Modelling Zero-Delay Feedback Transistor Ladder Filter using OTA Abstraction

Peter Corbett

January 2024

1 Introduction

The transistor ladder filter is a Voltage-Controlled Filter structure developed by Bob Moog, and popularized in iconic synths such as the MiniMoog Model D. Capturing the non-linearities in a DSP model can be challenging, as the equations used in this model are all implicit. As a result of the local feedback in each stage, each of the four equations contains an output term that is nested in a hyperbolic tangent function, and cannot be solved for algebraically. One approach to mitigate this would be to add a single sample delay (unit delay) in the feedback path, allowing for the use of the previous output samples to be used in the current calculations. However, with filters, the consequence of this unit delay is a skewing of the frequency response. Another approach, and the one used in this model, is to use numeric methods to solve for the output without adding a unit delay (coined "Zero-delay feedback")[4].

In this model, the output of the previous equation is required for the next equation as the stages are cascaded. Adding to the complexity, the output of the final equation is required as an input of the first equation, owing to the global resonance feedback loop. One way to address this without adding a unit delay is to solve for each stage sequentially and then iteratively solve for the entire system. However, using nested solvers is very computationally inefficient, yielding $O(n^2)$. Instead, a better approach is to solve all of the stages simultaneously using a multi-dimensional solver[9].

2 Circuit Analysis

My intention for this paper is not to explore the details of the circuit analysis of the Transistor Ladder Filter (for that, I recommend exploring these sources:[1][6][5]). Instead, this analysis will start from the basis of Urs Heckmann's blog articles[2][3], and resume from where his work left off. For completion, I have included a simplified schematic of the original transistor ladder filter, so that the reader can get a feel for the underlying structure (Figure 1).

In his article *One Pole Unlimited*[3], Heckmann suggests that starting from a one-pole structure is ideal, as one can either leave the system linear, or impose different nonlinearities to model the behaviour of several real-world circuits. Figure 2 shows such a structure that makes use of a component called an Operational Transconductance Amplifier (OTA), and clarifies that it is useful to abstract this to a generalized Voltage Controlled Current Source (VCCS), as only one of the two non-linearities discussed is representative of the behavior of an OTA. For simplicity, I have kept the

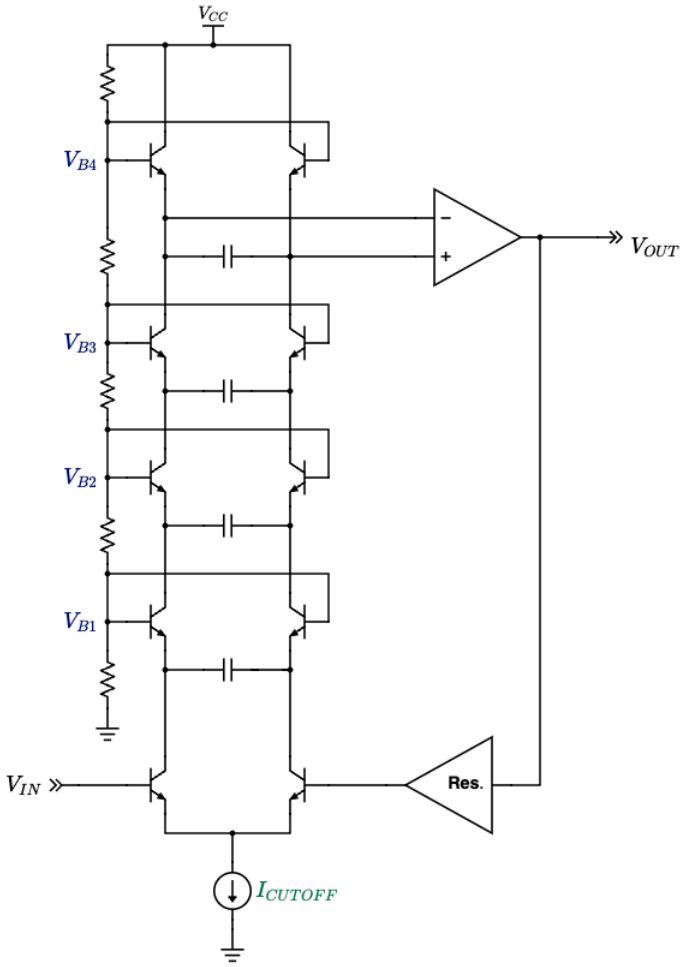


Figure 1: Transistor Ladder Filter (simplified)

circuit analysis in terms of an OTA device, with the implicit understanding that the most appropriate non-linearity would be selected for a given application.

The current output of the OTA block is a function of the differential voltage input and the control voltage g, and can be characterized with Equation 1. This allows the reuse of the same fundamental structure to model both a Moog Transistor Ladder and a Roland Juno 6/60/106-style OTA filter. A further advantage is that g can be used to tune the cutoff frequency of the filter without directly modelling the capacitance, making this model much more controllable relative to direct component-level modelling approaches.

$$\begin{aligned} I_{LINEAR} &= g(V^+ - V^-) \\ I_{OTA} &= g(tanh(V^+ - V^-)) \\ I_{LADDER} &= g(tanh(V^+) - tanh(V^-)) \end{aligned} \tag{1}$$

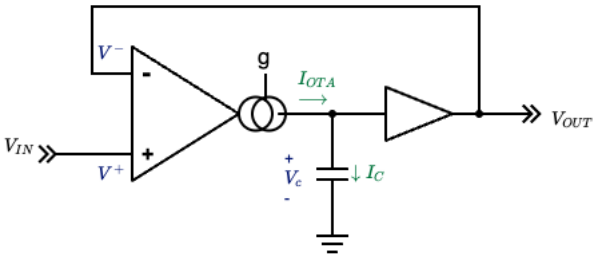


Figure 2: One-pole Lowpass Filter

Through the use of nodal-analysis, we can observe that the current leaving the OTA must equal the current entering the capacitor Equation 2. We can make this assertion only if we consider the input to the buffer and differential voltage inputs of the OTA to be high impedance (or in the ideal case, ∞), after which we can conclude that no current will input the buffer - it will merely read the the capacitor voltage (V_C) at this node. It is worth noting that this is likely where this OTA-structure deviates most significantly from the original transistor ladder structure. Depending on the details of the Bipolar Junction Transistor (BJT) model, one could consider the base current, leakage current, and various parasitic capacitances that we will happily ignore.

$$\begin{aligned} I_{OTA} &= I_C \\ I_{OTA} - I_C &= 0 \end{aligned} \tag{2}$$

After ascertaining our current equations, we will notice that we do not yet have an equation that describes our capacitor current (I_C). To develop one, it is helpful to first consider the standard capacitor current equation:

$$i_c = C\frac{\mathrm{d}V}{\mathrm{d}t}$$

Keeping in mind that this derivative represents a change in voltage at this node, we could substitute this directly into our current equation. However, both Heckmann and Andrew Simper (Cytomic) propose a more elegant approach, replacing this term with a state variable $s[n]$ that can be updated each discrete time-step, known as a tick[4][8][7]. To accommodate the shift from the continuous-time domain to discrete-time domain, $[n]$ will indicate the current sample, and $[n+1]$ will be used to represent the subsequent sample during the update step. Using this approach, we can arrive at the following equations to describe a one-pole ladder lowpass filter:

$$\begin{aligned} y &= V_{OUT} \\ x &= V_{IN} \end{aligned}$$

$$g = \frac{\pi F_C}{F_s} \tag{3}$$

$$y = g(tanh(x) - tanh(y)) + s[n] \tag{4}$$

$$s[n+1] = 2y[n] - s[n] \tag{5}$$

where we have replaced the input voltage V_{in} with x and the output voltage V_{OUT} with y for simplicity. We have also assigned the variable F_C to represent the filter cutoff frequency, and the variable F_S to represent the sample rate, both of which are used in the evaluation of g.

3 System Equations

By using our one-pole filter model as the basis, we can construct a 4-pole ladder filter by cascading 4 stages together in series, then returning the global output to the global input in an inverting feedback configuration to produce resonance. The amount of resonance is controlled by a gain r, which replaces a potentiometer in a voltage-divider configuration on the original circuit.

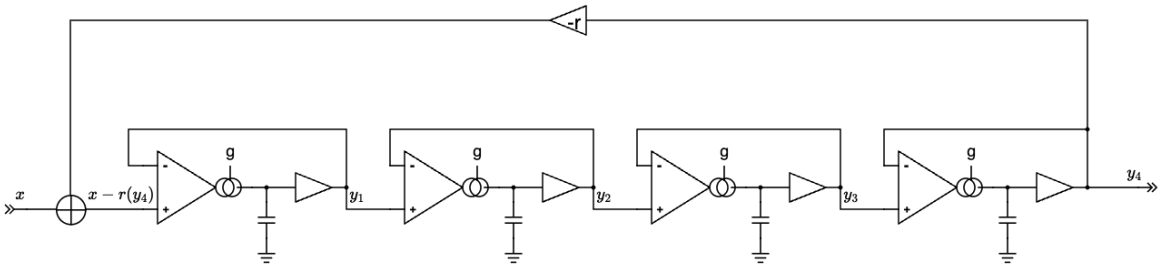


Figure 3: 4-stage Ladder Filter

The following set of equations describes the "OTA-Model" output of 4 cascaded one-pole transistor ladder filter stages with resonance feedback.

$$\begin{aligned}
y_1 &= g(tanh(x - ry_4) - tanh(y_1)) + s_1 \\
y_2 &= g(tanh(y_1) - tanh(y_2)) + s_2 \\
y_3 &= g(tanh(y_2) - tanh(y_3)) + s_3 \\
y_4 &= g(tanh(y_3) - tanh(y_4)) + s_4
\end{aligned} \tag{6}$$

Each equation is implicit, depending on its own output, with this term nested inside of a $tanh()$ function. Each equation also uses the previous stage's output as the second input (or in the case of y_1, the output from the final stage, y_4). One method for solving the system would be to use an iterative method such as Newton-Raphson (NR) or a direct method such as 4th-order Runge-Kutta (RK-4) to solve each equation, and an outer loop to iteratively solve the entire system. This would be computationally expensive (O^2), so an alternative method may be preferable. Fortunately, the multi-dimensional form of Newton-Raphson (Equation 7) can iteratively solve multiple equations in a non-linear system simultaneously.

$$y^{(k+1)} = y^{(k)} - \mathbf{J}^{-1}(y^{(k)}) f_y^{(k)} \tag{7}$$

Here, y and f_y are vectors, and $\mathbf{J}^{-1}(y)$ is the inverse Jacobian matrix of y. The k superscript denotes the iteration step, and should not be confused with the sample notation n. We need to do some preparations to our system described in Equation 6 to bring it to a compatible form. First let's declare our output vector y, which is comprised of the outputs of each stage, and has been left in terms of "y-variables":

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{8}$$

We will declare a term $\Phi(y)$ that holds the "right-hand side" of our system as described in Equation 6, noting that:

$$y = \Phi(y)$$

$$\Phi(y) = \begin{bmatrix} g(tanh(x - ry_4) - tanh(y_1)) + s_1 \\ g(tanh(y_1) - tanh(y_2)) + s_2 \\ g(tanh(y_2) - tanh(y_3)) + s_3 \\ g(tanh(y_3) - tanh(y_4)) + s_4 \end{bmatrix} \tag{9}$$

Next, we will manipulate the expression $y = \Phi(y)$ by subtracting y from both sides. This sets it equal to zero, making it an ideal form for root-finding. We will use a function f as shorthand to contain the expression $\Phi(y) - y$:

$$\Phi(y) - y = 0$$

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \Phi - y$$

$$f = \begin{bmatrix} g(tanh(x - ry_4) - tanh(y_1)) + s_1 - y_1 \\ g(tanh(y_1) - tanh(y_2)) + s_2 - y_2 \\ g(tanh(y_2) - tanh(y_3)) + s_3 - y_3 \\ g(tanh(y_3) - tanh(y_4)) + s_4 - y_4 \end{bmatrix} \tag{10}$$

The final requirement to solve Equation 7 is a Jacobian matrix, which maps the partial derivatives of f and y. The one-dimensional NR method divides the function by its derivative to predict the zero-crossing of a function. By using the output of the previous iteration, a tangent line can be projected to converge on the root. This operation is analogous to the function of the Jacobian matrix in the multi-dimensional form, only that the Jacobian matrix must perform this operation across multiple dimensions. It is first inverted so that we can perform matrix multiplication rather than matrix division. The Jacobian matrix takes the following form:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial y_3} & \frac{\partial f_1}{\partial y_4} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \frac{\partial f_2}{\partial y_3} & \frac{\partial f_2}{\partial y_4} \\ \frac{\partial f_3}{\partial y_1} & \frac{\partial f_3}{\partial y_2} & \frac{\partial f_3}{\partial y_3} & \frac{\partial f_3}{\partial y_4} \\ \frac{\partial f_4}{\partial y_1} & \frac{\partial f_4}{\partial y_2} & \frac{\partial f_4}{\partial y_3} & \frac{\partial f_4}{\partial y_4} \end{bmatrix} \tag{11}$$

To obtain the partial derivatives, we can make use of the tanh() derivative identity and the chain rule. It is worth noting that the state terms (s) should be treated as a constant and will differentiate to zero, in addition to the following hyperbolic tangent derivation needed to construct the Jacobian matrix:

$$\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - tanh^2(x)$$

$$\mathbf{J} = \begin{bmatrix} -g(1-tanh^2(y_1))-1 & 0 & 0 & -gr(1-tanh^2(x-ry_4)) \\ g(1-tanh^2(y_1)) & -g(1-tanh^2(y_2))-1 & 0 & 0 \\ 0 & g(1-tanh^2(y_2)) & -g(1-tanh^2(y_3)-1 & 0 \\ 0 & 0 & g(1-tanh^2(y_3)) & -g(1-tanh^2(y_4))-1 \end{bmatrix} \tag{12}$$

While we have currently derived all the expressions necessary to solve the system, we have yet to define our stopping criteria for the Newton-Raphson loop. At the start of each sample, we must make an estimate for each term in the y vector. Although there are superior methods of making an initial guess, I have used the previous sample's output for simplicity, and found that this usually solves within four iterations. We can use the residue to determine when to stop looping by requiring that it fall below a threshold that is sufficiently close to zero. Since we only care about the accuracy of our final output, we only need to consider the residue of y_4 for our error calculations.

$$\begin{aligned} y_{est}[n] &= y[n-1] \\ residue &= y_4 - y_{4,EST} \\ |residue| &< \text{error threshold} \end{aligned} \tag{13}$$

We are now ready to solve Equation 7, however, we should consider the fact that performing a matrix inversion is computationally expensive, and not recommended for real-time use. Instead, we can use the following modified NR expression:

$$\delta y^{(k)} = y^{(k+1)} - y^{(k)}$$

$$\mathbf{J}(y^{(k)})\delta y^{(k)} = -f_y^{(k)} \tag{14}$$

We have introduced a new term $\delta y^{(k)}$ that allows us to put this into the familiar matrix form $\mathbf{A}x = b$. We can now solve for $\delta y^{(k)}$ using traditional methods such as Gaussian elimination. Subsequently, we can obtain our solution for this iterative step by rearranging the previous $\delta y^{(k)}$ definition:

$$y^{(k+1)} = \delta y^{(k)} + y^{(k)} \tag{15}$$

In the spirit of further optimization, we should consider our use of tanh() terms in a real-time application (especially as they are compounded by the number of NR iterations). For each iteration of the NR loop, we can further optimize by pre-computing the results of the tanh() terms, as they will be used in both the functions and the derivative functions within the Jacobian matrix, and across multiple matrix cells. There is also some algebra that can be shared throughout multiple processes. This is likely worthwhile, as even if we arrive at the solution within four iterations of Newton's Method, this still implies that the loop iterations are being calculated at four times the sample rate.

Many programmers often gravitate towards using an approximation of tanh() or to seek further optimization gains through the use of lookup-tables (LUT). I have refrained from employing such methods for this application as the output is not bounded within ± 1, and I was concerned of a loss of accuracy or an exponential explosion.

4 Results

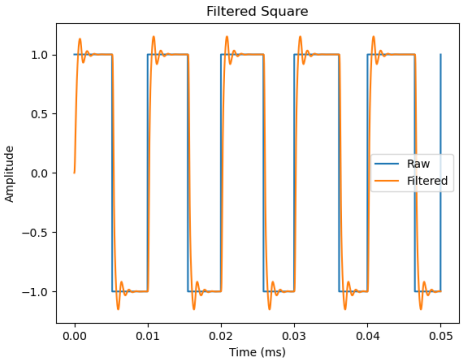
To test the model, I have written a Python script that makes use of the vector and matrix operations in the NumPy library. By feeding several basic signals through the filter model, we can get a sense of how it performs under various operating parameters. Square waves allow us to clearly see the rise and fall time of the filter, as well as overshoot, as increasing the resonance parameter will push the filter into under-damped territory, and eventually self-oscillation. Square waves deconstruct into only odd-order harmonics, whereas, saw-tooth waves produce both even and odd order harmonics. Both are harmonically rich, and an ideal candidate for harmonic analysis (using FFT's) to determine the spectral response of the filter.

The Python code (JupyterLab) and support material is available on my GitHub (`https://github.com/pscorbett/LadderFilter`). This should serve as a suitable template or reference for related filter circuit modelling.

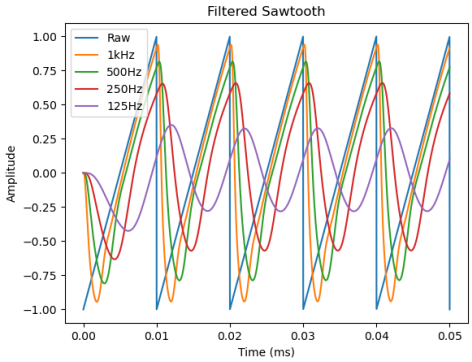
5 Addendum: Non-linear Positive Feedback

As the overarching goal of this project was to formalize a practical structure for modelling nonlinear ladder filters, one glaring omission has been the inclusion of the MiniMoog "Feedback Loop Trick". This was achieved on a Model D synthesizer by patching the headphone output back into the external audio input (pre-filter), with the amount of positive feedback being controlled using the external input's level potentiometer. Depending on the gain-staging, this can overdrive both the external input amplifier as well as the ladder filter. Since this is a crucial facet of the sound of a MiniMoog synthesizer (particularly with bass tones), it is a natural candidate for model extension with only minor adaptations to the core design.

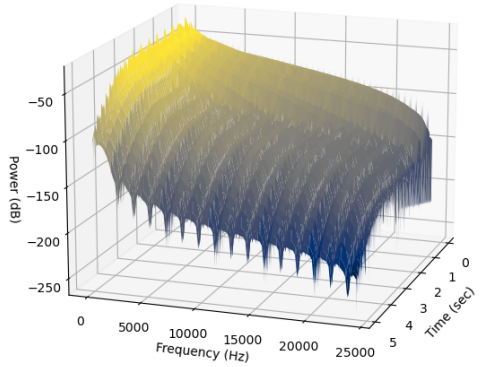
Figure 5 illustrates the combined schematics of the headphone output amplifier and the external (EXT) input amplifier. This circuit was simulated in LTSpice to observe the nature of the distortion of sinusoidal signals across various input levels (Figure 6). From "probing" various nodes in the circuit, I discovered that the output of the headphone amplifier showed no observable distortion with any reasonable input level, however, the EXT input amplifier could be heavily distorted, as it was driven by the headphone amplifier.



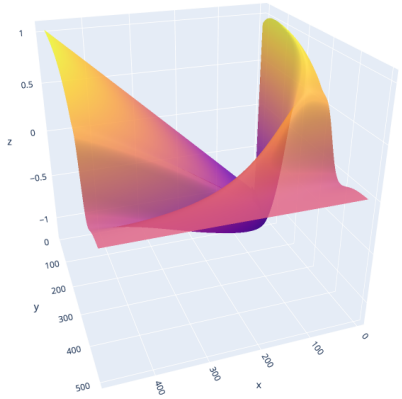
(a) Square + Resonance



(b) Saw Sweep: Oscilloscope



(c) Saw Sweep: 3D Spectrogram



(d) Saw Sweep: 3D Time Domain

Figure 4: Filter response of 100Hz basic wave-shapes

One approach to modelling the distortion observed in the SPICE simulation would be to export a data array representing a transfer function, which will map an input value to an output value. This is also commonly referred to as a "waveshaper", and incredibly computationally efficient, even though they require linear-interpolation. However, for this application, we also require the partial derivatives of the non-linearity to fill the Jacobian matrix, which would be somewhat more challenging. Instead, I opted to approximate the transfer function curve using a biased hyperbolic tangent. This function is easily parameterized and differentiated, and does a good job of characterizing the clamping effects of injecting a biased large signal into a BJT amplifier without sufficient headroom.

$$
\begin{aligned}
b &= \text{bias} \\
A_f &= \text{feedback gain} \\
y &= tanh(A_f x - A_f b)
\end{aligned}
\tag{16}
$$

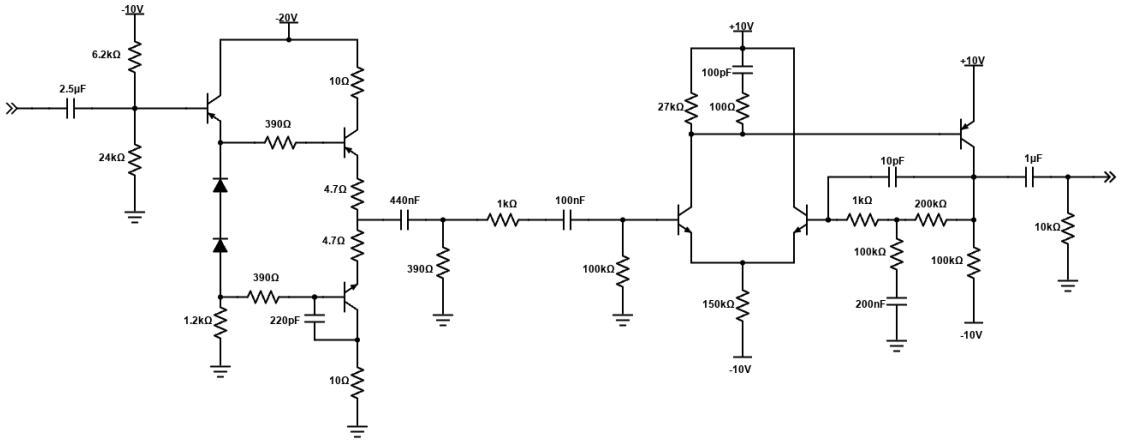


Figure 5: PHONES and EXT Input Amplifier

Here, A_f scales both the input, x, and the bias. This is fantastic for our application, as when $A_f = 0$, the entire argument of the hyperbolic tangent function (and thus the output) also become zero. The expected behavior of the EXT input potentiometer is to mix no positive feedback into the filter input when turned all the way counterclockwise. This makes A_f an ideal parameter for controlling the level of feedback in our model.

After initial testing of the implementation of the saturated feedback into the LPF model, I found that the resulting signal tended to bias with some combinations of resonance and positive feedback. This stands to reason, as the integrated feedback signal is heavily weighted below zero. Referring to Figure 5, we should take note of the $1\mu F$ coupling capacitor at the output of the circuit. This blocks DC to remove the bias on the output amplifier signal, including any weighting from asymmetric waveforms. We can achieve the same result using a highpass filter with a low cutoff frequency (less than 20Hz to not impact audio range). As a point of interest, the effects of these capacitors accounts for the inward slope of the heavily driven waveforms in Figure 6.

For simplicity, I have continued to use the same state-accumulator one-pole filter topography for this highpass filter, although I opted for the "linear" form, as we will be processing our distortion prior to the filter's non-inverting input terminal. Figure 8 shows our revised signal-flow diagram.

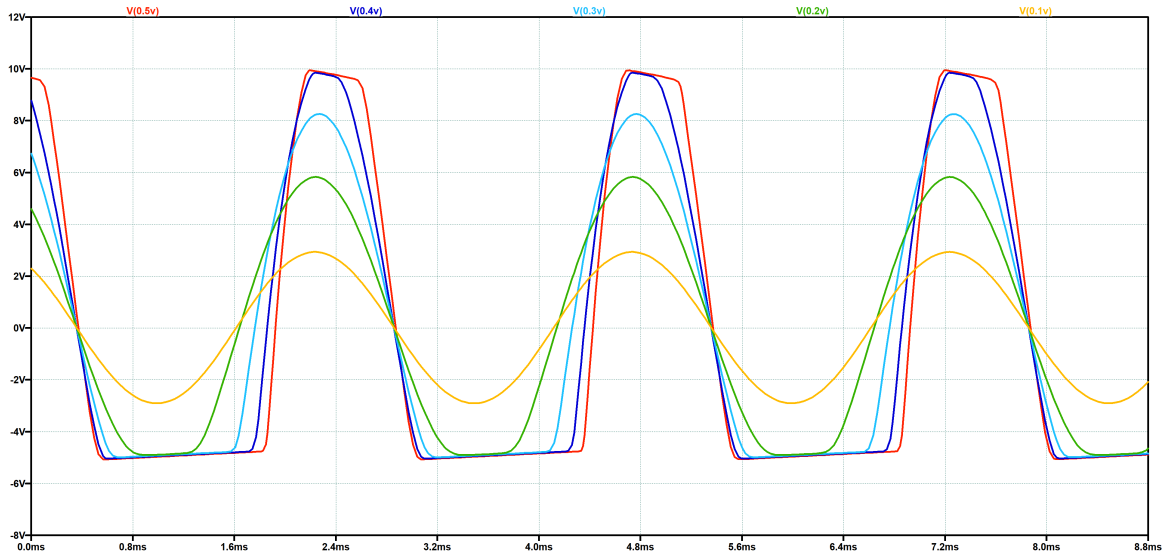


Figure 6: Large signal saturation in EXT input amplifier

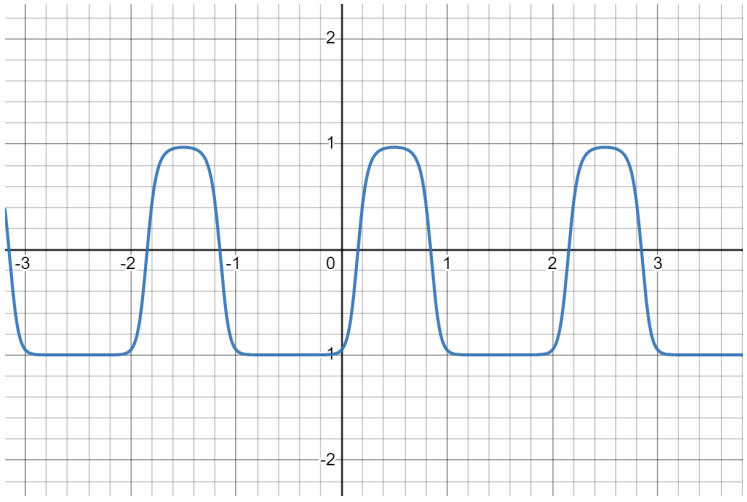


Figure 7: Tanh() approximation of overdriven BJT

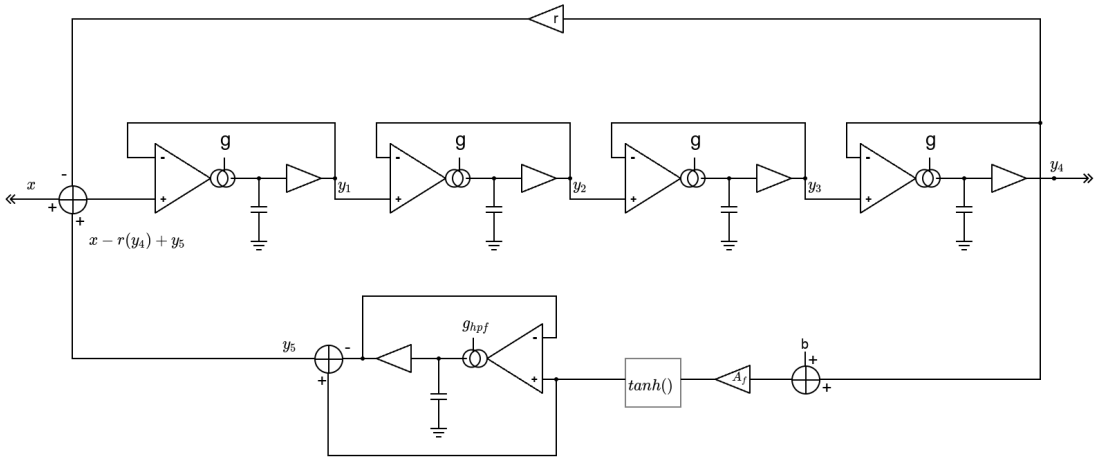


Figure 8: 4-Pole Ladder Filter with Feedback

Recall (from Equation 4)that our one-pole LPF response took the following form:

$$y = g(x - y) + s$$

To transform a lowpass filter into a highpass filter, we must subtract the output of the LPF output from the original signal:

$$y = x - (g(x - y) + s)$$
$$y = x - g(x - y) - s$$

Now we can start to construct our equations. I found it useful to consider an intermediate output y_5 between the output of the non-linearity and the HPF input, although this eventually collapses down to a single function to encapsulate both the non-linearity and the filter.

$$y_5 = tanh(A_f y_4 - A_f b)$$
$$y_6 = y_5 - g_{hpf}(y_5 - y_6) - s_6$$

Since we are using the non-linear form of the one-pole filter, we can solve for y_6 algebraically:

$$
\begin{aligned}
y_6 &= y_5 - g_{hpf}y_5 + g_{hpf}y_6 - s_6 \\
y_6 - g_{hpf}y_6 &= y_5 - g_{hpf}y_5 - s_6 \\
y_6(1 - g_{hpf}) &= y_5 - g_{hpf}y_5 - s_6 \\
y_6 &= \frac{y_5 - g_{hpf}y_5 - s_6}{1 - g_{hpf}} \\
y_6 &= \frac{y_5(1 - g_{hpf}) - s_6}{1 - g_{hpf}} \\
y_6 &= y_5 - \frac{s_6}{1 - g_{hpf}}
\end{aligned}
$$

Now, we can replace y_5 with tanh() expression above. We can also remove any reference to our intermediary node by renaming y_6 to y_5 and renaming s_6 to s_5:

$$y_5 = tanh(A_f y_4 - A_f b) - \frac{s_5}{1 - g_{hpf}}$$

You may notice that our new y_5 expression is not actually implicit. However, for convenience, it is likely easier to extend the the dimensions of our vectors and matrix and throw this into our NR solver than it would be to try and substitute this directly into our y_1 equation. Additional work would be required to determine if the additional computational load would be excessive in a real-time system.

It is also worth noting that the second "s_5" term in y_5 is constant, and will differentiate to zero for all terms of our Jacobian matrix (very convenient!). From here, we can derive our updated f vector and **J** matrix.

Here are the revised equations for f_1 and the first row of the Jacobian:

$$
\begin{aligned}
f_1 &= g(tanh(x - ry_4 + y_5) - tanh(y_1)) + s_1 - y_1 \\
\frac{\partial f_1}{\partial y_1} &= -g(1 - tanh^2(y_1)) - 1 \\
\frac{\partial f_1}{\partial y_4} &= -rg(1 - tanh^2(x - ry_4 + y_5)) \\
\frac{\partial f_1}{\partial y_5} &= g(1 - tanh^2(x - ry_4 + y_5))
\end{aligned}
$$

f_2, f_3, and f_4 will remain the same, as will rows 2-4 of the Jacobian, with the exception of adding a column of zeros to extend the dimension. The equations for the final function f_5, and the associated derivatives are as follows:

$$
\begin{aligned}
f_5 &= tanh(A_f(y_4 - b)) - \frac{s_5}{1 - g_{hpf}} - y_5 \\
\frac{\partial f_5}{\partial y_4} &= A_f(1 - tanh^2(A_f y_4 - A_f b))
\end{aligned}
$$

$$\frac{\partial f_5}{\partial y_5} = -1$$

Putting this all together, we arrive at our final f vector and $\mathbf{J}$ matrix:

$$f = \begin{bmatrix} g(tanh(x - ry_4 + y_5) - tanh(y_1)) + s_1 - y_1 \\ g(tanh(y_1) - tanh(y_2)) + s_2 - y_2 \\ g(tanh(y_2) - tanh(y_3)) + s_3 - y_3 \\ g(tanh(y_3) - tanh(y_4)) + s_4 - y_4 \\ tanh(A_f(y_4 - b)) - \frac{s_5}{1 - g_{hpf}} - y_5 \end{bmatrix} \tag{17}$$

$$\mathbf{J} = \begin{bmatrix} -g(1 - tanh^2(y_1)) - 1 & 0 & 0 & -rg(1 - tanh^2(x - ry_4 + y_5)) & g(1 - tanh^2(x - ry_4 + y_5)) \\ g(1 - tanh^2(y_1)) & -g(1 - tanh^2(y_2)) - 1 & 0 & 0 & 0 \\ 0 & g(1 - tanh^2(y_2)) & -g(1 - tanh^2(y_3) - 1 & 0 & 0 \\ 0 & 0 & g(1 - tanh^2(y_3)) & -g(1 - tanh^2(y_4)) - 1 & 0 \\ 0 & 0 & 0 & A_f(1 - tanh^2(A_f y_4 - A_f b)) & -1 \end{bmatrix} \tag{18}$$

6 Future Improvements

Beyond the ladder filter implementation outlined in this paper, this method could be easily adapted to model a Roland Juno-style OTA filter by expressing both differential inputs (of a given stage) inside a single tanh() term, rather than as a difference of two. Although no "feedback trick" was possible on the original line of Juno synthesizer hardware, this could be kept in the model if desired.

Perhaps the most practical and obvious next step for this model is to port the concepts proved in the Python prototypes by developing a real-time C++ model. This could take several forms, including a JUCE plugin or a Max MSP external.

One other possibility would be to experiment with a "spread" parameter that slightly de-tunes the cutoff frequencies by a static or variable amount. The intention of this would be to capture the effect of component variability across the stages. Most ladders are constructed with a matched transistor pair for each stage, but are much less likely to attempt to match components between stages (or use a package with more than two transistors sharing the same die in the case of transistor array integrated circuits). They are also more susceptible to slight thermal differences as they would physically occupy different space on the PCB. One possibility to implement such a variance model could take the following form:

$$\begin{aligned} g_1 &= g(1 + spread \cdot random) \\ g_2 &= g(1 + spread \cdot random) \\ g_3 &= g(1 + spread \cdot random) \\ g_4 &= g(1 + spread \cdot random) \end{aligned}$$

where spread is a decimal percentage and random is a random number between zero and one. This would likely be a very small number, and additional logarithmic correction may be desired so that spread is expressed in pitch rather than absolute frequency. The numbered g parameters could then replace the original g in all equations, with the exception of the term to set the frequency of the highpass filter in the feedback loop.

References

[1] Sam Gallagher. *Analyzing the Moog Filter - Technical Articles*. URL: https://www.allaboutcircuits.com/technical-articles/analyzing-the-moog-filter/ (visited on 01/13/2024).

[2] Urs Heckmann. *One Pole Monster*. URL: https://urs.silvrback.com/one-pole-monster (visited on 01/13/2024).

[3] Urs Heckmann. *One Pole Unlimited*. URL: https://urs.silvrback.com/one-pole-unlimited (visited on 01/13/2024).

[4] Urs Heckmann. *Zero Delay Feedback*. URL: https://urs.silvrback.com/zero-delay-feedback (visited on 01/13/2024).

[5] Antti Huovilainen. "Non-Linear Digital Implementation of the Moog Ladder Filter". In: (). URL: https://www.dafx.de/paper-archive/2004/P_061.PDF.

[6] Lantertronics - Aaron Lanterman. *ECE4450 L22: Moog Ladder Filters Analyzed (Analog Circuits for Music Synthesis, Georgia Tech course)*. Apr. 2, 2020. URL: https://www.youtube.com/watch?v=IRxeIEeAAnU (visited on 01/13/2024).

[7] Andrew Simper. *ADC-2020 - Circuit to Code*. 2020. URL: https://cytomic.com/files/dsp/adc-2020-andrew-simper-circuit-to-code-slides.pdf.

[8] Andrew Simper. "Direct Numerical Integration of a One Pole Linear Low Pass Filter". In: (). URL: https://cytomic.com/files/dsp/OnePoleLinearLowPass.pdf.

[9] Vadim Zavalishin. *The Art of VA Filter Design*. 2020. URL: https://www.discodsp.net/VAFilterDesign_2.1.2.pdf.