Last revised : 17.03.2013

Matlab scripts for interfacing rational function-based models with circuit solvers using discrete convolution

Y-parameters, Z-parameters, S-parameters, transfer functions

Bjørn Gustavsen
SINTEF Energy Research
N-7465 Trondheim
NORWAY

e-mail: bjorn.gustavsen@sintef.no

Download site (**interfacing_with_circuit_solvers.zip**):
http://www.energy.sintef.no/Produkt/VECTFIT/index.asp

Table of Contents

1. INTRODUCTION

The Matrix Fitting Toolbox (downloadable from this site) can be used for producing rational-function-based models for Y-parameters, Z-parameters and S-parameters. In addition, vectfit3.m can be used for generating models of general transfer functions. The returned models are given both on pole-residue form and on the equivalent state-space form, see Table I. The state-space model has its **A**-matrix diagonal and **B** sparse.

Table I. Parameter type and Models.

	Pole-residue model	State-space model
Y-parameters	$\mathbf{Y}(s)=\sum_{m=1}^{N}\frac{\mathbf{R}_m}{s-a_m}+\mathbf{R}_0+s\mathbf{E}$	$\mathbf{Y}(s)=\mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{R}_0+s\mathbf{E}$
Z-parameters	$\mathbf{Z}(s)=\sum_{m=1}^{N}\frac{\mathbf{R}_m}{s-a_m}+\mathbf{R}_0+s\mathbf{E}$	$\mathbf{Z}(s)=\mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{R}_0+s\mathbf{E}$
S-parameters	$\mathbf{S}(s)=\sum_{m=1}^{N}\frac{\mathbf{R}_m}{s-a_m}+\mathbf{R}_0$	$\mathbf{S}(s)=\mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{R}_0$
Transfer functions	$\mathbf{H}(s)=\sum_{m=1}^{N}\frac{\mathbf{R}_m}{s-a_m}+\mathbf{R}_0+s\mathbf{E}$	$\mathbf{Y}(s)=\mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{R}_0+s\mathbf{E}$

It is often desirable to integrate the model in a time domain simulation environment. Ref. [1] shows an efficient procedure for achieving this using discrete convolution. This download allows the user to replicate the simulation results for one of the examples in [1] where a two-port model was created from a small electrical circuit, for Y-, Z- and S-parameters. In addition, the use of a transfer function for calculating voltage transfer between two nodes was demonstrated. The procedure is applicable to models of any order and number of ports.

We recall that Y-, Z- and S-parameter models interact with the circuit over the connecting ports. Therefore, passivity must in general be enforced. This is not the case for pure transfer function models which have no feedback to the circuit.

Notation: bold, uppercase: matrix; bold, lowercase: vector; non-bold: scalar.

- In Chap. 2, we derive the discrete form of the convolution between an input and output, with a fixed time step length and trapezoidal integration.
- In Chap. 3, we introduce as an example a small circuit from which we extract two-port parameters.
- In Chap. 4 we utilize the two-port parameters in a circuit level simulation based on time domain nodal analysis. We show how to achieve this by formulating Norton equivalents for the alternartive parameter types, and we show the implementation in Matlab script.

Restrictions of use:

- If the code is used in a scientific or commercial work, then reference should me made to [1].

2. DISCRETE CONVOLUTION

In the continuous time domain, the state equation associated with the state-space model reads

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{1a}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{E}\dot{\mathbf{u}} \tag{1b}$$

Applying trapezoidal integration with a fixed time step length Δt gives the discrete, recursive formula (2) at the *k*th time step [1],

$$\mathbf{x}_k = \boldsymbol{\alpha}\mathbf{x}_{k-1} + \boldsymbol{\lambda}\mathbf{B}\mathbf{u}_{k-1} \tag{2a}$$
$$\mathbf{y}_k = \tilde{\mathbf{C}}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k \tag{2b}$$

where

$$\boldsymbol{\alpha} = (\mathbf{I} - \mathbf{A}\frac{\Delta t}{2})^{-1}(\mathbf{I} + \mathbf{A}\frac{\Delta t}{2}) \tag{3a}$$
$$\mathbf{G} = (\mathbf{D} + \mathbf{C}\boldsymbol{\mu}\mathbf{B}) \tag{3b}$$
$$\tilde{\mathbf{C}} = \mathbf{C}(\boldsymbol{\alpha}\boldsymbol{\mu} + \boldsymbol{\lambda}) \tag{3c}$$

with

$$\boldsymbol{\lambda} = \boldsymbol{\mu} = (\mathbf{I} - \mathbf{A}\frac{\Delta t}{2})^{-1}\frac{\Delta t}{2} \tag{3d}$$

In the case of a non-zero **E**, we get the augmented matrices [1]

$$\tilde{\mathbf{C}} \rightarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{I} \end{bmatrix}, \quad \boldsymbol{\alpha} \rightarrow \begin{bmatrix} \boldsymbol{\alpha} & 0 \\ 0 & -\mathbf{I} \end{bmatrix},$$
$$\mathbf{B} \rightarrow \begin{bmatrix} \mathbf{B} \\ -\dfrac{4\mathbf{E}}{\Delta t} \end{bmatrix}, \mathbf{G} \rightarrow \mathbf{G} + \frac{2\mathbf{E}}{\Delta t} \tag{4}$$

Equation (2) forms the corner stone in the subsequent implementations, whatever the type of parameters the model is representing.

In the case that the model includes complex poles, one can choose between a complex or a real-only model. We shall assume that the model is kept in its complex form, which results in that <u>**A** is diagonal</u> and <u>**B** is a sparse matrix</u> which has a single one in each row. In this case, one can reduce the model size by deleting from **A** the second part of all complex pairs and the ditto columns in **C** and rows in **B**, provided that one multiplies the rows in **B** associated with the first pole parts, with a factor two, and that one retains only the real part of $\tilde{\mathbf{C}}\mathbf{x}_k$ in (2b). This operation has been conveniently implemented in the function **reducecmplx**.

reducecmplx

```
[SER1]=reducecmplx(SER);

Processing of model before time domain simulation, for improved speed.
The Model size is reduced by throwing out the second part of all complex pairs
and multiplying rows in B associated with first part by factor 2.
07.03.2013. B. Gustavsen
```

3. TWO-PORT EXAMPLE

As an example we use the electrical circuit in Fig. 1. Our objective is to create a model with respect to ports 1 and 2, and to employ that model in a time domain simulation.

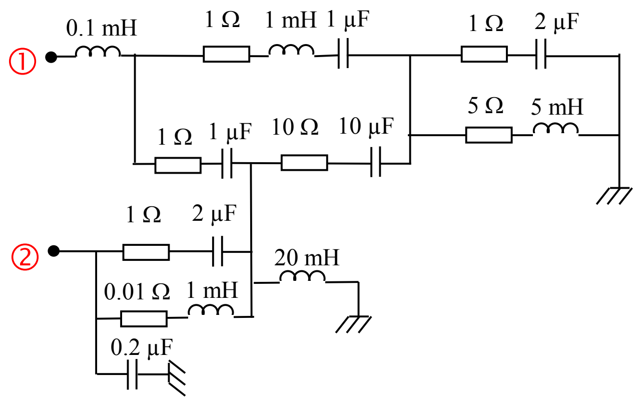

Fig. 1. Electrical circuit [1].

Using nodal analysis, we establish the frequency domain 5×5 nodal admittance matrix **Y**(*s*) with respect to the five nodes in Fig. 2. The procedure amounts to establishing the nodal admittance stamp for each branch between any to nodes and between any node and ground, and to add this stamp to the 5×5 **Y** in the correct position.

- For a branch with admittance y between node i and ground, we add the contribution y to element $\mathbf{Y}_{ii}$.
- For a branch between nodes i and j, we add the contribution y to element $\mathbf{Y}_{ii}$ and $\mathbf{Y}_{jj}$, and the contribution $-y$ to elements $\mathbf{Y}_{ij}$ and $\mathbf{Y}_{ji}$.

In the code, we have conveniently introduced a function **add_branch** for adding in the branch contributions.

add_branch

```
Y=add_branch(n1,n2,y,Y);

The function adds the contribution fom an admittance y between nodes
n1 and n2 to the admittance matrix Y. Node number 0 denotes ground.
07.03.2013. B. Gustavsen
```

Fig. 2 shows the adopted branch numbering (blue) along with the node numbering (red)

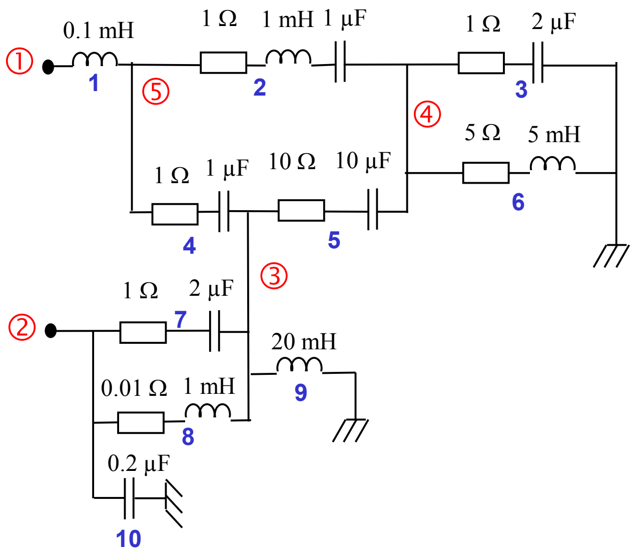

Fig. 2. Electrical circuit with node numbering (red) and branch numbering (blue).

Variables for the circuit component values associated with Fig. 2 are first introduced:

```
%Component values:
         L1=0.1e-3;
R2=1;    L2=1e-3;    C2=1e-6;
R3=1;                C3=2e-6;
R4=1;                C4=1e-6;
R5=10;               C5=10e-6;
R6=5;    L6=5e-3;
R7=1;                C7=2e-6;
R8=1e-2; L8=1e-3;
         L9=20e-3;
                     C10=0.2e-6;
```

In the frequency loop, the branch admittances at each frequency s_k are calculated:

```
%Branch admittances
  y1 =1/(sk*L1);
  y2 =1/( R2+sk*L2+1/(sk*C2) );
  y3 =1/( R3+1/(sk*C3) );
  y4 =1/( R4+1/(sk*C4));
  y5 =1/( R5+1/(sk*C5));
  y6 =1/( R6+sk*L6) ;
  y7 =1/( R7+1/(sk*C7) );
  y8 =1/( R8+sk*L8 );
  y9 =1/(sk*L9)
  y10=sk*C10;
```

…and added to the global admittance matrix, Y:

```
%Adding contribution from branch admittances:
  Y=add_branch(5,4,y2,Y);
  Y=add_branch(4,0,y3,Y);
  Y=add_branch(5,3,y4,Y);
  Y=add_branch(3,4,y5,Y);
  Y=add_branch(4,0,y6,Y);
  Y=add_branch(2,3,y7,Y);
  Y=add_branch(2,3,y8,Y);
  Y=add_branch(3,0,y9,Y);
  Y=add_branch(2,0,y10,Y);
  Y=add_branch(1,5,y1,Y);
```

The admittance matrix is reduced with respect to terminals 1 and 2. To see how this is done we introduce the partitioning

$$\begin{bmatrix} \mathbf{i}_a \\ \mathbf{i}_b \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{aa} & \mathbf{Y}_{ab} \\ \mathbf{Y}_{ba} & \mathbf{Y}_{bb} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_a \\ \mathbf{v}_b \end{bmatrix} \tag{5}$$

Where "a" denotes modes 1-2 and "b" denoted nodes 3-5. Since the current injection to nodes 3-5 is zero, we have $\mathbf{i}_b$=0. From the second row in (1) we express $\mathbf{v}_b$ by $\mathbf{v}_a$

$$\mathbf{v}_b = -\mathbf{Y}_{bb}^{-1}\mathbf{Y}_{ba}\mathbf{v}_a \tag{6}$$

Inserting (2) into the first row in (1) gives the final result:

$$\mathbf{i}_a = (\mathbf{Y}_{aa} - \mathbf{Y}_{ab}\mathbf{Y}_{bb}^{-1}\mathbf{Y}_{ba})\mathbf{v}_a = \mathbf{Y}_{red}\mathbf{v}_a \tag{7}$$

```
%Eliminating nodes 3-5:
  Yred=Y(1:2,1:2)-Y(1:2,3:5)*Y(3:5,3:5)^(-1)*Y(3:5,1:2);
```

The resulting 2×2 $\mathbf{Y}_{red}$ is stored in 3-D array bigY:

```
  bigY(:,:,k)=Yred;
```

In the remainder, we will skip subscript "*red*" when denoting the 2×2 admittance matrix. The admittance matrix is converted into the other parameters sets (Z-, S-) in the pertaining examples.

4. MODEL INTERFACES

4.1 Circuit level simulation

We wish to simulate the example in Fig. 3 where a unit step voltage behind a 5 Ω resistor is connected to node #1 with node #2 being open. We will calculate the current flowing into node #1 and the voltage response on node #2.

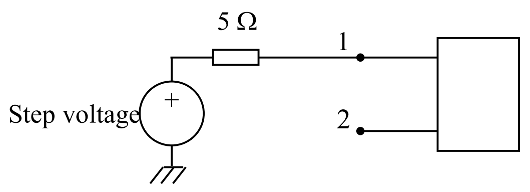

Fig. 3. Simulation example.

To do this, we replace the voltage source behind the resistor with its Norton equivalent, and we represent the two-port model with a Norton equivalent, see Fig. 4. The way the Norton equivalent is obtained from the rational model depends on which type of parameters it represents.

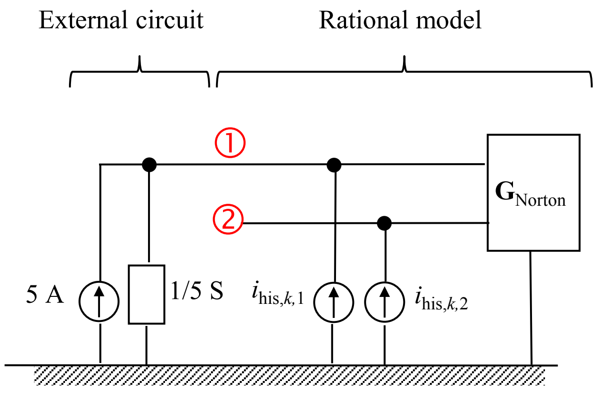

Fig. 4. Circuit simulation via Norton equivalents.

The time step loop for Fig. 4 can in its simplest form be realized as shown by the pseudo-code in Fig. 5, where subscripts "ind" and "his" denotes "independent" and "history", respectively:

```
$\mathbf{Z}_{\text{Global}} = \mathbf{G}_{Global}^{-1}$
for k=1: Nt
    $\mathbf{i}_{k,\text{Global}} = \mathbf{i}_{\text{ind,k, Global}} + \mathbf{i}_{\text{his,k, Global}}$
    $\mathbf{v}_{\text{k, Global}} = \mathbf{Z}_{\text{Global}}\, \mathbf{i}_{k,\text{ Global}}$
    $\mathbf{i}_{\text{his,k}} = f(\mathbf{v}_{k-1})$
end
```

Fig. 5. Pseudo-code for time step loop.

- "Global" refers to all nodes in the system, which in general is larger than the number of ports in the system.
- $\mathbf{G}_{\text{Global}}$ is obtained by assembling the contribution from all branches and Norton equivalents in the circuit. Its dimension is equal to the number of nodes.
- $\mathbf{i}_{\text{ind,k, Global}}$ and $\mathbf{i}_{\text{his,k, Global}}$ are current source vectors with as many entries as there are nodes in the system.
- The function $f(:)$ for updating the history current source depends on which parameter type the model is representing (Y-, Z, or S).

4.2 Y-parameters

Y-parameters define the relation between voltage and current at the terminals of the device, when voltage is input,

$$\mathbf{i} = \mathbf{Y}\mathbf{v} \tag{8}$$

Model extraction

Using routine **VFdriver** from the Matrix Fitting Toolbox we calculate a rational approximation for **Y**(s), held in structure SERY:

```
opts.N=10; %Model order
opts.poletype='logcmplx';
opts.asymp=3; %Model includes R0 and E
opts.stable=1;
opts.NE=0;
poles=[];
[SERY,rmserr,bigYfit,opts2]=VFdriver(bigY,s,poles,opts);
```

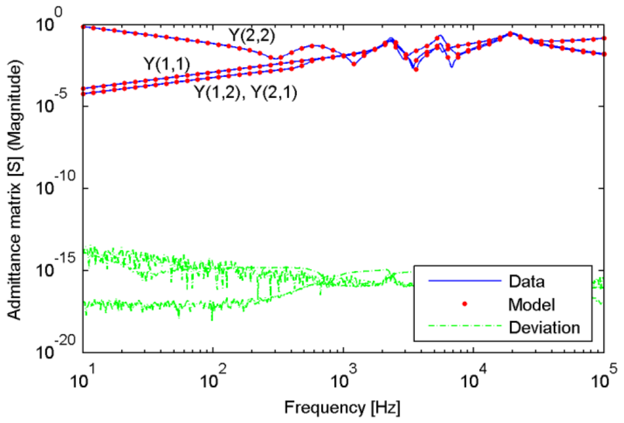

Fig. 6. Rational approximation of Y(s).

The resulting model is now in structure "SERY", both on pole-residue form and on state-space form.

Since we are in this example "fitting to machine precision", the model error is virtually zero and so we do not care about passivity enforcement. (In the general case you would have to process the result (SERY) using function **RPdriver**, also found in the Matrix Fitting toolbox).

Time domain simulation

In the case of Y-parameters, voltage **v** is input and current **i** is output. This gives for (2),

$$\mathbf{x}_k = \boldsymbol{\alpha}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{v}_{k-1} \tag{9a}$$

$$\mathbf{i}_k = \tilde{\mathbf{C}}\mathbf{x}_k + \mathbf{G}\mathbf{v}_k \tag{9b}$$

and we recognize that (9b) represents a Norton equivalent (Fig. 7) with $\mathbf{G}_{\mathrm{Norton}}$=**G** and $\mathbf{i}_{\mathrm{his},k}$=$-\tilde{\mathbf{C}}\mathbf{x}_k$.

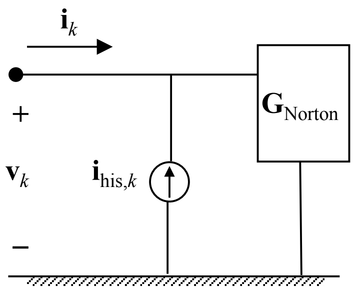

Fig. 7. Norton equivalent.

We will us a time step length of 10 μs and simulate until 5000 μs.

```
%++++++++++++++++++++++++++++++++
%**Simulation parameters:
Dt=1e-5; time=(0:Dt:5e-3); Nt=length(time);
```

The conductance "matrix" of the external circuit:

```
%++++++++++++++++++++++++++++++++
%**External circuit:
Rsour=5;
gsour=1/Rsour;
```

First, we reduce the model size:

```
%Throw out second part of complex poles:
[SER]=reducecmplx(SERY);
A=full(diag(SER.A)); %A is now a column vector
B=SER.B
C=SER.C
D=SER.D;
E=SER.E;
```

Next, we calculate the parameters for the convolution, and the Norton conductance:

```
%++++++++++++++++++++++++++++++++
%**Initialization:
N=length(A);
Nc=length(D);
X=zeros(N,1); %state vector

a=A.';
alfa=(1+a*Dt/2)./(1-a*Dt/2);
lamb=(Dt/2)./(1-a*Dt/2);
mu=lamb;
GY=D+real(C*diag(lamb)*B); %Norton conductance for model
```

```
dum=alfa.*lamb +mu;
Ctilde=zeros(Nc,N);
for row=1:Nc
  Ctilde(row,:)=C(row,:).*dum; %Ctilde
end

%Contribution from E-matrix, if present:
if max(max(abs(E)))>0
  alfa=[alfa -ones(1,Nc)];
  Ctilde=[Ctilde eye(Nc)];
  B=[B;-4*E./Dt];
  GY=GY+2*E./Dt;
  X=[X;zeros(Nc,1)];
end
```

Assembling global conductance matrix, calculating impedance matrix:

```
GG=zeros(2,2);                %Global conductance matrix
GG(1:2,1:2)=GG(1:2,1:2)+GY;  %Contribution from model
GG(1,1)=GG(1,1)+gsour;       %Contribution from external circuit

ZZ=GG^(-1);    %Impedance matrix
Ihis=[0 0].'; %Global history current source
```

Time step loop:

```
t=0;
for k=1:Nt
  t=t+Dt;

  Iind=[gsour 0].'; %Global vector of independent current sources 1
  Itot=Iind+Ihis; %Total current
  V=ZZ*Itot;      %Node voltages

  %Updating history current source of model
  X=alfa.'.*X +B*V;
  Ihis=-real(Ctilde*X);

  pgbY(1,k)=gsour-gsour*V(1);  %Saving current flowing into node #1
  pgbY(2,k)=V(2);;             %Saving voltage at node #2

  bigV(k)=V(1); %to be used as excitation in voltage transfer computation
end %for k=1:Nt
```

From the above we see that the model is included in the time step loop using only two lines of code. Note that in an actual implementation, we can utilize the sparsity of **B** to achieve an even faster code.

The simulated waveforms as stored in pgbY are presented in Chapter 4.6.

4.3 Z-Parameters

Z-parameters define the relation between voltage and current at the terminals of the device, when current is input,

$$\mathbf{v} = \mathbf{Z}\mathbf{i} \tag{10}$$

Model extraction

First, we compute the impedance data from the admittance data in Chap. 3.:

```
for k=1:Ns
  Y=squeeze(bigY(:,:,k));
  bigZ(:,:,k)=Y^(-1);
end
```

Using routine **VFdriver** from the Matrix Fitting Toolbox we calculate a rational approximation for **Z**(s), held in structure SERZ:

```
opts.N=10; %Model order
poles=[];
opts.poletype='logcmplx';
opts.asymp=3; %Model includes R0 and E
[SERZ,rmserr,bigZfit,opts2]=VFdriver(bigZ,s,poles,opts);
```

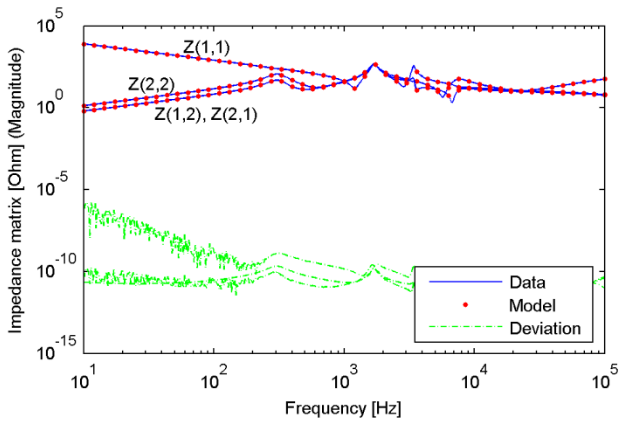

Fig. 8. Rational approximation of Z(s).

Time domain simulation

In the case of Z-parameters, current **i** is input and voltage **v** is output. This gives for (2),

$$\mathbf{x}_k = \boldsymbol{\alpha}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{i}_{k-1} \tag{11a}$$

$$\mathbf{v}_k = \tilde{\mathbf{C}}\mathbf{x}_k + \mathbf{G}\mathbf{i}_k \tag{11b}$$

and we recognize that (11b) represents a Thevenin equivalent with $\mathbf{Z}_{\text{Norton}}$=**G** and $\mathbf{v}_{\text{his,k}} = -\tilde{\mathbf{C}}\mathbf{x}_k$. The Thevenin equivalent is conveniently converted into a Norton equivalent as shown in Fig. 9, where $\mathbf{G}_{\text{Norton}} = \mathbf{Z}_{\text{Norton}}^{-1}$ and $\mathbf{i}_{\text{his,k}} = \mathbf{G}_{\text{Norton}}\,\mathbf{v}_{\text{his,k}}$.

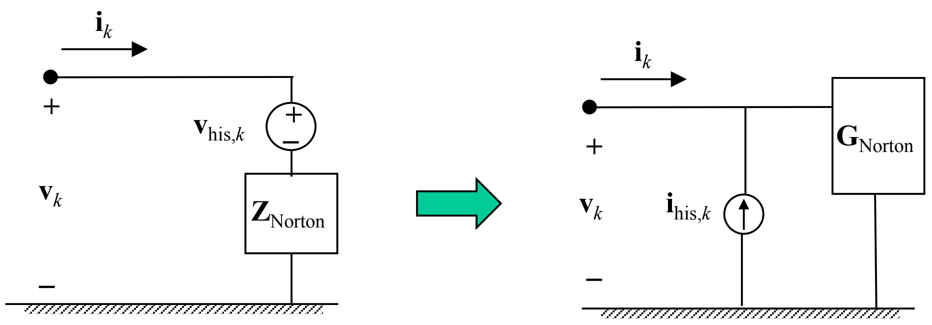
$\mathbf{i}_k$
+
$\mathbf{v}_{\text{his},k}$
$\mathbf{v}_k$
$\mathbf{Z}_{\text{Norton}}$
–
$\mathbf{i}_k$
$\mathbf{G}_{\text{Norton}}$
+
$\mathbf{v}_k$
$\mathbf{i}_{\text{his},k}$
–

Fig. 9. Conversion of Thevenin equivalent into Norton equivalent.

```
%+++++++++++++++++++++++++++++++
%**Simulation parameters:
Dt=1e-5; time=(0:Dt:5e-3); Nt=length(time);

%+++++++++++++++++++++++++++++++
%**External circuit:
Rsour=5;
gsour=1/Rsour;

%+++++++++++++++++++++++++++++++
%**rational model:

[SERZ]=reducecmplx(SERZ);
A=full(diag(SERZ.A));
B=SERZ.B;
C=SERZ.C;
D=SERZ.D;
E=SERZ.E;

%+++++++++++++++++++++++++++++++
%**Initialization:
N=length(A);
Nc=length(D);
X=zeros(N,1);

a=A.';
alfa=(1+a*Dt/2)./(1-a*Dt/2);
lamb=(Dt/2)./(1-a*Dt/2);
mu=lamb;

Zthevenin=zeros(Nc);
Zthevenin=real(C*diag(lamb)*B);
Zthevenin=Zthevenin+D;

dum=alfa.*lamb +mu;
Ctilde=zeros(Nc,N);
for row=1:Nc
  Ctilde(row,:)=C(row,:).*dum; %Ctilde
end
%Contribution from E-matrix, if present:
if max(max(abs(E)))>0
  alfa=[alfa -ones(1,Nc)];
  Ctilde=[Ctilde eye(Nc)];
  B=[B;-4*E./Dt];
  Zthevenin=Zthevenin+2*E./Dt;
  X=[X;zeros(Nc,1)];
end
```

```
Gnorton=Znorton^(-1); %Converting Zthevenin into Gnorton

%Assembling global conductance matrix, calculating impedance matrix:
GG=GY;
GG(1,1)=GG(1,1)+gsour;
ZZ=GG^(-1);   %Impedance matrix
Ihis=[0 0].'; %Global history current source

%Time step loop:
t=0;
V=[0 0].';
for k=1:Nt
  t=t+Dt;

  Iind=[gsour 0].'; %Global vector of independent current sources
  Itot=Iind+Ihis;     %Total current
  V=ZZ*Itot;          %Node voltages

  %Updating history currnt source:
  I=(GY*V-Ihis); %current flowing into terminals
  X=alfa.'.*X +B*I;
  Ihis=GY*real(Ctilde*X);

  pgbZ(1,k)=I(1);  %saving current flowing into node #1
  pgbZ(2,k)=V(2);              %Saving voltage at node #2

end %for k=1:Nt
```

4.4 S-Parameters

Definitions

S-parameters define the relation between incoming power waves **a** and reflected power waves **b** at the ports of a device being terminated by reference impedances. With the reference impedances held in the (diagonal) entries of a diagonal matrix $\mathbf{Z}_0$, we can write

$$\mathbf{b} = \mathbf{S}\,\mathbf{a} \tag{12}$$

with

$$\mathbf{a} = \sqrt{\mathbf{Z}_0}^{\,-1}\,\mathbf{v}_i \tag{13a}$$

$$\mathbf{b} = \sqrt{\mathbf{Z}_0}^{\,-1}\,\mathbf{v}_r \tag{13b}$$

$\mathbf{v}_i$ and $\mathbf{v}_r$ are incident and reflected voltage waves (see Fig. 10). **a** and **b** are the associated power waves.

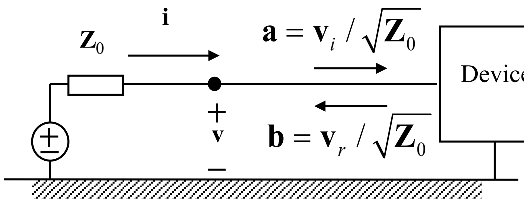

Fig. 10. Incident and reflected waves at the ports of the device.

Model extraction

In the example, we compute **S** from **Y** using the relation

$$\mathbf{S} = (\mathbf{I} + \sqrt{\mathbf{Z}_0}\,\mathbf{Y}\sqrt{\mathbf{Z}_0})^{-1}(\mathbf{I} - \sqrt{\mathbf{Z}_0}\,\mathbf{Y}\sqrt{\mathbf{Z}_0}) \tag{14}$$

We use for the reference impedances,

$$\mathbf{Z} = \begin{bmatrix} 100 & 0 \\ 0 & 200 \end{bmatrix} \Omega$$

```
R=diag([100 200]); %Reference impedances
sqR=sqrt(R);       %Square-root of reference impedances
I=eye(Nc); %Nc=2
for k=1:Ns
  Y=bigY(:,:,k);
  bigS(:,:,k)=(I+sqR*Y*sqR)^(-1)*(I-sqR*Y*sqR);
end
```

Using routine **VFdriver** from the Matrix Fitting Toolbox we calculate a rational approximation for **Z**(s), held in structure SERS:

```
opts.N=11; %Model order
opts.poletype='logcmplx';
poles=[];
opts.stable=0;
opts.asymp=2; %Model includes R0
[SERS,rmserr,bigSfit,opts2]=VFdriver(bigS,s,poles,opts);
```

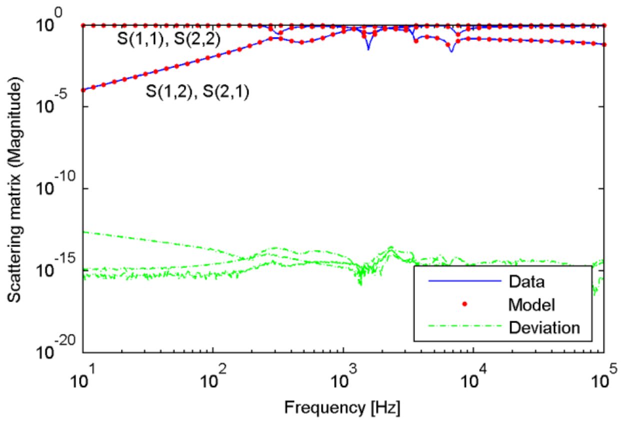

Fig. 11. Rational approximation of S(s).

Time domain simulation

In the simulation, **a** is input and **b** is output, which gives for the convolution (2)

$$\mathbf{x}_k = \boldsymbol{\alpha}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{a}_{k-1} \tag{15a}$$

$$\mathbf{b}_k = \tilde{\mathbf{C}}\mathbf{x}_k + \mathbf{G}\mathbf{a}_k \tag{15b}$$

We wish to obtain the Norton equivalent in Fig. 12.

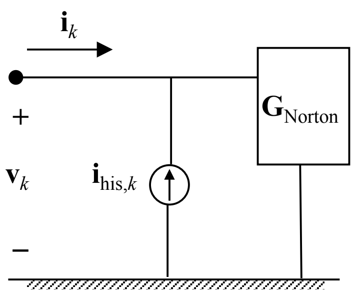

Fig. 12. Norton equivalent for interface of S-parameter model.

As shown in [1], the conductance matrix $\mathbf{G}_{\text{Norton}}$ and history current surce $\mathbf{i}_{his,k}$ are obtained as

$$\mathbf{G}_{\textbf{Norton}} = \sqrt{\mathbf{Z}_0}^{-1}[(\mathbf{I}-\mathbf{G})(\mathbf{I}+\mathbf{G})^{-1}]\sqrt{\mathbf{Z}_0}^{-1} \tag{16a}$$

$$\mathbf{i}_{\text{his},k} = 2\sqrt{\mathbf{Z}_0}^{-1}(\mathbf{I}+\mathbf{G})^{-1}\tilde{\mathbf{C}}\mathbf{x}_k = \quad {}_k \tag{16b}$$

The procedure for calculating $\mathbf{x}_k$ in (16b) is as follows:

1. Calculate $\mathbf{a}_{k-1}$ from $\mathbf{v}_{k-1}$ and $\mathbf{x}_{k-1}$

$$\mathbf{a}_{k-1} = (\mathbf{I}+\mathbf{G})^{-1}(\sqrt{\mathbf{Z}_0}^{-1}\mathbf{v}_{k-1} - \tilde{\mathbf{C}}\mathbf{x}_{k-1}) \tag{17}$$

2. Calculate $\mathbf{x}_k$ using (15a)
3. Calculate $\mathbf{i}_{his,k}$ using (16b)

In actual code,

```
%**Simulation parameters:
Dt=1e-5; time=(0:Dt:5e-3); Nt=length(time);

%++++++++++++++++++++++++++++++++
%**External circuit:
Rsour=5;
gsour=1/Rsour;

%++++++++++++++++++++++++++++++++
%**Rational model:
[SERS]=reducecmplx(SERS);
A=full(diag((SERS.A)));
B=SERS.B
C=SERS.C
D=SERS.D;

%++++++++++++++++++++++++++++++++
%**Initialization:
N=length(A);
Nc=length(D);
X=zeros(N,1);
II=eye(Nc);

a=A.';
alfa=(1+a*Dt/2)./(1-a*Dt/2);
lamb=(Dt/2)./(1-a*Dt/2);
mu=lamb;

G=D+real(C*diag(lamb)*B);
GS= R^(-0.5)*(II-G)*(II+G)^(-1)*R^(-0.5);

dum=alfa.*lamb +mu;
Ctilde=zeros(Nc,N);
for row=1:Nc
  Ctilde(row,:)=C(row,:).*dum;
end
GAMMA=2*(R^(-0.5))*(II+G)^(-1)*Ctilde;

%++++++++++++++++++++++++++++++++
%Assembling global conductance matrix, calculate impedance matrix:
GG=GS;
GG(1,1)=GG(1,1)+gsour;
```

```
ZZ=GG^(-1);
Ihis=[0 0].';

%++++++++++++++++++++++++++++++++
%Time domain simulation:
t=0;
for k=1:Nt
  t=t+Dt;

  Iind=[gsour 0].';
  I=Iind+Ihis;
  V=ZZ*I; %Node voltages

  %Calculating incoming power wave
  a=(II+G)^(-1)*( R^(-0.5)*V -real(Ctilde*X) );

  %Updating history current source
  X=alfa.'.*X +B*a;
  Ihis=real(GAMMA*X);

  pgbS(1,k)=gsour-gsour*V(1); %Saving currentvflowing into node #1
  pgbS(2,k)=V(2);             %Saving voltage at node #2

end %for k=1:Nt
```

4.5 Transfer Functions

The general transfer function **H** defines the relation between an input **u** and an output **y**,

$$\mathbf{y} = \mathbf{H}\mathbf{u} \tag{18}$$

Model extraction

In the example, we calculate the voltage on the open port 2 using the voltage on port 1 as input. From the condition i_2=0 we obtain the voltage transfer function is calculated as

$$H_{21} = -Y_{22}^{-1} Y_{21} \tag{19}$$

```
% Voltage transfer function from node 1 to node 2
for k=1:Ns
  Y=squeeze(bigY(:,:,k));
  bigH(:,:,k)=-(Y(2,2))^(-1)*Y(2,1);
end
```

Since the transfer function is in this example a scalar, we can in this case use routine **VFdriver** from the Matrix Fitting Toolbox for calculating a rational approximation for $H(s)$, held in structure SERS:

```
opts.N=11; %Model order
opts.poletype='logcmplx';
poles=[];
opts.stable=1
[SERH,rmserr,bigHfit,opts2]=VFdriver(bigH,s,poles,opts);
```

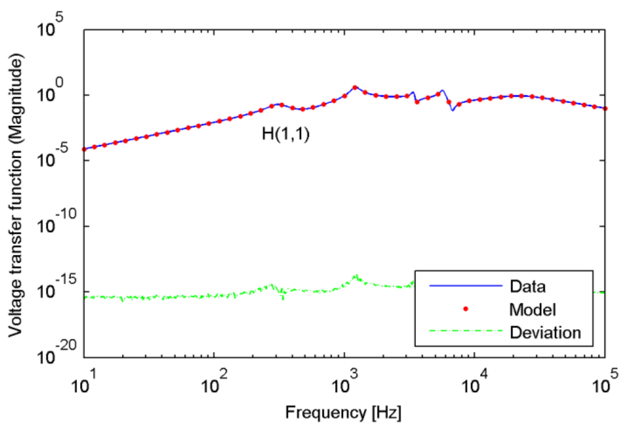

Fig. 13. Rational approximation of $H_{21}(s)$.

<u>**Note**</u>: You can only use **VFdriver** when **H** is symmetrical. When this is not the case, you will have to fit **H** via **vectfit3.m** and assemble the total state-space model yourself.

Time domain simulation

With input **u** and output **y**, we get

$$\mathbf{x}_k = \alpha \mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \tag{20a}$$

$$\mathbf{y}_k = \tilde{\mathbf{C}}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k \tag{20b}$$

Since the model is not interacting with the circuit, we do not need the Norton equivalent.

```
%**Simulation parameters:
Dt=1e-5; time=(0:Dt:5e-3); Nt=length(time);

%++++++++++++++++++++++++++++++++
%**Initialization:
N=length(A);
Nc=length(D);
X=zeros(N,1);

a=A.';
alfa=(1+a*Dt/2)./(1-a*Dt/2);
lamb=(Dt/2)./(1-a*Dt/2);
mu=lamb;

dum=alfa.*lamb +mu;
Ctilde=zeros(1,N);
for row=1:1
  Ctilde(row,:)=C(row,:).*dum;
end

GH=real(C*diag(lamb)*B);
GH=GH+D;

%++++++++++++++++++++++++++++++++
%Time domain simulation:
t=0;
oldVmode=zeros(N,1);
```

```
for k=1:Nt
  u=bigV(k); %Voltage at node #1 calculated using Y-parameter model
  t=t+Dt;
  Vmode=B*u;
  X=alfa.'.*X +oldVmode;
  oldVmode=Vmode;
  y = GH*u +real(Ctilde*X);
  pgbH(1,k)=0;
  pgbH(2,k)=y; %The voltage on port #2
end
```

4.6 Comparison of time domain waveforms

We will now compare the obtained results with that obtained by a conventional time domain circuit simulator (PSCAD/EMTDC v4.2) where the circuit elements in Fig. 1 are represented individually. The circuit simulator uses trapezoidal integration with the same time step length as in the previous Matlab simulations.

```
%PSCAD simulation of RLC circuit
load -ASCII C.m;
t_pscad=C(:,1).';
v_pscad=1e-3*C(:,2).';
i_pscad=1e-3*C(:,3).';

%==============================
% CURRENT at port #1:
%==============================
figure(101),
h1=plot(1e3*time,pgbY(1,:),'b-');hold on
h2=plot(1e3*time,pgbZ(1,:),'r--');
h3=plot(1e3*time,pgbS(1,:),'g-.'); hold off
legend([h1 h2(1) h3(1)],'Y-parameters','Z-parameters','S-parameters',1);
xlabel('Time [ms]');
ylabel('Current [A]')
title('Simulation result')
hold off
```

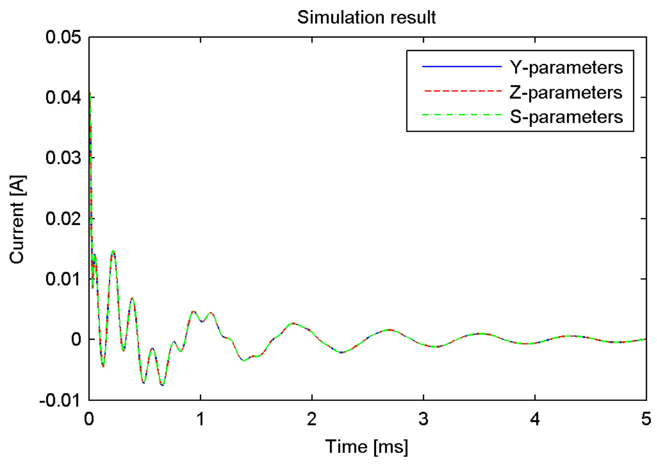

Fig. 14. Simulated current i_1 at port #1.

```
figure(102),
h1=plot(1e3*time,pgbY(1,:)-i_pscad,'b-');hold on
h2=plot(1e3*time,pgbZ(1,:)-i_pscad,'r--');
```

```
h3=plot(1e3*time,pgbS(1,:)-i_pscad,'g-.'); hold off
title('Deviation from PSCAD simulation result')
legend([h1 h2(1) h3(1)],'Y-parameters','Z-parameters','S-parameters',1);
xlabel('Time [ms]');
ylabel('Current [A]')
hold off
```

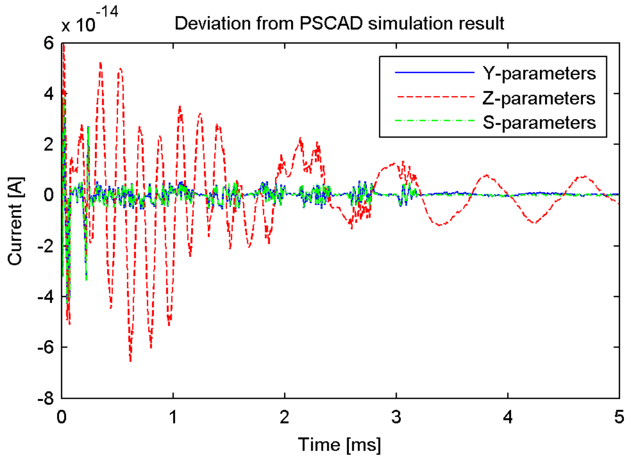

Fig. 15. Deviation from result by conventional circuit simulation.

```
%==============================
%  VOLTAGE at port #2:
%==============================
figure(103),
h1=plot(1e3*time,pgbY(2,:),'b-');hold on
h2=plot(1e3*time,pgbZ(2,:),'r--');
h3=plot(1e3*time,pgbS(2,:),'g-.');
h4=plot(1e3*time,pgbH(2,:),'k:','linewidth',1.0); hold off
title('Simulation result')
legend([h1(1) h2(1) h3(1) h4(1)],'Y-parameters','Z-parameters','S-
parameters','Transfer function',1);
xlabel('Time [ms]');
ylabel('Voltage [V]')
```

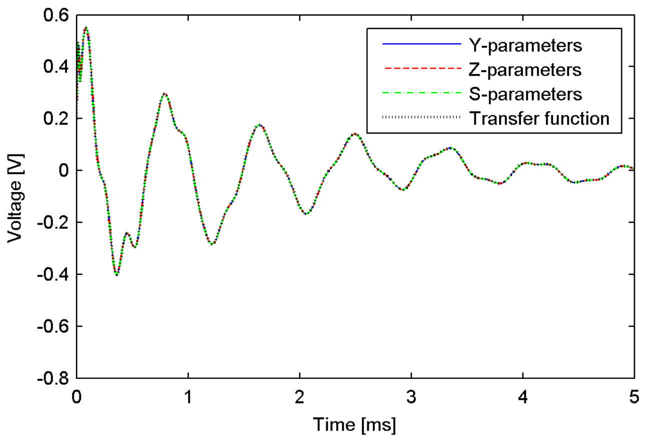

Fig. 16. Simulated voltage v_2 at port #2.

```
figure(104),
h1=plot(1e3*time,pgbY(2,:)-v_pscad,'b-');hold on
h2=plot(1e3*time,pgbZ(2,:)-v_pscad,'r--');
```

```
h3=plot(1e3*time,pgbS(2,:)-v_pscad,'g-.');
h4=plot(1e3*time,pgbH(2,:)-v_pscad,'k:'),hold off
title('Deviation from PSCAD simulation result')
legend([h1(1) h2(1) h3(1) h4(1)],'Y-parameters','Z-parameters','S-
parameters','Transfer function',1);
xlabel('Time [ms]');
ylabel('Voltage [V]')
```

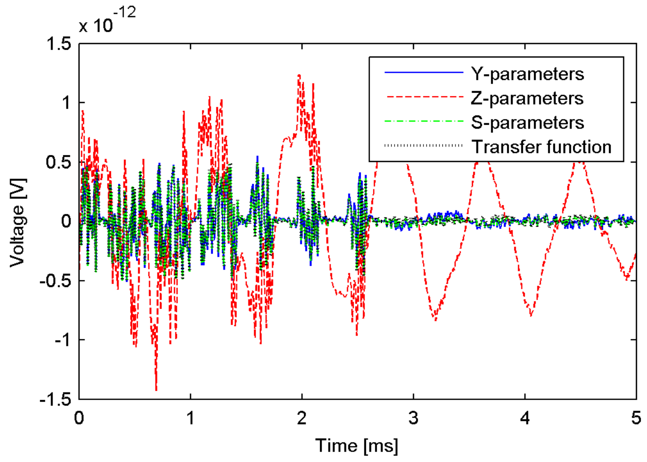

Fig. 17. Deviation from result by conventional circuit simulation.

We note that the difference between the result by the detailed PSCAD simulation and that by the two-port models is extremely small. Although we have created highly accurate rational model approximations, we could still expect significant differences due to the discretization of the convolution integrals. However, in this case we have with all approaches adopted the same integration technique (trapezoidal integration) with the same time step length, leading to consistent discretization errors. That way, we have "proved" the correctness of the implementations.

5. THE PACKAGE

The package (**simulation_1.zip**) is downloadable from
http://www.energy.sintef.no/Produkt/VECTFIT/index.asp

Documentation

`model_interface_for_simulation.pdf`	This document
`TPWRD_Inclusion_of_rational_model_2013.`	Reference [1]

Matlab routines:

`circuit_small_example.m`	Runs all simulation examples

Auxiliary routines:

`add_branch.m`	Add admittance stamp to **Y**
`reducecmplx.m`	Compacting of state-space model for faster simulation

Other files:

`C.m`	Simulation result by PSCAD (used in Chap. 4.5)

6. REFERENCES

[1] B. Gustavsen and H.M.J. De Silva, "Inclusion of rational models in and electromagnetic transients program – Y-parameters, Z-parameters, S-parameters, transfer functions", *IEEE Trans. Power Delivery*, 2013.

7. ACKNOWLEDGEMENT

The development of these templates was supported by the KPN project "Electromagnetic transients in future power systems" (ref. 207160/E20), financed by

- Research Council of Norway (Norway),
- Dong Energy (Denmark),
- EdF (France),
- EirGrid (Ireland),
- Hafslund Nett (Norway),
- National Grid (UK),
- Nexans (Norway),
- RTE (France),
- Siemens WP (Denmark),
- Statnett (Norway),
- Statkraft (Norway),
- Vestas WS (Denmark)