Assignment: Moog VCF

S. Bilbao, Acoustics and Audio Group, University of Edinburgh

Assignment structure

This assignment has three main parts for the basic coding.

1. Design a standalone script (not a function) that simulates the Moog VCF using the forward Euler and backward Euler schemes, as described in the lecture. The parameters which should be set in the preamble to your code are:

   - SR, the sample rate, in Hz
   - Tf, the total duration of the simulation, in seconds
   - $f0$, the cutoff frequency, in Hz
   - r, the resonance parameter (between 0 and 1)

   Your initial conditions should be set to zero. In addition, you will need to supply a forcing function u, which, for the generation of impulse responses and transfer functions, should have the form of a Kronecker delta function (i.e., a 1 followed by zeros).

   For forward Euler, your code should have an in-built check to verify the stability condition that you will derive for the scheme (framed as a condition on k, the time step, in terms of the parameters $\omega_0 = 2\pi f_0$ and r), as described below, and should exit with an error message if it is violated. Your code should output a plot of the transfer function magnitude of the Moog VCF against frequency, on a loglog scale, for both the FE and BE schemes, overlaid with the exact transfer function magnitude for the ODE system, which can be derived directly from the state space system, as described in lecture. These plots should have a form similar to those shown in the lecture; please use different colors for the three overlaid plots.

   [`'MoogVCF_Surname.m'`]

Your code should be a "flat" scripts, and not a function, so that it requires no subsidiary function calls to bespoke functions you have written.

Resources

- DAFX 2006 and 2011 papers by Thomas Helie (freely available online).
- Stinchcombe's epic Moog VCF analysis: `www.timstinchcombe.co.uk`
- "Numerical Sound Synthesis", S. Bilbao. Wiley (2009).
- "Elementary Finite Different Schemes" (online lecture course). CCRMA, Stanford University. Julius O. Smith III.

  Link: `https://ccrma.stanford.edu/~jos/DigitizingNewton/DigitizingNewton.html`
- "Finite Difference Methods for Ordinary and Partial Differential Equations", R.J. LeVeque, SIAM (2007)

4.1 Modelling the Moog VCF Ladder Filter

In this assignment we will program a digital model of the classic Moog analogue voltage-controlled filter (VCF). This was originally patented by Bob Moog in the mid 1960s, and is now considered a classic piece of circuit design. In musical circles this filter is legendary, both for its sonically-appealing characteristics, and its easy, intuitive control. Some say it was the first 'musical' filter, and helped to spark the electronic music revolution. It was originally designed to lie within an analog signal chain for subtractive synthesis. Spectrally rich input waveforms, such as sawtooth and square waves, were filtered by the Moog VCF to create new and interesting sounds. This synthesis technique lent itself particularly well to bass and lead lines.

4.1.1 Virtual-Analog Approach

By the term *virtual-analog* we mean the practice of: 1. finding a suitable mathematical model describing the behaviour of musical analog gear and 2. solving the model using a digital computer. Notice that this approach is true for both "classic" virtual-analog (i.e. predating machine learning), as well as for machine learning-based approaches, though the nature of the mathematical models varies greatly between the two! In this lecture, we will approach the problem of modelling the Moog VCF by means of "classic" techniques. These techniques were originally developed by electrical engineers to model non-musical circuits, thus allowing to predict the behaviour of circuits without having to assemble the components together. For each circuit element, it is possible to write a relationship between the voltage across the element and the current flowing through it. Some elements, such as capacitors and inductors, behave linearly, and thus

$$\frac{dV_C}{dt} = \frac{I_C}{C}, \qquad V_L = L\frac{dI_L}{dt}. \tag{1}$$

where V_C, V_L are the capacitor's and inductor's voltages, I_C, I_L the currents, C is the capacitance, and L the inductance. On the other hand, some more complicated elements, such as transistor and diodes, are nonlinear. Thus, more complicated models are needed. For transistors, one popular model is due to *Ebers and Moll*, and it involves complicated exponential relationships between the base, emitter and collector currents and voltages.

4.1.2 A Quick Overview of The Moog Ladder Filter

Unsurprisingly, the Moog filter presents several nonlinear elements organised in a sequential (i.e. ladder-like) fashion. A figurative representation of the filter is given in Figure 1. The ladder comprises five differential pairs, yielding four stages in total. A differenetial pair, sketched in Figure 2, consists of two transistors $Q1$, $Q2$, whose emitters share the same voltage V_E and from which current I is drawn. Following the analysis of Stinchcombe (see *Resources*), one can derive a relationship between the output current I and the differential base voltage $V1 - V2$. The analysis is omitted here. Instead, we are focusing on the mathematical form of the resulting model.

4.1.3 State-Space Representation

Given a set of relationships such as (1), one may establish a model of a circuit as

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{b}u(t) \tag{2a}$$

$$y = \mathbf{c}^T\mathbf{x} \tag{2b}$$

In this form, called *state-space* form, we have defined an input $u(t)$ (which is just a scalar function of time), a *state vector* $\mathbf{x}(t)$, and an output $y(t)$. It turns out that this is a general description for single input-single output (SISO) *linear* circuits. What about the Moog? Of course, as we mentioned earlier, the transistors behave nonlinearly. However, in order to understand how the filter works in principle, we will now make the assumption of linearity. In practice, we will assume that the current and voltages are so small that the complicated Ebers-Moll model can in fact be written as a system of linear equations.

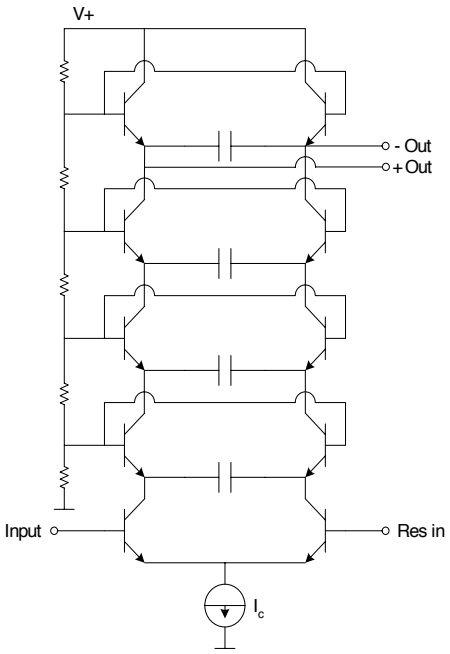

Figure 1: A schematic representation of the original Moog transistor ladder voltage-controlled filter (courtesy of Huovilainen, DAFx 2004).

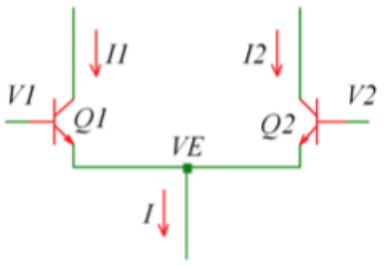

Figure 2: Differential pair (courtesy of Stinchcombe, see Resources).

4.1.4 Eigenvalues and Passivity

Before analysing the linear Moog filter, let us derive some general results valid for any circuit that can be written in the form (2). Of particular interest, is the nature of solutions to (2a) in case of zero-input. Hence, given

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} \tag{3}$$

how is $\mathbf{x}$ going to behave over time? You may start from the simple case of scalar $x(t)$, i.e.

$$\frac{dx}{dt} = Ax \qquad \rightarrow \qquad x = x_0 e^{At} \tag{4}$$

Hence, $x(t)$ will either grow or decay exponentially from its initial value x_0, according to the sign of A. Of course, in real circuits $x(t)$ cannot grow exponentially, and thus we expect any reasonable model to have sign(A)$= -1$. What about the vector case (3)? Here, the situation is a little more complicated. If you have notions of linear algebra, you will now that the evolution of $\mathbf{x}$ may be inferred by looking at the eigenvalues of $\mathbf{A}$. Now, $\mathbf{A}$ is a matrix with *real* coefficients. But the eigenvalues may nonetheless be complex, and in fact they appear in complex pairs. In order for $\mathbf{x}$ to decay over time, the eigenvalues of $\mathbf{A}$ must be such that

$$\text{Real}\left(\text{eig}\left(\mathbf{A}\right)\right) \leq 0 \tag{5}$$

Why must this be the case? You can use a frequency domain approach. Consider

$$\mathbf{x} = \hat{\mathbf{x}} e^{st}, \qquad s \in \mathbb{C} \tag{6}$$

Substituting in (3) gives

$$s\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} \tag{7}$$

which is the eigenvalue equation. Hence, $\mathbf{x}$ in (6) will decay if and only if the eigenvalues have negative real part. If *all* the eigenvalues of $\mathbf{A}$ have negative real part, then system (2) is said to be *passive*. Passivity is quite an important property to anyone wishing to model and simulate virtual-analog systems. First of all, every continuous model is expected to be passive. Second of all, the associated numerical scheme must preserve this property in the discrete setting, to avoid instability of the simulation. The case of the Moog VCF is a great example of how all of this blends in together. You are expected to understand passivity in both the continuous and digital realms.

4.1.5 Transfer Function

The frequency domain approach may be extended to include input and output. Define

$$\mathbf{x} = \hat{\mathbf{x}} e^{st}, \quad y = \hat{y} e^{st}, \quad u = \hat{u} e^{st}, \qquad s \in \mathbb{C} \tag{8}$$

and substitute these expressions in (2). You get

$$s\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{b}\hat{u} \tag{9a}$$

$$\hat{y} = \mathbf{c}^T \hat{\mathbf{x}} \tag{9b}$$

which implies that

$$\hat{y} = \underbrace{\mathbf{c}^T \left(s\mathbf{I} - \mathbf{A}\right)^{-1} \mathbf{b}}_{H(s)} \hat{u} \qquad \frac{\hat{y}(s)}{\hat{u}(s)} = H(s) \tag{10}$$

$H(s)$ is the input to output transfer function of the system. The meaning of the transfer function $H(s)$ is that one may compute the output $y(t)$ via an inverse Laplace transform. However, there is more useful information that can be extracted from the transfer function. One reason, which has already surfaced this discussion, has to do with *stability*: computing the poles of the transfer function gives an indication of how the solution will change over time (exponential decay/growth): the calculation of the poles is performed in the s domain. Second, one may restrict the attention to the case $s = j\omega$, and

look at both the *magnitude* and the *phase* of $H(j\omega)$ (i.e. the *frequency response*): such approach will tell you how each sinusoidal component is amplified/damped and phase-shifted by the filter.

In your assignment, you will need to plot the exact transfer function of the Moog VCF. In order to calculate this, the easiest way is to first define a vector of real frequencies at which you wish to evaluate the transfer function (and this should correspond to the frequency bins used in your plot of the transfer function for the numerical scheme). For each value f in the vector of frequencies, you can evaluate the continuous time transfer function H using $s = 2\pi jf$; you can then collect these values into a vector representing the transfer function evaluated for $s = 2\pi jf$.

4.1.6 State Space Form of the Moog VCF

We are finally ready to look at the linear model for the Moog VCF. If you are curious to know where the equations come from, have a look at the *Resources* listed above. The equations that describe the filter can be written as

$$\frac{1}{\omega_0}\frac{d}{dt}x_1 = -x_1 - 4rx_4 + u \tag{11a}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_2 = -x_2 + x_1 \tag{11b}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_3 = -x_3 + x_2 \tag{11c}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_4 = -x_4 + x_3 \tag{11d}$$

$$y = x_4 \tag{11e}$$

Here, $u(t)$ is an input voltage signal, $y(t)$ is an output signal, and $x_1, \ldots, x_4$ are the voltages across four capacitors in the system. $\omega_0 = 2\pi f_0$ is the resonance frequency for the filter, and $0 \leq r \leq 1$ is a *feedback* coefficient which determines the bandwidth of the filter. Here, u, the input signal, drives the first capacitor, and output y is drawn from the last capacitor. This system can be written in state space form as

$$\frac{d}{dt}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{b}u \tag{12a}$$

$$y = \mathbf{c}^T\mathbf{x} \tag{12b}$$

with

$$\mathbf{A} = \omega_0 \begin{bmatrix} -1 & 0 & 0 & -4r \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \qquad \mathbf{b} = \omega_0[1\ 0\ 0\ 0]^T \qquad \mathbf{c} = [0\ 0\ 0\ 1]^T \tag{13}$$

We can immediately write the transfer function as in (10). It is a little tedious to write it down directly (because you need to invert $(s\mathbf{I} - \mathbf{A})$). But it is not too hard to calculate it and plot it (this is part of your assignment). One thing which you can do, fairly easily, is find the system poles $s_1, \ldots, s_4$ (even if you don't know the transfer function). There are four of them, which are just given by the eigenvalues of $\mathbf{A}$, which are

$$s_m = \omega_0(-1 + \sqrt{2}r^{1/4}e^{j\frac{\pi+2\pi m}{4}}), \qquad m \in [0, 1, 2, 3] \tag{14}$$

Notice that the real parts of all these eigenvalues are negative or zero when $0 \leq r \leq 1$, so the system exhibits a non-increasing response, as pointed out in 4.1.4. The system poles form a well-known "cross" pattern in the left half plane, "centred" on $s = -\omega_0$. Two of the poles approach the $j\omega$ axis as r approaches 1...at which point the VCF becomes very highly resonant around the region of ω_0; this leads to the characteristic "ringy" sound of the filter around this cutoff frequency.

The condition $0 \leq r \leq 1$, then, is a condition for the Moog VCF to be well-behaved. This has nothing to do with simulation or discrete time, and is a condition we need to impose a priori.

4.1.7 Discrete Time

We are now turning the attention to discrete time and simulation. There are a number of ways for discretising system (12). Among them, one finds Euler's methods, Runge-Kutta methods, trapezoid rule, ... the list could go on. If you are curious about numerical methods, the book by LeVeque in the *Resources* section is a good place to start. Here, we are going to look at two methods, known as Forward and Backward Euler. These are

$$\text{(Forward)} \qquad \delta_{t+}\mathbf{x}^n = \mathbf{A}\mathbf{x}^n + \mathbf{b}u^n \qquad y^n = \mathbf{c}^T\mathbf{x}^n \tag{15}$$

$$\text{(Backward)} \qquad \delta_{t-}\mathbf{x}^n = \mathbf{A}\mathbf{x}^n + \mathbf{b}u^n \qquad y^n = \mathbf{c}^T\mathbf{x}^n \tag{16}$$

These can be expanded, respectively, as

$$\text{(Forward)} \qquad \mathbf{x}^{n+1} = (\mathbf{I} + k\mathbf{A})\,\mathbf{x}^n + k\mathbf{b}u^n \qquad y^n = \mathbf{c}^T\mathbf{x}^n \tag{17}$$

$$\text{(Backward)} \qquad (\mathbf{I} - k\mathbf{A})\,\mathbf{x}^{n+1} = \mathbf{x}^n + k\mathbf{b}u^{n+1} \qquad y^n = \mathbf{c}^T\mathbf{x}^n \tag{18}$$

In the above, we introduced the symbol k for the *time step*, which is the multiplicative inverse of the sample rate, $k = 1/f_s$. The index n indicates the time index. The two methods differ in that the forward method is *explicit*, whilst the backward method is *implicit*. Explicit methods are easy to compute: just use previous state and input values to get the update. Implicit methods, on the other hand, require the inversion of a matrix and, thus, a lot more effort to be computed. In spite of this added computational effort, implicit methods remain a popular choice in practice, particularly for nonlinear systems. Why? Because of their superior stability properties. Let us now derive the stability conditions for the two schemes above.

4.1.8 Stability of Numerical Methods

As for the continuous case, discrete numerical methods can be analysed in the frequency domain. The discrete-time version of the Laplace transform, the z-transform, gives

$$\mathbf{x}^n = \hat{\mathbf{x}}z^{-n}, \qquad z \in \mathbb{C} \tag{19}$$

From this definition, you can infer that exponentially growing behaviour can be avoided if and only if

$$|z^{-1}| \leq 1 \tag{20}$$

Applying the z-transform to both (17) and (18) (set $u^n = 0$), gives, respectively

$$\hat{\mathbf{x}}z^{-1} = (\mathbf{I} + k\mathbf{A})\,\hat{\mathbf{x}} \tag{21}$$

$$\hat{\mathbf{x}}z^{-1} = (\mathbf{I} - k\mathbf{A})^{-1}\,\hat{\mathbf{x}} \tag{22}$$

Hence, z^{-1} is just computed as any of the eigenvalues of the schemes' matrices. Hence, the stability conditions are, respectively,

$$|\text{eig}\,(\mathbf{I} + k\mathbf{A})\,| \leq 1 \tag{23}$$

$$|\text{eig}\,(\mathbf{I} - k\mathbf{A})^{-1}\,| \leq 1 \tag{24}$$

Exercise. *Solve* (23) *and state a stability condition for scheme* (15). First, let us write

$$\text{eig}\,(\mathbf{I} + k\mathbf{A}) = 1 + k\text{eig}(\mathbf{A}). \tag{25}$$

The eigenvalues of $\mathbf{A}$ are given by (14). By using the trigonometric expression in place of the complex exponential, we get

$$|1 + k\omega_0(-1 + ap_m + jaq_m)| \leq 1 \tag{26}$$

where we set

$$a \triangleq \sqrt{2}r^{1/4}, \quad p_m \triangleq \cos\left(\frac{\pi + 2m\pi}{4}\right), \quad q_m \triangleq \sin\left(\frac{\pi + 2m\pi}{4}\right), \quad m \in [0, 1, 2, 3] \tag{27}$$

Now, the hard part, and this is for you. The condition above eventually becomes a stability condition of the following form:

$$k \leq k_{max}\left(\omega_0, r\right) \tag{28}$$

You need to find an explicit expression for the function k_{max}! This will go into your code as a check on the stability of FE for the Moog VCF.

Exercise. *Solve* (24) *and prove that scheme* (16) *is unconditionally stable.* First, let us rewrite (16) as

$$|1 - k\text{eig}(\mathbf{A})| \geq 1 \tag{29}$$

This gives

$$|1 + k\omega_0 - ak\omega_0(p_m + jq_m)| \geq 1 \tag{30}$$

Now, because $ap_m \leq 1$, one gets $1 + k\omega_0 - ak\omega_0 p_m \geq 1$, and this should be enough for you to complete the proof.

4.2 Assignment: Coding the Moog VCF Filter

The basic assignment involves

1. Coding the Forward Euler scheme 15
2. Coding the Backward Euler scheme 16
3. Computing the transfer function magnitudes of both schemes
4. Plotting these on top of the transfer function magnitude for the continuous model system (which you will need to compute separately).

Structure This is a helpful way of structuring your code

1. Begin with a header (in comments), containing title, date and author

```
%++++++++++++++++++++++++++++++++++++++++
%           Moog VCF
%
%     Stefan Bilbao
%       1 December 2020
%++++++++++++++++++++++++++++++++++++++++
```

2. Clear the workspace, close all figures, and reset the command line

3. Then, define a section with the *main parameters* (the parameters that the user can modify): these should be the base sample rate `SR`, the resonant frequency `f0` (in Hz), the value of the feedback coefficient `r`, the length of the simulation `Tf` (in seconds)

```
%++++++++++++++++++++++++++++++++++++++++
%   parameters
SR      = 44100 ;          % sample rate [Hz]
Tf       = 0.2 ;           % total simulation time [s]
f0      = 120 ;            % resonant filter frequency [Hz]
r       = 0.7 ;            % feedback coeff [choose 0 \leq r \leq 1]
```

4. The second section should contain all the *derived parameters*: the angular resonant frequency `om0 = 2*pi*f0`, the total number of samples for the simulation `Nf=floor(SR*Tf)`, the time step `k=1/SR`.

5. In the next section, check if the stability condition for Forward Euler is satisfied, and if it is not, ask Matlab to display an error message and exit the code. You will also have to make sure that Forward Euler is not computed in the main loop if the stability condition is violated. You will need the stability condition you calculated in this case! If you do not arrive at this expression, your code will still run sometimes, but there will be no warning if it starts producing explosive results.

6. In the next section, initialise the loop matrices and vectors, and allocate memory for the output vector. This should be

```
I      = {\tt 4 by 4 identity matrix} ;
A       = {\tt 4 by 4 matrix A};
b       = {\tt 4 by 1 vector b};
c       = {\tt 1 by 4 vector c};

%Bf      = {\tt set 4 by 4 matrix used for matrix multiplication in FE};
%Bb      = {\tt set 4 by 4 matrix used for matrix inversion or linear system solution in B

xf      = {\tt initialize 4 by 1 vector for FE state};
xb      = {\tt initialize 4 by 1 vector for BE state};

yf      = {\tt initialise Nf by 1 vector to hold output y from FE};
yb      = {\tt initialise Nf by 1 vector to hold output y from BE};

u      = {\tt initialise Nf by 1 vector to hold output input sequence (for impulse respons

tvec    = [0:Nf-1]'*k ;          %-- time vector for plots
fvec    = [0:Nf-1]'*SR/Nf        %-- frequency vector for plots
```

Note that I used the indices `f`, `b` for forward and backward, respectively.

7. Now, compute the main loop. This should look something like

```
%++++++++++++++++++++++++++++++++++++++++
%   main loop
tic;
for n = 1 : Nf

    % update state xf from n to n+1, with current sample of u as input (FE)
    % update state xb from n to n+1, with current sample of u as input (BE)

% write sample to output vector yf (FE)
% write sample to output vector yb (BE)

end
simTime = toc ;
%++++++++++++++++++++++++++++++++++++++++
```

   Make sure that the rest of the scheme is compact, you can literally use just a handful of lines. Avoid any intricacies and try to keep a plain and efficient style.

8. At the end of the loop, you can compute the transfer function via an FFT

```
% discrete transfer functions
Hf   = fft(yf) ;
Hb   = fft(yb) ;
```

   Note that for plotting, you will need to take the magnitude of these, which will be complex-valued!

   You will need to experiment with your run duration (Tf) to make sure you have computed the numerical solution for long enough such that the impulse has effectively died out, and you are not truncating a significant portion of the "tail" of the IR.

9. In the last section, use `subplot` to plot the magnitudes of the transfer functions, as well as the transfer function for the continuous system, which you will need to calculate separately. Make sure to add legends, titles, units, grids. Please use a loglog plot, using distinct colours for FE, BE and the exact transfer function magnitudes.

4.2.1 Matrix Inversion in Matlab

In Matlab, if you need to solve a system like

$$\mathbf{Q}\mathbf{w} = \mathbf{d} \tag{31}$$

with $\mathbf{Q}$ an $N \times N$ invertible matrix, and where $\mathbf{d}$ is a known $N \times 1$ vector, and where $\mathbf{w}$ is the unknown $N \times 1$ vector to be solved for, there are essentially two approaches:

- Calculate the inverse of $\mathbf{Q}$ directly as $\mathbf{Q}_{inv} = \mathbf{Q}^{-1}$, and then solve for $\mathbf{w}$ as $\mathbf{w} = \mathbf{Q}^{inv} * \mathbf{d}$.
- Solve the linear system directly, without calculating the inverse, as $\mathbf{w} = \mathbf{Q} \setminus \mathbf{d}$, using Matlab's backslash operation.

You should probably make use of the second in your code, as it should save on memory costs, and is good practice in general! We don't actually need to store the inverse.

4.3 Further Exploration

There are many avenues of further exploration! Here are just a few suggestions:

4.3.1 Transfer Functions

It's useful to think about how you would compute the transfer functions from the numerical schemes purely in the frequency domain—without having to run a scheme at all! Try to figure out how you would extend the frequency domain technique you've used to compute the transfer function magnitude for the continuous system to the discrete time case. This is how the videos in the lecture were created, incidentally. This is a better way to do it, as you don't need to truncate the infinite tail of the IR and take a Fourier transform...

4.3.2 Trapezoidal Integration

The forward and backward integration methods discussed here are only the starting point in ODE simulation design. There are many more advanced techniques. Here is one method, also implicit like the backward integration method, second-order accurate, and based on the use of the so-called trapezoidal integation rule. It may be written, in state-space form, as

$$(\mathbf{I} - k\mathbf{A}/2)\,\mathbf{x}^{n+1} \;=\; (\mathbf{I} + k\mathbf{A}/2)\,\mathbf{x}^n + \frac{k}{2}\mathbf{b}\left(u^n + u^{n+1}\right) \qquad y^n = \mathbf{c}^T\mathbf{x}^n \tag{32}$$

It is also unconditionally stable.

Perform the same steps as for the forward and backward integration methods, as outlined in section 4.2. That is, create a script which generates an impulse response using the numerical method given in (32) above, and plot the transfer function H as a function of frequency. Also, as in the previous assignment, plot the exact transfer function magnitude on the same axis.

4.3.3 The Full Nonlinear Moog VCF

The big extension for the Moog VCF, which is not really covered here in detail, is to include nonlinearity...leading to the characteristic analog warmth due to harmonic distortion. Here is the complete nonlinear system:

$$\frac{1}{\omega_0}\frac{d}{dt}x_1 = -\tanh(x_1) - \tanh(4rx_4 + u) \tag{33a}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_2 = -\tanh(x_2) + \tanh(x_1) \tag{33b}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_3 = -\tanh(x_3) + \tanh(x_2) \tag{33c}$$

$$\frac{1}{\omega_0}\frac{d}{dt}x_4 = -\tanh(x_4) + \tanh(x_3) \tag{33d}$$

$$y = x_4 \tag{33e}$$

This system, due to the presence of the five hyperbolic tangent functions, is clearly not linear! You can read about this system in more detail in Thomas Hélie's papers from the DAFx conferences in 2006 and 2011. You will score a lot of points if you decided to use the trapezoidal integration scheme for this one, involving a nonlinear root-finding algorithm such as Newton-Raphson. The simple choice of Forward Euler will probably be ok in terms of stability in some cases, although it will become unstable under various choices of the input parameters.