APPLICATION NOTE

Making Better Models with User Data

PATHWAVE

How to import custom, file-based model data into Genesys and SystemVue

Abstract:

RF system designers need a fast, intuitive architecture system and frequency-planning tool. Genesys and SystemVue provide users the option to create custom models with measured data by using the built-in MATLAB interface. This paper details the process of combining MATLAB and the RF System analysis to improve the accuracy of simulations via personalized models.

Data Entered into an Equation Page

Many of the models in the RF Design library used for RF System simulations have parameters that you might want to make a function, rather than a constant value. For instance, the Gain parameter of an Amplifier can be set to measured data of the gain vs. input power to simulate gain compression.

In Genesys and SystemVue, the language used to create variables, write expressions, and manipulate data is MATLAB. Using the built-in MATLAB, you can enter your own gain compression data for an amplifier in the following way:

```
1  % Parameters for schematic
2  RFpower = tune(-20);
3
4  % List of input power values (Independent variable)
5  Pin_List = [-25; -20; -15; -10; -5; 0; 5; 10];
6
7  % Gain values (Dependent variable)
8  Gain_List = [29.48; 29.42; 29.27; 29.06; 29.03; 28.93; 25.80; 20.75];
9  Phase_List = [64.06; 64.14; 64.35; 64.17; 64.28; 64.47; 56.41; 50.47];
10
11 AmpGain = interp1(Pin_List, Gain_List, RFpower, 'spline', 'extrap');
12 AmpPhase = interp1(Pin_List, Phase_List, RFpower, 'spline', 'extrap');
13
```

Let's take the lines one at a time. Line 5 creates an array called **Pin_List** that contains a list of input power values. The values are separated by semicolons. This is the x-axis of the gain compression curve.

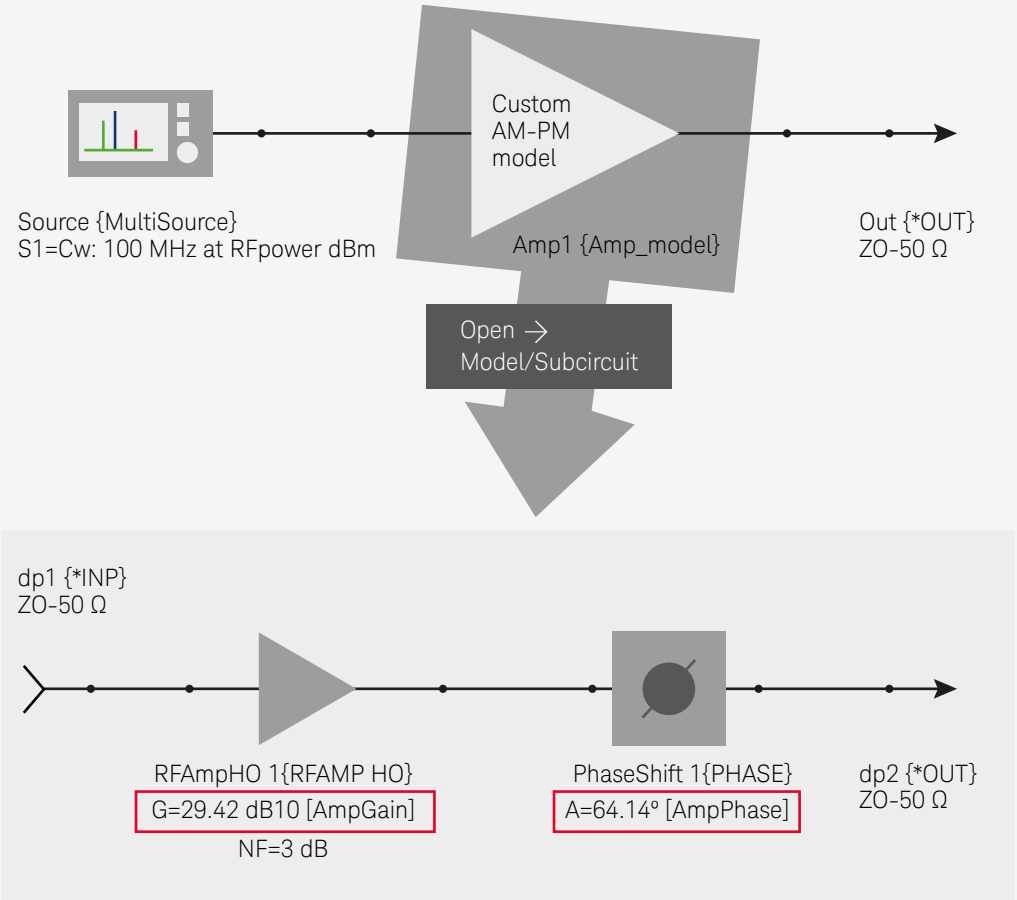
On line 8, **Gain_List** is created which is simply the y-axis of the gain compression curve. Notice that there is exactly one gain value for each power value— you must follow this rule.

KEYSIGHT TECHNOLOGIES

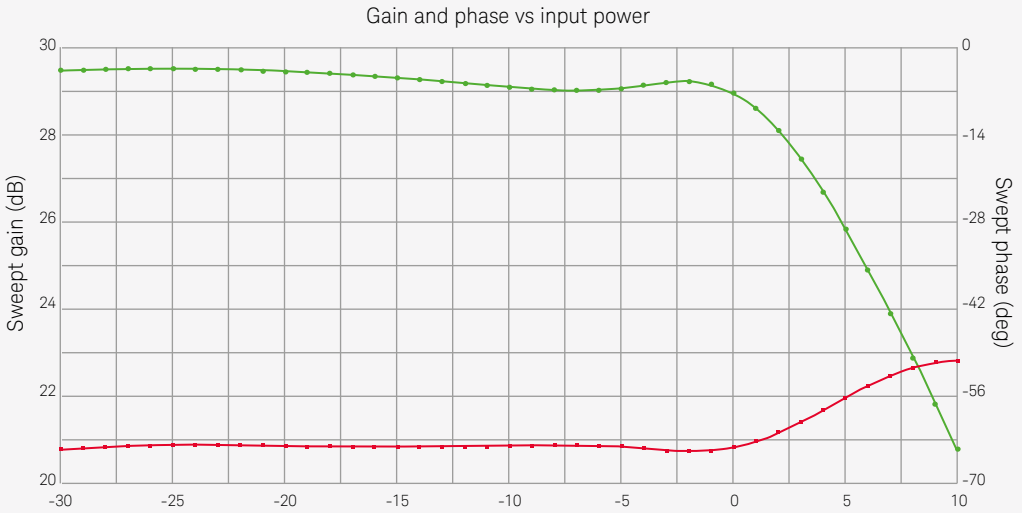
Line 9 creates a similar variable, **Phase_List**, that describes the phase at each input power point.

Lines 11 and 12 is where the x- and y-values are tied together into one variable for the gain, **AmpGain**, and another for the phase, **AmpPhase**. A very useful built-in function *interp1()* does this and also interpolates values between the points entered. So, based on the value of RFpower (which is nominally set to −20 on Line 2), an interpolated gain value will be read from the data.

The above expressions are placed on an Equations page. Then the AM-to-PM data are applied to parts on a schematic as shown below, by specifying the amplifier's gain parameter G as **AmpGain** and the phase shifter's phase A as **AmpPhase**:



Simulation with this amplifier results in gain and phase dependent on input power:



This technique can be used whenever you want to make a model's parameter a function of another variable (i.e. TOI vs. power, loss vs. temperature, impedance vs. bias voltage, etc.).

interp1() also allows other interpolation methods, as well as extrapolation. See its manual page for more information.

Reading Data from a File

Now, what if instead of entering values for the gain compression, you want to read that data from a file?

This is very simple to do. There is no special file format that you must follow.

Here is an example of amplifier data in a file, **amp_data.txt**, shown below.

There are four columns. The 1st column is the input power in dBm, the 2nd is the output power, the 3rd column is the gain in dB, and the 4th is the output phase in degrees.

amp_data.txt - Notepad

File Edit Format View Help

```
-30      -0.49287742      29.50712258      64.08360119
-29.84   -0.3283291       29.5116709       63.97625489
-29.68   -0.19440476      29.48559524      64.15289633
-29.52   -0.0067179       29.5132821       64.06938155
-29.36   0.14336463       29.50336463      64.08708468
-29.2    0.31138407       29.51138407      64.04408898
-29.04   0.47165585       29.51165585      64.0310008
-28.88   0.61285578       29.49285578      64.05977267
-28.72   0.77618139       29.49618139      64.03510223
-28.56   0.94334248       29.50334248      64.11641142
-28.4    1.1006172        29.5006172       64.03547444
-28.24   1.26045139       29.50045139      64.07536187
-28.08   1.41196329       29.49196329      63.94537186
-27.92   1.57439915       29.49439915      64.08353703
```

The equations necessary to read this file are:

```
18
19  cd (getworkspacedir);
20  PiPo_data = readvector('amp_data.txt');
21  Amp_Pi = PiPo_data(1:4:end);
22  Amp_Gain = PiPo_data(3:4:end);
23  Amp_Phase = PiPo_data(4:4:end);
24
25  AmpGain = interp1(Amp_Pi, Amp_Gain, RFpower, 'linear', 'extrap');
26  AmpPhase = interp1(Amp_Pi, Amp_Phase, RFpower, 'linear', 'extrap');
27
```

The magic begins on Line 20. You could certainly use *fopen()* and *scanf()* in the usual way. The *readvector()* function is a wonderful built-in function whose purpose is to read an entire file at once. Immediately after this line, the contents of "PiPo_data" are as shown at right.

Command Prompt

```
>> PiPo_data

PiPo_data =

   -30.0000
    -0.4929
    29.5071
    64.0836
   -29.8400
    -0.3283
    29.5117
    63.9763
   -29.6800
    -0.1944
    29.4856
    64.1529
   -29.5200
    -0.0067
```

See how the column values are all mixed up? We would like to separate them, and that's what lines 21 through 23 do.

When you see the syntax,

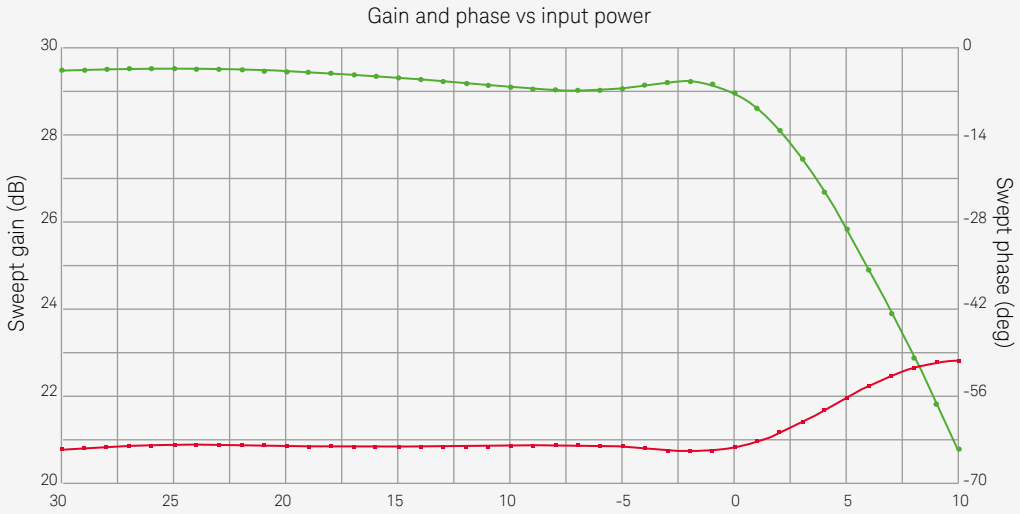
Amp_Pi = PiPo_data(1:4:end);

It means, "Start with the 1st value: collect every 4 values: go to the **end** of the file."

This will recover the 1st column only, which is exactly what we want. Similarly, line 22 does the same for the 3rd column, the gain values, and line 23 for the phases.

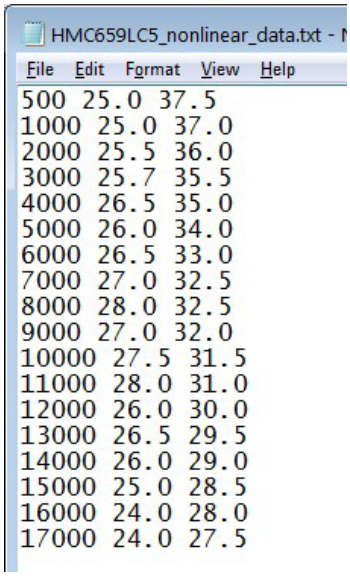
The first number inside the parentheses will always be the column number you want to keep. The middle number will always be the number of columns in the file.

Line 25 then creates the **AmpGain** variable in the usual way. Interpolation and extrapolation will be performed. The result is shown here:



Another example: If the 1st column were frequency, the 2nd column were P1dB, and a 3rd column were IP3, you can have those amplifier parameters as a function of input frequency. You can now make **any** parameter a function of **any other** parameter by repeating this technique.

For instance, here is another data file with exactly that information called **HMC659LC5_nonlinear_data.txt**:



The expressions to extract the separate columns would be:

File_data = readvector('HMC659LC5_nonlinear_data.txt');
Amp_freqs = File_data(1:3:end);
Amp_P1dBs = File_data(2:3:end);
Amp_IP3s = File_data(3:3:end);
.
.
.

Now for even better fidelity: Going back to the gain compression example, we used a variable **RFpower** as the x-axis value to do the interpolation. The exact expression was on line 25:

AmpGain = interp1(Amp_Pi, Amp_Gain, RFpower, 'linear', 'extrap');

Where does this RFpower variable come from? Ideally, we want to calculate the actual input power to the amplifier, no matter where it appears in a chain, and use that input power to interpolate a value for gain. How can we do this?

One way to do it is to estimate the RFpower at the amp and simply set it to that value. Another way is to pre-compute the RFpower with Genesys/SystemVue, then run the simulation with that power value.

Let's look at some more lines from that Equations page:

```
28
29  % The following lines 'pre-calculate' the input power
30  % to the amp so we can determine the correct AM-to-PM.
31
32  for i = 1:1:2
33  % Run the simulation once to pre-calculate nodes; then
34  % again to fine-tune.
35      runanalysis('System1');
36      using('System1_Data');
37  % P4 is the input node of the amp
38      RFpower = max(dbm(P4));
39  end
```

We're interested in lines 32 through 39. The result of this code is that each time these equations are evaluated, the **System1** simulation will run (twice) to converge on the correct value of input power and, thus, gain of the amp. Line 38 simply extracts this input power level, which is used in any other simulations done at a higher level. In this way, a correct value for input power will always be used when simulating this custom amplifier part.

As you can see, the *runanalysis()* function causes a simulation to run. It is frequently used to run multiple simulations in a desired order, script the same simulation multiple times, or simply to run a single sim.

getinterpdata()

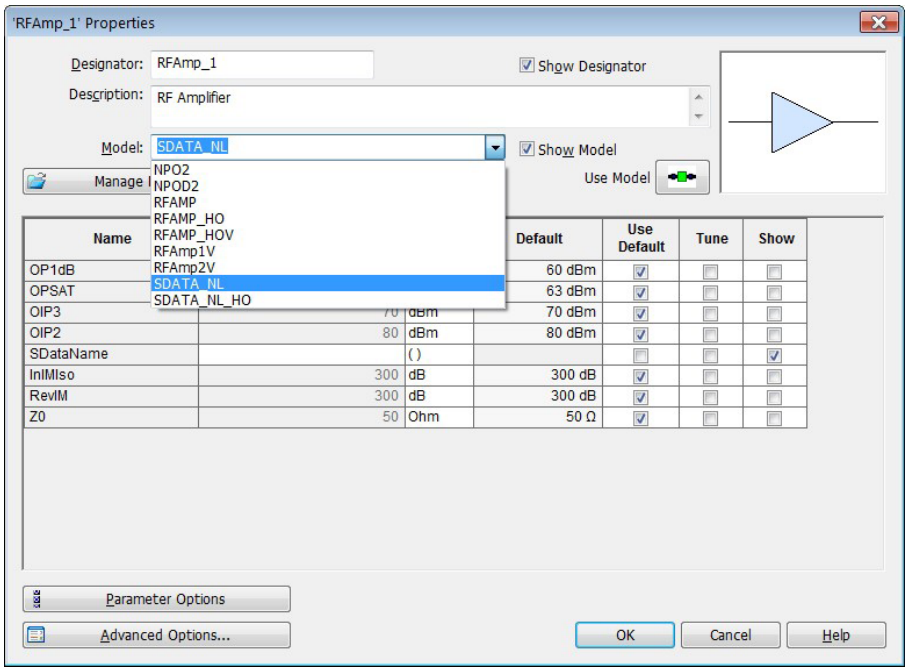
Recently a new function was created for dealing with multidimensional data: *getinterpdata()* can read and interpolate data that has multiple independent variables. This is very useful when dealing, for instance, with MDIF data files, or S2PMDIF data. There are three excellent examples of using this new function in the documentation; just search for "getinterpdata" in the manual.

Alternate Amplifier Models

The above AM-to-PM example has been made somewhat obsolete with the addition of the **AMtoPM_Mode** parameter of the Amplifier model, which allows you to enter lists of points of phase vs. drive, just like we did above. However, our example still holds as a generic guide for any data read from a file that you want to apply to a part on a schematic, which is a common requirement. But for an excellent example of this new built-in capability of the Amplifier, please see the Genesys example workspace:

Examples\Spectrasys\AM to PM\AM to PM Basics.wsg

The last thing to be aware of is the existence of the **SDATA_NL** amplifier model. Selecting this alternate model from the **Model:** option list…



…results in a model that takes S-parameters as well as nonlinear parameters OP1dB, OPSAT, OIP3, and OIP2. [There is no separate parameter for noise figure because noise data can be included within the S-data file.] This model is an excellent intermediate step between the simple behavioral models in the RF Design library and a high-fidelity X-parameter model – the ultimate nonlinear model that can be consumed in Genesys and SystemVue. It is often used when you have S-data plus nonlinear parameters from a datasheet for a part.

It's easy to create custom models with measured or other data using the built-in MATLAB in Genesys and SystemVue. The RF System analysis runs with either built-in or custom-built models. The accuracy of the simulation improves when you replace RF System's built-in models with your own more realistic models.

Learn more at: www.keysight.com

For more information on Keysight Technologies' products, applications or services, please contact your local Keysight office. The complete list is available at: www.keysight.com/find/contactus

KEYSIGHT TECHNOLOGIES

This information is subject to change without notice. © Keysight Technologies, 2019, Published in USA, March 5, 2019, 5992-2433EN