Differentiable Filters: Applications, Advancements, and Challenges

8.1.2025, IIS Academia Sinica

Chin-Yun Yu, Queen Mary University of London

Something about me

- Yu, Chin-Yun（尤靖允）
- B.S. in CS @NYCU (2018)
- RA & Intern @MCTLab (2017~2018)
- Software engineer (day) & Independent researcher (night) (2019~2022)
- PhD candidate in AI & Music @QMUL (2022~)

- Research interests
  - Music Information Retrieval
  - Voice synthesis
  - Audio Effects Modelling
  - Binaural Audio
- Publications
  - ICASSP, ISMIR, DAFx, Interspeech, WASPAA, AES
- https://iamycy.github.io/

What's on the menu?

I **will** cover:

- **Automatic differentiation (AD)** in **digital filters**
- LTI & LPV systems
- GPU acceleration
- **PyTorch**
- **PhilTorch**
  - github.com/yoyolicoris/philtorch

I **won't** cover:

- Non-linear systems
  - only a few slides
- Continuous systems
  - H(z) 😍
  - ~~H(s)~~ 😨
- Gradients of analog filters
  - No, but that sounds interesting!
- Applications outside audio
  - should be generalisable

Outline

- Part I: Digital filters and their differentiable applications
- Part II: Achieving fast automatic differentiation of filters
- Part III: Unsolved challenges and promising directions

Part I:
Digital filters and their differentiable applications

What are digital filters?

- In the Z domain

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_M z^{-M}}$$

- In the time domain

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_M x(n-M) - a_1 y(n-1) - \cdots - a_M y(n-M) \quad \overset{\mathcal{Z}}{\longleftrightarrow} \quad Y(z) = X(z)H(z)$$

Advantages over neural nets

- ★ Readable and controllable
- ★ Low degree of freedom → Embed strong prior knowledge
  - Faster convergence
  - Cheap computation
  - Compact representation
- ★ Widely used in audio applications
  - easier to deploy

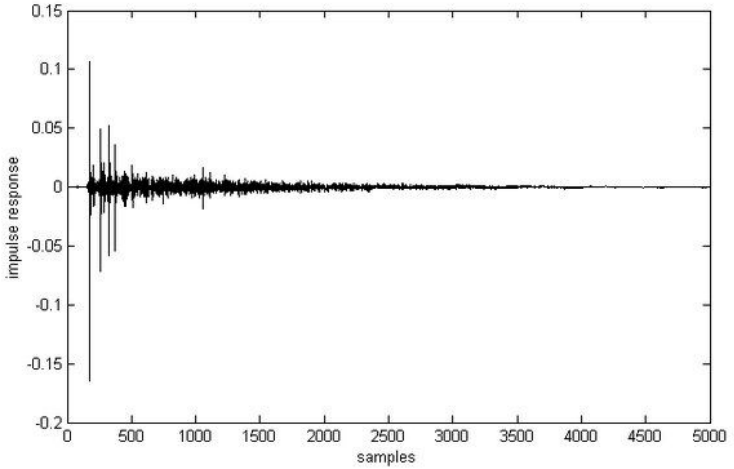
Note: *Still empirical compare to white-box method (e.g., physical modelling)*

Case 1: FIR filters

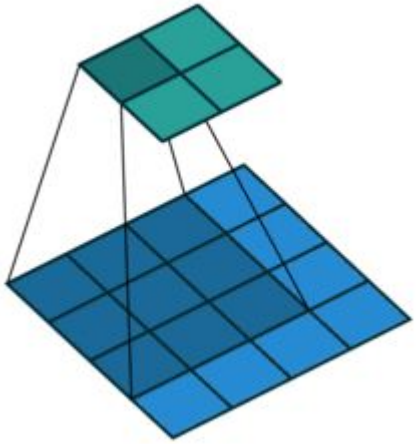
- Feedforward coefficients only
- Has finite-length impulse response (FIR)

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_M x(n-M) = b * x(n)$$

★ Room impulse response (RIR) measurements



In the deep learning era…



Pros

- Easy to parallelise → Fast on the GPU
- Convolutional neural network (CNN)

Cons

- Limited context window → **limited expressiveness**

End-to-end optimisation

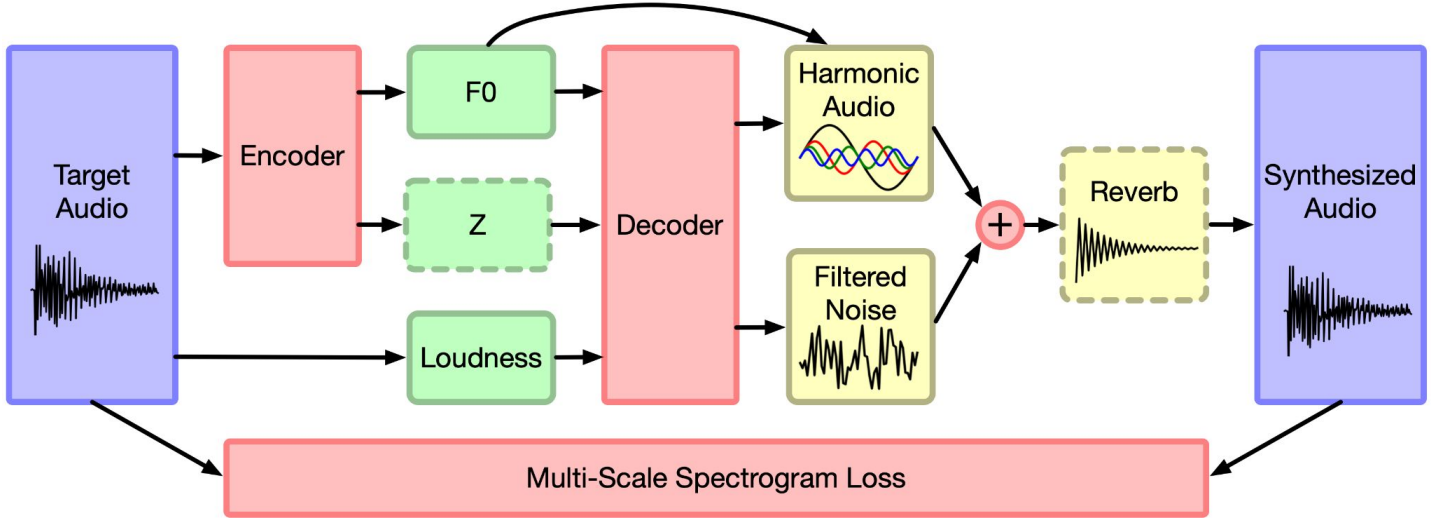
$$
\begin{aligned}
v(n) &= f_\theta \circ u(n) \\
&= f_1 \circ f_2 \circ \cdots \circ h * f_k \circ \cdots \circ u(n)
\end{aligned}
$$

$$
\min_\theta \mathcal{L}\left(v(0), v(1), \ldots, v(N-1)\right)
$$

$$
\theta \mapsto \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}
$$

$$
\frac{\partial \mathcal{L}}{\partial \theta} = \cdots + \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial h}\frac{\partial h}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial \theta} + \cdots
$$

Differentiable digital signal processing (DDSP)



Engel, Jesse, Chenjie Gu, and Adam Roberts. "DDSP: Differentiable Digital Signal Processing." International Conference on Learning Representations.

Case 2: All-pole filters

- a.k.a linear prediction filter
- Feedback coefficients only
- Has infinite-length impulse response (IIR)

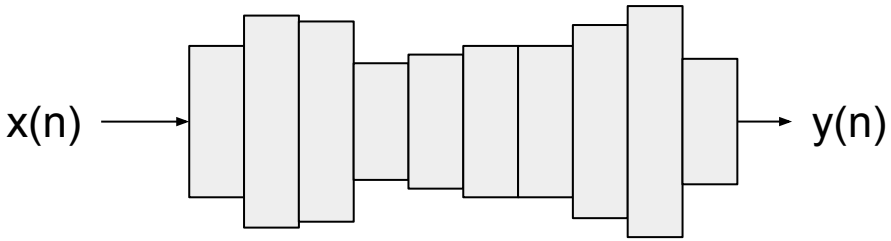
★ Voice synthesis
  - linear predictive coding (LPC)

★ ≈ air flow through a tube with varying diameter

★ ≈ resonators

$$y(n) = x(n) - \sum_{i=1}^{M} a_i y(n-i)$$

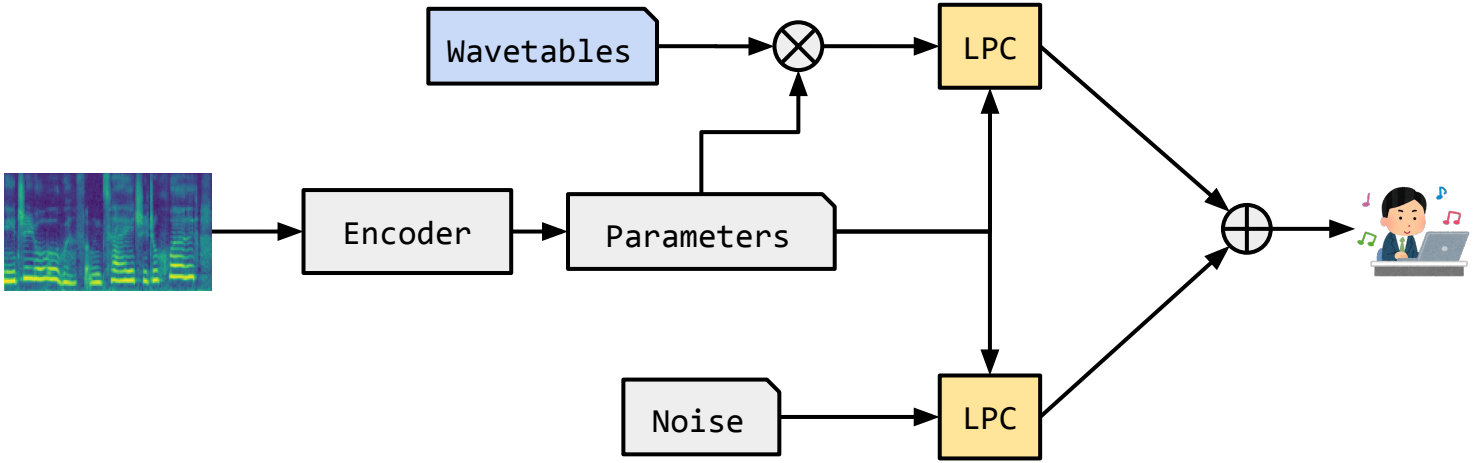
$$= x(n) + \sum_{i=1}^{\infty} h(i) x(n-i)$$



Pink Trombone by Neil Thapen.
https://dood.al/pinktrombone/

GlOttal-flow LPC Filter (GOLF) vocoder

Yu, Chin-Yun and George Fazekas. "GOLF: A Singing Voice Synthesiser with Glottal Flow Wavetables and LPC Filters." TISMIR.



LP Coefficients Parameterisation

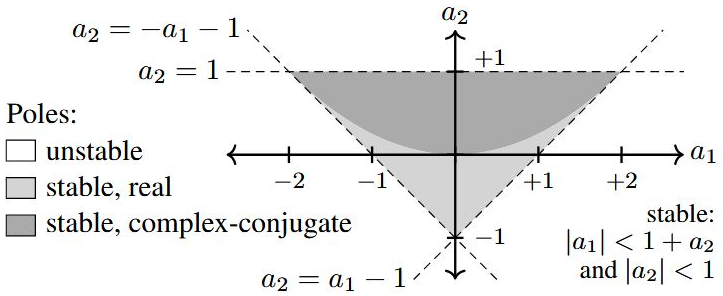


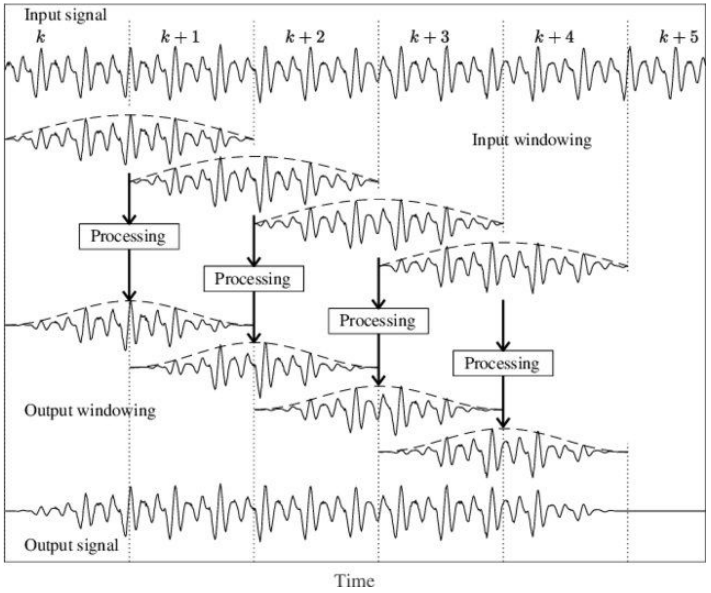
Fig. 1. Biquad triangle ensuring filter stability.

$$\frac{G}{1+\sum_{i=1}^{M} a_i z^{-i}} = G \prod_{k=1}^{\frac{M}{2}} \frac{1}{(1+\eta_{k,1} z^{-1}+\eta_{k,2} z^{-2})}$$

- Ensure poles are inside unit circle (stable LTI system)
- M = 22, sr = 24 kHz

Nercessian, Shahan, Andy Sarroff, and Kurt James Werner. "Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads." ICASSP 2021.

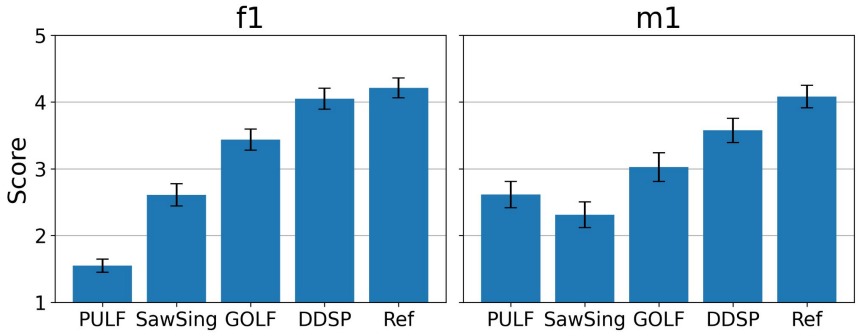
Frame-wise time-varying linear prediction



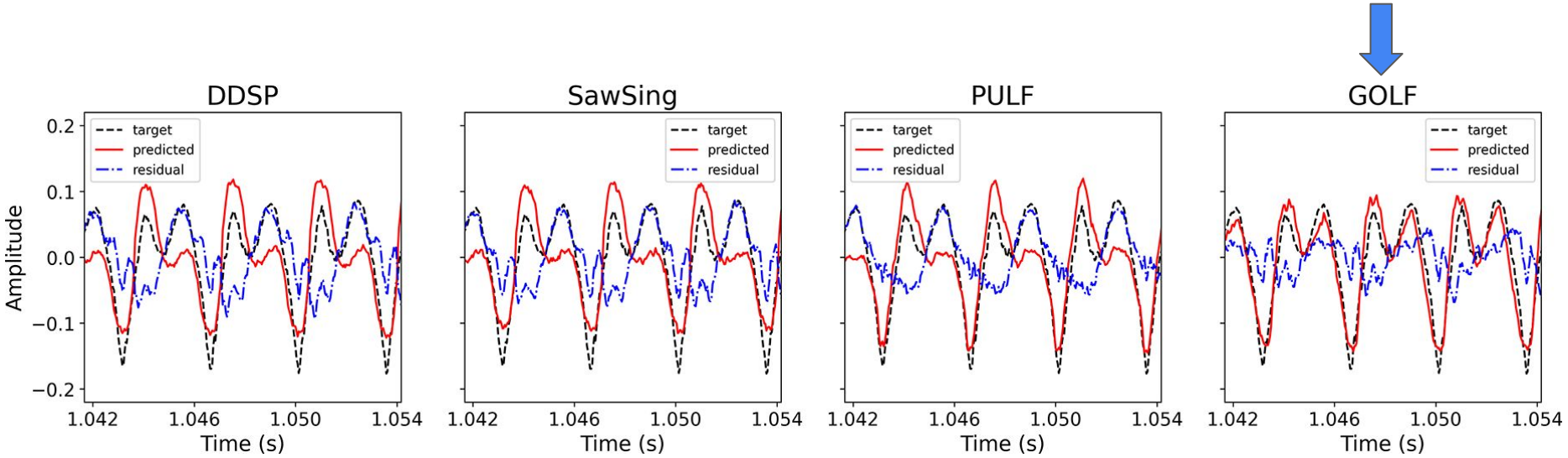
- Overlap-add and windowing
- Each frame is treated as LTI and synthesise **in parallel**

Evaluations

Singers	Models	MSSTFT	MAE-f0 (cent)	FAD
f1	DDSP	**3.09**	**74.47**±1.19	0.50±0.02
	SawSing	3.12	78.91±1.18	**0.38**±0.02
	GOLF	3.21	77.06±0.88	0.62±0.02
	PULF	3.27	76.90±1.11	0.75±0.04
m1	DDSP	**3.12**	**52.95**±1.03	0.57±0.02
	SawSing	3.13	56.46±1.04	**0.48**±0.02
	GOLF	3.26	54.09±0.30	0.67±0.01
	PULF	3.35	54.60±0.73	1.11±0.04



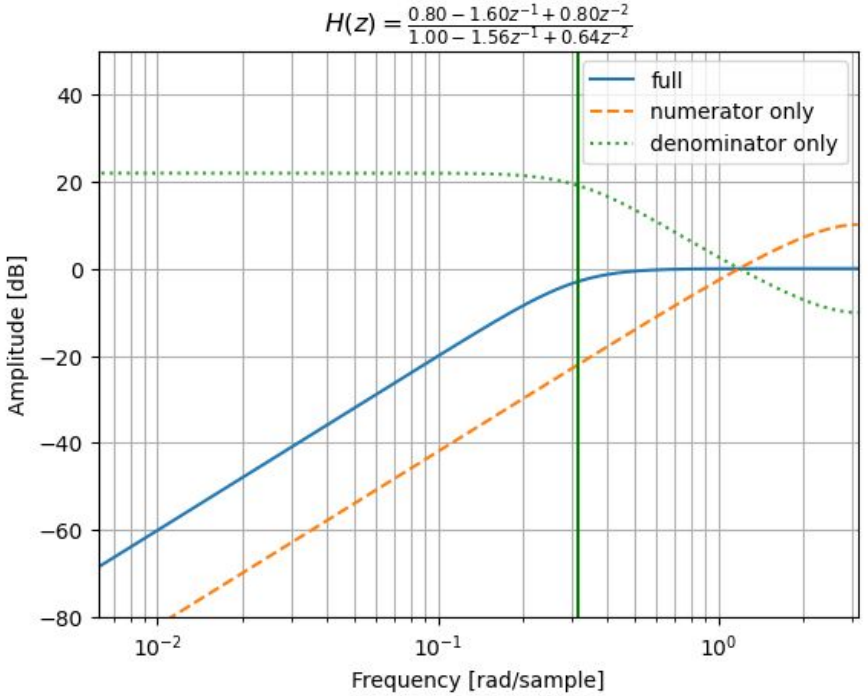
Waveform matching



Case 3: Biquad

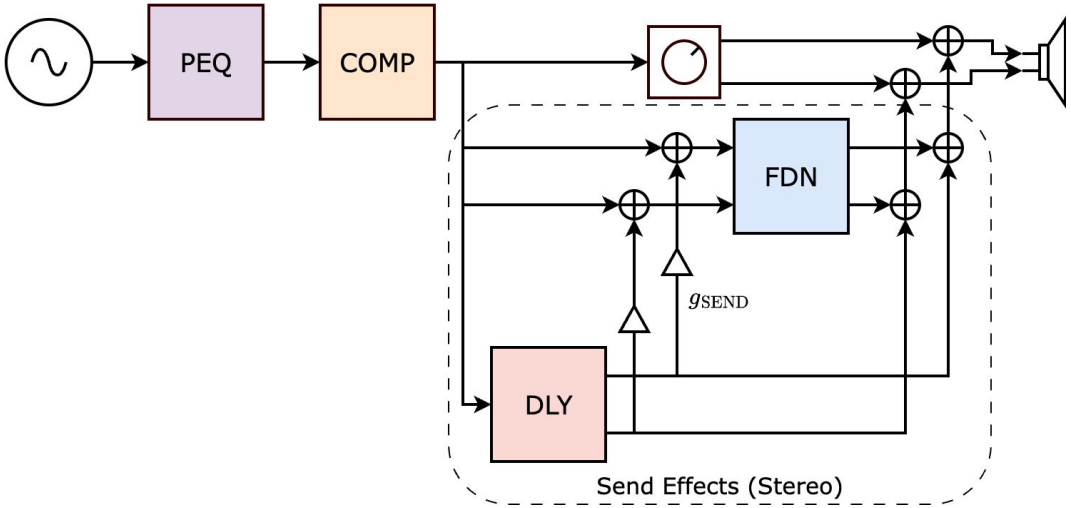
- Mixed feedforward & feedback sections
- Various filters
  - low/high/band pass
  - low/high shelves
  - peak
  - .etc

$$H_{\mathrm{BQ}}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$



Differentiable Vocal Fx (DiffVox)

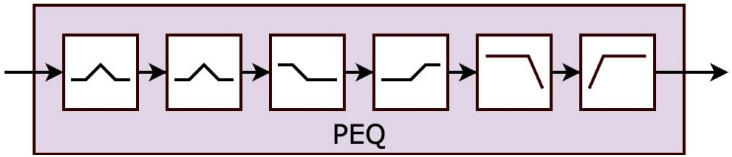
Yu, Chin-Yun, et al. "DiffVox: A differentiable model for capturing and analysing vocal effects distributions." Proc. DAFx. 2025.



Sony AI

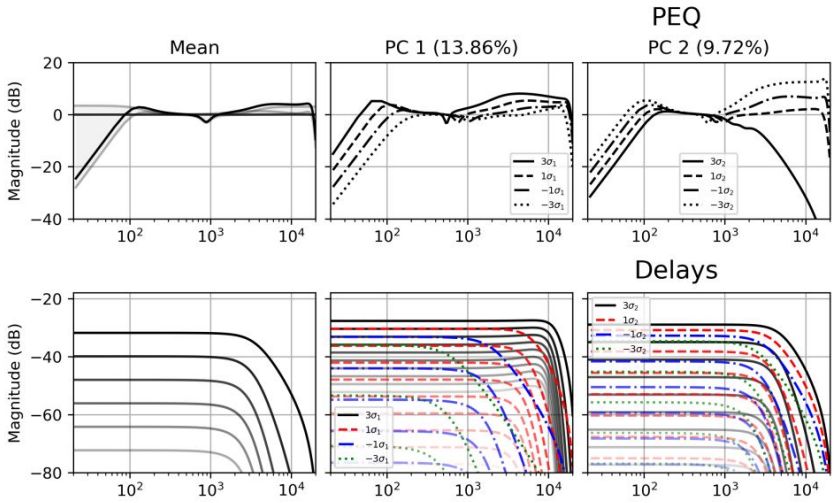
6-band parametric equaliser (PEQ)

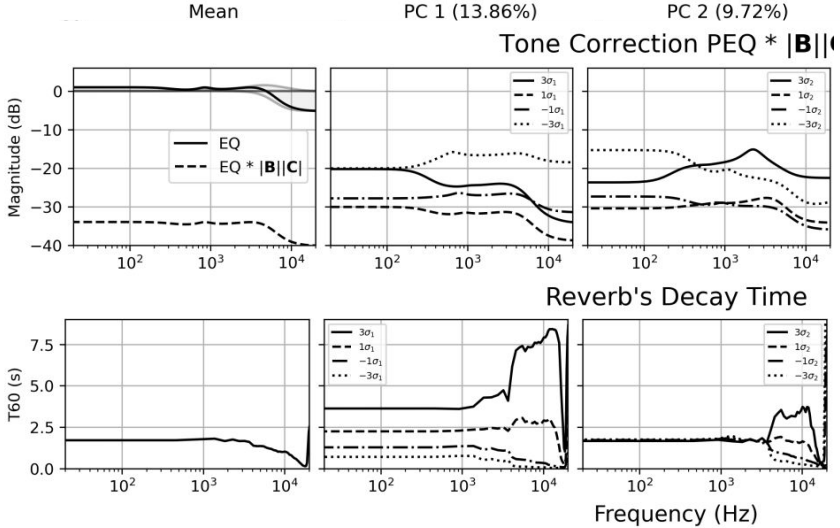
- Biquad filters
- peak filter x2, low-shelf, high-shelf, low-pass, high-pass
- ≈ IK Multimedia Classic Equalizer VST



Principal component analysis (PCA)

Fit DiffVox on Sony's 365 proprietary vocal tracks





Recursive filters (IIR)

Pros

- Unlimited context window → extremely expressive with a few coefficients
- Physical meaning (e.g., formant frequencies)
- Friendly to audio engineers

Cons

- Sequential computation (?) → non-trivial to parallelise (**are they?**)
- Can be unstable and cares should be taken

Part II:
Achieving fast automatic differentiation of filters

Issue in the most popular AD framework

- Popularity: PyTorch >>> JAX ≈ Tensorflow > etc.
- No naive ops for recursion → Slow to compute
  - ML people understand it as autoregression

`torch.nn.functional.conv1d`

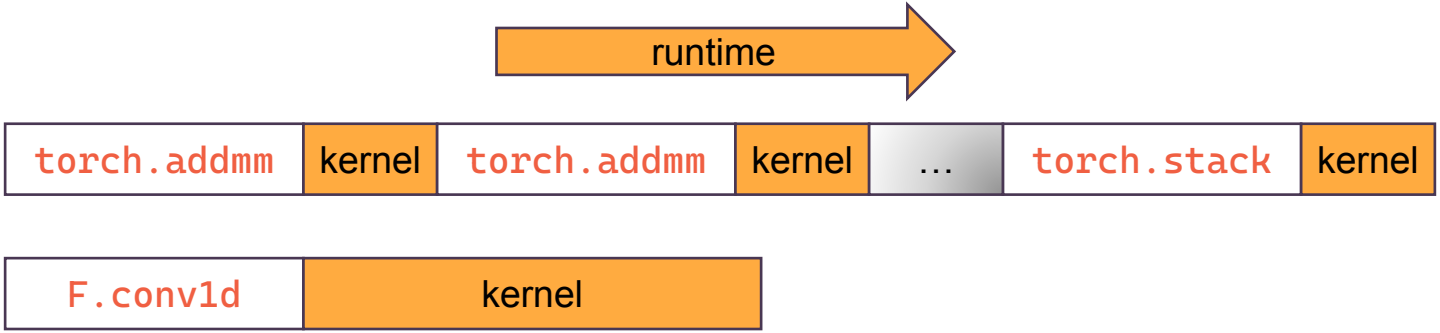
$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_M x(n-M) - a_1 y(n-1) - \cdots - a_M y(n-M)$$

`torch.* ?`

A naive implementation

```
h = zi
for xn in x.unbind(1):
    h = torch.addmm(xn, h, AT)
    results.append(h if out_idx is None else h[:, out_idx])
output = torch.stack(results, dim=1)
```

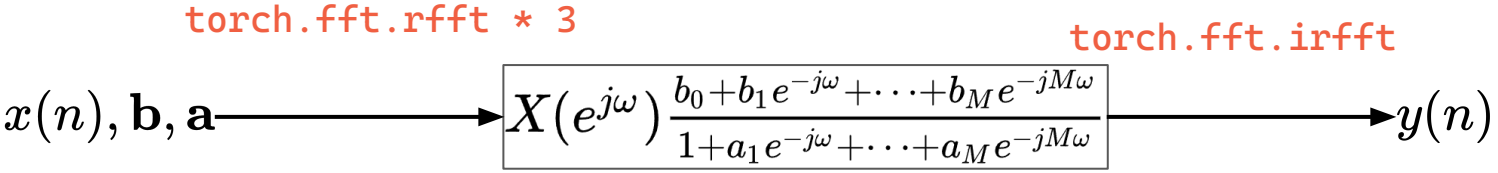
philtorch/lti/ssm.py, L127-131



Just-in-time (JIT) compilers

- Freeze the computation graph to optimise further
- `torch.jit.script/trace`
  - 2 ~ 3 times faster
  - Deprecated ≥ v2.9
- `torch.compile`
  - for-loop? **unroll them!** 🤯
  - million audio samples → your programme hang 4ever **during compilation**
  - Wu, Yidi, et al. "Control Flow Operators in PyTorch." 2025.
- Fundamental cause (personal opinion)
  - Eager mode first & entry-level friendly to beginner

Frequency sampling (FS)




```
y = irfft(
        rfft(x) * rfft(b, n=x.shape[0]) / rfft(a, n=x.shape[0]),
        n=x.shape[0]
)
```

Time aliasing

- S&S 101: **Sampling → periodic response**
- Duplicates IIR at location …,-2N, -N, 0, N, 2N,...
  - = folded and wrapped inside n ∈ [0, N)

$$y(n) = x(n) + \alpha y(n-1) \rightarrow \mathbf{h} = [1, \alpha, \alpha^2, \ldots]^\top$$

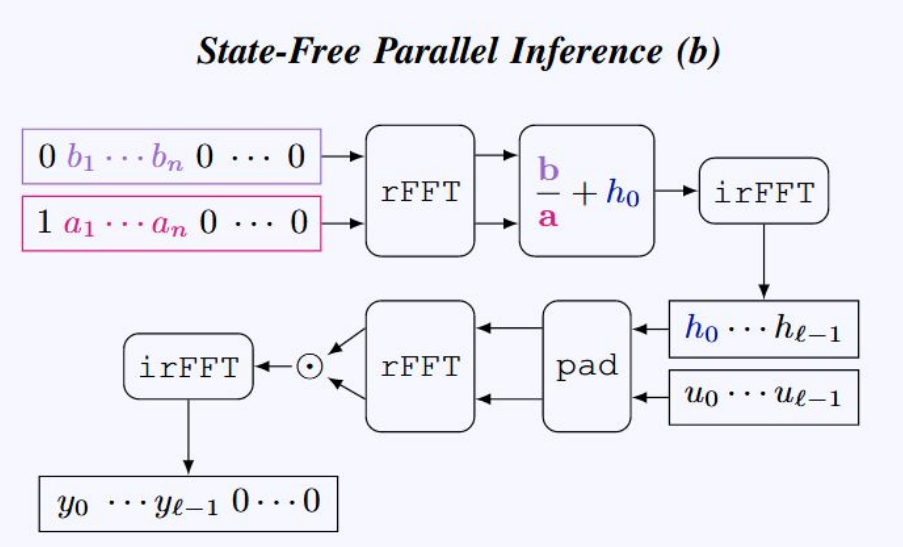
Frequency sampling

$$\begin{aligned}\hat{\mathbf{h}} = &[1, \alpha, \alpha^2, \ldots, \alpha^{N-1}]^\top \\ &+ [\alpha^N, \alpha^{N+1}, \alpha^{N+2}, \ldots, \alpha^{2N-1}]^\top \\ &+ [\alpha^{2N}, \alpha^{2N+1}, \alpha^{2N+2}, \ldots, \alpha^{3N-1}]^\top \\ &+ \ldots\end{aligned}$$

Circular convolution

- Periodic IR → circular convolution
- FIR: zero-padding to 2N - 1
- IIR: how? there's no end point!

Solution: IIR truncation



$$H(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + b_0$$
$$H_N(z) = \mathbf{C}(\mathbf{I} - \mathbf{A}^N)(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + b_0$$
$$\overset{\mathcal{Z}}{\longleftrightarrow} \mathbf{h}_N = [1, \alpha, \ldots, \alpha^{N-1}]^\top$$
$$\tilde{\mathbf{b}} = \mathbf{C}(\mathbf{I} - \mathbf{A}^N)$$

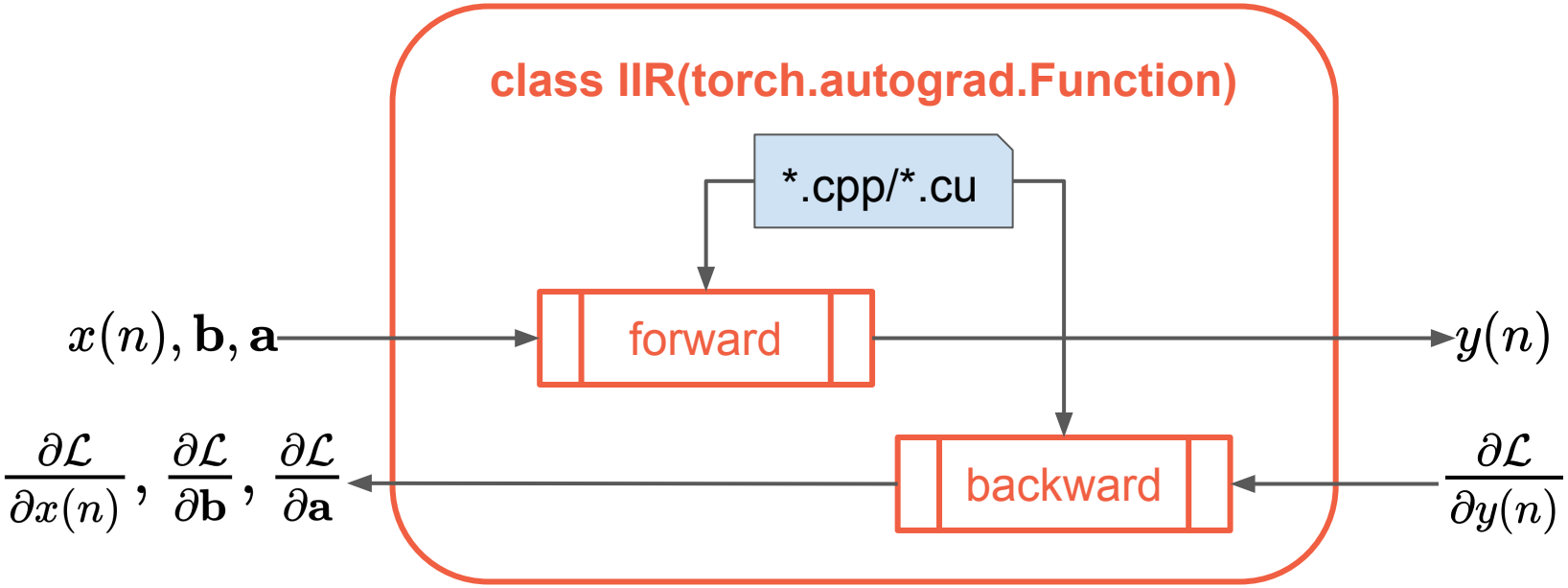
- **A:** companion matrix of the **a** polynomial
- Evaluate matrix exponential $\mathbf{A}^N$

Parnichkun, Rom N., et al. "State-free inference of state-space models: the transfer function approach." ICML. 2024.

Time domain filtering

- FS
  - Overkill when M <<<< N
  - Sensitive to numerical errors
  - Higher memory usage
- Direct computation is favourable for most filters

Make filter the first class citizen



https://intro2ddsp.github.io/filters/iir_torch.html

Step 1: Analytical gradients

Gradient backpropagation

Reverse-mode AD

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial \theta} + \dots$$

- Full jacobian ❌: expensive
- Vector-jacobian product ✅

$$\mathrm{VJP}_{\mathbf{J}} : \mathbf{v} \mapsto \mathbf{vJ}$$

$$\frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial \theta} = \left(\text{VJP}_{\frac{\partial b}{\partial \theta}} \circ \text{VJP}_{\frac{\partial y}{\partial b}} \circ \text{VJP}_{\frac{\partial \mathcal{L}}{\partial y}}\right) (1)$$

- Compute order: →
- Apply transformation **without transformation matrices**

Filtering as matrix transformation

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$\mathbf{W} = \begin{bmatrix} h(0) & 0 & 0 & \cdots & 0 \\ h(1) & h(0) & 0 & \cdots & 0 \\ h(2) & h(1) & h(0) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h(-N+1) & h(-N+2) & h(-N+3) & \cdots & h(0) \end{bmatrix}$$

Matrix calculus 101

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

forward filtering

$$\downarrow$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \mathbf{W} = \left(\mathbf{W}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{y}^\top} \right)^\top$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \mathbf{x} \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$

reverse filtering

The two-pass algorithm: for x

$$\mathbf{W}\mathbf{x} \longleftrightarrow \text{AllPole}(\mathbf{x})$$

$$\mathbf{W}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{y}^\top} \longleftrightarrow (\text{Flip} \circ \text{AllPole} \circ \text{Flip}) \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^\top} \right)$$

Yu, Chin-Yun, and György Fazekas. "Singing Voice Synthesis Using Differentiable LPC and Glottal-Flow-Inspired Wavetables." ISMIR. 2023.
Forgione, Marco, and Dario Piga. "dynoNet: A neural network architecture for learning dynamical systems." International Journal of Adaptive Control and Signal Processing 35.4 (2021): 612-626.

The two-pass algorithm: for a

$$\frac{\partial y(n)}{\partial a_i} = -y(n-i) - \sum_{k=1}^{M} a_k \frac{\partial y(n-k)}{\partial a_i}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{a}} = \mathrm{AllPole}(-\mathbf{y})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{a}}$$

- Filtering x2
- Matrix multiplications x1

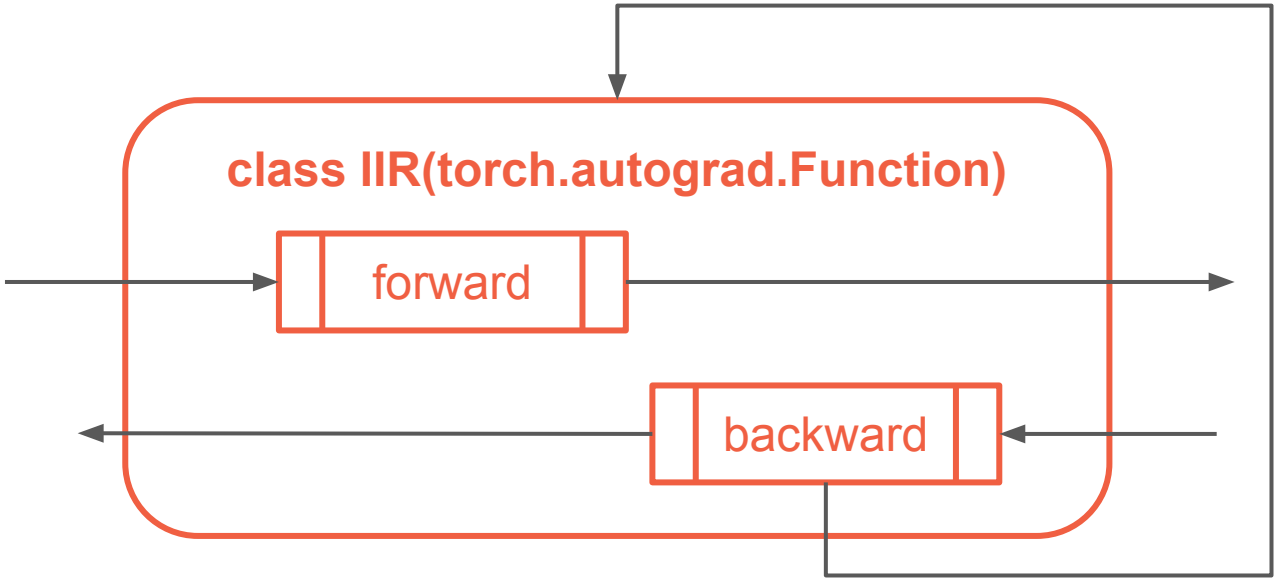
The one-pass algorithm

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = -\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \begin{bmatrix} y(-1) & y(-2) & \cdots & y(-M) \\ y(0) & y(-1) & \cdots & y(-M+1) \\ \vdots & \vdots & \ddots & \vdots \\ y(N-2) & y(N-3) & \cdots & y(N-M-1) \end{bmatrix}$$

- Filtering x1
- Matrix multiplications x1
- TorchAudio `lfilter`

Yu, C.Y., Mitcheltree, C., Carson, A., Bilbao, S., Reiss, J.D. and Fazekas, G., 2024. Differentiable All-pole Filters for Time-varying Audio Systems. DAFx 2024.

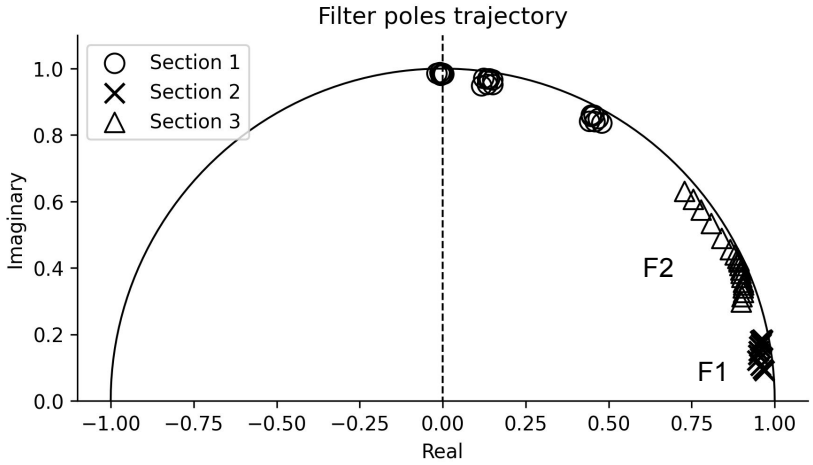
N-order gradients are recursively defined

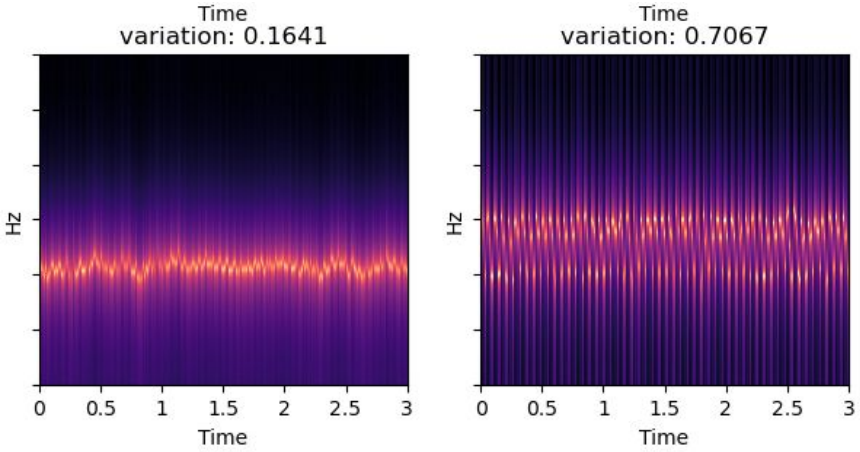


You only implement once (YOIO)

Extending to parameter-varying systems

~ to remove frame-wise approximation errors in GOLF





- Each point is covered by four frames (75 % overlap)
- Poles are interleaving between three groups (<4)

Yu, Chin-Yun and George Fazekas. "GOLF: A Singing Voice Synthesiser with Glottal Flow Wavetables and LPC Filters." TISMIR.

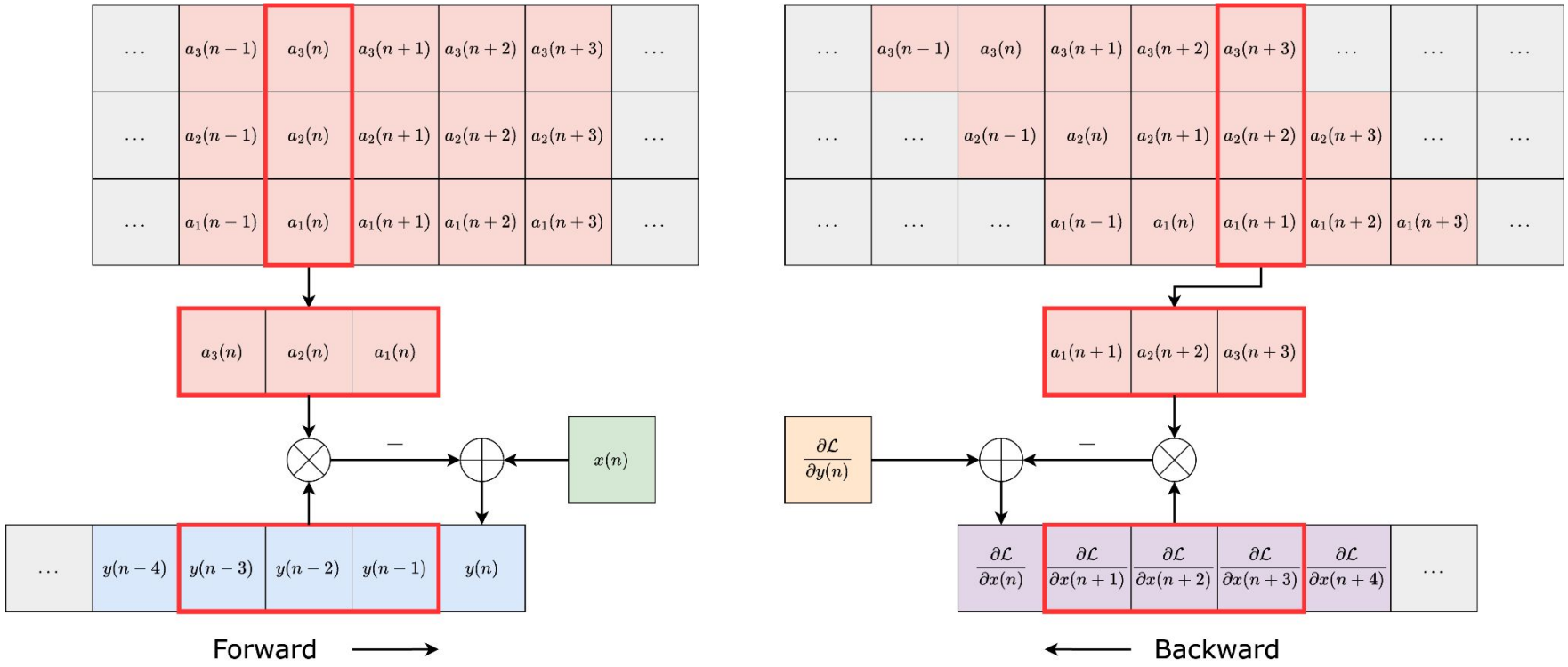
Gradients of a LPV all-pole

$$y(n) = x(n) - \sum_{i=1}^{M} a_i(n) y(n-i)$$

$$\frac{\partial \mathcal{L}}{\partial x(n)} = \frac{\partial \mathcal{L}}{\partial y(n)} - \sum_{i=1}^{M} a_i(n+i) \frac{\partial \mathcal{L}}{\partial x(n+i)}$$

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

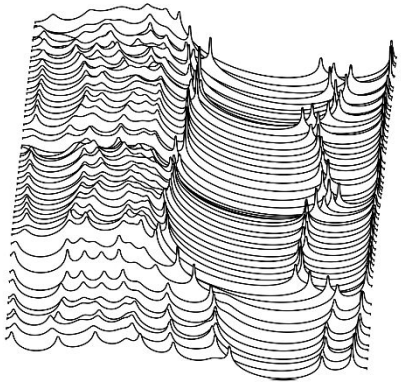
Yu, C.Y., Mitcheltree, C., Carson, A., Bilbao, S., Reiss, J.D. and Fazekas, G., 2024. Differentiable All-pole Filters for Time-varying Audio Systems. DAFx 2024.



Yu, C.Y., Mitcheltree, C., Carson, A., Bilbao, S., Reiss, J.D. and Fazekas, G., 2024. Differentiable All-pole Filters for Time-varying Audio Systems. DAFx 2024.

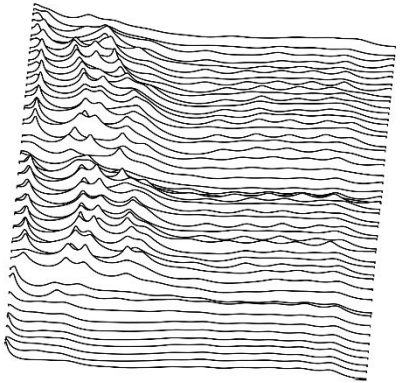
A better GOLF vocoder

GOLF-v1



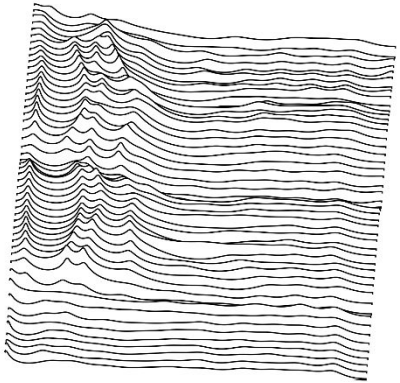
Harmonic-plus-noise
frame-wise LP

GOLF-ff



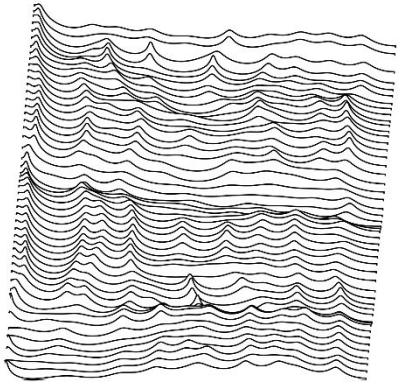
Source-filter
frame-wise LP

GOLF-ss



Source-filter
sample-wsie LP

SPTK

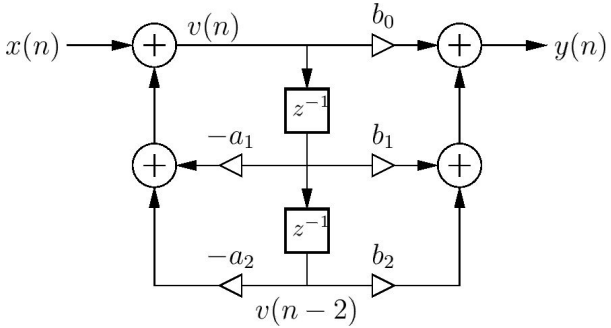


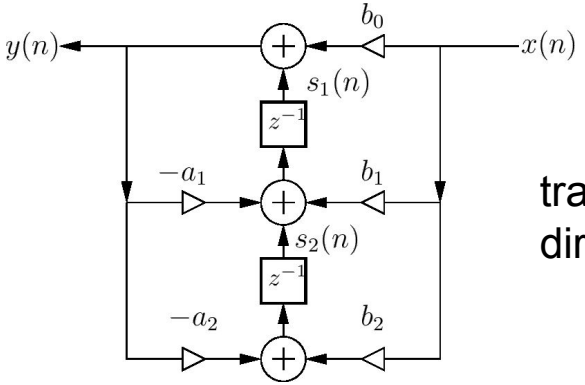
Yu, Chin-Yun, and György Fazekas. "Differentiable Time-Varying Linear Prediction in the Context of End-to-End Analysis-by-Synthesis." Proc. Interspeech 2024.

Limitations

- Not all direct forms are FIR/all-pole separable
  - DF-I/II ✅ (TorchAudio)
  - TDF-I/II ❌ (SciPy)
- Sequential computation → not fully utilise GPU 💪

direct form





transposed direct form

The state-space realisation (of DF-II)

$$\mathbf{v}(n+1) = \mathbf{A}\mathbf{v}(n) + \mathbf{B}x(n)$$
$$y(n) = \mathbf{C}^\top \mathbf{v}(n) + b_0 x(n)$$
$$\mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} b_1 - a_1 b_0 \\ b_2 - a_2 b_0 \\ \vdots \\ b_M - a_M b_0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} -\mathbf{a} \\ \mathbf{I} \quad \mathbf{0} \end{bmatrix}$$

TDF-II state-space form

$$\mathbf{s}(n+1) = \mathbf{A}^\top \mathbf{s}(n) + \mathbf{C}x(n)$$
$$y(n) = \mathbf{B}^\top \mathbf{v}(n) + b_0 x(n)$$

Duality: The two sides of the same coin

TDF-II, incremental in time

$$\mathbf{v}(n+1) = \mathbf{A}\mathbf{v}(n) + \mathbf{u}(n)$$

$$\downarrow$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n)} = \mathbf{A}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n+1)} + \frac{\partial \mathcal{L}}{\partial \mathbf{v}^\top(n+1)}$$

DF-II, decremental in time

DF-II, incremental in time

$$\mathbf{s}(n+1) = \mathbf{A}^\top \mathbf{s}(n) + \mathbf{u}(n)$$

$$\downarrow$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n)} = \mathbf{A} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n+1)} + \frac{\partial \mathcal{L}}{\partial \mathbf{s}^\top(n+1)}$$

TDF-II, decremental in time

Yu, Chin-Yun, and György Fazekas. "Accelerating Automatic Differentiation of Direct Form Digital Filters." DIffSys Workshop @EurIPS 2025.

LPV state-space

$$\mathbf{v}(n+1) = \mathbf{A}(n)\mathbf{v}(n) + \mathbf{u}(n)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n)} = \mathbf{A}^\top(n)\frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n+1)} + \frac{\partial \mathcal{L}}{\partial \mathbf{v}^\top(n+1)}$$

$$\mathbf{A}, a_i(n) \longleftrightarrow \mathbf{A}^\top, a_i(n+i)$$

https://iamycy.github.io/posts/2025/04/differentiable-tdf-ii/

Step 2: Parallel implementations

SSM is parallelisable!

Sequential so non-parallelisable 😞

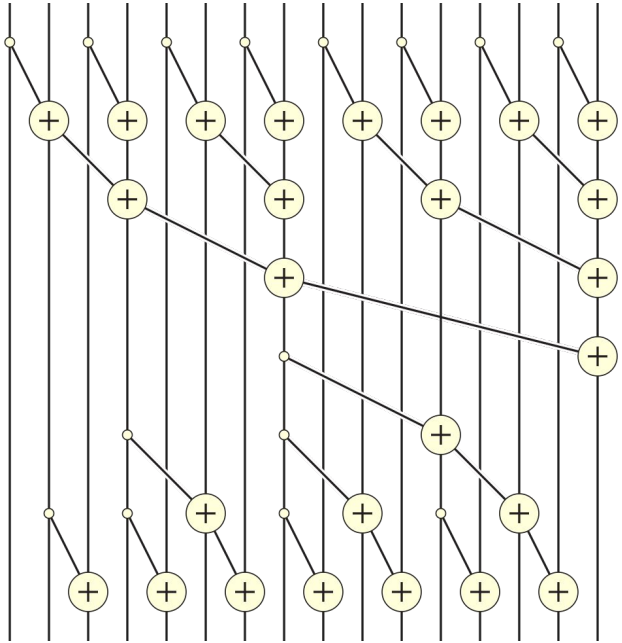
$$\begin{aligned}\mathbf{v}(n+1) =& \mathbf{A}(n)(\mathbf{A}(n-1)(\mathbf{A}(n-2)(\ldots) \\ &+ \mathbf{u}(n-2)) + \mathbf{u}(n-1)) + \mathbf{u}(n)\end{aligned}$$

Parallelisable 😍

$$(\mathbf{U}, \mathbf{x}) \oplus (\mathbf{V}, \mathbf{z}) \mapsto (\mathbf{VU}, \mathbf{Vx} + \mathbf{z})$$

$$\begin{aligned}(\mathbf{0}, \mathbf{v}(n+1)) =& (\mathbf{0}, \mathbf{v}(0)) \oplus (\mathbf{A}(0), \mathbf{u}(0)) \oplus (\mathbf{A}(1), \mathbf{u}(1)) \\ &\oplus \cdots \oplus (\mathbf{A}(N), \mathbf{u}(N))\end{aligned}$$

Parallel associative scan



- Prefix sum/inclusive scan
- `std::inclusive_scan`
- `thrust::inclusive_scan`
- Doable using `torch.matmul`
  - https://iamycy.github.io/posts/2025/06/28/unroll-ssm/
- O(N) → O(log(N))

https://en.wikipedia.org/wiki/Prefix_sum

Even faster: Diagonalised SSM (LTI only)

$$\tilde{\mathbf{v}}(n+1) = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_M \end{bmatrix} \tilde{\mathbf{v}}(n) + \mathbf{P}^{-1}\mathbf{u}(n)$$

$$\mathbf{v}(n) = \mathbf{P}\tilde{\mathbf{v}}(n), \quad \mathbf{A} = \mathbf{P} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_M \end{bmatrix} \mathbf{P}^{-1}$$

Yu, Chin-Yun, et al. "DiffVox: A differentiable model for capturing and analysing vocal effects distributions." Proc. DAFx. 2025.

PFE interpretation of diagonalised SSM

- Partial Fraction Expansion
- M first-order filters run in parallel
  - or M/2 biquads

$$H(z) = b_0 + z^{-1} \sum_{i=1}^{M} \frac{\tilde{c}_i \tilde{b}_i}{1 - \lambda_i z^{-1}}$$

$$= b_0 + \sum_{i=1}^{\frac{M}{2}} \frac{r_{i,1} z^{-1} + r_{i,2} z^{-2}}{1 + a_{i,1} z^{-1} + a_{i,2} z^{-2}}$$

Part II Summary

- Number of kernel calls is **input-dependent** → pack them into one kernel
  - so it is **input-independent**
- Backpropagation of filters = filter the gradients backward in time
- Linear **recursive filters are parallelisable** across time dimension

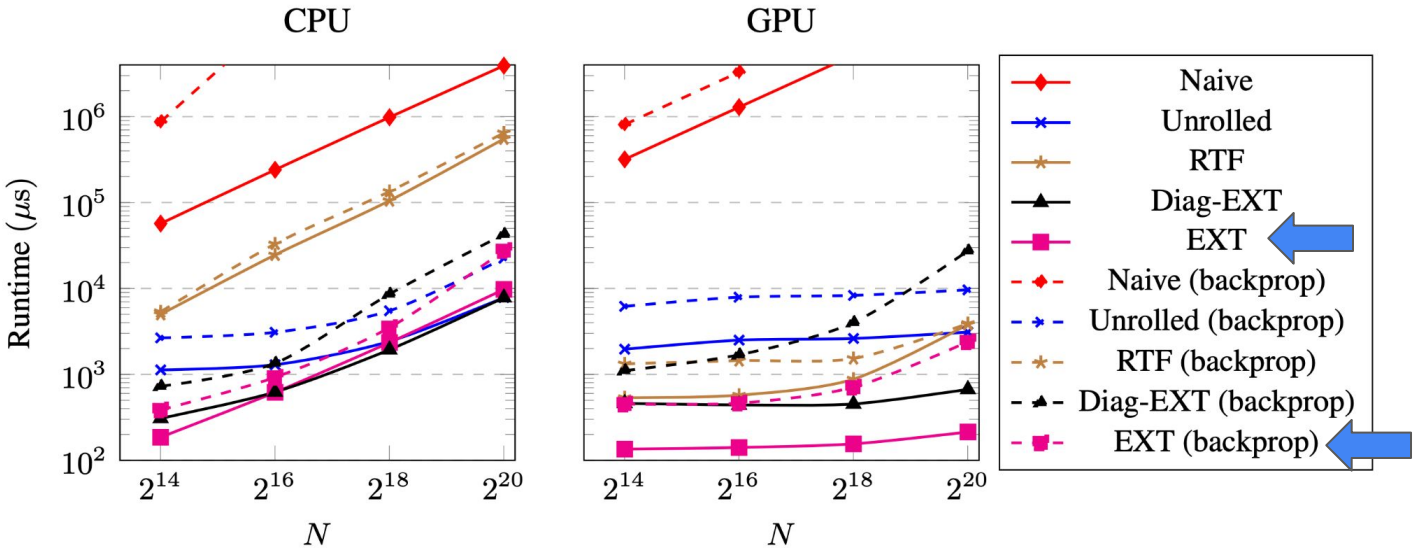
PhilTorch Φ🔥

- Fast and differentiable digital filters/SSMs on GPU
- Same interface and numerical robustness as SciPy
- LTI & LPV systems
- Arbitrary orders of gradients

```
pip install philtorch
```

Code

Benchmarks (biquad)



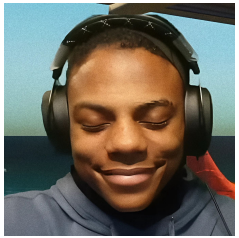
Yu, Chin-Yun, and György Fazekas. "Accelerating Automatic Differentiation of Direct Form Digital Filters." DiffSys Workshop @EurIPS 2025.

Part III:
Unsolved challenges and promising directions

Topics

1. **Tasks that have parallel filters**
2. Non-linear filters
3. Forward-mode gradient computations
4. **Learnable delay lengths**
5. **FDN**
6. LPV stability
7. Gradients' numerical accuracy

Parallel form >>>>> others

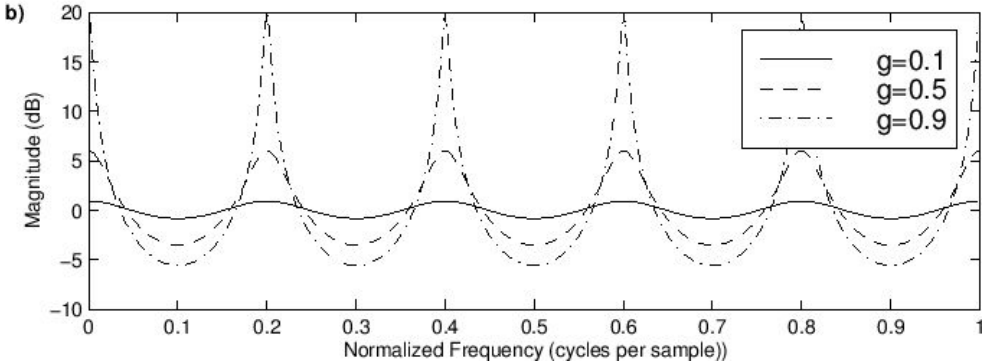


- For fast development we need ***Speed!!!***
- Work on tasks that have parallel forms
- Avoid repeated poles as much as possible

$$H(z) = b_0 + \sum_{i=1}^{\frac{M}{2}} \frac{r_{i,1} z^{-1} + r_{i,2} z^{-2}}{1 + a_{i,1} z^{-1} + a_{i,2} z^{-2}}$$

Example 1: Feedback comb filter

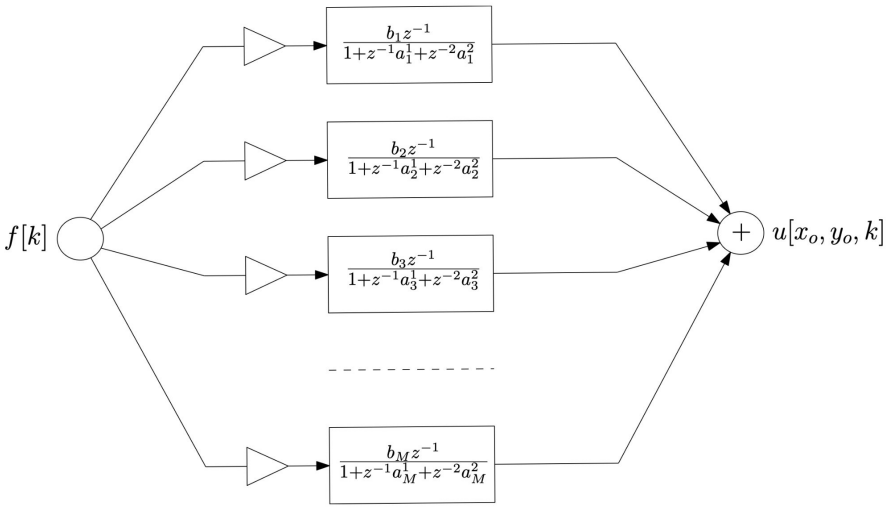
$$
\begin{aligned}
y(n) &= x(n) - \alpha y(n-M) \\
&\downarrow \\
y_1(n) &= x(nM) - \alpha y_1(n-1) \\
y_2(n) &= x(nM+1) - \alpha y_2(n-1) \\
&\vdots \\
y_M(n) &= x(nM+M-1) - \alpha y_M(n-1) \\
\mathbf{y} &= \text{Interleave}(\mathbf{y}_1, \ldots, \mathbf{y}_M)
\end{aligned}
$$

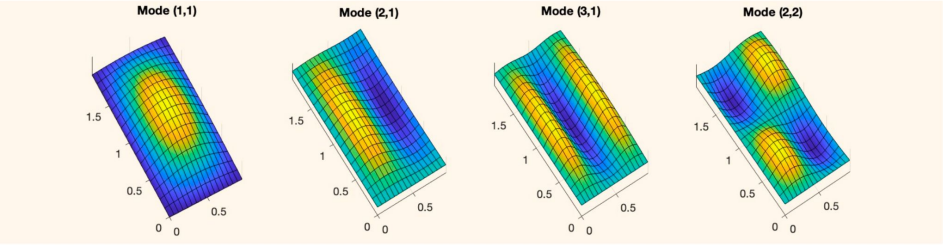


`philtorch.lti.comb_filter`

https://www.dsprelated.com/freebooks/pasp/Comb_Filters.html

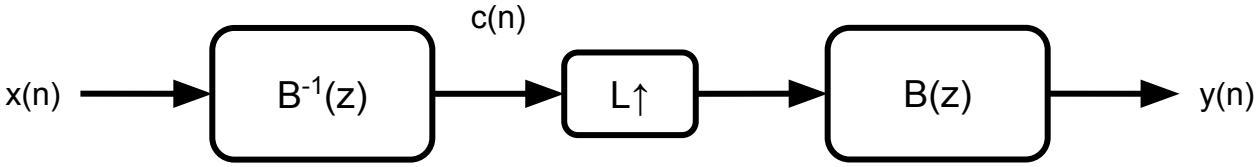
Example 2: Modal synthesis of a plate reverb





- 1st DAFx Parameter Estimation Challenge
  - now ~ 31.5.2026
- https://github.com/LOGUNIVPM/1st-DAFx-Challenge

Example 3: Cubic spline interpolation



$$B^{-1}(z) = \frac{6}{z + 4 + z^{-1}}$$
$$= \frac{-6\alpha}{(1 - \alpha z^{-1})(1 - \alpha z)}$$
$$= \frac{-6\alpha}{1 - \alpha^2}\left(\frac{1}{1 - \alpha z^{-1}} + \frac{1}{1 - \alpha z} - 1\right)$$

`philtorch.lti.cubic_spline`

Unser, Michael, Akram Aldroubi, and Murray Eden. "Fast B-spline transforms for continuous image representation and interpolation." IEEE Transactions on pattern analysis and machine intelligence 13.3 (1991): 27

Accelerate AD of non-linear filters

- Their forward pass cannot be parallelised
- …but their backprop can!
- Challenges: dense Jacobians

$$\mathbf{v}(n+1) = f(\mathbf{v}(n)) + \mathbf{u}(n)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n)} = \sum_{i=n+1}^{N} \left(\prod_{j=n+1}^{i-1} \left(\frac{\partial \mathbf{v}(j+1)}{\partial \mathbf{v}(j)} \right)^\top \right) \frac{\partial \mathcal{L}}{\partial \mathbf{v}^\top(i)}$$

$$= \left(\frac{\partial \mathbf{v}(n+2)}{\partial \mathbf{v}(n+1)} \right)^\top \frac{\partial \mathcal{L}}{\partial \mathbf{u}^\top(n+1)} + \frac{\partial \mathcal{L}}{\partial \mathbf{v}^\top(n+1)}$$

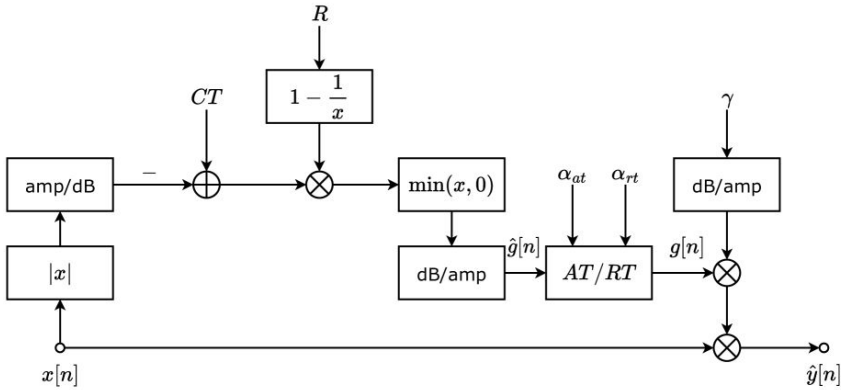
https://justintchiu.com/blog/pscan_diff/

Wang, Shang, Yifan Bai, and Gennady Pekhimenko. "BPPSA: Scaling back-propagation by parallel scan algorithm." Proceedings of Machine Learning and Systems 2 (2020): 451-469.

Example: Differentiable gain reduction in a compressor

$$g(n) = \begin{cases} \alpha_{at}\,\hat{g}(n) + (1-\alpha_{at})\,g(n-1) & \text{if } \hat{g}(n) < g(n-1), \\ \alpha_{rt}\,\hat{g}(n) + (1-\alpha_{rt})\,g(n-1) & \text{otherwise,} \end{cases}$$

$$\frac{\partial g(n)}{\partial g(n-1)} = \begin{cases} 1-\alpha_{at} & \text{if } \hat{g}(n) < g(n-1), \\ 1-\alpha_{rt} & \text{otherwise,} \end{cases}$$



Yu, Chin-Yun, and György Fazekas. "Sound Matching an Analogue Levelling Amplifier Using the Newton-Raphson Method." AES AIMLA. 2025.
Chin-Yun Yu, Christopher Mitcheltree, Alistair Carson, Stefan Bilbao, Joshua D. Reiss, and György Fazekas, “Differentiable all-pole filters for time-varying audio systems,” in DAFx, 2024, pp. 345–352.

Forward-mode AD

Newton's method needs second-order information

$$\theta \mapsto \theta - \left(\frac{\partial^2 \mathcal{L}}{\partial^2 \theta}\right)^{-1} \frac{\partial \mathcal{L}}{\partial \theta}$$

- The Hessian = the Jacobian of backprop = vector-to-vector function
- Jacobian-vector product (JVP) 🏆

$$\mathrm{JVP}_{\mathbf{J}} : \mathbf{v} \mapsto \mathbf{J}\mathbf{v}$$

reverse-mode AD (backprop)

$$\frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial \theta} = \left(\mathbf{VJP}_{\frac{\partial b}{\partial \theta}} \circ \mathbf{VJP}_{\frac{\partial y}{\partial b}} \circ \mathbf{VJP}_{\frac{\partial \mathcal{L}}{\partial y}}\right)(1)$$

$$= \left(\mathbf{JVP}_{\frac{\partial \mathcal{L}}{\partial y}} \circ \mathbf{JVP}_{\frac{\partial y}{\partial b}} \circ \mathbf{JVP}_{\frac{\partial b}{\partial \theta}}\right)(1)$$

forward-mode AD

JVP of a scalar recursion

… is the same filter forward in time.

$$y(n) = x(n) + \alpha(n)y(n-1)$$

$$\frac{\partial y(n)}{\partial \theta} = \frac{\partial x(n)}{\partial \theta} + \frac{\partial a(n)}{\partial \theta} y(n-1) + \alpha(n)\frac{\partial y(n-1)}{\partial \theta}$$

Yu, Chin-Yun, and György Fazekas. "Sound Matching an Analogue Levelling Amplifier Using the Newton-Raphson Method." AES AIMLA. 2025.

Different Hessian compute strategies

Table 1: The memory cost (in MB) and runtime (in milliseconds) of different hessian computation strategies in PyTorch.

	`autograd`	`func`			
	rev-rev	rev-rev	fwd-rev	rev-fwd	fwd-fwd
Memory	**1066**	2358	3534	6306	5364
Time	**26.5**	28.1	28.1	64.7	38.9

Yu, Chin-Yun, and György Fazekas. "Sound Matching an Analogue Levelling Amplifier Using the Newton-Raphson Method." AES AIMLA. 2025.

Differentiable continuous delay length

Learning continuous delay time of a digital comb filter

$$y(n) = x(n) + \alpha y(n - M)$$

Frequency sampling: learning a sinusoidal

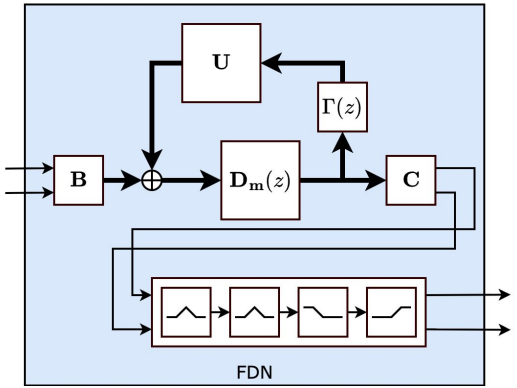
$$H(e^{j\omega}) = \frac{1}{1-\alpha e^{-j\omega M}}$$

Time domain learning: non-trivial

- Ideal continuous time filter: **anti-causal, infinite order**
- Proposed: work on the poles directly

Fast AD of Feedback Delay Network

- Artificial reverbereration
- Fast time domain training: possible
- Challenges
  - super high order (>10000) thus non-trivial to parallelise
  - differentiable delay lines



$$\begin{bmatrix} v_1(n+d_1) \\ v_2(n+d_2) \\ \vdots \\ v_M(n+d_M) \end{bmatrix} = \mathbf{A}\mathbf{v}(n) + \mathbf{B}x(n)$$

Parameter-varying stability

- LTI systems: coefficient parametrisation is enough
  - e.g., Biquad triangle
- LPV systems: LTI stable ≠ LPV stable
  - Forms that guarantee stability
    - **Parallel filters**
    - Lattice filters

Gradients' numerical robustness

- TDF-II → more robust in forward pass
- DF-II → more robust in backpropagation
- Can we have the best of both worlds?
- Suggestion
  - TDF-II: forward-mode AD
  - DF-II reverse-mode AD
- Needs some benchmarks…

Special thanks

- Christopher Mitcheltree, Ben Hayes, and Rodrigo Diaz, for helpful feedback and comments from our conversations.
- Julius Smith for his valuable online materials about filters.

Resources

1) Yu, C.Y., Mitcheltree, C., Carson, A., Bilbao, S., Reiss, J.D. and Fazekas, G., 2024. Differentiable all-pole filters for time-varying audio systems. DAFx 2024
2) Yu, C.-Y., Fazekas, G., 2024. Differentiable Time-Varying Linear Prediction in the Context of End-to-End Analysis-by-Synthesis. Proc. Interspeech 2024, 1820-1824
3) Yu, C.-Y. and Fazekas, G., 2024 ‘GOLF: A Singing Voice Synthesiser with Glottal Flow Wavetables and LPC Filters’, TISMIR, 7(1)
4) Yu, C.-Y. and Fazekas, G., 2025. "Sound Matching an Analogue Levelling Amplifier Using the Newton-Raphson Method." AES AIMLA 2025
5) Yu, C.-Y., . "DiffVox: A differentiable model for capturing and analysing vocal effects distributions." Proc. DAFx. 2025.
6) Yu, Chin-Yun, and György Fazekas. "Accelerating Automatic Differentiation of Direct Form Digital Filters." DIffSys Workshop @EurIPS 2025.
7) Yu, Chin-Yun. "PhilTorch: Accelerating Automatic Differentiation of Digital Filters In PyTorch". ADC 2025
8) Yu, C.-Y., 2025. Notes on differentiable TDF-II filter. https://iamycy.github.io/posts/2025/04/differentiable-tdf-ii/
9) Smith, J.O. Introduction to Digital Filters with Audio Applications, http://ccrma.stanford.edu/~jos/filters/, online book, 2007 edition