pmm: A Matlab Toolbox for Passive Macro-Modeling

Zuochang Ye
yezuochang@gmail.com

August 31, 2012

Contents

1 Introduction

1.1 What is *pmm*?

pmm is a Matlab toolbox for passive macro-modeling from frequency-domain (measured or EM-simulated) tabulated data, such as S-parameter data. It takes industrial-standard touchstone format file as input, and generate state-space model in circuit netlist format which can be used in commercial SPICE simulator for doing all types of analyses. It composes of different state-of-the-art modeling techniques, and it is designed to be extensible, i.e. new methods can be easily plugged into the flow.

1.2 Why *pmm*?

There are a couple reasons to use this tool

1. A few popular algorithms published by other authors are implemented in *pmm*. These algorithms are only accessible from literature, and there is no source code released. Although it may not have the same performance as the original ones, it illustrated a way to compare between different algorithms in the same platform. In addition, interfaces to some public-domain tools are provided.

2. *pmm* incorporates some new techniques only used in this toolbox. These techniques provides more or less improvement over existing methods. For the worst case, these techniques can be a useful supplementary to existing techniques.

3. *pmm* generates Noise Companion State-Space (NCSS) model that supports noise simulation, including small-signal AC noise analysis, phase noise analysis and transient noise analysis. This is exclusively supported in *pmm*. It is useful if the model is to be used in noise-critical circuits, and the passive components to be modeled play a non-negligible role in these circuits. Typical such applications include most RF/MM-wave module design, e.g. LNA, VCO, PA, etc.

1.3 Purpose of this toolbox

The intention of providing this toolbox is two-fold:

1. To help engineers, e.g. RF/MM-wave IC designers, model passive components, such as inductors, transformers, baluns, transmission lines, and etc, in their designs. Traditionally, such components are modeled in foundry and the designers are given the compact, RLC-based equivalent subcircuit models. However, it is now a trend that passive components are more and more customized by designers with EM simulators. Developing compact models, or even determine the element values in existing compact models, is a headache. *pmm* can be a useful tool to generate the required model that can be directly embedded in circuits to perform simulations.

2. To provide an easy-to-use tool for researchers in the area of passive macromodeling to have a platform to do apple-to-apple comparison between different methods. Sometimes, different algorithms need to work together to achieve good performance. In *ppm*, new algorithms can be easily plugged into the tool and replace one of the steps. This is useful as the algorithm developer only need to focus on one single step and does not need to implement the whole tool set.

1.4 License

1.5 How to acknowledge this tool

1.6 A Quick Start

Before using *pmm*, you should run the following command within Matlab environment to setup path and etc.

```
>> pmm_setup;
```

After this, the simplest way to use *pmm* is

```
>> pmm(infile, q);
```

In the above command *infile* is the name of a data file in touchstone format, q is an integer indicating the desired order of the resulting model. Notice that the actual size of the output system will be $n = mq$, where m is the number of ports of the system. Touchstone format is an industry-standard format, and it describes the transfer function matrix at separate frequency samples.

Notice that in the above syntax there is no output parameter given. In this case, by default *pmm* will parse the input filename, and extract the path to the file, and generate a netlist file in Spectre format with filename the same as the input filename and with extension ‘scs’, which is the typical extension of a Spectre netlist. The output file will be placed in the same directory as the input file.

Choosing a proper order q could be an issue for users unfamiliar to this tool. As a rule of thumb, let $q = 5, 10, 20$ for low order, medium order, and high order systems respectively, except for very high order systems such as filters. Generally, interconnects can be viewed as low-order systems, inductors, when operating under its self resonant frequency, can be viewed as low to medium order systems. Baluns and transformers can be viewed as medium to high order systems.

The modeling tool will automatically choose an appropriate set of algorithms to generate the model.

2 The Problem

2.1 The Basics

The task of macro-modeling is to find matrices A, B, C, D such that the transfer function of the following dynamic system

$$\begin{aligned} \dot{x} &= Ax + Bu && (1) \\ y &= Cx + Du && (2) \end{aligned}$$

i.e.

$$Y(\omega) = C(j\omega I - A)^{-1}B + D \quad (3)$$

approximates the measured or simulated transfer function matrices $\tilde{Y}(s_k)$ at frequencies s_k, $k = 1 \cdots n_s$.

This problem can be formulated as the following optimization

$$\begin{cases} \text{variables} : A, B, C, D \\ Y(s) = C(sI - A)^{-1}B + D \\ \min \sum_{k=1}^{n_s} |Y(s_k) - \tilde{Y}(s_k)| \end{cases} \quad (4)$$

The above optimization are usually solved in a two-stage manner. In the first stage, matrix A is determined. This is equivalent to determine the poles of the system. In a second stage, other matrices, the so-called residual matrices are determined. Notice that B and C are correlated and thus only one of them (usually C) needs to be treated as variable, and the other (usually B) can be fixed.

2.2 Two Types of Systems

Among the many types of representation that describe a physical dynamic systems, we consider the following two cases.

2.2.1 Hybrid Systems

Hybrid systems include impedance (Z-parameter) system and admittance (Y-parameter) system.

2.2.2 Scattering Systems

Scattering systems are used to describe the scattering (S-) parameters of a system.

2.3 Passivity

In simple words, the generated system needs to be passive. Passivity of the system plays an important role here. It is so important that a non-passive macro-model will anger engineers who are developing simulators.

A system is passive if it cannot generate energy. This is crucial since the interconnection of passive system is guaranteed to be stable. In other words, the interconnection of a stable, yet non-passive model and a passive system may be instable. And this instability is the root of evil–it will cause non-convergence in simulation.

Thus it is always demanded that the macro-model for a passive system is still passive. Mathematically, for hybrid (Y- or Z- parameter) systems, passivity requires that the Hermitian part of the transfer matrix must be semi-positive define on the imaginary axis, i.e.

$$G(s) = H(s) + H^H(s) \geq 0, \quad s = j\omega \tag{5}$$

For scattering (S-parameter) systems, passivity requires that the $\mathcal{H}_\infty$-norm to be always less than 1, or

$$\lambda_{\max}(H^H(s)H(s)) \leq 1 \tag{6}$$

A straightforward method to check the passivity of a system is to look directly at a set of discrete frequency points to see if the transfer matrix has a negative eigenvalue. However, with this kind of methods, the completeness is never guaranteed.

A more reliable method is to inspect the eigenvalues of the Hamiltonian matrix corresponding to the state-space system or its variants [1, 2].

Solving the modeling problem (4) is not very difficult, especially if the poles of the system (i.e. matrix A) is given. However, when passivity is required, the problem becomes much more difficult, and most of the time ill-posed.

Lets look at the passivity constrained problem.

$$\begin{cases} \text{variables} : A, B, C, D \\ Y(s) = C(sI - A)^{-1}B + D \\ \min \sum_{k=1}^{n_s} |Y(s_k) - \tilde{Y}(s_k)| \\ \text{s.t.} Y(j\omega) \geq 0 \ \text{ for } \ 0 \leq \omega \leq \infty \end{cases} \tag{7}$$

The key difference between this problem and (4) is the convexity. Problem (4) is convex, as the objective function is convex, and there is no constraints. The objective function of (7) is the same as (4), which is still convex. However, the constraints of (7) is non-convex, and this makes the whole problem non-convex, which is much more difficult than a convex problem.

2.4 Noise Modeling

A state-space model itself does not tell anything about the noise. It only describes the port parameter, e.g. with certain voltages applied on the ports, the system will generate how much current on the ports.

Physically, noise of a passive system is all due to resistive elements inside the system. Such elements generate thermal noise. To model noise of a passive system, noise current sources can be applied on the ports of the system. This results in a Noise-Companion State-Space model.. A tricky part here is that, although thermal noise generated by resistors is white noise, i.e. constant power spectrum density alone the frequency axis, the equivalent noise sources on the ports are frequency dependent. Details of solving this issue is described in our submitted work [3].

3 Advanced Usage

The more detailed syntax of calling *pmm* is as follows:

```
pmm(infile, q, opt)
```

where *opt* can be use to control the algorithm used, and many other settings in *pmm*.

The workflow of *pmm* is as follows. The philosophy behind this flow is that most of existing modeling algorithms require no only the S-parameter data, but also an initial model as the starting point. For example, most of existing passivity enforcement methods require an initial “non-passive” model as the starting point and perform iterative enforcement to fix the passivity. Some at least require the poles of the system are given.

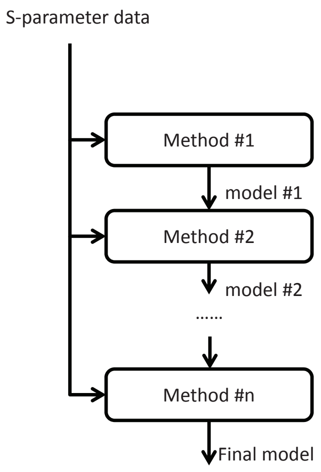

Figure 1: Workflow of *pmm*.

3.1 Built-in Methods

There are a collection of state-of-the-art modeling methods implemented in *pmm* as build-in methods. These methods have already been installed, and they can be invoked directly with input options.

3.1.1 VF: Vector Fitting

Vector fitting method [4] is use to find the state-space model to fit the input data without considering passivity issue. The key to this methods is to first find the proper poles through an iterative procedure. After finding the poles, the problem becomes a linear regression problem and can be solved easily.

Since almost all other methods require an initial model as the starting point, and the initial model has better to be accurate, it is almost always placed at the first of the sequence.

In this *pmm* toolbox, VF method is provided by its original author through the third-party package Matrix Fitting Toolbox.

3.1.2 SDP: Semi-Definite Programming

Semi-definite programming method, also called convex programming method [5], is a passivity guaranteed method. The key to this method is to convert the passivity-constrained optimization problem, which is non-convex, to a convex programming problem by using the Positive Real Lemma

$$\begin{bmatrix} -A^T K - KA & -KB + C^T \\ -B^T K + C & D + D^T \end{bmatrix} \geq 0 \tag{8}$$

$$K \geq 0 \tag{9}$$

The beauty of the above lemma is that it is convex, not like most other equivalent description of passivity constraint. Because of this, SDP method will always generate the proven optimal passive model (given fixed system poles). The drawback is that it is expensive when dealing with large-scale problems. This is partially because that the number of variables in the optimization is $O(n^2)$, which is roughly the same as the number of independent entries in matrix K. For small to medium size problems, it is the first method that should be tried.

In this *pmm* toolbox, SDP method is implemented following the description in [5]. Within the implementation, a third-party toolbox *cvx* is used for performing the convex optimization.

3.1.3 FRP: Vector Fitting

FRP method [6] detects passivity violation regions and based on which applies passivity constraints at discrete frequencies. This makes the problem easy to solve, and sometimes it shows much higher efficiency compared with other methods. The drawback is that it may sometimes encounter convergence issue.

In this *pmm* toolbox, FRP method is provided by its original author through the third-party package Matrix Fitting Toolbox.

3.1.4 EPM: Eigenvalue Perturbation Method

EPM method [1] uses Hamiltonian matrix for passivity violation detection. Purely imaginary eigenvalues of the Hamiltonian matrix correspond to boundaries of passivity violations. A straightforward idea is to iteratively move those paired imaginary eigenvalues towards each other. Conceptually doing this will compress and finally eliminate the violation region.

The drawback is still the convergence issue. When eliminating a violation, it is possible that one or more other violations are "created". If this occurs, the procedure may encounter convergence issue.

In this *pmm* toolbox, EPM method is implemented following the descriptions in [1, 7]. It should be emphasized that it is difficult to exactly repeat the work in the literature as some of the details are not provided.

3.1.5 LC: Local Compensation

Local compensation method [8] identify and fix passivity violations individually and locally by adding poles and residuals to the system. This method guarantees to converge, and it is quite efficient. The pay-off is that 1) the resulting system may be larger than the original one, and 2) it preserves the absolute accuracy, and sometimes the relative accuracy could be poor.

3.1.6 DAO: Domain Alternated Optimization

DAO method [9] is a non-convex optimization method. The philosophy of DAO method is that existing methods, such as FRP, EPM, LC, etc, usually provide solutions which are not optimal, even locally. With taking the solution of these methods as the initial model, DAO will perform local optimization and provide (more or less) accuracy improvement.

Notice that the quality of the resulting model from DAO highly depends on the initial model. When the initial model is sufficiently close to the global optimal solution, DAO method may finally reach the optimal solution. On the other hand, if the initial model is poor, then you should not expect solution to be very good.

Comparing to SDP method, the potential advantage is that the number of variables reduces from $O(n^2)$ to $O(nm)$. The pay-off is that as the problem becomes non-convex, the result is not globally optimal. Instead, it provides a local optimal solution.

3.2 Adding New Algorithms

pmm is designed from begining to be easily extended. New algorithms, called methods here, in the form of an interface *Matlab* functions can be "installed" and then inserted into proper stages as needed via option setting. In fact, all built-in methods are "installed" in such way and they are treated in the same way as external methods.

3.3 Interface Function

The method definition should be in the following form

```
[G,W]=MethodName(G0, W0, F, H, opt);
```

where the arguments are

- G_0: the starting SS system
- W_0: the spectral factorization of G_0 (optional)
- F: frequency samples (n_s-by-1, double)
- H: transfer function data (m-by-m-by-n_s, double)
- $opts$: options (optional)
- G: the output SS system
- W: the spectral factorization of G

G_0, W_0 and W can be left empty, e.g.

```
[G,W]=MethodName([], [], F, H, opt);
```

This is for some methods that do not need an initial model, e.g. VF method.

3.4 Installing New Methods

Installing new methods can be done via built-in function *pmm_install*.

```
pmm_install(Name, Desctiption, passive_in, passive_out);
```

where *MethodName* is a string which is the same as the name of the function to be installed. *Description* is a string for describing the algorithm. Input

- $MethodName$: Name of the function to be installed.
- $Description$: A brief description to the method.
- $passive\_in$: 1 if require the input system to be passive, 0 otherwise.
- $passive\_out$: 1 if the output system is guaranteed to be passive, 0 if no such guarantee.

Output: None

If $passive\_in = 1$, the passivity of the input system is ensured by the previous stage. If $passive\_out = 1$, *pmm* will check the passivity of the output system. If the system fails to pass the passivity check. Then *pmm* will abort without proceeding to the next stage.

Following commands are from pmm_setup.m.

```
pmm_install('VF', 'Vector Fitting', 0, 0);
pmm_install('SDP', 'SDP Method', 0, 1);
pmm_install('LC', 'Local Compensation', 0, 1);
pmm_install('EPM', 'Eigenvalue Perturbation', 0, 1);
pmm_install('FRP', 'FRP Method', 0, 1);
pmm_install('DAO', 'DAO Method', 1, 0);
```

3.5 Arranging Sequence of Methods

Methods can be arranged in a sequence to accomplish the modeling task. This is done with the input option *opt.Func*.

In order to apply *Method1*, *Method2*, $\cdots$, *MethodN* to generate the model, the following commands can be executed.

```
opt.Func{1} = ’Method1’;
opt.Func{2} = ’Method1’;
......
opt.Func{N} = ’MethodN’;
```

And then apply *pmm* as follows:

```
pmm(infile, q, opt);
```

4 Options Setting

4.1 Choosing Algorithms

There are already some combination of built-in methods pre-defined in *pmm* toolbox. Option opts.scheme can be used to choose these combinations easily.

```
opts.scheme='vf_only';  % VF
opts.scheme='sdp';      % VF SDP
opts.scheme='lc';       % VF LC
opts.scheme='epm';      % VF EPM
opts.scheme='frp';      % VF FRP
opts.scheme='lc_dao';   % VF LC DAO
opts.scheme='epm_dao';  % VF EPM DAO
opts.scheme='frp_dao';  % VF FRP DAO
```

For example, the following commands will tell *pmm* to first use VF to generate an initial model, and use LC to fix the passivity, and then use DAO to improve the accuracy.

```
opts.scheme='lc_dao';
pmm(infile, q, opts);
```

4.2 Weighting Scheme

Weighting scheme is used to control the weighting during curve fitting.

```
opts.wgt_scheme = 1; % No weighting
opts.wgt_scheme = 2; % weight(s)=1/abs(Hij(s))
opts.wgt_scheme = 3; % weight(s)=1/sqrt(abs(Hij(s)))n
opts.wgt_scheme = 4; % weight(s)=1/norm(H(s))
opts.wgt_scheme = 5; % weight(s)=1/sqrt(norm(H(s))
```

4.3 Parameter Type

opts.parametertype is used to tell *pmm* the input data is for scattering parameters or for hybrid parameters. Passivity conditions for these two kind of systems are different.

```
opts.parametertype = ’S’; % S-parameter system
opts.parametertype = ’Y’; % Hybrid (Y- or Z- parameter) system
```

4.4 Pole Type

opts.poletype is used to tell *pmm* how to choose initial poles in vector fitting methods.

```
opts.poletype = ’lincmplx’; %  linearly spaced,
                            %  complex conjugate pairs
opts.poletype = ’logcmplx’; %  logarithmically spaced,
                            %  complex conjugate pairs
```

4.5 Preserving DC

In some applications, DC operating points need to be preserved. In this case opts.preserveDC can be used to tell *pmm* to ensure that at DC operating points the transfer function be identical to the input data.

```
opts.enforceDC = 1; %  preserve DC operating point
opts.enforceDC = 0; %  don’t preserve DC operating point
```

4.6 Plot Control

opts.verbose is use to tel *pmm* whether or not to plot curves to examine the fitting accuracy, passivity violation, etc.

```
opts.plot = 1; % plot curves
opts.plot = 0; % don’t plot curves
```

4.7 Verbose Control

opts.verbose is use to tel *pmm* whether or not to print detailed information when running.

```
opts.verbose = 1; % output detailed information
opts.verbose = 0; % don’t output detailed information
```

5 Third-Party Packages

The following third-party packages are included in the release of *pmm*.

5.1 *cvx*

cvx is a Matlab toolbox for disciplined convex programming. It can be downloaded from http://cvxr.com/cvx/download/.

5.2 Matrix fitting toolbox

Matrix fitting toolbox is a Matlab toolbox for vector fitting and it also includes passivity enforcement routines.

Website: http://www.energy.sintef.no/produkt/VECTFIT/index.asp.

6 Application Examples

Provided in the demo/ directory are some S-parameter data files, these data files can be used to get familiar with this toolbox. Following we provide some examples for showing how to use this tool.

Remember, before running *pmm*, the following command should be first executed.

```
>> pmm_setup;
```

In the first example, we use the default setting in *pmm*.

```
>> [G,W]=pmm('demo/HHM1506.s3p',5);

opts =

             plot: 0
          verbose: 0
    parametertype: 'Y'
           method: 'auto'
         poletype: 'lincmplx'
              tol: 0
        enforceDC: 0
       wgt_scheme: 1
                q: 5
             Func: {'VF'  'SDP'}


  FuncName      Time          Error       Passivity
------------------------------------------------------
        VF  5.928038e-001  1.255528e+001  non-passive
       SDP  1.185608e+000  1.260059e+001  passive
```

It shows above that *pmm* gives the detail setting in opts. In this case, the method is set to 'auto', which means the tool will choose schemes automatically. The scheme chosen is VF+SDP, which is, as mentioned before, the best combination for small to medium size problems.

During the modeling process, the tool prints the information for each method in the queue. It gives the time elapsed by the methods, and the error of the resulting model. Also given is the passivity of the resulting model in each stage.

Next we would like to alter the scheme. The commands and the output are given below.

```
>> opts=pmm_default;
>> opts.method='lc_only';
>> [G,W]=pmm('demo/HHM1506.s3p',5,opts);

opts =

             plot: 0
          verbose: 0
    parametertype: 'Y'
           method: 'lc_only'
         poletype: 'lincmplx'
              tol: 0
        enforceDC: 0
       wgt_scheme: 1
                q: 5
             Func: {'VF'  'LC'}


  FuncName     Time            Error     Passivity
------------------------------------------------------
        VF  6.240040e-001  1.255528e+001  non-passive
        LC  4.992032e-001  1.282014e+001  passive
```

To alter the options, first the default option is obtained through

```
>> opts=pmm_default;
```

And then the scheme is altered by

```
>> opts.method='lc_only';
```

The rest is similar to the previous case, except that opts is given as an input

```
>> [G,W]=pmm('demo/HHM1506.s3p',5,opts);
```

The tool will generate a netlist in *spectre* format for the resulting model. The resulting netlist is shown below:

```
subckt HHM1506_ncp p1 n1 p2 n2 p3 n3
parameters A=[ -1.1954551885375362e+011 0.0000000000000000e+000 0.
parameters B=[ 1.0000000000000000e+000 0.0000000000000000e+000 0.0
parameters C=[ -9.4653083277303219e+009 1.1044190976108851e+008 2.
parameters D=[ 1.0110967206486376e-001 2.6662972333300282e-003 -5.
parameters Aw=[ -1.1954551885375362e+011 0.0000000000000000e+000 0
parameters Bw=[ -7.4535441984150299e+010 2.3523635976437593e+009 -
parameters Cw=[ 1.0000000000000000e+000 2.0000000000000000e+000 -0
parameters Dw=[ 4.4968805202020601e-001 0.0000000000000000e+000 0.
parameters noisetemp=27
parameters noiseref=2*P_K*(noisetemp+P_CELSIUS0)

Xmod (p1 n1 p2 n2 p3 n3) cktrom a=A b=B c=C d=D
I1 (p1 n1) cccs gain=-1 probe=V1
I2 (p2 n2) cccs gain=-1 probe=V2
I3 (p3 n3) cccs gain=-1 probe=V3
Xnoise (s1 0 s2 0 s3 0) cktrom a=Aw b=Bw c=Cw d=Dw
V1 (s1 0) vsource  noisevec=[1 noiseref]
V2 (s2 0) vsource  noisevec=[1 noiseref]
V3 (s3 0) vsource  noisevec=[1 noiseref]
ends HHM1506_ncp
```

References

[1] S. Grivet-Talocia, "Passivity enforcement via perturbation of hamiltonian matrices," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 51, no. 9, pp. 1755–1769, Sept. 2004.

[2] Z. Ye, L. M. Silveira, and J. R. Phillips, "Fast and reliable passivity assessment and enforcement with extended hamiltonian pencil," in *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 774–778.

[3] Z. Ye, "Noise companion state-space (ncss) passive macro-modeling for rf/mm-wave circuit design," submitted.

[4] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting," *Power Delivery, IEEE Transactions on*, vol. 14, no. 3, pp. 1052–1061, Jul 1999.

[5] C. Coelho, J. Phillips, and L. Silveira, "A convex programming approach for generating guaranteed passive approximations to tabulated frequency-data," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 293–301, Feb. 2004.

[6] B. Gustavsen, “Fast passivity enforcement for pole-residue models by perturbation of residue matrix eigenvalues,” *Power Delivery, IEEE Transactions on*, vol. 23, no. 4, pp. 2278 –2285, oct. 2008.

[7] S. Grivet-Talocia and A. Ubolli, “On the generation of large passive macromodels for complex interconnect structures,” *Advanced Packaging, IEEE Transactions on*, vol. 29, no. 1, pp. 39–54, Feb. 2006.

[8] T. Wang and Z. Ye, “Robust passive macro-model generation with local compensation,” *Submitted*.

[9] M. G. Z. Y. Zuochang Ye, Yang Li, “A novel framework for passive macromodeling,” 2011, pp. 546–551.