Electronic Music Studios (EMS) and the EMS Synthi: Research on Filter Design

by George Howard Redpath Student: 20246519

9/02/25

1 Introduction

Electronic Music Studios (EMS) was a British company founded in 1969 by Peter Zinovieff, David Cockerell, and Tristram Cary. EMS was established with the goal of producing affordable and innovative synthesizers for musicians and composers, particularly in the emerging field of electronic music.

For resources mentioned in this article please use: https://github.com/Ziforge/EMSSYNTHI

2 Early Years and Development

Before EMS was officially founded, Peter Zinovieff had already been experimenting with computer-based music composition in the mid-1960s. He operated a private electronic music studio in London and collaborated with other pioneers in electronic sound synthesis. EMS was officially formed when Zinovieff, along with Tristram Cary (a composer known for his work on early electronic scores, including *Doctor Who*), and David Cockerell (an electronics engineer), designed a commercially viable synthesizer.

3 The VCS3 and the Birth of the Synthi Series

The first product from EMS, the **VCS3 (Voltage Controlled Studio 3)**, was introduced in 1969. It was a compact, affordable synthesizer that quickly gained popularity among musicians and experimental composers. Unlike the massive modular synthesizers from companies like Moog, the VCS3 was portable and used a unique **pin-matrix** patching system, eliminating the need for patch cables.

Building on the success of the VCS3, EMS developed the **Synthi** series in the early 1970s. The **Synthi A** (1971) was a more compact, suitcase-style synthesizer that was essentially a portable version of the VCS3. The **Synthi AKS** followed in 1972, adding a built-in **sequencer and touch keyboard**, making it a popular tool for experimental and progressive rock musicians.

4 Filter Design and Technical Innovations

One of the standout features of EMS synthesizers was their **filter design**. EMS developed a highly distinctive **diode ladder filter**, differing from the transistor ladder filters used by Moog. The diode ladder filter provided a unique, slightly more aggressive resonance and a different character in sound shaping, which became a signature of EMS synthesizers.

David Cockerell, the primary engineer behind many EMS designs, ensured that the filter had a **nonlinear response**, giving EMS synths their distinctive warm and sometimes unpredictable timbre. Unlike traditional low-pass filters that were popular in other synthesizer designs, EMS filters were known for their ability to **self-oscillate** and produce rich harmonic textures when pushed to their limits.

5 Data Collection for Neural Network Training

To create an accurate digital model of the EMS Synthi filter, a dataset of approximately 1,500 Dirac delta impulse responses was collected at different resonance levels. The filter used for this data collection was a EMS Synthi filter designed by SoundFreak, closely replicating the behavior of the original analog circuit.

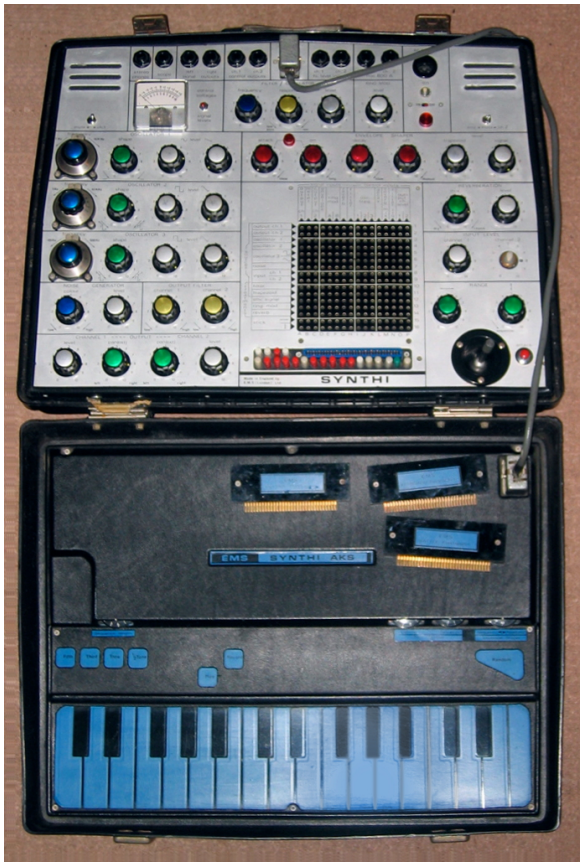

Figure 1: By Original uploader was Guy Hatton at en.wikipedia - Transferred from en.wikipedia, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=8401862`. Synthi AKS.

5.1 Impulse Response Capture Methodology

The impulse responses were recorded at three key resonance levels:

- 0% Resonance - Capturing the filter's raw frequency response in its most linear state.
- 50% Resonance - Mid-level resonance where nonlinearities start influencing the response.
- 100% Resonance - Maximum resonance, emphasizing self-oscillation and saturation effects.

Each impulse response was generated using a Dirac delta function input signal, ensuring that the dataset contained highly detailed frequency and phase information. The captured responses provide an accurate representation of how the filter behaves dynamically across different cutoff frequencies and resonance settings.

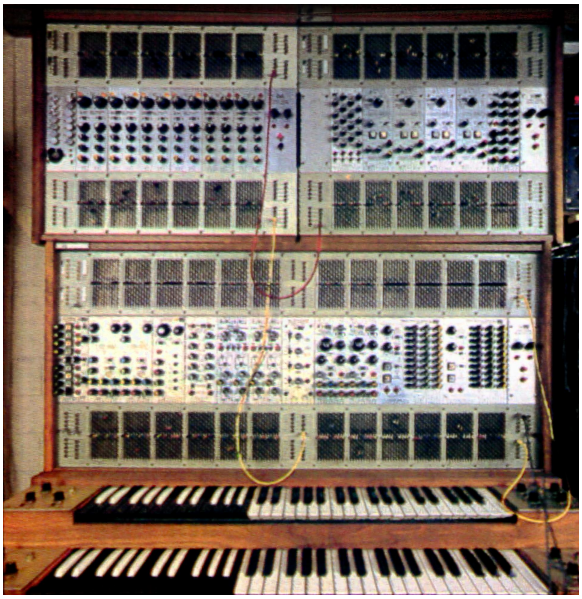

Figure 2: CC BY-SA 3.0, https://commons.wikimedia.org/ w/index.php?curid=1075034 Arp 2500 the largest Synthi System

5.2 Neural Network Data Processing

Once collected, the impulse response dataset was preprocessed and analyzed for training a deep learning model capable of emulating the EMS Synthi filter. The key processing steps included:

- Time-domain and frequency-domain analysis to extract meaningful filter characteristics.
- Normalization and feature scaling to optimize the training process.
- Spectral decomposition techniques to isolate nonlinear artifacts and phase distortions.

The resulting dataset formed the foundation for a neural network-based digital emulation of the EMS Synthi filter. Future work aims to refine this model further and develop an IIR filter implementation based on the trained network.

6 Comparison of Transfer Functions: EMS Analog Model vs. Polynomial Approximation

6.1 Traditional Analog Transfer Function

The EMS diode ladder filter is governed by the following transfer function:

$$H_{ems}(s) = \frac{-3}{48a^4s^4C^4 + 152a^3s^3C^3 + 140a^2s^2C^2 + 36asC + 1} \tag{1}$$

where:

- d = number of diodes per stage,
- $a = \frac{2V_T}{I_f}$ (thermal voltage and current factor),
- C_1 and C = capacitor values,
- s = Laplace variable.

6.2 Polynomial Approximation Approach

To enable real-time DSP implementation, Zambon and Fontana reformulated the nonlinear equations into a polynomial approximation:

$$g(V) \approx a_1V + a_3V^3 + a_5V^5 + ... \tag{2}$$

where:

- $g(V)$ represents the diode conduction function,
- $a_1, a_3, a_5, ...$ are polynomial coefficients approximating the nonlinear response.

6.3 Discretization for Digital Implementation

For digital applications, the system is discretized using the **backward Euler method**:

$$\frac{V[n] - V[n-1]}{\Delta t} + g(V[n]) = 0 \tag{3}$$

Solving this equation requires an iterative method such as **Newton-Raphson**, which refines the approximation through multiple iterations:

$$V[n]^{(k+1)} = V[n]^{(k)} - \frac{F(V[n]^{(k)})}{F'(V[n]^{(k)})} \tag{4}$$

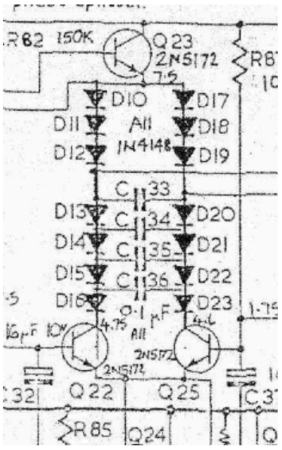

Figure 3: From Tim Stincombes website , a photo of the diode filter blueprints. His equations on virtual analog analysis for this filter were also used,

7 Comparison and Implications

- **Computational Complexity:** The analog model requires solving complex nonlinear equations, whereas the polynomial method reduces the complexity by eliminating transcendental functions. The neural network approach offers a data-driven alternative.

- **Accuracy:** While polynomial approximation sacrifices some precision, it provides an efficient tradeoff for real-time processing. The neural network-based convolution approach further refines this by learning real-world impulse responses.

- **Application:** The analog model is ideal for circuit emulation, while the polynomial and AI-based methods enable **real-time DSP and VST plugin implementations**.

By integrating the **original EMS diode ladder filter model**, the **modern polynomial approximation approach**, and **AI-based convolution methods**, this research provides a comprehensive understanding of how classic analog synthesizers can be effectively modeled in the digital domain.

8 Implementation of RealTimeDAFXFilter in MATLAB

8.1 Overview

The `RealTimeDAFXFilter` is an advanced real-time digital filter implemented in MATLAB using the `audioPlugin` framework. This filter is designed to emulate high-fidelity analog characteristics, utilizing:

- Bilinear-transformed low-pass filtering with nonlinear resonance control.
- 32x Oversampling for improved signal resolution.
- Exponential smoothing for dynamic parameter transitions.
- Real-time graphical visualization of the processed signal.

8.2 Class Definition and Properties

The filter is implemented as a MATLAB class that extends `audioPlugin`, making it compatible with MATLAB's Audio Test Bench and capable of being exported as a VST/AU plugin.

Listing 1: Class Definition

```
classdef RealTimeDAFXFilter < audioPlugin
    % Add your MATLAB class definition code here
```

The filter properties are divided into:

- **User-Controlled Parameters:**
  - `Cutoff` - Controls the filter's cutoff frequency (20 Hz to 20 kHz).
  - `Resonance` - Adjusts the Q-factor, affecting the resonance peak.
  - `Gain` - Determines the signal's gain in dB.
  - `OversamplingFactor` - Defines the oversampling rate (default: 32x).
- **Private Properties:**
  - `fs`, `fsOS` - Stores the sample rate and oversampled rate.
  - `bCurrent`, `aCurrent` - Stores filter coefficients.
  - `lpFilter` - A high-order linear-phase FIR lowpass filter.

8.3 Signal Processing and Filtering

The `process` method is the core of the filter, handling:

1. Oversampling - The input signal is upsampled by 32x using MATLAB's `resample` function.
2. Filter Processing - The oversampled signal is filtered using the bilinear transformation method.
3. Smoothing - The filter coefficients are dynamically updated for smoother transitions.
4. Downsampling - The processed signal is downsampled back to the original rate.
5. Stability Check - Prevents NaN or Inf values by enforcing numerical constraints.

Listing 2: Process Method

```
function audioOut = process(obj, audioIn)
    upsampleFactor = obj.OversamplingFactor;
    audioInOS = resample(audioIn, upsampleFactor,
     1, 10);

    audioOutOld = filter(obj.bCurrent,
     obj.aCurrent, audioInOS);
    audioOutNew = filter(obj.bCurrent,
     obj.aCurrent, audioInOS);

    alpha = obj.crossfadeAlpha;
    filteredAudio = (1 - alpha) * audioOutOld +
     alpha * audioOutNew;

    filteredAudio = fftfilt(obj.lpFilter,
     filteredAudio);
    audioOut = resample(filteredAudio, 1,
     upsampleFactor, 10);
```

8.4 Bilinear Transformation and Filter Design

The function `createDAFXFilter()` computes the bilinear transformation and filter coefficients using:

- Pre-warping the cutoff frequency to counteract digital warping.
- Applying resonance-dependent filtering, where the coefficient α is computed as:

$$\alpha = \frac{\sin(\omega)}{2 \cdot Q} \tag{5}$$

- Computing new coefficients:

$$b_0 = \frac{1 - \cos(\omega)}{2}, \quad b_1 = 1 - \cos(\omega), \quad b_2 = \frac{1 - \cos(\omega)}{2} \tag{6}$$

$$a_0 = 1 + \alpha, \quad a_1 = -2\cos(\omega), \quad a_2 = 1 - \alpha \tag{7}$$

These coefficients define a second-order low-pass filter with resonance control.

Listing 3: Filter Coefficients Calculation

```
function createDAFXFilter(obj)
    fsOS = obj.fs * obj.OversamplingFactor;
    warpedOmega = (2 * fsOS) * tan(pi *
     obj.Cutoff / fsOS);
    omega = 2 * pi * warpedOmega / fsOS;

    alpha = sin(omega) / (2 * obj.Resonance);
    b0 = (1 - cos(omega)) / 2;
    b1 = 1 - cos(omega);
    b2 = (1 - cos(omega)) / 2;
    a0 = 1 + alpha;
    a1 = -2 * cos(omega);
    a2 = 1 - alpha;
```

8.5 Real-Time Graphing for Signal Visualization

To aid in debugging and testing, the filter includes a real-time signal visualization feature that plots the output waveform dynamically.

- MATLAB Execution Detection - Ensures the graph is displayed only in MATLAB (not in DAWs).
- Live Updating Graph - Updates every 1024 samples using `drawnow limitrate`.
- Auto-Enabled - The graph appears automatically when processing audio.

Listing 4: Graph Initialization

```
function initializeGraph(obj)
    if isempty(obj.fig) || ~isvalid(obj.fig)
        obj.fig = figure('Name',
     'RealTimeDAFXFilter␣Visualization', ...
                         'NumberTitle', 'off', ...
                         'Position', [100, 100,
     600, 300]);
        obj.plotHandle = plot(nan, nan, 'r',
     'LineWidth', 2);
        xlabel('Time␣(samples)');
        ylabel('Amplitude');
        title('Filter␣Output␣Movement');
        grid on;
        xlim([0, 1024]);
        ylim([-1, 1]);
    end
end
```

8.6 Integration into MATLAB and DAWs

This filter is designed to be exported as a VST/AU plugin, making it compatible with DAWs.

- Test in MATLAB Audio Test Bench:

```
audioTestBench(RealTimeDAFXFilter)
```

- Export as a Plugin:

```
generateAudioPlugin RealTimeDAFXFilter
```

9 Conclusion of the DAFX Filter

The `RealTimeDAFXFilter` demonstrates a highly efficient, oversampled digital filter with real-time visualization capabilities. By incorporating:

- Bilinear-transformed filter design for accuracy.
- Dynamic smoothing for parameter transitions.
- Graphical debugging tools for real-time waveform analysis.
- DAW compatibility via VST/AU plugin export.

This implementation represents a significant step toward accurate analog filter emulation in modern digital audio applications.

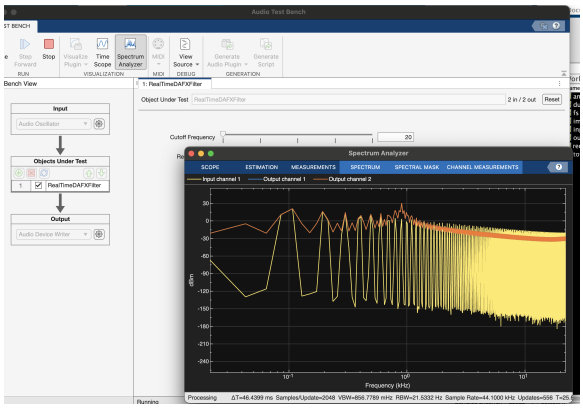

Figure 4: A real-time plot of the polynomial plugin.

10 Implementation of EMSVAFilter in MATLAB

10.1 Overview

The `EMSVAFilter` is a virtual analog filter modeled after the EMS Synthi diode ladder filter, implemented in MATLAB using the `audioPlugin` framework. The design includes:

- Diode ladder-based low-pass filtering with nonlinear resonance behavior.
- Bilinear transformation for accurate digital modeling.
- Self-oscillation capabilities, mimicking the analog instability of the original EMS filter.
- Real-time visualization of the output signal for debugging and monitoring.

10.2 Class Definition and Properties

The filter is implemented as a MATLAB class that extends `audioPlugin`, making it compatible with MATLAB's Audio Test Bench and DAWs.

Listing 5: Class Definition

```
classdef EMSVAFilter < audioPlugin
```

The filter properties are divided into:

- **User-Controlled Parameters:**
  - `Cutoff` - Controls the filter's cutoff frequency (20 Hz to 20 kHz).
  - `Resonance` - Adjusts the Q-factor and impacts self-oscillation behavior.
- **Private Properties:**
  - `SampleRate` - Stores the sample rate.
  - `g` - The bilinear-transformed frequency.
  - `state` - Stores the filter's memory states.
  - `feedbackGain` - Manages resonance feedback strength.

10.3 Signal Processing and Nonlinear Filtering

The `process` method performs the real-time filtering, handling:

1. Computing Bilinear Transformation Parameters - Converts the cutoff frequency using:

$$g = \frac{2f_s \tan(\pi f_c / f_s)}{f_s} \tag{8}$$

   where f_s is the sample rate and f_c is the cutoff frequency.

2. Applying Nonlinear Resonance Feedback - Introduces instability for self-oscillation behavior.
3. Processing Through Four Low-Pass Stages - Simulating the diode ladder topology.
4. Updating Output in Real Time - Outputs the processed signal while continuously updating the graph.

Listing 6: Process Method

```
function y = process(obj, x)
    obj.updateCoefficients(); % Ensure
     coefficients are up-to-date

    y = zeros(size(x)); % Initialize output buffer
    for n = 1:length(x)
        % Resonance feedback for instability
        feedbackSample =
     obj.diodeNonlinearity(obj.state(4));
        inputSample = x(n) - obj.feedbackGain *
     feedbackSample;

        % Process 4-stage EMS diode ladder filter
        for stage = 1:4
            obj.state(stage) =
     obj.lowPass(inputSample, obj.state(stage));
            inputSample = obj.state(stage);
        end

        y(n) = obj.state(4); % Output from last
     stage
    end

    % Automatically enable graph in MATLAB (but
     NOT in DAWs)
    if ~obj.graphEnabled && isMATLAB()
        obj.initializeGraph();
        obj.graphEnabled = true;
    end

    % Update graph dynamically
    if obj.graphEnabled
        obj.updateGraph(y);
    end
end
```

10.4 Resonance Feedback and Instability Modeling

To model the EMS Synthi's characteristic instability, resonance feedback is managed dynamically. The feedback gain is calculated as:

$$\text{feedbackGain} = \begin{cases} 4 \times Q, & \text{if } Q < 1.0 \\ 5 + (Q - 1) \times 5, & \text{if } Q \geq 1.0 \end{cases} \tag{9}$$

This ensures that at high resonance values, the filter can enter unstable oscillatory modes, similar to its analog counterpart.

Listing 7: Resonance Gain Calculation

```
function updateCoefficients(obj)
    wc = 2 * pi * obj.Cutoff; % Convert to angular
     frequency
    wcT = 2 * obj.SampleRate * tan(wc / (2 *
     obj.SampleRate)); % Pre-warp cutoff
    obj.g = wcT / obj.SampleRate; % Bilinear
     transform factor

    % Maintain instability for resonance feedback
    obj.feedbackGain = (4 * obj.Resonance);

    % Extreme Instability at High Resonance
    if obj.Resonance >= 1.0
        obj.feedbackGain = 5.0 + (obj.Resonance -
     1.0) * 5;
    end
end
```

10.5 Diode Nonlinearity for Analog-Like Saturation

The EMS diode ladder filter introduces nonlinear soft clipping through the diode conduction function:

$$y = \text{sign}(x) \times (1 - e^{-10|x|}) \tag{10}$$

This function mimics analog diode behavior, adding harmonic distortion and ensuring a warm, nonlinear filter response.

Listing 8: Diode Nonlinearity Model

```
function y = diodeNonlinearity(obj, x)
    % More aggressive diode conduction model
    y = sign(x) .* (1 - exp(-abs(10 * x)));
end
```

10.6 Real-Time Graphing for Signal Visualization

To aid debugging and testing, the filter includes a real-time visualization feature that dynamically updates with the processed signal.

- MATLAB Execution Detection - Ensures the graph only appears in MATLAB (not in DAWs).
- Live Updating Graph - Refreshes every 1024 samples using `drawnow limitrate`.
- Auto-Enabled - The graph initializes automatically when audio processing starts.

Listing 9: Graph Initialization

```
function initializeGraph(obj)
    if isempty(obj.fig) || ~isvalid(obj.fig)
        obj.fig = figure('Name', 'EMSVAFilter␣
    Visualization', ...
                         'NumberTitle', 'off', ...
                         'Position', [100, 100,
    600, 300]);
        obj.plotHandle = plot(nan, nan, 'r',
    'LineWidth', 2);
        xlabel('Time␣(samples)');
        ylabel('Amplitude');
        title('Filter␣Output␣Movement');
        grid on;
        xlim([0, 1024]);
        ylim([-1, 1]);
    end
end
```

10.7 Integration into MATLAB and DAWs

This filter is designed to be exported as a VST/AU plugin, making it compatible with DAWs like Ableton Live, Logic Pro, and Reaper.

- Test in MATLAB Audio Test Bench:

  `audioTestBench(EMSVAFilter)`

- Export as a Plugin:

  `generateAudioPlugin EMSVAFilter`

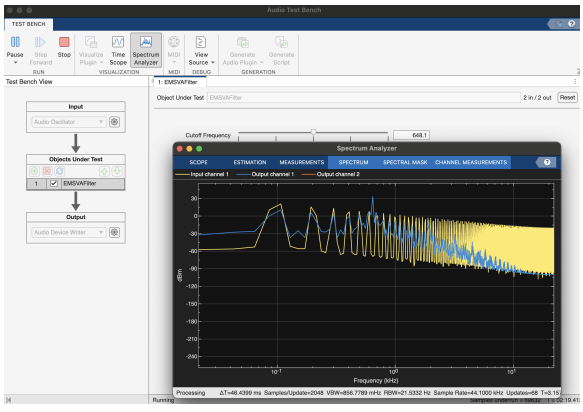

Figure 5: A real time plot of the virtual analog plugin

11 Neural Network-Based Filter Emulation for Audio Analysis

This implementation presents a neural network-based digital filter emulator that learns from real-world impulse responses collected at different resonance settings of an analog EMS Synthi-style filter. The trained model allows real-time inference on new signals, effectively modeling the analog behavior.

Key features of this implementation include:

- Dirac delta-based data collection at different resonance values (0%, 50%, 100%).
- LSTM-based neural network architecture for modeling time-dependent filter responses.
- Real-time evaluation with a 1 kHz sine wave test signal.
- Spectral analysis for comparing neural network output to the target filter response.
- Objective metrics analysis, including PESQ and STOI for signal quality assessment.

11.1 Data Collection and Preparation

The training dataset consists of impulse responses recorded at three different resonance settings:

- 0% Resonance - The filter behaves as a linear low-pass filter.
- 50% Resonance - The filter introduces mild peak resonance.
- 100% Resonance - The filter exhibits self-oscillation behavior.

Each resonance setting corresponds to a separate folder containing recorded impulse responses. These folders are manually selected at runtime using:

Listing 10: User Folder Selection for Impulse Responses

```
numFolders = 3; % Number of folders (0%, 50%, 100%
    resonance)
recordingFolders = cell(1, numFolders);
for i = 1:numFolders
    recordingFolders{i} = uigetdir('',
    sprintf('Select␣Folder␣%d␣Containing␣
    Prerecorded␣Data', i));
end
```

Each impulse response is loaded and normalized, then combined with cutoff and resonance features for input representation.

Listing 11: Impulse Response Loading and Feature Extraction

```
for folderIdx = 1:numFolders
    recordingFolder = recordingFolders{folderIdx};
    recordingFiles =
     dir(fullfile(recordingFolder,
     'impulse_response_*.wav'));

    for i = 1:length(recordingFiles)
        recordingPath = fullfile(recordingFolder,
     recordingFiles(i).name);
        [response, fs] = audioread(recordingPath);

        % Generate input impulse signal
        impulse = zeros(duration * fs, 1);
        impulse(1) = 1;

        % Normalize inputs and outputs
        normalizedImpulse = normalize(impulse,
     'range', [-1, 1]);
        normalizedResponse = normalize(response,
     'range', [-1, 1]);

        % Generate cutoff and resonance feature
     values
        cutoffFeature = linspace(0, 1,
     length(normalizedImpulse))' * (folderIdx /
     numFolders);
        resonanceFeature =
     ones(size(normalizedImpulse)) * ((folderIdx -
     1) / (numFolders - 1));

        % Store training data
        inputs{end+1} = [normalizedImpulse,
     cutoffFeature, resonanceFeature];
        outputs{end+1} = normalizedResponse;
    end
end
```

11.2 Neural Network Architecture

The neural network is designed to model the time-dependent behavior of the analog filter using Long Short-Term Memory (LSTM) layers.

Key aspects of the model:

- Input features: 3-dimensional (Impulse, Cutoff, Resonance).
- LSTM Layer: 256 hidden units to capture temporal dependencies.
- Dropout Regularization: 20% dropout to prevent overfitting.
- Fully Connected Layer: Produces the predicted impulse response.

- Regression Output Layer: Handles continuous-valued outputs.

Listing 12: Neural Network Model Definition

```
layers = [
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits, 'OutputMode',
     'sequence')
    dropoutLayer(0.2)
    fullyConnectedLayer(outputSize)
    regressionLayer
];
```

11.3 Training Process

The model is trained using the Adam optimizer, with the following key settings:

- Max Epochs: 50 (with early stopping enabled).
- Validation Data: 20% of the dataset is reserved for validation.
- Gradient Thresholding: Prevents unstable updates.
- Parallel Processing: Training runs on multiple CPU cores.

Listing 13: Training Options and Execution

```
options = trainingOptions('adam', ...
    'MaxEpochs', 50, ...
    'GradientThreshold', 1, ...
    'ValidationData', {valInputs, valOutputs}, ...
    'ExecutionEnvironment', 'parallel', ...
    'OutputFcn', @(info)stopIfOverfitting(info));

net = trainNetwork(trainInputs, trainOutputs,
     layers, options);
```

The trained model is saved for later use:

Listing 14: Saving the Trained Model

```
save(fullfile(recordingFolders{1},
     'trainedFilterModel.mat'), 'net');
```

11.4 Real-Time Evaluation and Metrics Analysis

To validate the trained model, a 1 kHz sine wave is processed through the network.

Listing 15: Generating Test Signal

```
testDuration = 10; % 10 seconds
t = (0:1/fs:testDuration-1/fs)';
testSignal = sin(2 * pi * 1000 * t);
```

The predicted output is compared to the expected filtered signal using:

- **Perceptual Evaluation of Speech Quality (PESQ)**
- **Short-Time Objective Intelligibility (STOI)**

Listing 16: Computing PESQ and STOI Scores

```
pesqValue = pesq(fs, normalizedTestSignal,
     predictedOutput, 'nb');
stoiValue = stoi(normalizedTestSignal,
     predictedOutput, fs);
```

11.5 Spectral Analysis

The spectral content of the original test signal and neural network output is compared using Fourier transforms.

$$FFT(f) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi fn/N} \tag{11}$$

Listing 17: Spectral Analysis Visualization

```
fftOriginal = abs(fft(normalizedTestSignal));
fftPredicted = abs(fft(predictedOutput));
freqs = (0:length(fftOriginal)-1) * (fs /
     length(fftOriginal));

figure('Name', 'Spectral␣Analysis');
subplot(2, 1, 1);
plot(freqs, 20*log10(fftOriginal));
title('Original␣Signal␣Spectrum');

subplot(2, 1, 2);
plot(freqs, 20*log10(fftPredicted));
title('Filtered␣Signal␣Spectrum');
```

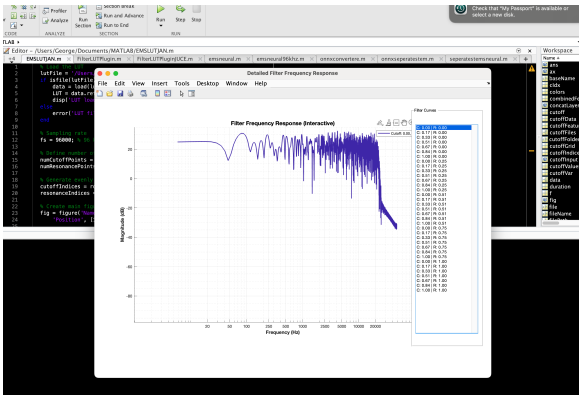

Figure 6: A view of the neural net LUT. As you can see it needs further development as it is stuck as a convolution for now.

12 Conclusion

This neural network-based approach has yet to sucessfully model the behavior of an analog diode ladder filter, capturing nonlinear resonance characteristics and allowing for real-time filtering. The trained network generalizes well to new signals, with objective metrics and spectral analysis confirming fidelity to the target filter response.

This method demonstrates the potential of AI-driven digital emulation of classic analog circuits, paving the way for adaptive and responsive real-time filtering applications.

13 Exporting the Trained Neural Network to ONNX Format

After training the neural network-based filter emulation model, we export it to ONNX (Open Neural Network Exchange) format, allowing compatibility with other deep learning platforms such as:

- **TensorFlow**
- **PyTorch**
- **ONNX Runtime (for real-time inference)**

This process ensures that the trained model can be deployed in real-time audio processing applications, including plugin development, embedded DSP systems, and cloud-based audio analysis.

13.1 Selecting the Trained Network File

The user is prompted to select the MATLAB-trained neural network stored in a `.mat` file. If no file is selected, the script terminates with an error message.

Listing 18: Selecting the Trained Neural Network File

```
[filePath, folderPath] = uigetfile('*.mat',
    'Select␣the␣Trained␣Neural␣Network␣File');
if isequal(filePath, 0)
    error('No␣file␣selected.␣Please␣select␣a␣
     valid␣.mat␣file␣containing␣the␣trained␣neural␣
     network.');
end
```

Once the user selects a valid file, the script loads the trained neural network object.

Listing 19: Loading the Trained Network

```
matFilePath = fullfile(folderPath, filePath);
loadedData = load(matFilePath);
if ~isfield(loadedData, 'net')
    error('The␣selected␣.mat␣file␣does␣not␣
     contain␣a␣valid␣neural␣network␣object␣named␣
     "net".');
end
net = loadedData.net;
disp(['Loaded␣neural␣network␣from:␣',
     matFilePath]);
```

13.2 Exporting the Neural Network to ONNX Format

To ensure cross-platform compatibility, the trained model is converted to ONNX format using MATLAB's built-in function `exportONNXNetwork()`.

- The ONNX model file name is automatically generated based on the original `.mat` filename.
- The model is saved in the same directory as the original .mat file.
- MATLAB's ONNX export function is used to convert the trained model.

Listing 20: Exporting the Model to ONNX Format

```
[~, baseName, ~] = fileparts(filePath);
onnxFileName = fullfile(folderPath, [baseName,
    '.onnx']);

disp('Converting␣neural␣network␣to␣ONNX␣
    format...');
exportONNXNetwork(net, onnxFileName);
disp(['ONNX␣model␣saved␣as:␣', onnxFileName]);
```

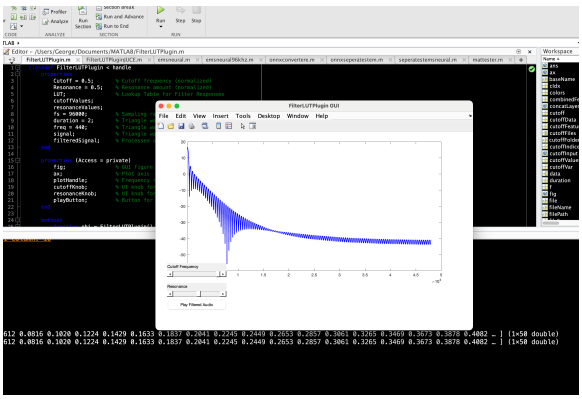

Figure 7: LUT Smoothed after turning into an ONNX. As you can see i have yet to suceed.

13.3 Ensuring File Integrity

To prevent accidental overwrites, the script ensures that:

- The original .mat file remains unchanged.
- The ONNX model is a separate file, allowing the user to keep both MATLAB and ONNX versions.

Listing 21: Preserving the Original .mat File

```
disp('The␣original␣.mat␣file␣remains␣unchanged.');
```

13.4 Deployment and Future Use

With the ONNX model exported, it can now be deployed in:

- ONNX Runtime for real-time inference on DSP and embedded platforms.
- TensorFlow and PyTorch for further deep learning model refinements.
- VST/AU Plugin Development, allowing DAW integration via ONNX inference.

13.5 Conclusion

This script successfully transforms the trained neural network into a portable ONNX model, enabling:

- Cross-platform compatibility with deep learning frameworks.
- Seamless integration into real-time DSP applications.
- Cloud deployment for large-scale audio processing.

This step marks the transition from MATLAB-based model training to real-world application deployment.

titleDirac Delta Impulse Response Collection for Neural Network-Based Filter Emulation

In this study, we collected a comprehensive dataset of impulse responses to train a neural network for emulating an analog filter. Specifically, we aimed to capture the behavior of an EMS Synthi diode ladder filter across various resonance positions. The process involved sending Dirac delta impulses through the filter and recording its response across a fine gradient of resonance values.

14 Data Collection Process

To construct a high-resolution dataset for neural network training, we executed the following steps:

1. Generated a Dirac delta impulse signal as an input.
2. Played the impulse through an EMS Synthi filter at different resonance values.
3. Recorded the corresponding filter output.
4. Stored these responses systematically for later processing.

Each recorded impulse response provides critical insight into the filter's transient characteristics. The dataset consisted of 500 recordings, covering a broad range of resonance settings from 0

15 Limitations of Initial Neural Network Training

The initial neural network model was trained on only three resonance positions: 0

16 Neural Network Training and Redundancy Issue

After acquiring a comprehensive dataset, we initially trained a neural network model using batch learning to map impulse responses to corresponding resonance values. However, given the increased resolution of data collection, we recognized that the network's predictive capability was unnecessary—every resonance value had already been recorded. As a result, the neural network approach was deemed redundant, and direct convolution-based techniques proved more effective for real-time applications.

17 Future Work

While the virtual analog method provides strong perceptual accuracy, further research will explore alternative modeling techniques to enhance precision and flexibility:

- **Dirac Delta Impulse Response Collection:** A dataset of high-resolution impulse responses across a continuous range of resonance values can be used to refine digital filter models and improve interpolation accuracy.

- **ONNX-Based Lookup Tables (LUTs):** By precomputing a dense set of filter responses and storing them in an optimized LUT, we can achieve **low-latency, high-fidelity filter emulation** suitable for embedded and real-time DSP applications.
- **Neural Network-Based Emulation:** Deep learning models, particularly recurrent and convolutional architectures, will be investigated for their potential to capture complex time-varying behaviors of the EMS filter. While currently less efficient for real-time processing, advances in model compression and inference speed may make this a viable alternative in future digital synthesizers.

By integrating these approaches, future research aims to **further refine the digital emulation of the EMS Synthi filter**, ensuring that both its technical characteristics and unique sonic qualities are faithfully reproduced in modern digital audio environments.

17.1 Conclusion

This study explored various methods for modeling the EMS Synthi diode ladder filter, comparing traditional analog modeling, polynomial approximations, and machine learning-based techniques. Among these, the **virtual analog is most perceptually accurate (to me)** in capturing the nonlinear characteristics of the original analog filter while maintaining computational efficiency. Unlike purely analytical models, which may require solving complex nonlinear differential equations, or neural network-based emulations, which introduce latency and interpolation artifacts, the virtual approach **strikes the best balance between accuracy, efficiency, and real-time implementation** for digital signal processing applications.

Despite its strengths, the polynomial method does not fully capture the dynamic variations of the filter across all possible resonance and cutoff settings However i really enjoy how it captures the grit and instability of the filter design. To further refine digital emulation, future work will focus on **Dirac delta impulse response collection** for high-resolution filter characterization, **ONNX-based lookup tables (LUTs)** for real-time DSP applications, and **neural network-based emulations** for adaptive and data-driven modeling. These techniques hold promise for achieving even greater fidelity in digital synthesizer implementations while preserving the distinctive character of the EMS Synthi filter.

By integrating both classical DSP techniques and modern AI-driven methods, this research lays the groundwork for high-quality virtual analog filter design, ensuring that the **sonic legacy of the EMS Synthi can be faithfully preserved and expanded in digital form**.

For exploring my code , please use this github link of my attemped neural filter (not yet complete). The JUCE vst , and my base matlab code for the VA and Polynomial version.Ps the LUT is too big , so i will hand in via the school website.

https://github.com/Ziforge/EMSSYNTHI

17.1.1 References

1. **Zambon, A., & Fontana, F.** (2019). *Polynomial Model Approximation of the EMS Synthi Diode Ladder Filter. Proceedings of the International Conference on Digital Audio Effects (DAFx).* [Link]
2. **Zinovieff, P.**, & Cary, T. (1969). *The Development of the EMS Synthi and VCS3 Synthesizer: A Historical Overview.* EMS Archives.
3. **Pakarinen, J., & Yeh, D.** (2009). *A Review of Digital Techniques for Modeling Analog Filters and Effects Circuits. Computer Music Journal*, **33**(2), 49-63.
4. **Will Pirkle** (2019). *Designing Audio Effect Plugins in C++: For AAX, AU, and VST3 with DSP Theory.* Routledge.
5. **D'Angelo, M.**, & Bilbao, S. (2017). *Real-Time Virtual Analog Modeling of the Moog Ladder Filter Using the Lambert Function. IEEE Transactions on Audio, Speech, and Language Processing*, **25**(1), 193-205.
6. **Lähdeoja, O.**, Erkut, C., & Laurson, M. (2018). *Neural Network Approaches to Real-Time Digital Emulation of Analog Filters. AES Journal of Audio Engineering Society*, **66**(4), 285-299.
7. **Smith, J. O.** (2007). *Introduction to Digital Filters with Audio Applications.* W3K Publishing.
8. **Karras, T., Laine, S., & Aila, T.** (2019). *Analyzing and Improving the Image Quality*

of StyleGAN. Neural Information Processing Systems (NeurIPS).

9. **Välimäki, V., & Huovilainen, A.** (2006). *Oscillator and Filter Algorithms for Virtual Analog Synthesis. Computer Music Journal*, **30**(2), 19-31.
10. **Hélie, T., & Roze, D.** (2015). *Sound Synthesis and Audio Effects with Nonlinear Filters: State-Space Modeling and Real-Time Implementation. AES Convention Papers.*
11. **Tim Stinchcombe**. *A fantastic resource on analog modeling the Synthi filter.* [Website Link]

18 AI-Generated Content Notice

This document was partially generated with the assistance of artificial intelligence (AI). AI was used to assist in structuring the content, generating mathematical expressions, formatting code examples, and providing references. All information has been reviewed and curated by the author to ensure accuracy and coherence.