“Cheat Sheet” for MAPP Hands-on Workshop

1. Simple linear resistor model:

```
>> model_starter();
```

Please enter the model's name (e.g., my_r_model):	myR
Please enter the number of terminals in the model (e.g., 2):	2
Hit Enter when ready to enter the names of your nodes:	↵
Name of the first terminal (e.g., a):	p
Name of the last terminal (reference terminal, e.g., ref):	n
The I/O names for this model are: vpn, ipn Among them, which one(s) can be expressed explicitly in your model equations?	ipn
Enter the names of the model's internal unknowns (e.g., unk1, unk2). If there are no internal unknowns, just hit enter:	↵

```
>> edit myR.m;
```

Changes to make to myR.m:

% Please enter the parameter(s) using the following template: % MOD = add_to_ee_model(MOD,'parms',{'parm1_name', default_val1}); % ^ ^ % CHANGE THESE % MOD = add_to_ee_model(MOD,'parms',{parm2_name', default_val2}); % ^ ^ % CHANGE THESE DELETE THIS LINE WHEN DONE ENTERING YOUR PARAMETERS	MOD = add_to_ee_model(MOD, 'parms', {'R', 1000.0});
ipn_fe = ... FILL THIS IN	ipn_fe = vpn/R;
ipn_qe = ... FILL THIS IN	ipn_qe = 0;

```
>> MOD = myR();
>> check_ModSpec(MOD);
>> MEO = model_exerciser(MOD);
>> MEO.display(MEO);
>> MEO.ipn_fe(-1:0.1:1, MEO);
>> MEO.plot('ipn_fe', -1:0.1:1, MEO);
>> MEO.plot('ipn_fe', -1:0.1:1, 'R', 1e3:0.5e3:10e3, MEO);
```

Optional: create myR2.m, add another parameter R2 to it, change model equation to ipn = vpn/R + vpn^3/R2,
then plot ipn_fe of the model.

2. Capacitor:

```
>> model_starter();
```

Please enter the model's name (e.g., my_r_model):	myC
Please enter the number of terminals in the model (e.g., 2):	2
Hit Enter when ready to enter the names of your nodes:	↵
Name of the first terminal (e.g., a):	p
Name of the last terminal (reference terminal, e.g., ref):	n
The I/O names for this model are: vpn, ipn Among them, which one(s) can be expressed explicitly in your model equations?	ipn
Enter the names of the model's internal unknowns (e.g., unk1, unk2). If there are no internal unknowns, just hit enter:	↵

```
>> edit myC.m
```

Changes to make to myC.m:

```
MOD = add_to_ee_model(MOD, 'parms', {'C', 1e-6});
ipn_fe = 0;
ipn_qe = C*vpn;
```

```
>> MOD = myC();
>> check_ModSpec(MOD);
>> MEO = model_exerciser(MOD);
>> MEO.display(MEO);
>> MEO.plot('ipn_qe', -1:0.1:1, MEO);
```

3. RC circuit:

```
>> edit myRC_ckt.m;

>> ckt = myRC_ckt();
>> DAE = MNA_EqnEngine(ckt);
>> dcop = dot_op(DAE);
>> dcop.print(dcop);
>> dcSol = dcop.getSolution(dcop);
>> uDC = dcop.getDCinputs(dcop);

>> acObj = dot_ac(DAE, dcSol, uDC, 1, 1e6, 10, 'DEC');
>> acObj.plot(acObj);

>> tstart = 0; tstep = 1e-5; tstop = 5e-3;
>> tranObj = dot_transient(DAE, [], tstart, tstep, tstop);
>> tranObj.plot(tranObj);
```

or simply run:

```
>> run_myRC_ckt;
```

Optional: use myR2.m in myRC_ckt, run transient simulation again, and see what happens.

4. Resistor models (V=I*R, 0=V-I*R):

>> model_starter();

Please enter the model's name (e.g., my_r_model):	myR_vpn
… …	
The I/O names for this model are: vpn, ipn Among them, which one(s) can be expressed explicitly in your model equations?	vpn

>> edit myR_vpn.m;

Changes to make to myR_vpn.m:

```
MOD = add_to_ee_model(MOD, 'parms', {'R', 1000.0});
vpn_fe = R * ipn;
vpn_qe = 0;
```

```
>> MOD = myR_vpn();
>> check_ModSpec(MOD);

>> MEO = model_exerciser(MOD);
>> MEO.display(MEO);
>> MEO.plot('vpn_fe', 1e-3*(-1:0.1:1), MEO);
```

Please enter the model's name (e.g., my_r_model):	myR_implicit
… …	
The I/O names for this model are: vpn, ipn Among them, which one(s) can be expressed explicitly in your model equations?	↵
… …	
The model has 1 implicit equation(s). Would you like to specify their names? Y/N [N]:	Y
Name of the first implicit equation (e.g., eqn_1):	Ohm_law

>> edit myR_implicit.m;

Changes to make to myR_implicit.m:

```
MOD = add_to_ee_model(MOD, 'parms', {'R', 1000.0});
% algebraic part of Ohm_law
out(1, 1) = vpn - R*ipn;
% d/dt part of Ohm_law
out(1, 1) = 0;
```

```
>> MOD = myR_vpn();
>> check_ModSpec(MOD);
```

The model has only an implicit equation. So instead of using model_exerciser, it will be tested in a circuit in the next step.

5. Resistive divider circuit (with three different resistor models):

```
>> ckt = Rdivider_ckt();
>> DAE = MNA_EqnEngine(ckt);
>> swp = dot_dcsweep(DAE, [], 'V1:::E', 0, 10, 50);
>> swp.plot(swp);
```

or simply run:

```
>> run_Rdivider_ckt;
```

6. Diode with capacitance:

Instead of using model_starter, you can copy myC.m to diodeC.m. Then add parameters and edit ipn_fe section:

```
MOD = add_to_ee_model(MOD, 'parms', {'Vt', 0.0026, 'Is', 1e-12}); % thermal voltage and saturation current
ipn_fe = Is*(exp(vpn/Vt) - 1);
```

Check the model, then use model_exerciser to plot 'ipn_fe' and 'ipn_qe'.

7. Diode-resistor circuit:

```
>> ckt = myR_diodeC_ckt();
>> DAE = MNA_EqnEngine(ckt);

>> dcop = dot_op(DAE);
>> dcop.print(dcop);

>> swp = dot_dcsweep(DAE, [], 'V1:::E', 0, 10, 50);
>> swp.plot(swp);
```

or simply run:

```
>> run_myR_diodeC_ckt;
```

Then edit myR_diodeC_ckt.m, change {'DC', 1} to {'DC', 10}, regenerate DAE, calculate operating point again:

```
>> ckt = myR_diodeC_ckt();
>> DAE = MNA_EqnEngine(ckt);
>> dcop = dot_op(DAE);
>> dcop.print(dcop);
```

How to deal with convergence problems?

8. Simple MOS model: Shichman Hodges model:

Use model_starter to create myNMOS model, with 3 terminals, 2 explicit I/Os (ids, igs), no internal unknowns.

Changes to make to myNMOS.m:

```
MOD = add_to_ee_model(MOD, 'parms', {'Beta', 1e-2});
MOD = add_to_ee_model(MOD, 'parms', {'Vth', 0.5});
MOD = add_to_ee_model(MOD, 'parms', {'Cgs', 1e-12});
MOD = add_to_ee_model(MOD, 'parms', {'Cgd', 1e-12});
```

% Fill in fe function:	% Fill in qe function:

if vgs < Vth
  ids_fe = 0;
elseif vds < vgs - Vth
  ids_fe = Beta * vds * (vgs - Vth - 0.5*vds);
else % vgs >= Vth && vds >= vgs - Vth
	Vgd = vgs - vds;
	
	Qgd = Cgd * Vgd;
	Qgs = Cgs * vgs;

ids_fe = 0.5 * Beta * (vgs - Vth)^2;
end
igs_fe = 0;
	ids_qe = -Qgd;
	igs_qe = Qgd + Qgs;

```
>> MOD = myNMOS();
>> check_ModSpec(MOD);

>> MEO = model_exerciser(MOD);
>> MEO.display(MEO);
>> MEO.plot('ids_fe', 0:0.05:1, 0:0.1:1, MEO);
```

9. Common source amplifier:

Run DC, AC and transient analyses on myNMOSamp_ckt:

```
>> run_myNMOSamp_ckt;
```

10. Diode with internal node:

Use model_starter to create diodeC_Rd model, with 2 terminals, 1 explicit I/O (ipn), 1 internal unknown (vpx), 1 implicit equation (KCL_x).

Changes to make to diodeC_Rd.m:

MOD = add_to_ee_model(MOD, 'parms', {'Vt', 0.0026, 'Is', 1e-12}); % thermal voltage and saturation current MOD = add_to_ee_model(MOD, 'parms', {'C', 1e-6}); MOD = add_to_ee_model(MOD, 'parms', {'Rd', 1.0});	
%add a MATLAB function in the file: function out = mydiode(vpx, Vt, Is) out = Is*(safeexp(vpx/Vt, 1e15) - 1); end	
% Fill in fe function:	% Fill in qe function:
ipn_fe = mydiode(vpx, Vt, Is);	out(1, 1) = C*vpx;
% Fill in fi function:	% Fill in qi function:
vRd = vpn-vpx; out(1, 1) = mydiode(vpx, Vt, Is) - vRd/Rd;	out(1, 1) = C*vpx;

Change run_myR_diodeC_ckt.m to use myR_diodeC_Rd_ckt circuit. Then run it again.

11. Diode model in implicit form:

Another way to model the diode with internal node is to write an implicit equation that directly describe the relation between I/Os (ipn and vpn), without using an internal unknown. This is similar to writing the resistor model as 0 = vpn – R*ipn.

Use model_starter to create a 2-terminal device diodeC_Rd_implicit with no explicit outputs and no internal unknowns. Then edit diodeC_Rd_implicit.m:

MOD = add_to_ee_model(MOD, 'parms', {'Vt', 0.0026, 'Is', 1e-12}); % thermal voltage and saturation current MOD = add_to_ee_model(MOD, 'parms', {'C', 1e-6}); MOD = add_to_ee_model(MOD, 'parms', {'Rd', 1.0});	
%add a MATLAB function in the file: function out = mydiode(vpx, Vt, Is) out = Is*(safeexp(vpx/Vt, 1e15) - 1); end	
% Fill in fi function:	% Fill in qi function:

Vd = vpn - ipn*Rd; out(1, 1) = mydiode(Vd, Vt, Is) - ipn;	Vd = vpn - ipn*Rd; out(1, 1) = C*Vd;

Change run_myR_diodeC_ckt.m to use myR_diodeC_Rd_implicit_ckt circuit. Then run it again.

Optional: we have introduced the concepts of internal unknowns and implicit equations in MAPP. A MOSFET device with drain and source resistances can be modelled in a similar way.
The model DSAwareSHMOSWithParasitics_ModSpec_wrapper.m uses the same SH equations as in myNMOS.m, but it also includes P-type, drain/source-inversion and drain/source parasitic resistances. To plot its characteristic curves in a circuit, run:

```
>> test_DSAwareSHMOSWithParasitics_ModSpec_wrapper;
```

12. A device with hysteresis:

```
>> model_starter();
```

Please enter the model's name (e.g., my_r_model):	Hys
… …	
The I/O names for this model are: vpn, ipn Among them, which one(s) can be expressed explicitly in your model equations?	vpn

```
>> edit Hys.m;
```

Changes to make to Hys.m:

```
MOD = add_to_ee_model(MOD, 'parms', {'A', 1});
MOD = add_to_ee_model(MOD, 'parms', {'B', -1});
MOD = add_to_ee_model(MOD, 'parms', {'C', 1});
MOD = add_to_ee_model(MOD, 'parms', {'I', 1});
MOD = add_to_ee_model(MOD, 'parms', {'tau', 1e-7});
```

```
vpn_fe = A*(ipn-I)^3 + B*(ipn-I) + C;
```

```
vpn_qe = tau*ipn;
```

```
>> MOD = Hys();
>> check_ModSpec(MOD);

>> MEO = model_exerciser(MOD);
>> MEO.display(MEO);
>> MEO.plot('vpn_fe', (0:0.01:2.2), MEO);
```

When we connect this device with a resistor and voltage source in series, load line analysis tell us that at certain supply voltages, the circuit has multiple stable operating points. The following script sweeps the supply voltage, in both DC sweep and transient simulations:

```
>> run_myR_Hys_ckt;
```

Using the hysteresis in the device, we can also build a relaxation oscillator. The following script runs transient simulation on the oscillator:

```
>> run_Hys_osc;
```