UCSB ECEN 194BB/594BB

Instructor: Prof. Peng Li

Matlab Circuit Parser User's Manual

The purpose of our Matlab parser is to parse in the SPICE like input circuit deck, and store information in Matlab arrays. All required files for the parser are provided in **matlab_parser.zip**.

1. The circuit description

1.1 Circuit elements

The syntax for the circuit elements is SPICE-like for all elements except MOSFET's, for which a simplified description is used.

Resistors:

Rxxx <N1> <N2> <VALUE>

Yxxx <N1> <N2> <VALUE>

Capacitor:

Cxxx <N1> <N2> <VALUE> [<INIT_VOLTAGE>]

Inductors:

Lxxx <N1> <N2> <VALUE> [<INIT_CURRENT>]

Kxxx <L1> <L2> <VALUE> {*mutual inductor*}

(Note: the value of a K element is NOT the mutual inductance. It's defined in this way: suppose a K element represents the mutual inductance L12 between two inductors with self inductance L1 and L2, then the value of this K element is $K = \frac{L_{12}}{\sqrt{L_1 L_2}}$. This is consistent with the convention for mutual inductor elements in HSPICE.)

Susceptors:

Sxxx <N1> <N2> <VALUE> [<INIT_CURRENT>]

Wxxx <S1> <S2> <VALUE> {*mutual susceptor*}

(Note: the W elements follow the similar convention as the K elements, which is: suppose a W element represents the mutual susceptance S12 between two susceptors with self susceptance S1 and S2, then the value of this W element is $W = \frac{S_{12}}{\sqrt{S_1 S_2}}$.)

Independent sources:

Vxxx <N1> <N2> <SOURCE DESCRIPTION>

Ixxx <N1> <N2> <SOURCE DESCRIPTION>

Our parser accept three types of independent sources (V_TYPE_ or I_TYPE_):

1. DC source:

   DC <VALUE>

2. AC source:

   AC <MAGNITUDE> <PHASE>

3. PWL (piecewise linear) source:

   PWL <VOLTAGE> <TIME> <VOLTAGE> <TIME> <VOLTAGE>...

(Note: There is a small difference between the syntax of our PWL source and SPICE PWL source. Our PWL source start at time = 0 by default, therefore, the first value after PWL keyword is the voltage at time = 0 instead of starting point time as in SPICE. The number of time-value pairs except the initial one is specified in **elem**(V_POINTS_) or **elem**(I_POINTS_), where **elem** is the row in **LINELEM** corresponding to a PWL voltage or current source.)

Controlled sources:

Exxx <N1> <N2> <CN1> <CN2> <VALUE> {*Voltage Controlled Voltage Source*}

Gxxx <N1> <N2> <CN1> <CN2> <VALUE> {*Voltage Controlled Current Source*}

Fxxx <N1> <N2> <ELEM_NAME> <VALUE> {*Current Controlled Current Source*}

Hxxx <N1> <N2> <ELEM_NAME> <VALUE> {*Current Controlled Voltage Source*}

The MOSFET's are described by the 'M' card:

Mxxx <ND> <NG> <NS> <MOS_TYPE> <WIDTH> <LENGTH> <MODEL_ID>

Here, ND, NG and NS are the Drain, Gate and Source nodes respectively. MOS_TYPE is ‘n’ or ‘N’ for NMOS and ‘p’ or ‘P’ for PMOS. The MODEL_ID is an integer (>= 1) which indicates the model card to be used for this MOSFET. The model cards begin with a ‘.MODEL’ (as in SPICE) and have the following syntax:

.MODEL <MODEL_ID> VT <V_T> MU <μ> LAMBDA <λ> COX <C_{OX}> CJ0 <C_{J0}>

V_T is the threshold voltage, μ is the mobility, λ is the channel-width modulator, Cox is the oxide capacitance, and C_{J0} is the junction capacitance.

1.2 Control cards

The syntax of the analysis cards are similar to SPICE syntax except for the ‘.TRAN’ card which is modified to include model order reduction analysis.

For dc analysis use

.DC

For ac analysis use

.AC <points per decade> <starting frequency> <final frequency>

For transient analysis use

.TRAN <algorithm> <time step> <stop time> <AWE order>

Here, the *algorithm* specifies the algorithm to use for transient analysis. It should be one of the following:

For numerical integration:

FE -- Forward Euler

BE -- Backward Euler

TR -- Trapezoidal

For model order reduction:

AWE -- Use AWE algorithm

PRIMA -- Use PRIMA algorithm

When using model order reduction algorithms, the *AWEorder* specifies the order for the reduced model.

The netlist PRINT directives do not follow the SPICE convention and are greatly simplified. There are only three types, all beginning with the ‘.’, and they are listed below.

.PRINTNV <NODE1> <NODE2> .... {*Node voltages*}

.PRINTBV	<ELEM1> <ELEM2>	{*Branch voltages*}
.PRINTBI	<ELEM1> <ELEM2>	{*Branch currents*}

For MOSFET's, the current is the Drain-Source current. Note that the PRINT commands have to appear at the end of the circuit description.

The netlist PLOT directives are quite similar to the PRINT directives and are listed below.

.PLOTNV	<NODE1> <NODE2>	{*Node voltages*}
.PLOTBV	<ELEM1> <ELEM2>	{*Branch voltages*}
.PLOTBI	<ELEM1> <ELEM2>	{*Branch currents*}

2. Parsing the circuit

You need either to copy the provided parser files to your matlab working directory or add the directory where you put the parser to your matlab path. You can add the path by executing the following at the Unix prompt before invoking Matlab:

setenv MATLABPATH *Directory_of_the_Parser*:${MATLABPATH}

Alternately, you can give the command: *path(path, **'Directory_of_the_Parser')*** or *addpath **Directory_of_the_Parser*** within the Matlab environment.

Note: To get help about any function (including the ones that we provide you), type "help <function_name>" within Matlab. To see what the function actually does, type "type <function_name>".

2.1 Initializing constants for the parser

Before using the parser, it is convenient to predefine some constants. From within Matlab, type:

>> parser_init

Short Cuts for Circuit Parsing Loaded

This function will set some constants that you will use later to manipulate the data. You can also find the **parser_init.m** file as part of the provided **matlab_parser.zip** file. For example, the value of R_ (refers to the resistor type) is defined to be abs('R') which is 82. There are also other variables defined within this function for easy access to the entries of an element card. The parser will parse the input file and store the results in several matrics, and each element will be

represented by a vector (let us call this vector ‘elem’). For example, a resister can appear as follows:

82 20 1 2

which corresponds to the following definition in the input file:

Rxxx 1 2 20

The type of the element can be checked by accessing the ‘TYPE_’ entry of the ‘elem’ vector and comparing it with that for the resistor, (R_), so the comparison would look like

if (elem(TYPE_)==R_)

The node numbers and values for the elements can be accessed in a similar fashion: elem(R_N1_) gives the first node for that resistor, elem(R_N2_) gives the second node, and elem(R_VALUE_) gives the value of the resistance itself.

Parameters for other elements can be accessed in a similar fashion. See the file ***parser_init.m*** for a complete listing.

Note:
1. DO NOT access the individual entries of a data card using the corresponding indices. Always use the variables described above so that the indices can be changed later without affecting any of your code.
2. Let us suppose the ‘elem’ vector corresponds to a capacitor. When the initial value of this capacitor is not specified, elem(C_IC_) is set to be NaN. The same rule applies to inductors and susceptors.

2.2 Using the parser

A Matlab script function ‘parser.m’ has been written to handle the circuit parsing. You never need look at the parsing function (or any other functions which the parser may call). The parser can be invoked by calling *parser()* with the appropriate arguments:

[LINELEM,NLNELEM,INFO,NODES,LNAME,NNAME,PRINTNV,PRINTBV,PRINTBI,PLOTNV,PLOTBV,PLOTBI] = parser(CIRC)

The input argument **CIRC** is the name of a circuit file. Each of the output arguments corresponds to some part of the information contained in the circuit netlist file.

LINELEM = Array of Linear elements (each row corresponds to one element)

NLNELEM = Nonlinear elements (in our case, it will be the list of MOSFET’s)

INFO = Auxiliary information on the simulation

NODES = Actual numbers (“names”) of the nodes as given in the circuit file

LNAME = Names of the linear elements as given in the circuit file

NNAME = Names of the nonlinear elements as given in the circuit file

PRINTNV = Node voltages to be printed

PRINTBV = Branch voltages to be printed

PRINTBI = Branch currents to be printed

PLOTNV = Node voltages to be plotted

PLOTBV = Branch voltages to be plotted

PLOTBI = Branch currents to be plotted

The **LINELEM** array contains the data for all the elements except the MOSFET’s. The **NLNELEM** array contains all the relevant information about MOSFET’s. Each row of this array corresponds to a MOSFET and contains all the information about the MOSFET. The following lines describe the method for accessing the various parameters of that MOSFET:

elem(M_TYPE_) : Type of the MOSFET (either NMOS_ or PMOS_)

elem(M_ND_) : Drain node

elem(M_NG_) : Gate node

elem(M_NS_) : Source node

elem(M_W_) : Width of the MOSFET

elem(M_L_) : Channel length of the MOSFET

elem(M_LAMBDA_) : Channel length modulation parameter

elem(M_VT_) : Threshold voltage

elem(M_MU_) : Mobility of the carriers.

elem(M_COX_) : Gate oxide capacitance per unit area.

elem(M_CJ0_) : The drain-ground and source-ground capacitances.

The array **NODES** gives the mapping between the internal nodes (“names”) of the circuit netlist and the numbers that the parser assigns them to:

NODES(int) = ext

The arrays **LINNAME** and **NLNNAME** contain the actual names of the elements in the **LINELEM** and **NLNELEM** arrays. This information is required to print out branch currents or voltages because only the element names are provided in the PRINT cards.

The **INFO** matrix gives the information on the analysis. Type of simulation is determined by checking METHOD_ entry of **INFO**. The list of analysis types are given in Table 1.

Table 1: Types of Simulation Analysis

Analysis Type	INFO(METHOD_)
DC Analysis	DC_
AC Analysis	AC_
FE (Forward Euler) transient analysis	FE_
BE (Backward Euler) transient analysis	BE_
TR (Trapezoidal) transient analysis	TR_
AWE, Transient or frequency analysis by using AWE	AWE_
PRIMA, Transient or frequency analysis by using PRIMA	PRIMA_

If transient analysis will be performed, one can obtain the timestep and timestop variables with the values in TSTEP_ and TSTOP_ indices of **INFO** array. If transient analysis will be performed by model order reduction, the ORDER_ entry of **INFO** array will give you the order of the reduced-order model. For ac analysis, the AC_PPD_, AC_FSTART_, AC_FSTOP_ entries will hold the number of point per decade, starting frequency and stop frequency respectively.

Since we have two kinds of representations for the magnetic components, the LSTYPE_ entry of **INFO** matrix tells you whether the circuit is a RLMC circuit or a RSWC circuit. If only L's and K's (inductors and mutual inductors) appear in the input file, **INFO**(LSTYPE_) is L_CIR_. On the other hand, if only S's and W's (susceptors and mutual susceptors) appear in the input file, INFO(LSTYPE_) is S_CIR_.

The **PRINTNV**, **PRINTBV** and **PRINTBI** arrays contain information about which node voltages or branch voltages or branch currents need to be printed. **PRINTNV** gives the internal nodes that were referenced in a .PRINTNV control card. **PRINTBV** records the indices of the elements that were referenced .PRINTBV control card. For each row of **PRINTBV**, a branch voltage will be a simulation output. One can determine whether the branch element belongs to

LINELEM or **NLNELEM** by checking the second entry in that row. An entry 1 in that second position represents a linear branch element and 2 represents a nonlinear element. The first entry in that row records the index of the element in its proper list. **PRINTBI** is very similar to **PRINTBV**, it holds the information of branch currents requested in the input deck. It is referenced with .PRINTBI control card. The **PLOTNV**, **PLOTBV** and **PLOTBI** arrays follow the conventions as their PRINT counterparts.

2.3 An Example

The example is an inverter with a complex load: (filename: test_inv)

VDD 103 0 DC 3

Vin 101 0 PWL 0 5.0e-10 3.0 1.0e-9 3.0

Rin 101 102 10

M1 104 102 103 p 30e-6 0.35e-6 1

M2 104 102 0 n 10e-6 0.35e-6 2

C1 104 0 0.1e-12 0

R2 104 105 25

L1 105 106 0.1e-9 0

C2 106 0 0.5e-12 0

.MODEL 1 VT -0.75 MU 5e-2 COX 0.3e-4 LAMBDA 0.05 CJ0 4.0e-14

.MODEL 2 VT 0.83 MU 1.5e-1 COX 0.3e-4 LAMBDA 0.05 CJ0 4.0e-14

.TRAN TR 1.0e-11 2.0e-9

.PRINTNV 102 104 106

.PRINTBI C1 M2

.PLOTNV 102 104 106

.PLOTBV L1

.end

After calling parser routines.

parser_init;

[LINELEM, NLNELEM, INFO, NODES, LINNAME, NLNNAME, PRINTNV, PRINTBV, PRINTBI, PLOTNV, PLOTBV, PLOTBI] = parser(‘test_inv’);

We get a lot of variables in the workspace (You can check the list by typing who). Our main data is already stored in the matrices **LINELEM**, **NLNELEM**, **INFO**, **NODES**, **LINNAME**, **NLNNAME**, **PRINTN**, **PRINTB**, **PRINTI, PLOTN, PLOTB, PLOTI**.

LINELEM =

86.0000	3.0000	1.0000	-1.0000	0	0	0	0	0	0
86.0000	0	2.0000	-1.0000	2.0000	2.0000	0.0000	3.0000	0.0000	3.0000
82.0000	10.0000	2.0000	3.0000	0	0	0	0	0	0
67.0000	0.0000	4.0000	-1.0000	0	0	0	0	0	0
82.0000	25.0000	4.0000	5.0000	0	0	0	0	0	0
76.0000	0.0000	5.0000	6.0000	0	0	0	0	0	0
67.0000	0.0000	6.0000	-1.0000	0	0	0	0	0	0

NLNELEM =

Columns 1 through 6

7.7000e+01	0	4.0000e+00	3.0000e+00	1.0000e+00	3.0000e-05
7.7000e+01	1.0000e+00	4.0000e+00	3.0000e+00	-1.0000e+00	1.0000e-05

Columns 7 through 12

3.5000e-07	-7.5000e-01	5.0000e-02	3.0000e-05	5.0000e-02	4.0000e-14
3.5000e-07	8.3000e-01	1.5000e-01	3.0000e-05	5.0000e-02	4.0000e-14

INFO’ =

3.0000e+00	1.0000e-11	2.0000e-09	0	0	0	0	1.0000e+00	0	0

NODES’ =

103	101	102	104	105	106

LINNAME =

86	68	68	32	32	32
86	73	78	32	32	32
82	73	78	32	32	32

67 49 32 32 32 32

82 50 32 32 32 32

76 49 32 32 32 32

67 50 32 32 32 32

NLNNAME =

77 49 32 32 32 32

77 50 32 32 32 32

PRINTNV ‘=

6 4 3

PRINTBV =

[]

RINTBI =

2 2

4 1

PLOTNV’ =

6 4 3

PLOTBV =

6 1

PLOTBI =

[]

(Note: The format of a row in LINELEM corresponding to a PWL voltage source is the following: ElemType Init_Value Node1 Node2 V_TYPE_ V_POINTS_ time1 value1 time2 value2 ... timeN valueN. V_POINTS_ represents the number of time-value pairs

following it. The voltage value will be valueN for the simulation time beyond timeN. A similar rule applied to PWL current sources.)

3. Stamping values into the MNA matrix

To reduce the work-load for programming, you have been provided with two example routines for stamping conductances and ideal voltage sources into the MNA formulaion. These two routines are given in the appendix of this manual. The syntax and usage of the stamping functions is given below.

[new_M] = stamp_conductance(old_M,D)

Here, **new_M** and **old_M** represent the new and old MNA matrices. **D** is the data vector corresponding to the conductance. This data vector is just one row of the array **LINELEM**, which is output from the parser function.

The syntax for independent voltage sources is:

[new_M,new_I,new_row] = stamp_ind_vsource(old_M,old_I,D)

new_M, **old_M** and **D** are similar to that explained above. **new_I** and **old_I** are the current matrices or the right hand side of the MNA equations. *new_row* is the row number corresponding to this voltage source and is returned to the main function so that values corresponding to this voltage source can be accessed using this row number.

Note: Those functions are just some sample code. You can handle these elements in whatever way you like. Also, for other elements, you still need to write your functions to do stamping

4. Appendix -- samples of stamping functions (non-optimized)

```
function [new_M]=stamp_conductance(old_M,D);
%STAMP_CONDUCTANCE : Stamps conductances  into the MNA matrix
%
%               syntax : [new_M]=stamp_conductance(old_M,D)
%
%               new_M,old_M are self-explanatory
%               D is the data vector corresponding to the conductance or resistance.
```

```
global Y_N1_ Y_N2_ Y_ Y_VALUE_;
new_M=old_M;
n1 = D(Y_N1_);
n2 = D(Y_N2_);

if n1>length(new_M),  new_M(n1,n1)=0;end;
if n2>length(new_M),  new_M(n2,n2)=0;end;

value=D(Y_VALUE_);

if (n1>0) & (n2>0),
  new_M(n1,n1) = new_M(n1,n1) + value;
  new_M(n1,n2) = new_M(n1,n2) - value;
  new_M(n2,n1) = new_M(n2,n1) - value;
  new_M(n2,n2) = new_M(n2,n2) + value;
elseif (n1<0)
  new_M(n2,n2) = new_M(n2,n2) + value;
elseif (n2<0)
  new_M(n1,n1) = new_M(n1,n1) + value;
end
```

```
function [new_M,new_I,new_row] = stamp_ind_vsource(old_M,old_I,D);
%STAMP_IND_VSOURCE : stamps entries corresponding to an independent voltage source.
%
%               syntax :  [new_M,new_I,new_row] = stamp_ind_vsource(old_M,old_I,D)
%
%               new_M,old_M are the new and old MNA matrices
%               new_I,old_I,are the new and old current matrices (right hand side)
%               D is the data vector corresponding to the source
%               "new_row" is the row number corresponding to this new source
%               (This number has to be returned to the main function so that
%                the row corresponding to this voltage source can be accessed later.)

global V_N1_ V_N2_ V_ V_VALUE_
new_M=old_M;
new_I=old_I;
length_M=length(old_M);
n1 = D(V_N1_);
n2 = D(V_N2_);

if n1>length_M,  new_M(n1,n1)=0;end;
if n2>length_M,  new_M(n2,n2)=0;end;

if n1>0, new_M(length_M+1,n1)=1;new_M(n1,length_M+1)=1;end;
if n2>0, new_M(length_M+1,n2)=-1;new_M(n2,length_M+1)=-1;end;
new_I(length_M+1)=D(V_VALUE_);
```

```
new_row=length_M+1;
```
