MATLAB Muwave Toolbox
User's Guide

Christian Fager
christian.fager@chalmers.se

April 14, 2009

Contents

1 Introduction

The *Matlab-Muwave toolbox*[1] is a collection of functions to facilitate easy handling of S-parameters in the MATLAB [2] environment. The computational power and programming framework of MATLAB can thereby be applied to calculations involving measured or simulated data. This makes it particularly suited for device modelling applications, for which e.g. CAD tools are of limited use.

The purpose of this document is to describe the basic functionality offered by the toolbox in order for new users to get started and acquainted to the most commonly used functions.

1.1 Further help

For a more detailed description of each function encountered in this text, users are referred to the Matlab-Muwave Reference manual and on-line help provided by typing `>> help <command name>` or `>> helpwin <command name>` at the MATLAB command prompt. It is, however, assumed that readers of this document are reasonably familiar with the general MATLAB environment.

1.2 Feedback

Muwave is under continuous development. Most certainly you will find bugs and annoying inconsistencies as you start using it. It is therefore of great help for us to get feedback on features that are missing or if errors are found. Therefore, please do not hesitate to contact us if you have anything you are wondering about, report errors you have found, or if you have developed your own functionality that you want to share.

Kristoffer Andersson
kristoffer.andersson@chalmers.se

Christian Fager
christian.fager@chalmers.se

[1] Muwave is hereafter used as a short-hand notation for the MATLAB -Muwave toolbox.

[2] MATLAB is a trademark of The Mathorks Inc., Natick,MA, U.S.A

2 Installation

Muwave has been developed using MATLAB v.6.5 (Release 13), which is assumed to be installed on the user's computer. Please check the Help/About menu to make sure you are running this or a later version.

The installation of the Muwave-toolbox consists of a few simple steps:

1. Download the latest version from the Muwave web page at
   http://www.chalmers.se/mc2/EN/laboratories/microwave-electronics/education/graduate-courses/empirical-modeling.
   The downloaded zip-file contains all MATLAB m-files and documentation necessary.

2. Extract all files to a directory of your choice. In WinZip, make sure to have the "Use folder names" option checked.

3. Add Muwave to MATLAB 's path. This is done from the File menu by selecting "Set Path...". Click "Add Folder" in the dialog box that appears and browse to select the "muwave" directory, which was created where you extracted the zip-file. Please make sure to press the "Save button" before closing the Set Path dialog box.

4. Normally, Muwave is now ready for action. However, it may in rare cases be necessary to execute the following line before Muwave is used the first time:

```
>> clear classes; rehash;
```

You can verify that Muwave has been successfully installed by creating an empty measurement object. This is done by executing: `>> meassp`. The following should be displayed at the command prompt:

```
>> meassp
Measurement info
    Date : 18-Oct-2004 20:33:57
    Origin :
    Operator :
    Info :
Measurement state
    MeasType:   SP
Measurement Data
xparam-object
    type:  S
    reference: 50
    ports: 2
    elements:  0
```

3 Muwave data hierarchy

Before describing how to use it, some effort will be spent on briefly explaining the data structure used to store measurement data in Muwave.

3.1 Overview

A typical S-parameter measurement is in the form of 2-by-2 complex matrices versus n frequency points. Most calculations such as gain, impedance, and stability are then carried out on a "per-frequency" basis.

The way in which Muwave stores this data is a complex three dimensional ($2 \times 2 \times n$) matrix. MATLAB , however, has no built in functionality to perform operations in this manner—it simply does not understand along which dimensions it should perform the operations.

Muwave, however, uses the object-oriented capabilities offered by MATLAB to define a low-level *S/Y/Z/T-parameter data class*, for which all common MATLAB operations, such as *, +, -, /, ./ etc. have been redefined to operate "per-frequency". On a higher level, an *S-parameter measurement class* has been defined, which handles information about the frequencies, the bias settings, the measurement file source, etc. All these properties are automatically populated by parsing the measurement files when read into MATLAB .

Fig. 1 shows an overview of the different Muwave classes implemented, and their respective purpose. A short description of each of them is given below.

3.2 meassp

meassp is the top level class for handling S-parameters. All functions for reading single measurement files generate meassp-objects. The resulting measurement information is then contained its three sub-classes: *measmnt*, *measstate*, and *xparam*.

3.3 measmnt

measmnt is intended to store information about the measurement. This can typically include date of measurement, the measurement operator, the file origin, the transistor type etc.

3.4 measstate

measstate contains numerical data describing the measurement conditions. Typically, this is gate bias voltage, drain current, gate width, pulse length, temperature, etc.

3.5 xparam

xparam contains the actual S-parameter data and the corresponding frequencies. Or, in fact, it can be data of either S, Y, Z, or T-type. The class keeps track of which type of data it contains and has built in functionality to convert between them. Moreover, it stores the reference impedance so that S-parameters may be converted to other impedances if required. The numerical complex S/Y/Z/T-data is stored in an *arraymatrix* object.

3.6 arraymatrix

arraymatrix is the low level class that is used to implement matrix multiplication, inversion, conjugate transpose, addition etc. “per frequency”, as is needed e.g. for conversion between Z- and S parameters.

3.7 Additional classes

In addition to the classes described above, there exists also a few other classes that handle sweeps of measurements and model/netlist generation. These are described later in section 8 and section 7, respectively.

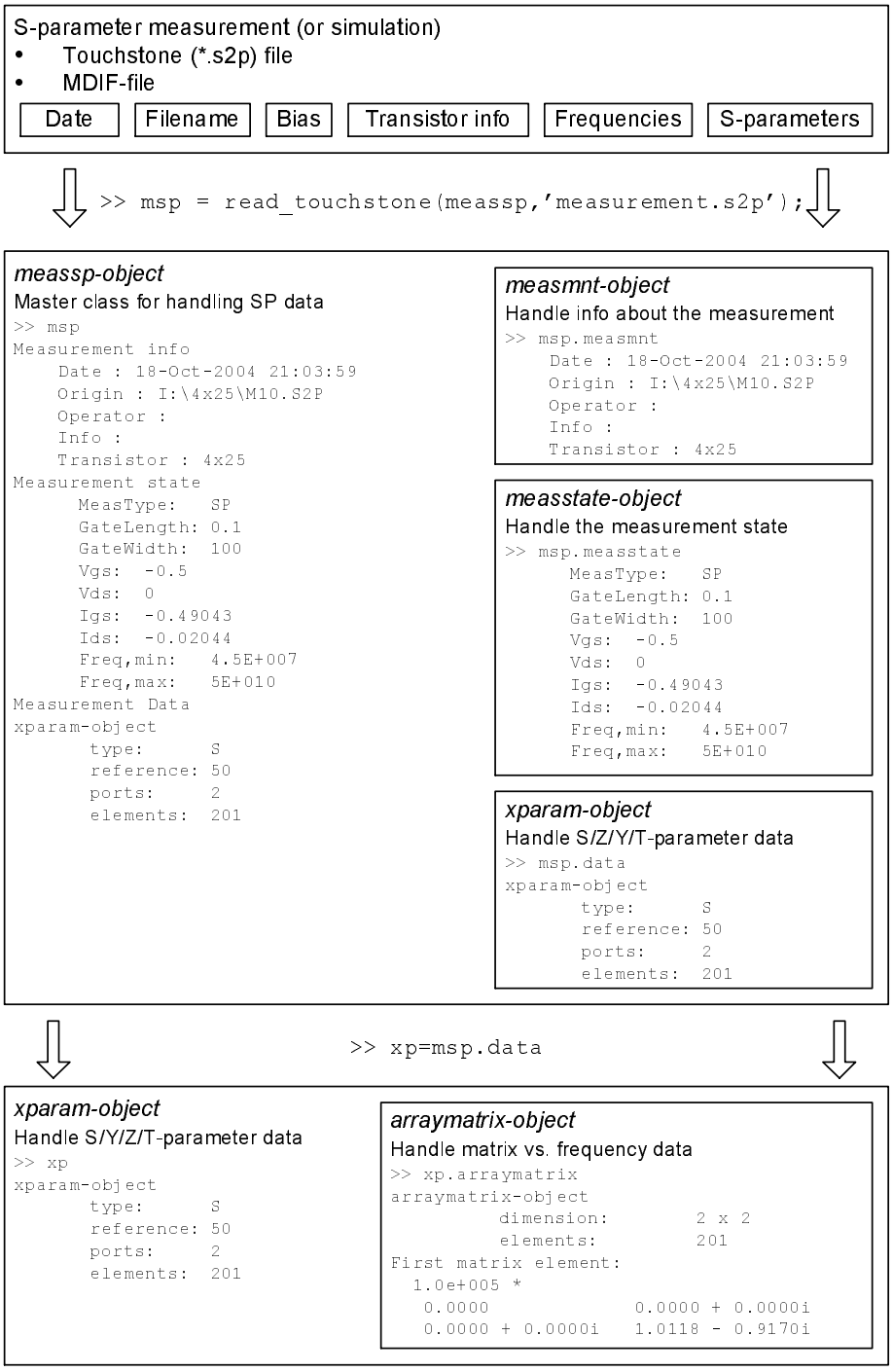

Figure 1: Overview of the Muwave class hierarchy.

4 Reading and storing single measurement files

The easiest and best way to introduce the functionality of Muwave is probably by an example. The example measurement used here is for an InP-HEMT device biased in the active region. The measurement is stored as a TouchStone file at:
`muwave/test/d0509/d0509_20`.

4.1 The TouchStone file format

The most commonly used file format for storing S-parameter files is the TouchStone format[3]. These files are typically identified by a .S2P file extension (although not the case for the example file above), where 2 denotes the number of ports in the measurement data. Below is shown an example of how the first couple of lines of a Touchstone file typically look like:

```
!Date: 2002-10-25, 08:50

!Vgs = -1.000000E-1
!Vds = 1.500000E+0
!Ig = -2.784000E-7
!Id = 1.974500E-2

#HZ S RI R 50
1.000000E+9 9.921265E-1 -6.970215E-2 -3.964478E+0 ...
1.490000E+9 9.887085E-1 -1.027222E-1 -3.976196E+0 ...
1.980000E+9 9.827881E-1 -1.362915E-1 -3.920410E+0 ...
2.470000E+9 9.783020E-1 -1.674194E-1 -3.943237E+0 ...
2.960000E+9 9.721069E-1 -1.990662E-1 -3.927979E+0 ...
    ...
```

Rows starting with exclamation marks (!) indicate comments, but are here used to store additional information about the measurements. In this case, which is typical, the comments reveal the biasing conditions as well as the date of measurement.

The next important section is the line starting with `#`. This line reveals the format of the following data. First on that line, `HZ` is written to indicate that the first column of the data, which is always the frequency, is given in Hertz. Thereafter indicates `S` that the data is indeed S-parameter data. The following `RI` tells us that the S-parameters are given in "$\mathrm{Re}(S)$ $\mathrm{Im}(S)$" format. Other alternatives are `MA` and `DB`, corresponding to "$|S|$ $\angle S$" and "$20log_{10}(|S|)$ $\angle S$", respectively. Finally, `R 50` indicates that the S-parameters are given for a 50Ω system impedance. The remainder of the file consists of lines in the following format:

```
freq real(S11) imag(S11) real(S21) imag(S21) real(S12)...
```

where in this case the `RI` format has been assumed.

4.2 read_touchstone

The `read_touchstone` function allows a straightforward way of importing TouchStone files into Muwave:

[3]TouchStone is a registered trademark of Agilent Corporation. A complete specification of the file format can be found in either www.eda.org/pub/ibis/connector/touchstone_spec11.pdf or the Agilent ADS manual.

```
>> msp = read_touchstone(measSp,'muwave/test/d0509/d0509_20')
Measurement info
    Date : 2002-10-25, 08:50
    Origin : muwave/test/d0509/d0509_20
    Operator :
    Info :
Measurement state
 MeasType: SP
 Vgs: -0.1
 Vds: 1.5
 Igs: -2.784e-007
 Ids: 0.019745
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

The variable `msp` is now an instance of an `measSp` object, as illustrated in Fig. 1. The parameters displayed indicate that `read_touchstone` has parsed the header in the TouchStone file and put the information in the corresponding sub-objects. The following sections will describe how to access and manipulate these parameters and properties.

4.3 write_touchstone

Naturally, it is also possible to store a `measSp` object as a TouchStone file. The syntax is similar as for `read_touchstone`:

```
>> write_touchstone(msp,'c:\users\nn\temp\datatest.s2p');
```

5 Basic access and manipulation of properties

Once the measurement data has been read into an `meassp` object, there are several methods implemented to access and manipulate their contents.

First, it will be described how to access and manipulate the `measstate` and `measmnt` object properties. The S-parameter data, which is stored in an `xparam` object is treated separately in section 6.

5.1 set and get

Generally, the two functions `set` and `get` are used to manipulate and access the properties of an `meassp` object. It is important to note that these functions can be applied directly to the `meassp` object, although the property that is being accessed is in fact member of the `measmnt` or `measstate` sub-classes. A typical usage of them is shown below:

```
>> msp = read_touchstone(meassp,'muwave/test/d0509/d0509_20');get(msp,'Date')
ans =
2002-10-25, 08:50

>> get(msp,'Ids')
ans =
    0.0197

>> get(msp,'Freq')
ans =
1.0e+010 *
    0.1000
    0.1490
    ...

>> msp = set(msp,'Operator','Tintin','Igs',1,'Date',datestr(now))
Measurement info
    Date : 14-Apr-2009 10:00:04
    Origin : muwave/test/d0509/d0509_20
    Operator : Tintin
    Info :
Measurement state
 MeasType: SP
 Vgs: -0.1
 Vds: 1.5
 Igs: 1
 Ids: 0.019745
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

5.2 The dot-operator

A more convenient way of accessing and manipulating `measp` data than using `set` and `get` is to use the "dot"- operator. The following expressions are equivalent and illustrate the principle of operation:

```
>> msp = set(msp,'Operator','Phyllis');
>> msp.Operator = 'Phyllis';

>> f = get(msp,'Freq');
>> f = msp.Freq;
```

The dot-assignments above automatically call the `@measp/subsasgn.m` function, which overloads the built-in `subsasgn` function that is used to handle the dot-operator for other input types. See `>> help subsasgn` for further details. Similar to `get`, `subsref` plays the same role when the dot-operator is used to access a parameter value.

5.3 Adding a property

It is sometimes required to add a property that was not defined in the TouchStone file. For example, we might want to add a `VNA` property that we can use to identify which VNA was used during the measurements. The procedure to add a `measmnt`-property is easy:

```
>> msp = addprop(msp,'VNA','HP 8510C')
Measurement info
    Date : 2002-10-25, 08:50
    Origin : matlab_milou\test\d0509\d0509_20
    Operator :
    Info :
    VNA : HP 8510C
    ...
```

To add a `measstate` property, the procedure is a bit different. First, the `measstate` object must be extracted from the `measp` object. The new property can then be added to the `measstate` object. This modified `measstate` object should then be put back into the original `measp` object. It is maybe more easy to understand in code:

```
>> mstate = get(msp,'measstate');
>> mstate = addprop(mstate,'Temp',75);
>> msp = set(msp,'measstate',mstate)
...
Measurement state
    MeasType:   SP
    Vgs:    -0.1
    Vds:    1.5
    Igs:    -2.784E-007
    Ids:    0.019745
    Temp:   75
...
```

5.4 Reduction and extension of frequency range

Another operation that is commonly encountered is to remove undesired frequencies, or to reduce the number of frequency points. Two methods can be used.

The first case applies when the frequencies remain the same, but some points should be removed. The `()` indexing method can then be applied. The following example shows an example of how to carry out this:

```
>> f = msp.freq;
>> freq_index = find(f>1e9 & f<10e9);
>> msp_reduced = msp(freq_index)
...
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 18
  freq: 1.49E+009 -> 9.82E+009
```

If the desired frequencies differ from the ones during measurement, an interpolation method can be used to find a new `meassp` object corresponding to the new frequency vector. The `meassp` function `interp` is designed for this purpose:

```
>> new_freq = linspace(1e9,1e10,51);
>> msp_new = interp(msp,new_freq)
...
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 51
  freq: 1E+009 -> 1E+010
```

6 X-parameter data access and manipulation

The most valuable feature of Muwave is for most users the possibility to handle and manipulate S/Y/Z/T[4] in a convenient way. This section gives an overview of the variety of possibilities offered.

6.1 Direct parameter access

Any of the X-parameters can be directly accessed from the `meassp` object by using the "dot-operator" notation or a get command:

```
>> msp.Z11
ans =
1.0e+002 *
1.0467 - 5.6792i
0.8499 - 3.9010i
...

>> 20*log10(abs(get(msp,'S21')))
ans =
11.9806
12.0251
11.9318
...
```

By this it was also illustrated how Muwave automatically converts from the S-parameters in `msp` into the desired type, as in `msp.Z11`. Note that these functions all return a row vector that can be involved in any complex mathematical vector manipulation that MATLAB supports.

6.2 Parameter assignment

It is also possible to assign new values for a particular X-parameter data using a simple assignment:

```
>> msp.S11 = 0.1*msp.S11;
```

If the parameter type being assigned differs from the type of the `meassp` object it is operating on, a conversion takes place to match the type of parameter that is being assigned:

```
>> msp
...
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
...
>> msp.Z11 = zeros(length(msp.freq),1);
>> msp
```

[4]Hereafter "X-parameter" is used to refer either of the S/Y/Z/T-parameters.

```
...
xparam-object
  type: Z
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

6.3 Matrix operations

There are three levels one can choose from when doing matrix operations on one or several X-parameter objects. Which to choose depends on the complexity of the operations involved.

6.3.1 xparam

On the highest level it is possible to operate directly with the `xparam`-objects. The `xparam`-object is extracted from the `meassp` object using:

```
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

where, again, the dot-operator has been applied to the `meassp` object.

The `xparam` objects contain information about i.e. the type of data (S/Z/Y/T). Muwave therefore automatically converts between these types so that they match i.e. when adding two objects.

```
>> Zp = msp.Z;
>> Yp = msp.Y;
>> Zp_sum = Zp + Yp
Warning: XPARAM.PLUS: Arguments not of same type. Conversion performed
> In xparam.plus at 25
xparam-object
  type: Z
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

Although only shown for addition here, a lot of other algebraic manipulations have been implemented for the `xparam` class. However, be careful that the conversions sometimes taking place when operating on `xparam` objects of different type may give undesired results. Please check out the online help for details on how each operation works.

It often turns out to be more reliable to operate on the data directly by accessing the underlying `arraymatrix` object.

6.3.2 arraymatrix

The `arraymatrix` is accessed from the `xparam` object:

```
>> Sp = msp.S;
>> Sp_array = get(Sp,'arraymatrix')
arraymatrix-object
    dimension:  2 x 2
    elements:   101
First matrix element:
0.9921 - 0.0697i   0.0009 + 0.0037i
-3.9645 + 0.2474i   0.8495 - 0.0128i
```

The `arraymatrix` can also be directly accessed from the `meassp` object by

```
>> Sp_array = get(msp,'arraymatrix');
```

Individual parameters of the `arraymatrix` object are accessed and assigned using parentheses `()`,

```
>> Sp_array2 = Sp_array;
>> Sp_array2(1,1) = Sp_array2(2,2)
arraymatrix-object
    dimension:  2 x 2
    elements:   101
First matrix element:
0.8495 - 0.0128i   0.0009 + 0.0037i
-3.9645 + 0.2474i   0.8495 - 0.0128i
```

Matrix operations are carried out in a similar straightforward fashion,

```
>> Sp_array3 = Sp_array2*Sp_array
arraymatrix-object
    dimension:  2 x 2
    elements:   101
First matrix element:
0.8376 - 0.0866i   0.0016 + 0.0063i
-7.2809 + 0.7830i   0.7172 - 0.0365i
```

where the matrix multiplication of `Sp_array2` and `Sp_array` is carried out independently for each frequency point. A variety of other algebraic operations are implemented for the `arraymatrix` class.

Finally, it is usually to put the manipulated `arraymatrix` object back into the `meassp` context:

```
>> Sp = xparam(Sp_array3,'S',50);
>> msp_new = msp;
>> msp_new.data = Sp;
```

6.3.3 3d-matrix

In rare cases, the functions implemented for the `arraymatrix` class are not sufficient to solve a particular problem. The way out is then to access the X-parameter data in its raw form, i.e. in the way it is *actually* stored in MATLAB .

The data is stored as a 3-dimensional matrix, where the first two dimensions correspond to the row and columns of the X-parameter matrix at each frequency. The third dimension is the frequency. This means that, e.g. a two-port S-parameter measurement with 101 frequency points will be stored as a 2x2x101-sized complex vector.

This raw data matrix can be accessed directly from the `xparam`, `arraymatrix`, or the `meassp` objects by getting the `mtrx` property:

```
>> Sp = msp.S;
>> raw_data = get(Sp,'mtrx')
raw_data(:,:,1) =
0.9921 - 0.0697i   0.0009 + 0.0037i
-3.9645 + 0.2474i   0.8495 - 0.0128i

raw_data(:,:,2) =
0.9887 - 0.1027i   0.0011 + 0.0053i
-3.9762 + 0.3613i   0.8527 - 0.0303i
...
...
raw_data(:,:,101) =
-0.5639 - 0.5608i   0.0775 + 0.0227i
1.1758 + 1.5074i   0.2195 - 0.7032i
```

After manipulation, it can be used to create an `arraymatrix` object,

```
>> new_array = arraymatrix(raw_data);
```

which, in turn can be converted back into `xparam` and `meassp` objects as was described above.

7 Model representation using netlists

So far, the description has been focused on handling measured S-parameter data originating from TouchStone files etc. However, in many applications it is also desired to handle the S-parameters of an ideal electrical circuit diagram. This is typically the case in modelling applications, where the parameters of a model are determined by minimizing the difference between the S-parameters generated by the model and the measured ones.

7.1 Netlist format and importing

The circuit shown in Fig. 2 will hereafter serve as an example.

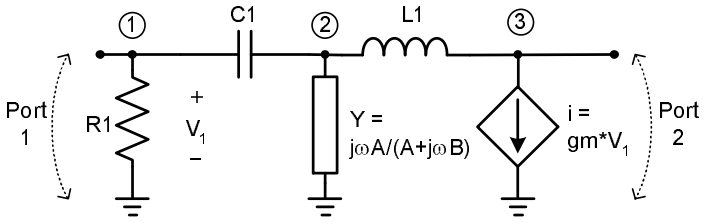

Figure 2: Example circuit.

Electrical circuits are entered into Milou by means of a *netlist* file, similar to the ones used in Spice. The expressions shown in Table 1 are recognized by the Muwave netlist parser:

Table 1: Valid netlist expressions.

Component	Netlist expression
Resistance	`R value`[a] `node1 node2`
Conductance	`G value node1 node2`
Capacitance	`C value node1 node2`
Inductance	`L value node1 node2`
General admittance	`X expression`[b] `node1 node2`
Transconductance	`VCCS value node1+ node2+ node1- node2-`
Transcond. with τ	`VCCSD value node1+ node2+ node1- node2-`
Gyrator	`GY value node1+ node2+ node1- node2-`
General transcond.	`X2 expression node1+ node2+ node1- node2-`
Port	`P portname node1 node 2`
Comment	`% Comment text`

[a] A *value* can be either a numeric value or a parameter name.

[b] An *expression* can include an arbitrary algebraic expression. s should be used instead of $j\omega$. Other names that are not recognized functions will be considered user parameters.

The netlist corresponding to the circuit in Fig. 2 becomes (`circuit.net`):

```
% Example circuit
R R1 1 0
C C1 1 2
X s*A/(A+s*B) 2 0
L L1 2 3
```

```
VCCS gm 1 3 0 0
P P1 1 0
P P2 3 0
```

The netlist can now be imported into Milou by using the `read_netlist` function,

```
>> model = read_netlist(mna,'test/circuit.net')
Admittance matrix:
    '+1/(R1)+s*C1'    '-s*C1'                 []
    '-s*C1'           [1x26 char]    '-1/(s*L1)'
    '+gm'             '-1/(s*L1)'    '+1/(s*L1)'

Parameters:
    'R1'    'C1'    'A'    'B'    'L1'    'gm'

Parameter types:
    'R'    'C'    'X'    'X'    'L'    'VCCS'

Frequencies
```

The output variable `model` is now an `mna` object. mna stands for *modified nodal admittance*, which is the type of matrix normally used to represent circuits in simulators. Muwave currently uses the ordinary nodal admittance matrix for representing circuits[5]. It is, by the way, shown in the first part of the displayed `mna` object information displayed above.

The next thing displayed is the list of parameters. These are the unique parameter names found in the netlist. The order in which the parameters appear is important when, at a later stage, their values should be assigned. The parameter names can also be accessed by the `params` command,

```
>> params(model)
ans =
    'R1'    'C1'    'A'    'B'    'L1'    'gm'
```

Next in the results displayed is the types associated with the parameters. This is not applicable for parameters defined in `X` or `X2` elements.

Finally, there is a list of partial derivatives that are used for calculating sensitivities. This feature is, however, not treated in this manual text.

7.2 Evaluation of the model X-parameters

Before it is possible to assign numbers to the model parameter and evaluate its X-parameters, one needs to specify the frequencies used for consequent calculations. The reason that the frequencies are entered separately is to speed up the X-parameter evaluations later. The frequencies are assigned to the `mna`-object using the `freq` command,

```
>> model = freq(model,msp.freq);
```

where the same frequencies as in the previous measurement examples have been used.

It is now possible to evaluate the model for any set of parameters using a call to either the `calc` or the `reduce` functions. The model parameter values should be arranged in a vector with the order being the same as described above.

[5]The modified nodal admittance matrix uses a different representation of components that may present infinite conductance, e.g. inductances at low frequency, and thus provides better numerical stability.

```
>> R1 = 100;C1 = 1e-12;A = 0.7e-12;B = 2e-12;
>> L1 = 1e-9; gm = 1e-3;
>> model_params = [R1,C1,A,B,L1,gm];
```

Once the parameters have been defined it is possible to calculate either the full N-by-N matrix, where N is the number of nodes, or a reduced matrix describing the circuit as observed at the ports that were defined in the netlist.

The first case is evaluated by using the `calc` function,

```
>> calc(model,model_params)
xparam-object
  type: Y
  reference: 50
  ports: 3
  elements: 101
  freq: 1E+009 -> 5E+010
```

The resulting object is, for our three-node example a 3-by-3 Y-type `xparam` object.

The function `reduce` is used in a similar fashion to calculate the port-reduced Z-parameters,

```
>> xp_model = reduce(model,model_params)
xparam-object
  type: Z
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
```

Note that the resulting object is a 2-by-2 Z-type `xparam` object, which can be handled in exactly the same manner as the measurements were used before.

8 Sweeps of measurements

Muwave has not only functionality to handle single measurements, using the `meassp` class, but also a series of measurements. Typically, this is a series of measurements obtained at different bias points. The class used to handle these swept measurements is called `meassweep`.

Muwave can create `meassweep` objects from either a directory with multiple TouchStone files or from an MDIF file[6].

8.1 Importing multiple Touchstone files

Multiple TouchStone files are imported into an `meassweep` object using the function `read_milousweep`. The following example shows a typical call with this function:

```
>> sweep = read_milousweep(meassweep,...
Measurement info
    Date : 14-Apr-2009 10:54:07
    Origin : muwave\test\d0509_all\d0506_*
    Operator :
    Info :
Sweep info
 Vgs,min: -0.5
 Vgs,max: 0.2
 Vds,min: 0.5
 Vds,max: 1.5
 Igs,min: -7.94141e-007
 Igs,max: 2.05474e-006
 Ids,min: 0.00343814
 Ids,max: 0.0297255
 Number of measurements: 24
```

Please consult `>> help read_milousweep` to find the details on how to call this function.

8.2 Importing MDIF-files

MDIF files allow a convenient way of transferring multi dimensional data to and from Agilent ADS. The structure of the MDIF files are described in the Agilent ADS manual and not repeated here.

The function `read_mdif` can be used to import MDIF data into a `meassweep` object in Milou. It should be stressed that, at the time of writing, this function is still under development and may not operate as expected!

8.3 Importing Maury ATS swept bias S-parameters

The function `read_s2b` allows Milou to import swept bias S-parameter files created by the Maury Microwave ATS software. Please consult the on-line help for more information about this function.

[6] See Agilent ADS documentation for details about the MDIF file format.

8.4 Accessing and indexing meassweep objects

A `meassweep` object can thought of as a vector of `meassp` objects. An individual `meassweep` object, or a subset of the `meassweep` can be accessed using the parenthesis `()` operator,

```
>> sweep(1)
Measurement info
    Date : 2002-10-24, 19:23
    Origin : muwave\test\d0509_all\d0506_0
    Operator :
    Info :
Measurement state
 MeasType: SP
 Vgs: -0.5
 Vds: 0.5
 Igs: -6.43067e-007
 Ids: 0.00343814
xparam-object
  type: S
  reference: 50
  ports: 2
  elements: 101
  freq: 1E+009 -> 5E+010
>> sweep(1:10)
Measurement info
    Date : 14-Apr-2009 10:54:07
    Origin : muwave\test\d0509_all\d0506_*
    Operator :
    Info :
Sweep info
 Vgs,min: -0.5
 Vgs,max: 0.2
 Vds,min: 0.5
 Vds,max: 1
 Igs,min: -6.97489e-007
 Igs,max: 7.27892e-007
 Ids,min: 0.00343814
 Ids,max: 0.0233942
 Number of measurements: 10
```

8.5 The dot-operator

An individual parameter of the sweep objects can be accessed using the dot-operator. The following example shows how to access the `Vgs`-parameter and also how this can be used to use reduce the sweep-object to contain only bias points with $\texttt{Vgs} = 0$.

```
>> sweep.Vgs
ans =
Columns 1 through 7
-0.5000   -0.4000   -0.3000   -0.2000   -0.1000   -0.0000    0.1000
Columns 8 through 14
    0.2000   -0.5000   -0.4000   -0.3000   -0.2000   -0.1000   -0.0000
...
```

```
>> bias_index = find(abs(sweep.Vgs)<0.01);
>> sweep(bias_index)
Measurement info
    Date : 14-Apr-2009 10:54:07
    Origin : muwave\test\d0509_all\d0506_*
    Operator :
    Info :
Sweep info
 Vgs: -2.77556e-017
 Vds,min: 0.5
 Vds,max: 1.5
 Igs,min: -1.09436e-007
 Igs,max: -4.34143e-008
 Ids,min: 0.0212195
 Ids,max: 0.0257597
 Number of measurements: 3
```

8.6 Adding measurements

Single measurements can be added to the `meassweep` object with the `add` function,

```
>> msp = read_touchstone(measp,'muwave\test\d0509\d0509_20');
>> sweep_total = add(sweep,msp);
>> length(sweep_total)-length(sweep)
ans =
    1
```

The same function, `add`, can also be used to merge two `meassweep` objects.

8.7 Writing swept data

For the moment, no functions have been implemented to write a `meassweep` object to multiple TouchStone files or into an MDIF-file.

9 Plotting functions

Finally, some useful functions for plotting `meassp` and `measswеep` data are described.

9.1 Combined Smith/polar diagram

A combined Smith/polar diagram for depicting two-port S-parameters is easily produced with the `smithplot` function. It takes a `meassp` object as input and generates a combined Smith/polar plot for all four S-parameters (See Fig. 3):

```
>> smithplot(msp);
```

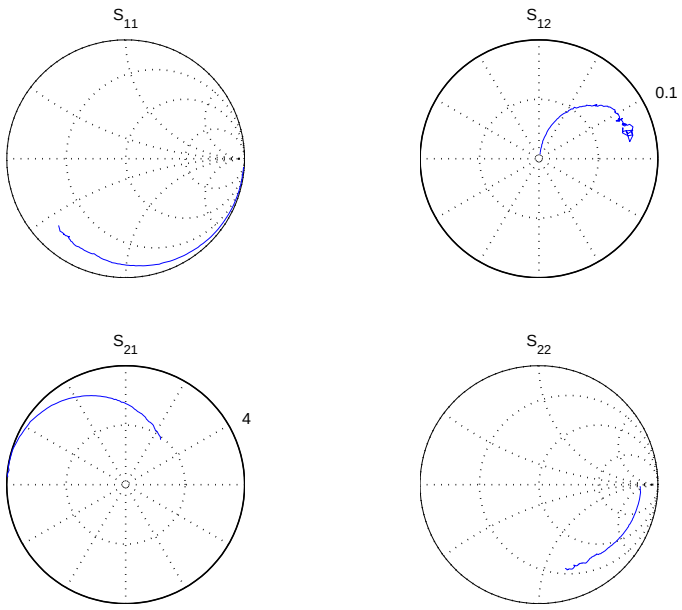

Figure 3: Combined Smith/polar diagram produced with smithplot.

9.2 Single parameter plot

Single parameters may be plotted using the `paramplot` command. It is particularly useful for producing single parameter Smith plots. The syntax for this is given below.

```
>> paramplot(msp,'S11','smith',gca);
```

Other plotting methods are available as described in the help at
`>> help paramplot`.

9.3 Swept data

A modified `plot` function has been implemented to plot parameters of `measswеep` object against each other. This is useful for producing I/V plots from the biasing information contained in the measurement files. The syntax is exemplified by,

```
>> plot(sweep,'Vgs','Ids')
```

The plot produced is shown in Fig. 4.

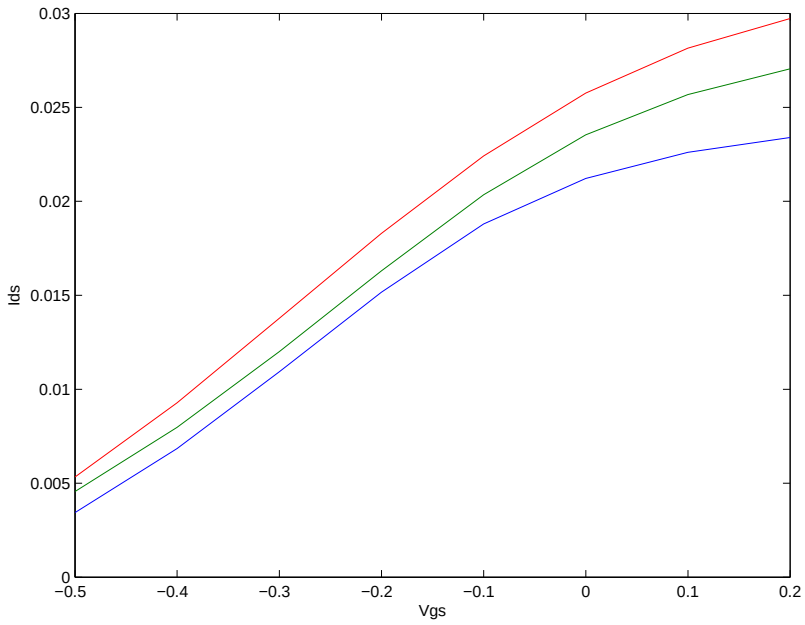

Figure 4: Ids-Vgs characteristics produced with the meassweep-plot function.