

Last revised: 14.06.2026

NONLINEAR VECTOR FITTING TOOLBOX

for rational modeling of systems defined by input/output responses

in the frequency domain

User's GUIDE and REFERENCE

version 1.0 for Matlab®

Abner Ramirez, Bjørn Gustavsen
CINVESTAV-Guadalajara, MEXICO
SINTEF Energy Research, N-7465 Trondheim NORWAY

e-mails: abner.ramirez@cinvestav.mx
bjorn.gustavsen@sintef.no

Table of Contents

1.	INTRODUCTION.....	3
2.	THE PACKAGE.....	4
2.1	Program Files.....	4
2.2	Instalment.....	5
3.	FUNCTION CALL	5
3.1	NL_VF.m	5
3.2	Other functions.....	7
4.	USER'S GUIDE.....	7
4.1	Hints and advice	7
4.2	Example 1: Boost-converter circuit.....	7
4.2.1	Data case.....	8
4.2.2	Running the example.....	9
4.3	Example 2: EMTP circuit.....	12
4.3.1	Data case.....	12
4.3.2	Running the example.....	15
4.4	Example 3: Three-phase photovoltaic system and nonlinear load.....	16
4.4.1	Data case.....	16
4.4.2	Running the example.....	18
5.	REFERENCE FOR COMPUTATIONAL APPROACH	19
5.1	Pole-residue modeling by NL_VF.....	19
5.1.1	General	19
5.1.2	Standard NL_VF.....	20
5.1.3	Relaxed NL_VF.....	21
5.1.4	Expansion into State Space model.....	21
6.	REFERENCES.....	22
7.	ACKNOWLEDGEMENT.....	Error! Bookmark not defined.

1. INTRODUCTION

This manual describes the Matlab routine useful for rational multi-port modeling of systems given by (1.1), having available the input/output responses available either in the time domain $\mathbf{X}(t)/\mathbf{Y}(t)$, e.g., obtained from an EMT-type simulation tool, or in the frequency domain $\mathbf{X}(s)/\mathbf{Y}(s)$, e.g., obtained from a frequency domain model. If the input/output responses are available in time domain, they are first converted to frequency domain and then fed into the routine. Although the components/systems may involve nonlinear components, such as switching devices, nonlinear reactors, among others, on the identified rational model (1.1) will be applicable around the specified operating point. The output is a rational model on pole-residue form (1.2) and a corresponding state space model (1.3), both with stable poles.

$$\mathbf{Y}(s) = \mathbf{H}(s)\mathbf{X}(s) \quad (1.1)$$

$$\mathbf{H}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s-p_n} + \mathbf{D} \quad (1.2)$$

$$\mathbf{H}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \quad (1.3)$$

- **NL_VF.m** identifies models (1.2) and (1.3) using the pole relocating Vector Fitting technique (VF) [1] formulated with both input $\mathbf{X}(s)$ and output $\mathbf{Y}(s)$ in the frequency domain [4] and related to $\mathbf{H}(s)$ via (1.1) If input/output variables come from a time domain solution method, they have to be transformed within a preliminary step to the frequency domain. The transfer function $\mathbf{H}(s)$ is fitted by using a common pole set. The implementation includes relaxation of the nontriviality constraint [2], [5] (improved convergence). A fast implementation of the pole identification step [3] (reduced memory requirements and computation time) is left for the next version of **NL_VF.m**.
- After obtaining the fitted $\mathbf{H}(s)$ via **NL_VF.m**, the model perturbation routine **RPdriver.m**, so that it becomes passive, and the one to export the rational model into the ATP simulation environment, **netgen_ATP.m**, can be applied, as in classical VF.

An overview of the procedure is shown in Fig. 1.1.

The program has been tested on Matlab R2025a. All timing results are with a desktop computer running under Windows 11 with an Intel(R) Core(TM) Ultra 7, 2.2 GHz, 32 GB RAM.

Download site (**nonlinear_fitting_toolbox_1.zip**):

<https://www.sintef.no/en/software/vector-fitting/>

Restrictions of use:

- Embedding any of (or parts from) the routines of the Nonlinear Fitting Toolbox in commercial software, or software requiring licensing, is **strictly prohibited**. This applies to all routines, see Section 2.1.
- If the code is used in a scientific work, then reference should be made as follows:
 - `NL_VF.m`: References [4], [5]

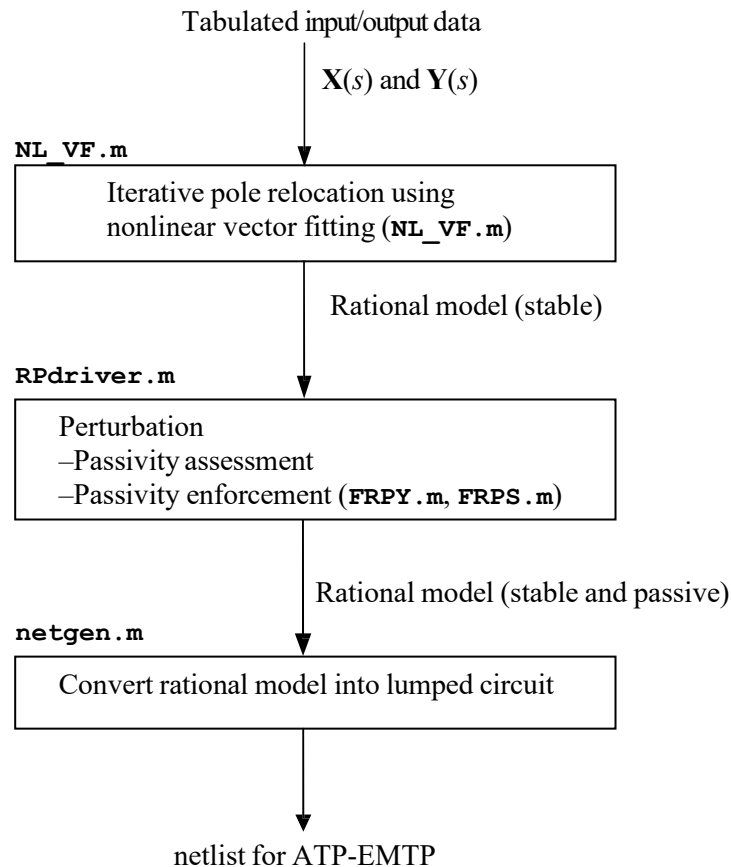


Fig. 1.1 Program overview

2. THE PACKAGE

2.1 Program Files

The package (`Nonlinear_Fitting_Toolbox_1.zip`) consists of the following files:

Documentation:

`user_manual_Nonlinear_Vector_Fitting_Toolbox.pdf` This document

Matlab routines:

`NL_VF.m` Routine for rational fitting

Other files:

<code>ex1_BC.m</code>	Tutorial example (boost-converter circuit)
<code>ex2_circuit_emtp.m</code>	Tutorial example (circuit taken from EMTP)
<code>ex3_PV_nonlinear.m</code>	Tutorial example (system involving photovoltaic farm)

Reference material (technical papers):

<code>VF.pdf</code>	[1] Standard Vector Fitting (VF)
<code>VFrelaxed.pdf</code>	[2] VF with relaxation
<code>VFfast.pdf</code>	[3] VF with fast implementation
<code>NL_VF.pdf</code>	[4] Nonlinear VF, classical implementation
<code>NL_VF_relaxed.pdf</code>	[5] Nonlinear VF, relaxed version

2.2 Instalment

- Place all files in a common directory, e.g.
`c:\user\nonlinfit`
- Include the directory in the Matlab search path,
`>>addpath c:\user\nonlinfit`

3. FUNCTION CALL

3.1 *NL_VF.m*

The following function call will generate a pole-residue model for a data set (s , $\mathbf{X}(s)$, $\mathbf{Y}(s)$) with n ports and N_s frequency samples.

```
[SER,poles,rmserr,fit,opts]=VFdriver(X,Y,s,poles,weight,opts)
```

Input:

X,Y :	(Nc,Ns)	Input and output matrix holding the X- and Y-samples
s :	(1,Ns)	vector holding the frequency samples, $s = j\omega$ [rad/sec].
poles :	(1,N)	vector holding the initial poles (manual specification of initial poles). Use opts for automated specification.
weight :	(1,Ns)	Common weighting for all vector elements, or
weight :	(Nc,Ns)	Individual weighting for vector elements

opts is an optional structure that can be used for overriding defaults settings, and for requesting plots. (Example: `opts.N=30; opts.poletype='lincmplx'`).

Parameter	Purpose/Description	Default
N polestype nu	Automated generation of initial poles, taken as complex conjugate pairs that are distributed over the frequency band. Specify linear or logarithmic distribution, or a mix of the two. (<u>Note</u> : To invoke this option, specify the input array "poles" to be empty (poles=[])). Fitting order (integer) Allowed values: 'lincmplx','logcplx','linlogcplx' Ratio between real and imaginary part (pole=-nu*beta ± j*beta)	- 'lincmplx' 0.001
weight	Array (Nc,Nc,Ns) containing user-defined weight for sample H(i,j,k) in least squares problem.	[] 1
asympt	Control type of rational model =1 --> D=0, E=0 =2 --> D~0, E=0	2

stable	Control handling of unstable poles =1 --> Will enforce stable poles by flipping any unstables into the left half plane	1
relaxed	=1 --> Will use VF with "relaxed" non-triviality constraint. (faster convergence)	1
plot	=1 --> magnitude plot of fitting result	1
logx	=1 --> plot using logarithmic freq. axis	0
logy	=1 --> plot using logarithmic y-axis	1
errplot	=1 --> include deviation (error) in plot	1
phaseplot	=1 --> additional plot of phase angles	1
cplx_ss	=1 --> complex state space model (A,B,C,D,E) with diagonal A. =0 --> real state space model with block-diagonal A.	1

Output:

SER is a structure with the model on pole-residue form. For a model with n ports and N residue matrices, the dimensions are

SER.poles: (N,1)
SER.R: (n,n,N) (residue matrices)
SER.D: (n,n)

The returned **SER** also holds matrices **A**, **B**, **C** for the associated state space model.

SER.A: (nN,nN)
SER.B: (nN,n)
SER.C: (n,nN)

rmserr : the resulting RMS-error of the fitting

fit: (**Nc**,**Ns**) matrix holding the **Y**-samples given by the product $\mathbf{H}(s)\mathbf{X}(s)$

Note: Since $\mathbf{H}(s)$ is not known beforehand, as in classical VF, the output **fit** corresponds to the **Y**-samples given by the product $\mathbf{H}(s)\mathbf{X}(s)$.

3.2 Other functions

The **RPdriver.m** (passivity enforcement) and **netgen_ATP.m** (exports the rational model into a data file for ATP) functions can be called after the **NL_VF.m** is utilized. Details on the use of those functions can be seen in the Matrix Fitting Toolbox user's guide.

4. USER'S GUIDE

4.1 Hints and advice

NL_VF.m

- **poletype**. This parameter is used for automated generation of an initial pole set. The choice of this parameter depends on the nature of the frequency response to be fitted.

If the poles (resonances) appear to be

- linearly distributed in frequency, choose '**lincmplx**';
- logarithmically distributed in frequency, choose '**logcmplx**';

Sometimes (for instance when fitting a transformer response), the poles appear to be logarithmically distributed (and real) at low frequencies and linearly distributed (and complex) at high frequencies. In such cases, it may be best to choose '**linlogcmplx**';

Inaccurate fitting result

If an accurate fitting result cannot be obtained (**NL_VF.m**) no matter what order you try, then there is probably something wrong with your frequency response. A rational function in the frequency domain has a real and imaginary part, which are related in a “special way”. This means that not all functions are fittable; they have to be “physical”. So, the first requirement is that the frequency input/output responses (s , $\mathbf{X}(s)$, $\mathbf{Y}(s)$) are not corrupted in some way. The default setting is to enforce stable poles (**opts.stable**=1). Try to allow unstable poles (**opts.stable**=0) and see if the problem goes away.

4.2 Example 1: Boost-converter circuit

File: **ex1_BC.m**

In this small example the usage of **NL_VF.m** with frequency domain input variables is demonstrated.

4.2.1 Data case

Consider the boost converter circuit in Fig. 4.2.1 with parameters listed in Table 4.1 (quantities given in units $[\Omega]$, $[H]$, $[F]$). For this case study, the input and output are assumed to be the DC voltage source $V_{dc}(s)$ and voltage at capacitor terminals $V_c(s)$, respectively. The solution variables are obtained by a numerical Laplace transform (NLT) algorithm [6] programmed in Matlab®. The NLT provides both FD variable vectors, $V_{dc}(s)$ and $V_c(s)$, loaded into the `ex1_BC.m` file, as shown below Table 4.1.

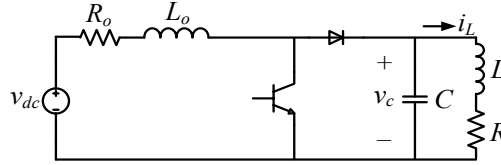


Fig. 4.2.1 Boost converter circuit

Table 4.1. Parameters of boost converter circuit

v_{dc}	1,000 V	Source voltage
R_o	0.1 Ohm	Source resistance
L_o	9 mH	Source inductance
f_s	2 kHz	Switching frequency
d	0.5	Duty cycle
R	1 Ohm	Load resistance
L	10 mH	Load inductance
C	2200 μF	Capacitance

```
% This file is part of the Nonlinear Vector Fitting Toolbox, v1.
% Filename: ex1_BC.m
% Package: Nonlinear_Vector_Fitting_Toolbox_1.zip.
% Programmed by A. Ramirez and B. Gustavsen. February 13, 2026

clear variables

%Loads input/output vectors from NLT algorithm
load res_bc.mat% s N Vdc Vc

%X => input, Y => output (takes positive frequencies only)
X(1,:)=Vdc(1:N/2+1); Y(1,:)=Vc(1:N/2+1); w(1,:)=imag(s(1:N/2+1));

opts.stable=1; % Forcing unstable poles to be stable
weight=ones(1,N/2+1);
opts.asymp=2; % 1 => strictly proper (SERD = 0), 2 => proper (SERD ~= 0)
opts.relax=1;
opts.logx=0;

%=====
%=      POLE-RESIDUE FITTING      =
%=====

opts.N=20; %Order of approximation. (Is used when opts.poles=[]).
opts.poletype='lincmplx'; %Will use logarithmically spaced, complex poles. (Is used
                        when opts.poles=[]).
poles=[]; %[] --> N initial poles are automatically generated as defined by
opts.startpoleflag
```



```

iter=10;
to=cputime;
for ii=1:iter
    [SER,poles,rmserr,fit,opts]=NL_VF(X,Y,s(1:N/2+1),poles,weight,opts);
    disp(['... Iteration ' num2str(ii)])
end
tf=cputime-to;
disp([' Elapsed time is ',num2str(tf), ' seconds'])

```

4.2.2 Running the example

Running `ex1_BC.m` from Matlab's command window gives the output to screen shown below.

```

>> ex1_BC
... Iteration 1
... Iteration 2
... Iteration 3
... Iteration 4
... Iteration 5
... Iteration 6
... Iteration 7
... Iteration 8
... Iteration 9
... Iteration 10
Elapsed time is 1.5156 seconds

```

The results in Fig. 4.2.2 show that the approximation of $V_c(s)$ is accurate past the second harmonic of switching frequency (2 kHz). (The plot is automatically generated).

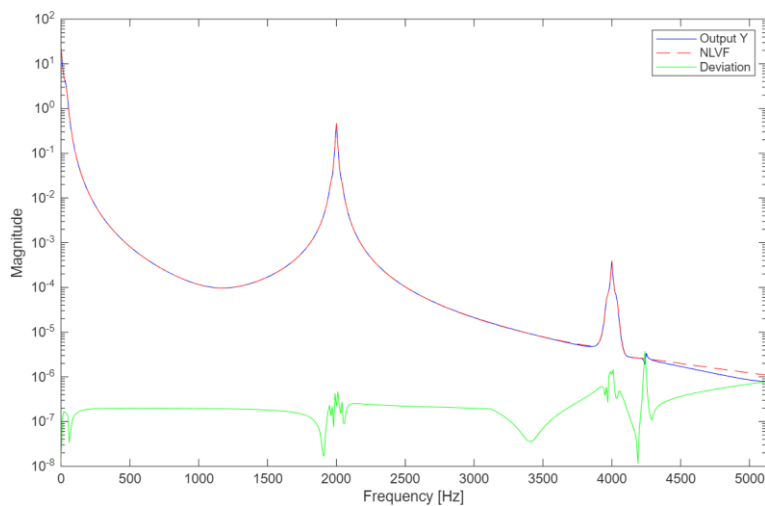


Fig. 4.2.2 Generated plot of rational fitting

Pole-residue model of $H(s)$:

```

>> SER.poles

1.0e+05 *

-4.9858 + 0.0000i
-0.0003 + 0.0000i
-0.0004 + 0.2537i
-0.0004 - 0.2537i
-0.0004 + 0.2490i
-0.0004 - 0.2490i
-0.0000 + 0.2513i
-0.0000 - 0.2513i
-0.0003 + 0.2514i
-0.0003 - 0.2514i
-0.0004 + 0.0023i
-0.0004 - 0.0023i

```

```

-0.0004 + 0.1280i
-0.0004 - 0.1280i
-0.0004 + 0.1233i
-0.0004 - 0.1233i
-0.0000 + 0.1257i
-0.0000 - 0.1257i
-0.0003 + 0.1257i
-0.0003 - 0.1257i

>> SER.C

ans(:, :, 1) = 59.1080 + 0.0000i
ans(:, :, 2) = 31.4961 + 0.0000i
ans(:, :, 3) = 0.0388 + 0.0883i
ans(:, :, 4) = 0.0388 - 0.0883i
ans(:, :, 5) = -0.0467 + 0.0816i
ans(:, :, 6) = -0.0467 - 0.0816i
ans(:, :, 7) = 0.0911 - 0.5708i
ans(:, :, 8) = 0.0911 + 0.5708i
ans(:, :, 9) = -0.0765 + 0.3994i
ans(:, :, 10) = -0.0765 - 0.3994i
ans(:, :, 11) = -15.7489 -53.1969i
ans(:, :, 12) = -15.7489 +53.1969i
ans(:, :, 13) = -6.7777 + 4.3778i
ans(:, :, 14) = -6.7777 - 4.3778i
ans(:, :, 15) = 6.4826 + 4.2859i
ans(:, :, 16) = 6.4826 - 4.2859i
ans(:, :, 17) = 2.2017e+00 - 4.1350e+02i
ans(:, :, 18) = 2.2017e+00 + 4.1350e+02i
ans(:, :, 19) = -1.9049e+00 + 4.0484e+02i
ans(:, :, 20) = -1.9049e+00 - 4.0484e+02i

>> SER.D

ans = -1.1840e-04

```

State-space model of $H(s)$:

The state-space model has a diagonal **A** with complex-valued **A** and **C**, as requested by parameter

```
opts.cmplx_ss=1.
```

```

>> SER.A

1.0e+05 *

(1,1)    -4.9858 + 0.0000i
(2,2)    -0.0003 + 0.0000i
(3,3)    -0.0004 + 0.2537i
(4,4)    -0.0004 - 0.2537i
(5,5)    -0.0004 + 0.2490i
(6,6)    -0.0004 - 0.2490i
(7,7)    -0.0000 + 0.2513i
(8,8)    -0.0000 - 0.2513i
(9,9)    -0.0003 + 0.2514i
(10,10)  -0.0003 - 0.2514i
(11,11)  -0.0004 + 0.0023i
(12,12)  -0.0004 - 0.0023i
(13,13)  -0.0004 + 0.1280i
(14,14)  -0.0004 - 0.1280i
(15,15)  -0.0004 + 0.1233i
(16,16)  -0.0004 - 0.1233i
(17,17)  -0.0000 + 0.1257i
(18,18)  -0.0000 - 0.1257i
(19,19)  -0.0003 + 0.1257i
(20,20)  -0.0003 - 0.1257i

>> SER.B

1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

```

1
1
1
1
1
1
1
1
1
1
1

>> SER.C

```
ans(:, :, 1) = 59.1080 + 0.0000i
ans(:, :, 2) = 31.4961 + 0.0000i
ans(:, :, 3) = 0.0388 + 0.0883i
ans(:, :, 4) = 0.0388 - 0.0883i
ans(:, :, 5) = -0.0467 + 0.0816i
ans(:, :, 6) = -0.0467 - 0.0816i
ans(:, :, 7) = 0.0911 - 0.5708i
ans(:, :, 8) = 0.0911 + 0.5708i
ans(:, :, 9) = -0.0765 + 0.3994i
ans(:, :, 10) = -0.0765 - 0.3994i
ans(:, :, 11) = -15.7489 -53.1969i
ans(:, :, 12) = -15.7489 +53.1969i
ans(:, :, 13) = -6.7777 + 4.3778i
ans(:, :, 14) = -6.7777 - 4.3778i
ans(:, :, 15) = 6.4826 + 4.2859i
ans(:, :, 16) = 6.4826 - 4.2859i
ans(:, :, 17) = 2.2017e+00 - 4.1350e+02i
ans(:, :, 18) = 2.2017e+00 + 4.1350e+02i
ans(:, :, 19) = -1.9049e+00 + 4.0484e+02i
ans(:, :, 20) = -1.9049e+00 - 4.0484e+02i
```

The **D** matrix is the same as for the pole-residue model.

With `opts.cmplx_ss=0`, a real-only state-space model is produced,

>> SER.A

1.0e+05 *

```
(1,1)      -4.9858
(2,2)      -0.0003
(3,3)      -0.0004
(4,3)      -0.2537
(3,4)       0.2537
(4,4)      -0.0004
(5,5)      -0.0004
(6,5)      -0.2490
(5,6)       0.2490
(6,6)      -0.0004
(7,7)      -0.0000
(8,7)      -0.2513
(7,8)       0.2513
(8,8)      -0.0000
(9,9)      -0.0003
(10,9)     -0.2514
(9,10)      0.2514
(10,10)    -0.0003
(11,11)    -0.0004
(12,11)    -0.0023
(11,12)     0.0023
(12,12)    -0.0004
(13,13)    -0.0004
(14,13)    -0.1280
(13,14)     0.1280
(14,14)    -0.0004
(15,15)    -0.0004
(16,15)    -0.1233
(15,16)     0.1233
(16,16)    -0.0004
(17,17)    -0.0000
(18,17)    -0.1257
(17,18)     0.1257
(18,18)    -0.0000
(19,19)    -0.0003
(20,19)    -0.1257
(19,20)     0.1257
(20,20)    -0.0003
```

1
1
2
0
2
0
2
0
2
0
2
0
2
0
2
0
2
0

```
>> SER.C
```

```
ans(:, :, 1) = 59.1080
ans(:, :, 2) = 31.4961
ans(:, :, 3) = 0.0388
ans(:, :, 4) = 0.0883
ans(:, :, 5) = -0.0467
ans(:, :, 6) = 0.0816
ans(:, :, 7) = 0.0911
ans(:, :, 8) = -0.5708
ans(:, :, 9) = -0.0765
ans(:, :, 10) = 0.3994
ans(:, :, 11) = -15.7489
ans(:, :, 12) = -53.1969
ans(:, :, 13) = -6.7777
ans(:, :, 14) = 4.3778
ans(:, :, 15) = 6.4826
ans(:, :, 16) = 4.2859
ans(:, :, 17) = 2.2017
ans(:, :, 18) = -413.5045
ans(:, :, 19) = -1.9049
ans(:, :, 20) = 404.8422
```

4.3 Example 2: EMTP circuit

File: ex2_circuit_emtp.m

In this example the usage of `NL_VF.m` with time domain input variables is demonstrated.

4.3.1 Data case

The circuit in Fig. 4.3.1 can be found with the name `test_ss.ecf` in the example files of the EMTP commercial software. The parameters are indicated in Fig. 4.3.1 (quantities given in units [V], [Ω], [F]). For this case study, the time domain input and output variables are assumed to be the voltage at R4 resistor terminals (`vf_CIR`) and the current from the AC source (`CIR`), respectively. These time domain input/output variables, presented in Fig. 4.3.2, are transformed to frequency domain via the NLT algorithm [6] and fed into `NL_vf.m` for the system's dynamics identification, as shown below.

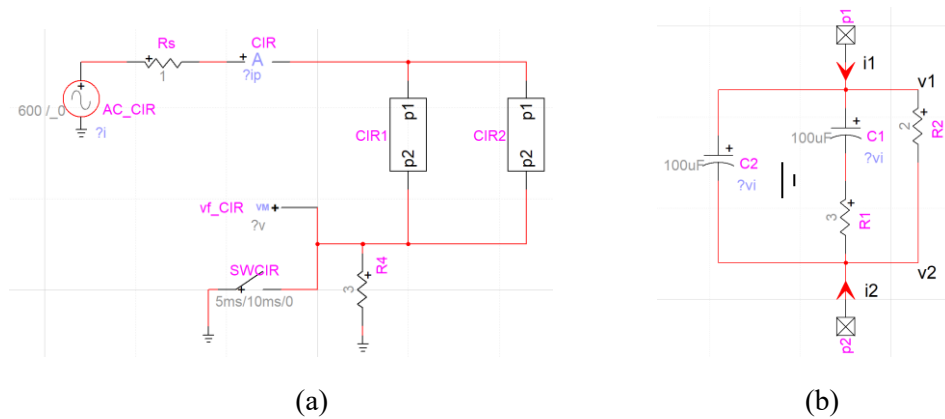


Fig. 4.3.1 (a) Circuit from EMTP examples list, (b) details of CIR1 (identical to CIR2)

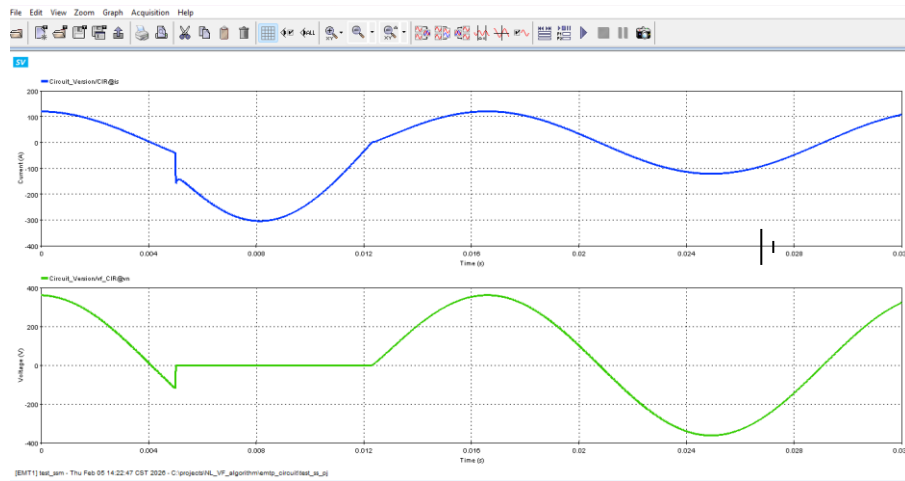


Fig. 4.3.2 Time domain waveforms obtained from EMTP. Top: current from the AC source (CIR). Bottom: voltage at R4 resistor terminals (vf_CIR)

```
% This file is part of the Nonlinear Vector Fitting Toolbox, v1.
% Filename: ex2_circuit_emtp.m
% Package: Nonlinear_Vector_Fitting_Toolbox_1.zip.
% Programmed by A. Ramirez and B. Gustavsen. February 13, 2026
```

```
clear variables, clc, warning off
```

```
% Loads time domain (TD) input/output vectors from EMTP
load res_circuit_emtp.mat
```

```
%=====
% =   Samples converted from TD to frequency domain (FD)   =
%=====
```

```
t=x; % time variable
xin=Circuit_Version_vf_CIR_vn; % input variable in TD
yout=Circuit_Version_CIR_is; % output variable in TD

T=t(end); N=length(t); dt=T/N; wmax=2*pi/dt; dw=2*pi/T; w=(0:dw:wmax/2)';
omega=wmax/2;
```

```
alp=-(log(0.001))/T;
vec=exp(-alp*t);
```

```

s=alp+1i*w; Ns=length(s);

%data window (hanning) for diminishing Gibbs oscillations
sig2=.5*(1+cos(pi*w/omega)); sighalf=sig2(2:N/2+1); sig=[sig2(1); sighalf;
flipud(conj(sighalf(1:N/2-1)))];

Xin=dt*fft(xin.*vec); % input variable in FD
Yout=dt*fft(yout.*vec); % output variable in FD

% Takes only positive frequencies
X=Xin(1:Ns).'; Y=Yout(1:Ns).';

% Optional: verifies conversion from TD to FD and viceversa
% xx=inlt_pos(X,t',alp); yy=inlt_pos(Y,t',alp); % Calculates TD response by using
only positive frequency samples
% Np=round(0.99*N); % Takes 99% of samples discarding samples showing Gibbs
oscillations
% figure, plot(t,xin,'k',t(1:Np),xx(1:Np),'r',t,yout,'b',t(1:Np),yy(1:Np),'m'),

%=====
% = POLE-RESIDUE FITTING =
%=====

opts.stable=1; % Forcing unstable poles to be stable
weight=ones(1,Ns);
opts.asymp=1; % 1 => strictly proper (SERD = 0), 2 => proper (SERD ~= 0)
opts.relax=1;
opts.logx=0;
opts.cmplx_ss=1;

opts.N=450; %Order of approximation. (Is used when opts.poles=[]).
opts.poletype='logcmplx'; %Will use logarithmically spaced, complex poles. (Is used
when opts.poles=[]).
poles=[]; %[] --> N initial poles are automatically generated as defined by
opts.startpoleflag

iter=30;
to=cputime;
for ii=1:iter
    [SER,poles,rmserr,fit,opts]=NL_VF(X,Y,s.',poles,weight,opts);
    disp(['... Iteration ' num2str(ii)])
end
tf=cputime-to;
disp([' Elapsed time is ',num2str(tf), ' seconds'])

%=====
% = VERIFICATION =
%=====

yfit=inlt_pos(fit,t',alp); % Calculates TD response by using only positive
frequency samples

Np=round(0.99*N); % Takes 99% of samples discarding samples showing Gibbs
oscillations
t=t(1:Np);
figure, plot(t,yout(1:Np),'k',t,yfit(1:Np),'r',t,abs(yout(1:Np)-yfit(1:Np).'),'g'),
xlabel('time (s)'), ylabel('current (A)'), legend('accurate','NLT-VF','error')

%=====
% = inlt_pos calculates inverse numerical Laplace transform =
% = by using positive frequencies only =
%=====

function ft = inlt_pos(Fs,t,c)

N_2=length(Fs);

```

```

N=(N_2-1)*2;
delt=t(2);
%sigma=lanczos(N);           %Lanczos data window
sigma=hamming2(N_2,0.5);    %Hamming, Hanning data window
Fss=Fs.*sigma;
ft=real(ifft(Fss,N,'symmetric').*exp(c*t))/delt;
ft=ft/2;

```

end

4.3.2 Running the example

Running `ex2_circuit_emtp.m` from Matlab's command window gives the output to screen shown below.

```

>> ex2_circuit_emtp
... Iteration 1
... Iteration 2
...
... Iteration 30
Elapsed time is 37.375 seconds

```

The results in Fig. 4.3.3 show that the approximation of the output is accurate (the plot is automatically generated). Note the challenge involved in fitting the dynamics of this system due to the large number of peaks (around 125 peaks) along the frequency range. The number of poles required to achieve that accuracy is of 450, which have been specified to be logarithmically distributed in frequency.

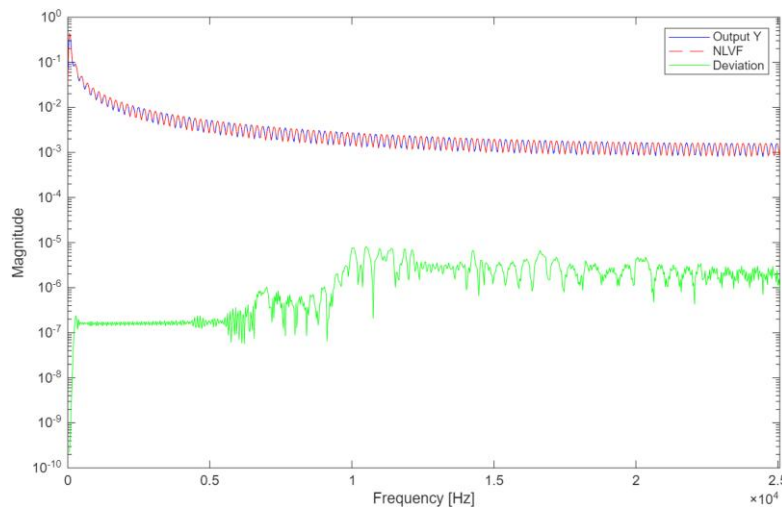


Fig. 4.3.3 Generated plot of rational fitting

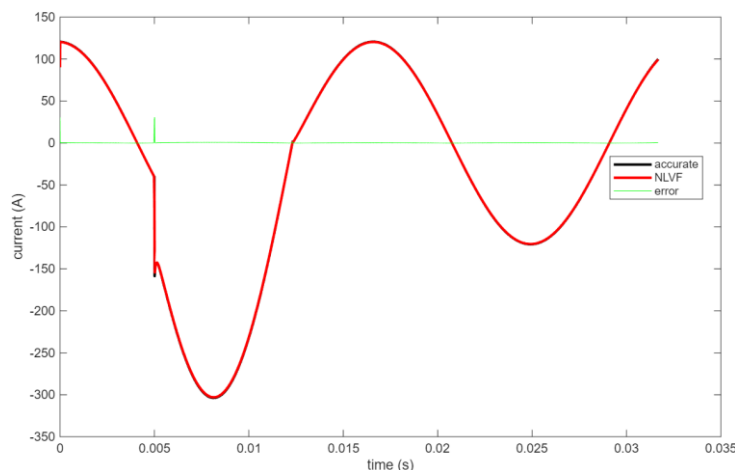


Fig. 4.3.4 Time domain comparison between the output variable given by EMTP (black trace) and the one by NL-VF

4.4 Example 3: Three-phase photovoltaic system and nonlinear load

File: `ex3_PV_nonlinear.m`

In this example the usage of `NL_VF.m` with frequency domain input variables is demonstrated.

4.4.1 Data case

The system in Fig. 4.4.1 has been reported in [4]. The system includes a small transmission network, one three-phase photovoltaic (PV) generator connected at node 4, and a three-phase nonlinear reactor connected at node 3. The underground cable (UC) and overhead transmission lines (TL) are modeled as two-port network frequency dependent components [4]. The system of Fig. 4.4.1 is simulated in the FD via the NLT algorithm [6] programmed in Matlab®. The FD input and output variables are assumed to be the voltage at the point of common coupling (Vpcc), node 4, and the current going out from the PV's filter (If), respectively. These FD input/output variables are fed into `NL_VF.m` for the system's dynamics identification, as shown below.

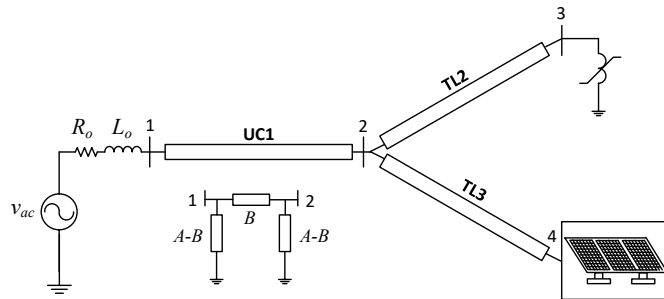


Fig. 4.4.1 PV system involving a nonlinear reactor, taken from [4]

```
% This file is part of the Nonlinear Vector Fitting Toolbox, v1.
% Filename: ex3_PV_nonlinear.m
% Package: Nonlinear_Vector_Fitting_Toolbox_1.zip.
% Programmed by A. Ramirez and B. Gustavsen. April 15, 2026

clear variables, clc, warning off

%Loads input/output vectors from NLT algorithm
load res_pv.mat % N s If Vpcc

%X => input, Y => output (note that H is a square matrix with frequency coupling)
X=Vpcc(:,1:N/2+1);
Y=[If(1:N/2+1).';If(N+1:N+N/2+1).';If(2*N+1:2*N+N/2+1).'];
w=imag(s(1:N/2+1));

opts.stable=1; % Forcing unstable poles to be stable
weight=ones(1,N/2+1);
opts.asymp=2; % 1 => strictly proper (SERD = 0), 2 => proper (SERD ~= 0)
opts.relax=1;
opts.logx=0;
opts.cmplx_ss=1;

%=====
% =      POLE-RESIDUE FITTING      =
%=====

opts.N=150; %Order of approximation. (Is used when opts.poles=[]).
opts.poletype='lincmplx'; %Will use linearly spaced, complex poles. (Is used when
opts.poles=[]).
```



```

poles=[]; %[] --> N initial poles are automatically generated as defined by
opts.startpoleflag

iter=10;
to=cputime;
for ii=1:iter
    [SER,poles,rmserr,fit,opts]=NL_VF(X,Y,s(1:N/2+1).',poles,weight,opts);
    disp(['... Iteration ' num2str(ii)])
end
tf=cputime-to;
disp([' Elapsed time is ',num2str(tf), ' seconds'])

%=====
% =   VERIFICATION =
%=====

Ya=[Y(1,1:N/2+1).';conj(flipud(Y(1,2:N/2).'))];
fita=[fit(1,1:N/2+1).';conj(fliplr(fit(1,2:N/2).'))];

figure,
semilogy(w/(2*pi),abs(Ya(1:N/2+1)), 'k',w/(2*pi),abs(fita(1:N/2+1)), 'b',w/(2*pi),abs
(Ya(1:N/2+1)-fita(1:N/2+1)), 'g', 'LineWidth',2),
xlabel('frequency (Hz)'), ylabel('|I_f_a| (A)'), legend('NLT','NL-VF','|error|')

%=====
% =   TD VERIFICATION (optional)   =
%=====
% Frequency domain discretization
fo=50; NT=2; T=NT/fo; N=256*2; dt=T/N; n=(0:N-1).'; t=n*dt; wmax=2*pi/dt;
dw=2*pi/T; w=(0:dw:wmax/2)'; omega=wmax/2;
alf=-(log(0.001))/T/2;
vecexp=exp(alf*t); vec2=exp(-alf*t);
s1=alf+1i*w; shalf=s1(2:N/2+1); s2=[s1(1); shalf; flipud(conj(shalf(1:N/2-1)))];
Np=round(0.99*N);

%data window (Hanning) for diminishing Gibbs oscillations
sig2=.5*(1+cos(pi*w/omega)); sighalf=sig2(2:N/2+1); sig=[sig2(1); sighalf;
flipud(conj(sighalf(1:N/2-1)))];

% includes + and - frequencies in input X
Nc=3;
Xaux=zeros(Nc,N);
for ii=1:Nc
    Xaux(ii,:)=[X(ii,:) fliplr(conj(X(ii,2:N/2)))];
end
X=Xaux;

% calculating transfer function
A=SER.A; B=SER.B; C=SER.C; D=SER.D;
Haprox=zeros(Nc,Nc,N);
y_fit=zeros(Nc,N);
AA=diag(A(1:length(poles),1:length(poles)));
for ii=1:N
    aux=[];
    for jj=1:Nc
        aux=blkdiag(aux,(1./(s2(ii)-AA)));
    end
    Haprox(:, :, ii)=C*aux+D;
    y_fit(:,ii)=Haprox(:, :, ii)*X(:,ii);
end

```

```
%plotting original output and fitted one
ifa_fit=real(vecexp.*ifft((y_fit(1,:)).'*sig))/dt;
ifa_orig=real(vecexp.*ifft((If(1:N)).*sig))/dt;
figure,
plot(t(1:Np),ifa_orig(1:Np),'k',t(1:Np),ifa_fit(1:Np),'r',t(1:Np),abs(ifa_orig(1:Np)
)-ifa_fit(1:Np)), 'g', 'LineWidth',2)
xlabel('time (s)'), ylabel('|I_f_a| (A)'), legend('NLT', 'NL-VF', 'error')
```

4.4.2 Running the example

Running `ex3_PV_nonlinear.m` from Matlab's command window gives the output to screen shown below.

```
>> ex3_PV_nonlinear
.. Iteration 1
... Iteration 2
... Iteration 3
... Iteration 4
... Iteration 5
... Iteration 6
... Iteration 7
... Iteration 8
... Iteration 9
... Iteration 10
Elapsed time is 20.6875 seconds
```

The results in Fig. 4.4.2 show the approximation of the output for phase *a* only. The number of linearly spaced complex poles is 150 and 10 iterations are defined. The relaxed version of `NL_VF.m` is applied. Finally, Fig. 4.4.3 presents the comparison of the time-domain corresponding waveforms, i.e., original (black trace) and fitted (red trace) outputs, which are basically superimposed.

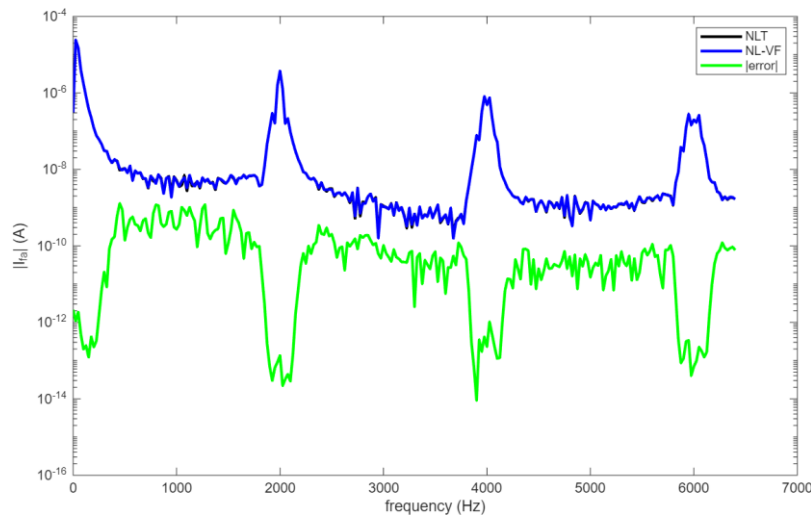


Fig. 4.4.2 Plot of rational fitting corresponding to output I_f , phase *a*.

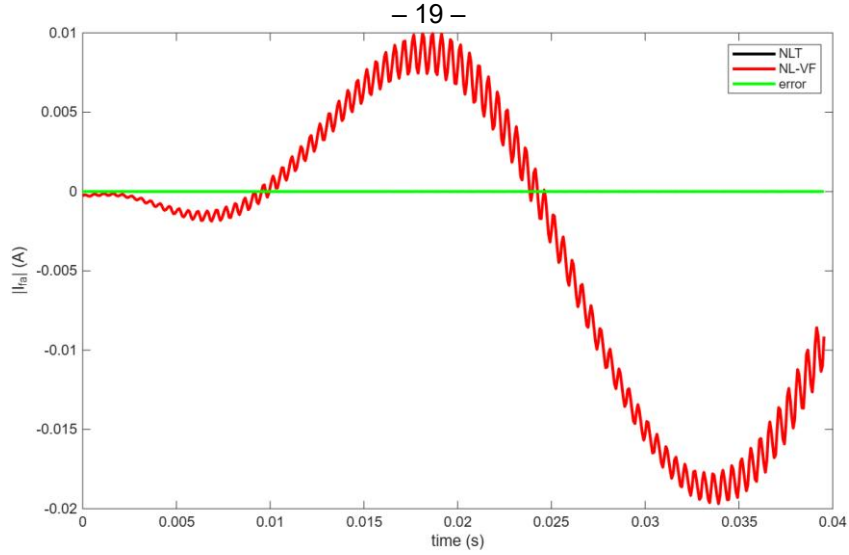


Fig. 4.4.3 Instantaneous current I_f through the PV filter, phase a , obtained from original waveform and fitted response.

5. REFERENCE FOR COMPUTATIONAL APPROACH

5.1 Pole-residue modeling by *NL_VF*

5.1.1 General

NL-VF has as main objective to generate a rational approximation of the nonlinear behavior of a system, having as available data the input/output variables in frequency domain (FD) at the port(s) where the behavior approximation is under interest. The NL-VF has its fundament in a previously published time domain VF approach (TD-VF), noting that the former utilizes input/output variables in the FD for the transfer function characterization whilst TD-VF uses TD variables.

The main outcome of NL-VF is the $P \times Q$ transfer matrix $\mathbf{H}(s)$ of a system, as in (5.1). It consists of a N th order pole-residue model (5.1) where matrix \mathbf{H}_∞ corresponds to the direct coupling matrix. The model uses a common pole set $\{p_n\}$ for all elements within the associated residue matrices $\{\mathbf{R}_n\}$.

$$\mathbf{H}(s) = \sum_{n=1}^N \frac{\mathbf{R}_n}{s-p_n} + \mathbf{H}_\infty \quad (5.1)$$

As in classical VF, NL-VF is divided into two steps: poles identification and residues calculation.

5.1.2 Standard NL_VF

Considering that both input $\mathbf{X}(s)$ and output $\mathbf{Y}(s)$ are available, the poles identification step utilizes (5.2) with q_n as the set of initial poles:

$$\sigma(s)\mathbf{Y}(s) = \left[\sum_{n=1}^N \frac{\mathbf{M}_n}{s-q_n} + \mathbf{M}_\infty \right] \mathbf{X}(s) \quad (5.2)$$

Where:

$$\sigma(s) = \left[1 + \sum_{n=1}^N \frac{k_n}{s-q_n} \right] \quad (5.3)$$

The set of residues $\{k_n\}$ permits to calculate the zeros of $\sigma(s)$ which correspond to a new set of poles $\{p_n\}$. The zeros of $\sigma(s)$ are calculated by solving the eigenvalue problem (5.4) where \mathbf{A} is a diagonal matrix holding the poles $\{q_n\}$, \mathbf{b} is a column of ones, and \mathbf{c} holds the residues $\{k_n\}$

$$\text{eig}(\mathbf{A} - \mathbf{b}\mathbf{c}^T) \quad (5.4)$$

At this point, the new set $\{p_n\}$ can be used within a pole relocation scheme to obtain a better set of poles estimate. During the iterations, unstable poles may occasionally occur. Any unstable pole is flipped into the left half plane by inverting the sign of the real part, thereby guaranteeing a rational model (5.1) with *stable poles* only. (The pole flipping is requested by setting parameter `opts.stable=1`, which is the default setting).

As for the residues calculation, the obtained set of poles $\{p_n\}$ is used to calculate matrices \mathbf{R}_n and \mathbf{H}_∞ in (5.5), related to $\mathbf{X}(s)$ and $\mathbf{Y}(s)$ as

$$\mathbf{Y}(s) = \left[\sum_{n=1}^N \frac{\mathbf{R}_n}{s-p_n} + \mathbf{H}_\infty \right] \mathbf{X}(s) \quad (5.5)$$

The pole identification and subsequent residue identification both lead to solving an overdetermined problem of the form

$$\mathbf{A}\mathbf{x} \cong \mathbf{b} \quad (5.6)$$

A transformation of variables is used to ensure that complex poles and residues come in conjugate pairs. In the actual implementation, the conditioning of \mathbf{A} is improved by scaling its columns to unit length. The preferred solver is (sparse) QR decomposition with rank revealing column pivoting. (`NL_VF.m` solves the equation using the “\” operator in Matlab).

The initial poles to be specified for the first VF iteration consist of weakly attenuated conjugate pairs that are distributed over the frequency band of interest,

$$q_n = -\alpha + j\beta, \quad q_{n+1} = -\alpha - j\beta \quad (5.7a)$$

$$\alpha = \nu \cdot \beta \quad (5.7b)$$

The factor ν in (5.7b) is usually taken as 0.01 or smaller. This choice of initial poles ensures a well-conditioned system matrix \mathbf{A} in (5.6).

5.1.3 Relaxed NL_VF

Equation (5.3) has been normalized by setting one coefficient to unity. It was found that this normalization can seriously impair the pole relocation process when fitting noisy responses. This problem was overcome by the *relaxed* formulation [2] where the fixed coefficient (unity) is replaced with an unknown coefficient \tilde{d} ,

$$\sigma(s) = \sum_{k=1}^N \frac{\tilde{c}_k}{s - q_k} + \tilde{d} \quad (5.8)$$

As normalization, an additional row is introduced in the least squares (LS) problem (N_s is the number of samples).

$$Re \left\{ \sum_{m=1}^{N_s} \left(\sum_{k=1}^N \frac{\tilde{c}_k}{s_m - q_k} + \tilde{d} \right) \right\} = N_s \quad (5.9)$$

This equation is given a LS weighting in relation to the size of $\mathbf{H}(s)$ by

$$weight = \frac{\|w(s) \cdot H(s)\|_2}{N_s} \quad (5.10)$$

The new poles are now calculated as

$$eig(\mathbf{A} - \mathbf{b} \cdot \tilde{d}^{-1} \mathbf{c}^T) \quad (5.11)$$

It is remarked that the new constraint simply imposes that that integral of $\sigma(s)$ is non-zero, without fixing any coefficients. This gives improved convergence compared to usage of (5.3), where $\sigma(s)$ is enforced to approach unity at infinite frequency. In particular, the original VF formulation is biased to relocating poles from high frequencies towards low frequencies.

5.1.4 Expansion into State Space model

The pole-residue model (5.1) can be directly converted into the form

$$\mathbf{H} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \quad (5.12)$$

The expansion process is straightforward, as shown in [7]. \mathbf{A} is a diagonal matrix that holds the poles $\{p_m\}$, repeated as many times as \mathbf{H} has columns. \mathbf{C} holds the elements of the residue matrices $\{\mathbf{R}_m\}$. \mathbf{B} is a selector matrix containing ones and zeros that associate each input to a separate block (column set) in \mathbf{A} and \mathbf{C} .

The building of \mathbf{A} , \mathbf{B} , \mathbf{C} is done as shown in Fig. 5.1 for a third-order, two-port case. It is seen that \mathbf{C} is established by copying the columns of the \mathbf{R} -matrices into the appropriate locations in \mathbf{C} . \mathbf{B} is a selector matrix that associates the inputs to blocks (columns) of \mathbf{A} and \mathbf{C} . It is noted that \mathbf{A} is diagonal with the poles repeated as many times as there are ports.

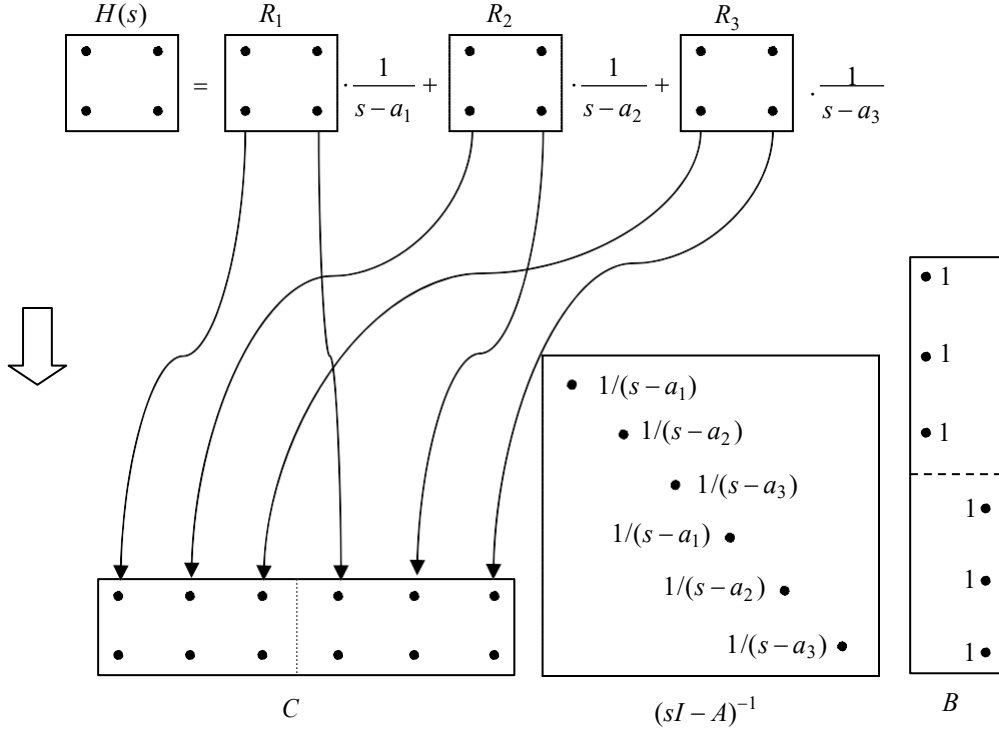


Fig. 5.1. Converting (5.1) into a state-space model

The state-space model can be converted into a real-only model (via a similarity transformation) as follows.

$$\hat{\mathbf{A}} = \begin{bmatrix} \text{Re}\{a\} & \text{Im}\{a\} \\ -\text{Im}\{a\} & \text{Re}\{a\} \end{bmatrix}, \quad \hat{\mathbf{c}} = [\text{Re}\{c\} \quad \text{Im}\{c\}] \quad (5.13)$$

$$\hat{\mathbf{b}} = \begin{bmatrix} 2\text{Re}\{b^T\} \\ -2\text{Im}\{b^T\} \end{bmatrix} = \begin{bmatrix} 2b^T \\ 0^T \end{bmatrix} \quad (5.14)$$

6. REFERENCES

- [1] B. Gustavsen and A. Semlyen, “Rational approximation of frequency domain responses by Vector Fitting”, *IEEE Trans. Power Delivery*, vol. 14, no. 3, pp. 1052-1061, July 1999.
- [2] B. Gustavsen, “Improving the pole relocating properties of vector fitting”, *IEEE Trans. Power Delivery*, vol. 21, no. 3, pp. 1587-1592, July 2006.
- [3] D. Deschrijver, M. Mrozowski, T. Dhaene, and D. De Zutter, “Macromodeling of multiport systems using a fast implementation of the vector fitting method”, *IEEE Microwave and Wireless Components Letters*, vol. 18, no. 6, pp. 383-385, June 2008.
- [4] A. Ramirez, B. Gustavsen, and I. Ramirez, “Rational approximation of nonlinear systems via NL-VF,” *IEEE Trans. Power Delivery*, vol. 38, no. 3, pp. 1918-1926, June 2023.
- [5] A. Ramirez and B. Gustavsen, “Relaxed nonlinear vector fitting for calculation of rational approximation of systems defined by input/output responses in the frequency domain,” *IEEE Trans. Power Delivery*, vol. 39, no. 5, pp. 2965-2972, Oct. 2024.

- [6] P. Moreno and A. Ramirez, “Implementation of the numerical Laplace transform: A review”, *IEEE Trans. Power Delivery*, vol. 23, no. 4, pp. 2599-2609, Oct. 2008.
- [7] B. Gustavsen, “Computer code for rational approximation of frequency dependent admittance matrices”, *IEEE Trans. Power Delivery*, vol. 17, no. 4, pp. 1093-1098, October 2002.