Virtual analog modeling - new ways how to achieve old sounds

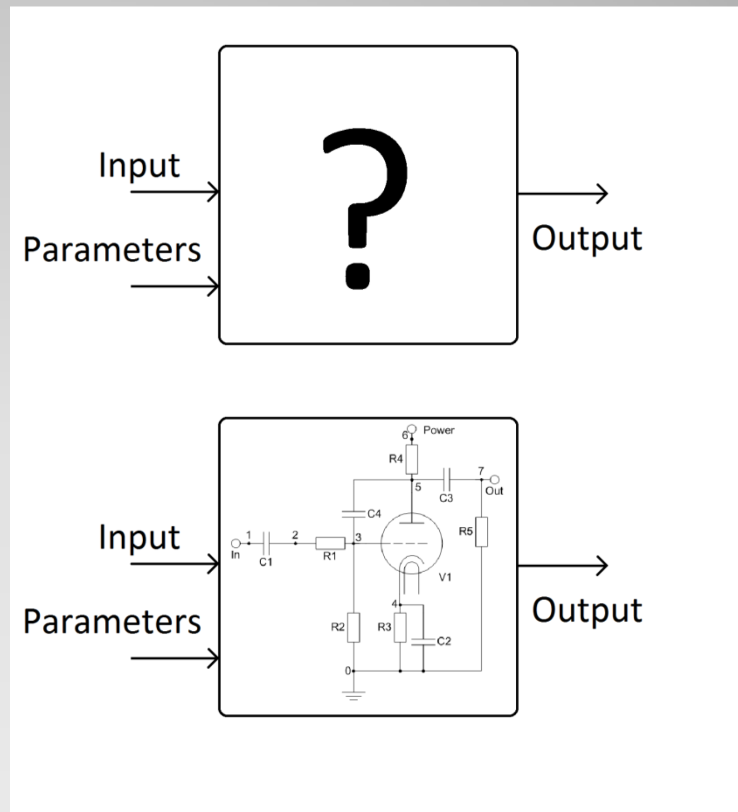
Jaromír Mačák
DAFx-16 Brno
5.9.2016

Outline

- Overview of methods for simulation of analog circuits in real time
- Circuit modeling using nodal DK method
- Example of the simulation step by step

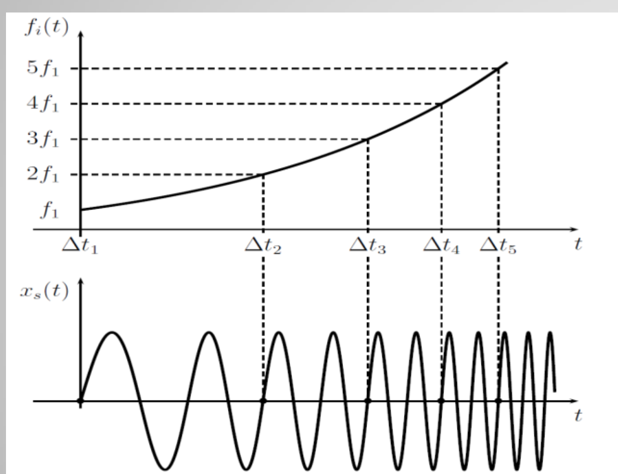
Simulation methods overview

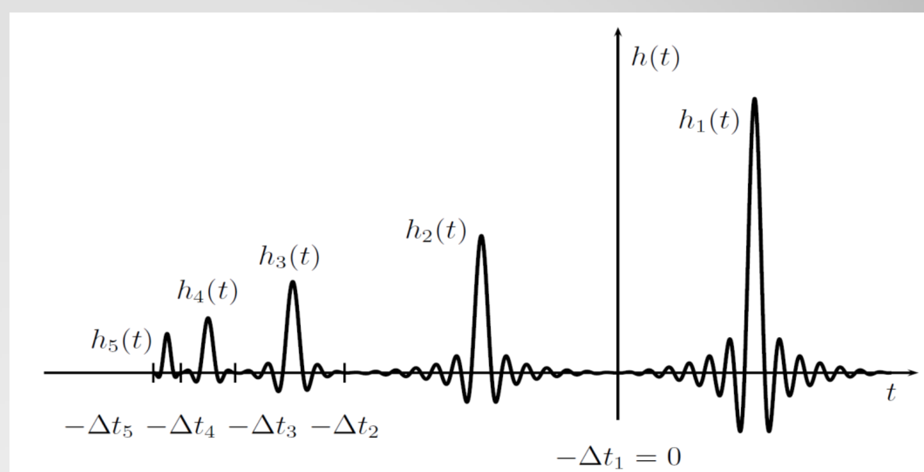
- Black box
  - Unknown structure
  - Measurement
  - General model
- White box
  - Known structure
  - Parametric model



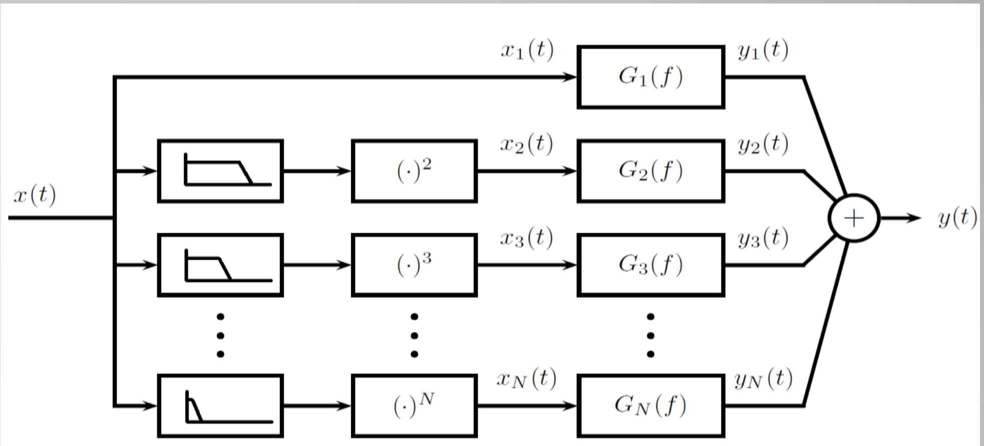
Black box approach
Introduction I

- Nonlinear system identification
- Swept sine signal analysis
- Extraction of impulse responses [Novak2010]





Black box approach Hammerstein model



$T_0(x) = 1$
$T_1(x) = x$
$T_2(x) = 2x^2 - 1$
$T_3(x) = 4x^3 - 3x$
$T_4(x) = 8x^4 - 8x^2 + 1$
$T_5(x) = 16x^5 - 20x^3 + 5x$
$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$
$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x$

- Chebyshev polynomials
- Computationally demanding for high order of system
- Can be reduced using PCA (principal component analysis) [Paiva2012]

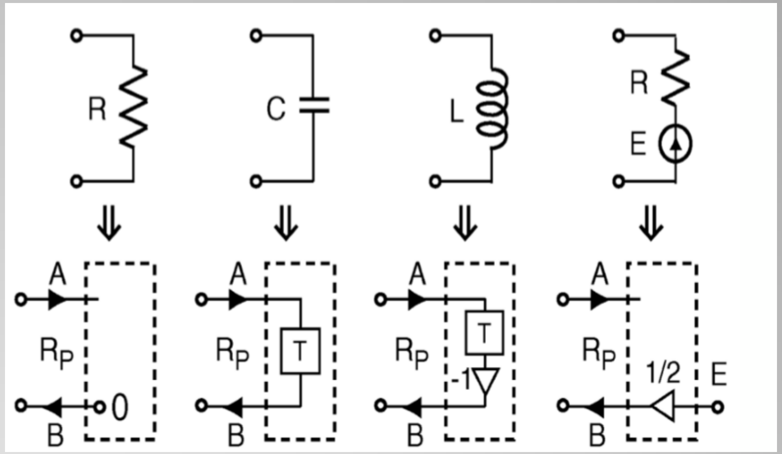
White box approach
Methods overview

- Based on analysis of circuit
- State space representation
  - $x[n] = Ax[n-1] + Bu[n]$
  - $y[n] = Dx[n-1] + Eu[n]$
  - Can be extended to nonlinear model

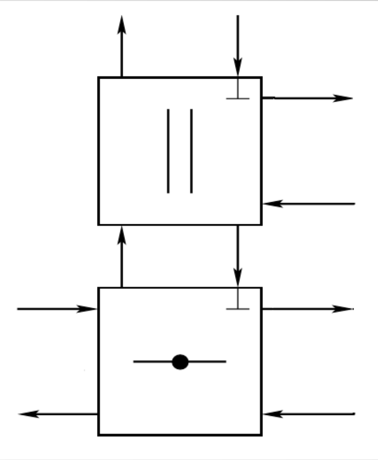
- Wave digital filters
  - Transformation of K variables V, I, R to wave variables
  - $A = V + RI$
  - $B = V - RI$
  - Can be extended to nonlinear model

Wave digital filters Introduction I

- WDF ports
  - transformed circuit elements
  - Incident wave A
  - Reflected wave B
  - Discretized using Bilinear transform
- WDF adaptors
  - Three port
  - Series, parallel connections
  - One reflection free port
  - All incident waves from the root must be reflection free



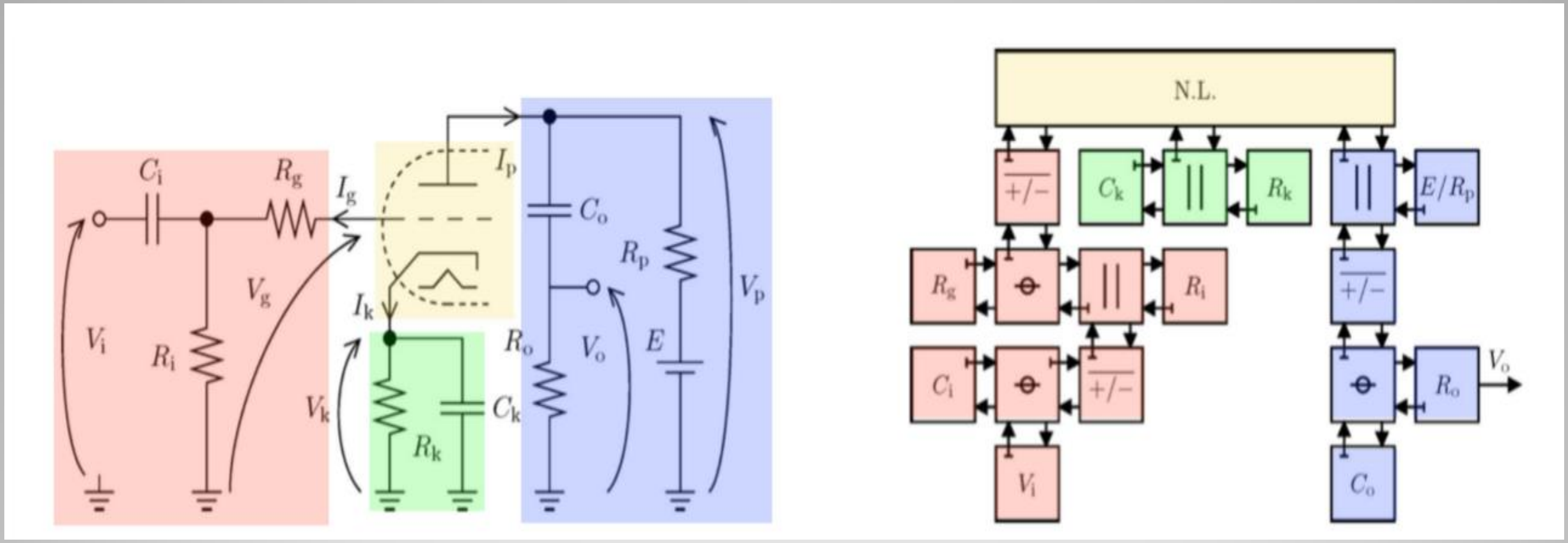
[Smith2015]



Wave digital filters
Introduction - BCT

- Building a binary connection tree BCT by connecting reflection free ports to match port resistances
- One root node in the tree
- Computationally very efficient and robust

Nonlinear wave digital filters
Triode amplifier example

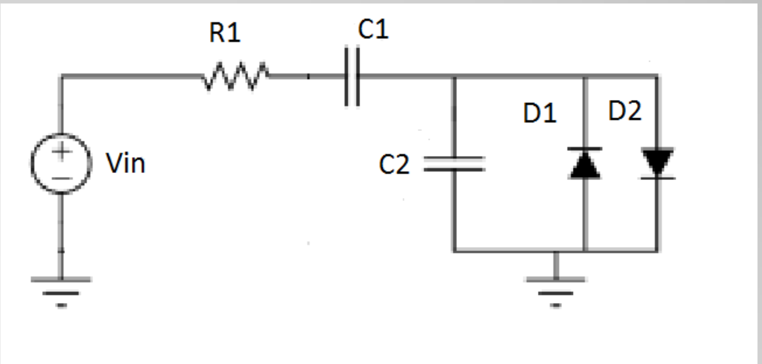


[Smith2015]

- Compute wave from the bottom leaves of the tree to the top
- Solving the nonlinearity at the root of the tree
- Update port resistances due to nonlinearity
- Compute waves back to the bottom leaves

Nonlinear wave digital filters
Nonlinear elements

- Efficient for single nonlinearity placed as the root of the binary connection tree
  - Distortion effect
  - Overdrive effect



- Multiple nonlinearities
  - Consolidated into single nonlinearity
  - Cross-control
  - Iterative schemes

Nonlinear wave digital filters Challenges

- Missing general method for handling multiple nonlinearities
- Missing general method for deriving WDF topology from the circuit topology
- Complex topologies
- Recent research about multiport nonlinearities

Nonlinear wave digital filters
How to start using it

- WDF Matlab framework
  - Available under DAFx book 2nd edition [Zölzer2011], matlab codes available at http://www.dafx.de/
- WDF JUCE framework
  - Available at https://forum.juce.com/t/wave-digital-filter-wdf-with-juce/11227
  - Not specified license terms
  - Explained at DAFx Keynote speech [Smith2015]

State space representation State space model & K method

- Provides general description for nonlinear systems in form:

  - $x[n] = Ax[n-1] + Bu[n] + Ci(v[n])$
  - $y[n] = Dx[n-1] + Eu[n] + Fi(v[n])$
  - $v[n] = Gx[n-1] + Hu[n] + Ki(v[n])$

  - Nonlinear function to be solved:
  - $0 = Gx[n-1] + Hu[n] + Ki(v[n]) - v[n] =$
    $= p + Ki(v[n]) - v[n] => i(v[n])$

State space representation
State space model & K method

- Provides general description for nonlinear systems in form:
  - $x[n] = Ax[n-1] + Bu[n] + Ci(v[n])$
  - $y[n] = Dx[n-1] + Eu[n] + Fi(v[n])$
  - $v[n] = Gx[n-1] + Hu[n] + Ki(v[n])$

  - Nonlinear function to be solved:
  - $0 = Gx[n-1] + Hu[n] + Ki(v[n]) - v[n] =$

    $= p + Ki(v[n]) - v[n]$

    $\Rightarrow i(v[n])$

State space representation
State space model & K method

- Provides general description for nonlinear systems in form:
  - $x[n] = Ax[n-1] + Bu[n] + Ci(v[n])$
  - $y[n] = Dx[n-1] + Eu[n] + Fi(v[n])$
  - $v[n] = Gx[n-1] + Hu[n] + Ki(v[n])$

  - Nonlinear function to be solved:
  - $0 = Gx[n-1] + Hu[n] + Ki(v[n]) - v[n] =$

    $= p + Ki(v[n]) - v[n]$

    $\Rightarrow i(v[n])$

State space representation
Deriving the model

- Several methods how to get the state space model introduced during last 6 years:
  - Automated DK model from netlist [Yeh2012]
  - DK method from mesh analysis [Dempwolf 2010]
  - Nodal DK method [Holters2011]
  - Nodal DK method framework [Benois2013]

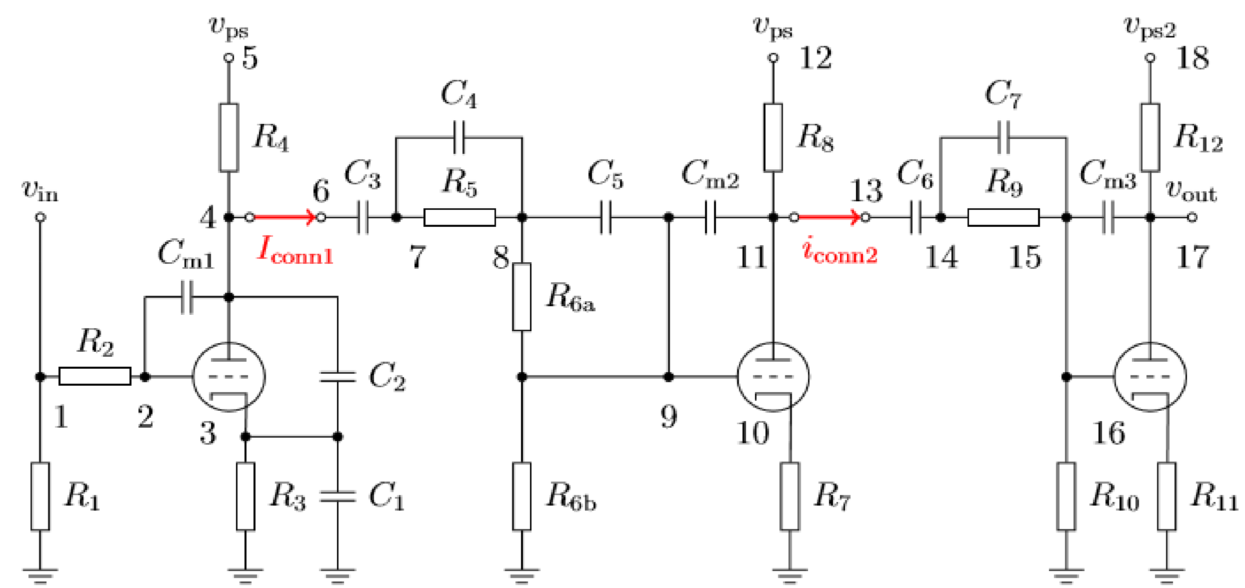
State space representation Nodal DK method

- Circuit described by several matrices defining the position of circuit elements and values
  - Resistors: N_R, G_R
  - Capacitors and Inductors: N_X, G_X
  - Voltage sources and OPAs: N_U
  - Nonlinear elements: N_N
  - Output ports: N_O
- General transformation from incidence matrices to state space model
- Solving the DK method model
  - Iterative numeric solver: $0 = p + Ki(v[n]) - v[n]$
  - Look-up table: $i = look\_up(p)$

State space representation Complex topologies I

- The method can be used for complex topologies – whole circuit
- Order of the nonlinear system to be solved is given by number of nonlinear functions
  - Problems with convergence
  - Computational heavy
  - Huge look-up tables

State space representation Complex topologies II



State space representation Complex topologies III

- Connection currents - iterative scheme for connection inner circuit blocks
- Works well together with inner block look-up tables

$$\mathbf{K} = \begin{bmatrix} 6.69\times10^{4} & 7.71\times10^{2} & 7.35\times10^{2} & 1.41\times10^{1} & 1.34\times10^{1} & 3.15\times10^{-1} \\ 7.71\times10^{2} & 4.75\times10^{4} & 4.62\times10^{4} & 8.88\times10^{2} & 8.45\times10^{2} & 1.98\times10^{1} \\ 7.35\times10^{2} & 4.62\times10^{4} & 7.69\times10^{4} & 1.13\times10^{4} & 1.22\times10^{3} & 2.87\times10^{1} \\ 1.41\times10^{1} & 8.88\times10^{2} & 1.13\times10^{4} & 9.01\times10^{4} & 7.62\times10^{4} & 1.79\times10^{3} \\ 1.34\times10^{1} & 8.45\times10^{2} & 1.22\times10^{3} & 7.62\times10^{4} & 9.39\times10^{4} & 3.00\times10^{3} \\ 3.15\times10^{-1} & 1.98\times10^{1} & 2.87\times10^{1} & 1.79\times10^{3} & 3.00\times10^{3} & 9.85\times10^{4} \end{bmatrix}$$

Nonlinear function:
0 = p + Ki(v[n]) - v[n]

$$\mathbf{K} = \begin{bmatrix} k_{11} & k_{12} & 0 & 0 & 0 & 0 & k_{17} & 0 \\ k_{21} & k_{22} & 0 & 0 & 0 & 0 & k_{27} & 0 \\ 0 & 0 & k_{33} & k_{34} & 0 & 0 & k_{37} & k_{38} \\ 0 & 0 & k_{43} & k_{44} & 0 & 0 & k_{47} & k_{48} \\ 0 & 0 & 0 & 0 & k_{55} & k_{56} & 0 & k_{58} \\ 0 & 0 & 0 & 0 & k_{65} & k_{66} & 0 & k_{68} \\ k_{71} & k_{72} & k_{73} & k_{74} & 0 & 0 & k_{77} & k_{78} \\ 0 & 0 & k_{83} & k_{84} & k_{85} & k_{86} & k_{87} & k_{88} \end{bmatrix}$$

Building DK method framework in Matlab

- Integration of the nodal DK method into Matlab classes
- Simplifies the design of the simulation model from the circuit schematic
- Can be used for direct generation of a VST plugin

Building DK method framework DKmodel class - properties

```
classdef DKmodel
       properties
              A,B,C,D,E,F,G,H,K % DK method matrices
              x % state variable
              U % inputs vector
              v % unknown voltages
              T % sampling period
              ...
       end
       methods
       ...
       end
end
```

Building DK method framework DKmodel class – basic functions

```
classdef DKmodel
    properties
    ...
    end
    methods
        function out = process(obj,in)
        function obj = buildModel(obj, components,
            components_count)
        function [v, I] = solve_nonlinear_func(obj, p, K)
        % OVERRIDE functions
        function [i, J] = nonlinearity(obj,v)
        function obj = load_input(obj,in)
        ...
     end
end
```

Building DK method framework DKmodel class – process

```
function [out, obj] = process(obj,in)
   out = zeros(size(in));% allocate output signal buffer
   for channel = 1:size(in,2) % channels loop
      for sample = 1:size(in,1) % samples loop
         obj = load_input(obj, in(sample,channel),channel);
         % calculate p vector
         p = obj.G*obj.x(:,channel) + obj.H*obj.U;
         % find the currents
         [obj.v, I] = solve_nonlinear_func(obj,p, obj.K);
         % output sample
         s = (obj.D*obj.x(:,channel) + obj.E*obj.U + obj.F*I);
        [out(sample,channel), obj] = store_output(obj,s,channel);
         % update model state
         obj.x(:,channel) = obj.A*obj.x(:,channel) + obj.B*obj.U +
                obj.C*I;
      end
   end
end
```

Building DK method framework DKmodel class – load and store

```
function obj = load_input(obj,in,channel)
   % loads the input signal sample given by in into
       inputs vector U
   % this method should be override in subclass but if the model
       has just one signal input declared as the first voltage
       source, than the parent class method can be used
   obj.U(1,1) = in;
end

function [out, obj] = store_output(obj,output,channel)
   % return real output signal sample
   % this method should be override in subclass
   % when the output should be stored for another processing
   out = output(1);
end
```

Building DK method framework DKmodel class – solve_nonlinear

```
function [v, I] = solve_nonlinear_func(obj, p, K)
   % Newton-Raphson method
   iter = obj.maxIter; % load max iterations
   v0 = obj.v; % initial guess
   v = v0 + 2*obj.eps;
   while(iter > 0 || any(abs(v-v0) > obj.eps))
      [it, Jt] = nonlinearity(obj,v); % Circuit nonlinearities
      % Form DK method nonlinearity
      e = p + K*it - v; % residual
      J = K*Jt - eye(length(v)); % Jacobian
      v = v0 - J\e; % update unknowns
      v0 = v; % update initial guess
      iter = iter-1; % decrement iterations
   end
   I = nonlinearity(obj,v); % Circuit nonlinearities
end
```

Building DK method framework Nonlinear solvers

- Simple Newton-Raphson method doesn't ensure convergence for all cases – e.g. BJT nonlinearities
  - Damped newton method [Eichas2014]
  - $v_{i+1} = v_i - 2^{-m} J^{-1}(v_i) f(v_i)$
  - Find m to fulfill condition $|v_i - 2^{-m} J^{-1}(v_i) f(v_i)| < |f(v_i)|$

- Research of several methods suitable for real-time audio processing presented in [Holmer2015]
  - Improved initial guess estimation

- Nonlinear solver available from Matlab
  - Fsolve function

```
v = fsolve(@(v)nonlinear_func(v,p,K),v0)
```

  - Wide range of parameters and good cornvergence options
  - Cannot be used when C code is generated from Matlab code

Building DK method framework Solvers - real-time constraints

- How to choose the solver parameters
  - *Eps* condition – precision of the solution
    - Over-precise vs. numerical noise
    - Different unknowns may require different precision
  - *Max iterations* condition
    - Prevents deadlocks of the algorithm
    - Should we continue when max iterations limit has been reached?

Building DK method framework DKmodel class – basic functions

```
classdef DKmodel
   properties
   ...
   end
   methods
       function out = process(obj,in)
       function obj = buildModel(obj, components,
           components_count)
       function [v, I] = solve_nonlinear_func(obj, p, K)
       % OVERRIDE functions
       function [i, J] = nonlinearity(obj,v)
       function obj = load_input(obj,in)
       ...
    end
end
```

Building DK method framework DKmodel class – buildModel I

- Complex function which builds the incidence matrixes from circuit components (defined later)
- Builds the State space model from incidence matrixes
  - x[n] = Ax[n−1] + Bu[n] + Ci(v[n])
  - y[n] = Dx[n−1] + Eu[n] + Fi(v[n])
  - v[n] = Gx[n−1] + Hu[n] + Ki(v[n])
- Finds steady state solution
  - $x = (I - A)^{-1}(Bu - Ci)$
  - Substituted into $v[n] = Gx[n-1] + Hu[n] + Ki(v[n])$ and solved
- Can be used for computing look-up tables

Building DK method framework DKmodel – build state space I

- Incidence matrixes: N_R, G_R, N_X, G_X, N_N, N_U, N_O
- Relation between incidence matrixes and state space model defined in [Holters2011]
  - Based on MNA equation in form $\begin{pmatrix} v \\ i_s \end{pmatrix} = S^{-1}\left(\begin{pmatrix} N_x^T \\ O \end{pmatrix} x + \begin{pmatrix} I \\ O \end{pmatrix} u + \begin{pmatrix} N_n^T \\ O \end{pmatrix} i_n\right)$
  - With conductance matrix $S = \begin{pmatrix} N_r^T G_r N_r + N_x^T G_x N_x & N_u^T \\ N_u & 0 \end{pmatrix}$
  - Output $v = (N_o \quad 0)\begin{pmatrix} v \\ i_s \end{pmatrix} = (N_o \quad 0) S^{-1}\begin{pmatrix} N_x^T \\ O \end{pmatrix} + \begin{pmatrix} I \\ O \end{pmatrix} u + \begin{pmatrix} N_n^T \\ O \end{pmatrix} i_n$
  - Output $y[n] = Dx[n-1] + Eu[n] + Fi(v[n])$
  - $D = (N_o \quad 0) S^{-1}\begin{pmatrix} N_x^T \\ O \end{pmatrix} = (N_o \quad 0) S^{-1}(N_x \quad 0)^T$

Building DK method framework DKmodel – build state space II

- Various nonlinear elements can be used
  - Single port nonlinearity
    - Diodes
    - One nonlinear function $i = f(u)$
    - One line in the N_n matrix
  - Two port nonlinearity
    - Triode, BJT, JFET
    - Two nonlinear functions $i_1 = f_1(u_1, u_2)$, $i_2 = f_2(u_1, u_2)$
    - Two lines in the N_n matrix
  - Multi port nonlinearities: Pentodes
- Operational amplifiers
  - Substituted with voltage controlled voltage source
  - Directly integrated into incidence matrixes
  - Nonlinear model $v_{Out} = v_{EE} + (0.5 \tanh(A(v_+ - v_-)) + 0.5) (v_{CC} - v_{EE})$

Building DK method framework
Components struct – resistor

```
function obj = resistor(name__, nodes__, value__)
    obj.nodes = zeros(2,2); % some components can be two-ports
    obj.name = zeros(1,8);  % the name must have the same length
    l = min(8,length(name__));
    obj.nodes(1,:) = nodes__; % set the nodes
    obj.name(1:l) = name__(1:l); % set the name
    obj.value = value__; % set the value
    obj.type = 'res'; % 3char ID
end
```

Building DK method framework Components struct – capacitor

```
function obj = capacitor(name__, nodes__, value__)
    obj.nodes = zeros(2,2); % some components can be two-ports
    obj.name = zeros(1,8);  % the name must have the same length
    l = min(8,length(name__));
    obj.nodes(1,:) = nodes__; % set the nodes
    obj.name(1:l) = name__(1:l); % set the name
    obj.value = value__; % set the value
    obj.type = 'cap'; % 3char ID
end
```

Building DK method framework Components struct – triode

```
function obj = triode(name__, nodes__, value__)
obj.nodes = zeros(2,2); % some components can be two-ports like
this
    obj.name = zeros(1,8);  % the name must have the same length
    l = min(8,length(name__));
    obj.nodes(1:2,:) = nodes__; % set the nodes - two-port
    obj.name(1:l) = name__(1:l); % set the name
    obj.value = value__; % set the value
    obj.type = 'trd'; % 3char ID
end
```

Building DK method framework Components struct – potmeter

```
function obj = potmeter(name__, nodes__, value__)
obj.nodes = zeros(2,2); % some components can be two-ports like
this
    obj.name = zeros(1,8);  % the name must have the same length
    l = min(8,length(name__));
    obj.nodes(1,:) = nodes__(1:2); % set first resistor
    if(length(nodes__) > 2)
       obj.nodes(2,:) = nodes__(2:3); % set second resistor
    end
    obj.name(1:l) = name__(1:l); % set the name
    obj.value = value__; % set the value
    obj.type = 'pot'; % 3char ID
end
```

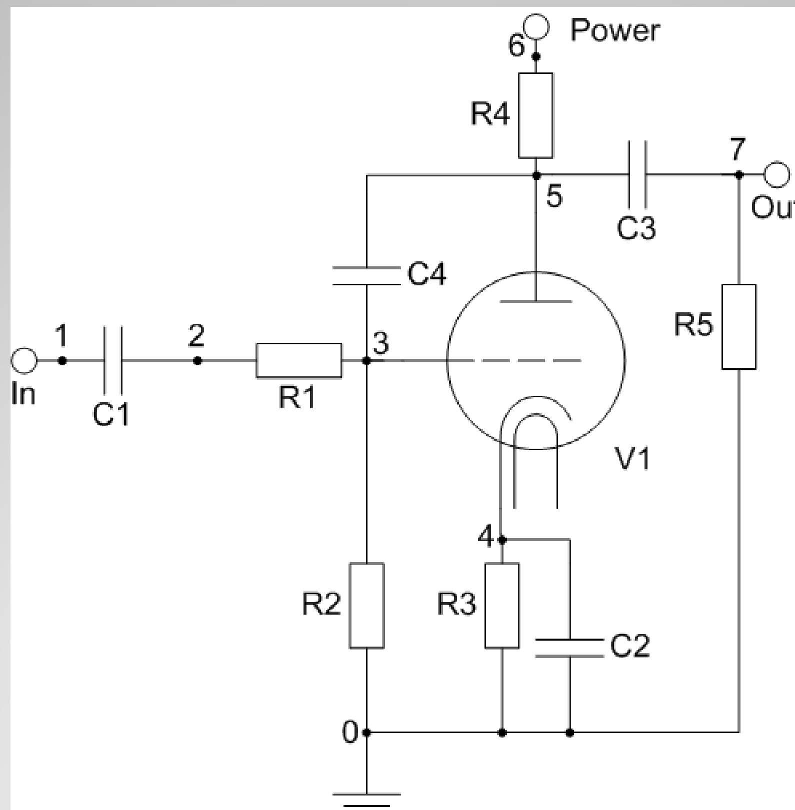
Building DK method framework Components struct – OPA

```
function obj = opa(name__, nodes__, value__)
obj.nodes = zeros(2,2); % some components can be two-ports like
this
    obj.name = zeros(1,8);  % the name must have the same length
    l = min(8,length(name__));
    % inputs
    obj.nodes(1,:) = nodes__(1:2);
    % outputs
    obj.nodes(2,1) = nodes__(3);

    obj.name(1:l) = name__(1:l); % set the name
    obj.value = value__; % set the value
    obj.type = 'opa'; % 3char ID
end
```

Building DK method framework DKmodel class – incidence matrix

```
for i = 1:length(components) % iterate all components
    if(strcmp(components(i).type, 'res')) % this is resistor

    numResistors = numResistors+1;
    if(components(i).nodes(1,1) > 0) % first node
       Nr(numResistors,components(i).nodes(1,1)) = 1;
    end
    if(components(i).nodes(1,2) > 0)% second node
       Nr(numResistors,components(i).nodes(1,2)) = -1;
    end

    Gr(numResistors,numResistors) = 1/components(i).value;

    elseif(strcmp(components(i).type, 'cap')) % this is capacitor
    ...
    end
end
```

Building DK method framework Triode amplifier example



Building DK method framework Triode amplifier example I

```
classdef preampModel < DKmodel
    properties(Constant)
        components_def = [resistor('R1',[2,3],68000),...
                resistor('R2',[3,0],1000000),...
                resistor('R3',[4,0],2700),...
                resistor('R4',[5,6],100000),...
                resistor('R5',[7,0],1000000),...
                capacitor('C1',[1,2],20e-9),...
                capacitor('C2',[4,0],20e-6),...
                capacitor('C3',[5,7],20e-9),...
                capacitor('C4',[3,5],2e-12),...
                inputPort('In',[1,0],0),...
                inputPort('power',[6,0],300),...
                outputPort('Out',[7,0]),...
                triode('v1',[3,4;5,4],0)];
                ...
```

Building DK method framework Triode amplifier example II

```
        ...
        components_count = struct('numResistors',
            5,'numCapacitors', 4,...
            'numInputPorts', 2, 'numOutputPorts', 1,...
            'numNonlinearComponents',2, 'numPotmeters', 0,...
            'numNodes', 7);
end
```

- Component count can be derived from the structure from previous slide
- Explicit definition improves model performance significantly when generating C code from Matlab

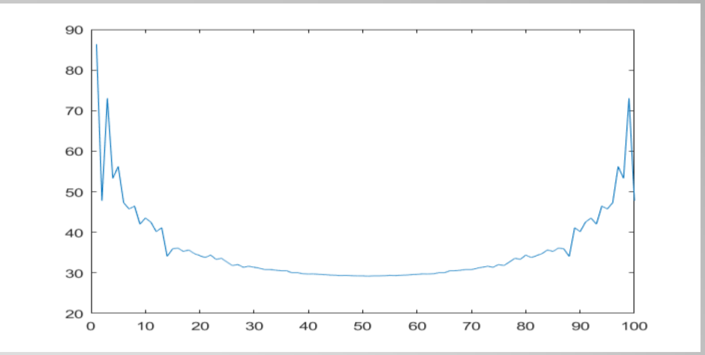
Building DK method framework Triode amplifier example III

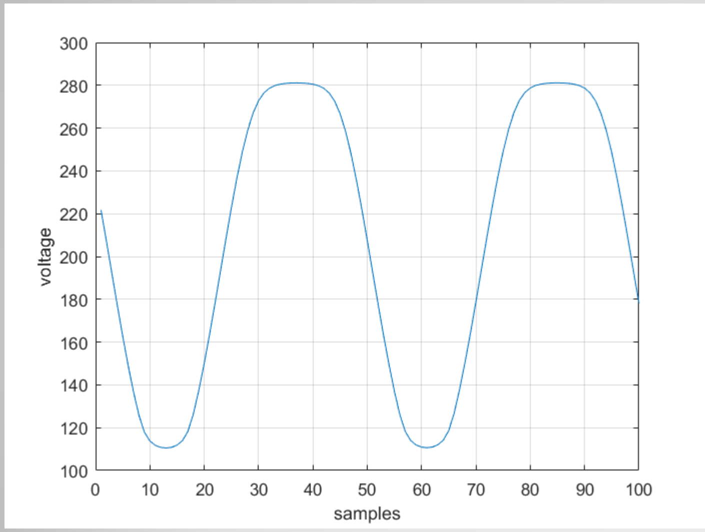
```
methods
   function obj = preampModel(fs)
      obj.T = 1/fs;
      obj = buildModel(obj,
               obj.components_def,obj.components_count);
   end

   function [i, J] = nonlinearity(obj,v)
      i=zeros(2,1);
      J = zeros(2,2);
      [i(1), i(2), J(1,1), J(1,2), J(2,1), J(2,2)] =
               tube_model(obj,v(1), v(2));
   end
end
```

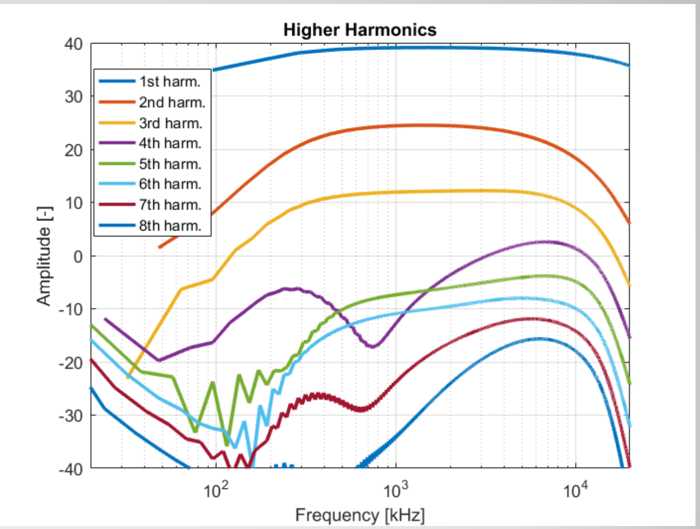
Several tube models available

Building DK method framework Triode amplifier example IV

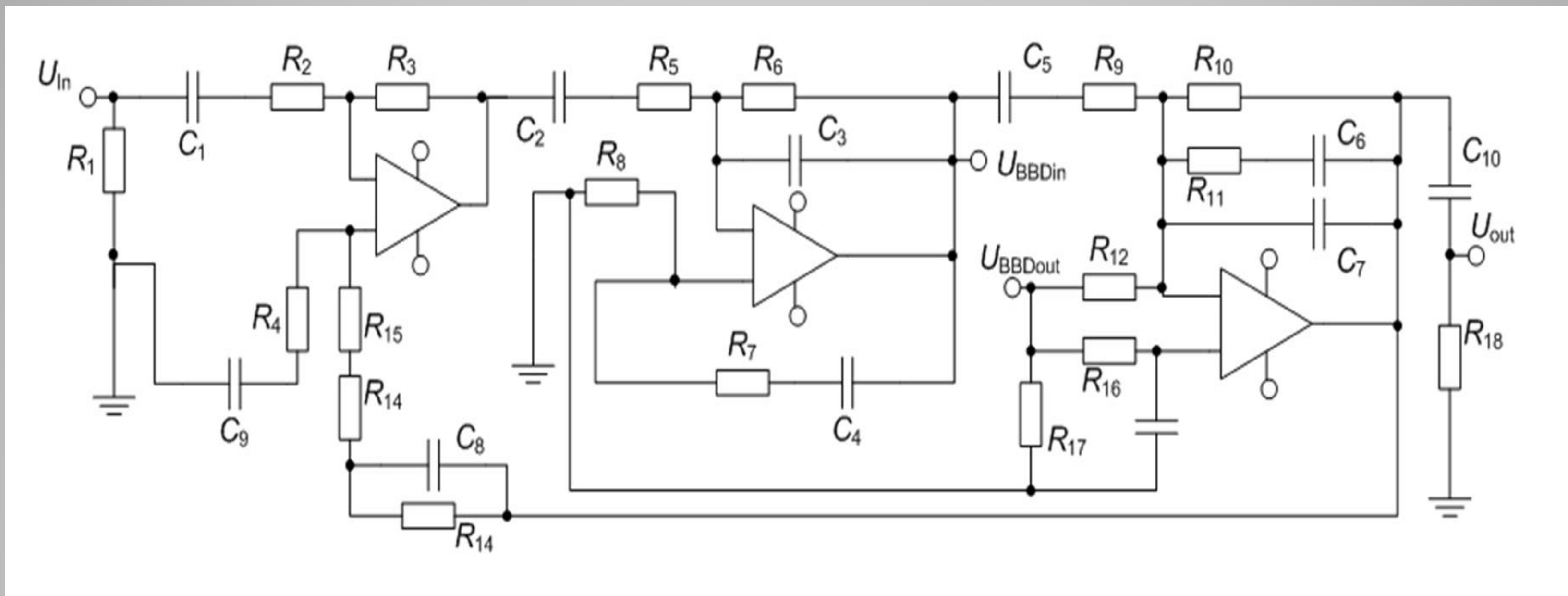
```
fs = 48000;
preamp = preampModel(fs);
input = Amp*sin(...);
output = preamp.process(input’);
plot(output)
```







Building DK method framework Flanger example



Building DK method framework Flanger example II

```
classdef flangerModel < DKmodel
    properties(Constant)
        components_def = [resistor('R1',[1,0],120000),...
                resistor('R2',[2,3],390000),...
                ...
                capacitor('C10',[19,0],5e-6),...
                potmeter('P1',[17,5,18],10e3),...
                inputPort('In',[1,0],0),...
                inputPort('InBBD',[15,0],0),...
                outputPort('Out',[14,0]),...
                outputPort('OutBBD',[8,0]),...
                opa('opa1',[3,5,4],1e8),...
                opa('opa1',[10,7,8],1e8),...
                opa('opa1',[12,19,14],1e8)];
                ...
```

Building DK method framework Flanger example III

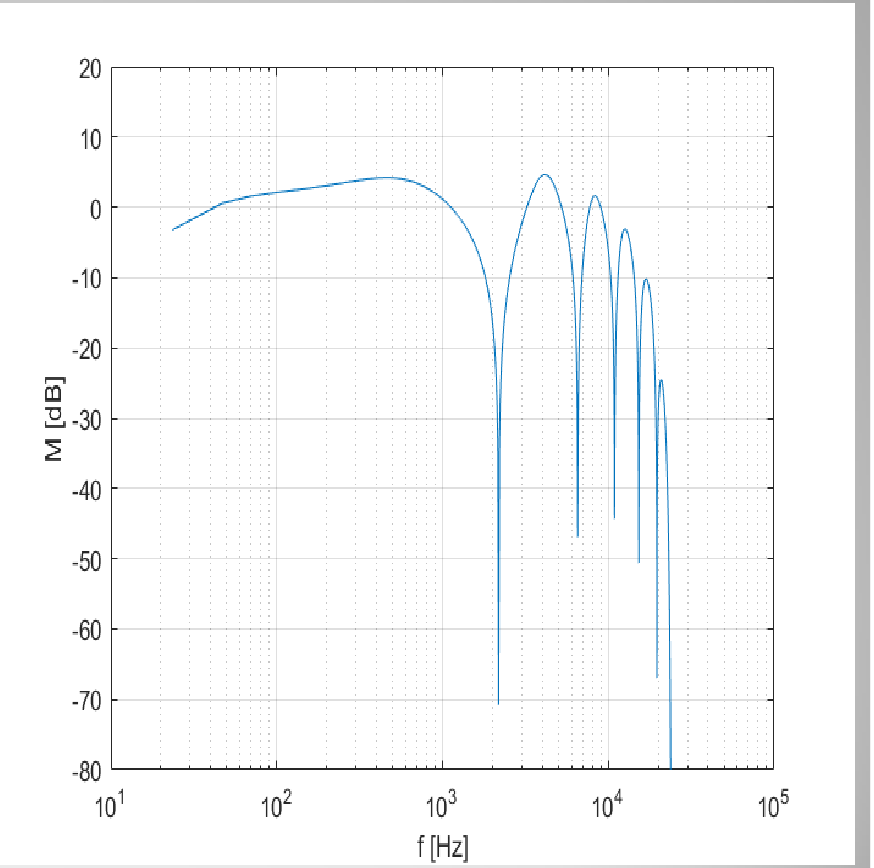
```
            ...
 components_count = struct('numResistors', 15,'numCapacitors',
       10,...
        'numInputPorts', 2, 'numOutputPorts', 2,...
        'numNonlinearComponents',0, 'numPotmeters', 1,...
        'numNodes', 19, 'numOPAs',3);
end
Properties (Access = private)
   delay_line = zeros(2,1024);
   pointer = 1;
end
```

Building DK method framework Flanger example IV– load & store

```
function obj = load_input(obj,in,channel)
   obj.U(1) = in;
   obj.U(2) = obj.delay_line(channel,obj.pointer);
end

function [out, obj] = store_output(obj,output,channel)
   out = output(1);
   obj.delay_line(channel, obj.pointer) = output(2);
   obj.pointer = obj.pointer+1;
   if(obj.pointer >= obj.current_delay)
        obj.pointer = 1;
   end
end
```

Building DK method framework Flanger example V

```
fs = 48000;
flanger = flangerModel(fs);
flanger.setColor(0.7);
intput = [1;zeros(1,4095)];
output = flanger.process(input);
[H,w] = freqz(output,1,1024,fs);
semilogx(w,20*log10(abs(H)));
ylabel('M [dB]')
xlabel('f [Hz]')
grid on
ylim([-80 20])
```



Building DK method framework
Building VST plug-in

- Matlab Audio System Toolbox
- Defines audioPlugin class
- Generation of VST plug-in from Matlab audioPlugin class
- Few lines of code
  - Implement constructor of plugin
  - Implement process function
  - Implement reset function

Building DK method framework using audioPlugin class I

```
classdef preampPlugin < audioPlugin
    properties
       Gain = 50;
       Output = 50;
    end
    properties (Access=private)
        preamp
    end
    properties (Constant)
        PluginInterface = audioPluginInterface(...
            audioPluginParameter('Gain',...
            'DisplayName','Gain',...
            'Mapping',{'lin',0,100},'Label','%'),...
             audioPluginParameter('Output',...
            'DisplayName','Output',...
            'Mapping',{'lin',0,100},'Label','%'))
    end
```

Building DK method framework using audioPlugin class II

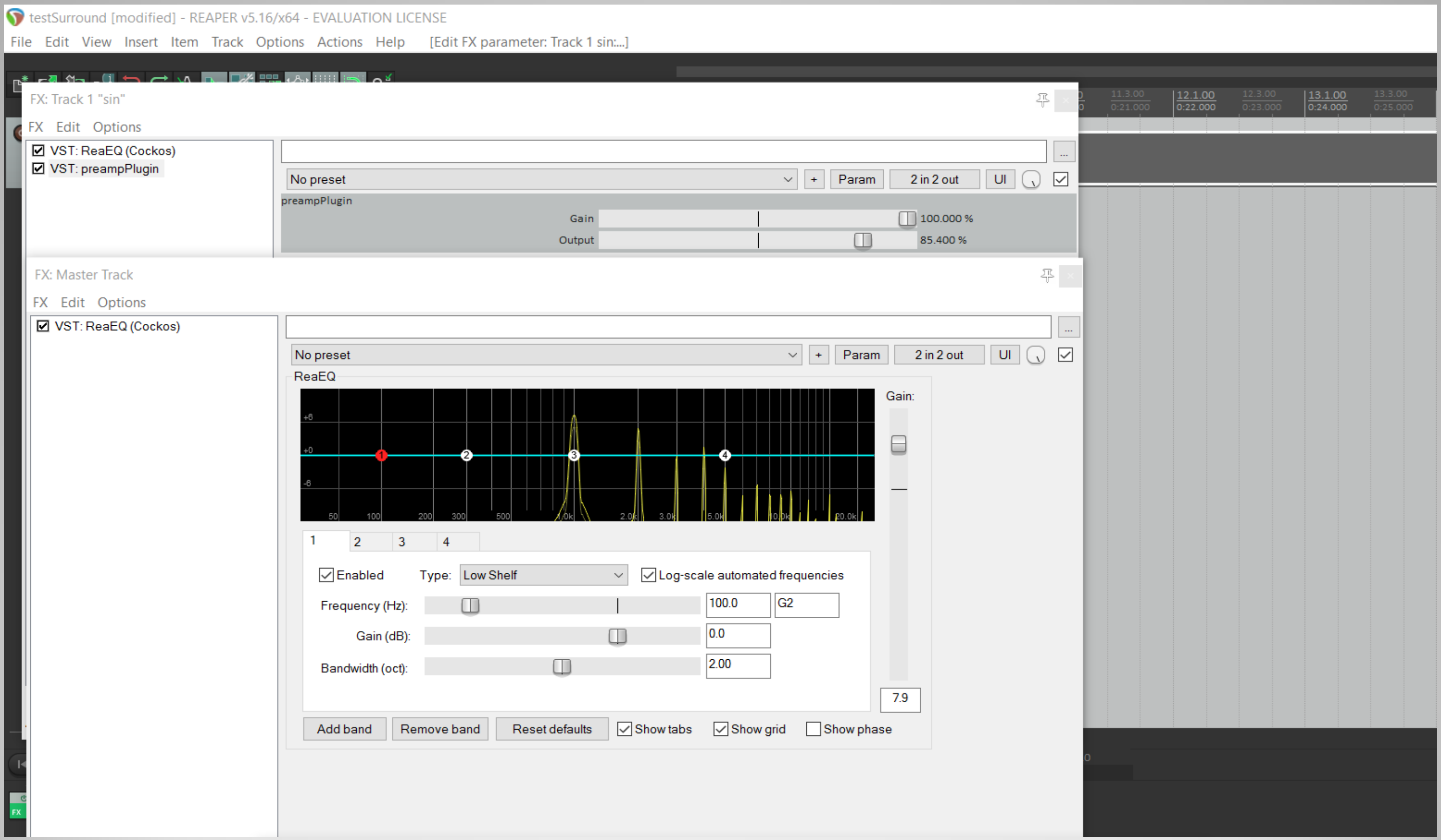
```
methods
    function plugin = preampPlugin()
        plugin.preamp = preampModel(plugin.getSampleRate);
    end
    function out = process(plugin, in)
        [sig, plugin] = plugin.preamp.process(in*plugin.Gain*0.01);
        out = sig*plugin.Output*0.01;
    end
    function reset(plugin)
        plugin.preamp = preampModel(plugin.getSampleRate);
    end
    function set.Gain(plugin, val)
            plugin.Gain = val;
    end
    function set.Output(plugin, val)
            plugin.Output = val;
    end
end
```

Building DK method framework Generate plug-in file

```
validateAudioPlugin preampPlugin;
generateAudioPlugin -win32 preampPlugin;
```

- Generate VST2 plug-in preampPlugin.dll
- Parameters available in plugin default graphical user interface
- Matlab coder is used
- Possible to generate C source files for the internal function using codegen

Building DK method framework Plug-in loaded in DAW



Summary

- Brief overview of methods used for virtual analog modeling
  - Black box modeling
  - Wave digital filters
  - State space model
- Nodal DK method framework
- Examples of triode amplifier and flanger effect simulation shown
- Audio system toolbox introduced

References:

- [Benois2013] P. R. Benois, Simulation Framework for Analog Audio Circuits based on Nodal DK Method, Master Thesis
- [Dempwolf2010] K. Dempwolf, M. Holters and U. Zölzer, Discretization of Parametric Analog Circuits for Real-time Simulations, Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10), Graz, Austria , September 6-10, 2010[Eichas2014]
- F. Eichas, M. Fink, M. Holters, U. Zölzer, Physical Modeling of the MXR Phase 90 Guitar Effect Pedal, Proc. of the 17th Int. Conference on Digital Audio Effects (DAFx-14), Erlangen, Germany, September 1-5, 2014
- [Holmer2015] B. Holmes, M. van Walstijn, Improving the Robustness of the Iterative Solver in State-Space Modelling of Guitar Distortion Circuitry, Proc. of the 18th Int. Conference on Digital Audio Effects (DAFx-15), Trondheim, Norway, Nov 30 - Dec 3, 2015
- [Holters2011] M. Holters, 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, Proc. of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23, 2011
- [Novak2010] A. Novak, L. Simon, F. Kadlec, P. Lotton (2010), "Nonlinear system identification using exponential swept-sine signal", Instrumentation and Measurement, IEEE Transactions on. Vol. 59(8), pp. 2220-2229.
- [Paiva2012] R. C. D. de Paiva, J. Pakarinen, and V. Välimäki, "Reduced-Complexity Modeling of High-Order Nonlinear Audio Systems Using Swept-Sine and Principal Component Analysis", Audio Engineering Society Conference: 45th International Conference: Applications of Time-Frequency Processing in Audio, 2012
- [Smith2015] J. O. Smith, K. J. Werner, Recent Progress in Wave Digital Audio Effects WDF Software Overview and Demo, DAFx15, Trondheim, avalilable at https://www.ntnu.edu/documents/1001201110/0/DAFx-2015-jos-keynote2part2.pdf/b6dbef08-f552-4d8b-8a29-eeaee0b14b99
- [Yeh2012] D.T. Yeh, Automated Physical Modeling of Nonlinear Audio Circuits for Real-Time Audio Effects -- Part II: BJT and Vacuum Tube Examples , IEEE Trans on Audio, Speech, and Language Processing, 20(4), 2012, pp. 1207 -- 1216
- [Zölzer2011] U. Zölzer, DAFX - Digital Audio Effects (Second Edition), ISBN: 978-0-470-66599-2, John Wiley & Sons, 2011

Thank you for listening!

- If you are interested to cooperate or use the framework, don’t hesitate and contact me.
- https://github.com/jardamacak/NodalDKFramework
- Email: jarda.macak@seznam.cz