DIFFERENTIABLE ALL-POLE FILTERS FOR TIME-VARYING AUDIO SYSTEMS

Chin-Yun Yu *Christopher Mitcheltree*
Joshua Reiss *György Fazekas*

Centre for Digital Music
Queen Mary University of London
London, UK
`{ chin-yun.yu, c.mitcheltree, joshua.reiss, george.fazekas }@qmul.ac.uk`

Alistair Carson
Stefan Bilbao

Acoustics and Audio Group
University of Edinburgh
Edinburgh, UK
`{ alistair.carson, sbilbao }@ed.ac.uk`

ABSTRACT

Infinite impulse response filters are an essential building block of many time-varying audio systems, such as audio effects and synthesisers. However, their recursive structure impedes end-to-end training of these systems using automatic differentiation. Although non-recursive filter approximations like frequency sampling and frame-based processing have been proposed and widely used in previous works, they cannot accurately reflect the gradient of the original system. We alleviate this difficulty by re-expressing a time-varying all-pole filter to backpropagate the gradients through itself, so the filter implementation is not bound to the technical limitations of automatic differentiation frameworks. This implementation can be employed within any audio system containing filters with poles for efficient gradient evaluation. We demonstrate its training efficiency and expressive capabilities for modelling real-world dynamic audio systems on a phaser, time-varying subtractive synthesiser, and feed-forward compressor. We make our code available and provide the trained audio effect and synth models in a VST plugin[1].

1. INTRODUCTION

Infinite impulse response (IIR) filters are commonly used in many time-varying audio processing units, such as subtractive synthesisers, phaser effects, and dynamic range compression. Their recursive computation, using results from previous time steps, allows modelling a wide range of responses with low computational costs. Since differentiable DSP (DDSP) [1] emerged as an effective solution in attaining controllable audio systems, there have been attempts to incorporate recursive filters into automatic differentiation frameworks such as PyTorch. However, naive implementations result in deep computational graphs due to recursion that cannot be parallelised [2] and slow down the training speed [3, 4, 5].

A common acceleration approach is to evaluate the filters in the frequency domain [3] and approximate time-varying behaviour by filtering on overlapping frames in parallel [6, 7]. Despite their popularity, these approximations have some potential drawbacks. Frame windowing for overlap-add can smear the spectral peaks in the frequency domain, resulting in filters with artificially high resonance. Sampling the filters in the frequency domain sometimes truncates the IR length, and the circular convolution with the truncated IR caused by frequency sampling (FS) can lead to artefacts at the frame boundaries. Most importantly, systems trained in this fashion are not guaranteed the same results when operating sample-by-sample at the audio rate to achieve low latency.

In this paper, we propose a solution to these problems by deriving and implementing an efficient backpropagation algorithm for a time-varying all-pole filter. Our filter can be employed in various audio systems by separating the system's poles and zeros and explicitly handling the poles (where the recursion is) with our proposed method fully end-to-end. Our contributions are three-fold:

1. We significantly increase the forward and backpropagation speed of time-varying recursive filters without introducing any approximation to the filter.
2. The systems trained with our implementation can be converted to real time without any generalisation issues.
3. We show that our implementation efficiently and accurately models time-varying analog audio circuits with a recursive structure.

2. RELATED WORKS

Differentiable training of IIR filters has been explored in [8, 9, 10, 3, 11, 4, 7]. To sidestep the problems inherent in training over a recursion, some authors approximate IIR filters in the frequency domain using the fast Fourier transform (FFT) [9, 10, 3, 11, 4], or the short-time Fourier transform (STFT) [7] for time-varying effects. This is known as the frequency-sampling (FS) method. It approximates the filter as time-invariant and with a finite impulse response (FIR) over the duration of a short frame, thus the accuracy of FS depends heavily on the choice of STFT parameters: the hop-size, the frame length, the FFT length, and the windowing function [12]. In machine learning applications, these choices add extra hyper-parameters to models, which may require prior knowledge of the target system to set appropriately.

Time-varying all-pole filters have been used for decades in linear prediction (LP) voice synthesis [13]. Training them jointly with neural networks was first proposed in LPCNet [14]. The authors achieve training efficiency by using inverse-filtered speech as the target because the inverse filter has no recursion. Other works seek to parallelise LP with frame-based processing, either filtering in the time domain [15, 2] or via frequency sampling [6, 16] on each frame.

Dynamic processing effects like compressors and limiters also employ time-varying recursive filters. The filter is usually first order, and the coefficients are time-varying and dependent on the

[1] https://christhetree.github.io/all_pole_filters/

attack or *release* phases of the gain reduction signal [17]. In the differentiable learning context, frequency-sampling can be used if the compressor's attack and release time are configured to be the same [4, 5], which simplifies the filter to be time-invariant. Colonel et al. [18] proposed dividing the gain reduction signal into attack and release passages and filtering them separately with different filters, and Guo et al. [19] downsample the signal to reduce the number of recursions.

A distinct approach that provides significant acceleration is to derive the closed-form solution of calculating the gradients of the filter parameters and implementing it in highly optimised programming languages. Bhattacharya et al. [8] derived the instantaneous backpropagation algorithm for peak and shelving filters and trained them jointly to match the response of head-related transfer functions. Their solutions reveal that gradient backpropagation through an IIR filter could also be expressed using IIR filters. Forgione et al. and Yu et al. [2, 20] decompose a time-invariant IIR filter into zeros and poles, and they show that backpropagation through an all-pole filter can be expressed using the same all-pole filter. This method is fast because the underlying filters do not have to be implemented in an automatic differentiation framework. Nevertheless, these solutions are made for specific filters or are only applied to time-invariant systems.

3. PROPOSED METHODOLOGY

Consider an M^{th}-order time-varying all-pole system:

$$\begin{aligned} y(n) &= f_{\mathbf{a}(n)}(x(n)) \\ &= x(n) - \sum_{i=1}^{M} a_i(n) y(n-i), \end{aligned} \tag{1}$$

where $\mathbf{a}(n)$ are filter coefficients at time n. In some applications, we can assume that $\mathbf{a}(n)$ varies at a control rate F_c much lower than the audio sampling rate F_s, and can then be up-sampled to the audio rate before the filter is implemented.

The following sections describe the proposed method, in which the exact gradients for each parameter of filter f are derived and expressed in a form that can be computed efficiently. Our method aligns most with the instantaneous backpropagation algorithms proposed in [8, 20, 2], extending their contributions to a time-varying all-pole filter that can be used in various recursive filters. We refer to this method as the time-domain (TD) method.

3.1. Gradients of $\mathbf{x}(n)$

If we unwrap the recursive Eq. (1) so there is no y variable on the right-hand side, we get

$$y(n) = x(n) + \sum_{i=1}^{\infty} b_i(n) x(n-i) \tag{2}$$

$$b_i(n) = \sum_{\mathbf{p} \in Q_i} (-1)^{|\mathbf{p}|-1} \prod_{j=1}^{|\mathbf{p}|-1} a_{p_j}(n - \sum_{k=0}^{j-1} p_k) \tag{3}$$

$$Q_i = \left\{ \mathbf{p} : p_0 = 0, p_{>0} \in \{1, \dots, M\}, \sum_{j=0}^{|\mathbf{p}|-1} p_j = i \right\}. \tag{4}$$

Here, $\mathbf{b}(n)$ is the IIR at time n, and Q_i is every combination of steps that goes from 0 to i with M the maximum size of each step. $\mathbf{p}$ is an array of integers and $|\mathbf{p}|$ is the number of elements in $\mathbf{p}$.

Assuming we have computed f up to step N, evaluated $y(n)|_{\leq N}$ with a differentiable function $\mathcal{L}(y(n))$ and have its instantaneous gradients $\frac{\partial \mathcal{L}}{\partial y(n)}$, we can backpropagate them through f as

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x(n)} &= \frac{\partial \mathcal{L}}{\partial y(n)} + \sum_{i=1}^{N-n} \frac{\partial \mathcal{L}}{\partial y(n+i)} \frac{\partial y(n+i)}{\partial x(n)} \\ &= \frac{\partial \mathcal{L}}{\partial y(n)} + \sum_{i=1}^{N-n} b_i(n+i) \frac{\partial \mathcal{L}}{\partial y(n+i)}. \end{aligned} \tag{5}$$

We use the fact that $\frac{\partial y(n)}{\partial x(n-i)} = b_i(n)$ from (2). Eq. (5) means filtering the gradients $\frac{\partial \mathcal{L}}{\partial y(n)}$ with non-causal filters $b_i(n+i)$.

Nevertheless, Eq. (3) and (5) are not computable as an IIR has an infinite length. We aim to express backpropagation using f, which is feasible to compute. To know how to parameterise coefficients for reusing f, let us evaluate (3) for $b_i(n+i)$ and get

$$\begin{aligned} b_i(n+i) &= \sum_{\mathbf{p} \in Q_i} (-1)^{|\mathbf{p}|-1} \prod_{j=1}^{|\mathbf{p}|-1} a_{p_j}(n + i - \sum_{k=0}^{j-1} p_k) \\ &= \sum_{\mathbf{p} \in Q_i} (-1)^{|\mathbf{p}|-1} \prod_{j=1}^{|\mathbf{p}|-1} a_{p_j}(n + \sum_{k=j}^{|\mathbf{p}|-1} p_k) \\ &= \sum_{\tilde{\mathbf{p}} \in Q_i} (-1)^{|\tilde{\mathbf{p}}|-1} \prod_{j=1}^{|\tilde{\mathbf{p}}|-1} \hat{a}_{\tilde{p}_j}(n + \sum_{k=0}^{j-1} \tilde{p}_k) \end{aligned} \tag{6}$$

where $\tilde{\mathbf{p}} = [0, p_{|\mathbf{p}|-1}, \dots, p_1]$, $\hat{a}_i(n) = a_i(n+i)$. Eq. (6) looks very similar to (3) but using $\hat{\mathbf{a}}(n) = [\hat{a}_1(n), \dots, \hat{a}_M(n)]$ as coefficients and the plus sign inside the product changes to the minus sign, which means filtering the signal in reverse ($N \to 0$). Using this similarity and (2), we can express (5) as

$$\frac{\partial \mathcal{L}}{\partial x(n)} = \text{FLIP} \circ f_{\text{FLIP}(\hat{\mathbf{a}}(n))} \circ \text{FLIP}\left(\frac{\partial \mathcal{L}}{\partial y(n)}\right), \tag{7}$$

where $\text{FLIP}(x(n)) = x(N-n)$, $f_1 \circ f_2(x) = f_1(f_2(x))$. The operator $\circ$ represents sequential functions computed from right to left as read. FLIP allows us to reuse f without introducing its reversed-time version and is cheap to compute. The backpropagation algorithm and how to arrange $\mathbf{a}(n)$ into $\hat{\mathbf{a}}(n)$ is shown in Figure 1. The initial conditions, $\frac{\partial \mathcal{L}}{\partial y(n)}|_{>N}$, are zeros because they are not evaluated in $\mathcal{L}$. The computed results are re-used for gradients to the coefficients discussed in the next section.

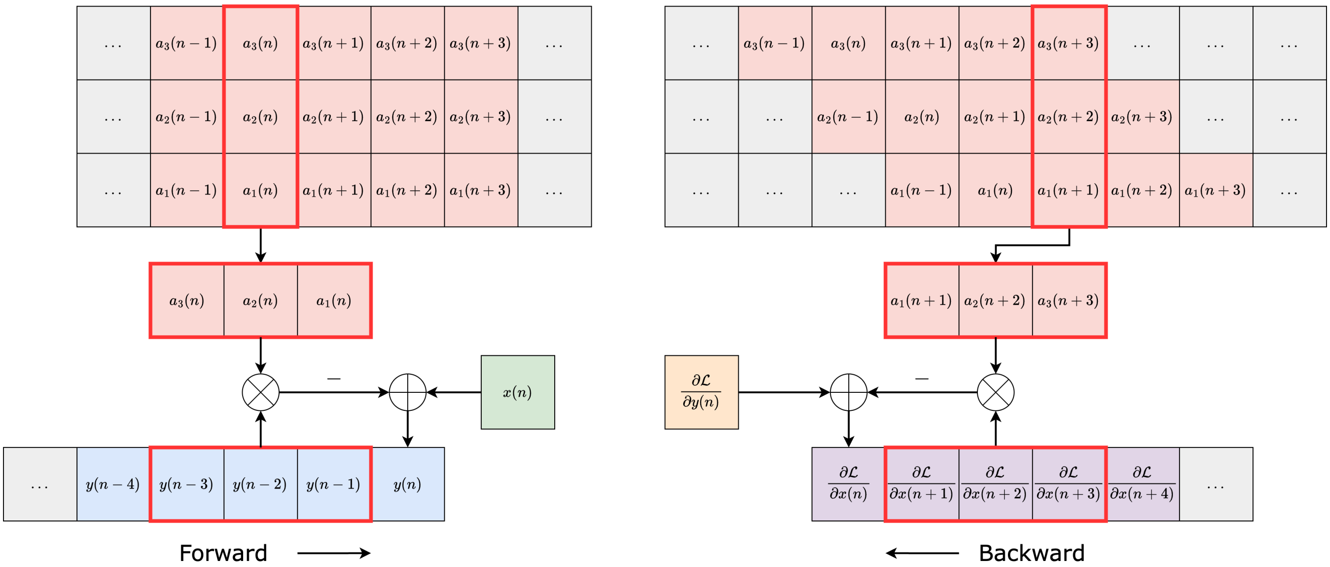
3.2. Gradients of $\mathbf{a}(n)$

Let $u_i(n) = -a_i(n)y(n-i)$ so $y(n) = x(n) + u_1(n) + \cdots + u_M(n)$. Because $\frac{\partial y(n)}{\partial x(n)} = \frac{\partial y(n)}{\partial u_i(n)} = 1$, the variables should have the same derivatives ($\frac{\partial \mathcal{L}}{\partial x(n)} = \frac{\partial \mathcal{L}}{\partial u_i(n)}$) due to the chain rule. Since we can compute $\frac{\partial \mathcal{L}}{\partial x(n)}$ from (7), the gradients of the coefficients are simply

$$\frac{\partial \mathcal{L}}{\partial a_i(n)} = \frac{\partial \mathcal{L}}{\partial u_i(n)} \frac{\partial u_i(n)}{\partial a_i(n)} = -\frac{\partial \mathcal{L}}{\partial x(n)} y(n-i). \tag{8}$$

In summary, we can calculate all the gradients with one pass of f and the multiplications in (8), which are fast to compute. We implement this efficient f for both forward and backward computation using Numba and register it as a custom operator in PyTorch. The implementation is available on GitHub [2].

[2] https://github.com/yoyololicon/torchlpc



...
$a_3(n-1)$
$a_3(n)$
$a_3(n+1)$
$a_3(n+2)$
$a_3(n+3)$
$a_2(n-1)$
$a_2(n)$
$a_2(n+1)$
$a_2(n+2)$
$a_2(n+3)$
$a_1(n-1)$
$a_1(n)$
$a_1(n+1)$
$a_1(n+2)$
$a_1(n+3)$
$-$
$x(n)$
$y(n-4)$
$y(n-3)$
$y(n-2)$
$y(n-1)$
$y(n)$
Forward
$\frac{\partial \mathcal{L}}{\partial y(n)}$
$\frac{\partial \mathcal{L}}{\partial x(n)}$
$\frac{\partial \mathcal{L}}{\partial x(n+1)}$
$\frac{\partial \mathcal{L}}{\partial x(n+2)}$
$\frac{\partial \mathcal{L}}{\partial x(n+3)}$
$\frac{\partial \mathcal{L}}{\partial x(n+4)}$
Backward

Figure 1: *The forward (left) and backpropagation (right) flow chart of a third-order time-varying all-pole filter.*

4. APPLICATIONS

We demonstrate our all-pole filter implementation on three commonly used dynamic audio systems: a phaser, a subtractive synthesiser, and a compressor. All three systems have time-varying recursive structures that are not easy to train in a differentiable way and would typically be modelled using FS approaches.

4.1. Phaser

We test our filter implementation on a virtual analog phaser model, based on [21, 7]. At the core of the model is a differentiable LFO that operates at the control rate F_c. The oscillator is implemented as a damped oscillator with learnable frequency f_0, decay rate σ, and phase ϕ:

$$s(m) = e^{-\sigma^2 m/F_c} \cos(2\pi f_0 m/F_c + \phi) \tag{9}$$

The inclusion of parameter σ alleviates some non-convexity issues when learning the frequency f_0, as shown in [22]. Note that the oscillator is unconditionally stable for all σ. The oscillator is passed through a multi-layer perceptron (MLP) network to obtain the control signal $p(m)$. The MLP, with parameters θ, contains 3x8 hidden layers, with $\tanh$ activation functions on all layers including the final. The control signal is then up-sampled with linear interpolation to obtain $p(n)$ and modulates the coefficients of four cascaded first-order all-pass filters (APF), each with the difference equation:

$$y_k(n) = p(n) \cdot [x_k(n) + y_k(n-1)] - x_k(n-1) \tag{10}$$

where x_k and y_k are the input and state of the k^{th} APF, respectively, $0 \leq k < 4$. Note that the $\tanh$ output activation in the MLP guarantees stability. The APFs are arranged in series, with a through path of gain g_1 and feedback loop g_2 as shown in Figure 2. It is common to include a unit delay in the feedback path for ease of implementation [21, 23], however here we use instantaneous feedback for a more realistic virtual analog model. In Figure 2, BQ represents a biquad filter with coefficients $\mathbf{b}^{(\mathrm{bq})} = [b_0^{(\mathrm{bq})}, b_1^{(\mathrm{bq})}, b_2^{(\mathrm{bq})}]$ and $\mathbf{a}^{(\mathrm{bq})} = [a_1^{(\mathrm{bq})}, a_2^{(\mathrm{bq})}]$. The entire model is a sixth-order time-varying IIR filter, and through some algebra the difference equation can be derived into the form:

$$y(n) = f_{\mathbf{a}(n)} \circ f_{\mathbf{b}(n)}\left(x(n)\right) \tag{11}$$

where $f_{\mathbf{b}(n)}(\cdot)$ is a time-varying FIR filter:

$$f_{\mathbf{b}(n)}\left(x(n)\right) = \sum_{i=0}^{M} b_i(n) x(n-i) \tag{12}$$

and $f_{\mathbf{a}(n)}(\cdot)$ is the time-varying all-pole filter (see Eq. (1)). Here $M = 6$ for $\mathbf{b}(n)$ and $\mathbf{a}(n)$, which are functions of $p(n)$, g_1, g_2, $\mathbf{b}^{(\mathrm{bq})}$ and $\mathbf{a}^{(\mathrm{bq})}$. In previous work [7], the control parameters of a similar time-varying filter were learned through gradient descent using the FS method. This frequency-sampling approach had some limitations, however. Firstly, the optimal frame size for the best training accuracy depended on the rate of the target LFO, which we ideally should not assume as prior knowledge. Secondly, it was not fully investigated whether the trained model could then be implemented in the time-domain at inference to avoid latency.

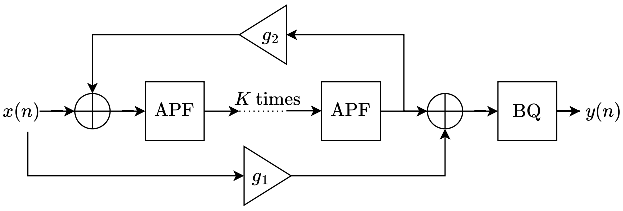


Figure 2: *Discrete-time phaser model considered in this work, where $K = 4$. APF represents a time-varying all-pass filter with difference equation* (10) *and BQ is a biquad filter.*

Here, we instead implement Eq. (11) directly in the time-domain during training using the method proposed in Section 3.

4.2. Time-varying Subtractive synthesiser

We test our filter implementation on a subtractive synthesiser roughly modelled after the *Roland TB-303 Bass Line* synth[3] which defined the acid house electronic music movement of the late 1980s. The TB-303 is an ideal synth for our use case because its defining feature is a resonant low-pass filter where the cutoff frequency is modulated quickly using an envelope to create its signature squelchy, "liquid" sound. Our synth is differentiable and consists of three main components: a monophonic oscillator, a time-varying biquad filter, and a waveshaper for adding distortion to the output.

The oscillator is the same as in the one in `TorchSynth` [24] and uses hyperbolic tangent waveshaping to generate sawtooth or square waves, and can sweep continuously between them. It is defined by the following equations:

$$\psi(n) = 2\pi n f_0/F_s + \phi \pmod{2\pi} \tag{13}$$

$$o(n) = \rho_{\text{osc}} s_{\text{saw}}(\psi(n)) + (1-\rho_{\text{osc}}) s_{\text{sq}}(\psi(n)) \tag{14}$$

$$e(n) = \begin{Bmatrix} \left(\dfrac{N_{\text{on}} - n}{N_{\text{on}}}\right)^{\rho_{\text{env}}} & 0 \le n \le N_{\text{on}} \\ 0 & \text{otherwise} \end{Bmatrix} \tag{15}$$

$$s(n) = g_{\text{osc}} e(n) o(n) \tag{16}$$

where F_s and f_0 are the sampling rate and fundamental frequency in hertz, and ϕ is the phase in radians. ρ_{osc} is a continuous control parameter to sweep between the wave shapes where 0 makes a square wave, and 1 makes a saw wave. The output audio of the oscillator is multiplied by gain g_{osc} and then shaped using a decaying envelope $e(n)$ of length N_{on} note on samples with control parameter ρ_{env}.

A time-varying biquad filter $h_{\text{bq}}(\cdot)$ (same as Equations 11 and 12 for the phaser, but $M = 2$) is then applied to the oscillator output audio. This filter takes as input 5 time-varying filter coefficients $\{a_1(n), a_2(n), b_0(n), b_1(n), b_2(n)\}$ at sample rate which can be passed in directly or generated from filter cutoff and resonance modulation signals $m_{\text{fc}}(n)$ and $m_{\text{q}}(n)$. These modulation signals are then used to calculate the coefficients for a biquad lowpass filter using the corresponding equations in the Audio EQ Cookbook [4].

Finally, the output of the filter is fed through a hyperbolic tangent waveshaper which adds distortion:

$$f_{\text{dist}}(x(n)) = \tanh(g_{\text{dist}} x(n)). \tag{17}$$

The amount of distortion is controlled by parameter g_{dist} which modifies the gain of the input of the waveshaper $x(n)$.

The entire synth is therefore controllable using 8 global parameters $\{f_0, F_s, \phi, N_{\text{on}}, \rho_{\text{osc}}, \rho_{\text{env}}, g_{\text{osc}}, g_{\text{dist}}\}$ and 2 or 5 time-varying parameters $\{m_{\text{fc}}(n), m_{\text{q}}(n)\}$ or $\{a_1(n), a_2(n), b_0(n), b_1(n), b_2(n)\}$. It is defined by composing Equation 16, $h_{\text{bq}}(\cdot)$, and Equation 17 together as follows:

$$y_{\text{synth}}(n) = f_{\text{dist}} \circ h_{\text{bq}}(s(n)) \tag{18}$$

4.3. Feed-forward Compressor

The compressor we consider here is a simple feed-forward compressor from [17], which is defined as:

$$x_{\text{rms}}(n) = \alpha_{\text{rms}} x^2(n) + (1-\alpha_{\text{rms}}) x_{\text{rms}}(n-1) \tag{19}$$

$$g(n) = \min\left(1, \left(\frac{\sqrt{x_{\text{rms}}(n)}}{10^{\frac{CT}{20}}}\right)^{\frac{1-R}{R}}\right) \tag{20}$$

$$\hat{g}(n) = \begin{Bmatrix} \alpha_{\text{at}} g(n) + (1-\alpha_{\text{at}})\hat{g}(n-1) & g(n) < \hat{g}(n-1) \\ \alpha_{\text{rt}} g(n) + (1-\alpha_{\text{rt}})\hat{g}(n-1) & \text{otherwise} \end{Bmatrix} \tag{21}$$

$$y(n) = x(n)\hat{g}(n)\gamma. \tag{22}$$

R, CT, and γ are the ratio, threshold, and make-up gain, respectively. $\alpha_{\text{rms/at/rt}}$ are the average smoothing coefficients. $\alpha_{\text{at/rt}}$ are chosen based on whether the compressor is operated in *attack* phase or *release* phase. This compressor's training efficiency bottleneck is in (21), as the coefficient of a first-order IIR filter is decided *on the fly* inside the recursion, so we cannot use f directly. Moreover, its operating rate is the same as the audio rate, so it is unsuitable for frame-based approximation.

4.3.1. Custom backward function

We backpropagate gradients through (21) using the proposed method in Sec. 3. To handle the if-else statement, we write the gain reduction filter (21) in Numba and record each if-else decision inside the recursion into a binary mask $m(n)$:

$$m(n) = \begin{Bmatrix} 1 & g(n) < \hat{g}(n-1) \\ 0 & \text{otherwise} \end{Bmatrix}. \tag{23}$$

After this, Eq. (21) is equal to the following time-varying IIR:

$$\begin{aligned} \beta(n) &= \alpha_{\text{at}}^{m(n)} \alpha_{\text{rt}}^{1-m(n)} \\ \hat{g}(n) &= \beta(n) g(n) + (1-\beta(n))\hat{g}(n-1). \end{aligned} \tag{24}$$

We can now use $\beta(n)$ and f to backpropagate the gradients through the compressor. The pseudo-code of the differentiable gain reduction filter is summarised in Algorithm 1, and the implementation can be found on GitHub [5].

5. EXPERIMENTS

All three systems are trained to model some target analog audio in an end-to-end fashion using gradient descent. We use the Hanning window for all the frame-based approaches. The ranges of all the interpretable parameters are summarised in Table 2. We provide audio samples for all experiments on the accompanying website.

5.1. Modelling the EHX Small Stone analog phaser

Here we explore using the DSP phaser model outlined in Sec. 4.1 to model an analog phaser pedal: the Electro-Harmonix Small-Stone. This is the same system modelled in [7] using the FS method. The circuit consists of four cascaded analog all-pass filters, a through-path for the input signal, and a feedback path [25] – so topologically the circuit is similar to the discrete-time phaser model considered in this paper. The pedal consists of one knob which controls the LFO rate, and a switch that engages the feedback loop. Six different parameter configurations are considered:

[3] https://www.roland.com/uk/promos/303day/

[4] https://www.w3.org/TR/audio-eq-cookbook/

[5] https://github.com/yoyololicon/torchcomp

Algorithm 1: Differentiable gain reduction filter (21).

```
def forward($g$, $\alpha_{\mathrm{at}}$, $\alpha_{\mathrm{rt}}$):
    $g(-1) \leftarrow 1$
    $\hat{g}(n) \leftarrow$ Eq. (21)
    $m(n) \leftarrow$ Eq. (23)
    return $\hat{g}$, $m$

def backward($\frac{\partial \mathcal{L}}{\partial \hat{g}}$, $\hat{g}$, $g$, $m$, $\alpha_{\mathrm{at}}$, $\alpha_{\mathrm{rt}}$):
    $\beta(n) \leftarrow$ Eq. (24)
    $\frac{\partial \mathcal{L}}{\partial \beta(n)g(n)} \leftarrow$ filtering $\frac{\partial \mathcal{L}}{\partial \hat{g}(n)}$ with Eq. (7) and
      $a_1(n) \equiv \beta(n) - 1$
    $\frac{\partial \mathcal{L}}{\partial g(n)} \leftarrow \frac{\partial \mathcal{L}}{\partial \beta(n)g(n)}\beta(n)$
    $\frac{\partial \mathcal{L}}{\partial \beta(n)} \leftarrow \frac{\partial \mathcal{L}}{\partial \beta(n)g(n)}(g(n) - \hat{g}(n-1))$
    $\frac{\partial \mathcal{L}}{\partial \alpha_{\mathrm{at}}} \leftarrow \sum_n \frac{\partial \mathcal{L}}{\partial \beta(n)}m(n)$
    $\frac{\partial \mathcal{L}}{\partial \alpha_{\mathrm{rt}}} \leftarrow \sum_n \frac{\partial \mathcal{L}}{\partial \beta(n)}(1 - m(n))$
    return $\frac{\partial \mathcal{L}}{\partial g}$, $\frac{\partial \mathcal{L}}{\partial \alpha_{\mathrm{at}}}$, $\frac{\partial \mathcal{L}}{\partial \alpha_{\mathrm{rt}}}$
```

- SS-A: feedback off, rate knob 3 o'clock ($f_0 \approx 2.3\,\mathrm{Hz}$)
- SS-B: feedback off, rate knob 12 o'clock ($f_0 \approx 0.6\,\mathrm{Hz}$)
- SS-C: feedback off, rate knob 9 o'clock ($f_0 \approx 0.09\,\mathrm{Hz}$)
- SS-D: feedback on, rate knob 3 o'clock ($f_0 \approx 1.4\,\mathrm{Hz}$)
- SS-E: feedback on, rate knob 12 o'clock ($f_0 \approx 0.4\,\mathrm{Hz}$)
- SS-F: feedback on, rate knob 9 o'clock ($f_0 \approx 0.06\,\mathrm{Hz}$)

The training data consists of a 30 s chirp-train both dry (input) and processed through the pedal (target). At each training iteration, the input signal is processed through the model in a single batch, and the loss function is computed as the error-to-signal ratio (ESR) between the model output and target. The learnable model parameters are $\{g_1, g_2, f_0, \sigma, \phi, \Theta, \mathbf{b}^{(\mathrm{bq})}, \mathbf{a}^{(\mathrm{bq})}\}$, as defined in Sec. 4.1, giving a total of 182 model parameters. An Adam optimiser with a learning rate 5×10^{-4} is employed to carry out parameter updates every iteration for a maximum of 10k iterations. The test data includes the training data plus the next 10 seconds of the same audio signal, which contains guitar playing. This ensures the learned LFO phase is always aligned to the same point in time.

As reported in [7], the accuracy and convergence of model training depended on the choice of hop-size and window-length. Furthermore, even for a fixed choice of hyper-parameters, the training convergence depended on the initial parameter values, which are pseudo-randomly initialised [7]. We observed that for some random seeds, the oscillator would not converge to the correct frequency f_0 and/or decay rate σ. In a successful run, the learned f_0 is approximately equal to that of the target, and σ converges to zero.

As a baseline, we train the model using the FS method with a single hop-size L for each parameter configuration. The window-length N_{WIN} is set to four times the hop-size, and the FFT length to $2^{\lceil \log_2(N_{\mathrm{WIN}}) \rceil + 1}$. The training process is repeated up to five times with different seeds until a model converges. The proposed time-domain (TD) implementation is then trained with the same hyper-parameters (where relevant) and initial seed as the FS method. Here, the hop-size determines the control rate.

To evaluate the two methods, we train the phaser model with the respective method and then test using both FS and TD at inference time. The test ESR can be seen in Table 1. For all datasets, it can be seen that the TD training results in a lower test loss across both FS and TD evaluation. This demonstrates that the proposed time-domain implementation can improve the accuracy in training time-varying IIR filters. Of course, only one hop-size has been considered here, so a more detailed analysis across a range of hop-sizes would be an interesting area of further work. However, these specific hop-sizes were chosen based on the recommendations in our previous work [7], in which they were heuristically found to give the best results (for the respective datasets) using the FS training method.

In early experiments we observed that the TD implementation could become unstable during training, causing the output signal to explode. This occurred even for a simplified problem of $g_2 = 0$ and with the exclusion of the BQ filter. It was verified that the APF poles were within the unit circle for all n, albeit close in some cases ($p(n) > 0.98$). We, therefore, suspect this instability was due to numerical inaccuracies associated with the transient response of the filters (e.g. as described in [12]) because changing from single-precision to double-precision resolved this problem. Therefore, we recommend operating at double precision when using the proposed filter implementation if instability arises.

Table 1: *Phaser evaluation results. "Method" refers to the training method; whereas the ESR was computed over the test dataset using both FS and TD at inference.*

Dataset	L/F_s	Method	ESR (%)	
			FS	TD
SS-A	10 ms	FS TD	**1.51** 1.63	**1.55** **1.55**
SS-B	40 ms	FS TD	1.62 **1.35**	1.67 **1.32**
SS-C	160 ms	FS TD	1.91 **1.25**	2.01 **1.27**
SS-D	10 ms	FS TD	22.39 **20.56**	23.37 **20.75**
SS-E	40 ms	FS TD	15.69 **14.74**	20.37 **17.78**
SS-F	160 ms	FS TD	8.83 **7.61**	10.17 **7.78**

5.2. Modelling the Roland TB-303 Acid Synth

We model analog TB-303 audio with our time-varying subtractive synth. The dataset is made from Sample Science's royalty free *Abstract 303* sample pack[6] consisting of 100 synth loops at 120 BPM recorded dry from a hardware TB-303 clone. All loops are concatenated together, resampled to 48 kHz, and then pitch and note on durations are extracted using Ableton Live 11's melody-to-midi conversion algorithm which we found to be highly accurate for these monophonic melody loops. Since the TB-303 is a 16 note sequencer, the resulting annotated notes are truncated or zero-padded to 6000 samples (one 16th note at 120 BPM and 48 kHz) with any notes shorter than 4000 samples in duration thrown out. This is then split into 60%, 20%, and 20% train, validation, and test sets, respectively, resulting in a total of 42.5 seconds of audio.

We use the same modulation extraction approach as [26] and DDSP [1] to model the synth. First, frame-by-frame features are

[6] https://www.samplescience.info/2022/05/abstract-303.html

extracted from some target audio and are processed by a neural network which predicts the temporal and global control parameters for our differentiable synth as defined in Section 4.2. Temporal parameters are linearly interpolated from frame-rate to sample-rate when required. The synth then generates some reconstructed audio from the control parameters which can be compared against the target audio using a differentiable loss function, thus enabling the system to be trained end-to-end using gradient descent. A diagram of the entire setup is shown in Figure 3.

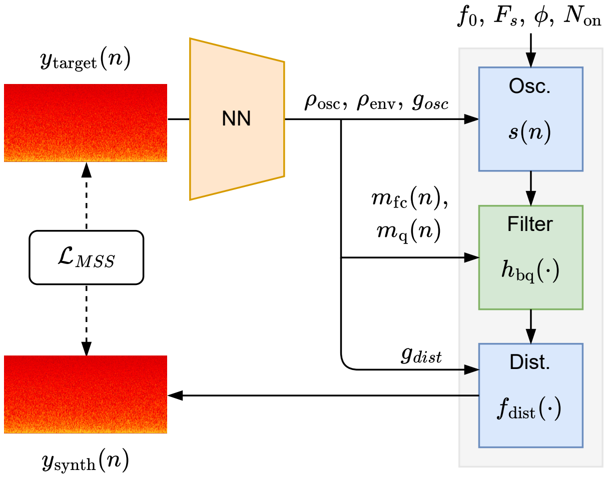


Figure 3: *Diagram of the differentiable synth modelling process. Our time-domain filter component is shown in green.*

Since the filter modulations of the TB-303 are very fast (around 125 ms for 16th notes at 120 BPM), we use a Mel spectrogram with 1024 FFT size, 128 Mel bins, and a short hop length which results in 188 frames for 6000 samples of audio. The neural network is the same architecture as LFO-net [26] except with 4 or 5 additional 2-layer MLP networks applied to the latent embedding averaged across the temporal axis to predict the global parameters of the synth. It contains 728 K parameters.

We conduct experiments with five different synth filter configurations:

1. Time-domain biquad coefficients (Coeff TD)
2. Frequency sampling biquad coefficients (Coeff FS)
3. Time-domain low-pass biquad (LP TD)
4. Frequency sampling low-pass biquad (LP FS)
5. Time-domain recurrent neural network (LSTM)

As discussed in Section 4.2, for configurations 1 and 2 the neural network outputs a 5-dimensional temporal control parameter of changing biquad coefficients. Before filtering, these coefficients are post-processed using the biquad triangle parameterisation of [3] to ensure stability. For configurations 3 and 4, a 2-dimensional modulation signal is returned, representing a changing filter cutoff and its Q factor (which is constant over time). These are then converted to five biquad coefficients. The raw coefficient filter configuration gives the synth as many degrees of freedom as possible whereas the lowpass filter configuration is based on the TB-303's analog design consisting of an envelope modulated lowpass filter with a global resonance control knob. Finally, the recurrent neural network filter (configuration 5) is based on the architecture in [5, 26] and enables us to compare against learning a time-varying IIR filter directly from scratch. It is conditioned with the same 2-dimensional modulation signal as configurations 3 and 4.

We train all models for 200 epochs on batches of 34 notes using the AdamW optimiser and single precision – the numerical issues noted in Sec. 5.1 were not observed here. For the frequency sampling synth filter configurations we train separate models for $N_{\text{WIN}} \in [128, 256, 512, 1024, 2048, 4096]$ while keeping the hop-size L fixed at 32 samples to match the frame-rate temporal outputs of the modulation extraction neural network. As in the phaser experiments, the FFT length is set to $2^{\lceil \log_2(N_{\text{WIN}}) \rceil + 1}$. The LSTM configuration is trained with 64 hidden units. Note pitch and durations are provided to the synth for reconstruction and the phase for the oscillator is random to improve robustness and reflect how synths behave in reality. As a result, the target and reconstructed audio may be misaligned which is why we use multi-resolution STFT loss (MSS) [27] for training which is phase agnostic.

We evaluate the performance of the different filter configurations by comparing their multi-resolution STFT loss values on the 20% test split. The frequency sampling configurations that operate at frame-rate are also evaluated at sample-rate by linearly interpolating their filter coefficients during inference. We also calculate the Fréchet Audio Distance (FAD) [28] for each model which has been shown to correlate with human perception. Since the individual audio files are very short, we first concatenate them into one audio file before calculating the FAD. To avoid harsh discontinuities in the concatenated file, we apply a 32-sample fade (one hop-size L) to both ends of individual clips. The evaluation results are summarised in Table 3. Finally, in Table 4 we show the speed benchmarks of the different synth configurations.

Looking at the evaluation and benchmarking results, we observe that both configurations of our time-domain filter perform well and generally match or outperform the corresponding frequency sampling implementations. Our method also provides roughly a 2x to 30x speedup over the FS and LSTM configurations. The low-pass FS methods perform significantly worse when applied in the time domain which we attribute to overfitting of the neural network to the window size of the filter and can be clearly heard in the resulting audio. This is less the case for the learned coefficient filters, especially according to the FAD metric. We hypothesise this could be due to the linear interpolation at inference of the learned filter coefficients which prevents harsh discontinuities from occurring at the frame boundaries, resulting in a smoother time-varying transfer function. We also found during training that the Coeff FS synths would sometimes not converge, even when using gradient clipping, whereas our time domain implementations never experienced stability issues.

5.3. Modelling the LA-2A Leveling Amplifier

For the compressor experiment, the targets we model are 1) the feed-forward compressor (FF) in Sec. 4.3 and 2) a Universal Audio LA-2A analog compressor (LA). We optimise the proposed differentiable FF compressor (∇FF) to match the target sounds, examining its capability to replicate and infer the parameters of dynamic range controllers. The following are the configurations we test.

- FF-A: $R = 3$, 1 ms attack and 100 ms release
- FF-B: $R = 5$, 30 ms attack and 30 ms release

Table 2: *Parameter limits in our differentiable DSP models.*

	Phaser					Time-varying subtractive synth						Feed-forward compressor					
	f_0	σ	ϕ	g_1	g_2	ρ_{osc}	ρ_{env}	g_{osc}	g_{dist}	$m_{\text{fc}}(n)$	$m_{\text{q}}(n)$	R	CT	α_{at}	α_{rt}	α_{rms}	γ
min.	$-F_c/2$	$-\infty$	$-\pi$	$-\infty$	0	0.0	0.1	0.01	0.01	0.1 kHz	0.7071	1.0	$-\infty$	0.0	0.0	0.0	0.0
max.	$F_c/2$	∞	π	∞	1	1.0	6.0	1.00	4.00	8.0 kHz	8.0	∞	∞	1.0	1.0	1.0	∞

Table 3: *Subtractive synth evaluation results (FS = frequency-sampling method, TD = time-domain method) with 95% confidence intervals for FAD scores.*

			MSS		FAD VGGish	
BQ	Method	N_{WIN}	FS	TD	FS	TD
Coeff.	FS	4096	1.80	1.93	3.29 ± 0.10	2.75 ± 0.09
		2048	1.71	1.74	2.59 ± 0.05	2.54 ± 0.09
		1024	1.66	1.74	3.06 ± 0.12	2.95 ± 0.10
		512	1.55	1.54	2.61 ± 0.08	2.56 ± 0.08
		256	1.54	1.53	**2.45 ± 0.12**	**2.26 ± 0.10**
		128	**1.54**	**1.52**	2.82 ± 0.12	2.34 ± 0.07
	TD	-	-	1.54	-	2.41 ± 0.09
LP	FS	4096	2.03	2.05	3.27 ± 0.10	**2.42 ± 0.09**
		2048	1.98	2.15	2.88 ± 0.05	5.16 ± 0.15
		1024	1.94	2.39	2.70 ± 0.06	5.38 ± 0.15
		512	1.94	2.78	2.47 ± 0.07	3.64 ± 0.10
		256	1.90	2.88	**2.46 ± 0.12**	3.00 ± 0.08
		128	**1.89**	2.85	2.58 ± 0.10	4.79 ± 0.15
	TD	-	-	**1.62**	-	3.49 ± 0.19
-	LSTM 64	-	-	1.78	-	3.67 ± 0.08

Table 4: *Synth runtime benchmarks on an M1 Pro MacBook for one optimisation step (forward + backward) on a single thread. The batch size is 34.*

Synth	TD	FS N_{WIN}					
		128	256	512	1024	2048	4096
Coeff.	49 ms	78 ms	118 ms	221 ms	394 ms	829 ms	1699 ms
LP	48 ms	72 ms	117 ms	212 ms	394 ms	824 ms	1705 ms
LSTM 64	1327 ms	-	-	-	-	-	-

- FF-C: $R = 8$, 0.1 ms attack and 200 ms release
- LA-D: compressor mode, 25 peak reduction
- LA-E: compressor mode, 50 peak reduction
- LA-F: compressor mode, 75 peak reduction

The α_{rms}, CT and γ for FF$_*$ are set to 0.03, -20 and 0 dB, respectively. We train and evaluate our compressors on the Signal-Train dataset [29], which consists of paired data recorded in 44.1 kHz from the LA-2A compressor with different peak reduction values. Following [5], we select the file with the sub-string `3c` in the file name. Each LA-2A setting has 20 min of paired input and target audio containing real-world musical sounds and synthetic test signals. We use the first 5 min for training and the rest for evaluation. The same input data are used for FF-A/B/C, and the targets are generated by applying the FF compressor with the target parameters. We pick the simplified compressor [4] as our baseline, which uses the same FF compressor but has $\alpha_{\text{at}} = \alpha_{\text{rt}}$. We denote this baseline as FS, as (19) and (21) are computed with frequency sampling.

The parameters we optimise are $\{\hat{R}, CT, \hat{\alpha}_{\text{at/bt/rms}}, \gamma\}$. We set $\alpha_* = \text{sigmoid}(\hat{\alpha}_*), R = \exp(\hat{R}) + 1$. The initial values are 50 ms attack/release, $R = 2$, $CT = -10$ dB, $\alpha_{\text{rms}} = 0.3$, and $\gamma = 0$ dB. The conversion from time t in seconds to coefficients α_* is $1 - \exp(-\frac{2.2}{44100t})$. We train each compressor on the whole 5 min sequence without mini-batches for at least 1000 epochs using stochastic gradient descent with a learning rate of 100 and 0.9 momentum, minimising the mean absolute error (MAE). For evaluation, we select the parameters with the lowest training loss. A pre-filter $\frac{1-z^{-1}}{1-0.995z^{-1}}$ is applied before calculating loss and evaluation metrics to remove undesired low-frequency characteristic [5]. We use double precision when computing gigantic FFTs in FS to avoid numerical overflow.

Table 5 shows that ∇FF mostly has a lower ESR than FS besides condition FF-B. The parameters learned for condition FF-A/B/C using ∇FF are close to the ground truth. FS can discover the parameters when attack and release are identical (FF-B), but fail otherwise. For LA-2A, ∇FF reasonably captures the analog characteristics (fast attack and slow release, shown in Table 6) of condition E/F. We found FS tends to learn unrealistic settings, such as a very large attack/release time and ratio, as seen in Table 6. This is likely due to its simplified design.

Table 7 shows our method is two to three times faster than frequency sampling. Training takes roughly 43 min for FS and 17 min for ∇FF on an M1 Pro MacBook.

Table 5: *Summary of compressor ESR (%) evaluation.*

Method	FF-A	FF-B	FF-C	LA-D	LA-E	LA-F
FS	2.362	**0.00780**	4.649	11.29	9.485	7.783
∇FF	**0.015**	0.00785	**0.017**	**10.58**	**9.356**	**7.639**

Table 6: *The learned parameters for matching a LA-2A.*

Method	Data	R	CT (dB)	Attack	Release	α_{rms}	γ (dB)
FS	D	9.1	-11.66	489.43 ms		0.008	0.69
	E	231.1	-19.08	44.62 ms		0.606	0.34
	F	2.9	-26.00	0.06 ms		0.002	-0.81
∇FF	D	39.0	-26.58	99.41 ms	0.06 ms	0.703	0.74
	E	13.1	-12.41	5.68 ms	420.56 ms	0.978	0.54
	F	5.4	-20.14	2.24 ms	229.15 ms	0.973	-0.13

Table 7: *Compressor runtime benchmarks for different lengths of* 44.1 kHz *audio. Ran on an M1 Pro MacBook for one optimisation step (forward + backward, one thread, single precision).*

Method	Sequence duration		
	30 s	60 s	120 s
FS	163.4 ms	320.8 ms	663.8 ms
∇FF	64.9 ms	117.9 ms	239.4 ms

6. CONCLUSIONS AND FUTURE WORK

In this work, we propose an efficient backpropagation algorithm and implementation for an all-pole filter that can be used to model time-varying analog audio systems end-to-end using gradient descent. We demonstrate its advantages over previous frequency sampling approximations by using it to model a phaser, a time-varying subtractive synthesiser, and a feed-forward compressor. Our method outperforms frequency sampling in accuracy and training efficiency, especially when using the systems at the sample-rate level. We make our code available and provide the trained audio effect and synth models in a VST plugin.

Our future work involves extending the backpropagation algorithm to work with differentiable initial conditions and applying it to relevant tasks. We also plan on benchmarking the forward-mode differentiation of our filter and investigating its numerical stability.

7. ACKNOWLEDGMENTS

Funded by UKRI and EPSRC as part of the "UKRI CDT in Artificial Intelligence and Music", under grant EP/S022694/1, and by the Scottish Graduate School of Arts & Humanities (SGSAH).

8. REFERENCES

[1] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, "DDSP: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[2] Chin-Yun Yu and György Fazekas, "Singing voice synthesis using differentiable LPC and glottal-flow-inspired wavetables," in *Proc. International Society for Music Information Retrieval*, 2023, pp. 667–675.

[3] Shahan Nercessian, Andy Sarroff, and Kurt James Werner, "Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads," in *ICASSP*. IEEE, 2021, pp. 890–894.

[4] Christian J Steinmetz, Nicholas J Bryan, and Joshua D Reiss, "Style transfer of audio effects with differentiable signal processing," *Journal of the Audio Engineering Society*, vol. 70, no. 9, pp. 708–721, 2022.

[5] Alec Wright and Vesa Valimaki, "Grey-box modelling of dynamic range compression," in *DAFx*, 2022, pp. 304–311.

[6] Lauri Juvela, Bajibabu Bollepalli, Junichi Yamagishi, and Paavo Alku, "GELP: GAN-excited liner prediction for speech synthesis from mel-spectrogram," in *Proc. INTERSPEECH*, 2019, pp. 694–698.

[7] Alistair Carson, Simon King, Cassia Valentini Botinhao, and Stefan Bilbao, "Differentiable grey-box modelling of phaser effects using frame-based spectral processing," in *DAFx*, 2023.

[8] Purbaditya Bhattacharya, Patrick Nowak, and Udo Zölzer, "Optimization of cascaded parametric peak and shelving filters with backpropagation algorithm," in *DAFx*, 2020, pp. 101–108.

[9] Shahan Nercessian, "Neural parametric equalizer matching using differentiable biquads," in *DAFx*, 2020, pp. 265–272.

[10] Joseph T Colonel, Christian J Steinmetz, Marcus Michelen, and Joshua D Reiss, "Direct design of biquad filter cascades with deep learning by sampling random polynomials," in *ICASSP*. IEEE, 2022, pp. 3104–3108.

[11] Taejun Kim, Yi-Hsuan Yang, Academia Sincia, and Juhan Nam, "Joint estimation of fader and equalizer gains of DJ mixers using convex optimization," in *DAFx*, 2022, pp. 312–319.

[12] J. O. Smith III, *Spectral Audio Signal Processing*, `http://ccrma.stanford.edu/~jos/sasp/`, accessed 27/3/24, online book, 2011 edition.

[13] John D. Markel and Augustine H. Gray, *Linear Prediction of Speech*, vol. 12 of *Communication and Cybernetics*, Springer, Berlin, Heidelberg, 1976.

[14] Jean-Marc Valin and Jan Skoglund, "LPCNet: Improving neural speech synthesis through linear prediction," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019, pp. 5891–5895.

[15] Achuth Rao MV and Prasanta Kumar Ghosh, "SFNet: A computationally efficient source filter model based neural speech synthesis," *IEEE Signal Processing Letters*, vol. 27, pp. 1170–1174, 2020.

[16] Suhyeon Oh, Hyungseob Lim, Kyungguen Byun, Min-Jae Hwang, Eunwoo Song, and Hong-Goo Kang, "ExcitGlow: Improving a WaveGlow-based neural vocoder with linear prediction analysis," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2020, pp. 831–836.

[17] Udo Zolzer, *DAFX: Digital Audio Effects*, chapter Nonlinear Processing, pp. 110–112, John Wiley & Sons, 2011.

[18] Joseph Colonel, Joshua D Reiss, et al., "Approximating ballistics in a differentiable dynamic range compressor," in *Audio Engineering Society Convention 153*. Audio Engineering Society, 2022.

[19] Zixun Guo, Chen Chen, and Eng Siong Chng, "DENT-DDSP: Data-efficient noisy speech generator using differentiable digital signal processors for explicit distortion modelling and noise-robust speech recognition," in *Proc. INTERSPEECH*, 2022, pp. 3799–3803.

[20] Marco Forgione and Dario Piga, "dynoNet: A neural network architecture for learning dynamical systems," *International Journal of Adaptive Control and Signal Processing*, vol. 35, no. 4, pp. 612–626, 2021.

[21] J. O. Smith III, *Physical Audio Signal Processing*, `http://ccrma.stanford.edu/~jos/pasp/`, accessed 28/2/23, online book, 2010 edition.

[22] B. Hayes, C. Saitis, and G. Fazekas, "Sinusoidal frequency estimation by gradient descent," in *ICASSP*. IEEE, 2023, pp. 1–5.

[23] Roope Kiiski, Fabián Esqueda, and Vesa Välimäki, "Time-variant gray-box modeling of a phaser pedal," in *DAFx*, 2016, pp. 31–38.

[24] Joseph Turian, Jordie Shier, George Tzanetakis, Kirk McNally, and Max Henry, "One billion audio sounds from GPU-enabled modular synthesis," in *DAFx*, 2021, pp. 222–229.

[25] J. Sleep, "Small Stone Information [Online]," `http://generalguitargadgets.com/effects-projects/phase-shifters/small-stone-information/`, accessed 26/3/23.

[26] Christopher Mitcheltree, Christian J Steinmetz, Marco Comunità, and Joshua D Reiss, "Modulation extraction for LFO-driven audio effects," in *DAFx*, 2023.

[27] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim, "Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram," in *ICASSP*. IEEE, 2020, pp. 6199–6203.

[28] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi, "Fréchet audio distance: A reference-free metric for evaluating music enhancement algorithms," in *Proc. INTERSPEECH*, 2019, pp. 2350–2354.

[29] Scott Hawley, Benjamin Colburn, and Stylianos Ioannis Mimilakis, "Profiling audio compressors with deep neural networks," in *Audio Engineering Society Convention 147*. Audio Engineering Society, 2019.