The Spectre/RF MATLAB Toolbox

The MATLAB® Toolbox provides an interface between both the Spectre and UltraSim® circuit simulation technologies and the MATLAB data manipulation environment. This section describes how to use the toolbox to read Spectre simulation results into MATLAB and to use MATLAB to perform some standard RF measurements.

The toolbox provides all Spectre and SpectreRF users with an alternative method for post-processing simulation data and making standard RF measurements in MATLAB. The toolbox allows experienced SpectreRF users to use Simulink® and MATLAB to make customized measurements on information extracted from Spectre simulation results. Experienced users can also perform high-level design tasks in Simulink and MATLAB.

MATLAB, a powerful mathematical and graphic tool, provides rich data processing and display functionality. Many users want to customize their measurements and displays. The toolbox supports these use models.

The MATLAB Toolbox includes a number of functions you can use to

- Read Spectre simulation results in PSF or SST2 format into MATLAB
- Perform basic data filtering and plotting tasks
- Support typical RF measurements such as IP3 and compression point

Install the Toolbox Package

The MATLAB Toolbox is a standalone package shipped with MMSIM version 6.1 and higher. The home directory for the toolbox is

```
<instdir>/tools/spectre/matlab
```

where <instdir> is the installation directory for the MMSIM simulation software. In the MATLAB Toolbox there are 3 MEX-files and 12 M-files.

Tip

Use the `cds_root spectre` command to determine the installation directory of your MMSIM simulation software.

Configure the Toolbox Package

The Spectre/RF Toolbox in MATLAB depends on the Spectre simulation run environment. It uses shared libraries located in the Spectre installation path.

1. Make sure you are running Spectre 6.0 or a higher version.
2. Verify the dynamic library path by checking the `LD_LIBRARY_PATH` environment variable. Make sure that both `<instdir>/tools/dfII/lib` and `<instdir>/tools/lib` are in the path. For C shell users, use the following command

```
setenv LD_LIBRARY_PATH `cds_root spectre`/tools/dfII/lib:`cds_root spectre`/tools/
lib:${LD_LIBRARY_PATH}
```

3. Solaris users also need to add the toolbox installation path to `LD_LIBRARY_PATH`. For C shell users, use the following command

```
setenv LD_LIBRARY_PATH `cds_root spectre`/tools/spectre/matlab:${LD_LIBRARY_PATH}
```

4. The MATLAB script sets the `MATLABPATH` environment variable to include the MATLAB toolbox directories and the user created directories. You need to add the toolbox installation path to the `MATLABPATH` environment variable. For C shell users, use the following command

```
setenv MATLABPATH `cds_root spectre`/tools/spectre/matlab:${MATLABPATH}
```

The Basic Toolbox Functions

Each toolbox function command has an associated help page which you can display by typing `help <command_name>` in MATLAB. This section introduces the basic toolbox functions, describes how to use each function and describes how to write measurement functions.

cds_srr

Lists the datasets in the result directory, lists signals in a dataset or reads a signal into MATLAB.

1. To list the datasets in the result directory, use the command

   ```
   datalist = cds_srr('result_directory')
   ```

   All `dataset` names in the result directory are returned with *`datalist`* as a string vector.

2. To list the signal names in a dataset, use the command

   ```
   signals = cds_srr('result_directory', 'dataset_name')
   ```

   All `signal` names and `property` names in the `dataset` are returned with *`signals`* as a string vector.

3. To read a signal into MATLAB, give both a `dataset` name and a `signal` name and use the command

   ```
   signal = cds_srr('result_directory', 'dataset_name', 'signal_name')
   ```

The value of the `signal` is returned with *`signal`*. The `signal` can be a single value or a structure containing a matrix.

Normally, when you use **`cds_srr`** to read a signal into MATLAB, **`cds_srr`** returns a structure containing fields. The first field is `info`, a string vector of `name, unit` pairs. In each pair, the first value is the `name` of the field, the second value is the `unit` of the field. The first `name, unit` pair in the `info` field describes the final value, the other pairs describe the sweep information if the signal was swept. An inner sweep is always listed before an outer sweep in `name, unit` pairs in the `info` field. The innermost sweep can be a matrix with variable sizes in each column, the other sweeps have to be a vector of fixed size.

In MATLAB,

- If a semi-colon (`;`) is at the end of a command, the system will not display the return results.
- Two single quotes (`''`) mean empty data.

Tip

The **cds_srr** command normally returns informational messages. To turn off these messages, use a zero (0) as the fourth parameter. For example, the following command runs silently.

```
datalist = cds_srr('results_directory', '', '', 0);
```

The **cds_srr** command supports both PSF and SST2 formats. **cds_srr** is an external function with **cds_innersrr** running in the background. The **cds_srr** and **cds_innersrr** commands have the same interface, but **cds_srr** adds some post processing to make the resulting matrix easier to read.

cds_evalsig

Filters the data from swept analyses. The command is

```
v = cds_evalsig(signal, expression)
```

The first parameter *`signal`* is the result of the **cds_srr** command. The second parameter *`expression`* is a string expression. You can use both relational operators (`<, <=, ==, >=, >>`) and logic operators (`&, |`).

If the *`signal`* has the swept fields `prf` and `time`, you can write an expression like the following,

```
prf < -20 | prf == -10 & time >= 2e-8
```

The logic operator `(|)` is only effective between expressions with the same field name. The following expression is meaningful.

```
prf==-10 | prf >= -20
```

When the logic operator `(|)` is used between two different field names, it is equivalent to the logic operator (`&`). The following two expressions are equivalent.

```
prf==-10 | time <= 2e-8
```

and

```
prf==-10 & time <= 2e-8
```

cds_plotsig

Plots swept signals. The command is

```
cds_plotsig(signal, expression, sweep_name, type_id)
```

The first two parameters *`signal`* and *`expression`* have the same meaning as described for the **cds_evalsig** command. The third parameter *`sweep_name`* is the name of a sweep field. You can define a special x-axis with *`sweep_name`* when the result has multiple sweeps. The default value for *`sweep_name`* is the name of the innermost sweep. The fourth parameter *`type_id`* is a string type id that can be any of the following: `mag`, `phase`, `real`, `imag`, `both`, `db10`, `db20` or `dbm`.

cds_harmonic

This command is specifically designed for RF results. It helps you to select harmonics and sidebands. In the toolbox data structure created by the **cds_srr** command, all sweep field names are listed in the `info` field. The harmonic information is not an independent sweep, it always combines frequencies and tones and it is not listed in the `info` field. The `harmonic` and `harmUnit` fields contain the harmonic information. The **cds_harmonic** command can select the interesting harmonics with these pieces of information. The command is

```
hsig = cds_harmonic(signal, harms)
```

For the PSS analysis, `harms` is a single integer. For the QPSS analysis `harms` is a vector. For example, use the following command for a QPSS analysis.

```
ord3 = cds_harmonic(rfout, [2 -1])
```

cds_interpsig

The intervals between time points in Spectre results are not equal. We provide an interpolation command to distribute the time points evenly. The command is

```
v = cds_interpsig(signal, sweep_name, num, method)
```

Where the first parameter *signal* results from other commands such as **cds_srr**. The second parameter *sweep_name* is the name of the inner sweep—normally it is `time`. The third parameter *num* is the vector length after interpolation. The fourth parameter *method* specifies a method for interpolation—it is a string value which can be `linear`, `cubic`, `nearest` or `spline`. The default interpolation method is `linear`.

Only the first parameter is required.

The Measurement Commands

cds_fft

The FFT (Fast Discrete Fourier Transform) is a typical RF measurement. MATLAB provides an internal `fft` function. The command **cds_fft** calls the internal MATLAB `fft` function. It prepares the input signal, and calls `fft` for each outer sweep. The command is

```
fsig = cds_fft(tsig)
```

The input parameter *tsign* is a time-domain signal. The output parameter *fsig* contains frequency domain data.

Tip

The **cds_fft** command is a MATLAB script. It is located in the file `cds_fft.m` in the installation directory. If you want to write your own measurement, this script offers a good example.

In `cds_fft.m`, the script checks the input value at the beginning. The *tsig* should exist and not be empty. As a time sweep result of **cds_srr**, it must have the fields `info` and `time.` Because the intervals between time points in *tsig* vary, the script calls **cds_interpsig** to distribute the time points evenly. From the vector of field times, the script gets the frequency vector. *tsig* can also be a multi-level sweep, so it does an fft for each outer sweep. After the fft, it copies additional sweep information from the *tsig* to *fsig*, and fills the field `time` with the field `freq`.

cds_compression

Returns the n-dB input referred compression point for the wave supplied. The command is

```
comp = cds_compression(vport, iport, harm, rport, gcomp, curve)
```

Where the first parameter *vport* is the voltage of the port. Normally it is the return value of **cds_srr**. The second parameter *iport* is the current of the port. It can be calculated from the voltage and impedance. The third parameter *harm* is the 1st order harmonic, the default value is 1. The parameter *rport* is the impedance of the port, the default value is 50. The parameter *gcomp* is gain compression in dB, the default value is 1. The parameter *curve* is the control flag of curve display. The default value of curve is `on`. It can be `off`.

cds_ipn

Returns the Intercept Point for the wave supplied. The command is:

```
[ipn_in, ipn_out] = cds_ipn(vport, harmspur, harmref, epoint, rport, ordspur,
iport, epref, ordref, curve)
```

The parameters of **cds_ipn** are similar to the parameters of **cds_compression**. The first parameter *vport* is the voltage of the port. The second parameter *harmspur* is the order harmonic value. The third parameter *harmref* is the reference order (1st order) harmonic value. The parameter *epoint* is the input power extrapolation point. The parameter *rport* is impedance of the port. The default value is 50.0. The parameter *ordspur* is order number. The default value is 3. The parameter *iport* is the port current. The default value is *vport*/*rport*. The parameter *epref* is the input power reference point. It will use *epoint* when this parameter is not specified. The parameter *ordref* is the reference order number. The default value is 1. The parameter *curve* is the control flag of curve display. The default value is `on`. It can be `on` or `off`.

The return parameter *ipn_in* is the input reference IPn value and *ipn_out* is the output reference IPn value.

Example

This section shows how to use the toolbox to make RF measurements in MATLAB. The example circuit is an LNA design from the RF workshop which is available at

```
<instdir>/tools/spectre/examples/SpectreRFworkshop
```

Please refer to *Lab3: Gain Compression and Total harmonic Distortion (Swept PSS)* in the workshop *LNA Design Using SpectreRF*.

After finishing the LNA Lab3, you get the results in

```
simulation/Diff_LNA_test/spectre/schematic/psf
```

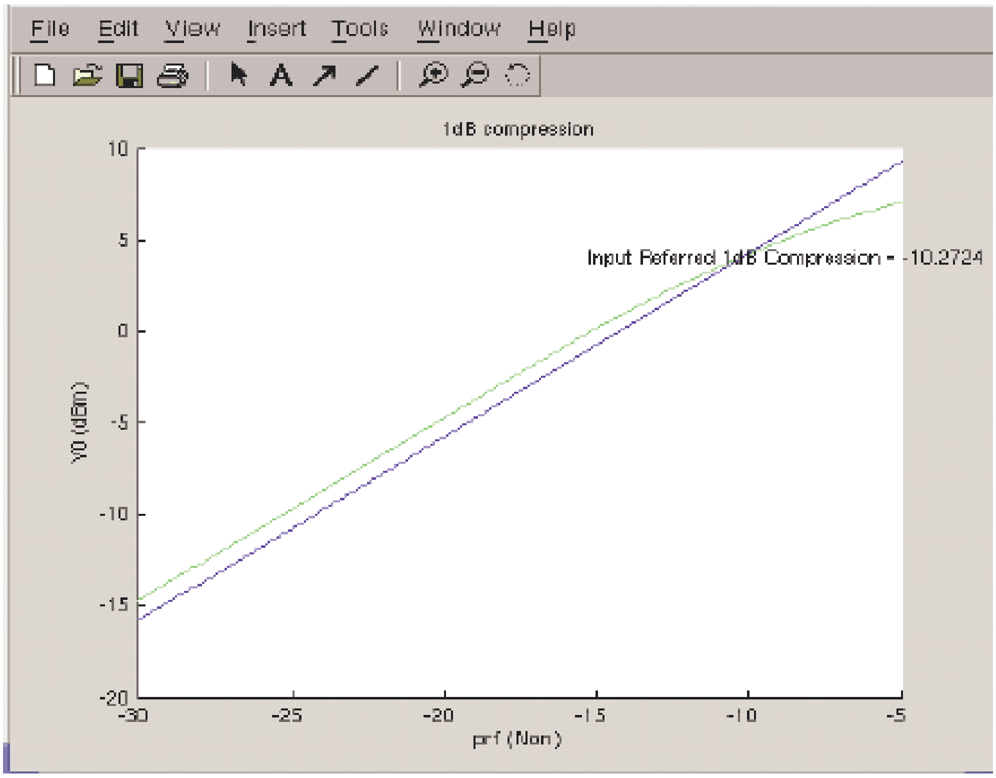
To start MATLAB from the system prompt, type the following command

```
matlab
```

In MATLAB, get time domain data with the command

```
>> rdir = 'simulation/Diff_LNA_test/spectre/schematic/psf';
>> tdout = cds_srr(rdir, 'sweeppss_pss_td-sweep', 'RFout');
```

Figure 1-20 Gain Compression (1dB Compression)



To get 1 dB compression point, use the following command

```
>> cds_compression(fdout)
```

A window with the result will pop up, as Figure 1-20 shows.

tdout is a power sweep result. For different `prf`, the time point numbers are different, and for each `prf`, the intervals between time points vary. The field *time* is a 161x11 matrix as the field `V`.

```
Execute command:
    >> td161=cds_interpsig(tdout);
    >> td100=cds_interpsig(tdout, 'time', 100);
    >> hold on
    >> cds_plotsig(tdout)
    >> cds_plotsig(td161)
    >> cds_plotsig(td100)
```

The waveforms shape of td161 and td100 are the same as *`tdout`*. But in td161 and td100, the time points are the same for each different `prf` and the intervals of time points are the same for each `prf`. So, the field time is stored and handled as a vector.

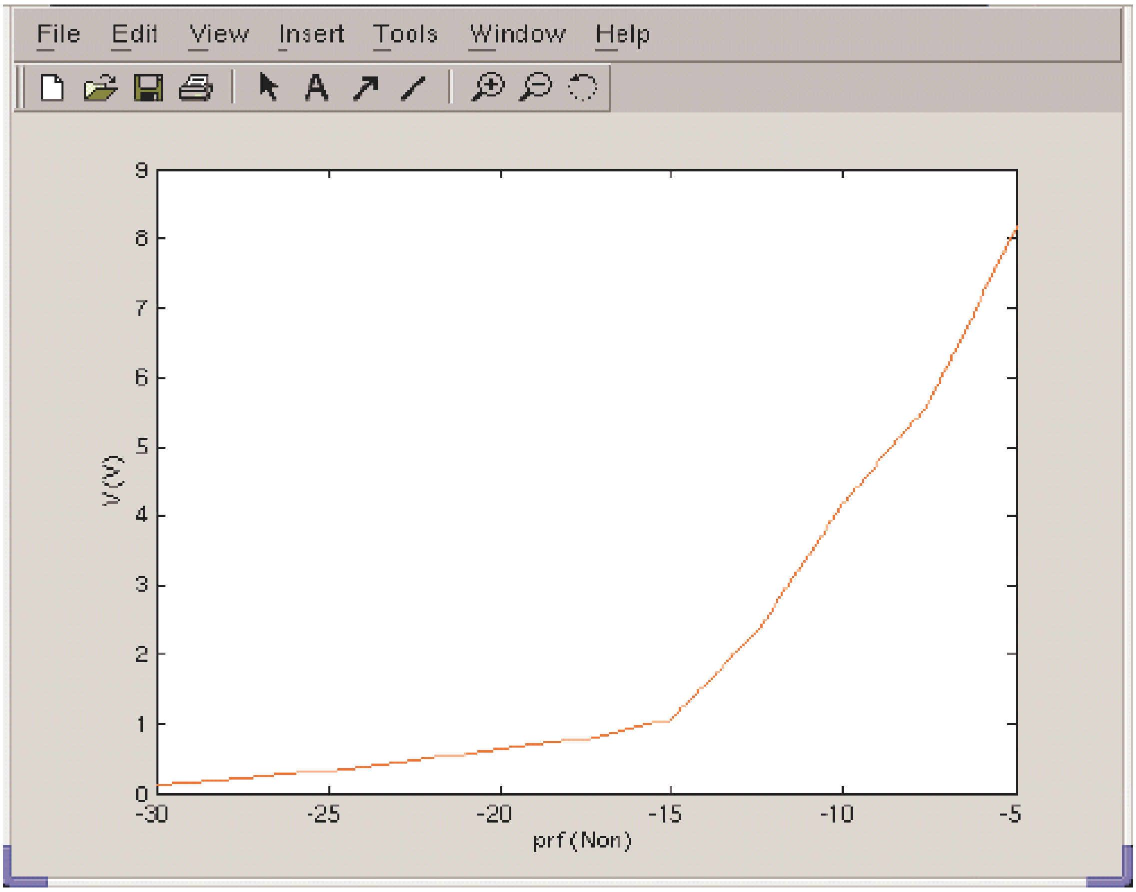
To get the frequency domain data, execute command:

```
>> fdout = cds_srr(rdir, 'sweeppss_pss_fd-sweep', 'RFout');
```

`fdout` contains 11 harmonics, the field `harmonic` lists the harmonic number. To get the 1st order harmonic signal, use the following command

```
>> fd1 = cds_harmonic(fdout, 1)
```

Figure 1-21 Total Harmonic Distortion



To get total harmonic distortion in percent, use the following command

```
>> thd = cds_thd(fdout);
>> cds_plotsig(thd);
```

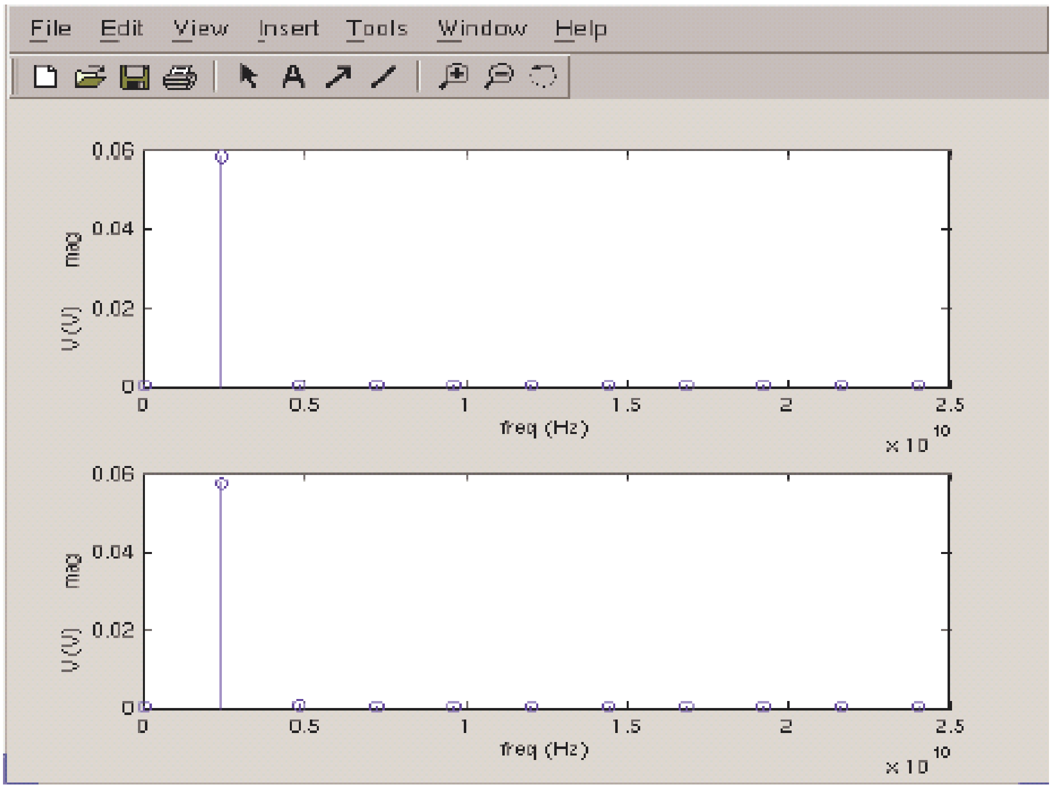
The result is shown in Figure 1-21.

To compare the frequency data with time domain data.

```
>> fd1 = cds_evalsig(fdout, 'prf = -30');
>> td2fd = cds_fft(tdout);
>> fd2 = cds_evalsig(td2fd, 'prf = -30 & freq<=2.5e10')
```

The result is shown in Figure 1-22.

Figure 1-22 Compare FD data before FFT and after



Compatibility with Aptivia MATLAB Functions

This topic is of interest to those users who are using acv measures in MATLAB. The document for these measures is in `<cdsinst>/doc/matlabmeasug/` Chapters 3 and 4.

The Spectre/RF MATLAB Toolbox is better able to handle both swept analyses and RF-related analyses such as Monte Carlo and PSS/PAC than acv measures. While acv measures provide more measurements for the transient and ac analyses, the MATLAB Toolbox can reuse acv measure functions with the command **cds2vsde** and the global variable `CDS2ACV`.

Set the global variable `CDS2ACV` to 1 in MATLAB with the following command.

```
>> global CDS2ACV; CDS2ACV = 1
```

When `CDS2ACV=1`, **cds_srr** prints `vsde compatibility information` after loading in the dataset. **cds_srr** will translate the signal to acv data structure for acv measures automatically.

The **cds2vsde** command is designed for translating signals to acv data structure. The command is

```
cds2vsde(sig, raw_file, variable, option)
```

The parameters *`raw_file`* and *`variable`* are string values to identify the signal. The parameter *`option`* can be `add` or `replace` as other acv functions.

Note: The acv data structure cannot handle multi-level sweeps, and for each *`raw_file`* the sweep value should be the same. So **cds2vsde** will use different variables or different *`raw_files`* for multi-level sweep signals.

If you want to convert variable *`td2fd`* to acv format, use the following command

```
>> cds2vsde(td2fd, 'sweeppss_pss_fd-sweep', 'td2fd');
```

It will print out:

```
vsde compatibility information
raw_file: sweeppss_pss_fd-sweep
variable: td2fd(prf=-30)
variable: td2fd(prf=-27.5)
variable: td2fd(prf=-25)
variable: td2fd(prf=-22.5)
variable: td2fd(prf=-20)
variable: td2fd(prf=-17.5)
variable: td2fd(prf=-15)
variable: td2fd(prf=-12.5)
variable: td2fd(prf=-10)
variable: td2fd(prf=-7.5)
variable: td2fd(prf=-5)
```

Then use command **meas_plot** for plotting:

```
>> meas_plot('sweeppss_pss_fd-sweep', 'td2fd(prf=-20)', 'stem');
```

Reference

[1] *Simulation Results Reader User Guide*, Product Version 5.0

[2] *MATLAB External Interfaces Reference* available at

`http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html`

[3] SpectreRF Workshop--LNA Design Using SpectreRF, MMSIM6.0USR2

Noise Separation in Pnoise and Qpnoise Analysis

This section describes how to analyze RF circuits using the noise separation features in the Pnoise and Qpnoise analyses. SpectreRF users in the Analog Design Environment (ADE) will find noise separation information to be useful.

Principles of Noise Separation in RF Circuits

For the Pnoise and Qpnoise analyses, input noise can be either *stationary* or *cyclostationary*. A simple instance of stationary noise is the white noise in a resistor with constant resistance. Cyclostationary noise is generally due to the fact that in most RF circuits the operating point of the nonlinear devices, primarily transistors, is periodic and time-varying.

To better illustrate the difference between stationary noise and cyclostationary noise, consider the white thermal noise generated by either the nonlinear drain-to-source or channel resistor in a MOS transistor. The formula for *white thermal noise* is given as

(1-34) $$\sqrt{4KTR(V)}$$

In Equation 1-34, $R(V)$ is the bias-dependent small signal drain-to-source resistance. Assuming the transistor is driven by a periodic large signal excitation, for example the clock in a switched capacitor filter, $R(V)$ will be time-varying and you can no longer model its associated thermal noise as a simple stationary noise source. You must treat the noise source as a cyclostationary random process.

When input noise passes through an RF circuit, the *aliasing* or *noise folding* effect is introduced by the frequency translation intentionally performed by the linear periodic time-varying (LPTV) characteristics of the RF circuit. You can summarize the noise transfer process as follows.

- First, for noise that is bias dependent, the time-varying operating point modulates the noise sources
- Second, the transfer function from the noise source to output modulates the noise source contribution to the output