SIMULATION OF FENDER TYPE GUITAR PREAMP USING APPROXIMATION AND STATE-SPACE MODEL

Jaromir Macak, Jiri Schimmel
Department of Telecommunications
Brno University of Technology
Brno, Czech Republic
jaromir.macak@phd.feec.vutbr.cz
schimmel@feec.vutbr.cz

Martin Holters
Department of Signal Processing and Communications
Helmut Schmidt University
Hamburg, Germany
martin.holters@hsu-hh.de

ABSTRACT

This paper deals with usage of approximations for simulation of more complex audio circuits. A Fender type guitar preamp was chosen as a case study. This circuit contains two tubes and thus four nonlinear functions as well as it is a parametric circuit because of an integrated tone stack. A state-space approach was used for simulation and further, precomputed solution is approximated using nonuniform cubic splines.

1. INTRODUCTION

Despite the significant progress in research of analog audio circuit simulations, this topic has not been closed yet. Many papers have been published recently and they mainly focus on two approaches of simulation – nonlinear wave digital filters [1, 2] and the state-space approach which is represented by the DK-method [2, 3]. Comparing nonlinear wave digital filters to the DK-method, the DK-method offers a more circuit-oriented approach and is also suitable for circuits with more nonlinear components, while wave digital filters require transformation of Kirchhoff variables into wave variables and also embedding of more nonlinear functions is more difficult and often is solved using delayed nonlinear functions [4]. Both approaches require usage of precomputed look-up tables to be able to work in real-time efficiently as was mentioned in [5, 3, 6] but unfortunately, there are not many details about look-up tables and used interpolations, except for the paper [6] where bilinear interpolations with grids of 100×100 and 25×25 points have been used.

Furthermore, all the simulated audio circuits mostly consisted of one nonlinear component and therefore, the maximum dimension of the look-up table was two (a triode model consists of a nonlinear function of two input variables). However, the circuits are often more complicated although division into simpler blocks is used. One must take care about mutual interaction between connected blocks as can be seen in [7, 8] and in case of a tube circuit, one block of the circuit must contain two tubes.

The first successful attempt to simulate more complicated circuit was in [9] where a wah-effect with two transistors was modeled. However, this audio effect operates in the linear part of the transistor transfer function and therefore, although the model was nonlinear, one can solve the nonlinear equations in real-time using a fixed point iteration numerical algorithm. But this is not true for simulation of guitar distortion effect pedals or guitar tube preamps.

Therefore, this paper focuses on real-time simulation of a guitar Fender type tube preamp which consists of two triodes and an integrated tone stack. Firstly, an efficient implementation of cubic spline interpolation will be described. To reduce the amount of data required for the simulation, a nonuniform grid interpolation will be used. In order to get nonuniformly gridded data, an algorithm based on removing of the least important data points is designed. In the second part of the paper, the Fender type preamp circuit is introduced and a state-space model of the preamp is made with respect to efficient handling of circuit parameter changes which was introduced in [9]. Subsequently, a precomputation and an approximation of the state-space model is performed.

2. EFFICIENT APPROXIMATION OF PRECOMPUTED SOLUTION

Approximations often offer an efficient way of implementation of complex functions but the main drawback is necessity of data to be approximated or interpolated. Generally, this is not so serious problem when using modern computers. However, a smooth interpolation of nonlinear functions, especially in more dimensions, often leads to large amounts of data, which can be unpractical for implementation. Therefore, a trade-off between speed, memory demands and quality of approximation of a function must be made.

One can make use of techniques known from digital image signal processing but not all of them are suitable for audio signal processing (e.g. nearest neighbor). One of the most often used methods in audio signal processing is the linear interpolation. It offers very fast implementation even in more dimensions but it requires a lot of data to be interpolated and the main problem is non-smooth behaviour. Moreover, when it is used for approximation of a transfer function, it has similar properties as a piece-wise linear transfer function [10] which has unlimited spectrum bandwidth.

Smooth behaviour is provided by cubic interpolation. However, the computational complexity is much higher than with linear interpolation. Much better performance can be achieved by using cubic spline interpolation which has similar properties as cubic interpolation (smooth derivative up to second order). Furthermore, one can use a nonuniform grid of data to be interpolated, which can significantly reduce the amount of data.

2.1. Cubic Spline Interpolation with Constant Access

The cubic spline interpolation can be found in two forms – piece-wise cubic spline and B-spline. Since we are looking for interpolation which goes through the data points, we will further consider only the piece-wise natural cubic spline interpolation that is given by

$$y = a_i x^3 + b_i x^2 + c_i x + d_i \tag{1}$$

where a_i, b_i, c_i and d_i are spline coefficients and i denotes the interval used. Spline coefficients are derived from the boundary point values of the intervals in such a way that the first and second derivatives at the boundary points of the intervals are continuous. From these conditions, it is possible to obtain a set of equations whose solution leads to computation of the spline coefficients. However, this technique is described in various literature e.g. [11] and therefore it is not further addressed here.

When using splines in piece-wise form, the critical part of computation is determination of the interval from which the spline coefficients have to be used. The most efficient way is determination of the interval directly from the input value – it means assigning an arbitrary independent variable x to an integer interval i. First of all, consider a uniform grid spline with a step denoted by Δ. The step Δ must be sufficiently small to capture the shape of the function. The very small Δ can be also used for approximation of discontinuous function which is transfered into a very steep continuous function. The variable x can be a possibly negative fractional number while the interval i must be a nonnegative integer. Therefore, we introduce a mapping function

$$i = \lfloor mx \rfloor + o \tag{2}$$

where m denotes a multiplier constant and o an interval offset and $\lfloor\rfloor$ is floor function. Values of m and o will depend on spline break points in such a way that if x is a break point of a spline, then the term mx will be an integer number. The multiplier constant can be found as $m = \frac{1}{\Delta}$. The vector $\mathbf{x}_{\text{breaks}}$ which will hold spline knot values and will be further used for precomputation of a nonlinear function is constructed according to

$$\mathbf{x}_{\text{breaks}} = \mathbf{n}\Delta \tag{3}$$

where

$$\mathbf{n} = \{N_{\min}, N_{\min}+1, \dots, -1, 0, 1, \dots, N_{\max}-1, N_{\max}\} \tag{4}$$

where $N_{\min} = \lfloor \frac{x_{\min}}{\Delta} \rfloor$, $N_{\max} = \lfloor \frac{x_{\max}}{\Delta} \rfloor + 1$ and the offset $o = -N_{\min}$.

When the vector $\mathbf{x}_{\text{breaks}}$ is known, the precomputation of a nonlinear function for $\mathbf{x}_{\text{breaks}}$ values proceeds. After the precomputation of the nonlinear function, the spline coefficients are determined. The processing using the uniform grid is the following:

1. interval computation $i = \lfloor mx \rfloor + o$,
2. fractional part computation $x_{\text{p}} = x - x_{\text{breaks}}[i]$,
3. and finally interpolation $y = ((a_i x_{\text{p}} + b_i)x_{\text{p}} + c_i)x_{\text{p}} + d_i$ using Horner scheme which is more efficient than (1).

The interpolation in more dimensions is available using tensor product. In case of 2-D splines, the interpolation is in the form of

$$f(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{i,j} x^i y^j, \tag{5}$$

where $c_{i,j}$ are spline coefficients. Totally, 16 spline coefficients are required for one function evaluation. The computational scheme of the nonuniform 2-D spline is:

1. $i = \lfloor m_{\text{x}} x \rfloor + o_{\text{x}}$,
2. $j = \lfloor m_{\text{y}} y \rfloor + o_{\text{y}}$,
3. $x_{\text{part}} = x - x_{\text{breaks}}[i]$,
4. $y_{\text{part}} = y - y_{\text{breaks}}[j]$,
5. $a = ((c_{1,i,j} y_{\text{p}} + c_{2,i,j})y_{\text{p}} + c_{3,i,j})y_{\text{p}} + c_{4,i,j}$,
6. $b = ((c_{5,i,j} y_{\text{p}} + c_{6,i,j})y_{\text{p}} + c_{7,i,j})y_{\text{p}} + c_{8,i,j}$,
7. $c = ((c_{9,i,j} y_{\text{p}} + c_{10,i,j})y_{\text{p}} + c_{11,i,j})y_{\text{p}} + c_{12,i,j}$,
8. $d = ((c_{13,i,j} y_{\text{p}} + c_{14,i,j})y_{\text{p}} + c_{15,i,j})y_{\text{p}} + c_{16,i,j}$,
9. $f = ((a x_{\text{p}} + b)x_{\text{p}} + c)x_{\text{p}} + d$.

The computational complexity of the 2D spline routine is five times higher because it requires computation of five 1D spline functions. However, it is possible to use parallel processing using SIMD instructions. In that case, the splines in rows from 5 to 8 are computed as one spline function if single precision floating numbers are used and then the computational complexity is only two times higher than for 1D spline. However, to allow parallel computation, the spline coefficients have to be properly ordered and aligned in memory.

Similarly, the 3D spline is given by

$$f(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} c_{i,j,k} x^i y^j z^k \tag{6}$$

and its computation requires 64 coefficients per one value and 21 of 1D spline computations, which can be reduced to 6 if parallel processing is used. However, it is obvious that the computational complexity and foremost number of coefficients start increasing rapidly. Therefore, an effective way of reducing data has to be found.

2.2. Nonuniform Spline Interpolation with Constant Access

When using nonuniform grid interpolation, some points from the abscissa vector $\mathbf{x}_{\text{breaks}}$ are removed and new spline coefficients are computed. However, in this case, the interval computation is more complicated because if (2) is used, the interval indeces are not consistent with the new ones anymore. Nevertheless, this can be solved by introduction of a simple interval mapping function given by a vector f_{mapping} which will translate the interval indexes from the uniform grid to the nonuniform in a such way that several original indexes will belong to one new index. The new computational scheme for 1D spline is

1. $j = \lfloor mx \rfloor + o$,
2. $i = f_{\text{mapping}}[j]$,
3. $x_{\text{part}} = x - x_{\text{breaks}}[i]$,
4. $y = ((a_i x_{\text{part}} + b_i)x_{\text{part}} + c_i)x_{\text{part}} + d_i$

and can be similarly extended to more dimensions. This modification enables reduction of the look-up table while there is only one extra assignment $i = f_{\text{mapping}}[j]$. The drawback is that extra memory for mapping data is required, but mapping functions are one dimensional even for more dimensional spline interpolations.

2.3. Reduction of Spline Coefficients

The efficient nonuniform gridded spline interpolation was shown in the previous section. However, the precomputed solution has a uniform grid. Therefore, an algorithm that removes unimportant data points from the regular grid is designed. There is a constraint given by the constant access that the data must remain on a regular grid and cannot be scattered. While some data points are being removed from the input vector in case of 1D splines, whole rows or columns must be removed from the input matrix in case of 2D splines and similarly in higher dimensions. The algorithm goes through all data points except the boundary points. Each point is removed and a spline function is constructed from the reduced data. Then a data point with the smallest error between the reduced spline interpolation and full data interpolation is removed. After that, the next iterations on the reduced data are performed until the error is equal to the given error. Finally, a mapping function between the uniform and nonuniform grid in each dimension must be determined. The algorithm for reduction of data in more dimensions is shown in listing 1.

Algorithm 1 `dataReduction(x,f,maxerr)`

```
1:  [$Dimensions$, $N$] = size($x$)
2:  $x_n = x$
3:  $f_n = f$
4:  while ( $err < maxerr$) do
5:      for $d$ = 1 to $Dimensions$ do
6:          for $i$ = 2 to $N[d]$ - 2 do
7:              $x_r = x_n$
8:              $f_r = f_n$
9:              remove $x_r[i,d]$ from $x_r$
10:             remove $f_r[i,d]$ from $f_r$
11:             $coef$ = buildspline($x_r$, $f_r$)
12:             $err[i,d] = f$ - inpterpolate($coef$,$x$)
13:         end for
14:     end for
15:     [$index$, $dim$] = position of min($err$)
16:     remove $x_n[index,dim]$ from $x_n$
17:     remove $f_n[index,dim]$ from $f_n$
18: end while
19: return $x_n$, $f_n$
```

3. APPROXIMATION OF STATE-SPACE NONLINEARITY

The nonlinear state-space system that will be used for modelling has the form [3]

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{B}\mathbf{u}(n) + \mathbf{C}\mathbf{i}(n) \quad (7)$$

$$y(n) = \mathbf{D}\mathbf{x}(n) + \mathbf{E}\mathbf{u}(n) + \mathbf{F}\mathbf{i}(n) \quad (8)$$

$$\mathbf{v}(n) = \mathbf{G}\mathbf{x}(n) + \mathbf{H}\mathbf{u}(n) + \mathbf{K}\mathbf{i}(n) \quad (9)$$

where $\mathbf{x}(n)$ holds the states, $\mathbf{u}(n)$ is the input vector, $y(n)$ is the output, the currents through the nonlinear circuit elements are collected in $\mathbf{i}(n)$ and the respective voltages in $\mathbf{v}(n)$.

In addition to the linear relationship of (9), the voltages and currents of the triodes are related by the current-voltage law

$$\mathbf{i}(n) = f\big(\mathbf{v}(n)\big) \quad (10)$$

of the nonlinear elements. The simulation thus proceeds by performing for each sample the following steps:

1. Calculate $\mathbf{p}(n) = \mathbf{G}\mathbf{x}(n) + \mathbf{H}\mathbf{u}(n)$.
2. Solve $\mathbf{v}(n) = \mathbf{p}(n) + \mathbf{K}\mathbf{i}(n)$ together with (10) to determine $\mathbf{i}(n)$ for the current values of $\mathbf{p}(n)$ and $\mathbf{K}$.
3. Compute the output with (8).
4. Update the states with (7).

The core of the state-space nonlinearity is given by the implicit formulation

$$\mathbf{v}(n) = \mathbf{K}\mathbf{i}(n) + \mathbf{p}(n) = \mathbf{K}f\big(\mathbf{v}(n)\big) + \mathbf{p}(n). \quad (11)$$

Considering a precomputation of such a system, the maximal dimension of the precomputed solution will be given by the number of inputs $\mathbf{p}$, however some input parameters $\mathbf{p}$ can be constant for some circuit topologies and therefore the dimension can be lower than the maximum and is equal to the number of non-constant inputs. For parametric circuits, one has to also consider that the $\mathbf{K}$ coefficient matrix is dependent on parameter values and then the dimension of the precomputed solution must be extended by number of variable $\mathbf{K}$ coefficients, which leads to maximal dimension given by $N^2 + N$ where N is the number of nonlinear functions. The solution can also be precomputed with a parameter value as an input variable to the system – this is more efficient when the number of parameters is lower than the number of $\mathbf{K}$ coefficients that are being changed by these parameters. As a result in both cases, interpolation functions of high dimensions must be used and as was stated in chapter 2.1, it would require a lot of data and also computational power even if the nonuniform grid is used. However, the requirement for smoothness of the interpolation function can be related only to the inputs $\mathbf{p}$ because they ensure the smoothness of the transfer function of the nonlinear system. An additional linear interpolation of the solution for variable $\mathbf{K}$ coefficients or values of parameters can be used because it will always produce a smooth transfer function and furthermore, the parameter values are never given very precisely, especially in case of analog circuits with potentiometers.

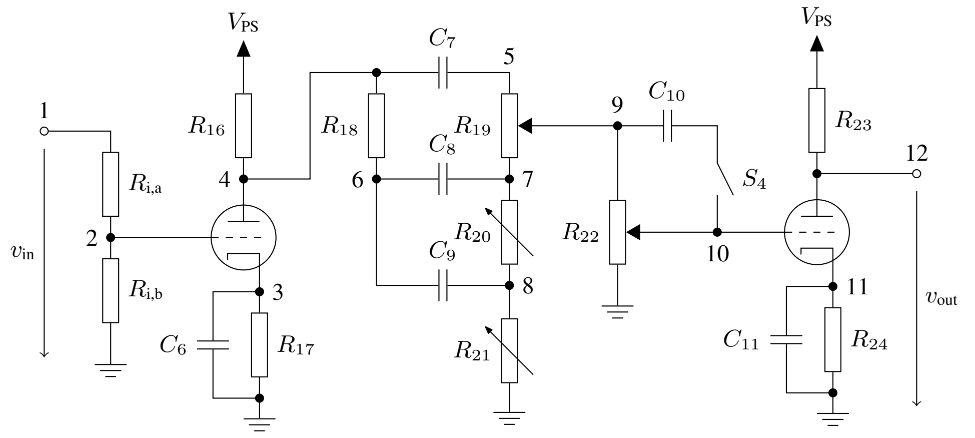
4. FENDER PREAMP SIMULATION

A Fender type tube guitar preamp is often used as a standard for clean and mildly overdriven guitar sounds, not just for Fender guitar amplifiers but also by other guitar amplifier manufacturers. The topology consists of two triodes and a tone stack that is connected between the triodes. The circuit schematic is shown in Figure 1.

4.1. Derivation of the state-space model

We follow the description of [9] to derive a state-space model of the circuit which allows for relatively easy handling of the potentiometers. The first step is to construct the incidence matrices which specify the connections of the circuit elements to the nodes. Dividing by type of element, we get $\mathbf{N}_\mathrm{R}$ for the constant resistors, $\mathbf{N}_\mathrm{v}$ for the variable resistors, $\mathbf{N}_\mathrm{x}$ for the capacitors, $\mathbf{N}_\mathrm{u}$ for the voltage sources (v_in and V_PS), $\mathbf{N}_\mathrm{n}$ for the triodes, and $\mathbf{N}_\mathrm{o}$ for the output v_out. These are sparse matrices with one row per respective element and one column per node in the schematic except for the reference node (ground). Each element is assigned a polarity and for element i having its positive terminal at node j_+ and its negative terminal at node j_-, the entry at (i, j_+) of the respective incidence matrix is set to 1, the entry at (i, j_-) is set to -1, where terminals connected to ground are ignored.

Each triode occupies two rows in $\mathbf{N}_\mathrm{n}$, representing the model with two voltage-controled current sources shown in Figure 2. Note that for consistency with [9], corresponding voltages and currents have opposite direction, yielding the somewhat unusual definition



C_6	22	µF
C_7	250	pF
C_8	100	nF
C_9	22	nF
C_{10}	120	pF
C_{11}	22	µF
$R_{i,a}$	34	kΩ
$R_{i,b}$	1	MΩ
R_{16}	100	kΩ
R_{17}	1500	Ω
R_{18}	100	kΩ
R_{19}	250	kΩ
R_{20}	250	kΩ
R_{21}	10	kΩ
R_{22}	1	MΩ
R_{23}	100	kΩ
R_{24}	1640	Ω

Figure 1: *Schematic of the modelled Fender type preamp.*

of grid and plate current to point out at the respective terminals. This results in the incidence matrix

$$\mathbf{N}_{\text{n}} = \begin{pmatrix} 0 & 1 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 1 & 0 \end{pmatrix}$$

where the all-zero columns 5–9 have been omitted. As the construction of the remaining incidence matrices is straight-forward, we shall not reproduce them here.

There are, however, two particularities which require a slight deviation from [9]. The first is the switch S_4, which changes the topology and even the effective order of the circuit, as C_{10} will not have any effect if S_4 is open. We handle this by deriving two models, one for S_4 closed, one for S_4 open. The latter model contains a 14th node to which only C_{10} is connected. The advantage compared to omitting the ineffective C_{10} is that the derived coefficient matrices will have the same dimensions with only somewhat different values. In paticular, the state update for S_4 open will leave the state variable corresponding to C_{10} unchanged. This allows for proper simulation of the switching behaviour.

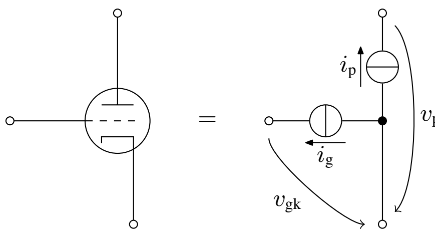
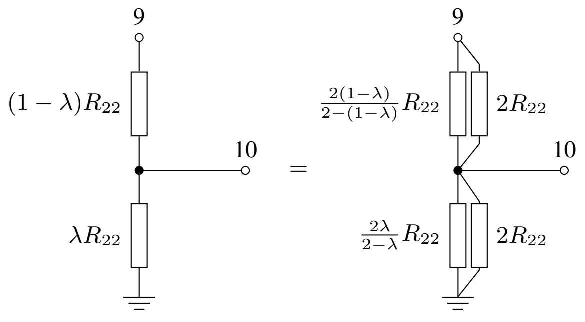


Figure 2: *Model of the triode as two voltage-controlled current sources.*

The second specialty arises from the way [9] handles variable parts. Namely, it first considers the circuit with all variable parts removed and assumes the modified nodal analysis' system matrix

$$\mathbf{S}_0 = \begin{pmatrix} \mathbf{N}_{\text{R}}^T \mathbf{G}_{\text{R}} \mathbf{N}_{\text{R}} + \mathbf{N}_{\text{x}}^T \mathbf{G}_{\text{x}} \mathbf{N}_{\text{x}} & \mathbf{N}_{\text{u}}^T \\ \mathbf{N}_{\text{u}} & \mathbf{0} \end{pmatrix} \tag{12}$$



9
9
10
10
=
$(1-\lambda)R_{22}$
λR_{22}
$\frac{2(1-\lambda)}{2-(1-\lambda)}R_{22}$
$2R_{22}$
$\frac{2\lambda}{2-\lambda}R_{22}$
$2R_{22}$

Figure 3: *Substitution of potentiometer R_{22} to avoid floating nodes 9 and 10.*

to be invertible, where $\mathbf{G}_{\text{R}}$ and $\mathbf{G}_{\text{x}}$ are diagonal matrices with the elements $\frac{1}{R_i}$ and $\frac{2C_i}{T}$, respectively, and T denotes the sampling interval. But the matrix $\mathbf{S}_0$ will be singular if there is any node not connected to the reference node with a chain of constant resistors and capacitors only. This, however is the case for nodes 9 and 10. To overcome this issue, we replace R_{22} as depicted in Figure 3, introducing virtual constant resistors of value $2R_{22}$ in parallel to the two sub-resistors of the potentiometer R_{22}. These sub-resistors in turn are changed from λR_{22} and $(1-\lambda)R_{22}$ to $\frac{2\lambda}{2-\lambda}R_{22}$ and $\frac{2(1-\lambda)}{2-(1-\lambda)}R_{22}$, respectively, where $\lambda \in [0,1]$ denotes the setting of the potentiometer, in order to yield the correct overall resistence. After augmenting $\mathbf{N}_{\text{R}}$ and $\mathbf{G}_{\text{R}}$ with the two extra resistors, we obtain an invertible $\mathbf{S}_0$.

With this modified $\mathbf{S}_0$, we derive the constant coefficient matrices $\mathbf{A}_0 \in \mathbb{R}^{6\times6}$, $\mathbf{B}_0 \in \mathbb{R}^{6\times2}$, $\mathbf{C}_0 \in \mathbb{R}^{6\times4}$, $\mathbf{D}_0 \in \mathbb{R}^{1\times6}$, $\mathbf{E}_0 \in \mathbb{R}^{1\times2}$, $\mathbf{F}_0 \in \mathbb{R}^{1\times4}$, $\mathbf{G}_0 \in \mathbb{R}^{4\times6}$, $\mathbf{H}_0 \in \mathbb{R}^{4\times2}$, $\mathbf{K}_0 \in \mathbb{R}^{4\times4}$ by

$$\begin{pmatrix} \mathbf{A}_0 + \mathbf{I} & \mathbf{B}_0 & \mathbf{C}_0 \\ \mathbf{D}_0 & \mathbf{E}_0 & \mathbf{F}_0 \\ \mathbf{G}_0 & \mathbf{H}_0 & \mathbf{K}_0 \end{pmatrix} = \begin{pmatrix} 2\mathbf{G}_{\text{x}}\mathbf{N}_{\text{x}} & \mathbf{0} \\ \mathbf{N}_{\text{o}} & \mathbf{0} \\ \mathbf{N}_{\text{n}} & \mathbf{0} \end{pmatrix} \mathbf{S}_0^{-1} \begin{pmatrix} \mathbf{N}_{\text{x}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{N}_{\text{n}} & \mathbf{0} \end{pmatrix}^T \tag{13}$$

where $\mathbf{0}$ and $\mathbf{I}$ are zero and identity matrices with dimensions as required by context. By further precalculating the constant helper

Table 1: *Parameters of the triode model used.*

t	G_t	C_t	ξ_t	μ
g	6.06×10^{-4}	13.9	1.354	—
k	2.14×10^{-3}	3.04	1.303	100.8

matrices

$$\begin{pmatrix}\mathbf{Q}\\ \mathbf{U}_\text{x}\\ \mathbf{U}_\text{o}\\ \mathbf{U}_\text{n}\\ \mathbf{U}_\text{u}\end{pmatrix} = \begin{pmatrix}\mathbf{N}_\text{v} & \mathbf{0}\\ \mathbf{N}_\text{x} & \mathbf{0}\\ \mathbf{N}_\text{o} & \mathbf{0}\\ \mathbf{N}_\text{n} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{pmatrix} \mathbf{S}_0^{-1} \begin{pmatrix}\mathbf{N}_\text{v}^T\\ \mathbf{0}\end{pmatrix} \tag{14}$$

we can then compute the final coefficient matrices by

$$\begin{pmatrix}\mathbf{A} & \mathbf{B} & \mathbf{C}\\ \mathbf{D} & \mathbf{E} & \mathbf{F}\\ \mathbf{G} & \mathbf{H} & \mathbf{K}\end{pmatrix} = \begin{pmatrix}\mathbf{A}_0 & \mathbf{B}_0 & \mathbf{C}_0\\ \mathbf{D}_0 & \mathbf{E}_0 & \mathbf{F}_0\\ \mathbf{G}_0 & \mathbf{H}_0 & \mathbf{K}_0\end{pmatrix} - \begin{pmatrix}2\mathbf{G}_\text{x}\mathbf{U}_\text{x}\\ \mathbf{U}_\text{o}\\ \mathbf{U}_\text{n}\end{pmatrix} (\mathbf{R}_\text{v} + \mathbf{Q})^{-1} \begin{pmatrix}\mathbf{U}_\text{x}\\ \mathbf{U}_\text{u}\\ \mathbf{U}_\text{n}\end{pmatrix}^T \tag{15}$$

where $\mathbf{R}_\text{v}$ is diagonal matrix with elements R_i for all variable resistors, i.e. θR_{19} and $(1-\theta)R_{19}$ for the treble control, βR_{20} for the bass control, μR_{21} for the middle control, and $\frac{2\lambda}{2-\lambda}R_{22}$ and $\frac{2(1-\lambda)}{2-(1-\lambda)}R_{22}$ for the level control as described above, where θ, β, μ, and λ denote the potentiometer settings.

For the nonlinear current-voltage law (10), we apply the triode model proposed in [12], given by

$$i_\text{g} = -G_\text{g} \cdot \left(\log\left(1 + \exp(C_\text{g} v_\text{g})\right) \cdot \frac{1}{C_\text{g}}\right)^{\xi_\text{g}}$$
$$i_\text{p} = -G_\text{k} \cdot \left(\log\left(1 + \exp\left(C_\text{k}(\tfrac{v_\text{p}}{\mu} + v_\text{g})\right)\right) \cdot \frac{1}{C_\text{k}}\right)^{\xi_\text{k}} - i_\text{g}$$

for the polarity defined above with the parameters of Table 1.

The values of $\mathbf{K}$ depend on the potentiometer settings and the position of the switch S_4. Closer examination reveals that only the elements k_{22}, k_{23}, k_{32}, and k_{33} change while the remaining twelve values remain fixed. This can be seen from the fact that all variable circuit elements are in the part connecting the plate of the first triode with the grid of the second triode, corresponding to the second and third elements of $\mathbf{v}$ and $\mathbf{i}$, and are not connected to the other triode terminals. By further observing that $\mathbf{K}$ is symmetric and hence $k_{23} = k_{32}$, the five variable parts affect only the three independent entries k_{22}, k_{23}, and k_{33} of the matrix $\mathbf{K}$.

4.2. Precomputation

As can be seen from (11) and the dimension of matrix $\mathbf{K}$, the nonlinear equation has 4 independent inputs for which it has to be precomputed. First of all, it is necessary to find ranges for input variables p_1, p_2, p_3 and p_4. This can be done using $\mathbf{p}(n) = \mathbf{G}\mathbf{x}(n) + \mathbf{H}\mathbf{u}(n)$ and it will depend on input signal values and circuit state values as well. The range of input values can be easily derived from input signal properties and also from power supply value V_PS, obtaining the range of state variables is much more complicated. The values of state variable will belong to interval $[0, V_\text{PS}]$ but mostly, the range will be much narrower. However, the ranges can be also estimated from performed simulations without an approximation. The ranges of input variables p_1, p_2, p_3 and p_4 used for this simulation are in Table 2. Similarly, the ranges for $\mathbf{K}$ matrix coefficients can be obtained from (15) for variable $\mathbf{R_v}$ dependent on parameter values θ, β, μ, and λ. In this case, the derivation of parameter ranges is much easier, the range is $[0, 1]$ for all parameters and the range of $\mathbf{K}$ matrix coefficients is given by minimal and maximal value for all combinations of discretized parameter values. The ranges of $\mathbf{K}$ coefficients are in Table 3.

Table 2: *Ranges of* $\mathbf{p}$ *parameters.*

	p_1	p_2	p_3	p_4
min	-4	200	-400	200
max	4	400	400	400
step	0.125	4	0.25	0.25

Table 3: *Ranges of* $\mathbf{K}$ *coefficients.*

	k_{22}	k_{23}	k_{33}
min	3.3×10^4	0.0	0.4
max	4.6×10^4	3.8×10^4	1.3×10^5

The process of precomputation itself is rather computationally demanding. Since the multivariate nonlinear equation tends to oscillate, one must use very small step between neighboring data points to force the solution to converge and it costs a lot of time and memory. However, the nonlinear equation (11) can be split in this case into

$$v_\text{gk1} = k_{11} i_\text{g}(v_\text{gk1}) + k_{12} i_\text{p}(v_\text{gk1}, v_\text{pk1}) + p_1$$
$$v_\text{pk1} = k_{21} i_\text{g}(v_\text{gk1}) + k_{22} i_\text{p}(v_\text{gk1}, v_\text{pk1}) + \underbrace{k_{23} i_\text{g}(v_\text{gk2}) + p_2}_{\overline{p_2}} \tag{16}$$

and

$$v_\text{gk2} = \underbrace{p_3 + k_{32} i_\text{p}(v_\text{gk1}, v_\text{pk1})}_{\overline{p_3}} + k_{33} i_\text{g}(v_\text{gk2}) + k_{34} i_\text{p}(v_\text{gk2}, v_\text{pk2})$$
$$v_\text{pk2} = k_{43} i_\text{g}(v_\text{gk2}) + k_{44} i_\text{p}(v_\text{gk2}, v_\text{pk2}) + p_4 \tag{17}$$

where terms $k_{23} i_\text{g}(v_\text{gk2})$ and $k_{32} i_\text{p}(v_\text{gk1}, v_\text{pk1})$ are mutual impacts of the adjacent triodes. Equations (16) and (17) can be precomputed separately for input variables p_1, $\overline{p_2}$ and $\overline{p_3}$,p_4 and further functions

$$\overline{f_\text{ip1}}(p_1, \overline{p_2}) = i_\text{p}(v_\text{gk1}(p_1, \overline{p_2}), v_\text{pk1}(p_1, \overline{p_2})) \tag{18}$$

$$\overline{f_\text{ig2}}(\overline{p_3}, p_4) = i_\text{g}(v_\text{gk1}(\overline{p_3}, p_4), v_\text{pk1}(\overline{p_3}, p_4)) \tag{19}$$

can be introduced. Subsequently, the plate current i_p1 can be computed from

$$i_\text{p1} = \overline{f_\text{ip1}}(p_1, p_2 + \overline{f_\text{ig2}}(p_3 + i_\text{p1}, p_4)) \tag{20}$$

for p_1, p_2, p_3 and p_4 then

$$i_\text{g2} = \overline{f_\text{ig2}}(p_3 + i_\text{p1}, p_4) \tag{21}$$

and finally, all the remaining currents or voltages can be derived from (16) and (17). Both approaches should give similar solutions of (11) – it depends only on the quality of approximation of the partial functions. As a result of the precomputation, there are four 4D look-up tables for circuit currents or voltages for constant parameters or 7D look-up tables for the parametric circuit. As was mentioned earlier, the p_1, p_2, p_3 and p_4 part of the look-up tables should be interpolated with smooth interpolation, while for the parametric part linear interpolation is sufficient. The number of data points can be reduced by algorithm (1). However, this was not performed due to extremely high computational demands.

4.3. Further Look-up Table Size Reduction

The look-up tables designed in the previous chapter are sufficient for working in real-time. Nevertheless, they are quite impractical for implementation of the algorithm which would simulate the circuit because the amount of data is still large. Therefore, a further compression of data would be beneficial. By closer look at the $\mathbf{K}$ matrix, one can observe some zero or low-value coefficients compared to the other values. Consequently, a correlation analysis of precomputed data was performed. Covariance matrices for input variables and function values were in form (expressed only for one row of input data)

$$C = \operatorname{cov}\left(\left[p_1^{i1}\, p_2^{i2}\, p_3^{i3}\, p_4^{i4}\, f(p_1^{i1}, p_2^{i2}, p_3^{i3}, p_4^{i4})\right]\right) \tag{22}$$

where the superscripts denote indexes $i_1 \in [1, N_1]$, $i_2 \in [1, N_2]$, $i_13 \in [1, N_3]$ and $i_4 \in [1, N_4]$ and function $f(p_1, p_2, p_3, p_4)$ was substituted with precomputed nonlinear current functions $i_{\mathrm{g1}}(\mathbf{p})$, $i_{\mathrm{p1}}(\mathbf{p})$, $i_{\mathrm{g2}}(\mathbf{p})$ and $i_{\mathrm{p2}}(\mathbf{p})$. The resulting covariance between nonlinear functions and inputs p_1, p_2, p_3 and p_4 is stated in Table 4.

Table 4: *Covariance between precomputed functions and inputs.*

	$i_{\mathrm{g1}}(\mathbf{p})$	$i_{\mathrm{p1}}(\mathbf{p})$	$i_{\mathrm{g2}}(\mathbf{p})$	$i_{\mathrm{p2}}(\mathbf{p})$
p_1	-5×10^{-4}	-7×10^{-3}	6×10^{-4}	9×10^{-4}
p_2	1×10^{-6}	-1×10^{-1}	7×10^{-3}	1×10^{-2}
p_3	-1×10^{-8}	1×10^{-3}	-1×10^{-2}	-3×10^{-2}
p_4	5×10^{-14}	5×10^{-9}	5×10^{-8}	-9×10^{-3}

As can be seen from the table, some nonlinear functions are almost independent of some input variables – functions $i_{\mathrm{g1}}(\mathbf{p})$, $i_{\mathrm{p1}}(\mathbf{p})$, $i_{\mathrm{g2}}(\mathbf{p})$ are independent of variable p_4 and the function $i_{\mathrm{g1}}(\mathbf{p})$ is further almost independent of variable p_3. Therefore, the look-up table $i_{\mathrm{g1}}(\mathbf{p})$ can have only 2 dimensions for constant parameters and 5 dimensions for the parametric circuit, the look-up tables $i_{\mathrm{p1}}(\mathbf{p})$, $i_{\mathrm{g2}}(\mathbf{p})$ are 3D for constant and 6D for parametric circuit and the look-up table $i_{\mathrm{p2}}(\mathbf{p})$ remains the same.

However, due to missing connections between nonlinear functions, a further simplification and decomposition can be introduced as was described in section 4.2 by equations (16) and (17). The whole model is split into two independent parts, each containing one tube. The grid and plate currents are computed from the approximated functions

$$i_{\mathrm{g1}} = i_{\mathrm{g1app}}(p_1, p_2) \tag{23}$$

$$i_{\mathrm{p1}} = i_{\mathrm{p1app}}(p_1, p_2, p_3) \tag{24}$$

where the redundant inputs were neglected. Then, the mutual interaction of both tubes is known and has been already included in the plate current i_{p1}. This current is subsequently used as an additional contribution to the input p_3 as can be seen in (17) and the currents of the second tube are obtained from

$$i_{\mathrm{g2}} = i_{\mathrm{g2app}}(p_3 + k_{32} i_{\mathrm{p1}}) \tag{25}$$

$$i_{\mathrm{p2}} = i_{\mathrm{g2app}}(p_3 + k_{32} i_{\mathrm{p1}}, p_4). \tag{26}$$

The big advantage of this decomposition is that it should not introduce any error (the results can differ but it is rather caused by numerical solving) if no p parameter is omitted. The last task is to consider efficient approximation regarding variable $\mathbf{K}$ matrix coefficients – k_{22}, $k_{23} = k_{32}$ and k_{33}. The grid current i_{g1} is independent on the second tube, therefore it should only depend on coefficient k_{22}, but in this case the current i_{g1} does not change with value of k_{22}. The plate current i_{p1} depends on all variable k coefficients and the currents of the second tube depend only on coefficient k_{33} because other coefficients have already been included in contribution of plate current to variable p_3. Eventually, it was found that using parameters θ, β, μ, and λ as inputs into look-up table for current i_{p1} gives smaller look-up table size than using coefficients k_{22},k_{23},k_{33} although the number of variable $\mathbf{K}$ coefficients is smaller than the number of parameters. This is because function i_{p1} changes dramatically with different k_{22},k_{23},k_{33} values and thus coefficients k_{22},k_{23},k_{33} must be more densely sampled, while with parameters θ, β, μ, λ and S_4 switch as the inputs into the look-up table, it seems to be sufficient to use only boundary values. Furthermore, the current i_{p1} is almost independent on bass parameter β – this is due to parallel combination of resistors R_{18} and R_{22} with serial combination $R_{20}\beta$ and R_{19}.

The final equations are

$$i_{\mathrm{g1}} = i_{\mathrm{g1app}}(p_1, p_2) \tag{27}$$

$$i_{\mathrm{p1}} = i_{\mathrm{p1app}}(p_1, p_2, p_3, \theta, \mu, \lambda, S_4) \tag{28}$$

$$i_{\mathrm{g2}} = i_{\mathrm{g2app}}(p_3 + k_{32} i_{\mathrm{p1}}, k_{33}) \tag{29}$$

$$i_{\mathrm{p2}} = i_{\mathrm{g2app}}(p_3 + k_{32} i_{\mathrm{p1}}, p_4, k_{33}) \tag{30}$$

and a detailed description of look-up tables is provided in Table 5 where the number of intervals per dimension per table is given. It consists of spline interpolations for $\mathbf{p}$ parameters and linear interpolations for circuit parameter changes. During the processing of the input signal, it is not necessary to compute all the interpolations, the parametric part can be interpolated only when the parameters are changed and resulting interpolated spline coefficients are stored in runtime memory of the algorithm. Then only the interpolation based on $\mathbf{p}$ variables is performed for each signal sample. The size of the runtime memory is shown in the third column of Table 5. The memory size required for the look-up tables is mentioned it Table 6. Comparing to original data size 53 GB per table (based on Table 2 and boundary values for θ, β, μ, and λ), a great reduction of data has been achieved. However, it is not sufficient for some applications, e.g. implementation on signal processors. In this case, it is possible to perform the interpolation directly from data points stored in look-up tables instead of spline coefficients. Then, the look-up table size is reduced (see the columns local in Table 6) but on the other hand, the runtime computation of whole splines will significantly increase the overall computational complexity. In such a case, it is better to use a local interpolation working on a nonuniform grid only from neighboring data points. However, these interpolations will be slower than the proposed spline interpolation – e.g. the cubic hermite interpolation between stored data points is approximately 3 times slower than the cubic spline interpolation.

Table 5: *Description of the look-up tables – number of intervals*

table	spline	lin. interp	runtime
i_{g1app}	13×6	-	13×6
i_{p1app}	$46 \times 14 \times 29$	$2 \times 2 \times 2 \times 2$	$46 \times 14 \times 29$
i_{g2app}	$15 \times 6 \times 13$	-	15×6
i_{p2app}	$38 \times 8 \times 12$	-	38×8

Table 6: *Data size of the look-up tables.*

table	global interp. [kB]		local interp. [kB]	
	total	runtime	total	runtime
i_{g1app}	3.75	3.75	0.30	0.30
i_{p1app}	65 520.00	4095.00	1167.26	72.95
i_{g2app}	210.00	4.38	4.57	0.35
i_{p2app}	712.25	16.20	14.25	1.19

5. RESULTS AND DISCUSSION

The algorithm for simulation of the preamp was written as a Matlab mex function in C language. The algorithm consists of spline interpolation up to third order for p variable interpolation and linear interpolation for parameter variable interpolation. The parameter interpolation is performed before the main processing loop. The Templetate Numerical Toolkit[1] was used for matrix operations. The computational complexity of the algorithm was around 10 % on a 2.66 GHz Intel processor but more than half of it was spent on the matrix operations. The original state-space model without the approximations consumes around 76 %. The algorithm was not optimizied using the parallel processing because it would require to write critical parts of the algorithm in assembly language or using intrinsic functions. As a result of this, a further reduction of the computational complexity is possible. The quality of approximation is illustrated in Figures 4, 5, 6, and 7. The input signal was a 100 Hz sinewave signal with an amplitude of 0.5 V and a sampling frequency of 48 kHz. The chosen error for the data reduction in the algorithm (1) was 1×10^{-6} A. Figures 4 and 5 show output signals in time and frequency domain for the numerical solution and the approximated solution for parameter values $\theta = 1$, $\beta = 1$, $\mu = 1$, $\lambda = 1$; that means without the interpolation of parameters. The interpolation for parameter values $\theta = 0.5$, $\beta = 0.5$, $\mu = 0.5$, $\lambda = 0.5$, which is the worst case, is shown in Figures 6 and 7. Although the difference of the numerical and approximated solution is quite large in time domain, the harmonic content is very similar. The sinewave signal mentioned earlier and a real guitar riff were used as input signals. There was only a very subtle difference for the sinewave signal and the version with parameters $\theta = 0.5$, $\beta = 0.5$, $\mu = 0.5$, $\lambda = 0.5$. The results for the guitar signal sounded similarly. Sound examples are available on the web page `www.utko.feec.vutbr.cz/~macak/DAFx12/`.

The most difficult part of the approximation is choosing the maximal error used during data reduction. Because the triode current functions were approximated, the output signal error is equal to the chosen error multiplied by the plate resistor value multiplied with the amplification factor of the next triode. The next error source are the constant p parameters. This error can be however only evaluated by comparing of the output signals with non-constant p parameters.

[1]http://math.nist.gov/tnt/

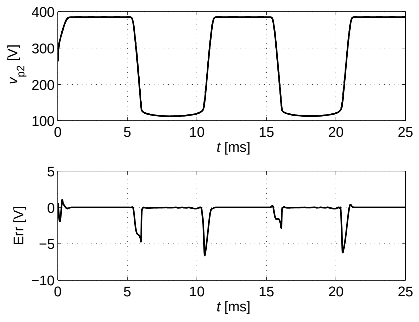


Figure 4: *Output signals (top, dashed line for numerical solution) and difference between numerical and approximated solution in time domain without parameter interpolation.*

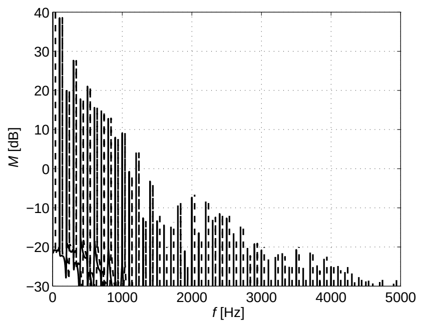


Figure 5: *Difference between numerical and approximated solution in time frequency without parameter interpolation. The numerical solution (dashed line) is shifted to the right.*

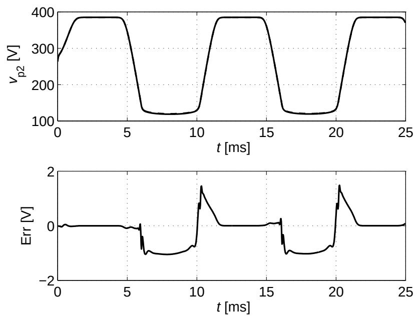


Figure 6: *Output signals (top, dashed line for numerical solution) and difference between numerical and approximated solution in time domain with parameter interpolation.*

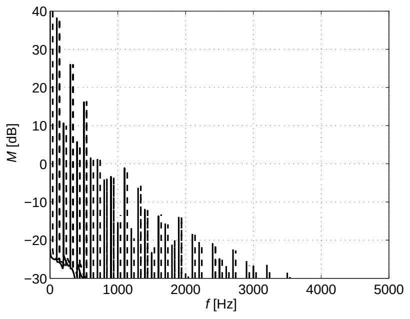


Figure 7: *Difference between numerical and approximated solution in time frequency with parameter interpolation. The numerical solution (dashed line) is shifted to the right.*

6. CONCLUSION

The usage of interpolations for real-time simulation of nonlinear analog audio circuits was discussed in this paper. As a case study, the Fender type guitar preamp was chosen as an example of a highly nonlinear and parametric circuit. The preamp was modeled using the state-space approach. Subsequently, a computationally efficient way of the implementation was discussed. The proposed algorithm makes use of the optimized piece-wise cubic spline interpolation up to three dimensions as the core. The parametric part of the nonlinear function is interpolated using linear interpolation.

Further reduction of look-up table sizes was achieved by decomposition of the nonlinearity into two independent nonlinear parts. The second nonlinear part can serve as an additional contribution to the input of the first nonlinearity. After the computation, the first nonlinearity is similarly used as the additional input to the second nonlinearity. Using this approach, even more complicated preamps with tubes connected in series can be simulated without decomposition into separate blocks. This will be verified in future work.

The algorithm provides quite low computational complexity considering the complexity of the simulated circuit and even further optimization is possible. The main drawback is the large amount of data required by the algorithm although the data was significantly reduced. Future work will be focused on the usage of efficient interpolations computed directly from stored data points instead of spline coefficients. A promising method seem to be Newton polynomial or spline interpolation with coefficients computed only from the neighborhood of the value to be interpolated. Future work can also be done on improvement of the algorithm for reduction of necessary data points. The criterial function used in this paper was the maximal difference of the transfer functions. A more efficient way would probably be the application of a criterial function based on a model of human hearing.

The modeled preamp has not been compared to the original one but the validity of using DK-method was proved in various literature and the main object of this paper was an efficient implementation of this method for a more complicated circuit. To compare the model with the original circuit, the tube parameters should be fitted to particular tubes used in the preamp, the circuit must be supplemented by capacitors responsible for the Miller effect and also by an appropriate load for the second tube. However, the DK-method is capable to handle all these modifications only by changing some matrices, while the nonlinear core remains the same.

7. ACKNOWLEDGMENTS

This research is part of the project reg. no CZ.1.07/2.3.00/20.0094 "Support for incorporating R&D teams in international cooperation in the area of image and audio signal processing" and is co-financed by the European Social Fund and the state budget of the Czech Republic.

8. REFERENCES

[1] J. Pakarinen and M. Karjalainen, “Enhanced wave digital triode model for real-time tube amplifier emulation,” *IEEE Trans. Audio, Speech & Language Processing*, vol. 18, no. 4, pp. 738–746, 2010.

[2] D. T. Yeh and J. O. Smith, “Simulating guitar distortion circuits using wave digital and nonlinear state-space formulations,” in *Proc. Digital Audio Effects (DAFx-08)*, Espoo, Finland, Sep. 1–4, 2008, pp. 19–26.

[3] D. T. Yeh, J. S. Abel, and J. O. Smith, “Automated physical modeling of nonlinear audio circuits for real-time audio effects - Part I: Theoretical development,” *IEEE Trans. Audio, Speech, and Language Processing*, vol. 18, no. 4, pp. 728–737, May 2010.

[4] R. C. D. de Paiva, J. Pakarinen, V. Välimäki, and M. Tikander, “Real-time audio transformer emulation for virtual tube amplifiers,” *EURASIP Journal on Advances in Signal Processing*, vol. 2011, pp. 15, 2011.

[5] J. Pakarinen and M. Karjalainen, “Wave digital simulation of a vacuum-tube amplifier,” in *Proc. Intl. Conf. on Acoustics, Speech, and Signal Proc.*, Toulouse, France, May 15–19, 2006, pp. 153–156.

[6] D. T. Yeh, “Automated physical modeling of nonlinear audio circuits for real-time audio effects - Part II: BJT and vacuum tube examples,” *IEEE Trans. Audio, Speech, and Language Processing*, vol. 20, no. 4, pp. 1207–1216, may 2012.

[7] J. Macak and J. Schimmel, “Real-time guitar preamp simulation using modified blockwise method and approximations,” *EURASIP Journal on Advances in Signal Processing*, vol. 2011, pp. 11, 2011.

[8] J. Macak and J. Schimmel, “Real-time guitar tube amplifier simulation using approximation of differential equations,” in *Proc. Digital Audio Effects (DAFx-10)*, Graz, Austria, Sep. 6–10, 2010.

[9] M. Holters and U. Zölzer, “Physical modelling of a wah-wah effect pedal as a case study for application of the nodal dk method to circuits with variable parts,” in *Proc. Digital Audio Effects (DAFx-11)*, Paris, France, Sept. 19–23,, 2011.

[10] J. Schimmel and J. Misurec, “Characteristics of broken-line approximation and its use in distortion audio effects,” in *Proc. Digital Audio Effects (DAFx-10)*, Bordeaux, France, Sept. 10–15, 2007.

[11] C. De Boor, *A Practical Guide to Splines*, Springer, New York, 1 edition, 2001.

[12] K. Dempwolf and U. Zölzer, “A physically-motivated triode model for circuit simulations,” in *Proc. Digital Audio Effects (DAFx-11)*, Paris, France, Sept. 19–23, 2011.