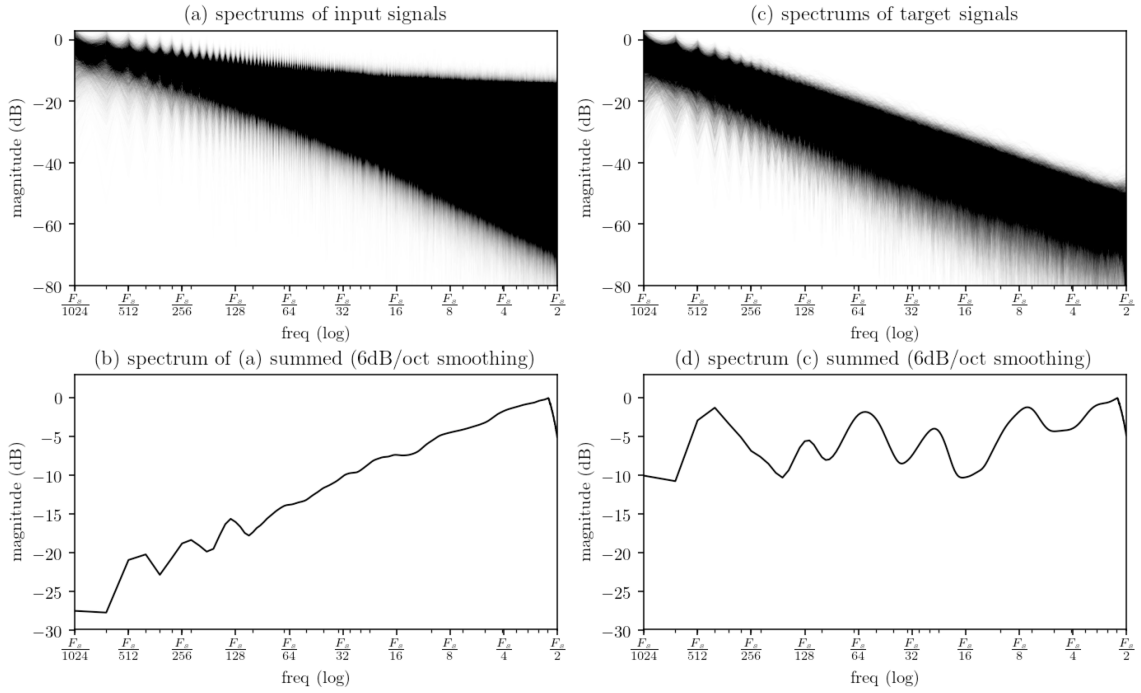

Figure 7
10 000 test signals, each a sum of 100 000 oscillations. Figures (a) and (c) depict input and target spectra respectively. Figures (b) and (d) depict the sums, representing the average frequency response of input and target signals.

It should be noted that generating another cumulative distribution of test signals with different random seed aligns the notches and peaks in the subfigures (b) and (d) in different locations. The distribution in subfigure (d) can therefore be considered close to flat.

3.5 Regularisation term $R(\hat{\mathbf{p}})$

Motivation for the use of a regularisation term

Regularisation error can be implement by adding an additional term to the cost function. The regularisation term should increase as the complexity of the model increases to limit the complexity of the model. In this work, a regularisation term is introduced to prevent overfitting to any particular test signal, and to aid in converging to a computationally fast solution.

With a constant length moving average, every new sample calculated with the moving average requires two arithmetic operations: adding the next sample to the running sum, and subtracting the last sample from the running sum. To penalise the optimisation process for any extraneous operations, and to keep the computation times of the final

parameters low, a cost is attached to every operation in the moving average that modifies the running sum. This encourages smoothness in the progression of the lengths, making the process converge towards solutions that are fast to compute.

Regularisation also prevents the moving averages from specialising to any specific frequency or position of the test signals. The proposed method will always have some error to it due to the limited number of moving average runs. Without the regularisation, the algorithm could potentially optimise towards minimising the error for random single oscillations, with the expense of a less general solution. The enforced smoothness in the length parameters avoids this by restricting the back-and-forth oscillation of the moving average lengths.

Formulation of the regularisation term

The formulation of the regularisation term used in this work is:

$$R(\hat{\mathbf{L}}) = \text{regularisation error} = \begin{cases} \frac{(n_{\text{ops}} - 2N_tM)^2}{N} & \text{if } n_{\text{ops}} > 2N_tM \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Where n_{ops} is the number of total operations for all subsequent moving average runs,
N_t is the length of the test signal, here $N/2$, and,
M is the number of moving average runs.

A moving average, when implemented, can be viewed as two running endpoints of the running sum. A moving average of constant length will take $2N$ operations to produce N filtered sample values. A time-dilating moving average will have as many operations per run as a constant length moving average would have. This is assuming that both endpoints have to run for a set length, here N_t.

In this formulation, effectively no regularisation error is added if the number of operations for a given time-dilating moving average is below that of a constant moving average. The main goal is to produce an algorithm with good approximation: It is sufficient if the final algorithm can perform with the same number of operations as a constant length moving average would. If a candidate moving average requires more operations than a constant-length moving average would, it is be punished quickly by the squared error of the regression term.

4 Optimising the problem with differential evolution

In this section, the steps that were made to adapt the problem to using differential evolution are discussed.

4.1 Motivation for the use of differential evolution

The problem formulation in section 3 reveals many elements that support the use of differential evolution as the optimisation method. Referring to the excerpt in section 2.4, the problem incorporates the following aspects that should be considered:

1. The parameters in the problem have strong dependency. Each subsequent moving average run relies on the previous runs, and the resulting point-wise frequency response is a function of all of the runs together. Because of this, optimising the problem one moving average at a time, or one position (or index) of the signal at a time, is likely to result in non-satisfactory results. As each moving average filter has an effect on the signal used by the subsequent moving average runs, the results for any set of parameters should be analysed after all of the runs have passed.

2. The problem can be considered noisy. As discussed in the section 3.4, a perfect test signal is not available, as a signal cannot contain all frequencies along the full length of the signal. Instead, the cost function should be evaluated with test signals that are different for every evaluation. Thus, evaluating the cost of parameters multiple times should always result in different cost, but with the better solutions providing smaller cost on average.

   Due to the noisiness, approaches that incorporate the best current individual as the basis for the generation of new individuals should be avoided. The best individual at any time can not be guaranteed to be the best basis for new individuals, as it may be a result from an unlucky overfitting between the individual and the test.

3. The problem is non-differentiable. The truncation-operator in the arbitrary-length moving average formulation causes non-continuity in the frequency responses of the moving averages as a function of moving average lengths.

Based on these observations, the differential evolution presents itself as a viable method to use, as it can be adapted to work around these aspects of the problem.

4.2 Parameters

As discussed in section 3.3, the moving average system $\mathbf{H}(\hat{\mathbf{L}})$ works by using a set of parameters as a parameter vector. The parameters in the vector are interpreted as moving average lengths by the algorithm, and used to process the audio signal vector.

Two properties of the parameter vector should be considered.

Search space size and the parameter ranges

The valid range for each parameter should be defined. The range determines the size of the search space, which affects the convergence speed. The search space should allow any possible meaningful combination of parameters, but larger sizes of search space are slower to operate on.

The *minimum* length for a moving average in discrete time is 1, since this reduces to a unit impulse function δ. In this work, the formulation of moving average can only represent time-units equal to or longer than the unit impulse. The unit impulse does not affect the input, as it is the identity operator of convolution. Thus, the minimum length can safely be restricted to 1.

The *maximum* length is restricted to the signal length N. Moving average longer than N would not bring in any new meaningful addition to the search space, as moving averages with length of N or more equal to summing all available samples in the signal.

Size of the parameter vector

The size of the parameter vector plays a crucial role in how well and fast the optimisation can work. The more parameters are needed to represent the problem, the slower and harder it will be to optimise the problem due to the amount of combinations available.

It can be expected that in the best available parameter vector, all the moving averages have the value of 1 at time index $t = N/2$ of the window. This follows from the formulation of the problem. The centre of the window is assumed to pass all frequencies equally, and thus equal to a unit impulse. As a result, the window function can be considered symmetric about the centre index $t = N/2$. The final window function can be constructed from two segments, with the first one mapping the lengths $[1, L]$ to the indices $[1, N/2]$ and the second one being inverted in time and mapping the lengths

$[L-1, 1]$ to the indices $[N/2+1, N]$. This effectively lets us solve only half of the parameters needed by the final window function. The solved lengths for the first segment can then be reused symmetrically to compute the second half of the window.

4.3 Adaptations to the differential evolution

Some adaptations were made to the nominal DE method [17] to make it more suited to the problem at hand.

Jitter instead of a constant mutation coefficient F

The mutation coefficient F can be made a random variable, referred to as jitter [3, p. 80–87]. Tests presented in the source would indicate that with large population sizes using the jitter, instead of a constant mutation factor, potentially helps DE to converge faster. This alternative implementation was used in this work. The jitter is implemented as normally distributed random deviation with mean 0.5 and standard deviation $\sigma = 1/7$. This keeps most of the random variables drawn from the distribution between the range $(0, 1)$. If the random variable drawn is outside of this range, a new value will be redrawn until the value is in the range. A new random number is drawn for every mutated parameter.

Adaptations to the crossover

In the proposed scheme, the parameters are grouped as sets of lengths for a single moving average, as discussed in section 4.2. To make the crossover more meaningful, it is performed on these subsets of the parameters. The crossover operation is performed individually for parameters representing each moving average, and the crossover always transfers the parameters from the same corresponding moving average run. For example, the second moving average run of the existing vector crosses over to the second moving average run of the trial vector, and so forth.

Adapting the cost function to compare against the same test signal

The third adaptation to the original method is due to the stochastic nature of the test signal. The test signal changes with every iteration of the cost function, making the error values non-comparable with previously calculated ones. The same cost function with the

same test signal should be performed for both the trial vector and the individual that is at risk of being replaced. This reduces the chance of introducing overfitted individuals to the population. This also makes replacing such overfitted individuals easier, as they are more likely to perform worse in subsequent iterations against new trial vectors.

4.4 Initialisation

The DE method proposes initialising the parameters of each individual with uniformly distributed random values in the range of the search space [17]. In practice, this leads to very long convergence times with this particular problem. A priori is known that the regularisation term in the cost function would make such parameter vector have very high cost. A set of randomly distributed moving average lengths is also slow to compute, as it cannot benefit from the fast running sum implementation. Initialising with uniform random numbers would thus make the initial convergence of the problem very slow.

Also suggested in the nominal paper [17], if a preliminary solution is available, the initial populations can be arranged around that with deviations distributed by normal distribution. An initial guess can be calculated with the assumption that the subsequent moving average runs are identical. Such an initial solution is derived in Appendix D, and is:

$$L_{\text{guess}} = t \sqrt{\frac{12}{M \ln 2} + 1}$$

where L_{guess} is the length of the moving average at time t,
t is the parameter index, and,
M is the number of moving average runs.

The initial population is arranged around these L_{guess}, with the normally distributed deviation applied to it. To introduce further deviation from the initial guesses, a drift is applied to the L_{guess}. The drift is implemented as normally distributed random walk, and applied as:

$$L_{\text{final guess}} = L_{\text{randomly deviated guess}}(1 + c_{\text{drift}}) \tag{14}$$

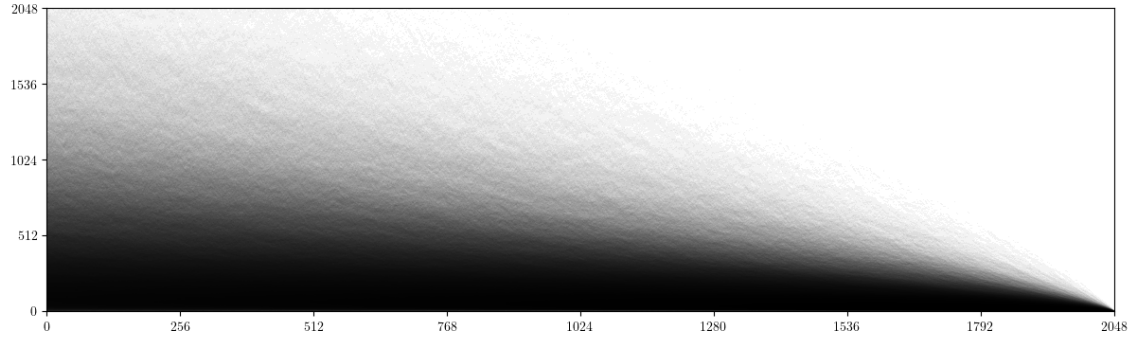

Figure 8
All individuals of the initial population for a single moving average run.

4.5 Parallelisation

To achieve reasonable computation times for the method, the parallelisation of the DE algorithm becomes a major consideration. In general, the DE method lends itself well to parallel computation, where the computation is performed by a network of computers simultaneously, due to the population-based approach [17].

The moving average system cannot be parallelised: Every new output sample of the moving average depends on the previous samples, and every level of the multiple moving average runs may depend on the entire previous level of the transformation. Thus, the parallelisation is hard to realise at the cost function level.

The parallelisation method utilised here is based on a technique proposed in [18]. The population is divided into subpopulations, and each subpopulation is allocated its own process. The concept of migration is introduced, where the subpopulations exchange individuals. In the proposed technique, the subpopulations are arranged into a ring, each migrating their individuals to the next subpopulation. A migration constant $\phi \in [0, 1]$ is defined. For each generation, a uniformly distributed random number r in the interval $[0, 1]$ is drawn. If $r < \phi$, the best individual is migrated from each subpopulation to the next one, based on the ring topology. In the target subpopulation, the migrating individual replaces a random non-best individual. The source [18] suggests the use of $\phi = 0.5$, which was adopted for the implementation.

This technique was adapted to suit the problem. Since the cost function changes constantly, as discussed in the section 3.4, the population may not have a single best solution at any time. Instead, the migration is performed from random individual of the source subpopulation to a random individual of the target subpopulation. To avoid

replacing a better individual, the cost is calculated using the same test signal for both the migrating and the existing individual. The migration will then only takes place if the error of the migrating individual is better than in the existing individual. This is in line with the [3, p. 80] regarding the convergence of evolutionary algorithms. Including selection as part of the migration operation is elitist, as the migrating individual is unlikely to replace a better individual in the target population.

5 Results

The optimisation problem was implemented as a standalone application written in C++. Albeit more laborious, this approach was selected to avoid the additional overhead that a higher level implementation would have introduced. The optimisation process of this size is computationally heavy to run, and the efficient low-level implementation helped to reduce the computation time and costs involved in running the optimisation process.

The implementation is based on the implementation of DE developed by Magnus Jonsson and Olli Niemitalo [21]. The implementation by Jonsson and Niemitalo was heavily modified to suit this particular problem, and provided an excellent starting point for the modifications. The optimisation was carried out on a Google Cloud Compute instance involving 96 computational cores with the task split between them as subpopulations. The computation ran for approximately 210 hours, using over 20.000 processor hours. For the reference, at the time of writing this equates to approximately US$640 on the Google Cloud Platform as on-demand computation.

The optimisation was performed for the window size of 4096 samples, effectively requiring 2048 parameters per moving averages filter run. The number of subsequent moving average filter runs was set to 4, resulting in total of 8192 parameters to optimise.

The population size was constant throughout the optimisation process, as is typical for the DE method. The selection of the population size depends on the problem and on the number of parameters involved, and no set rules apply to this. As a rule-of-thumb, at least 10 times the number of parameters is generally suggested. The size of the population was evaluated with smaller window sizes prior to the final optimisation job. For the final optimisation job a total population size of 98304 was selected, which corresponds to 12 times the number of parameters. This was divided into 96 subpopulations, so that each subpopulation contained 1024 individuals.

5.1 Studying the converging population

The following figures 9 – 13 plot the evolution of the population during the optimisation. Each of the subfigures corresponds to the lengths of a subsequent moving average filter run. The final frequency-variant window function is a combined response of four moving average filter runs. As such, each individual is a combination of one line in each of the subfigures. The order of the runs is from top to bottom with the topmost subfigure in blue representing the lengths for the first moving average run. Note, that from these plots it cannot be interpreted which four lines, one in each subfigure, comprise of one

individual.

In differential evolution method, the existing individuals compete against newly created ones, referred to as trial vectors. The trial vectors are created by mutating the existing population as described in 2.4. If the newly created trial vector is an improvement over the existing individual, the trial vector will be accepted into the population, replacing the existing individual. As the optimisation process progresses, the non-optimal combinations of moving average lengths will be discarded one by one, and the population starts to converge towards the best solutions. The best solutions are then narrowed down to just a handful of candidates, and eventually all of the individuals should be grouped very closely together.

The convergence can be observed on a generation level. One generation has elapsed, when each individual in the population has been evaluated against an equal number of created trial vectors. In the final optimisation job, one generation corresponds to 98304 iterations.

The convergence is illustrated in the following figures.

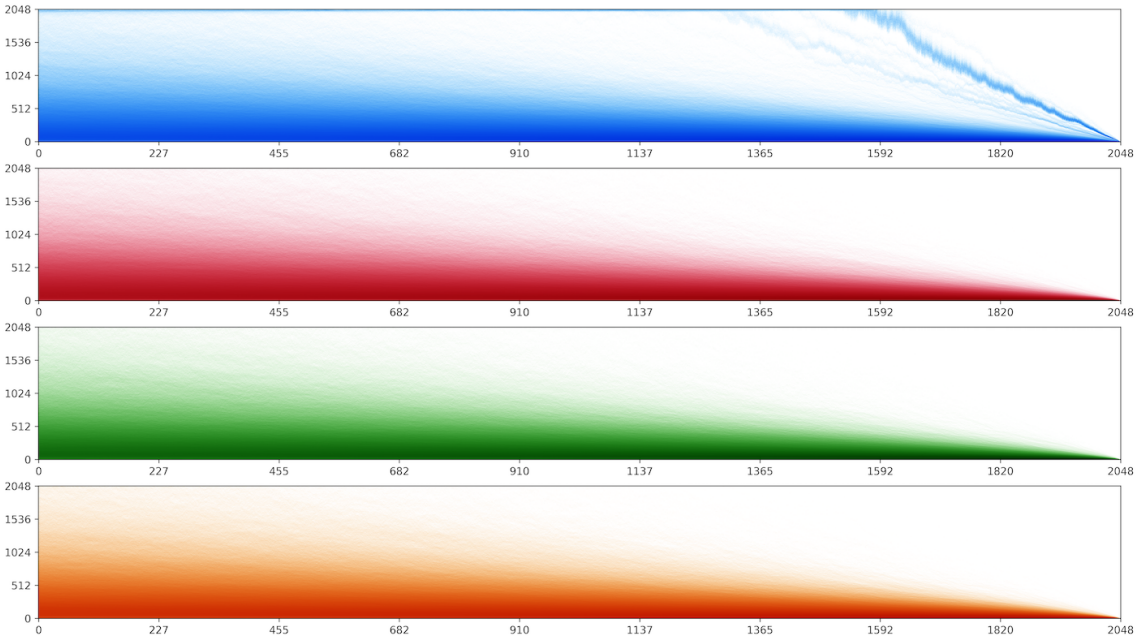

Figure 9
Approximately 225 generations (22 million iterations)
The population is still very spread out, and resembles the distribution of the initial population.

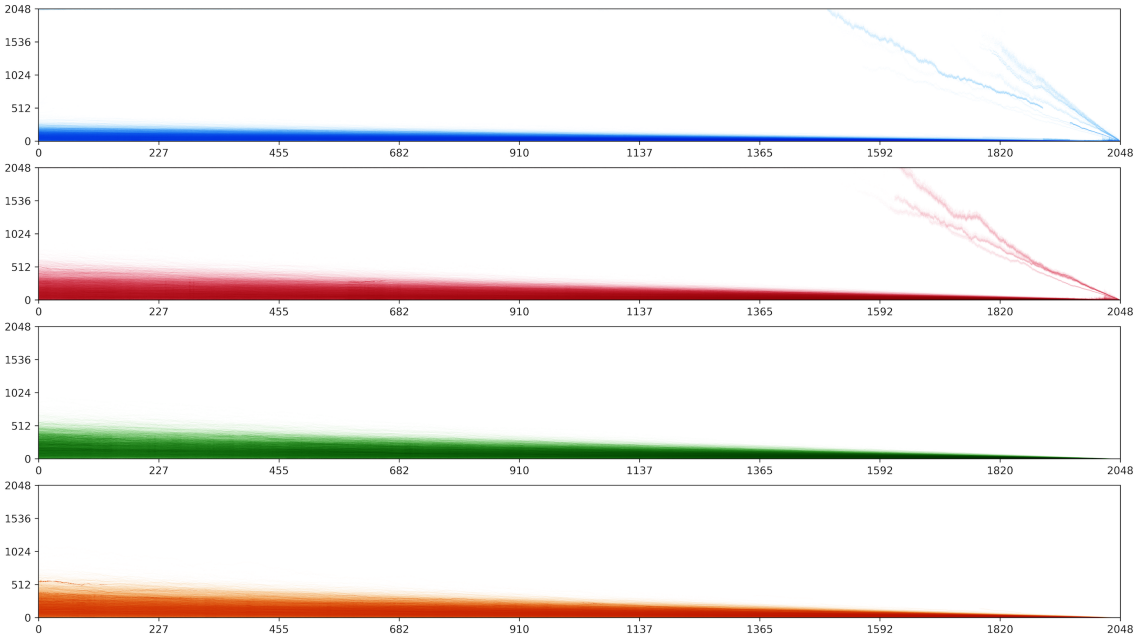

Figure 10

Approximately 2340 generations (230 million iterations)
The longest moving average lengths have already been discarded, and an established range of optimal solutions towards the lower part of the subfigures starts to form.

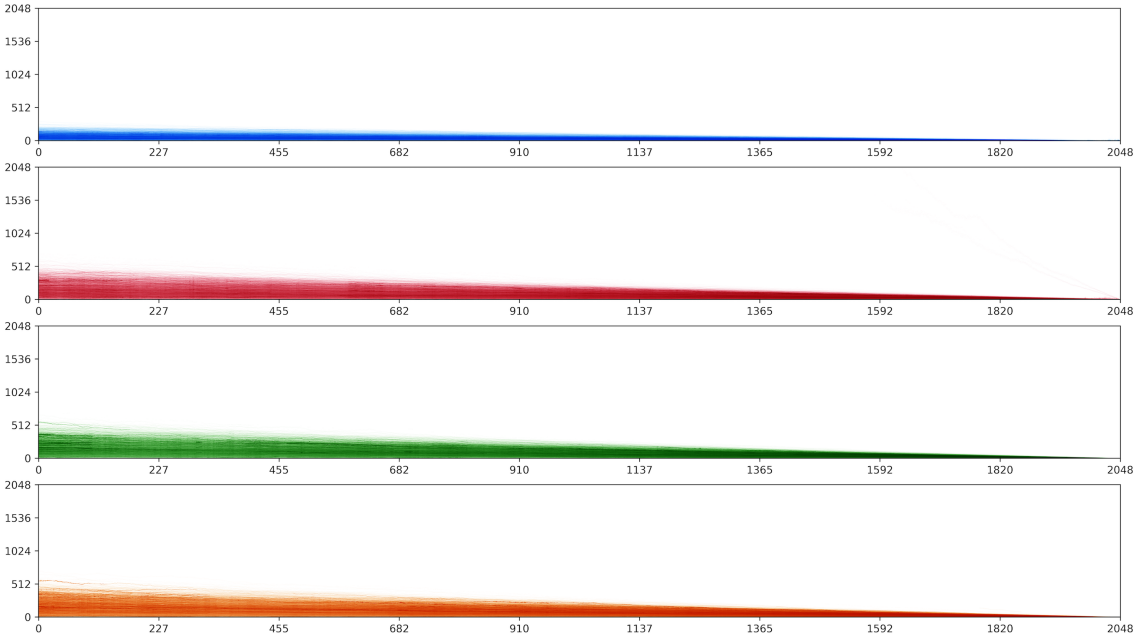

Figure 11

Approximately 4000 generations (390 million iterations)
The steeply declining ripples visible in the prior plots have been eliminated, and darker segments of concentrated individuals start to appear. These are formed as many individuals focus around a well performing combination of moving average lengths. The number of individuals deviating from the established range that started to form in the previous figure is already low.

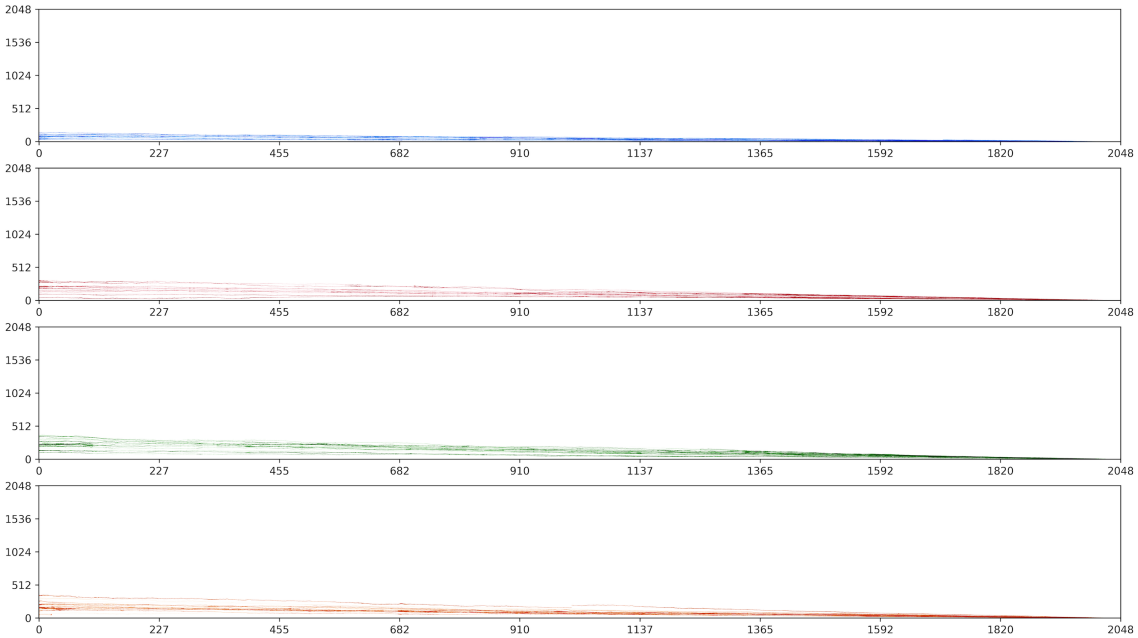

Figure 12

Approximately 7200 generations (710 million iterations)

The local minima of the cost function have been found. Most of the candidates are now concentrated towards a few well performing lines. These are the local minima that perform best compared to the surrounding areas. The DE method ensures that the space between the local minima is routinely probed by new trial vectors, but unless a new improvement is found, these candidates are discarded.

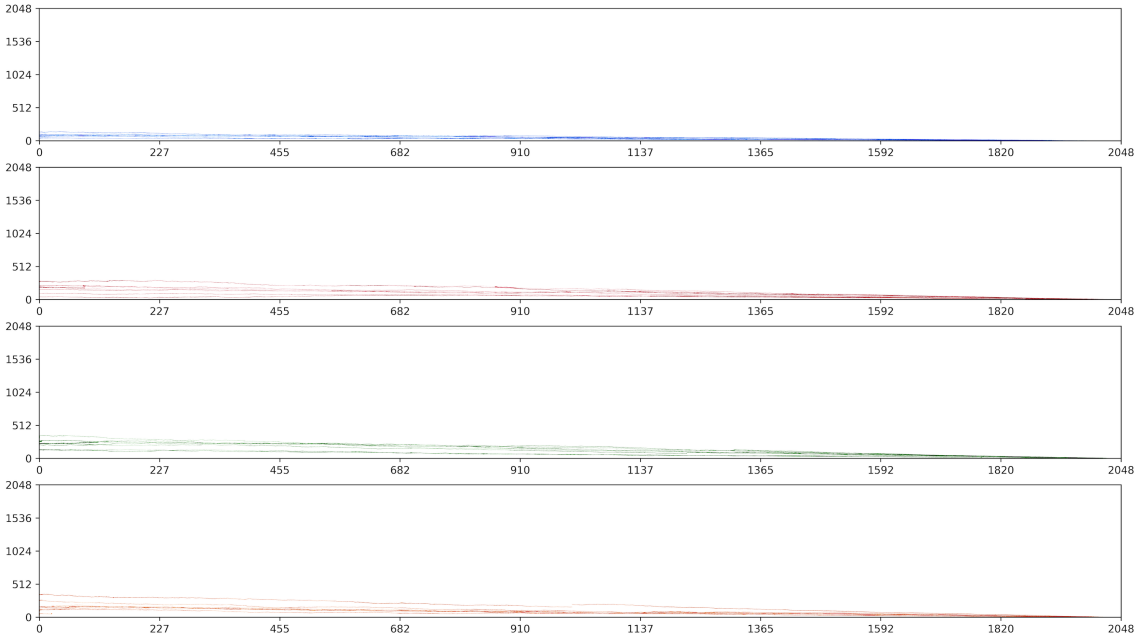

Figure 13

Approximately 13100 generations (1300 million iterations)

Comparing to the situation in Fig. 12, some of the lines representing the local minima have been eliminated.

Averaged error of the population

The convergence can also be observed from the averaged error of the whole population. The average error represent the rate at which the population converges. Since the initial population at the start of the process is based on an established guess instead of uniformly random vectors, the best values are already within a magnitude of the final error at the start of the process. The converging population is then used to improve on these guesses, and to ensure that the search space is fully explored.

The figure 14 presents the average error of the population as a function of generations computed. For this optimisation process, the error at the start of the process is approximately 7700, and proceeds to decline rapidly. The decline is somewhat linear until the averaged error reaches 1000 at around 1000 generations and continues the decline logarithmically. The likely explanation is that by 1000 generations the individuals with high regularisation error have been eliminated. The rate of the convergence speed continues to slow down as the process approaches the global minimum.

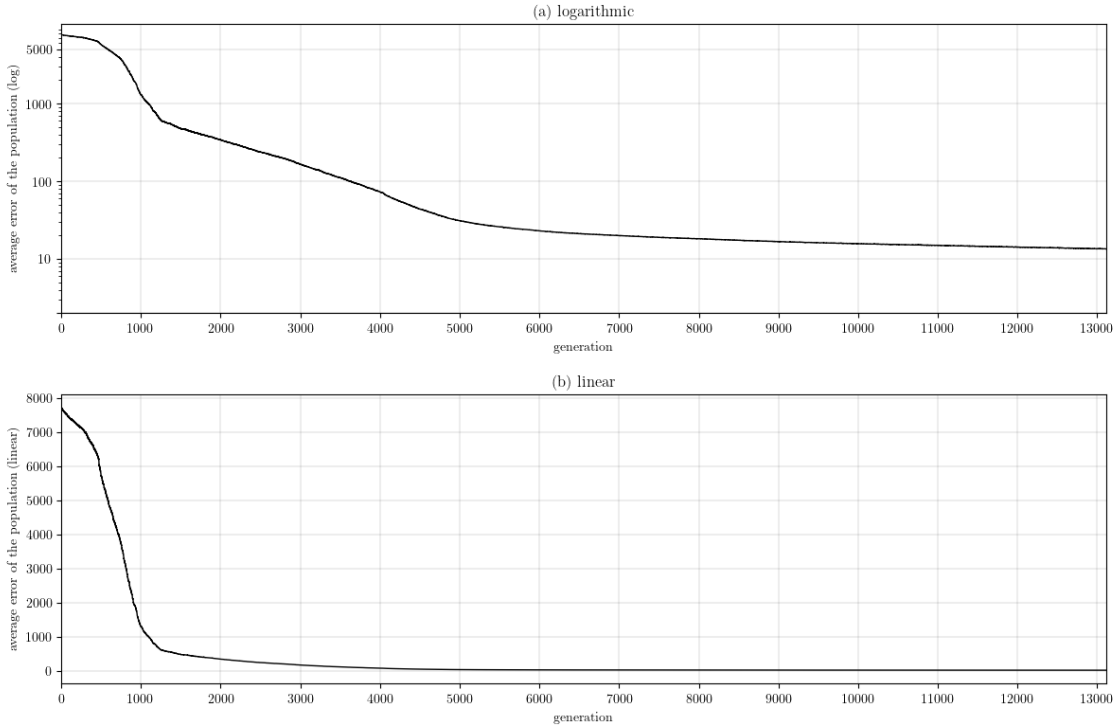

Figure 14
The average error of every individual in the population as generations are computed. Both subfigures plot the same data, with (a) being on logarithmic scale and (b) on linear scale.

5.2 Investigating the best obtained result

Due to the slowing convergence rate towards the end of the optimisation process, the optimisation had to be terminated before the population reached the global minimum. Figure 14 reveals that half of the runtime of the optimisation job was spent in diminishing the average error of the population from around 20 to 13.5. Assuming that the convergence follows exponentially decaying trend, halving the error from this would have likely required doubling the computational time for the optimisation process. This was not possible due to the costs involved.

The final parameters obtained with the optimisation are plotted in figure 15.

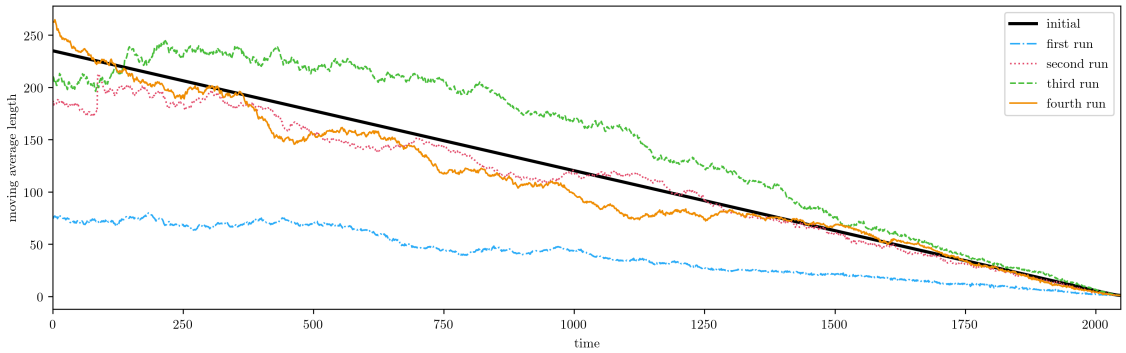

Figure 15
The final best parameters from the optimisation process. The initial guess derived in Appendix D is also visualised for reference.

Observations can be made from the final parameters, even if the obtained vector may not represent the best global result available. In both the best parameters (Fig. 15) and the population in total (Fig. 16) the moving averages align to follow different steepnesses. The first moving average has shortest lengths, and it also has the smoothest run. This would indicate that the first moving average run is subject to most pruning by the optimisation algorithm. This can be intuitively understood when considering the behaviour of the algorithm. If the lengths of the first moving average are set too high, the subsequent runs cannot correct this, as each of the moving averages is a lowpass filter. Thus, as the optimisation process is allowed to run for sufficiently long time, the rest of the moving average runs can be hoped to become smoother and more linear.

The initial guess provides a naive analytical solution to the problem. The *seconds* and *fourth* run seem to tend to this common slope, where as the *first* and the *third* run differ from it. A possible explanation is that the cumulative response of all the moving

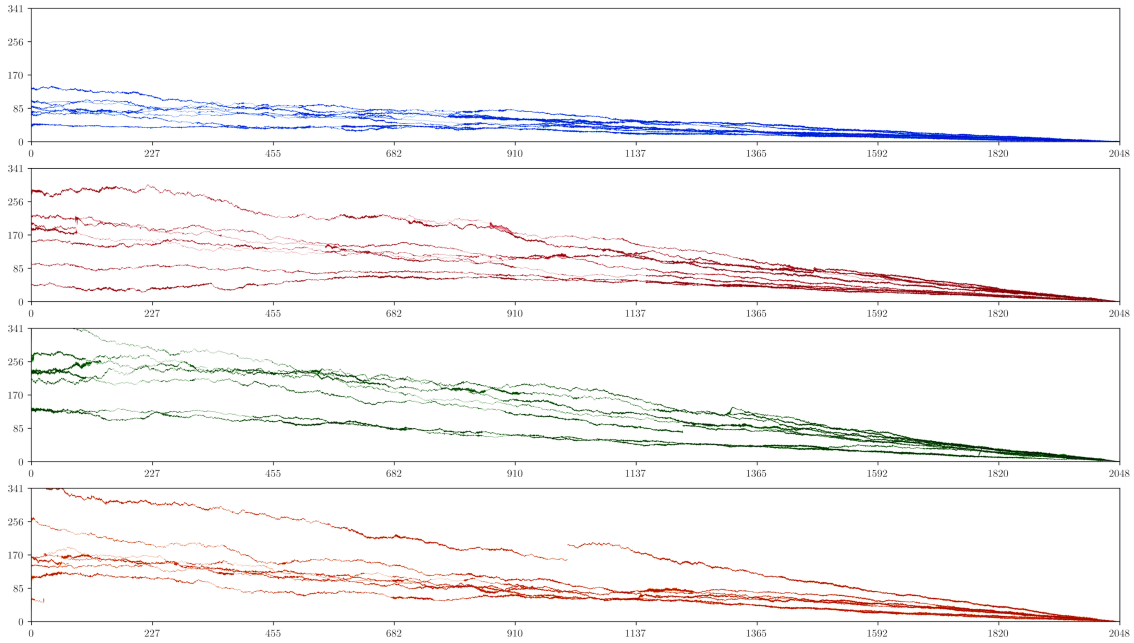

Figure 16

A more accurate plot of the population at the point of termination.

averages is not optimal, if all of the moving averages were to tend to this slope.

Figure 17 illustrates this. In this figure, the *left-hand-side* subfigures plot the magnitude spectrum responses of the moving averages at a specific point of the window. The *right-hand-side* subfigures plot the combined response of the moving averages, and also visualise what the spectrum would be if all the moving averages were set to follow the initial guess, along with what the optimal response of a Gaussian window would be.

It can be observed that between −20dB and 0dB the optimised response follows the response of the Gaussian more closely than the response of the initial guess does. Below this, the moving averages cannot catch the slope of the Gaussian.

If all of the moving averages were set to the initial guess, the response would have periodic notches similar to the response of a comb filter. With the optimised responses, with each moving average following a different slope, the combined response is smoother, and the individual notches from each moving average run tend to align to the peaks in the responses of the other moving averages. Increasing the number of moving averages would likely allow the combined response become smoother.

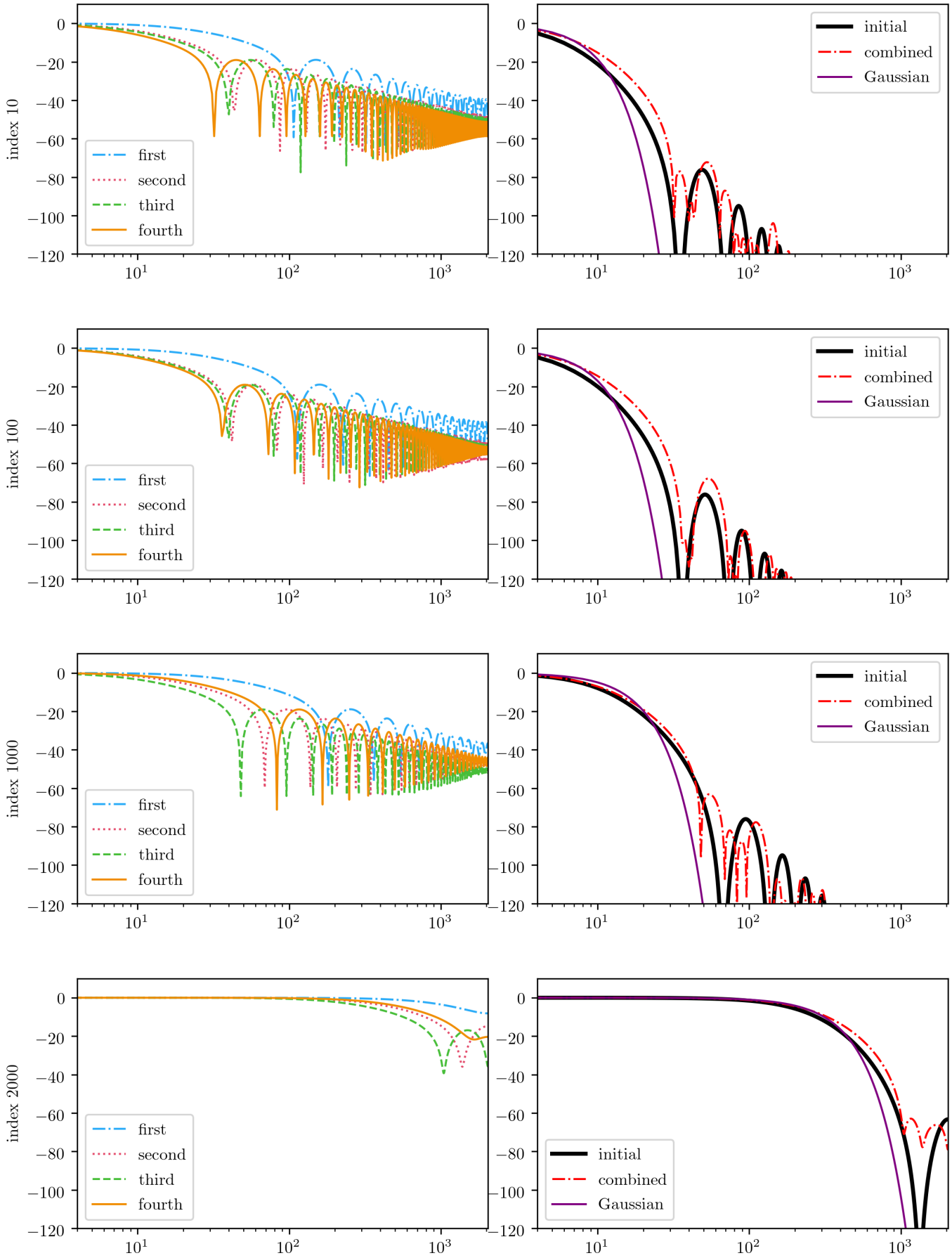

Figure 17

The point-wise moving average lengths for selected indices, and their cumulative response compared to the initial response and the optimal response at that point.

5.3 Illustrating the error of the best obtained result

The problem is ill-posed by definition, as can be easily understood by studying the figure 17. Approximating the response of a Gaussian function will always have some error to it, and the error grows as the number of moving average runs available to the optimiser decreases. With 4 moving average runs, the process can expected to have an error of considerable magnitude. One of the main design goals for this optimisation problem is to spread the error evenly along a chosen distribution.

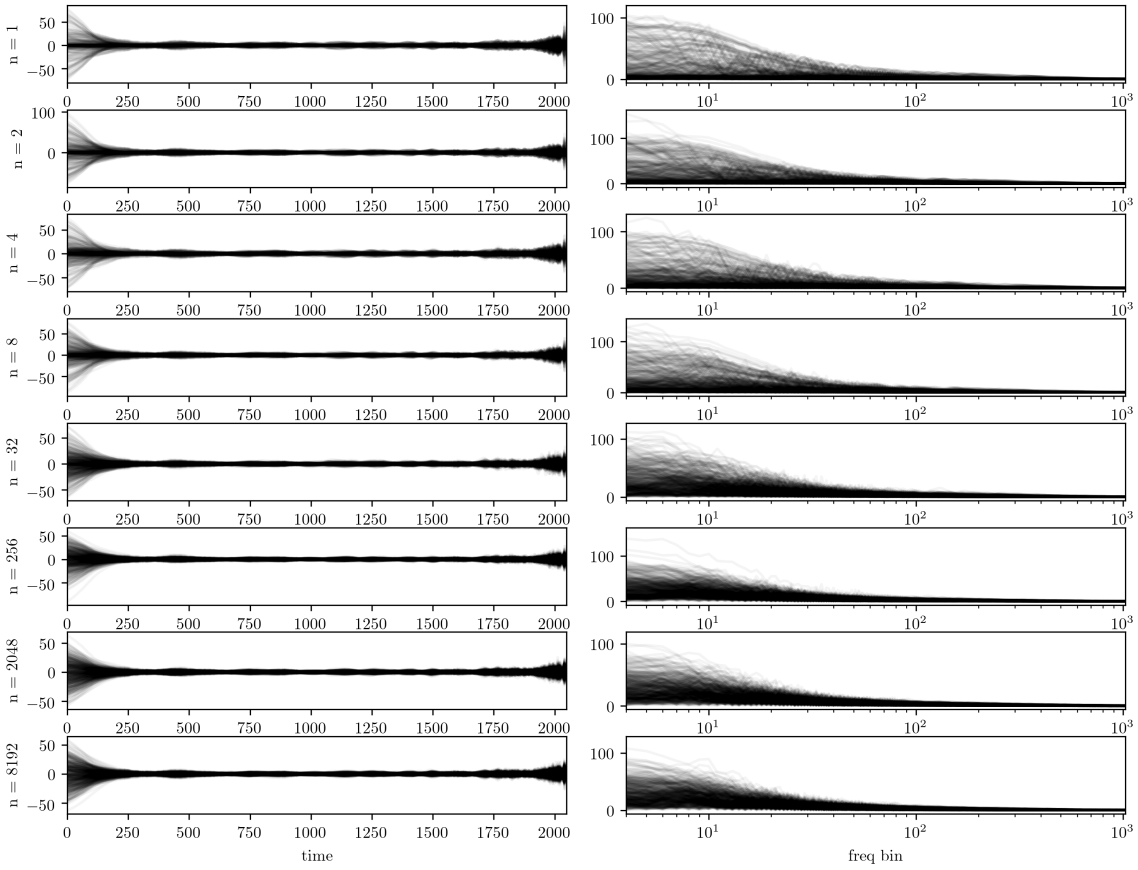

Figure 18
Error of the frequency-variant window function, when tested with a changing number of test oscillations.

Each row plots the error of 512 test signals processed with the obtained window function. The *left-hand-side* column is the time domain error, and the subfigures *right-hand-side* are the corresponding magnitude spectra of the errors. The number of the oscillations per test signal are marked on the left.

Figure 18 illustrates the error of the obtained frequency-variant window function.

Few things can be noted about the Fig. 18. The error is largest at the ends of the

window function. As the number of test oscillations in the evaluation increases, so does the build up of error at the low frequencies. Referring back to Fig. 7 in section 3.4, this can be seen to follow from the distribution of the test signals, where low frequencies have less variation in level, leading to a build-up of energy with larger exposure.

To help understand the error in Fig. 18, the error can also be represented as a function of frequency, as is done in Fig. 19.

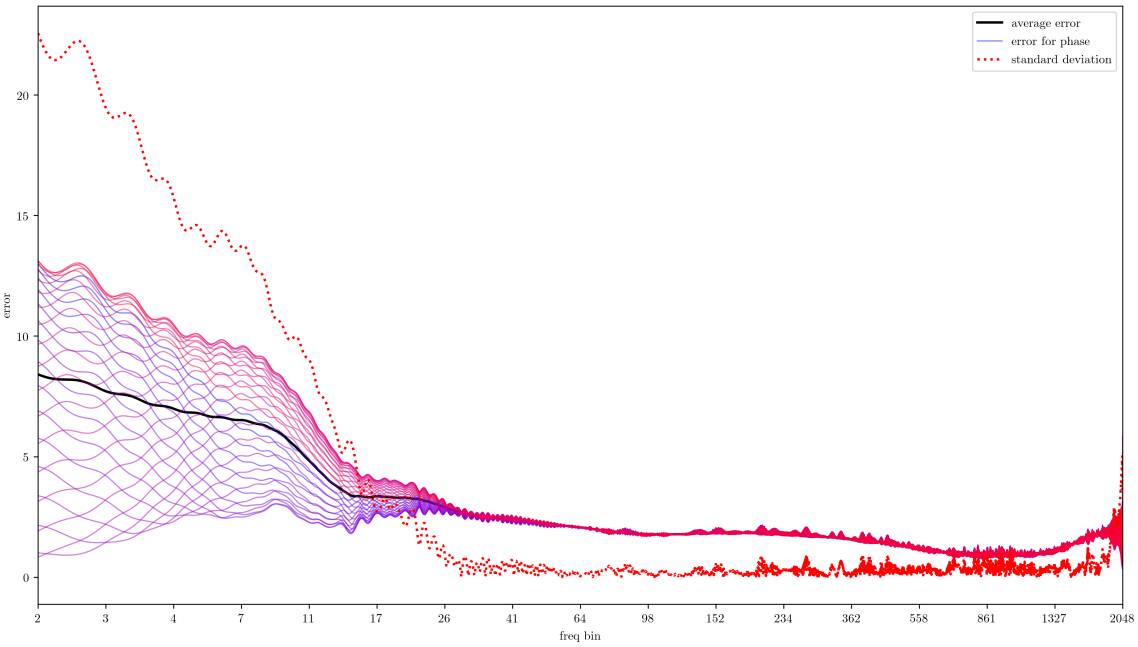

Figure 19
Error of the frequency-variant frequency function, tested with single oscillations of different phases

The figure illustrates how the error varies as the phase of the oscillation is altered. The x-axis represents frequencies of oscillations, and the y-axis represents the RMS-error between an oscillation processed with the window function and its target.

The error is measured with 32 phases, linearly sampling the range $[0, \pi]$. It can be seen that for low frequencies, the amount of error has strong dependency on the phase of the oscillation. For reference, the frequency bin 16 would represent the frequency of 172Hz with sampling rate of 44100.

To assess the scale of the error, the signal-to-noise ratio of the obtained parameters is illustrated in Fig. 20.

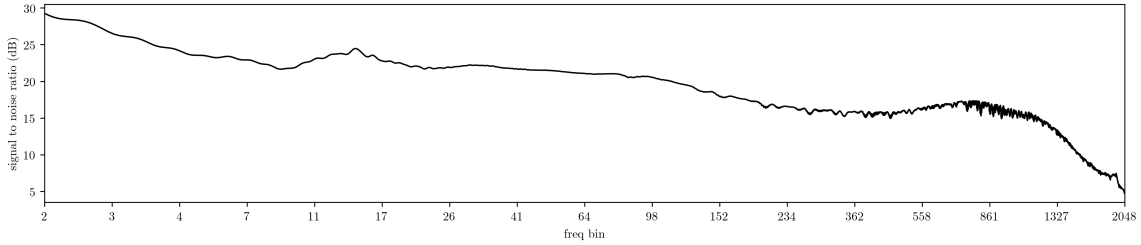

Figure 20
The signal-to-noise ratio of the obtained parameters. The readings are derived as an average per frequency from the data used for the Fig. 19

.

5.4 Understanding the error

The error of the obtained parameters for the frequency-dependent window function is most prominent for the very low and the very high frequencies.

The error for the low frequencies is due to the truncation of the Gaussian window. The Gaussian window for the low frequencies cannot fit into the vector size, as is illustrated in Fig. 21.

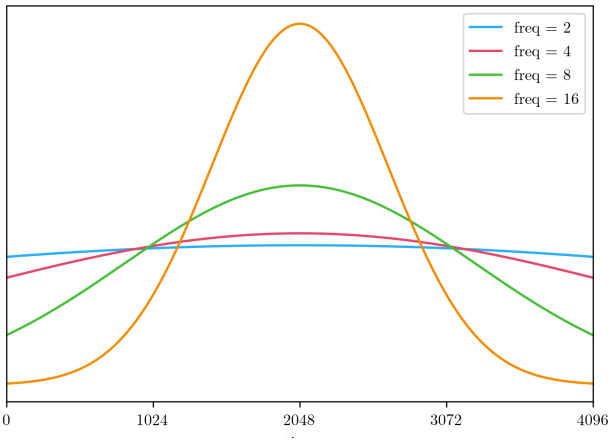

Figure 21
Gaussian windows for different frequencies in the scheme. The sampling rate F_s is set to 4096. For reference, the frequencies correspond to approximately 21Hz, 43Hz, 86Hz and 172Hz for the sampling rate of 44100.

As the frequency drops below $16/N$, the Gaussian window for the frequency no longer tapers to zeros in the support of the frequency-dependent window function. This causes

an error that depends on the phase of the oscillation. The error is at its maximum when the phase is set so that the oscillation has its maximum at the edge of the window function. To treat this, windowing should be incorporated into the optimisation scheme, so that the truncation never occurs.

In this work, additional windowing was not implemented to evaluate how the optimisation method is able to operate even in the presence of a low frequency noise. Essentially, all the frequencies that are below that of $16/N$ represent DC noise to the optimisation process.

Another increase in error is in the centre of the window function, and can be seen in the time domain part of Fig 18 at indices 1750 and above. The centre of the window is arguably the hardest segment for the optimisation method to operate on. Towards the centre, small changes in the moving average lengths cause large deviations in the response, resulting in larger relative error.

5.5 Discussing the findings

The signal-to-noise ratio per frequency depicted in Fig. 20 is useful for assessing the quality of the approximation. With 4 moving average runs, a signal-to-noise ratio of 20dB or over seems achievable. Based on the different representations of the error in figures 18 – 20, and the fact that additional low frequency handling was deliberately left out of the implementation, it can be argued that this approach to optimisation works reasonably well for frequencies between up until $F_s/4$. Above $F_s/4$ the error is not spread evenly, and the optimisation as such is poor, both according to the RMS error and the signal-to-noise ratio.

Apart from the high frequencies, the results obtained with the optimisation are in line with the expectations and the formulation of the problem, and are acceptable considering the computational efficiency of the obtained algorithm.

Using differential evolution for optimising audio signal processing problems that involve a transformation in both time and frequency domains simultaneously appears to be computationally very expensive. The main difficulty arises from having to work with a stochastic cost function. To avoid overfitting, the cost function has to be designed with robustness in mind. In this work, the robustness is achieved by computing each cost with several test signals per iteration, and each test signal is processed with both the candidate and the existing individual to get a comparable *apples to apples* error. This makes running the cost function relatively slow.

For an ill-posed problem with a large number of parameters, the convergence of DE method is slow. The convergence speed decreases exponentially as the error of the population diminishes. As the average error gets smaller, most of the population entries are clustered around a handful of local minima. Because the population size is kept constant throughout the optimisation process, this causes a considerable overhead to improve the population further with large number of parameters.

6 Conclusions

In this work, differential evolution method was applied for optimising an audio signal processing problem that relates to time-frequency representations. The approach was selected to study how the optimisation method could be adapted to fit the problem formulation and the domain of audio signal processing. The main goals of the work were to study how the problem should be formulated for optimisation, and to understand what difficulties rise from optimising audio-related problems. The obtained results are in line with the expectations, and further work is required to make the scheme usable in real use-cases. The work is hoped to be useful as an approachable preliminary research into optimising audio-related problems, and to serve as a basis for further research.

6.1 Further work

Applying a static windowing to the scheme

To reduce the error of the final algorithm, a static window function should be incorporated into the scheme. Currently, the discontinuity at the outer edge of the window function is not able to handle low frequency content gracefully. A static window function would reduce the error at the low frequencies, and could improve both the quality of the approximation and the convergence speed.

Approximating arbitrary window function shapes

The presented scheme could be easily adapted to approximate window functions other than the shape of the Gaussian. This is likely to increase the error of the approximation, but could provide more use-cases for the developed frequency-variant window function. Particularly interesting would be to approximate window functions that feature better overlapping properties, such as the Hann window function.

Need for a time-frequency error representation

To cost function formulation should be revised to better represent the error caused by the processing. The use of purely time or frequency domain error, as they are identical when using the L^2-error norm, is not sufficient. This can be seen from the high error and low signal-to-noise ratio concentrated around the centre of the window function.

A hybrid time-frequency error representation should be used to better estimate the error that rises from modifying a signal simultaneously in both time and frequency domains. Such error estimate should take into account the deviations between the target and the guess in both time and frequency domain, in a fashion where the L^2-norm error could be applied to both deviations simultaneously. This could be realised for example with the use of wavelet transform or some other signal decomposition that represents the frequency in both time and frequency domains simultaneously.

Replacing the differential evolution method

The optimisation software developed as part of this work provides an excellent basis for further experiments. The used optimisation method could be switched with relative ease, as the problem formulation may be kept the same. The original DE method is already quite dated, and was selected to this work due to a large body of documentation available for it, and for the ease of use. With better understanding of the factors involved, DE could now be replaced with for example FADE [22], a more recent adaptive version of DE which should offer better convergence speeds with large parameter sizes. Another interesting choice of optimisation method would be the Cuckoo Search [23], which introduces ranking of the solutions and discarding bad solutions as part of the method, likely making the convergence faster.

Increasing the computational efficiency of the final algorithm

The problem formulation could also be adapted for achieving better computational efficiency for the final algorithm. At the current problem formulation, the symmetric scaled taps (see Appendix C) at the end of the moving average filter kernels present a considerable increase in computation time. These could be replaced with just one scaled tap and the possibility for even-length moving averages, or even with the option to not have the additional tap available at all for certain moving average runs. However, this would require additional control parameters or running a set of optimisations with alternative settings and selecting the one with the best results. These in turn would increase the already high computational resource requirements that come with running the optimisation process.

Moving the processing to GPU

A prevalent trend in computational optimisation in the recent years has been to run the optimisation processes in massively parallel fashion on graphics processing units (GPU) instead of the central processing units (CPU). For this, the process should be parallelised into finer segments, and would essentially require redesigning and rewriting the entire optimisation software. The possibly lowered resource requirements for running the optimisation job could still make the work worth it, as running an optimisation of this size is always associated with hardware costs.

Appendix A Background to windowing and time-frequency plane

In many audio signal processing applications it can be useful to study the momentary spectrum of an audio signal. A typical application for analysing the momentary spectrum is in transet or onset detection, where the information about the momentary spectrum is used to determine whether and when a particular note or hit occurs, and what frequencies the onset occurs in [5].

The Fourier spectrum, obtained with Fourier transform (FT) of the signal, is often used in such applications due to the ease of understanding and utilising the information that can be gained from it. Also, Fourier transform is widely used in a number of fields, and due to the wide use, very fast computational implementations exist for it. The effects, artefacts and shortcomings of FT are also generally very well known and documented.

Fourier transform analyses the signal in terms of its frequency contents and represents the signal in frequency domain. It assumes that the signal consists of a set of weighted oscillations, and can be used to determine the amplitude and phase of these oscillations; in other words, Fourier transform decomposes the signal into an orthonormal basis function set of complex sinusoids. By definition, the oscillations in FT are considered to be of infinite length. To study the spectrum of recorded signals, which are often time-varying in nature, the signal is typically segmented into short segments of time and FT is performed for each segment individually. This allows to gain insight about the momentary spectrum of the signal and how the frequency content of the signal changes in time.

Segmenting the signal is performed by windowing the signal with a window function of desired properties. A window function is often designed to have finite support, meaning that it is non-zero for a defined range, and zero outside of this range. The shape of a window function also has properties to it, which can be used to reduce the artefacts to the spectrum that result from the windowing. Often used window functions, such as Hann or Blackman, taper smoothly towards the edges of the supported range.

Windowing a signal is performed by multiplying the signal with the window function, so that $y(t) = x(t) \cdot w(t)$. Windowing a signal in the time domain corresponds to convolution in the frequency domain [24]:

$$y(t) = x(t) \cdot w(t)$$
$$Y(\omega) = X(\omega) * W(\omega)$$

where $\cdot$ and $*$ denote the multiplication the convolution operations respectively.

This can be understood as the window function smearing the spectral information. Narrower window functions are better localised in time, but tend to spread the frequency domain representation more. Oscillations have dual nature in time and frequency: the more accurately it is known where an oscillation occurs, the less can be known about the exact frequency of the oscillations.

This duality, known as the uncertainty principle, causes the width of the window function to determine the resolution of the spectral analysis. The use of more localised, that is, narrower, window function causes the obtained spectrum to become the less accurate. This is especially true for lower frequencies. The frequency domain representation of the Fourier spectrum divides the spectrum into linearly spaced frequency *bins*. The linear division causes the lower frequencies to have less relative resolution compared to the higher frequencies due to the linear division. On the other hand, making the window wide enough for sufficient low-frequency resolution causes the window width to be very wide relative to the wavelengths of the higher frequencies, resulting in relatively worse time-localisation for the high frequencies. As such, the selection of the width of the window function presents itself as a compromise between the localisation and the frequency resolution.

A number of computationally efficient approaches have been proposed that allow obtaining the momentary spectrum with good relative resolution for all frequencies.

In 1946 Dennis Gabor applied the theories of quantum physics to signal representations and proposed decomposition of a signal to a set of complex oscillations multiplied by Gaussian windows. In his work, Gabor derives that such an window oscillation has the best localisation in both time and frequency planes. [25] This approach has since been incorporated into the larger family of short-time Fourier transform (STFT) based methods, which calculate the momentary spectrum via overlapping Fourier transforms. STFT divides the time-frequency plane into regions of uniform dimensions, and having its theoretical foundations in the Fourier transform, suffers from the tradeoff between the time and frequency localisation.

Another common approach is make without the Fourier spectrum altogether and divide the time-frequency plane in a different manner, as is done in wavelet transform [26] [27].

In Wavelet transform the signal is decomposed into dilated and translated versions of a mother wavelet. Each wavelet has an equal area in the time-frequency plane, but with wavelets corresponding to higher frequencies having better time resolution, and with the lower frequency wavelets having reduced time resolution. This effectively results in good time-frequency resolution and localisation for all frequencies, but can introduce other issues, such as shift-invariance for the analysis, and the lack of phase information, which again can make understanding the resulting data difficult.

More recent state-of-the-art approaches have also been proposed, such as the Constant Q-transform [28] and the Fast S-transform [29]. The Constant Q-transform, akin to WT, acts like a filter bank with each of the filters having a constant quality factor. In other words, this means that as the centre frequency for the frequency bands decreases, the sharpness of the filter increases. Fast real-time viable computational implementations have been proposed, but with the implementations known to the author, the speed comes partly as a tradeoff for latency, making the real-time use as part of audio processing complicated. An extremely promising approach, the Fast S-transform, would be viable for real-time use both in terms of computational efficiency and latency, but the proposed fast implementation is proprietary.

Any of the methods outlined could be used to determine the momentary spectrum of an audio signal, but each introducing its own difficulties. A common trend in the fast implementations for the time-frequency-based methods is that they operate directly on the Fourier spectrum, and achieve the time domain localisation as filtering in the frequency domain. This raises the question whether an opposite time-domain based approach could be taken to achieve similar results. The work presented in this thesis does not aim to replace any of the established methods, but rather study a complementary approach to the topic, and to use that as basis to study the use of optimisation in the context of time-frequency decompositions.

Appendix B Frequency-dependent window function

The properties of the frequency-dependent window function can be defined individually for each frequency component of the input signal.

For each frequency, the width of the window can be calculated from the equation of normal distribution:

$$w(t) = e^{-1/2*(t/\sigma)^2} \quad \text{where } w(n) \text{ is the normal distribution at point n, and, } \sigma \text{ is the standard deviation.} \tag{15}$$

If a single oscillation is to be windowed with a frequency-dependent window, it can be specified to have its -3dB point at certain distance from the centre of the window. This frequency-dependent point can be specified as m multiples of the frequency's wavelength. The window can thus fit $2m$ multiples of the specified frequency between its -3dB points.

From Eq. 2, we have the general form of the Gaussian function:

$$w(t) = \exp\left(-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2\right)$$

Let the value of the Gaussian $w(t)$ equal the -3dB point of the desired window:

$$w(t) = -3\text{dB} = 1/\sqrt{(2)} = \exp\left(-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2\right)$$

Solving the standard deviation σ:

$$\sigma = \sqrt{\frac{t^2}{\ln 2}}$$

and substituting:

$$t = \frac{F_s m}{f}, \quad \text{where } F_s \text{ is the sampling rate}$$

lets us express the σ as a function of frequency:

$$\sigma = \frac{F_s m}{f\sqrt{log_e 2}}$$

Substituting this back to the equation of normal distribution, we get the equation of our desired window for frequency f:

$$w_f(t) = \exp\left(-\frac{1}{2}(\frac{tm}{F_s f\sqrt{(\ln 2)}})^2\right)$$

Let the signal vector $\hat{\mathbf{s}}$ be an oscillation with frequency f, sampling rate F_s, and phase ϕ:

$$\hat{\mathbf{s}}_{f,\phi}(t) = \exp\left(\frac{j2\pi tf}{F_s} + j\phi\right)$$

The target function for frequency f_i would thus be $\hat{\mathbf{s}}(t)_{f_i,\phi_i}$ windowed with the window $w_{f_i}(t)$:

$$T_i(t) = \exp\left(\frac{j2\pi tf_i}{F_s} + j\phi_i\right)\ \exp\left(-\frac{1}{2}(\frac{tm}{F_s f_i\sqrt{(\ln 2)}})^2\right)$$

The input signal vector $\hat{\mathbf{x}}$ can be modelled as a sum of weighted oscillations.

$$\hat{\mathbf{x}} = \sum_i G_i\hat{\mathbf{s}}_{f_i,\phi_i} = \sum_i G_i \exp\left(\frac{j2\pi tf_i}{F_s} + j\phi_i\right)$$

The target function $T(\hat{\mathbf{x}})$ can thus be expressed as a sum of these oscillations, each multiplied by its corresponding window function:

$$T(\hat{\mathbf{x}}) = \sum_i G_i T_i(t) = \sum_i G_i \exp\left(\frac{j2\pi tf_i}{F_s} + j\phi_i\right) \exp\left(-\frac{1}{2}\left(\frac{tm}{F_s f_i\sqrt{\ln 2}}\right)^2\right) \tag{16}$$

for all oscillations i.

Appendix C Arbitrary-length moving average as a matrix operation

Moving average filtering a signal is a linear process. This means that a moving average filter satisfies both the superposition property and scaling property. Any linear operation on a signal vector can be expressed as a matrix multiplication $\mathbf{H} \cdot \hat{\mathbf{x}}$, where $\mathbf{H}$ is a $N \times N$ matrix and $\hat{\mathbf{x}}$ is a signal vector of the length N.

Expressing a moving average filtering as a matrix multiplication is of the following form:

$$H_i = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \dots & w_{0,N-1} \\ w_{1,-1} & w_{1,0} & w_{1,1} & \dots & w_{1,N-2} \\ w_{2,-2} & w_{2-1} & w_{2,0} & \dots & w_{2,N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N-1,1-N} & w_{N-1,2-N} & w_{N-2,3-N} & \dots & w_{N-1,0} \end{bmatrix} \tag{17}$$

$$\text{where:} \qquad w_{i,j} = \begin{cases} \frac{1}{L}, & \text{if} \quad -\frac{L_i-1}{2} \leq j \leq \frac{L_i-1}{2} \\ 0, & \text{otherwise} \end{cases}$$

Each element of the matrix is either $1/L$ or 0 depending on how close it is to the diagonal $w_{i,0}$-axis of the matrix. Values that are less than half the length of the moving average away from the diagonal centre axis are part of the moving average for that index i, and get the value $1/L$. The $1/L$ is used here to avoid introducing extra energy to the filtered signal.

If the length L of the moving average filter is constant, the matrix becomes a so-called Toeplitz matrix – a multiplication of a Toeplitz matrix and a signal vector corresponds to the convolution operation.

To realise the approximation of a Gaussian window which has different widths for different frequencies, a LTV formulation of the moving average filter is needed. The LTV moving average should be able to approximate lengths that are non-odd to let the optimisation scheme approximate high frequencies accurately. Further, even with the addition of the the non-odd lengths the computational cost should be kept low. In this appendix, we derive the used formulations for these.

C.1 Arbitrary-length moving average

With real-life signals, the matrix representation 17 for the LTV moving average always produces a considerable amount of error as the length of moving average becomes less than 3. In discrete time, a moving average of the length 1 can be understood as a point-wise convolution between a signal and an impulse function, resulting in unmodified signal. The next possible length of a symmetrical moving average, $N = 3$, already causes a jump in the frequency response of the window function, resulting in a poor approximation at the high frequencies.

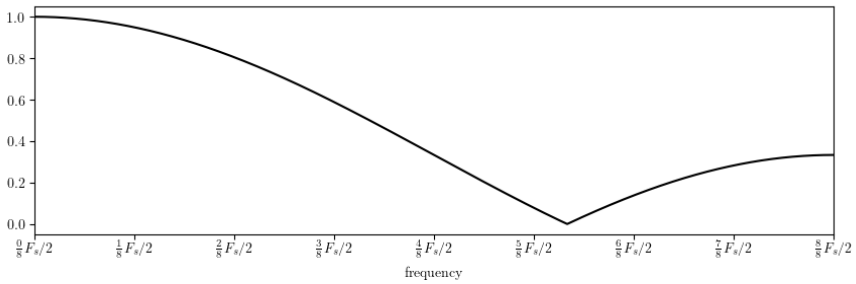

Figure 22

Magnitude spectrum of the moving average filter kernel of length 3.

To let moving average approximate the in-between odd lengths better, two additional filter taps are introduced at the ends of the moving average, with scalable amplitude between $[0, \frac{1}{2L}]$. Positioning the additional taps to both sides of the moving average preserves the linear-phase response of the centred moving average with acceptable computational efficiency.

This adapted moving average can then be written:

$$y[i] = \frac{1}{N+2q}\left(\left(\sum_{j=i-N/2}^{i+N/2} x[j]\right) + qx[i-\frac{N}{2}-1] + qx[i+\frac{N}{2}+1]\right) \tag{18}$$

When the value of scaling coefficient q equals 0, this formulation equals to the moving average of length N, and when $q = 1$, the equation becomes a moving average of length $N + 2$. This effectively allows us to approximate the in-between odd length moving averages.

The spectrum of a rectangular pulse is the sinc-function [30]. When compared to the continuous formulation of the moving average, a rectangular pulse of length N, we can see that the discrete formulation in Eq. 18 this is not exact. Empirically can be found

that the magnitude spectrum of a moving average of any length can be approximated with the additional filter taps. Comparing to the corresponding sinc function, the phase response will be different, and will thus act as a potential source of distortion in our scheme.

A closed form solution for the coefficient q is not readily available, and should thus be found numerically. To avoid having to do this manually, the problem of finding the optimal coefficients q is left to the optimisation algorithm, and instead care should be taken that the optimisation algorithm is able to apply the additional filter taps as part of the optimisation process.

C.2 Adapting the matrix

To let the optimisation algorithm easily apply any given q to the moving average, the value of q should be relative to the length of the moving average L. The proposed way to introduce the q is to derive it directly from the length L as the difference between the closest available odd number and the length L.

Let the length of the moving average be an arbitrary real number, with $L \geq 1, L \in \mathbb{R}$, we can write the odd length λ and the scaling coefficient q with the help of a FloorToOdd$(\cdot)$ -operation as:

$$\textsc{FloorToOdd}(x) = \textsc{Trunc}((x-1)/2) * 2 + 1 \tag{19}$$

$$\lambda = \textsc{FloorToOdd}(L) \tag{20}$$

$$q = (L - \lambda)/2, \tag{21}$$

where Trunc$(\cdot)$ is the truncation operation, rounding the number down to the closest integer.

We can now substitute the matrix 17 with the following:

$$H_i = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \dots & w_{0,N-1} \\ w_{1,-1} & w_{1,0} & w_{1,1} & \dots & w_{1,N-2} \\ w_{2,-2} & w_{2-1} & w_{2,0} & \dots & w_{2,N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N-1,1-N} & w_{N-1,2-N} & w_{N-2,3-N} & \dots & w_{N-1,0} \end{bmatrix} \tag{22}$$

$$\text{where:} \qquad w_{i,j} = \begin{cases} \frac{1}{L} & \text{if} \quad -\frac{\lambda_i - 1}{2} \leq j \leq \frac{\lambda_i - 1}{2} \\ q\frac{1}{L} & \text{if} \quad j = -(\frac{\lambda_i - 1}{2} + 1) \quad \text{or} \quad j = \frac{\lambda_i - 1}{2} + 1 \\ 0 & \text{otherwise} \end{cases}$$

C.3 Treating the q for faster convergence

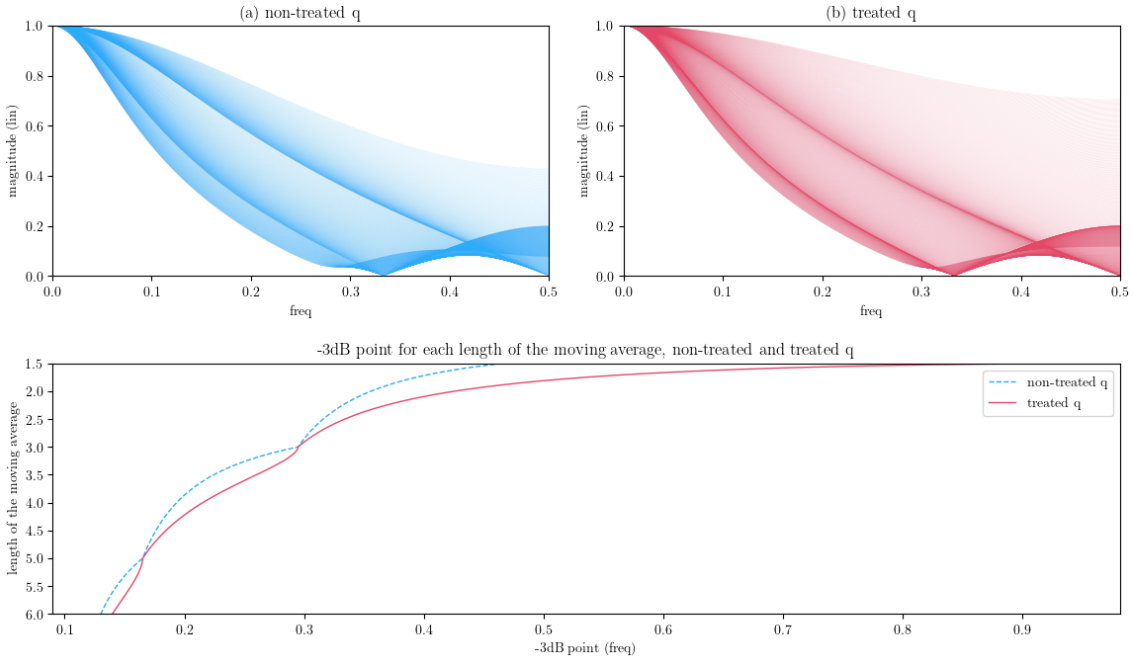

Figure 23
1024 spectra of moving averages with the lengths varying linearly in the range $[1.5, 6]$. Smaller lengths are higher in the plots.

From the plot (a) depicting the non-treated values of q the sharp edge can be seen. As the length of the moving average approaches the odd number from smaller lengths (less steep slope), the density of the frequency spectra increases, meaning that as q approaches the odd number, the resulting spectral line is less and less deviated from the spectral line of the odd-length moving average. As the length of the moving average passes the odd number, the density becomes noticeably larger, as indicated by the lighter tone in the plot.

In the plot (b) depicting the spectral lines with the treated q, the density is more even on both sides of the odd number, making it easier for the optimisation process to pass over the odd length value.

The plot on the bottom depicts length of the moving averages as a function of $-3dB$ point in frequency. The line for the treated values of q is is noticeably smoother than the line for the non-treated values of q.

As visualised in the figure 23, with the arbitrary-length moving average, every odd number of length will create a non-continuous point with a sharp edge to the $-3dB$ point of the moving average filter. These act like local minima and cause the optimisation to

get stuck when taking fine steps. To aid the optimisation process traverse the function and find the optimal non-odd part more accurately, the q value of the optimisation should be treated to make the filters' -3dB point progress more smoothly as a function of length. Empirically was found that substituting the q in equation 22 with:

$$q_t = \left(\frac{L-\lambda}{2}\right)^{1.7} = q^{1.7} \tag{23}$$

provides sufficient smoothness to prevent the optimisation getting sucked into the odd integers. The formulation q_t is an intermediate step only used with the optimisation algorithm. As soon as the desired lengths for the arbitrary-length moving average filters are found, each q_t can be turned back into the q values with the inverse operation.

Appendix D Deriving the initial guesses for the population

A preliminary solution for the individuals in the population can be calculated from the statistical representation of moving average filter. A moving average of length n can be considered to correspond to discrete uniform distribution with n possible values [cite needed]. The variance of discrete uniform distribution is [31]:

$$\sigma_{ma}^2 = \frac{1}{12}(n-1)(n+1)xt$$

Thus M passes of such moving average would have the distribution:

$$\sigma_{M \cdot ma} = \sqrt{\frac{M}{12}(n^2-1)}$$

Since we assume that a normal distribution can be approximated by several passes of moving average filters, we can calculate an initial guess for the moving average length by substituting the the standard deviation σ with the approximate $\sigma_{M \cdot ma}$:

$$\sqrt{\frac{M}{12}(n^2-1)} \approx \sqrt{\frac{t^2}{\ln 2}}$$

Solving the length of the moving average n lets us arrive at the initial guess:

$$n \approx t\sqrt{\frac{12}{M \ln 2} + 1}$$

Where n is the length for all subsequent moving average runs at time index t, when we have M runs in total.

References

[1] S. Koziel and X.-S. Yang, *Computational optimization, methods and algorithms.* Berlin, Heidelberg: Springer-Verlag, 2011, vol. 356.

[2] M. A. Nielsen, "Neural networks and deep learning," http://neuralnetworksanddeeplearning.com/, 2018, accessed 22.3.2019.

[3] K. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, ser. Natural Computing Series. Berlin, Heidelberg: Springer-Verlag, 2006.

[4] S. Chattopadhyay, S. K. Sanyal, and A. Chandra, "Comparison of various mutation schemes of differential evolution algorithm for the design of low pass fir filter," 2011.

[5] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A tutorial on onset detection in music signals," *IEEE Transactions on speech and audio processing*, vol. 13, no. 5, pp. 1035–1047, 2005.

[6] A. A. Nugraha, A. Liutkus, and E. Vincent, "Multichannel audio source separation with deep neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 9, pp. 1652–1664, 2016.

[7] Z. Fu, G. Lu, K. M. Ting, and D. Zhang, "A survey of audio-based music classification and annotation," *IEEE transactions on multimedia*, vol. 13, no. 2, pp. 303–319, 2011.

[8] V. Pulkki and M. Karjalainen, *Communication Acoustics: An Introduction to Speech, Audio and Psychoacoustics.* Wiley, 2015.

[9] S. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Pub., 1997.

[10] J. Klapper and C. Harris, "On the response and approximation of Gaussian filters," *IRE Transactions on Audio*, no. 3, pp. 80–87, 1959.

[11] S. Engelberg, "The central limit theorem and low-pass filters," in *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004.* IEEE, 2004, pp. 65–68.

[12] M. Najim, *Digital filters design for signal and image processing.* ISTE Ltd., 2006.

[13] J. Smith, *Introduction to Digital Filters: With Audio Applications*, ser. Music signal processing series. W3K, 2008.

[14] G. F. Margrave, "Theory of nonstationary linear filtering in the fourier domain with application to time-variant filtering," *Geophysics*, vol. 63, no. 1, pp. 244–259, 1998.

[15] Scholarpedia, "Metaheuristic optimization," http://www.scholarpedia.org/article/Metaheuristic_Optimization#Differential_Evolution, 2019, last accessed 25.3.2019.

[16] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 2. IEEE, 2004, pp. 2023–2029.

[19] N. J. Nilsson, "Introduction to machine learning," https://ai.stanford.edu/~nilsson/MLBOOK.pdf, 1998, last accessed 20.3.2019.

[20] E. W. Weisstein. Parseval's Theorem." From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/ParsevalsTheorem.html. Last accessed 12.3.2019.

[21] M. Jonsson and O. Niemitalo, "Evolutionary real variable optimization in C++, Optimization library," http://yehar.com/blog/?p=643, 2012, last accessed 15.2.2019.

[22] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput.*, vol. 9, no. 6, pp. 448–462, Jun. 2005.

[23] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*. IEEE, 2009, pp. 210–214.

[24] M. Karjalainen and T. Paatero, "Frequency-dependent signal windowing," in *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No. 01TH8575)*. IEEE, 2001, pp. 35–38.

[25] D. Gabor, "Theory of communication. Part 1: The analysis of information," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946.

[26] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE transactions on information theory*, vol. 36, no. 5, pp. 961–1005, 1990.

[27] S. Mallat, *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*, 3rd ed. Orlando, FL, USA: Academic Press, Inc., 2008.

[28] N. Holighaus, M. Dörfler, G. A. Velasco, and T. Grill, "A framework for invertible, real-time constant-Q transforms," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 4, pp. 775–785, 2013.

[29] R. A. Brown, M. L. Lauzon, and R. Frayne, "A general description of linear time-frequency transforms and formulation of a fast, invertible transform that samples the continuous S-transform spectrum nonredundantly," *IEEE Transactions on Signal Processing*, vol. 58, no. 1, pp. 281–290, 2010.

[30] E. W. Weisstein. "Sinc Function." From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/SincFunction.html. Last accessed 15.3.2019.

[31] ——. Discrete Uniform Distribution. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/DiscreteUniformDistribution.html. Last accessed 15.3.2019.