

Last revised : 26.02.2026

# **MFT-NNLS**

A Matlab toolbox for multi-port passive rational modeling  
from Y-, Z- and S-parameter data sets

TUTORIAL  
REFERENCE  
TEST CASES

version 1.0 for Matlab

Bjørn Gustavsen  
SINTEF Energy Research  
N-7465 Trondheim  
NORWAY

e-mail: [bjorn.gustavsen@sintef.no](mailto:bjorn.gustavsen@sintef.no)

## Licencing

This package consists of a set of Matlab scripts and a set of worked examples. You are free to use the Matlab scripts and modify as you like as there are no licensing requirements, except for those listed below. I only ask that you make proper referencing to this toolbox and underlying original works, at your own discretion. A minimal referencing could be:

B. Gustavsen, “MFT-NNLS: A toolbox for passive macromodeling from large immittance and scattering data sets using vector fitting and residue perturbation in Matlab”, *IEEE Trans. Power Delivery*, available online: [ieeexplore.org](http://ieeexplore.org).

### Licensing exceptions:

The program RPdriver.m (for passivity enforcement) has an option for specifying alternative NNLS solvers:

- ‘lsqnonneg’ (uses Matlab function lsqnonneg.m – no issue here!)
- ‘nnls’ (default option, uses function nnls.m)
- ‘tntnn’ (uses function tntnn.m)

**nnls.m** is a development of Bill Whiten, retrieved from Mathworks File Exchange,

<https://se.mathworks.com/matlabcentral/fileexchange/38003-nnls-non-negative-least-squares>

With use of it, you must adhere to the stated copyright notice, which is found in folder /nnls/, file “license.txt”).

**tntnn.m** is a development of Eric Frahm (frahm@physics.umn.edu) and Joseph Myhre,

(myre@stthomas.edu). The function is licensed under the “GNU General Public License 3.0”:

<https://github.com/ProfMyre/tnt-nn/blob/master/LICENSE> .

## Table of Contents

1. INTRODUCTION.....	5
1.1 Purpose.....	5
1.2 Program code.....	5
1.3 Document organization .....	7
2. THE PACKAGE .....	8
2.1 Program Files .....	8
2.2 Instalment.....	9
3. FUNCTION CALL .....	10
3.1 VFdriver.m.....	10
3.2 RPdriver.m .....	14
3.3 netgen_ATP.m.....	18
3.3 netgen_EMTP.m .....	19
3.3 netgen_PSCAD.m .....	19
4. TUTORIAL.....	20
4.1 Hints and advice.....	20
4.2 Example 1: Electrical circuit.....	22
4.2.1 Data case .....	22
4.2.2 Running the example.....	23
4.2.3 Generation of simulation model for ATP.....	28
4.3 Example 2: Network equivalencing.....	31
4.3.1 Generation of frequency domain data .....	31
4.3.2 Rational fitting.....	31
4.3.3 Passivity enforcement .....	33
4.4 Example 3: Network equivalencing of transmission line.....	36
4.5 Example 4: Network equivalencing: S-parameters.....	38
4.5.1 Rational fitting.....	38
4.5.2 Passivity enforcement .....	39
5. REFERENCE FOR COMPUTATIONAL APPROACH .....	41
5.1 Pole-residue modeling by Vector Fitting .....	41
5.1.1 General .....	41
5.1.2 Standard Vector Fitting .....	42
5.1.3 Relaxed Vector Fitting .....	43
5.1.4 Fast implementation .....	44
5.1.5 Expansion into State Space model .....	46
5.2 Y-parameters: Passivity assessment and enforcement .....	47
5.2.1 Passivity assessment via Singularity Test Matrix .....	47
5.2.2 Residue Perturbation by QR-based compacting and NNLS solving.....	48
5.2.3 Reducing the number of constraints.....	49
5.2.4 Robust iterations.....	50
5.3 S-parameters: Passivity assessment and enforcement.....	51
5.3.1 Passivity assessment via singularity test matrix.....	51
5.3.2 Passivity enforcement by QR-based compacting and NNLS solving.....	51
6. TEST CASES.....	53

6.1	FDNE modeling from Y-parameters.....	53
6.1.1	Rational fitting.....	53
6.1.2	Residue perturbation .....	54
6.1.3	Final validation: Efficient plotting of final result.....	55
6.2	Transformer black-box modeling from Y-parameters .....	57
6.2.1	Rational fitting.....	57
6.2.2	Residue perturbation .....	58
6.2.3	Final validation.....	59
6.3	Transformer white-box modeling from Z-parameters.....	61
6.3.1	Rational fitting.....	61
6.3.2	Residue perturbation .....	62
6.3.3	Final validation.....	63
6.4	High-speed interconnect modeling from S-parameters.....	65
6.4.1	Rational fitting.....	65
6.4.2	Residue perturbation .....	66
6.4.3	Final validation.....	67
6.	REFERENCES.....	69
7.	ACKNOWLEDGEMENT .....	70

# 1. INTRODUCTION

## 1.1 Purpose

This manual describes MFT-NNLS, which is a replacement for the past Matrix Fitting Toolbox (MFT). MFT-NNLS is capable of fitting passive models to large data sets (many ports/terminals and high model orders), for both Y-, Z- and S-parameter data sets. MFT-NNLS and its usage is shown in [1]. The interested reader may wish to consult [2] for an in-depth, general treatment of the subject of rational fitting, passivity enforcement, and time domain simulation.

The package contains a collection of Matlab routines for fitting a stable and passive rational model to a square, symmetrical, frequency response  $\mathbf{H}(s)$ , where  $\mathbf{H}(s)$  can represent admittance (Y-) parameters, impedance (Z-) parameters, or scattering (S-) parameters. Model export to some leading Electromagnetic Transient simulation programs is available.

The output is a rational model on pole-residue (PR) form (1.1) and a corresponding state space (SS) model (1.2), both with stable poles. The model parameters can be requested to be provided on complex form, or on real-only form. Note that the matrix  $\mathbf{E}$  is zero for the S-parameter case.

$$\mathbf{H}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} + s\mathbf{E} \quad (1.1)$$

$$\mathbf{H}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} + s\mathbf{E} \quad (1.2)$$

## 1.2 Program code

The user can apply the following Matlab scripts:

- **VFdriver.m** identifies models (1.1) and (1.2) using the pole relocating Vector Fitting technique (VF) [3]. The implementation includes relaxation of the nontriviality constraint [4] (improved convergence) as well as a fast implementation of the pole identification step [5] (reduced memory requirements and computation time).
- **RPdriver.m** perturbs the model so that it becomes passive. This is achieved by perturbing the elements of the residue matrices  $\{\mathbf{R}_m\}$  and those of  $\mathbf{D}$  and  $\mathbf{E}$  while minimizing the change to the model's behavior (Y-parameters [6]). The implementation achieves highly efficient calculations using problem compacting and conversion to NNLS problem [7],[8],[9]. The passivity assessment is by default based on half-size test matrices [10], [11], but the user can alternatively specify usage of frequency screening (which is faster for large scale problems).
- **netgen\_ATP.m** exports the obtained rational model to file, for use in the ATP simulation environment <https://atp-emtp.org/>, in the form of an equivalent electrical circuit (Y-parameters, only) [12].
- **netgen\_EMTP.m** exports the obtained rational model to file, for use in the EMTP simulation environment <https://www.emtp.com/> by the state-space block in EMTP. (Y-parameters, only)

- **netgen\_PSCAD.m** exports the obtained rational model to file, for use in the PSCAD simulation environment <https://www.pscad.com/> by the FDTE component in EMTP (Y-, Z- or S-parameters).

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

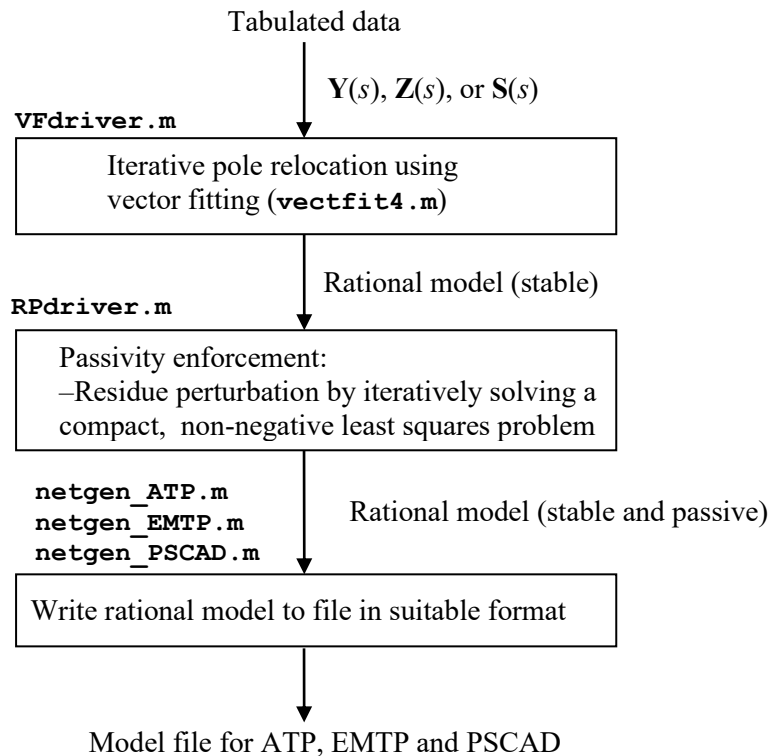


Fig. 1.1 Program overview

The program system has been tested on Matlab R2023b. All timing results are with a desktop computer running under Windows 11 using a laptop PC with 13th Gen Intel(R) Core(TM) i7-1370P at 1.90 GHz, and 32 GB RAM.

Download site (**MFT-NNLS\_2026\_1.zip**):

<http://www.energy.sintef.no/Produkt/VECTFIT/index.asp>

#### Suggested referencing:

- For general usage of the package:
  - Reference [1].
- For use in a scientific work:
  - **VFdriver.m** and/or **vectfit4.m**:
    - References [3],[4],[5].
  - **RPdriver.m** and/or, **RP\_QRNNLS\_Y.m**, **RP\_QRNNLS\_S.m**:
    - Reference [7] (Y/Z-parameters), [9] (S-parameters).

## **1.3 Document organization**

This document is organized as follows

- Section 2 describes the program package: Files and installation.
- Section 3 describes the relevant function calls.
- Section 4 is a Tutorial, describing passive macromodeling of some small examples.
- Section 5 is a Reference, describing the applied mathematical procedures of the package.
- Section 6 provides four test cases, involving large data sets.
  - FDNE modeling (Y-parameters)
  - Transformer black-box modeling (Y-parameters)
  - Transformer white-box modeling (Z-parameters)
  - High-speed interconnect modeling (S-parameters)

## 2. THE PACKAGE

### 2.1 Program Files

The package (**MFT-NNLS\_1.zip**) consists of the following files:

#### Documentation

**user\_manual\_MTF-NNLS.pdf** This document

#### Matlab routines:

<b>VFdriver.m</b>	Driver routine for rational fitting.
<b>RPdriver.m</b>	Driver routine for passivity enforcement
<b>netgen_ATP.m</b>	Generation of model for ATP simulation environment
<b>netgen_EMTP.m</b>	Generation of model for EMTP simulation environment
<b>netgen_PSCAD.m</b>	Generation of model for PSCAD simulation environment

#### Auxiliary Matlab routines:

**VFdriver.m** and **RPdriver.m** apply a set a set of auxiliary routines, provided in folder **/code/**.

#### Tutorial example files (small systems):

<b>ex1_Y.m</b>	Modeling and simulation of a small electrical circuit (Y-parameters)
<b>ex2_Y.m</b>	Network equivalencing (Y-parameters)
<b>ex3_Y.m</b>	Network equivalencing (Y-parameters)
<b>ex4_S.m</b>	Network equivalencing (S-parameters)
<b>ex2_Y.mat</b>	s-domain data for <b>ex2_Y.m</b>
<b>ex3_Y.mat</b>	rational model for <b>ex3_Y.m</b>
<b>ex4_S.mat</b>	s-domain data for <b>ex4_S.m</b>

#### Test cases (large systems):

Topic	Data set	Worked example
Transformer black-box modeling	<b>data_transformer_blackbox_Y.mat</b>	<b>transformer_blackbox_Y.m</b>
FDNE modeling	<b>data_FDNE_Y.mat</b>	<b>FDNE_Y.m</b>
Transformer white-box modeling	<b>data_transformer_whitebox_Z.mat</b>	<b>transformer_whitebox_Z.m</b>
High-speed interconnect modeling	<b>data_interconnect_S.mat</b>	<b>interconnect_S.m</b>

**circuit.atp** ATP data file for example in Section 4.2 (associated with **ex1\_Y.m**).



## 2.2 *Instalment*

- Place all files and folders in a common directory, e.g.,  
    `c:\user\MFT-NNLS`
- Add to the Matlab search path,  
    `>>addpath c:\user\MFT-NNLS`  
    `>>addpath c:\user\MFT-NNLS\code`  
    `>>addpath c:\user\MFT-NNLS\code\auxiliary`

### 3. FUNCTION CALL

#### 3.1 VFdriver.m

Fig. 3.1. gives an overview of **VFdriver.m**. Two call steps are made to **vectfit4.m**:

1. Fit the diagonal elements of  $\mathbf{H}(s)$  with a common pole set (using `Niter1` pole-relocating iterations)
2. Fit the upper triangle elements of  $\mathbf{H}(s)$  with common pole set (using `Niter2` pole-relocating iterations).

The rationale is to calculate an improved pole set by the cheap step 1, followed by the final pole relocation in step 2. With large problems, one should consider to use a low number (or even zero) `Niter2` iterations for step 2.

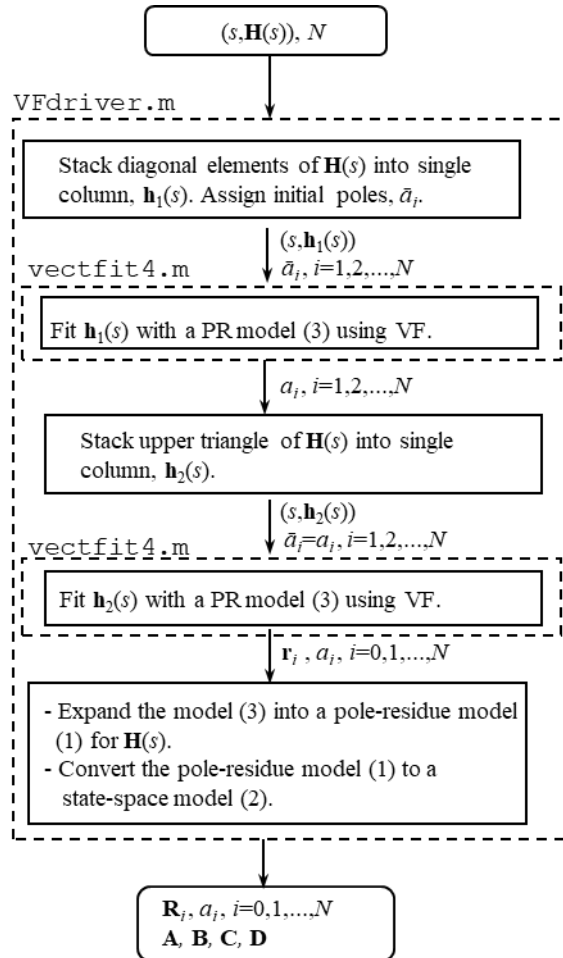


Fig. 3.1. Overview of VFdriver.m [1].

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

```
[SER,rmserr,Hfit,optsVF2]=VFdriver(H,s,poles)
[SER,rmserr,Hfit,optsVF2]=VFdriver(H,s,poles,optsVF)
```

Input:

**H** : ( $n, n, N_s$ ) 3D matrix holding the  $\mathbf{H}$ -samples  
**s** : ( $1, N_s$ ) vector holding the frequency samples,  $s = j\omega$  [rad/sec].  
**poles**: ( $1, N$ ) vector holding the initial poles (manual specification of initial poles). Use **optsVF** for automated specification.

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

Output:

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

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

The returned **SER** also holds matrices **A**, **B**, **C**, **D**, **E** for the associated state space model,  $s$

```
SER.A: (nN,nN)
SER.B: (nN,n)
SER.C: (n,nN)
SER.D: (n,n)
SER.E: (n,n)
```

**rmserr** : the resulting RMS-error of the fitting.  
**rmserrweight** : the resulting RMS-error of the fitting, considering the applied LS weighting.  
**Hfit** : ( $n, n, N_s$ ) 3D matrix holding the  $\mathbf{H}$ -samples of the rational model.  
**optsVF2** : structure containing *all* options parameters, including default settings.  
(Gives quick info about what settings are being used and what settings are available).

Below are listed the control parameters, being fields of **optsVF**. Default values are those being used if the parameter is not specified.

Table 3.1. **optsVF**

Parameter	Purpose/Description	Default
<b>General</b>		
parametertype	'Y' --> Will assume Y- or Z- params 'S' --> Will assume S- params (Information is used only when passive_DE=1, see below: <b>"Additional fields in structure optsVF"</b> )	'Y'
N poletype nu	Automated generation of N initial poles, taken as complex conjugate pairs ( $\text{pole} = -\text{nu} \cdot \beta \pm j \cdot \beta$ ) that are distributed over the frequency band (s). Specify linear or logarithmic distribution, or a mix of the two.  N : integer poletype: 'lincmplx', 'logcplx', or 'linlogcplx' nu : positive, real number  (Note: To use this option, specify the input array "poles" to be empty (poles=[])).	'lincmplx' 0.001 (nu)
Niter1	n.o. VF iterations (fitting diagonal elements of H to obtain an improved pole set).	4
Niter2	n.o. VF iterations (fitting upper triangle of H to obtain the final pole set and residue matrices)	4
<b>Model type and format</b>		
asympt	Control type of rational model =1 → D=0, E=0 =2 → D~0, E=0 =3 → D~0, E~0	2
cplx_ss	=1 → complex-valued state space model (A,B,C,D,E) with diag. A. =0 → real-valued state space model with block-diagonal A.	1
<b>Fitting control</b>		
weight	Array (Nc,Nc,Ns) containing user-defined weight for samples H(i,j,k) in least squares problem.	[]
weightparam	Automated weight array. Used when optsVF.weight=[] =1 → Same weight for all elements: weight(i,j,k)=1 =2 → weight(i,j,k)=1./abs(H(i,j,k)) =3 → weight(i,j,k)=1./sqrt(abs(H(i,j,k))) =4 → weight(k)=1/norm(H(:, :, k)) =5 → weight(k)=1/sqrt(norm(H(:, :, k)))	1

stable	Control handling of unstable poles =0 → Will not enforce stable 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
NE	=0 → Solve final residue problem using QR decomp =1 → Solve final residue problem using Normal Equations	0
<b>Monitoring progress</b>		
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 --> make additional plot of phase angles	1
screen	=1 --> will echo results to screen during fitting process.	1

**Additional fields in structure optsVF:**

Parameter	Purpose/Description	Default
remove_HFpoles	=1 --> Will remove poles at frequencies above factor_HF*s(end)	0
factor_HF		1.1
passive_DE	=1 Y-params: --> Will enforce that D and E have all eigenvalues >0 S-params: --> Will enforce that D has all sing. values <1	1
passive_DE.TOLD	Y-params: Negative eigenvals of D are made positive by this amount S-params: Sing.vals of D which are >1 are made <1 by this amount	1e-6
passive_DE.TOLE	Y-params: Neg. eigenvals of E are made positive by this amount	1e-16

### 3.2 *RPdriver.m*

Fig. 3.2. gives an overview of **RPdriver.m**. The method consists of two main steps within a loop:

1. Passivity assessment
2. Passivity enforcement

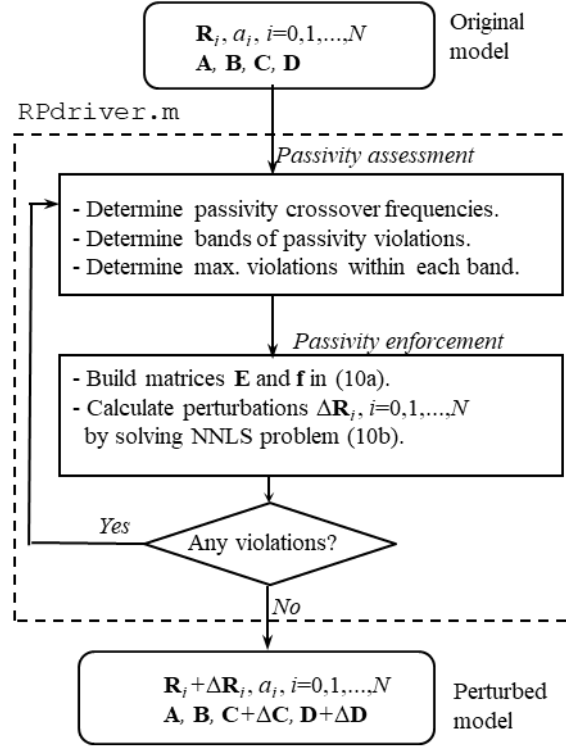


Fig. 3.2. Overview of **RPdriver.m** [1].

#### Passivity assessment

The passivity assessment determinates a set of cross-over frequencies, if any:

- Y-and Z-parameters: any frequency where an eigenvalue of  $(\text{Re}\{\mathbf{Y}\})$  crosses the zero line.
- S-parameters: any frequency where a singular value of  $\mathbf{S}$  crosses the unity line.

The crossover frequencies can be determined in two alternative ways (decided by user):

- a. By checking eigenvalues of a test matrix (which is calculated from the model's parameters)
- b. By performing a frequency sweep of eigenvalues (or singular values) over a set of frequency samples provided by the user.

Based on the identified crossover-frequencies, bands of passivity violations are determined, and the maximum passivity violations within each band are determined (global and local maxima). The maxima are transferred to the passivity enforcement step.

### Passivity enforcement

Passivity is enforced at a set of frequencies (max violations) provided from the passivity assessment step while minimizing the LS change to the original model. The passivity enforcement makes use of a compacting approach which results in a non-negative LS problem to be solved with as many free variables as there are passivity violations. The passivity enforcement is performed in an iterative loop together with the passivity assessment, as shown in Fig. 3.2. With use of Y- or Z-parameters, the user can also request usage of an inner loop which adds additional constraints.

The following function call will enforce passivity of a rational model generated by **VFdriver**,

```
[SER,Yfit,optRP2]=RPdriver(SER,s,optsRP);
```

#### Input:

**SER** is a structure with the model on pole-residue form. The contents is the same as for **VFdriver**, see Section 3.1.

- Fields **A**, **B**, **C** are the input.

- Field **SER.E** is ignored when applied to S-parameters (`opts.parametertype='S'`)

**s** : (1,**Ns**) A vector of frequency samples,  $s=j\omega$ . The perturbation seeks to minimize the change to the original model at the given samples.

**optsRP** Optional structure that can be used for overriding defaults settings, and for requesting plots.

#### Output:

**SER** : The perturbed model, on pole residue form and on state space form.

**Yfit** : (**n**,**n**,**Ns**) 3D matrix holding the Y-samples (or S-samples) of the perturbed model (at frequencies in **s**)

**optsRP2**: contains *all* options parameters, including default settings.  
(Gives quick info about what settings are being used and what settings are available).

Below are listed the control parameters, being fields of **optsVRF**. Default values are those being used if the parameter is not specified.

Table 3.2. **optsRP**

Parameters	Purpose/Description	Default
<b>General</b>		
parametertype	'Y' --> Specifies Y-parameter or Z-parameter model 'S' --> Specifies S-parameter model	'Y'
Niter_out	Max. n.o. iterations in outer loop in "Robust iterations"	20
Niter_in	Max. n.o. iterations in inner loop in "Robust iterations" (Available only with parameter type 'Y'.	0
continue	= 0 --> Include system matrix info in returned opts. (Makes it available for use with continue=1). = 1 --> Continue from a previous call to RPdriver (Enables lower cost for building matrices).	0
<b>Model format</b> (returned model)		
cmplx_ss	=1 --> perturbed state space model on complex form =0 --> perturbed state space model on real-only form	1
<b>Least squares control</b>		
weightparam	Automated generation of weight array. Used when optsRP.weightaray=[] =1 --> Same weight for all elements: weight(i,j,k)=1 =2 --> weight(i,j,k)=1./abs(H(i,j,k)) inverse weighting =3 --> weight(i,j,k)=1./sqrt(abs(H(i,j,k))) =4 --> weight(k)=1/norm(H(:, :, k)) inverse weighting =5 --> weight(k)=1/sqrt(norm(H(:, :, k)))	1
weightarray	Array (Nc,Nc,Ns) containing user-defined weight for each sample H(i,j,k) in least squares problem.	[]
weightfactor	Least Squares weight for out-of band auxiliary samples	1e-3
<b>Passivity enforcement control</b>		
TOLG, TOLD	Y- and Z- parameter model: Neg. eigenvalues of $G(s)=\text{Re}\{Y(s)\}$ (or $G(s)=\text{Re}\{Z(s)\}$ ) and D are attempted to be made positive by an amount $\beta=\text{TOLG}$ and $\beta=\text{TOLD}$ , respectively.	1E-6 1E-6
TOLG, TOLD	S-parametermodel: Singular values of S(s) and D are attempted to be made smaller than unity by an amount $\beta=\text{TOLG}$ and	1E-6 1E-6



TOLE	$\beta$ =TOLD, respectively Y- and Z- parameter model: Negative egenvalues of E are attempted to be made positive by an amount $\beta$ =TOLE	1E-12
alpha	The required vector of corrections <b>d</b> is replaced with a modified correction: $\mathbf{d}_{\text{new}} = \alpha \mathbf{d} + \beta$	1.0
localviol	=1 --> Will include constraints for all local violations in each violating band (i.e., not only for global violations)	1
usemoreconstraints	Will add extra frequency samples within each violating frequency band, and apply constraints at these samples.	0
nmax	Undocumented. Do not use.	1e16
bw	Controls which residue matrix elements will be perturbed 1 --> will use only diagonal elements 2 --> will use only diagonal elements + 1 element on each side of every diagonal element 3 --> will use only diagonal elements + 2 elements on each side of every diagonal element Etc. (Default 1e16 will include all residue matrix elements)	1e16
SERsubindex	Array of integers, defining index to the pole-residue terms that are to be included in the passivity enforcement. (If not provided, all terms are used).	[]
solver	'nnls' --> Will use solver nnls.m. NOTE: By using 'nnls' you must comply with the conditions set by Bill Whiten, see p.2. 'tntnn' --> Will use tntnn solver. NOTE: By using 'tntnn' you must comply with the license condition on p.2, 'lsqnonl' --> Will use Matlab default solver	'nnls'
<b>Passivity assessment control</b>		
half_size	=1 --> passivity assessment using half-size test matrix =0 --> passivity assessment using Hamiltonian (full-size) test matrix (which is slower).	1
assessbysweeping_s	Array of frequency samples (s=jw) for passivity assessment, instead of test matrix. Is applied whenever the array is non-empty.	[]
colinterch	=1 --> Will recover the correct sequence of eigenvalues (singular values) during passivity assesement.	1

Monitoring progress		
screen	=0 --> Essential output to command window =1 --> Detailed output to command window =2 --> Even more detailed output to command window	0
plot.spy	0 --> No plots are produced 1 --> Plots eigenvalues/singular values before and after passivity enforcement 2 --> Also plots eigenvalues/singular values during the outer-loop iterations 3 --> Also plots eigenvalues/singular values during the inner-loop iterations (if any).	
plot.s_pass	Array of frequency samples (jw). If provided, eigenvalues of G / singular values of S will during iterations be plotted at these frequencies (instead of at frequencies in array s).	
plot.xlim	If provided, the plot of $\text{eig}(\mathbf{G}(s))/\text{sing}(\mathbf{S})$ will be limited in frequency [Hz] to this band. Syntax: [xlow xhigh]	
plot.ylim	If provided, the plot of $\text{eig}(\mathbf{G}(s))$ will be limited in range to this band. Syntax: [ylo yhigh]	
plot.logx	0 --> linear abscissa 1 --> logarithmic abscissa	

### 3.3 netgen\_ATP.m

Auxiliary routine that exports the rational model into a data file for ATP, for representation of a Y-parameter model. The data file contains the branch cards of an equivalent electric circuit. The file can be imported into an ATP data file using the \$INCLUDE feature of ATP. The calculation of the network is explained in [9]. Compared to the earlier version in mtrxfit.zip, the current version makes use of two more digits in the representation of the circuit elements.

The (six-character) node names in the ATP-circuit are

X\_\_\_\_1, X\_\_\_\_2, X\_\_\_\_3, etc

where ‘x’ is a user provided node name, given in input variable NOD.

**netgen\_ATP(SER,NOD,fname)**

**SER**     Structure holding the rational model, as produced by VFdriver or RPdriver

**NOD**     Single character

**fname**   File name where data is to be written.

Example:

NOD='A';

```
fname = 'RLC_ATP.txt';  
netgen_ATP(SER,NOD,fname);
```

### 3.3 *netgen\_EMTP.m*

Auxiliary routine that exports the rational model into a data file for EMTP, for representation of Y-parameter model. The function converts the state space model into a real-only model, and thereafter write the model parameters to file in a format that can be read by EMTP.

**netgen\_EMTP(SER,fname)**

**SER**      Structure holding the rational model, as produced by VFdriver or RPdriver

**fname**    File name where data is to be written. The file extension should be “.mod”.

Example:

```
fname = 'model_EMTP.mod';  
netgen_EMTP(SER,fname);
```

### 3.3 *netgen\_PSCAD.m*

Auxiliary routine that exports the rational model into a data file for PSCAD, for representation of Y-, Z- and S-parameter models. The function converts the state space model into a real-only model, and thereafter write the model parameters to file in a format that can be read by PSCAD, using block “Frequency Dependent Network Equivalent”,

**netgen\_PSCAD(SER,fname,parametertype)**

**SER**      Structure holding the rational model, as produced by VFdriver or RPdriver

**fname**    File name where data is to be written. The file extension could be “.txt”.

**parametertype**    'Y', 'Z', or 'S'.

Example:

```
fname = 'model_PSCAD.txt';  
netgen_PSCAD(SER,fname,'Y');
```

## 4. TUTORIAL

### 4.1 Hints and advice

#### VFdriver.m

##### Parameters in optsVF

- `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'`;

Some times (for instance when fitting a transformer black-box admittance 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'`;
- `passive_DE`. Setting this parameter to unity will request **VFdriver** to enforce **D** and **E** of the rational model to be positive definite if `optsVF.asymp>1`, meaning that the model is asymptotically passive. In general, it is recommended to specify `passive_DE=1`, if passivity is to be enforced. The later passivity enforcement by **RPdriver** then performs better (more robust). You should consider whether the default parameters `passive_DE_TOLD=1e-6` and `passive_DE_TOLE =1e-16` are suitable values.

#### Inaccurate fitting result

If an accurate fitting result cannot be obtained using **VFdriver** 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 data set is not corrupted in some way. The default setting for **VFdriver** is to enforce stable poles (`opts.stable=1`). To diagnose the problem, try to allow unstable poles (`opts.stable=0`) and see if the problem goes away.

#### RPdriver.m

##### Parameters in optsRP

- `Niter_out`. This parameter defines the number of iterations for the *outer* loop in Section 5.2.3. The default value is 20 but it may be necessary to increase this value to get rid of all passivity violations.
- `Niter_in`. This parameter defines the number of iterations for the *inner* loop in Section 5.2.3. The default value is 0 but it may be necessary to increase this value if the passivity enforcement keeps creating new violations.

- `TOLG`, `TOLD`. Y- and Z-parameters: The routine tries to enforce negative eigenvalues of **G**(s) and **D** to become positive by these amounts. S-parameters: The routine tries to enforce singular values of **G**(s) and **D** which are greater than unity to become smaller than unity by these amounts.

The performance of **RPdriver** is dependent on the selected value: using a too small value (close to zero) will often result in an increase of the required number of iterations (due to the nonlinearity of the problem), and the calculated test matrix (if used) may become inaccurate due to near-singularity issues. On the other hand, using a too large value will cause a large perturbation of the model, possibly corrupting model behavior.

- `alpha`. Using a slightly larger value than the default value (`alpha=1`) results in increased step length, often reducing the number of required iterations.
- `solver`. All three available NNLS solvers (`'lsqnonneg'`, `'nnls'`, `'tntnn'`) are active type solvers. For small-scale problems, their performance appears similar. For large scale problems, use `'nnls'` (default), or alternatively `'tntnn'`.

#### Divergence of passivity enforcement

Observe what happens to “Max. violation” in the Matlab command window during iterations. If there is a tendency of divergence, specify a non-zero value for `opts.Niter_in` in order to enable the inner iteration loop (Y-parameters only). It may also help to increase the value of `alpha`, `TOLG`, and `TOLD`.

Also, if the model was created by **VFdriver** using `passive_DE=1`, make sure that `passive_DE_TOLD` is positive and non-zero. Try to increase the value of `passive_DE_TOLD`.

## 4.2 Example 1: Electrical circuit

File: **ex1\_Y.m**

In this small example is demonstrated the usage of **VFdriver** as well as the generation of simulation models for ATP and EMTP.

### 4.2.1 Data case

Consider the electrical network in Fig. 4.2.1 (quantities given in units  $[\Omega]$ ,  $[H]$ ,  $[F]$ ). We wish to calculate a black box model of the equivalent with respect to terminals 1 and 2 when only the frequency response at the terminals are known.

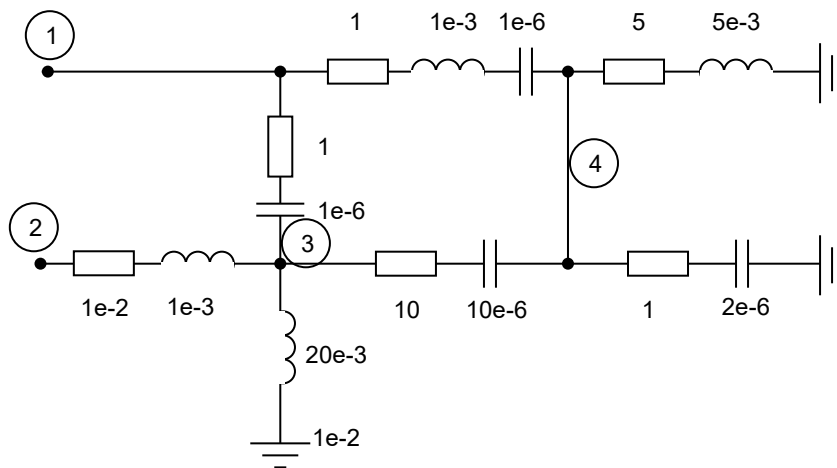


Fig. 4.2.1 Electric circuit

```
% This file is part of the MFT_NNLS Toolbox, v1.
% Filename: ex1_Y.m
% Package: MFT_NNLS_1.zip.
% Programmed by B. Gustavsen. January 11, 2026.
%
clear all

Ns=501;      %Number of frequency samples
Nc=2;        %Size of Y (after reduction)
bigY=zeros(Nc,Nc,Ns);
Y=zeros(4,4);
s=2*pi*i*logspace(1,5,Ns);

%Component values:
R1=1;        L1=1e-3; C1=1e-6;
R2=5;        L2=5e-3;
R3=1;        C3=1e-6;
L4=1e-3;
R4=1e-2;     L5=20e-3;
R6=10;       C6=10e-6;
R7=1;        C7=2e-6;

%Building Y, reduction:
```

```

for k=1:Ns
    sk=s(k);
    y1=1/( R1+sk*L1+1/(sk*C1) );
    y2=1/( R2+sk*L2 );
    y3=1/( R3+1/(sk*C3) );
    y4=1/( R4+sk*L4 );
    y5=1/(sk*L5);
    y6=1/( R6+1/(sk*C6) );
    y7=1/( R7+1/(sk*C7) );

    Y(1,1)= y1+y3;
    Y(2,2)= y4;
    Y(3,3)= y3 +y4 +y5 +y6;
    Y(4,4)= y1 +y2 +y6 +y7;

    Y(1,3)=-y3; Y(1,4)=-y1;
    Y(2,3)=-y4;
    Y(3,1)=-y3; Y(3,2)=-y4; Y(3,4)=-y6;
    Y(4,1)=-y1; Y(4,3)=-y6;

    %Eliminating nodes 3 and 4:
    Yred=Y(1:2,1:2)-Y(1:2,3:4)*Y(3:4,3:4)^(-1)*Y(3:4,1:2);
    bigY(:, :, k)=Yred;
end

%=====
%          POLE-RESIDUE FITTING          =
%=====
optsVF.N=8; %Order of approximation. (Is used when optsVF.poles=[]).
optsVF.poletype='logcmplx'; %Will use logarithmically spaced, complex poles. (Is used
because optsVF.poles=[]).
poles=[]; %[] -->N initial poles are automatically generated as defined by
optsVF.startpoleflag
optsVF.phaseplot=1;
optsVF.passive_DE=0;

%Calling VFdriver:
[SER,rmserr,bigYfit,optsVF2]=VFdriver(bigY,s,poles,optsVF); %Creating state-space model and
pole-residue model

```

## 4.2.2 Running the example

Running **ex1\_Y.m** from Matlab's command window gives the output to screen shown below.

```

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
****Vector Fitting upper triangle elements ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
Elapsed time is 0.072011 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...

```

```

rmserr:      2.3125e-16
rmserrweight: 2.3125e-16
----- E N D -----

****Creating equivalent circuit for ATP...

```

>> The result in Fig. 4.2.2 shows that the approximation is highly accurate as the deviation is close to machine precision. (The plot is automatically generated).

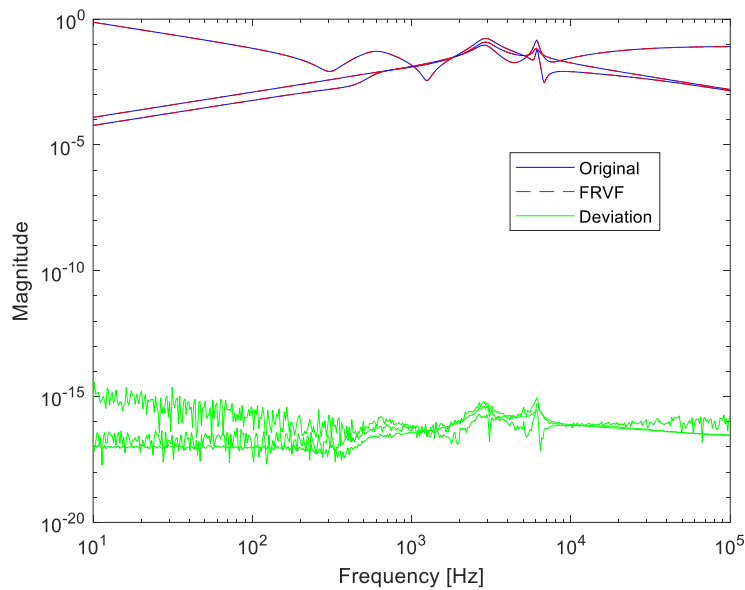


Fig. 4.2.2 Generated plot of rational fitting.

In addition to the parameters specified in the input `optsVF`, there several other parameters that are set internally in `vfdriver.m`. All settings that were used are returned in structure `optsVF2`. Using input array `optsVF`, all default settings can be changed, see Section 3.1.

```

optsVF2 =

struct with fields:

    parametertype: 'Y'
              N: 8
    poletype: 'logcmplx'
              nu: 1.0000e-03
              Niter1: 4
              Niter2: 4
              asymp: 2
    cmplx_ss: 1
    weight: []
    weightparam: 1
    stable: 1
    relaxed: 1
    NE: 0
    plot: 1
    logx: 1
    logy: 1
    errplot: 1
    phaseplot: 1
    screen: 1
    remove_HFpoles: 0
    factor_HF: 1.1000

```



```
passive_DE: 1
passive_DE_TOLD: 1.0000e-06
passive_DE_TOLE: 1.0000e-16
```

### Pole-residue model:

```
>> SER.poles
ans =
-4.7619e-001
-1.2876e+005
-1.0229e+003 +3.5994e+003i
-1.0229e+003 -3.5994e+003i
-2.2888e+003 +1.8044e+004i
-2.2888e+003 -1.8044e+004i
-1.0116e+003 +3.8290e+004i
-1.0116e+003 -3.8290e+004i

>> SER.R
ans(:, :, 1) =
-1.7930e-014 -2.1595e-007
-2.1595e-007 4.7619e+001
ans(:, :, 2) =
-1.0019e+004 -5.3834e+002
-5.3834e+002 -2.8926e+001
ans(:, :, 3) =
2.1958e-001 +2.2124e-001i 3.7368e+000 +1.9303e+000i
3.7368e+000 +1.9303e+000i 5.5986e+001 +9.2914e+000i
ans(:, :, 4) =
2.1958e-001 -2.2124e-001i 3.7368e+000 -1.9303e+000i
3.7368e+000 -1.9303e+000i 5.5986e+001 -9.2914e+000i
ans(:, :, 5) =
1.8288e+002 +6.5934e+001i -2.6855e+002 -7.6004e+001i
-2.6855e+002 -7.6004e+001i 3.9227e+002 +8.1793e+001i
ans(:, :, 6) =
1.8288e+002 -6.5934e+001i -2.6855e+002 +7.6004e+001i
-2.6855e+002 +7.6004e+001i 3.9227e+002 -8.1793e+001i
ans(:, :, 7) =
1.3290e+002 +1.8383e+001i 7.5653e+001 +9.8304e-001i
7.5653e+001 +9.8304e-001i 4.2401e+001 -4.7459e+000i
ans(:, :, 8) =
1.3290e+002 -1.8383e+001i 7.5653e+001 -9.8304e-001i
7.5653e+001 -9.8304e-001i 4.2401e+001 +4.7459e+000i

>> SER.D
ans =
8.3333e-02 2.6567e-17
2.6567e-17 -2.2524e-17

>> SER.E
ans =
0 0
0 0
```

### State-space model:

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

opts.cmplx\_ss=1.

```
>> SER.A
ans =
(1,1) -4.7619e-001
(2,2) -1.2876e+005
(3,3) -1.0229e+003 +3.5994e+003i
(4,4) -1.0229e+003 -3.5994e+003i
(5,5) -2.2888e+003 +1.8044e+004i
(6,6) -2.2888e+003 -1.8044e+004i
(7,7) -1.0116e+003 +3.8290e+004i
(8,8) -1.0116e+003 -3.8290e+004i
(9,9) -4.7619e-001
(10,10) -1.2876e+005
(11,11) -1.0229e+003 +3.5994e+003i
```

```

(12,12)  -1.0229e+003 -3.5994e+003i
(13,13)  -2.2888e+003 +1.8044e+004i
(14,14)  -2.2888e+003 -1.8044e+004i
(15,15)  -1.0116e+003 +3.8290e+004i
(16,16)  -1.0116e+003 -3.8290e+004i

>> SER.B

ans =

     1     0
     1     0
     1     0
     1     0
     1     0
     1     0
     1     0
     1     0
     0     1
     0     1
     0     1
     0     1
     0     1
     0     1
     0     1
     0     1
     0     1

>> (SER.C).'
ans =
-1.7930e-014          -2.1595e-007
-1.0019e+004          -5.3834e+002
 2.1958e-001 +2.2124e-001i   3.7368e+000 +1.9303e+000i
 2.1958e-001 -2.2124e-001i   3.7368e+000 -1.9303e+000i
 1.8288e+002 +6.5934e+001i  -2.6855e+002 -7.6004e+001i
 1.8288e+002 -6.5934e+001i  -2.6855e+002 +7.6004e+001i
 1.3290e+002 +1.8383e+001i   7.5653e+001 +9.8304e-001i
 1.3290e+002 -1.8383e+001i   7.5653e+001 -9.8304e-001i
-2.1595e-007          4.7619e+001
-5.3834e+002          -2.8926e+001
 3.7368e+000 +1.9303e+000i   5.5986e+001 +9.2914e+000i
 3.7368e+000 -1.9303e+000i   5.5986e+001 -9.2914e+000i
-2.6855e+002 -7.6004e+001i   3.9227e+002 +8.1793e+001i
-2.6855e+002 +7.6004e+001i   3.9227e+002 -8.1793e+001i
 7.5653e+001 +9.8304e-001i   4.2401e+001 -4.7459e+000i
 7.5653e+001 -9.8304e-001i   4.2401e+001 +4.7459e+000i

```

The **D** and **E** matrices are the same as for the pole-residue model.

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

```

(1,1)  -4.7619e-001
(2,2)  -1.2876e+005
(3,3)  -1.0229e+003
(4,3)  -3.5994e+003
(3,4)   3.5994e+003
(4,4)  -1.0229e+003
(5,5)  -2.2888e+003
(6,5)  -1.8044e+004
(5,6)   1.8044e+004
(6,6)  -2.2888e+003
(7,7)  -1.0116e+003
(8,7)  -3.8290e+004
(7,8)   3.8290e+004
(8,8)  -1.0116e+003
(9,9)  -4.7619e-001
(10,10) -1.2876e+005
(11,11) -1.0229e+003
(12,11) -3.5994e+003
(11,12)   3.5994e+003
(12,12)  -1.0229e+003
(13,13) -2.2888e+003

```

```
(14,13)  -1.8044e+004
(13,14)   1.8044e+004
(14,14)  -2.2888e+003
(15,15)  -1.0116e+003
(16,15)  -3.8290e+004
(15,16)   3.8290e+004
(16,16)  -1.0116e+003
```

```
>> SER.B
```

```
ans =
```

```
1      0
1      0
2      0
0      0
2      0
0      0
2      0
0      0
0      1
0      1
0      2
0      0
0      2
0      0
0      2
0      0
```

```
>> (SER.C) .'
```

```
ans =
```

```
-1.7930e-014 -2.1595e-007
-1.0019e+004 -5.3834e+002
 2.1958e-001  3.7368e+000
 2.2124e-001  1.9303e+000
 1.8288e+002 -2.6855e+002
 6.5934e+001 -7.6004e+001
 1.3290e+002  7.5653e+001
 1.8383e+001  9.8304e-001
-2.1595e-007  4.7619e+001
-5.3834e+002 -2.8926e+001
 3.7368e+000  5.5986e+001
 1.9303e+000  9.2914e+000
-2.6855e+002  3.9227e+002
-7.6004e+001  8.1793e+001
 7.5653e+001  4.2401e+001
 9.8304e-001 -4.7459e+000
```

## 4.2.3 Generation of simulation model for ATP

**ex1\_y.m** calls **netgen\_ATP.m** :

```
NOD='A';
fname = 'RLC_ATP.txt';
netgen_ATP(SER,NOD,fname); %Creating branch-cards for ATP
```

This produces a file **RLC\_ATP.txt** with the following contents:

```
$VINTAGE,1
C <BUS1><BUS2><BUS3><BUS4>< OHM >< milliH >< microF >
C
$VINTAGE,1
C <BUS1><BUS2><BUS3><BUS4>< OHM >< milliH >< microF >
C
C (1,1)
A ____1 1.20000000e+001
A ____1 3.00000000e-010
A ____1 -2.20505113e+006-4.63060716e+009
A ____1 -1.21959492e+001-9.47194746e-002
A ____1A 3__1 3.76645764e+002 1.26376926e+002
A 3__1 -2.27028923e+003
A 3__1 4.71374555e-001
A ____1A 5__1 -2.57338708e+001-5.83599210e+000
A 5__1 -1.14565657e+004
A 5__1 -5.19129996e-001
A ____1A 7__1 1.09495943e+001 2.39745382e+000
A 7__1 -1.39356194e+003
A 7__1 2.82071301e-001
C (1,2)
A ____1A ____2 -8.60370102e+015
A ____1A ____2 -1.00000000e-010
A ____1A ____2 2.20505112e+006 4.63060713e+009
A ____1A ____2 2.39175279e+002 1.85754765e+000
A ____1A 3__12 -3.85654189e+002-1.33803037e+002
A 3__12A ____2 2.62569679e+003
A 3__12A ____2 -4.55368132e-001
A ____1A 5__12 1.37691254e+001 1.86184206e+000
A 5__12A ____2 -2.32353531e+002
A 5__12A ____2 1.52736719e+000
A ____1A 7__12 -9.97399817e+000-6.60910817e+000
A 7__12A ____2 -1.88522825e+004
A 7__12A ____2 -1.03186254e-001
C (2,2)
A ____2 3.25513438e+015
A ____2 3.00000000e-010
A ____2 9.99999910e-003 2.09999971e+001
A ____2 -2.26979329e+002-1.76282818e+000
A ____2A 3__2 1.42263095e+001 8.37207251e+000
A 3__2 3.23949426e+002
A 3__2 8.90531215e+000
A ____2A 5__2 1.26628586e+001 4.04153239e+000
A 5__2 9.12933873e+002
A 5__2 7.58320546e-001
A ____2A 7__2 -8.84566299e-001 4.23533375e+000
A 7__2 2.78476100e+003
A 7__2 1.60882715e-001
$VINTAGE,0
```

Nodes **A\_\_\_\_1** and **A\_\_\_\_2** represent the terminals of the equivalent.

This circuit is conveniently imported into ATP-EMTP using the **\$INCLUDE** statement imbedded in an ATP data file.

Consider the situation that a unit step voltage is applied to terminal 1 and we wish to calculate the resulting current flowing into terminal 2, see Fig. 4.2.3.

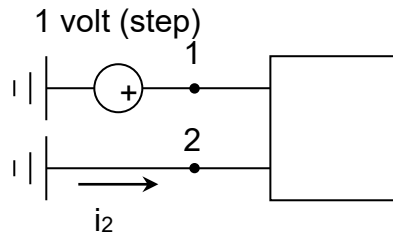


Fig. 4.2.3. ATP-simulation.

An ATP-file for the simulation case in Fig. 4.2.3 is shown below, given in file `circuit.atp`. (A  $1\ \mu\Omega$  resistor is used for the current measurement). Note that the equivalent electrical network is imported using the `$INCLUDE` statement, and that the node names are `A___1` and `A___2`. (To run this ATP-case, be sure to update the path statement in `circuit.atp`).

```
BEGIN NEW DATA CASE
C
C   Example described in user_manual.pdf
C
C   =====
C   =   File:   circuit.atp   =
C   =   Version 1.0           =
C   =   Last revised: 19.03.2002   =
C   =   Programmed by: Bjorn Gustavsen,   =
C   =   SINTEF Energy Research, N-7465 Trondheim, NORWAY =
C   =   This file is part of the "matrixfitter-package" =
C   =====
C
C $DUMMY, XYZ000
C
C
C FIRST MISCELLANEOUS DATA CARD
C $DUMMY, XYZ000
C FIRST MISCELLANEOUS DATA CARD
C dT >< Tmax >< Xopt >< Copt >
C 1.000E-6 .005
C
C SECOND MISCELLANEOUS DATA CARD
C 1-8 9-16 17-24 25-32 33-40 41-48 49-56 57-64 65-72 73-80
C PRINT PLOT NETWORK PR.SS PR.MAX I PUN PUNCH DUMP MULT. DIAGNOS
C 0=EACH 0=EACH 0= NO 0= NO 0= NO 0= NO 0= NO INTO ENERG. PRINT
C K=K-TH K=K-TH 1=YES 1=YES 1=YES 1=YES 1=YES DISK STUDIES 0=NO
C 10000 1 0 0 1
C
C BRANCHES
C 3456789012345678901234567890123456789012345678901234567890
C 3-8 9-14 15-20 21-26 27-32 33-38 39-44
C NODE NAMES REFERENCE RES. IND. CAP. (OUTPUT IN COLUMN 80)
C BRANCH MH UF I= 1
C <BUS1><BUS2><BUS3><BUS4> OHM OHM UMHO V= 2
C I.V 3
C A___2 1.E-6 1
$INCLUDE C:\user\mtrx_fitter_new\RLC_ATP.txt
C
C BLANK CARD TERMINATING BRANCH CARDS
C SWITCH CARDS
C 3456789012345678901234567890123456789012345678901234567890
C 3-8 9-14 15-24 25-34 35-44 45-54 55-64 65-74
C (OUTPUT OPTION IN COLUMN 80)
C NODE NAMES IE FLASHOVER SPECIAL REFERENCE
C TIME TO TIME TO OR VOLTAGE REQUEST SWITCH-NAME
C BUS1 BUS2 CLOSE OPEN NSTEP WORD BUS5 BUS6
C
C BLANK CARD TERMINATING SWITCH CARDS
```

```

C SOURCE CARDS
C 34567890123456789012345678901234567890123456789012345678901234567890
C COLUMN 1.2: TYPE OF SOURCE 1 17.(E.G. 11-13 ARE RAMP FUNCTIONS. 14 = COSINE)
C COLUMN 9.10: 0=VOLTAGE SOURCE. 1=CURRENT SOURCE
C 3-8 11-20 21-30 31-40 41-50 51-60 61-70 71-80
C NODE AMPLITUDE FREQUENCY TO IN SEC AMPL-A1 TIME-T1 T-START T-STOP
C NAME IN HZ DEGR SECONDS SECONDS SECONDS
14A 1 1.0 0.001 0. 0. 0. 1.
C
BLANK CARD TERMINATING SOURCE CARD
C NODE VOLTAGE OUTPUT
C 34567890123456789012345678901234567890123456789012345678901234567890
C 3-8 9-14 15-20 21-26 27-32 33-38 39-44 45-50 51-56 57-62 63-68 69-74 75-80
C BUS1 BUS2 BUS3 BUS4 BUS5 BUS6 BUS7 BUS8 BUS9 BUS10 BUS11 BUS12 BUS13
C A 1A 2
C NBY1A NBY1B NBY1C NBY6A NBY6B NBY6C NEUT NBY6G GENA GENB GENC
BLANK CARD TERMINATING OUTPUT CARDS
BLANK CARD ENDING PLOT CARDS
BEGIN NEW DATA CASE

```

At the end of ex1\_Y.m, the theoretical setp response is calculated:

```

% Plotting step response:
NN=length(SER.A) ; I=ones(NN,1);
t=(0:1e-5:5e-3); Nt=length(t);
for k=1:Nt
    if opts2.cmplx_ss==1
        y=SER.C*diag( diag(SER.A).^(-1).*(exp(diag(SER.A).*t(k))-I) ) *SER.B +SER.D;
    else
        y=SER.C*( (SER.A)^(-1))*((expm((full(SER.A)).*t(k))-diag(I)) ) *SER.B +SER.D;
    end
    yy(k)=y(2,1);
end
figure(4),plot(1000*t,yy);
xlabel('Time [ms]'); ylabel('Current [A]');

```

Fig. 4.2.4 Compares the theoretical solution with the ATP simulation result.

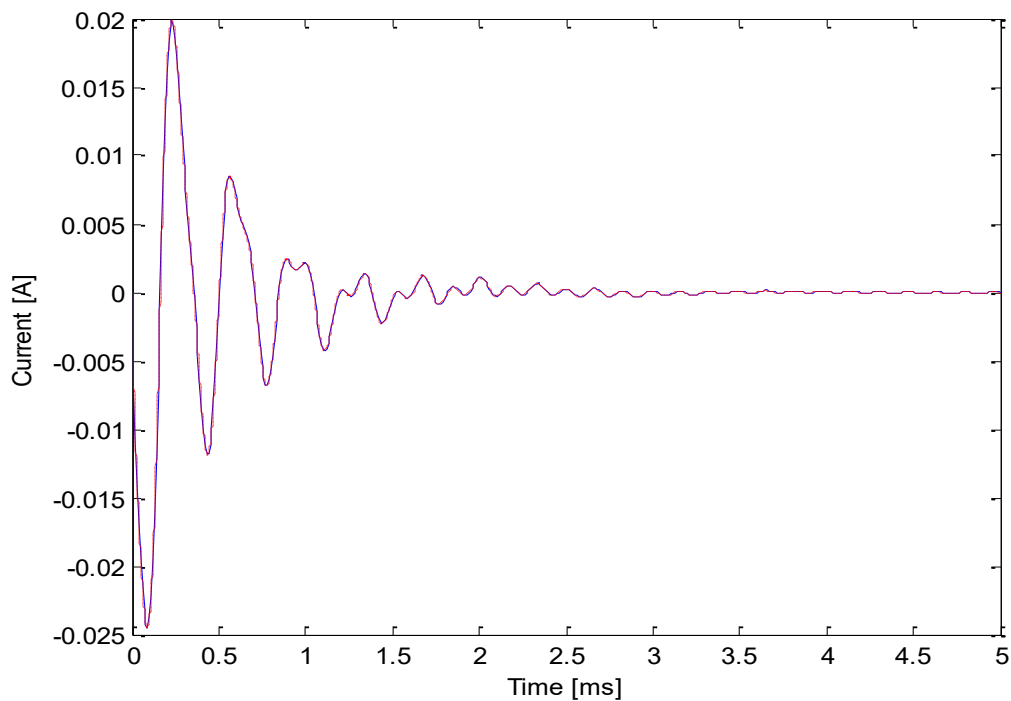


Fig. 4.2.4 Step voltage response. Blue: theoretical, red: ATP simulation.

## 4.3 Example 2: Network equivalencing

File: **ex2\_Y.m**

In this second example we will demonstrate both rational fitting (**VFdriver.m**) and passivity enforcement (**RPdriver.m**).

### 4.3.1 Generation of frequency domain data

We consider a three-conductor overhead line of length 12 km, see Fig. 4.3.1. The admittance matrix **Y** is computed with respect to one line end with other line end open circuited.

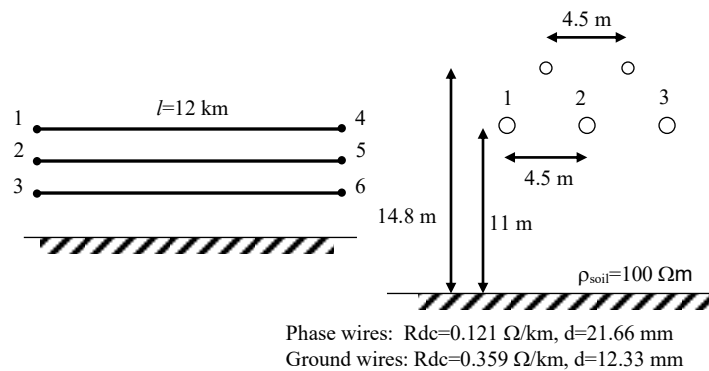


Fig. 4.3.1. 132 kV overhead line.

On top of **ex2\_Y.m**, the example case is loaded from file,

```
load fdne %-->s, bigY
```

### 4.3.2 Rational fitting

In **ex2\_Y.m**, the following call creates the a 50<sup>th</sup> order rational model of the 3×3 **Y**.

```
%=====
%          POLE-RESIDUE FITTING          =
%=====
optsVF.N=50 ;%          %Order of approximation.
optsVF.poletype='linlogcmplx'; %Mix of linearly spaced and logarithmically spaced poles
optsVF.weightparam=5; % 5--> weighting with inverse matrix norm
optsVF.Niter1=7;      %Number of iterations for fitting diagonal elements (fast!)
optsVF.Niter2=4;      %Number of iterations for matrix fitting
optsVF.asymp=2;       %Fitting includes D
optsVF.logx=0;        %=0 --> Plotting is done using linear abscissa axis
poles=[];

%Calling VFdriver:
[SER,rmserr,bigYfit,optsVF2]=VFdriver(bigY,s,poles,optsVF);
```

```

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
****Vector Fitting upper triangle elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
****Enforcing positive eigenvalues for D...
  --Refitting residues...
Elapsed time is 0.182378 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...
      rmserr:      6.5187e-05
      rmserrweight: 0.00079919
----- E N D -----

```

The magnitude plot is shown in Fig. 4.3.2. The order is in this case slightly too low so that an inaccurate fitting results locally.

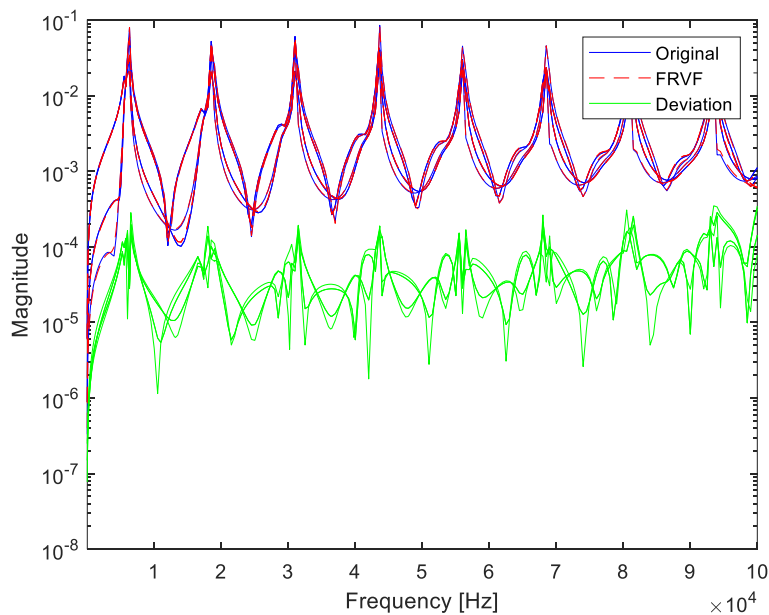


Fig. 4.3.2. Rational fitting.



### 4.3.3 Passivity enforcement

The inaccurate fitting results in that the obtained model is non-passive. In `ex2_Y.m`, the following enforces passivity,

```
%=====
%=          Passivity Enforcement          =
%=====

optsRP.parametertype='Y';

optsRP.plot.s_pass=2*pi*i*linspace(0,2e5,1001).';
optsRP.plot.ylim=[-2e-3 2e-3];
optsRP.plot.spy=2;
optsRP.plot.logx=0;
optsRP.outputlevel=1;
optsRP.weightparam=5;
optsRP.screen=0;

%Calling RPdriver:
[SER2,bigYfit_passive,optsRP2]=RPdriver(SER,s,optsRP);
```

This gives an output to the screen as shown below.

```
----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (Y-parameters) ***
  Initial model : Max. violation : -0.0028402
  Iteration  1  : Max. violation : -1.2165e-05
  Iteration  2  : Max. violation : None
-->Passivity was successfully enforced.
    rmserr:      7.3934e-06
    rmserrweight: 4.5999e-05

Time summary:
  Passivity assessment : 0.055137 sec
  Passivity enforcement: 0.067996 sec
      Build A (LS):    0.035454 sec
      Build C       :    0.0073096 sec
      Build E       :    0.0013845 sec
      Solve NNLS    :    0.0060732 sec
  Total: 0.12778 sec
----- E N D -----
```

The eigenvalues of  $\mathbf{G}=\text{real}(\mathbf{Y})$  are plotted during the iterations, since the parameter `optsRP.plot.spy` was provided in the call with value  $>0$ . Fig. 4.3.3 shows the final result. It is observed that the eigenvalues are all positive after the passivity enforcement step.

(The thick, black horizontal line indicates the “fitting range”, i.e., the frequencies in array  $s$ ).

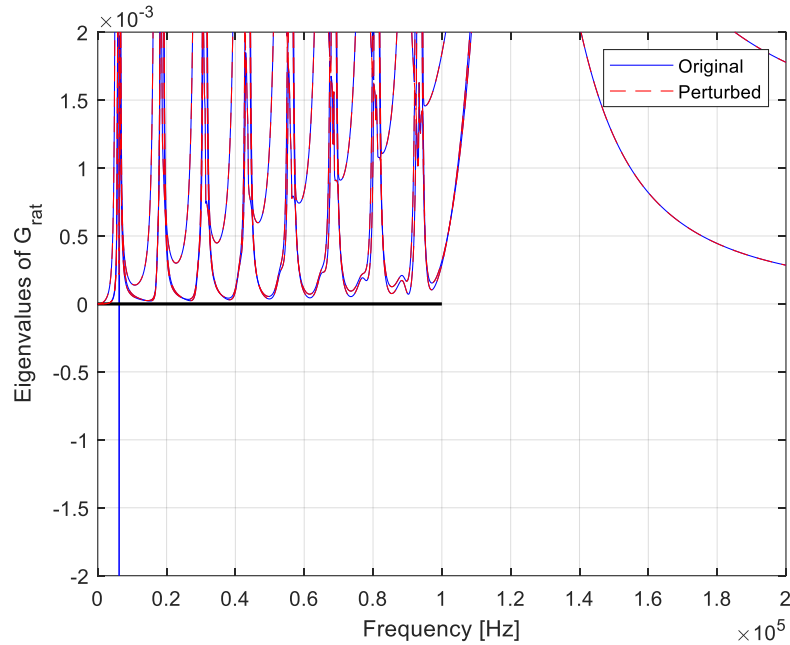


Fig. 4.3.3 Eigenvalues of  $G(s)$ .

You can check on all settings that were used in the passivation process via the returned output structure `optsRP2`. (These values can be changed via the input structure `optsRP`).

```
>> optsRP2
```

```
struct with fields:
```

```

    parametertype: 'Y'
        Niter_out: 20
        Niter_in: 0
        continue: 0
        cmplx_ss: 1
        weightparam: 5
        weightarray: []
        weightfactor: 1.0000e-03
        TOLG: 1.0000e-06
        TOLD: 1.0000e-06
        TOLE: 1.0000e-12
        alpha: 1
        localviol: 1
    usemoreconstraints: 0
        nmax: 1.0000e+16
        bw: 1.0000e+16
        SERsubindex: [1 2 3 4 5 6 7 8 9 10 11 12 13 ... ] (1x50 double)
        solver: 'nnls'
        halfsize: 1
    assessbysweeping_s: []
        colinterch: 1
        screen: 0
        plot: [1x1 struct]
        solverdata: [1x1 struct]
        oldDflag: 0
        oldEflag: 0

```

(Note: `oldDflag` and `oldEflag` are used internally only, and are not to be specified).

At the end of the file (`ex2_Y.m`), the original model is compared with the perturbed model (at samples defined by array `s` in call to `RPdriver.m`), see Fig. 4.3.4.

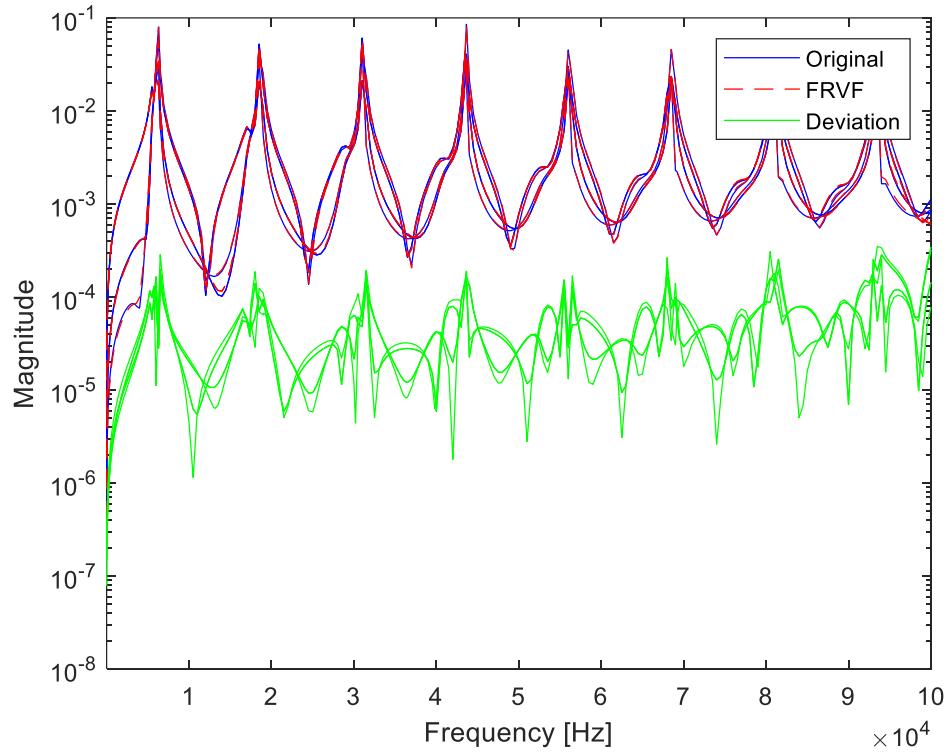


Fig. 4.3.4 The effect of perturbation on the admittance matrix,  $\mathbf{Y}$ .

## 4.4 Example 3: Network equivalencing of transmission line

File: **ex3\_Y.m**

This is a six-port equivalent of the transmission line in Fig. 4.3.1, with a line length of 45 km. The fitting range is 10 Hz to 10 kHz, using 30 pole-residue terms.

In this example we demonstrate usage of user-supplied frequency samples for passivity enforcement, provided in array “`optsRP.assessbysweeping_s`”.

```
clear all

load ex3      %--> s, SER

%=====
%=          Passivity Enforcement          =
%=====
optsRP.Niter_out=20;
optsRP.Niter_in=2;

optsRP.plot.s_pass=2*pi*i*linspace(0,3e4,1001).';
optsRP.plot.ylim=[-2e-3 2e-3];
optsRP.plot.spy=1;
optsRP.plot.logx=0;

optsRP.assessbysweeping_s=2*pi*i.*sort([linspace(0,1e6,1001) logspace(-4,7,1001)]);
%optsRP.assessbysweeping_s=[]; --> passivity assessment using test matrix

%optsRP.solver='lsqnonneg';
optsRP.solver='nnls';
%optsRP.solver='tntnn';

%Calling RPdriver:
[SER2,bigYfit_passive,optsRP2]=RPdriver(SER,s,optsRP);

----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (Y-parameters) ***
  Initial model : Max. violation : -0.0052095
  Iteration  1  : Max. violation : -7.861e-05
  Iteration  2  : Max. violation : None
-->Passivity was successfully enforced.
  rmserr:      2.6904e-06
  rmserrweight: 2.6904e-06

Time summary:
  Passivity assessment : 0.31692 sec
  Passivity enforcement: 0.15192 sec
      Build A (LS):    0.02706 sec
      Build C       :    0.024429 sec
      Build E       :    0.0020773 sec
      Solve NNLS    :    0.046258 sec
  Total: 0.50258 sec
----- E N D -----
```

- Fig. 4.4.1 shows plot of the eigenvalues of  $\mathbf{G}(s)$ , as generated by `RPdriver.m`. It is seen that the perturbation makes the eigenvalues positive.
- Fig. 4.4.2 shows the plot of the elements of  $\mathbf{Y}(s)$ , as generated by `ex3_Y.m`. It is seen that the change to the elements is very small inside the fitting band (10 Hz – 10 kHz).

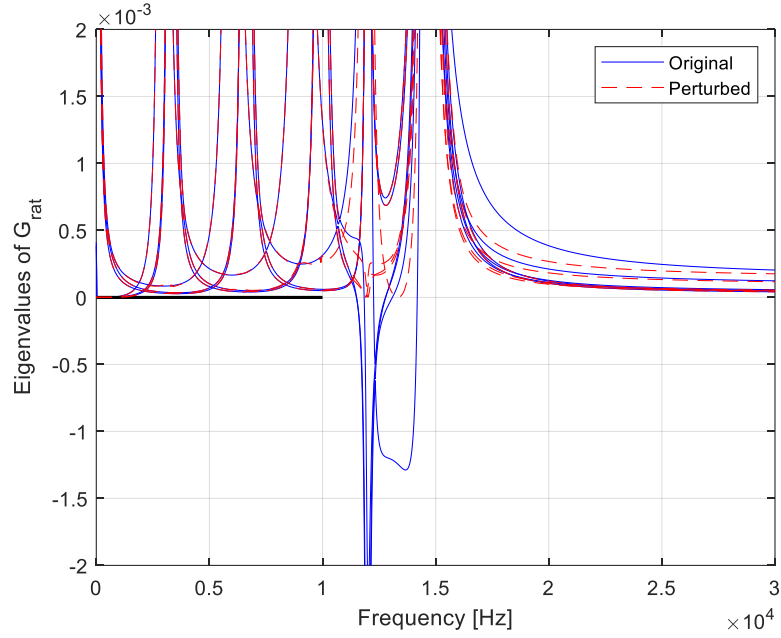


Fig. 4.4.1. Eigenvalues of  $\mathbf{G}(s)$ .

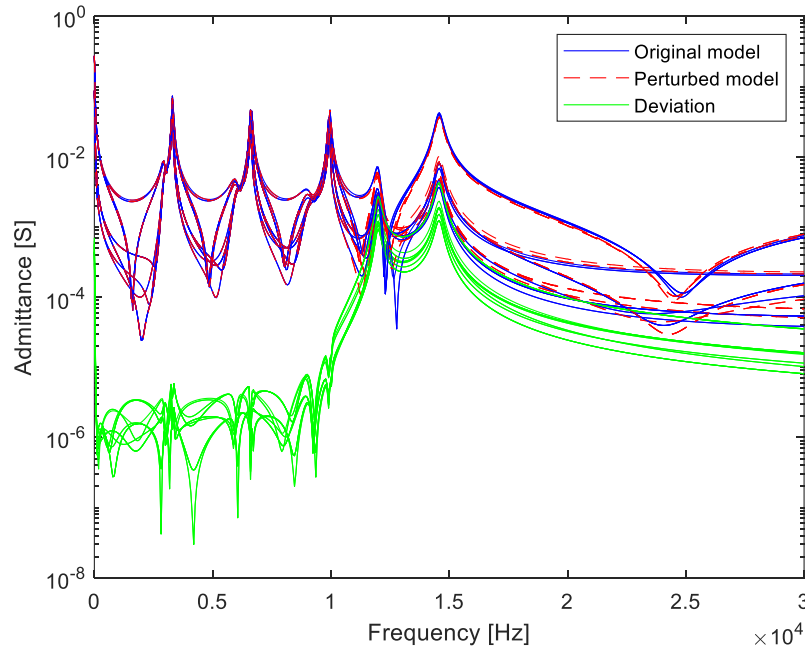


Fig. 4.4.2. The effect of perturbation on the admittance matrix,  $\mathbf{Y}(s)$ .

## 4.5 Example 4: Network equivalencing: S-parameters

This is the same example as in Section 4.2 (Example 2), but with the Y-parameters converted to S-parameters before carrying out the rational fitting. The example loads the S-parameters and calculates a rational model using **VFdriver**. In the subsequent passivity enforcement, no inner-loop iterations are used. We use inverse element magnitude weighting, in both VF and in passivity enforcement (`optsVF.weightparam=2, optsRP.weightparam=2`);

### 4.5.1 Rational fitting

```
%=====
%=          POLE-RESIDUE FITTING          =
%=====
optsVF.N=50 ;%                %Order of approximation.
optsVF.poletype='linlogcmplx'; %Mix of linearly spaced and logarithmically spaced poles
optsVF.Niter1=7;    %Number of iterations for fitting dikagonal elements (fast!)
optsVF.Niter2=4;    %Number of iterations for matrix fitting
optsVF.asymp=2;     %Fitting includes D
optsVF.logx=1;      %=0 --> Plotting is done using linear abscissa axis
poles=[];
optsVF.parametertype='S';
optsVF.weightparam=2;

%Calling VFdriver:
[SER,rmserr,bigSfit,optsVF2]=VFdriver(bigS,s,poles,optsVF); %Creating state-space model and
pole-residue model

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
    Iter 5
    Iter 6
    Iter 7
****Vector Fitting upper triangle elements ...
    Iter 1
    Iter 2
    Iter 3
    Iter 4
****Enforcing singular values of D to be smaller than unity...
    ---Found none.
Elapsed time is 0.168380 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...
    rmserr:      0.011192
    rmserrweight: 0.018853
----- E N D -----
```

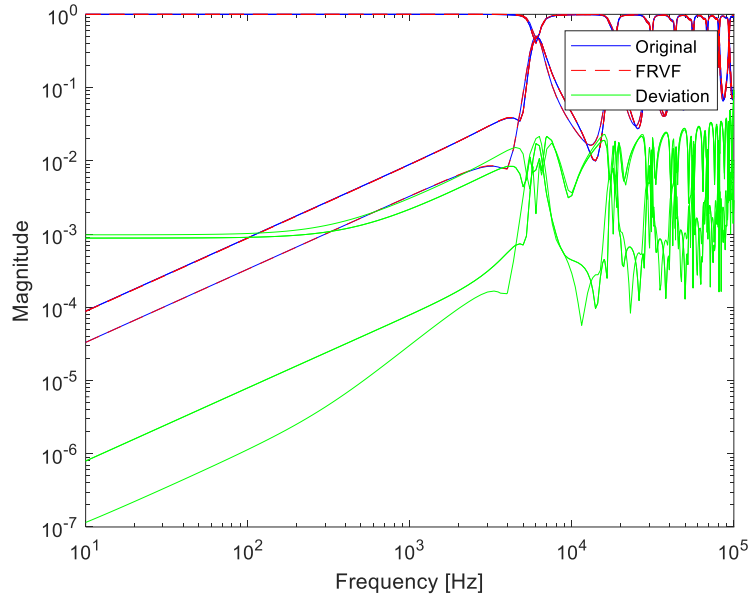


Fig. 4.5.1. Rational fitting.

## 4.5.2 Passivity enforcement

```
%=====
%=          Passivity Enforcement          =
%=====
optsRP.Niter_out=20;
optsRP.Niter_in=0;
optsRP.parametertype='S';
optsRP.weightparam=2;
optsRP.screen=0; %Min. output to screen

optsRP.plot.s_pass=2*pi*i*linspace(0,2e5,1001).';
optsRP.plot.ylim=[0.95 1.05];
optsRP.plot.spy=2; %0
optsRP.plot.logx=0;

%Calling RPdriver:
[SER2,bigSfit_passive,optsRP2]=RPdriver(SER,s,optsRP);
```

```
----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (S-parameters) ***
  Initial model : Max. violation : 0.038047
  Iteration 1   : Max. violation : 0.0018645
  Iteration 2   : Max. violation : 1.4005e-05
  Iteration 3   : Max. violation : None
-->Passivity was successfully enforced.
  rmserr:      0.0042185
  rmserrweight: 0.0045548

Time summary:
  Passivity assessment : 0.13701 sec
  Passivity enforcement: 0.11188 sec
    Build A (LS):      0.04076 sec
    Build C           : 0.032642 sec
    Build E           : 0.002665 sec
    Solve NNLS        : 0.011608 sec
  Total: 0.26732 sec
----- E N D -----
```

Fig. 4.5.2 shows that the singular values are enforced to be smaller than unity. The change to the model's behavior is small as seen in Fig. 4.5.2.

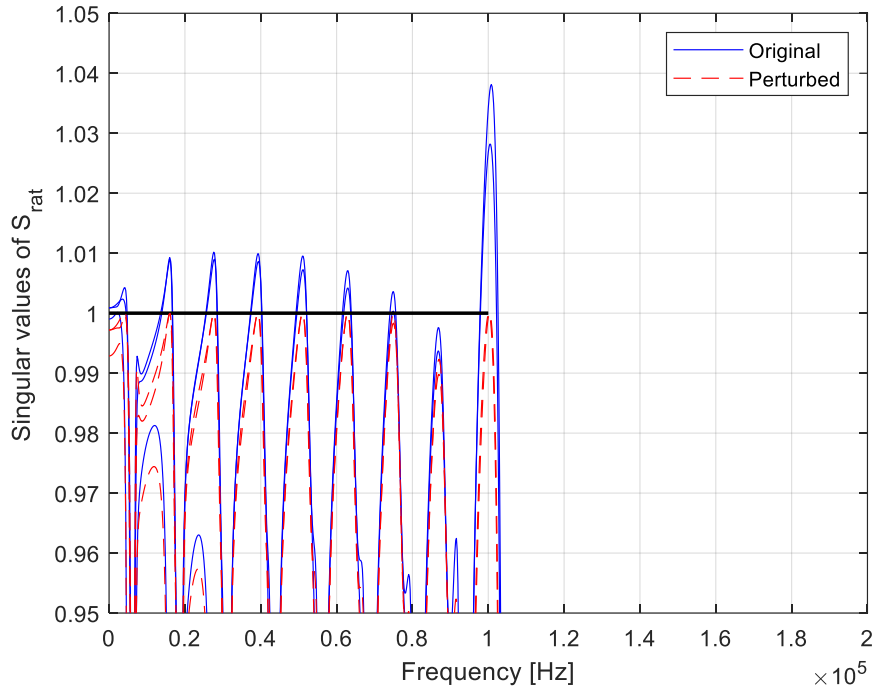


Fig. 4.5.2. Singular values of  $S$ .

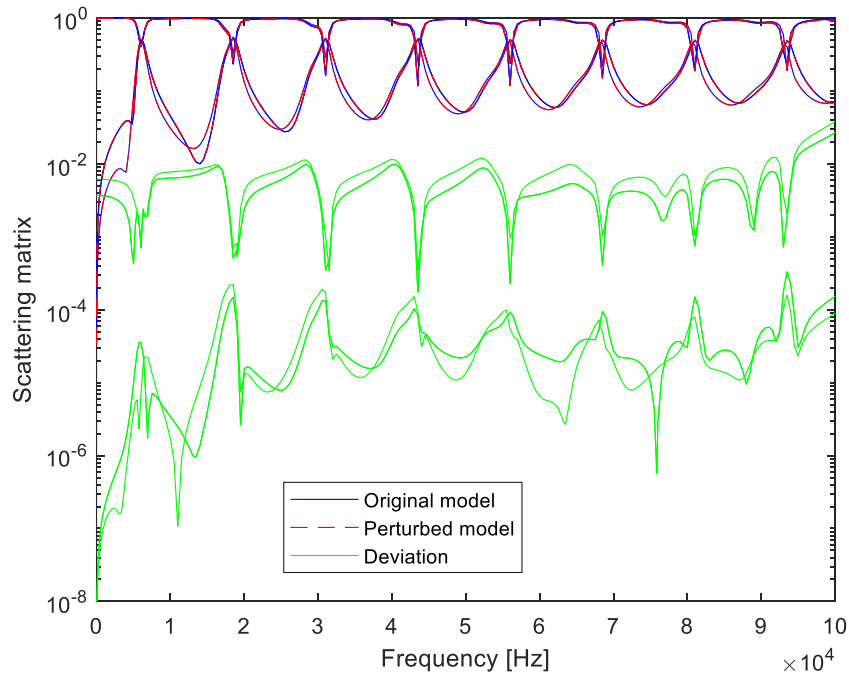


Fig. 4.5.2. The effect of perturbation on elements of  $S$ .



## 5. REFERENCE FOR COMPUTATIONAL APPROACH

### 5.1 Pole-residue modeling by Vector Fitting

#### 5.1.1 General

##### Y-parameters

Given a symmetric admittance matrix  $\mathbf{Y}$  as function of frequency ( $s, \mathbf{Y}(s)$ ), the objective is to calculate a pole-residue model (5.1.1) which approximates (“fits”) the original data as close as possible. ( $\mathbf{D}$  and  $\mathbf{E}$  are possibly zero)

$$\mathbf{Y}(s) \approx \mathbf{Y}_{rat}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} + s\mathbf{E} \quad (5.1.1)$$

In the case of a physical system, the following properties apply

1.  $\mathbf{Y}$  is a symmetrical matrix. Hence,  $\{\mathbf{R}_m\}$ ,  $\mathbf{D}$ , and  $\mathbf{E}$  are symmetric
2.  $\mathbf{D}$  and  $\mathbf{E}$  are real matrices
3. The poles and residues are either real or come in complex conjugate pairs (causality requirement)
4. The poles are in the left half plane
5. The model is passive, i.e. it cannot generate energy. For the symmetrical model, this implies that

$$\text{eig}(\text{Re}\{\mathbf{Y}_{rat}(s)\}) > 0 \quad (5.1.2)$$

$$\text{eig}(\mathbf{D}) > 0 \quad (5.1.3)$$

6. The capacitance matrix  $\mathbf{E}$  is positive, i.e.

$$\text{eig}(\mathbf{E}) > 0 \quad (5.1.4)$$

Unfortunately, there is no efficient method available that can fit (5.1.1) accurately while at the same time satisfy requirements 1-6. In the matrix fitter package, the approach is to first fit (5.1.1) while satisfying 1-4. Requirements 5-6 are subsequently enforced by perturbation of model parameters, see Sections 5.2 and 5.3.

### S-parameters

In the case of S-parameters, we are interested in calculating the (symmetrical) model

$$\mathbf{S}(s) \cong \mathbf{S}_{rat}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} \quad (5.1.5)$$

Similarly as with the admittance model, we require the poles to be stable and the poles/residues to be real or complex conjugate. The passivity requirement is however, different, now being related to the singular value decomposition,

$$\mathbf{S}(s) = \mathbf{U}(s)\mathbf{\Sigma}(s)\mathbf{V}^H(s) \quad (5.1.6)$$

where  $\mathbf{\Sigma}$  is a diagonal matrix which contains the singular values,  $\sigma_1(s) \dots \sigma_n(s)$ . Passivity of the model entails that all singular values are smaller than unity, i.e.

$$\sigma_i(s) < 1, i = 1 \dots n \quad (5.1.7)$$

Similarly as with Y-parameters, the passivity criterion is enforced by perturbing the model's parameters, see Section 5.3.

### 5.1.2 Standard Vector Fitting

The rational fitting is done via routine `vfdriver.m`. The approach is to stack the diagonal elements of  $\mathbf{Y}$  (or  $\mathbf{S}$ ) into a single vector of elements  $\mathbf{h}(s)$  that is subjected to rational fitting by the pole relocating Vector Fitting (VF) [3], as implemented in `vectfit4.m`. This is a fast process, as the vector  $\mathbf{h}(s)$  is short. Using the obtained poles as new starting poles, the upper triangle elements of  $\mathbf{Y}$  (or  $\mathbf{S}$ ) are stacked into a single (long) vector of elements ( $\mathbf{h}(s)$ ) and fitted using VF.

VF is an iterative technique which *relocates* an initial pole set to better positions by solving a linear least squares problem. The working of VF is in the following explained for the fitting of a scalar (single-element) function,  $h(s)$ ,

$$h(s) \cong \sum_{m=1}^N \frac{r_m}{s - a_m} + d + se \quad (5.1.8)$$

A set of initial poles  $\{a_m\}$  is first specified by the user. With the initial poles as known quantities, the linear problem (5.1.9) is solved as an overdetermined linear least squares problem.

$$\overbrace{\left( \sum_{m=1}^N \frac{\tilde{r}_m}{s - a_m} + 1 \right)}^{\sigma(s)} h(s) \cong \sum_{m=1}^N \frac{r_m}{s - a_m} + d + se \quad (5.1.9)$$

By writing each of the two sums of partial fractions in (5.1.9) as a product of zeros over poles, it can be shown [3] that the poles for  $h(s)$  must be equal to the zeros of  $\sigma(s)$ . These zeros are calculated by solving the eigenvalue problem (5.1.10) [12, Appendix C] where  $\mathbf{A}$  is a diagonal matrix holding the poles  $\{a_m\}$ ,  $\mathbf{b}$  is a column of ones, and  $\mathbf{c}$  holds the residues  $\{\tilde{r}_m\}$ .

$$\{a_m\} = \text{eig}(\mathbf{A} - \mathbf{b} \cdot \mathbf{c}^T) \quad (5.1.10)$$

Repeated application of (5.1.9) and (5.1.10) relocates the initial pole set to better positions. Convergence implies  $\{\tilde{r}_m = 0\}$  in (5.1.9). In practice, one will usually terminate the iterations before the convergence is complete. The final residues are calculated by solving (5.1.8) with known poles. 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 (5.8) with *stable poles* only. (The pole flipping is requested by setting parameter `optsVF.stable=1`, which is the default setting)

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

$$\mathbf{Ax} = \mathbf{b} \quad (5.1.11)$$

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 initial pole set to be specified for the first VF iteration consists of weakly attenuated conjugate pairs that are distributed over the frequency band of interest,

$$a_n = -\alpha + j\beta, a_{n+1} = -\alpha - j\beta \quad (5.1.12a)$$

$$\alpha = v \cdot \beta \quad (5.1.12b)$$

The factor  $v$  in (5.1.12b) can be taken as 0.001. This choice of initial poles ensures a well-conditioned system matrix  $\mathbf{A}$  in (5.1.11).

For the fitting of the matrix problem (5.1.11), the upper triangle of the matrix elements are stacked into a single vector and subjected to fitting by VF. This results in a symmetrical, rational model with a common pole set, i.e. a pole-residue model.

### 5.1.3 Relaxed Vector Fitting

Equation (5.1.9) 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 [4] where the fixed coefficient (unity) is replaced with an unknown coefficient,

$$\overbrace{\left(\sum_{m=1}^N \frac{\tilde{r}_m}{s - a_m} + \tilde{d}\right)h(s)}^{\sigma(s)} \cong \sum_{m=1}^N \frac{r_m}{s - a_m} + d + se \quad (5.1.13)$$

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

$$\text{Re}\left\{\sum_{k=1}^{N_s} \left(\sum_{m=1}^N \frac{\tilde{r}_m}{s_k - a_m} + \tilde{d}\right)\right\} = N_s \quad (5.1.14)$$

This equation is given a LS weighting in relation to the size of  $h$  by

$$\text{weight} = \|w(s) \cdot h(s)\|_2 / N_s \quad (5.1.15)$$

The new poles are now calculated as

$$\{a_m\} = \text{eig}(\mathbf{A} - \mathbf{b} \cdot \tilde{d}^{-1} \cdot \mathbf{c}^T) \quad (5.1.16)$$

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 (5.1.9), 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 Fast implementation

In the pole identification and residue identification steps, the system equation  $\mathbf{Ax}=\mathbf{b}$  is solved via QR-decomposition,

$$\mathbf{A} = \mathbf{QR} \quad (5.1.17a)$$

$$\mathbf{x} = \mathbf{R} \setminus (\mathbf{Q}^T \mathbf{b}) \quad (5.1.17b)$$

In the case of multi-port systems, the solution of the pole identification step (5.1.9) or (5.1.13) can be time consuming and memory demanding. This problem is overcome using the fast implementation in [5] which recognizes that we only need to calculate the free variables associated with  $\sigma$ . This allows to build a smaller, compact system matrix by exploring the block-structure of the system matrix. Each block is solved for independently via QR-decomposition, leading to a new system matrix that has as many columns as there are free variables in  $\sigma$ .

To see this, consider the fitting of two elements. The system matrix for the pole-identification step has structure (5.1.18) and we are only interested in calculating  $\mathbf{x}_3$ .

$$\begin{bmatrix} \mathbf{A} & 0 & \mathbf{B}_1 \\ 0 & \mathbf{A} & \mathbf{B}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} \cong \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \quad (5.1.18)$$

We first consider the first equation

$$[\mathbf{A} \ \mathbf{B}_1] \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_3 \end{bmatrix} = \mathbf{b}_1 \quad (5.1.19)$$

Applying QR-decomposition gives

$$\begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_3 \end{bmatrix} = \mathbf{Q}^T \begin{bmatrix} \mathbf{b}_1^1 \\ \mathbf{b}_1^2 \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1^1 \\ \mathbf{y}_1^2 \end{bmatrix} \quad (5.1.20)$$

where superscripts 1 and 2 denote upper and lower partition of the vector, respectively.

From this we get

$$\mathbf{R}_{22}\mathbf{x}_3 = \mathbf{y}_1^2 \quad (5.1.21)$$

Equation (5.1.21) is built for all equations (block-rows). When fitting a vector of  $n$  elements we thus get (5.1.22). In order to improve the numerical conditioning, the columns of the new system matrix are scaled to unit length before solving by (5.1.17).

$$\begin{bmatrix} \mathbf{R}_{22,1} \\ \mathbf{R}_{22,2} \\ \vdots \\ \mathbf{R}_{22,n} \end{bmatrix} \mathbf{x}_3 = \begin{bmatrix} \mathbf{y}_1^2 \\ \mathbf{y}_2^2 \\ \vdots \\ \mathbf{y}_n^2 \end{bmatrix} \quad (5.1.22)$$

### 5.1.5 Expansion into State Space model

The pole-residue model (5.1.1) can be directly converted into the form

$$\mathbf{Y}_{rat} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} + s\mathbf{E} \quad (5.1.23)$$

The expansion process is straightforward [12].  $\mathbf{A}$  is a diagonal matrix that holds the poles  $\{a_m\}$ , repeated as many times as  $\mathbf{Y}$  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 associated the inputs to blocks (columns) of  $\mathbf{A}$  and  $\mathbf{C}$ . It is noted that  $\mathbf{A}$  has the poles repeated as many times as there ports.

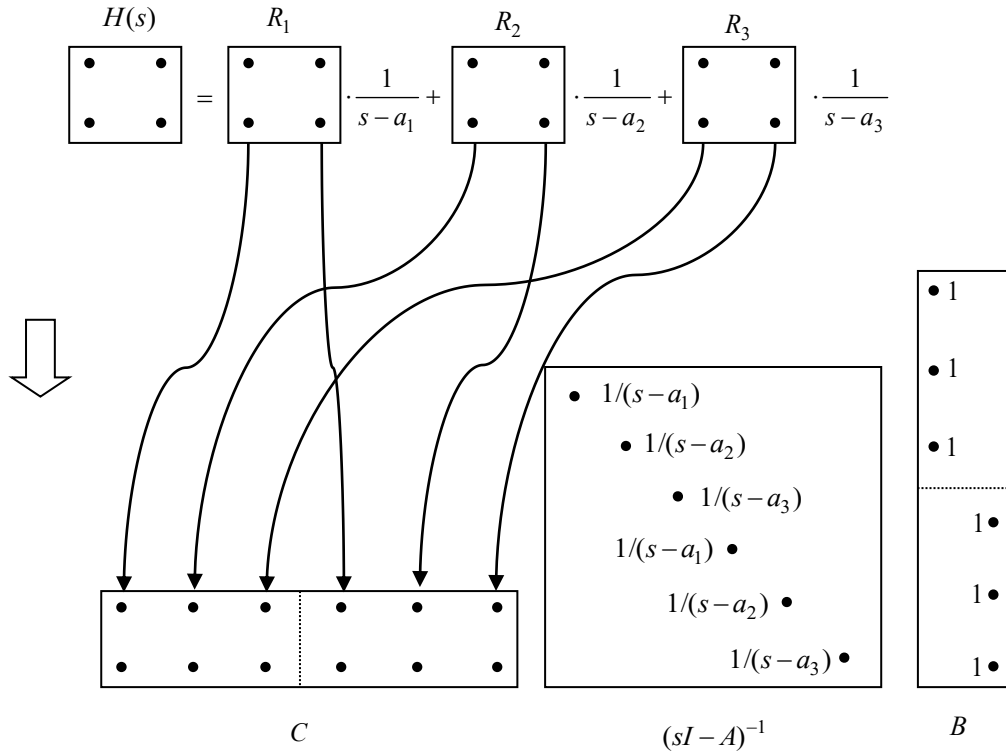


Fig. 5.1. Converting (5.1.23) into a state-space model.

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

$$\hat{\mathbf{A}} = \begin{bmatrix} \text{Re}\{a\} & \text{Im}\{a\} \\ -\text{Im}\{a\} & \text{Re}\{a\} \end{bmatrix}, \quad \hat{\mathbf{c}} = [\text{Re}\{\mathbf{c}\} \quad \text{Im}\{\mathbf{c}\}] \quad (5.1.24)$$

$$\hat{\mathbf{b}} = \begin{bmatrix} 2\text{Re}\{\mathbf{b}^T\} \\ -2\text{Im}\{\mathbf{b}^T\} \end{bmatrix} = \begin{bmatrix} 2\mathbf{b}^T \\ \mathbf{0}^T \end{bmatrix} \quad (5.1.25)$$

## 5.2 *Y*-parameters: Passivity assessment and enforcement

### 5.2.1 Passivity assessment via Singularity Test Matrix

Frequency bands with passivity violations can easily be detected by computing the eigenvalues of  $\text{Re}\{\mathbf{Y}\}$  as function of frequency. The presence of a negative eigenvalue at any frequency point means that the model is non-passive at that frequency. For a symmetrical model, this gives the passivity criterion

$$\text{eig}(\text{Re}\{\sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} + s\mathbf{E}\}) = \text{eig}(\mathbf{G}_{rat}(s)) > 0, \forall s \quad (5.2.1)$$

In practice, however, passivity violations are often very local in nature and are easily missed out in a sweep over discrete frequencies. In order to pinpoint Fortunately, an algebraic criterion exists which allows to precisely detect frequency boundaries of passivity violations.

The first step is to expand the pole-residue model (5.1.1) into a real-only state-space model via (5.1.24)-(5.1.25). From  $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ , the Singularity Test Matrix  $\mathbf{S}$  [10] is formed.

$$\mathbf{P}_Y = \mathbf{A}(\mathbf{B}\mathbf{D}^{-1}\mathbf{C} - \mathbf{A}) \quad (5.2.2)$$

$\mathbf{P}_Y$  gives, via the subset of its positive-real eigenvalues  $\omega^2$  the frequencies  $\omega$  where  $\mathbf{G}_{rat}$  becomes singular and these define crossover frequencies where an eigenvalue of  $\mathbf{G}_{rat}$  changes sign [10]. (Note that (5.2.2) is only applicable for symmetrical models,  $\mathbf{Y}=\mathbf{Y}^T$ , which is the type of model generated by **VFdriver** and **RPdriver**).

In the case that  $\mathbf{D}$  is singular, the following transformation of variables is introduced [13] before applying (5.2.2).

$$\bar{\mathbf{A}} = \mathbf{A}^{-1}, \quad \bar{\mathbf{B}} = -\bar{\mathbf{A}}\mathbf{B}, \quad \bar{\mathbf{C}} = \mathbf{C}\bar{\mathbf{A}}, \quad \bar{\mathbf{D}} = \mathbf{D} - \bar{\mathbf{C}}\bar{\mathbf{A}}\mathbf{B} \quad (5.2.3)$$

With (5.2.2), the positive-real eigenvalues of  $\mathbf{P}_Y$  gives the (squared) *inverse* crossover frequencies  $\omega^{-2}$ .

Note that the test matrix  $\mathbf{P}_Y$  is only half the size of the Hamiltonian matrix that has traditionally been used for passivity assessment. Therefore, usage of  $\mathbf{P}_Y$  is about eight times faster for large cases, and it more convenient to apply since the need for eigenvalue thresholding is avoided.

In the present work we detect bands of violations via  $\mathbf{P}_Y$  as follows:

1. Establish a sorted list of frequencies from the subset of positive real eigenvalues  $\omega$  of  $\mathbf{P}_Y$ .

$$\boldsymbol{\omega} = \{\omega_1, \omega_2, \dots, \omega_n\} \quad (5.2.4)$$

2. Evaluate the criterion (5.26) at the midpoint between frequencies.

$$s = \left\{ \frac{\omega_1 + \omega_2}{2}, \dots, \frac{\omega_{n-1} + \omega_n}{2} \right\} \quad (5.2.5)$$

3. If passivity is violated at the sample  $(\omega_i + \omega_{i+1})/2$  in (5.2.5), then the band  $[\omega_i \ \omega_{i+1}]$  defines a band of passivity violation.
4. In addition, the criterion (5.2.5) is evaluated at samples  $\omega_1/2$  and  $2\omega_n$  to check if passivity is violated between 0 and  $\omega_1$ , and between  $\omega_n$  and infinite frequency.

## 5.2.2 Residue Perturbation by QR-based compacting and NNLS solving

As shown in Section 5.1, application of VF to the data leads to a pole-residue model on the form

$$\mathbf{Y}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} + s\mathbf{E} \quad (5.2.6)$$

Matrices  $\{\mathbf{R}_m\}$ ,  $\mathbf{D}$ , and  $\mathbf{E}$  are assumed to be symmetrical.

Using the ideas in [6], passivity is enforced by perturbing the elements of the residue matrices,  $\{\mathbf{R}_m\}$  and  $\mathbf{D}$ -matrix. In addition,  $\mathbf{E}$  is enforced to be positive definite (has positive eigenvalues). This leads to the constrained optimization problem

$$\Delta \mathbf{Y} = \sum_{m=1}^N \frac{\Delta \mathbf{R}_m}{s - a_m} + \Delta \mathbf{D} + s\Delta \mathbf{E} \approx \mathbf{0} \quad (5.2.7a)$$

$$\text{eig}(\text{Re}\{\mathbf{Y} + \sum_{m=1}^N \frac{\Delta \mathbf{R}_m}{s - a_m} + \Delta \mathbf{D}\}) > \mathbf{0} \quad (5.2.7b)$$

$$\text{eig}(\mathbf{D} + \Delta \mathbf{D}) > \mathbf{0} \quad (5.2.7c)$$

$$\text{eig}(\mathbf{E} + \Delta \mathbf{E}) > \mathbf{0} \quad (5.2.7d)$$

The first part (5.2.7a) minimizes the change to the admittance matrix elements while the second part (5.2.7b) enforces that the perturbed model meets the passivity criterion (5.2.1). The third (5.2.7c) and fourth (5.2.7d) parts enforce that  $\mathbf{D}$  and  $\mathbf{E}$  become positive definite,

The constraints are calculated by use of first order eigenvalue perturbation, which for a perturbation  $\Delta \mathbf{G}$  gives

$$\Delta \lambda_j \approx \mathbf{v}_j^T (\Delta \mathbf{G}) \mathbf{v}_j, j = 1 \dots n \quad (5.2.8)$$

Applying the constraints of (5.2.8) to (5.2.7) leads to the form (5.2.9) where  $\mathbf{x}$  holds the perturbed elements of  $\{\mathbf{R}_m\}$ ,  $\mathbf{D}$ , and  $\mathbf{E}$ .

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\| \quad (5.2.9a)$$

$$\mathbf{C}\mathbf{x} > \mathbf{d} \quad (5.2.9b)$$



The actual solving is made highly efficient by converting the problem into the compacted form (5.2.10) via QR transformation, which is solved using non-negative least squares as (5.2.11). The reader is referred to [7] and [9] for details. The procedure is particularly efficient with use on unitary weighting, or common weighting for all matrix elements. The QR factorization of the block-diagonal  $\mathbf{A}$  is then done more efficient, as the blocks render identical.

$$\min_{\mathbf{y}} \|\mathbf{y}\| \quad s.t. \quad \mathbf{C}\bar{\mathbf{R}}^{-1}\mathbf{y} > \mathbf{d} \quad (5.2.10)$$

$$\min_{\mathbf{u}} \|\mathbf{E}\mathbf{u} - \mathbf{f}\| \quad s.t. \quad \mathbf{u} \geq \mathbf{0} \quad (5.2.11)$$

### 5.2.3 Reducing the number of constraints

Since the computation time needed for the perturbation is strongly dependent on the number of constraints (rows in  $\mathbf{B}_{sys}$ ), we use as few constraints as possible. Violating frequency bands that share common border frequencies are joined together into a single band. This is shown in Fig. 1 for an example with 3 bands,  $b_1$ ,  $b_2$ ,  $b_3$ . Within each concatenated band, the eigenvalues of  $\text{Re}\{\mathbf{Y}(s)\}$  are calculated by frequency sweeping while removing any “artificial” eigenvalue switchovers by the switching back procedure described in [14]. Within each band, the global minimum of all violating eigenvalues are included in the constraint equation, as indicated by the two black dots in Fig. 5.1. The passivity enforcement routine will then bring the two minima up to the zero line by an amount  $c_1$  and  $c_2$ . Because of the linearization between the eigenvalues and the free variables, the value for  $c_1$  and  $c_2$  is chosen slightly higher than the vertical distance to the zero line by an amount *tol*. This reduces the number of iterations needed for removing all passivity violations. The user of RPdriver can request to also include local violations as constraints, which is the default option (`optsRP.localviol=1`). This choice often results in fewer iterations.

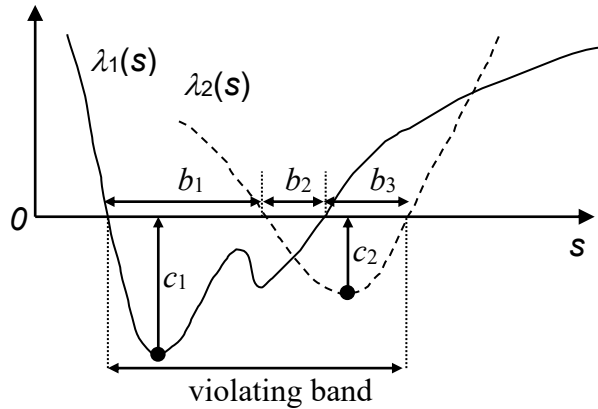


Fig. 5.2 Sample selection for passivity enforcement

### 5.2.4 Robust iterations

One problem with perturbation techniques is that passivity enforcement at selected frequency samples can result in that new violations arise at other frequencies, and divergence can result. In order to tackle this problem, the “robust iterations” [5] is used, see Fig. 5.3 is used. An inner loop is introduced where the solution is discarded if new violations are detected, and the problem is solved again with additional constraints added at the new frequencies of violating minima. That way, one prevents the new violations from appearing. (Matrices  $\mathbf{D}$  and  $\mathbf{E}$  are removed from the perturbation process as soon as they become positive definite).

The inner-loop feature is only available with use of Z- and Y-parameter fitting, i.e., with `optsRP.parametertype='Y'`.

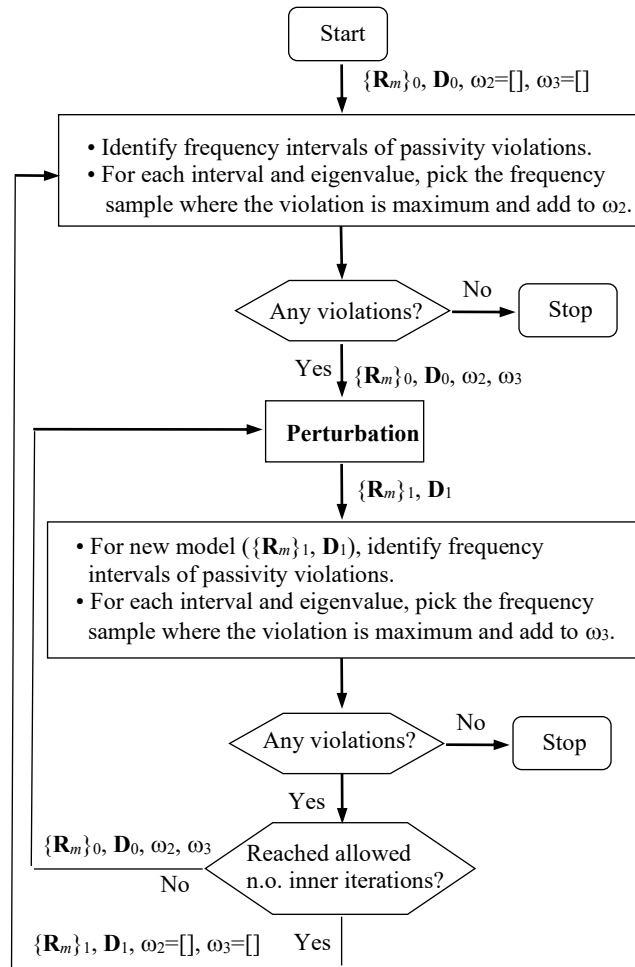


Fig. 5.3 Robust iterations.

### 5.3 S-parameters: Passivity assessment and enforcement

The implemented passivity enforcement scheme for S-parameter models is similar to that for Y-parameters. The following outlines the main differences.

#### 5.3.1 Passivity assessment via singularity test matrix

The passivity assessment is now done using the test matrix [11]

$$\mathbf{P}_S = (\mathbf{A} - \mathbf{B}(\mathbf{D} - \mathbf{I})^{-1}\mathbf{C})(\mathbf{A} - \mathbf{B}(\mathbf{D} + \mathbf{I})^{-1}\mathbf{C}) \quad (5.3.1)$$

$\mathbf{P}_S$  gives, via the subset of its negative-real eigenvalues  $-\omega^2$ , the crossover frequencies  $j\omega$  where the singular values of  $\mathbf{S}$  are unity. In the case that  $(\mathbf{D} + \mathbf{I})$  or  $(\mathbf{D} - \mathbf{I})$  is singular, the transformation (5.2.3) is applied.

#### 5.3.2 Passivity enforcement by QR-based compacting and NNLS solving

The S-parameter data set can be fitted with a model of type (5.3.2).

$$\mathbf{S}(s) = \sum_{m=1}^N \frac{\mathbf{R}_m}{s - a_m} + \mathbf{D} \quad (5.3.2)$$

Matrices  $\{\mathbf{R}_m\}$ , and  $\mathbf{D}$  are assumed to be symmetrical.

Passivity is enforced by perturbing the elements of the residue matrices,  $\{\mathbf{R}_m\}$  and  $\mathbf{D}$ -matrix. This leads to a constrained optimization problem involving singular values

$$\Delta\mathbf{S} = \sum_{m=1}^N \frac{\Delta\mathbf{R}_m}{s - a_m} + \Delta\mathbf{D} \approx \mathbf{0} \quad (5.3.3a)$$

$$\sigma_i(\mathbf{S} + \sum_{m=1}^N \frac{\Delta\mathbf{R}_m}{s - a_m} + \Delta\mathbf{D}) < 1, i = 1 \dots n \quad (5.3.3b)$$

$$\sigma_i(\mathbf{D} + \Delta\mathbf{D}) < 1, i = 1 \dots n \quad (5.3.3c)$$

The passivity constraint relies on first order approximation between a matrix perturbation and the singular values,

$$\Delta\sigma_i \approx \text{Re}\{\mathbf{u}_i^H \Delta\mathbf{S} \mathbf{v}_i\}, i = 1 \dots n \quad (5.3.4)$$

Similarly as for the Y-parameter case, we arrive at the constrained LS problem,

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}\| \quad (5.3.5a)$$

$$\mathbf{C}\mathbf{x} > \mathbf{d} \quad (5.3.5b)$$

which is converted into a compacted form via QR-factorization

$$\min_{\mathbf{y}} \|\mathbf{y}\| \quad s.t. \quad \mathbf{C}\bar{\mathbf{R}}^{-1}\mathbf{y} > \mathbf{d} \quad (5.3.6)$$

which is solved as a non-negative least squares problem,

$$\min_{\mathbf{u}} \|\mathbf{E}\mathbf{u} - \mathbf{f}\| \quad s.t. \quad \mathbf{u} \geq \mathbf{0} \quad (5.3.7)$$

The reader is referred to [9] for details.

As with Y-parameters, the number of constraints are kept low by keeping only violation extremas in the constraint equation.

## 6. TEST CASES

### 6.1 *FDNE modeling from Y-parameters*

This data set is a contribution from Elias Mberou, RTE, France.

- P=33 terminals
- K=2501 frequency samples

#### 6.1.1 Rational fitting

Here we use

- Inverse magnitude weighting

```
%=====
%=          POLE-RESIDUE FITTING          =
%=====
optsVF.N=200; %150; %120 ;% 150           %Order of approximation.
optsVF.poletype='lincmplx'; %Linearly spaced and logarithmically spaced poles
optsVF.weightparam=2; % --> weighting with inverse magnitude norm
optsVF.Niter1=10;      %Number of iterations for fitting diagonal elements (fast!)
optsVF.Niter2=4;       %Number of iterations for matrix fitting
optsVF.asymp=2;        %Fitting includes D
optsVF.logx=0;         %=0 --> Plotting is done using linear abscissa axis
optsVF.passive_DE=1;
optsVF.errrplot=0;
poles=[];
optsVF.NE=1;

%Calling VFdriver:
[SER,rmserr,bigYfit,optsVF2]=VFdriver(bigY,s,poles,optsVF);

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
  Iter 8
  Iter 9
  Iter 10
****Vector Fitting upper triangle elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
****Enforcing positive eigenvalues for D...
  --Found none.
Elapsed time is 124.197983 seconds.
****Transforming model of lower matrix triangle into state-space model of full matrix ...
****Generating pole-residue model ...
****Plotting of results ...
      rmserr:      0.0015945
```

rmserrweight: 0.026791

----- E N D -----

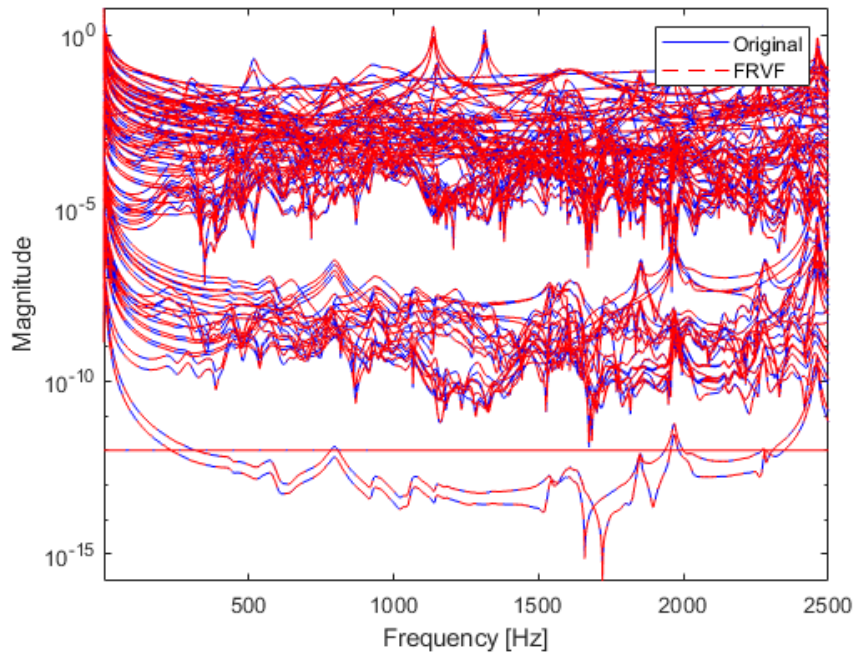


Fig. 6.1.1 Rational fitting.

## 6.1.2 Residue perturbation

Here we use

- Passivity assessment by frequency sweeping
- One inner-loop iteration
- Inverse magnitude weighting

```
%=====
%=          Passivity Enforcement          =
%=====

optsRP.Niter_out=40; %20
optsRP.Niter_in =1;
optsRP.weightparam=2;
optsRP.parametertype='Y';
optsRP.plot.s_pass=s;
optsRP.plot.ylim=[-1e-2 1e-2];
optsRP.plot.spy=2;
optsRP.plot.logx=0;
optsRP.plot.spy=1;

%-----
optsRP.solver='nnls';
optsRP.alpha=1.05;
optsRP.weightfactor=1;

optsRP.assessbysweeping_s=i*linspace(0,2*pi*3112,3113);
optsRP.plot.s_pass      =optsRP.assessbysweeping_s;
```

```
%Calling RPdriver:
[SERpass,bigYfit_passive,optsRP2]=RPdriver(SER,s,optsRP);
```

```
----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (Y-parameters) ***
Initial model : Max. violation : -0.23599
Iteration 1 : Max. violation : -0.027054
Iteration 2 : Max. violation : -0.028433
Iteration 3 : Max. violation : -0.0045443
Iteration 4 : Max. violation : -0.0027563
Iteration 5 : Max. violation : -0.00072021
Iteration 6 : Max. violation : -0.00051845
Iteration 7 : Max. violation : -4.3896e-05
Iteration 8 : Max. violation : -2.6245e-06
Iteration 9 : Max. violation : -1.4604e-06
Iteration 10 : Max. violation : None
-->Passivity was successfully enforced.
rmserr: 0.0005381
rmserrweight: 0.0078143

Time summary:
Passivity assessment : 40.9034 sec
Passivity enforcement: 24.718 sec
    Build A (LS): 14.4283 sec
    Build C : 4.2602 sec
    Build E : 2.2326 sec
    Solve NNLS : 0.66292 sec
Total: 65.6365 sec
----- E N D -----
```

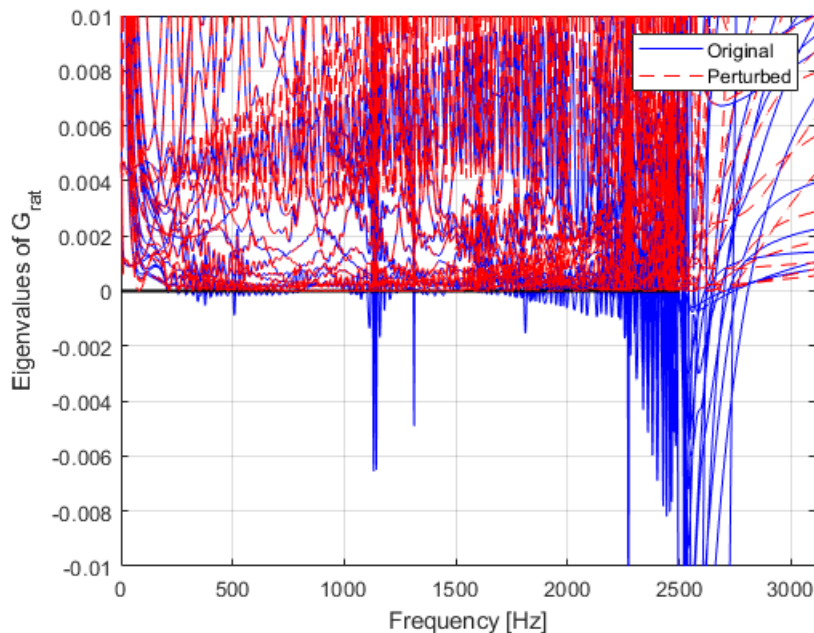


Fig. 6.1.2. Eigenvalues of  $G(s)$ .

### 6.1.3 Final validation: Efficient plotting of final result

Plotting of large dat sets can be slow. Below show a highly efficient approach.

```
%=====
% Creating plot of final result:
%=====

P=length(squeeze(bigYfit(:,1,1)));
f      =reshape(bigY,P*P,Ns);
fit     =reshape(bigYfit,P*P,Ns);
fit_passive=reshape(bigYfit_passive,P*P,Ns);

M=length(f(:,1));
bigfreq=repmat([freq(:).' NaN],1,M);
bigf    =[f NaN.*ones(M,1)].'; bigf=bigf(:).';
bigfit  =[fit NaN.*ones(M,1)].'; bigfit=bigfit(:).';
bigfit_passive=[fit_passive NaN.*ones(M,1)].'; bigfit_passive=bigfit_passive(:).';

figure(11),
h1=semilogy(bigfreq,abs(bigf),'b-'); hold on
h2=semilogy(bigfreq,abs(bigfit),'r--'); hold on
h3=semilogy(bigfreq,abs(bigfit_passive),'c-.'); hold off
xlim([freq(1) freq(end)]);
xlabel('Frequency [Hz]')
ylabel('Magnitude [Hz]')
legend([h1 h2 h3], 'Data', 'VF model', 'Passivated model', 'location', 'SE'),
```

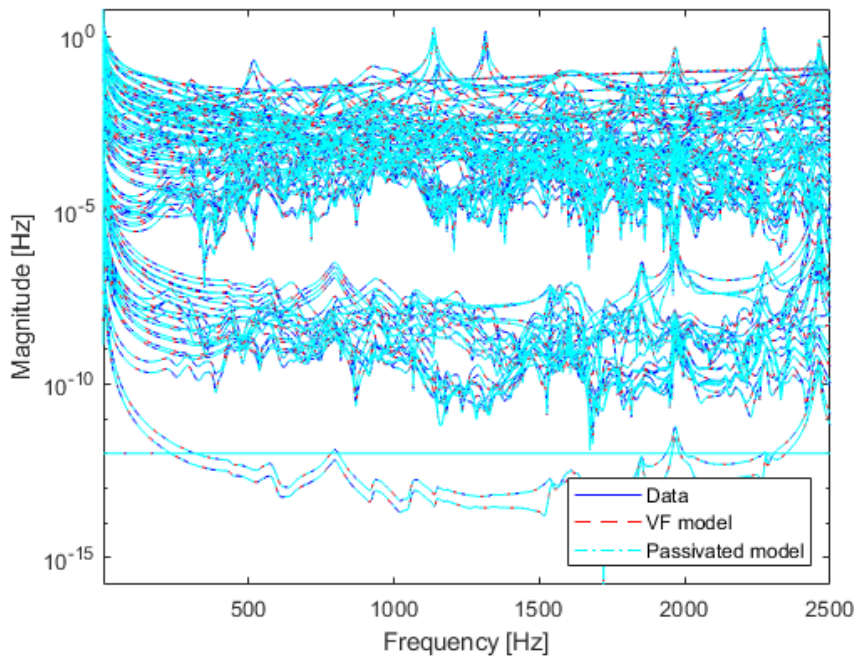


Fig. 6.1.3. Final validation.



## 6.2 Transformer black-box modeling from Y-parameters

This data set is a contribution from CIGRE JWG A2/C4.52.

### 6.2.1 Rational fitting

```
%=====
%=          POLE-RESIDUE FITTING          =
%=====
optsVF.N=120; %120 ;%          %Order of approximation.
optsVF.poletype='linlogcmplx'; %Mix of linearly spaced and logarithmically spaced poles
optsVF.weightparam=2; %5 --> weighting with inverse magnitude norm
optsVF.Niter1=10;      %Number of iterations for fitting sum of elements (fast!)
optsVF.Niter2=4;      %Number of iterations for matrix fitting
optsVF.asymp=2;       %Fitting includes D
optsVF.plot=1;
optsVF.logx=1;        %=0 --> Plotting is done using linear abscissa axis
optsVF.passive_DE=1;
optsVF.errplot=0;
optsVF.screen=1;
poles=[];

%Calling VFdriver:
[SER,rmserr,bigYfit,optsVF]=VFdriver(bigY,s,poles,optsVF); %Creating state-space model and
pole-residue model

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
  Iter 8
  Iter 9
  Iter 10
****Vector Fitting upper triangle elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
****Enforcing positive eigenvalues for D...
  ---Found none.
Elapsed time is 0.728443 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...
      rmserr:      0.00012081
      rmserrweight: 0.083954
----- E N D -----
```

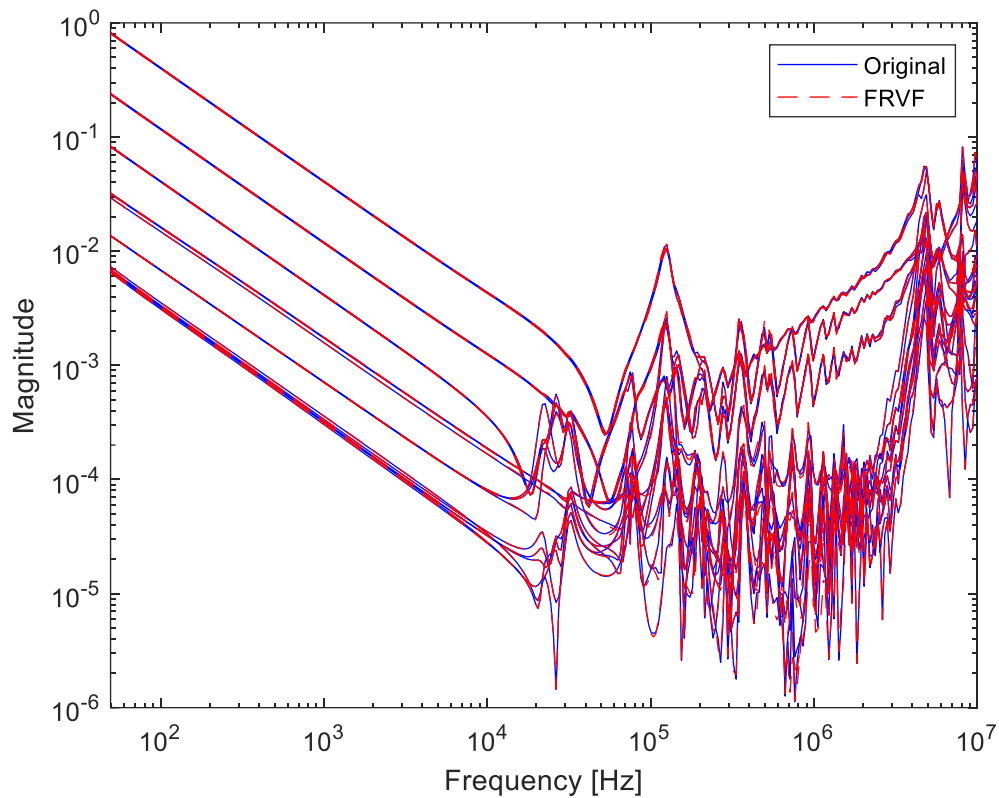


Fig. 6.2.1. Rational fitting.

## 6.2.2 Residue perturbation

```
%=====
%=          Passivity Enforcement          =
%=====

optsRP.Niter_out=20;
optsRP.Niter_in =2;
optsRP.parametertype='Y';

optsRP.plot=[];
optsRP.plot.s_pass=2*pi*i*logspace(6,7,1001); %Focus on the ghigh-frequency area
optsRP.plot.ylim=[-1e-2 1e-2];
optsRP.plot.spy=1; %0
optsRP.plot.logx=1;
optsRP.screen=0;

optsRP.weightparam=2; %Inverse magnitude weighting
optsRP.weightfactor=1;

optsRP.TOLG=1e-8; %1e-8
optsRP.TOLD=1e-6; %1e-3;
optsRP.TOLE=1e-12;
optsRP.alpha=1.05; %1.05

%Calling RPdriver:
[SER_passive,bigYfit_passive,optsRP]=RPdriver(SER,s,optsRP);

----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (Y-parameters) ***
```

```
Initial model : Max. violation : -0.015912
Iteration 1   : Max. violation : -0.010047
Iteration 2   : Max. violation : -0.0010505
Iteration 3   : Max. violation : -0.00052102
Iteration 4   : Max. violation : -0.00010575
Iteration 5   : Max. violation : -3.9729e-08
Iteration 6   : Max. violation : None
-->Passivity was successfully enforced.
rmserr:      0.00059863
rmserrweight: 0.021766
```

Time summary:

```
Passivity assessment : 1.9078 sec
Passivity enforcement: 0.16651 sec
  Build A (LS):      0.049638 sec
  Build C           : 0.044558 sec
  Build E           : 0.019393 sec
  Solve NNLS        : 0.013443 sec
Total: 2.0765 sec
```

----- E N D -----

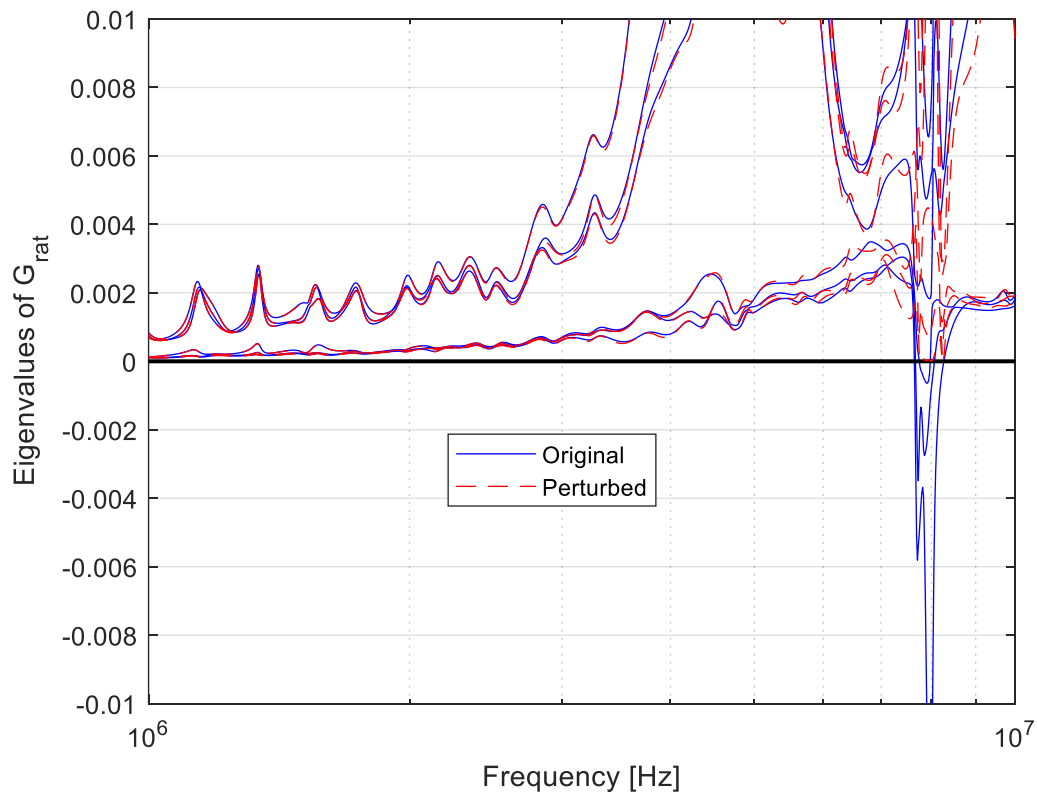


Fig. 6.2.2. Eigenvalues of  $G(s)$ .

## 6.2.3 Final validation

```
%=====
% Creating plot of final result:

P=length(squeeze(bigYfit(:,1,1)));
```

```
f      =reshape(bigY,P*P,Ns);
fit     =reshape(bigYfit,P*P,Ns);
fit_passive=reshape(bigYfit_passive,P*P,Ns);

M=length(f(:,1));
bigfreq=repmat([freq(:).' NaN],1,M);
bigf    =[f NaN.*ones(M,1)].'; bigf=bigf(:).';
bigfit  =[fit NaN.*ones(M,1)].'; bigfit=bigfit(:).';
bigfit_passive=[fit_passive NaN.*ones(M,1)].'; bigfit_passive=bigfit_passive(:).';

figure(11),
h1=loglog(bigfreq,abs(bigf),'b-'); hold on
h2=loglog(bigfreq,abs(bigfit),'r--'); hold on
h3=loglog(bigfreq,abs(bigfit_passive),'c-.'); hold off
xlim([freq(1) freq(end)]);
xlabel('Frequency [Hz]')
ylabel('Magnitude [Hz]')
legend([h1 h2 h3], 'Data', 'VF model', 'Passivated model', 'location', 'SE'),
```

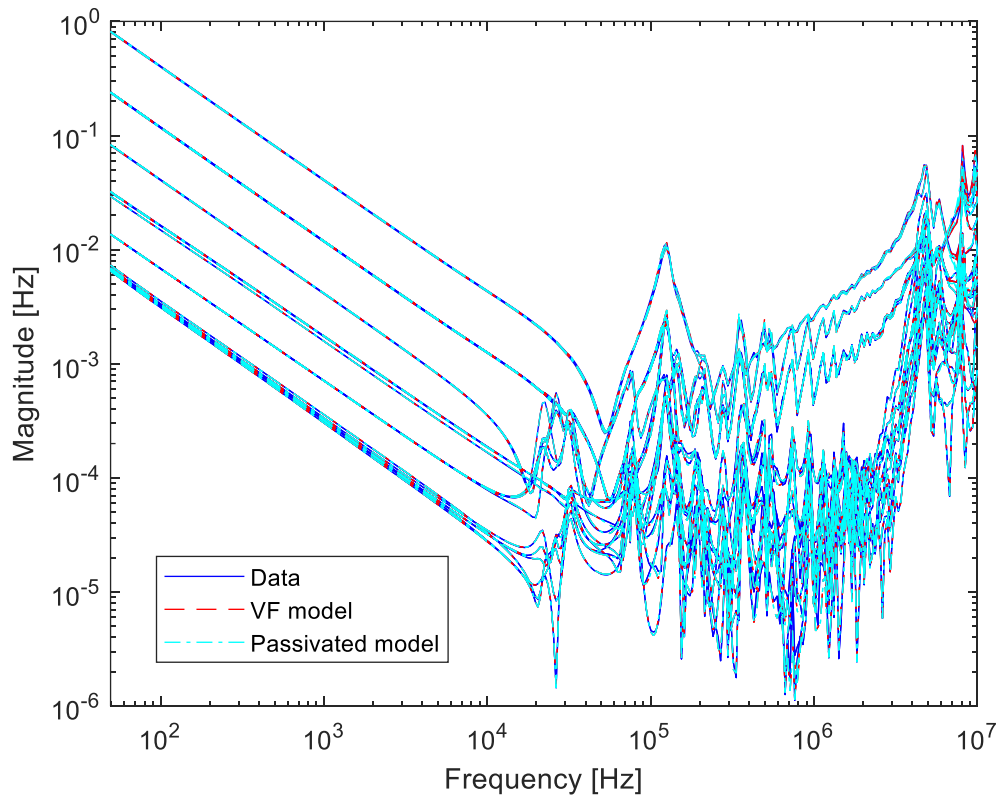


Fig. 6.2.3. Final validation.

## 6.3 Transformer white-box modeling from Z-parameters

This data set is a contribution from Luiz Fernando de Oliveira, Hitachi, Brazil.

### 6.3.1 Rational fitting

```
%=====
%=          POLE-RESIDUE FITTING          =
%=====
poles=-2*pi*logspace(log10(1*freq(1)),log10(1*freq(end)),N);
optsVF.poletype=[]; %Mix of linearly spaced and logarithmically spaced poles
optsVF.weightparam=2; %5 --> weighting with inverse magnitude norm
optsVF.Niter1=10;    %Number of iterations for fitting sum of elements (fast!) --> Improved
initial poles
optsVF.Niter2=10;    %Number of iterations for matrix fitting
optsVF.asymp=2;      %Fitting includes D
optsVF.logx=0;       %=0 --> Plotting is done using linear abscissa axis
optsVF.plot=1;

optsVF.logx=1;
optsVF.N=N;
optsVF.errplot=0;
optsVF.phaseplot=0;
optsVF.passive_DE=1;

%Calling VFdriver:
[SER,rmserr,bigZfit,optsVF2]=VFdriver(bigZ,s,poles,optsVF); %Creating state-space model and
pole-residue model

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
  Iter 8
  Iter 9
  Iter 10
****Vector Fitting upper triangle elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
  Iter 8
  Iter 9
  Iter 10
****Enforcing positive eigenvalues for D...
  ---Found none.
Elapsed time is 5.260795 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...
```

```
rmseerr:      0.22598
rmseerrweight: 0.00012411
```

----- E N D -----

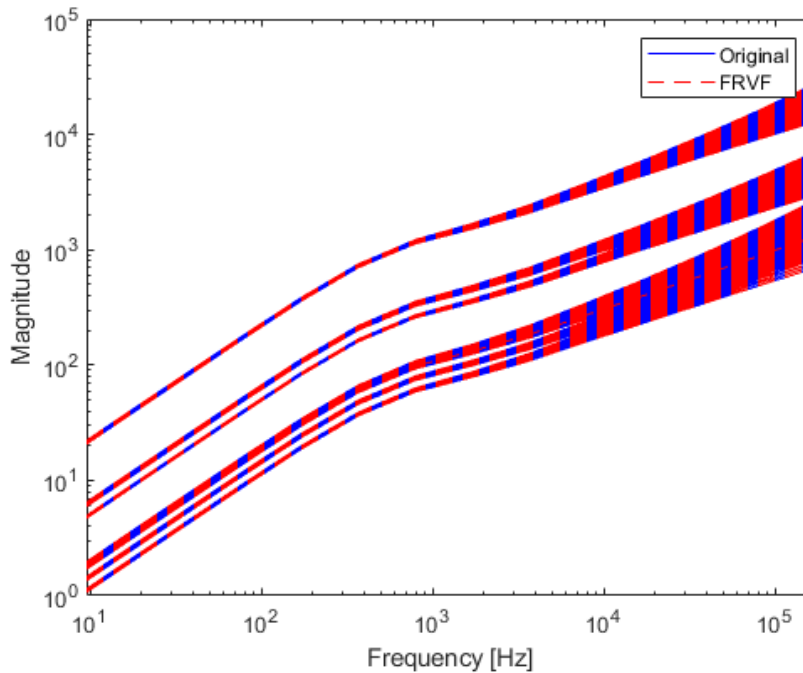


Fig. 6.3.1. Rational fitting.

## 6.3.2 Residue perturbation

```
----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (Y-parameters) ***
  Initial model : Max. violation : -0.0022204
  Iteration 1   : Max. violation : -0.00016508
  Iteration 2   : Max. violation : None
-->Passivity was successfully enforced.
  rmseerr:      0.00037612
  rmseerrweight: 8.3516e-07

Time summary:
  Passivity assessment : 1.1546 sec
  Passivity enforcement: 0.31675 sec
    Build A (LS):      0.1973 sec
    Build C           : 0.022888 sec
    Build E           : 0.046412 sec
    Solve NNLS        : 0.0009352 sec
  Total: 1.4756 sec
----- E N D -----

%=====
%=          Passivity Enforcement          =
%=====
optsRP.Niter_out=20;
optsRP.Niter_in=0;
optsRP.parametertype='Y';
```

```

optsRP.plot.s_pass=s;
optsRP.plot.ylim=[-1e-2 1e-2];
optsRP.plot.spy=1;
optsRP.plot.logx=1;

optsRP.weightfactor=1;%1e-2;
optsRP.weightparam=2;

optsRP.TOLG=1e-8; %1e-8
optsRP.TOLD=1e-6; %1e-3;
optsRP.TOLE=1e-12;

optsRP.alpha=1.1;
optsRP.plot.s_pass=i*logspace( log10(0.001*imag(s(1))),log10(1.5*imag(s(end))),201);

%Calling RPdriver:
[SERpassive,bigZfit_passive,optsRP2]=RPdriver(SER,s,optsRP);

```

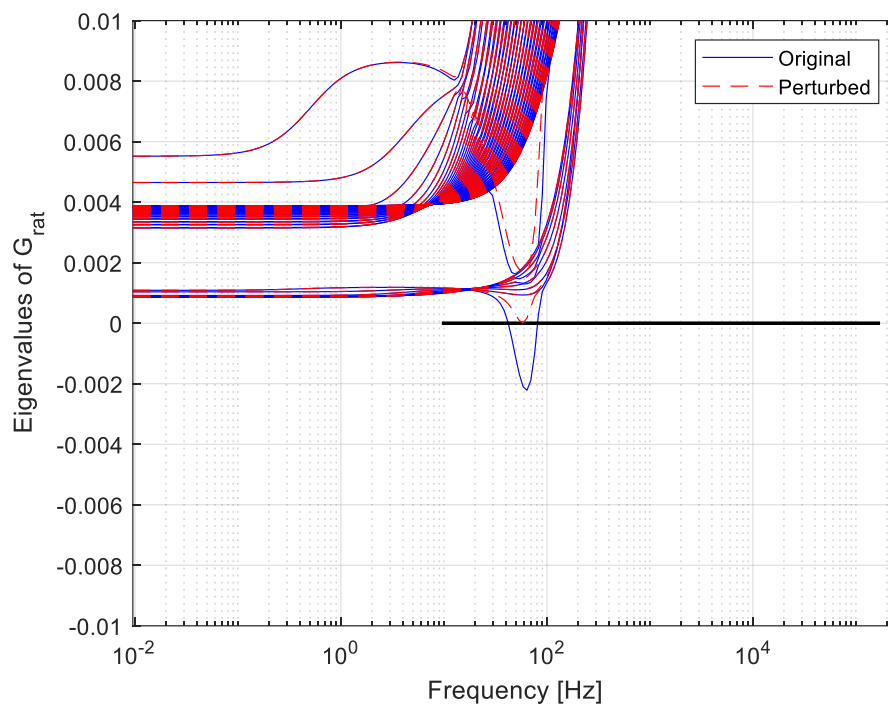


Fig. 6.3.2 Eigenvalues of  $\mathbf{G}(s)$ .

### 6.3.3 Final validation

```

%=====

% Creating plot of final result:

P=length(squeeze(bigZfit(:,1,1)));
f      =reshape(bigZ,P*P,Ns);
fit     =reshape(bigZfit,P*P,Ns);
fit_passive=reshape(bigZfit_passive,P*P,Ns);

M=length(f(:,1));
bigfreq= repmat([freq(:). ' NaN'],1,M);

```

```
bigf = [f NaN.*ones(M,1)].'; bigf=bigf(:).';  
bigfit=[fit NaN.*ones(M,1)].'; bigfit=bigfit(:).';  
bigfit_passive=[fit_passive NaN.*ones(M,1)].'; bigfit_passive=bigfit_passive(:).';  
  
figure(11),  
h1=loglog(bigfreq,abs(bigf),'b-'); hold on  
h2=loglog(bigfreq,abs(bigfit),'r--'); hold on  
h3=loglog(bigfreq,abs(bigfit_passive),'c-.'); hold off  
xlim([freq(1) freq(end)]);  
xlabel('Frequency [Hz]')  
ylabel('Magnitude [Hz]')  
legend([h1 h2 h3], 'Data', 'VF model', 'Passivated model', 'location', 'SE'),
```

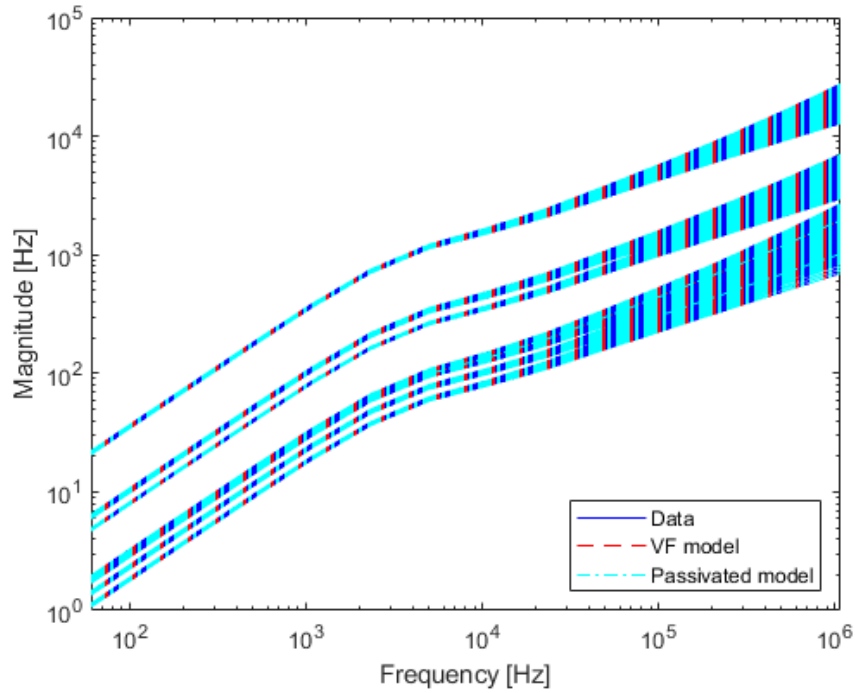


Fig. 6.3.3. Final validation.



## 6.4 High-speed interconnect modeling from S-parameters

This data set is a contribution from Prof. Dr. Chiu-Chih Chou, National Central University, Zhong-Li, Taiwan, ROC.

### 6.4.1 Rational fitting

Here we use

- Only iterations on the diagonal elements (optsVF.Niter2=0)
- Inverse magnitude weighting
- Normal Equations fir final fitting of residues

```
%=====
%=          POLE-RESIDUE FITTING          =
%=====
%optsVF.N=24; % Low order to demonstrate substantial deviations at low frequencies
optsVF.N=40; %
optsVF.parametertype='S';
optsVF.poletype='linlogcmplx'; %Initial poles: Mix of linearly and logarithmically spaced
poles
optsVF.weightparam=2; %Unitary weighting
optsVF.Niter1=7;      %Number of iterations for fitting diagonal elements (fast!)
optsVF.Niter2=0;      %Number of iterations for fittingthe upper triangle elements

optsVF.asymp=2;       %Fitting includes D
optsVF.logx=0;        %=0 --> Plotting is done using linear abscissa axis
poles=[];

optsVF.logx=0;
optsVF.phaseplot=0;
optsVF.errplot=0;

optsVF.passive_DE=1;

optsVF.NE=1;

%Calling VFdriver:
[SER,rmterr,bigSfit,optsVF2]=VFdriver(bigS,s,poles,optsVF); %Creating state-space model and
pole-residue model

----- S T A R T (VFdriver.m) -----
****Vector Fitting diagonal elements ...
  Iter 1
  Iter 2
  Iter 3
  Iter 4
  Iter 5
  Iter 6
  Iter 7
****Vector Fitting upper triangle elements ...
****Enforcing singular values of D to be smaller than unity...
  ---Found none.
Elapsed time is 5.641235 seconds.
****Transforming model of lower matrix triangle into state-space model of full
matrix ...
****Generating pole-residue model ...
****Plotting of results ...
```

```
rmterr:      0.00062752
rmterrweight: 0.075416
```

----- E N D -----

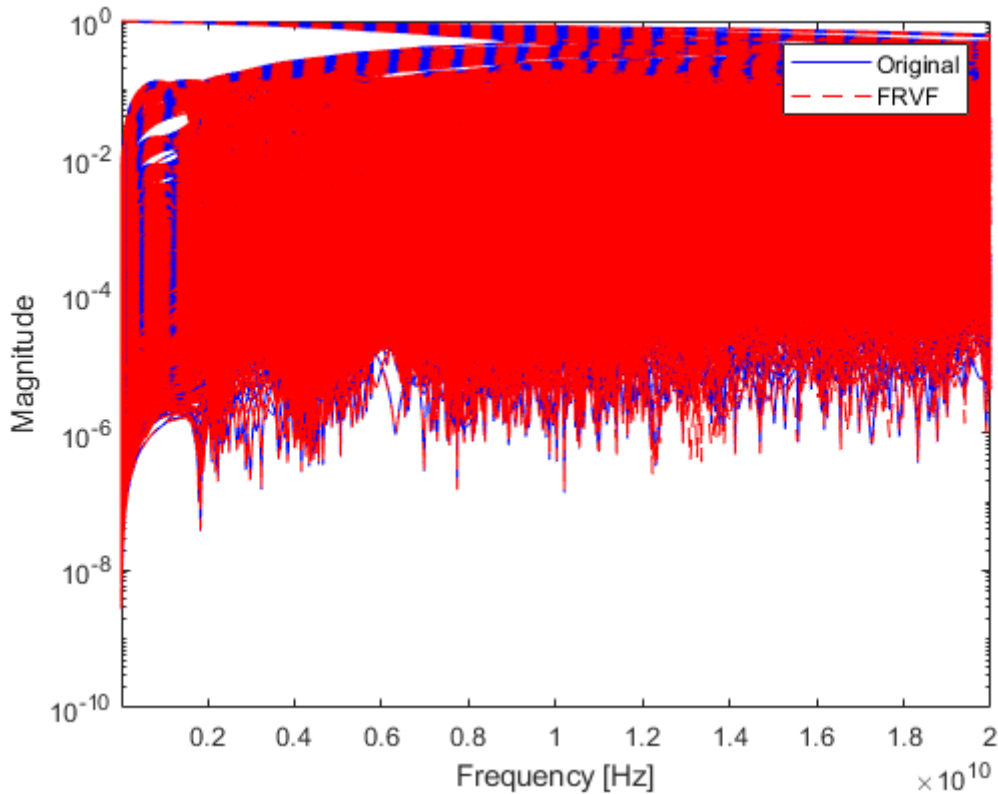


Fig. 6.3.1. Rational fitting.

## 6.4.2 Residue perturbation

```
----- S T A R T (RPdriver.m) -----
*** Passivity enforcement (S-parameters) ***
Initial model : Max. violation : 0.0048661
Iteration 1 : Max. violation : 0.00041007
Iteration 2 : Max. violation : 2.5117e-05
Iteration 3 : Max. violation : 5.6915e-06
Iteration 4 : Max. violation : 8.9681e-07
Iteration 5 : Max. violation : None
-->Passivity was successfully enforced.
rmterr:      5.869e-06
rmterrweight: 5.9509e-06

Time summary:
Passivity assessment : 4.0777 sec
Passivity enforcement: 10.2789 sec
  Build A (LS):      7.5795 sec
  Build C           : 1.2794 sec
  Build E           : 0.51752 sec
  Solve NNLS        : 0.06183 sec
Total: 14.3726 sec
----- E N D -----
```

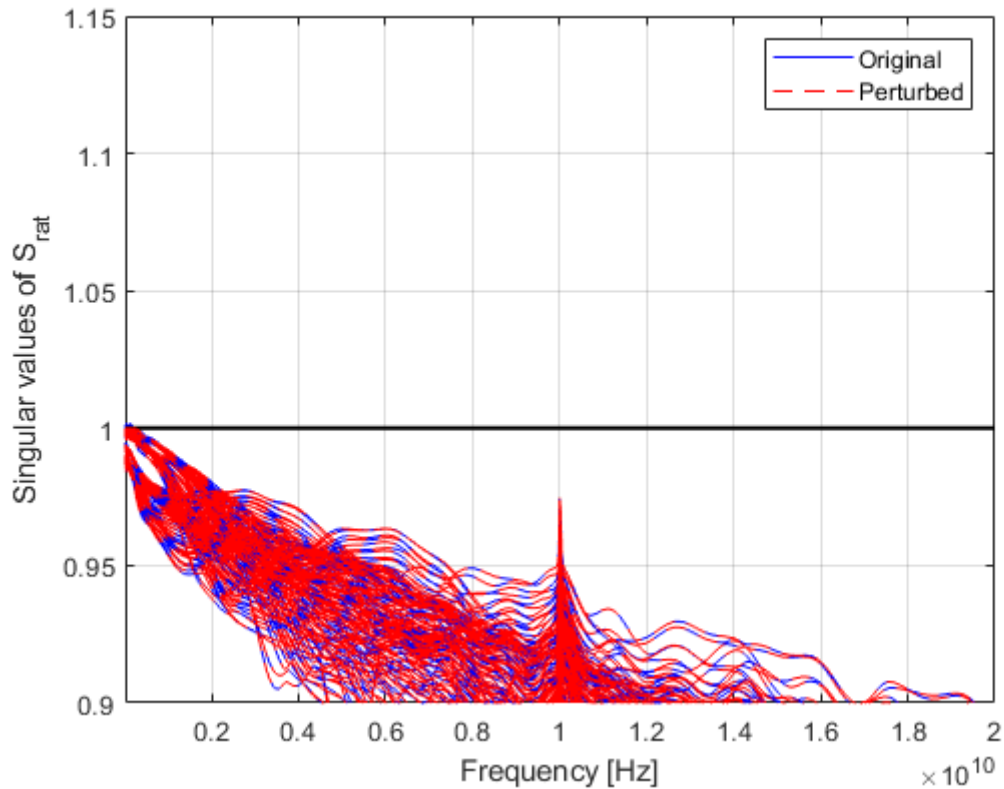


Fig. 6.3.2 Eigenvalues of  $G(s)$ .

### 6.4.3 Final validation

```
%=====
```

```
% Creating plot of final result:
```

```
P=length(squeeze(bigSfit(:,1,1)));
f      =reshape(bigS,P*P,Ns);
fit     =reshape(bigSfit,P*P,Ns);
fit_passive=reshape(bigSfit_passive,P*P,Ns);

M=length(f(:,1));
bigfreq= repmat([freq(:).' NaN],1,M);
bigf   =[f NaN.*ones(M,1)].'; bigf=bigf(:).';
bigfit =[fit NaN.*ones(M,1)].'; bigfit=bigfit(:).';
bigfit_passive=[fit_passive NaN.*ones(M,1)].'; bigfit_passive=bigfit_passive(:).';

figure(11),
h1=loglog(bigfreq,abs(bigf),'b-'); hold on
h2=loglog(bigfreq,abs(bigfit),'r--'); hold on
h3=loglog(bigfreq,abs(bigfit_passive),'c-.'); hold off
xlim([freq(1) freq(end)]);
xlabel('Frequency [Hz]')
ylabel('Magnitude [Hz]')
legend([h1 h2 h3], 'Data', 'VF model', 'Passivated model', 'location', 'SE'),
```

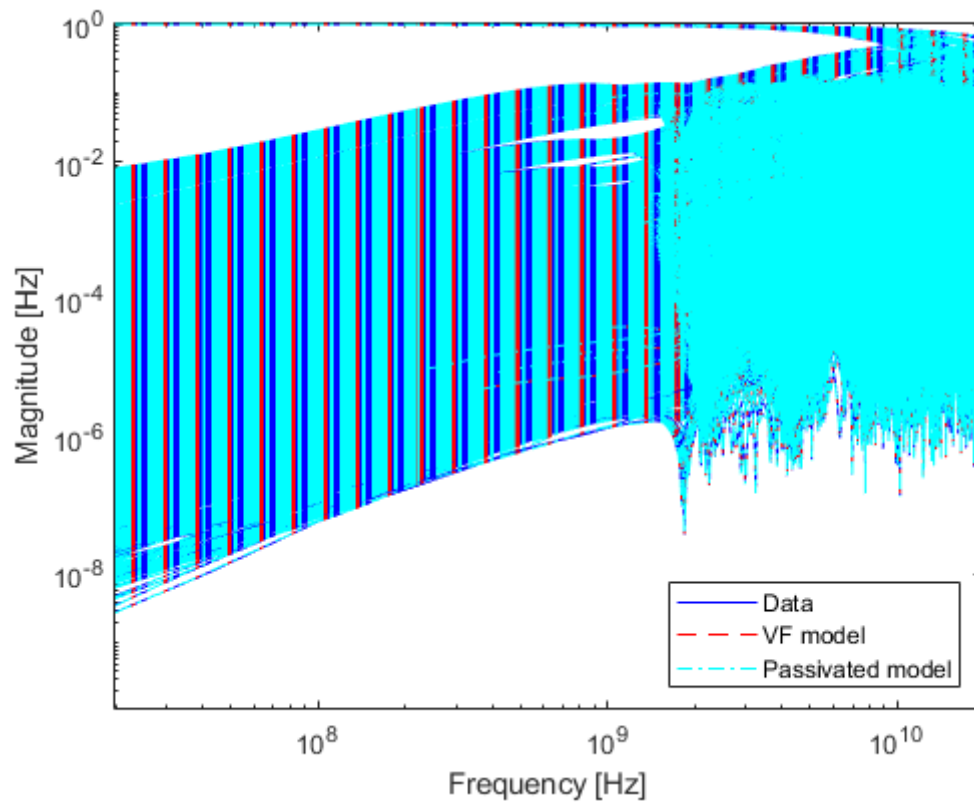


Fig. 6.3.3. Final validation.

## 6. REFERENCES

- [1] B. Gustavsen, “MFT-NNLS: A toolbox for passive macromodeling from large immittance and scattering data sets using vector fitting and residue perturbation in Matlab”, *IEEE Trans. Power Delivery*, 2026, available online: [ieeexplore.org](https://ieeexplore.org).
- [2] S. Grivet-Talocia and B. Gustavsen, *Passive Macromodeling: Theory and Applications*, John Wiley and Sons, 2016.
- [3] 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.
- [4] B. Gustavsen, “Improving the pole relocating properties of vector fitting”, *IEEE Trans. Power Delivery*, vol. 21, no. 3, pp. 1587-1592, July 2006.
- [5] 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.
- [6] B. Gustavsen, and A. Semlyen, “Enforcing passivity for admittance matrices approximated by rational functions”, *IEEE Trans. Power Systems*, vol. 16, no. 1, pp. 97-104, Feb. 2001
- [7] B. Gustavsen, "Passivity enforcement by residue perturbation via constrained non-negative least squares", *IEEE Trans. Power Delivery*, vol. 36, no. 5, pp. 2758-2767, October 2021.
- [8] B. Gustavsen, "Passivity assessment and enforcement utilizing eigenpairs information", *Electric Power Systems Research* 194 (2021), 107041.
- [9] B. Gustavsen, “An efficient residue perturbation scheme for passivity enforcement of S-parameter rational models”, *IEEE Trans. Electromagnetic Compatibility*, vol. 67, no. 3, pp. 913-920, June 2025.
- [10] A. Semlyen and B. Gustavsen, “A half-size singularity test matrix for fast and reliable passivity assessment of rational models”, *IEEE Trans. Power Delivery*, vol. 24, no. 1, pp. 345-351, January 2009.
- [11] B. Gustavsen and A. Semlyen, “Fast passivity assessment for S-parameter rational models via a half-size test matrix”, *IEEE Trans. Microwave Theory And Techniques*, vol. 56, no. 12, pp. 2701-2708, December 2008.
- [12] 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.
- [13] B. Gustavsen and A. Semlyen, “Simulation of transmission line transients using vector fitting and modal decomposition”, *IEEE Trans. Power Delivery*, vol. 13, no. 2, pp. 605-614, April 1998.
- [14] Bill. Whiten (2025), “Nnls Non negative least squares”, (<https://www.mathworks.com/matlabcentral/fileexchange/38003-nnls-non-negative-least-squares>), MATLAB Central File Exchange. Retrieved March 17, 2025.
- [15] J.M. Myhre, E. Frahm, D.J. Lilja, and Mo.O. Saar, “TNT-NN: A fast active set method for solving large non-negative least squares problems”, *Procedia Computer Science*, vol. 108, pp. 755-764, 2017.

## 7. ACKNOWLEDGEMENT

The development of this v2.0 of the Matrix Fitter Toolbox (MFT-NNLS) was in part supported by NorthWind FME project,  
<https://www.northwindresearch.no/>