λSpice

Seyed Nematollah Ahmadyan

This is a very minimalistic spice that I use for playing with circuit’s ODE. Now, this little tool supports MNA analysis, DC operating mode analysis, AC analysis and transient analysis. Currenlty, it supports standard voltage and current sources, RLC elements and diode and bjt transistors. To execute the program, you need to call the kernel function. Also configuration.m and init.m holds program and user variables (such as input file name, etc.)

For ease of use, I wrote some specific script (bjt\_ac.m, amp\_tran1.m, ) that will call the kernel for each test-case. To reproduce these results, please execute those scripts.

File Configuration.m contains global configuration parameter. The most important one is Config.filename which holds the input file name. Also Init.m will hold some execution parameters. Again, the most important parameter is the execution mode in core.mode for:

0 means DC analysis

1 means AC analysis

2 means Transient

Init would also define which node we want to probe. For adding a probe, the probe must be defined in init.m as a node number. The node number can be accessed through core.nodes variable. Again! I recommend executing my scripts (bjt\_ac, amp\_tran and amp741\_ac.m)

The rest of this report is as follows: First, I describe my results for each mode of operation (DC, AC and TR). In each section, I first show a small example or two to proof that my method works. Then I’ll show the result of the given test-case.

Later in section Model, I describe the device models and how I used them. I’ll also put the MATLAB code to make it more clearer.

# DC Analysis

To set DC mode, in init.m select core.mode=0,

To set file input, in configuration.name enter the filename

For dc analysis, in case the circuit have any nonlinear devices (Diodes and BJT transistors), we use newton-raphson iteration to replace them with their linear equivalent. We continue the NR loop until the error is below config. config.DCErrorThreshold (default=10e-7)

Result for Diode: (to reproduce: run diode\_dc.m)

It will converge in 4 iteration.

2.000000000000000e+00

7.030987560309102e-01

-6.484506219845449e-04

[info] Elapsed time = 0.0913795 seconds.

Result for BJT:

ans =

5.000000000000000e+00

4.345482690265371e+00

9.222950655469744e-02

6.552648514155015e-01

9.000000000000000e-01

9.000000000000000e-01

-1.980392156877698e-03

-2.447351485844986e-05

-2.447351485844986e-05

iteration 7, difference=0.000000

converged after 7 iteration, final error=0.000000

[info] Elapsed time = 0.0486986 seconds.

ans =

2.000000000000000e+00

7.030987560309102e-01

-6.484506219845449e-04

iteration 4, difference=0.000000

converged after 4 iteration, final error=0.000000

[info] Elapsed time = 0.410302 seconds.

These results are a bit different from TA’s golden results. Mainly for two reason:

I use a Level-4 BJT model which holds more data than default BJT model. This is my parameter list:

% .MODEL t2n2222a NPN

% + ISS= 0. XTF= 1. NS = 1.00000

% + CJS= 0. VJS= 0.50000 PTF= 0.

% + MJS= 0. EG = 1.10000 AF = 1.

% + ITF= 0.50000 VTF= 1.00000

% + BR = 40.00000 IS = 1.6339e-14 VAF= 103.40529

% + VAR= 17.77498 IKF= 1.00000

% + NE = 1.31919 IKR= 1.00000 ISC= 3.6856e-13

% + NC = 1.10024 IRB= 4.3646e-05 NF = 1.00531

% + NR = 1.00688 RBM= 1.0000e-02 RB = 71.82988

% + RC = 0.42753 RE = 3.0503e-03 MJE= 0.32339

% + MJC= 0.34700 VJE= 0.67373 VJC= 0.47372

% + TF = 9.693e-10 TR = 380.00e-9 CJE= 2.6734e-11

% + CJC= 1.4040e-11 FC = 0.95000 XCJC= 0.94518

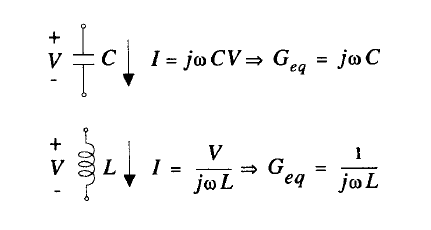
Also for helping the convergence of bjt, I add a small gmin to the linearized model with each iteration. This is the reason that my results are slightly (less than 1%) different from golden results, but they match the hspice.

To see the model for bjt and diode, please refer to appendix.

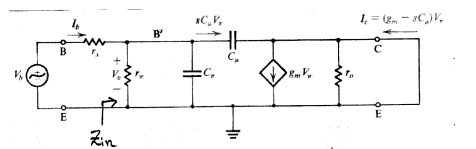
# AC Analysis

For AC analysis, first select the input file name in configuration and then enable the ac mode by setting core.mode=1 in init.m

In AC analysis, I cut the DC sources and replace Capacitors and Inductors by Cjw and jw/L.



Also I repalce the transistor with the small-signal bjt model around Q-point, I calculate the CCs, Cbe, Cbc and stamp them into the Y matrix.

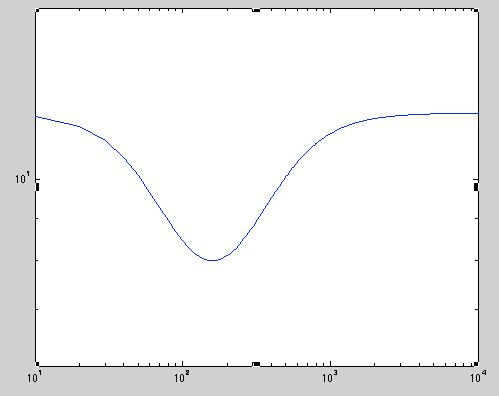


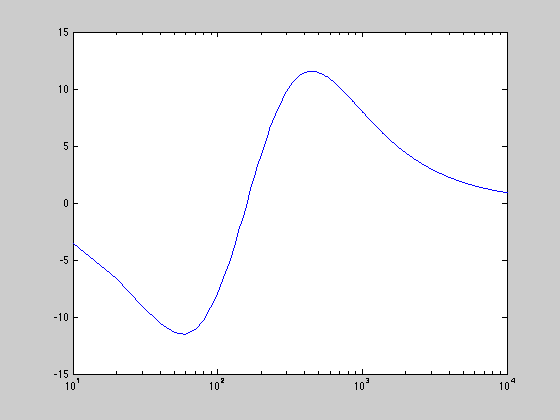
most of these variabes, including gm are computed during DC analysis. But the rest (capacitors) most be calculated at each frequency. The equations for this analysis (including functions to compute charge and capacitor of the transistor) are provided in Appendix.

**Example 1: RC Network**

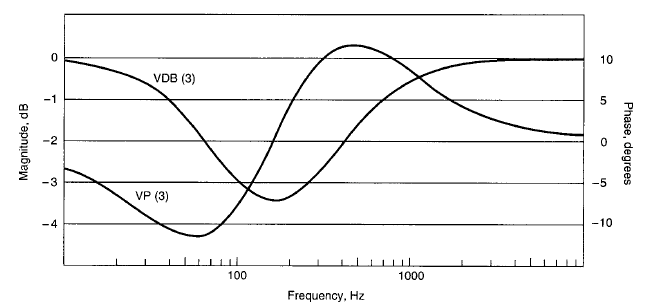
As a proof of concept, I analyzed a bridge-T netwok from the spice book.

For a simple Bridge-T network (From The SPICE Book, Page 145) this is my magnitude and phase results (shown in degrees and db in log scale).



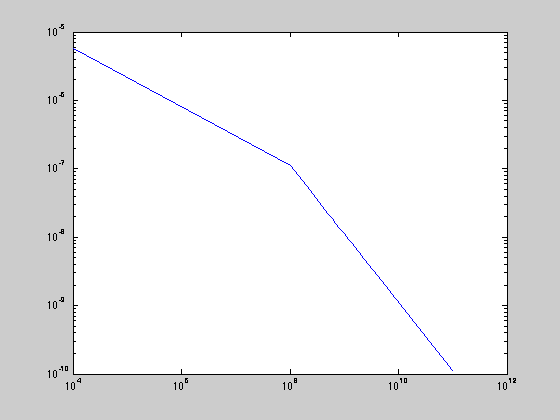


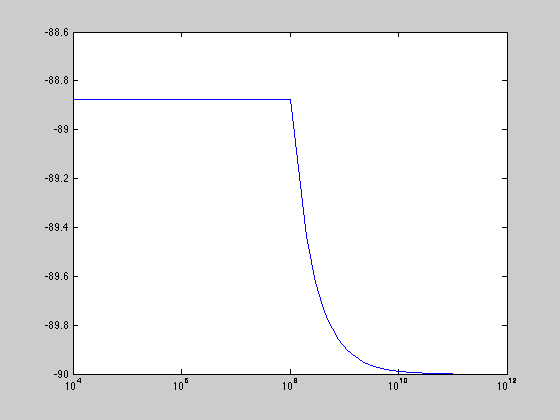
which verifies the result:



**Example 2: Common Emitter BJT:**

This is a single biased bjt transistor with ac input. With respect to the inaccuracies in our bjt modeling, this is an accurate frequency response that closely matches spice’s results. To reproduce these results, execute bjt\_ac2 script

****

****

**Case studies:**

**BJT Case study: (To reproduce this result, execute bjt\_ac)**

**Operating Point**

**DC Operating point information:**

**ans =**

**5.000000000000000e+00**

**4.345482690265371e+00**

**9.222950655469744e-02**

**6.552648514155015e-01**

**9.000000000000000e-01**

**9.000000000000000e-01**

**-1.980392156877698e-03**

**-2.447351485844986e-05**

**-2.447351485844986e-05**

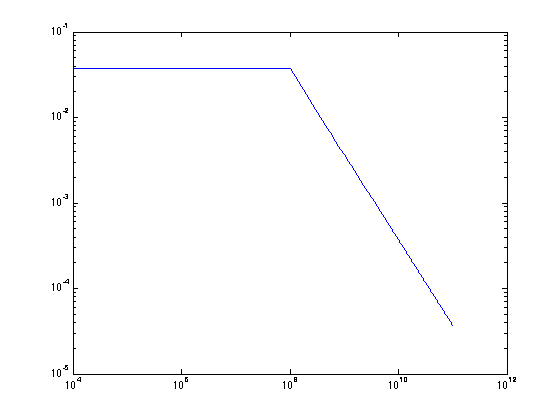
**iteration 7, difference=0.000000**

**converged after 7 iteration, final error=0.000000**

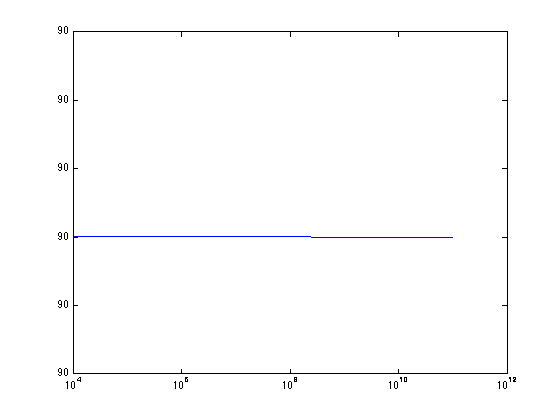
**[info] Elapsed time = 0.0533941 seconds.**

**AC Analysis took 6.87314 seconds.**

**Magnitude Diagram**

****

**Phase:**

****

# OpAmp 741:

**For op-amp 741, my program didn’t work ☹**

**DC Operating point:**

+1.00000000000e+00

1.00056805948e+00

-3.06337886154e-01

+1.43292813284e+01

-1.42345556261e+01

+1.42350559196e+01

-1.36756936314e+01

-1.34854441661e+01

-1.43378483480e+01

-1.49906487031e+01

-1.49906487031e+01

+3.46831056923e-01

+3.47115086665e-01

-1.41625109432e+01

-1.49277097252e+01

-4.85651138006e-01

+1.72212380071e+00

+1.04297243622e+00

+2.79035671847e-01

-1.49999999999e+01

+9.97381318928e-01

+1.00056805948e+00

+1.00375736380e+00

-1.49055366526e+01

+1.50000000000e+01

-1.50000000000e+01

-2.33876888433e-03

+2.33886280095e-03

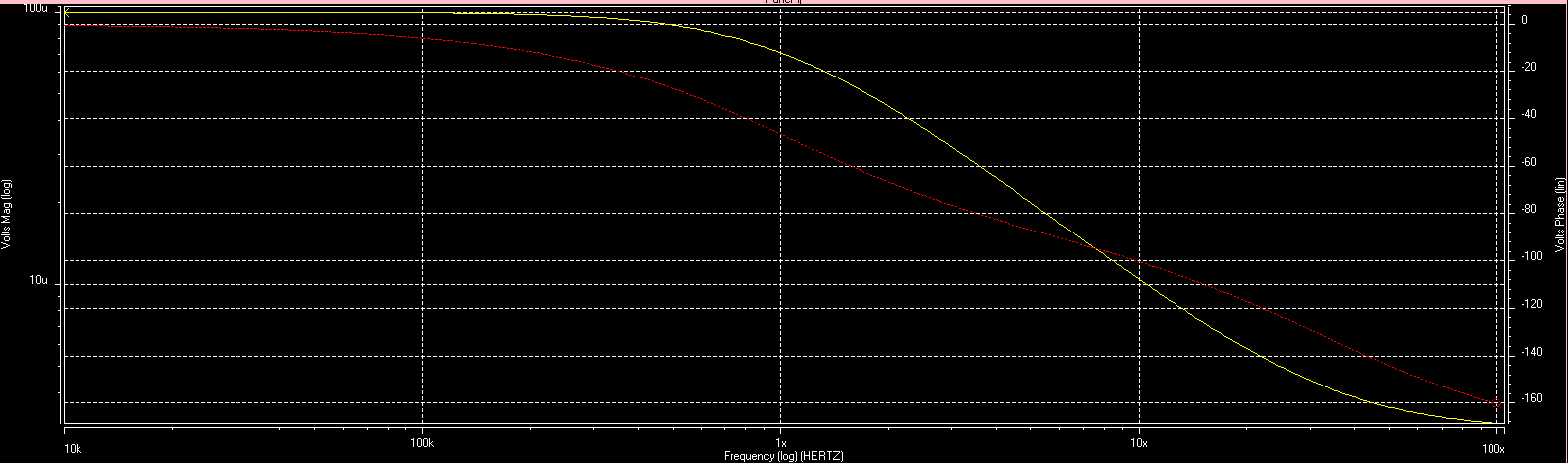
-9.49541493439e-08

-9.39166219288e-08

Iterated after 7.

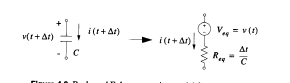
But my small signal models failed to produce a good results.

**This is the aaspice result.**

****

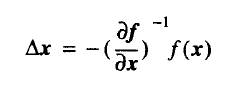
# Transient Analysis

For transient analysis, I used backward backward euler for integration of capacitor and inductors (including capacitors inferred by bjts). I use Norton’s circuit companion model for both Capacitor and Inductor, since it does not add a note to the network and it is easier to handle.

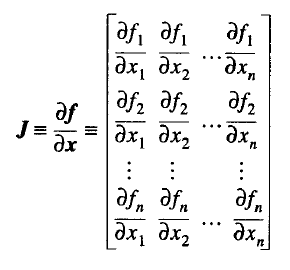


For BJT, I use a multi-dimentional newton-raphson method (as described in Sec. 10.5 of textbook)

So each at each time step, I compute this:



where



Therefore:

Stamping J

core = stampJ(core, b, -ib - (gec+gcc)\*vbc - (gee+gce)\*vbe);

core = stampJ(core, c, -ic + gce\*vbe + gcc\*vbc);

core = stampJ(core, e, -ie + gee\*vbe + gec\*vbc);

Stamping Y

core = stampY(core, b, b, -gee-gec-gce-gcc + gmin\*2);

core = stampY(core, b, c, gec+gcc - gmin);

core = stampY(core, b, e, gee+gce - gmin);

core = stampY(core, c, b, gce+gcc - gmin);

core = stampY(core, c, c, -gcc + gmin);

core = stampY(core, c, e, -gce);

core = stampY(core, e, b, gee+gec - gmin);

core = stampY(core, e, c, -gec);

core = stampY(core, e, e, -gee + gmin);

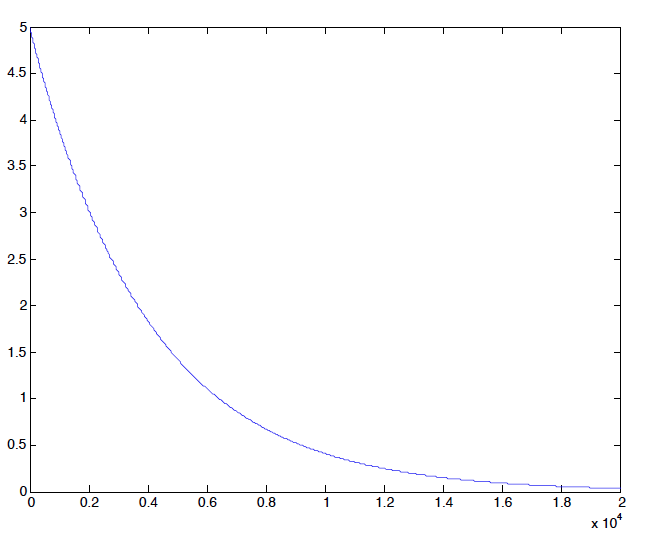
All the necessary equations to compute g(s) are shown in appendix.

To enable the transient analysis, first select the input file from configuration.m

Then in init.m, set core.mode=2

**Example 1: RC network**

For validating the transient analysis, I simulated a simple RC network:



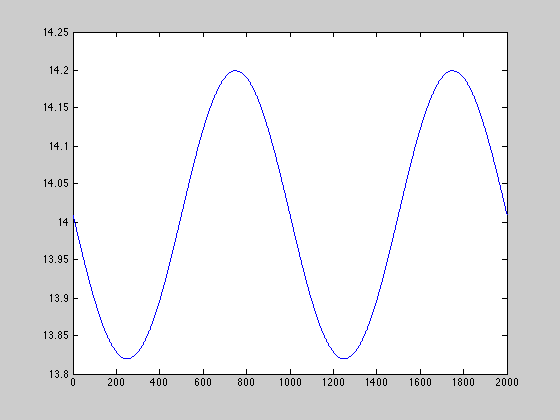
Case study: differential amplifier: (to reproduce: amp\_tran1.m)  
My results are matched with hspice result. For hspice, I used the following bjt model:

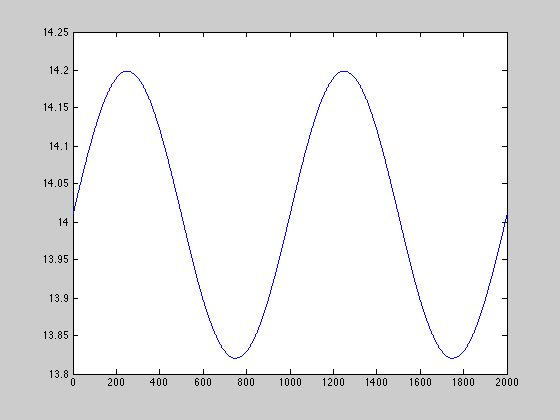
.model n1 npn is=1e-14 bf=100

+ ne=2 ikf=20ma ise=200

\*

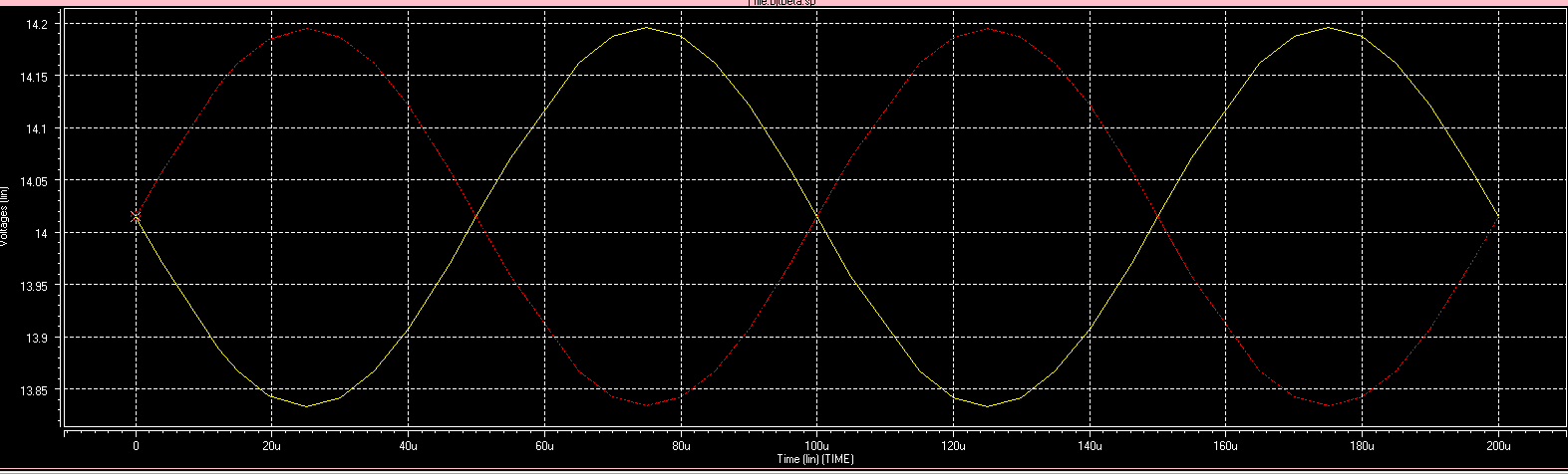
My result:





[info] Elapsed time = 11.9135 seconds.

Hspice result:



Example computation:

voltages: -0.000063 14.010781 -0.595287

param: beta=100.000000 vcrit=0.670366 fgain=0.990099

T: -14.010843 0.595224

gain 100.000001

T -14.010843 0.595224 -0.000999 0.000989

gee= -0.038648

gec= 0.000000

gce= 0.038266

gcc= -0.000000

gce+gcc= 0.038266

gee+gec= -0.038648

-------------------

voltages: 0.000000 14.008373 -0.595287

param: beta=100.000000 vcrit=0.670366 fgain=0.990099

T: -14.008373 0.595287

gain 100.000001

T -14.008373 0.595287 -0.001002 0.000992

gee= -0.038743

gec= 0.000000

gce= 0.038359

gcc= -0.000000

gce+gcc= 0.038359

gee+gec= -0.038743

ans =

-4.898587196589413e-18

1.500000000000000e+01

-1.500000000000000e+01

1.400957479823469e+01

-5.952558109617545e-01

1.400957479063045e+01

-9.904251966652978e-06

-1.980850411134855e-03

2.000658915144201e-03

iteration 1, difference=0.001704

-------------------

voltages: -0.000000 14.009575 -0.595256

param: beta=100.000000 vcrit=0.670366 fgain=0.990099

T: -14.009575 0.595256

gain 100.000001

T -14.009575 0.595256 -0.001000 0.000990

gee= -0.038696

gec= 0.000000

gce= 0.038312

gcc= -0.000000

gce+gcc= 0.038312

gee+gec= -0.038696

-------------------

voltages: 0.000000 14.009575 -0.595256

param: beta=100.000000 vcrit=0.670366 fgain=0.990099

T: -14.009575 0.595256

gain 100.000001

T -14.009575 0.595256 -0.001000 0.000990

gee= -0.038696

gec= 0.000000

gce= 0.038312

gcc= -0.000000

gce+gcc= 0.038312

gee+gec= -0.038696

ans =

-4.898587196589413e-18

1.500000000000000e+01

-1.500000000000000e+01

1.400957479312242e+01

-5.952557919069493e-01

1.400957479312242e+01

-9.904252017775848e-06

-1.980850413755161e-03

2.000658917790701e-03

iteration 2, difference=0.000000

converged after 2 iteration, final error=0.000000

# Models Capacitor

if(core.mode==1), %AC Analysis

g= c\*2\*pi\*j\*core.freq;

if(g>1e20) g=1e20;end

core=stampConductance(core, n1, n2, g);

disp(core.Y)

elseif(core.mode==2) %Transient Analysis

if(n1~=0), v1 = core.vNodeset(n1);

else v1 = 0; end

if(n2~=0), v2 = core.vNodeset(n2);

else v2 = 0; end

v=v1-v2;

% capacitor companion model using trapezoidal approximation

% (Norton equivalent) consists of a current source in

% parallel with a resistor. Trapezoidal is more accurate

% than backward euler but can cause oscillatory behavior

% if RC is small relative to the timestep.

%g = core.dt/(2\*c); %trapezoidal

%curSourceValue = -v/g-current;

g = c/core.dt; %BE

i = -g\*v;

% double voltdiff = volts[0] - volts[1];

%// we check compResistance because this might get called

%// before stamp(), which sets compResistance, causing

%// infinite current

%if (compResistance > 0)

%current = voltdiff/compResistance + curSourceValue;

%System.out.println("current=" + current + " voltdiff=" + voltdiff + " compresistance=" + compResistance + " curSourceValue=" + curSourceValue );

%disp(sprintf('g=%f c=%f i=%f v=%f v1=%f v2=%f', g, c, i, v, v1, v2));

core = stampConductance( core, n1, n2, g);

core = stampCurrentSource(core, n1, n2, i );

end

## Diode

## BJT

function core=stampBJT(core,t)

bF=100;% forward beta

bR=4; % reverse beta

Va=10; % Early voltage

%tau = 2 \* 10^(-11) ;

leakage = 1e-13;% saturation current

vt = 0.02585126075 ; % 1/40 % thermal voltage

Vtf= 1;

Cj = 10 ^ (-14) ;

%Vj = 0.8 ;

%fc = 0.5 ;

mj = 0.5 ;

beta=100;

fgain=beta/(beta+1);

gmin=0;

Is = 10^(-15); % saturation current

% .MODEL t2n2222a NPN

% + ISS= 0. XTF= 1. NS = 1.00000

% + CJS= 0. VJS= 0.50000 PTF= 0.

% + MJS= 0. EG = 1.10000 AF = 1.

% + ITF= 0.50000 VTF= 1.00000

% + BR = 40.00000 IS = 1.6339e-14 VAF= 103.40529

% + VAR= 17.77498 IKF= 1.00000

% + NE = 1.31919 IKR= 1.00000 ISC= 3.6856e-13

% + NC = 1.10024 IRB= 4.3646e-05 NF = 1.00531

% + NR = 1.00688 RBM= 1.0000e-02 RB = 71.82988

% + RC = 0.42753 RE = 3.0503e-03 MJE= 0.32339

% + MJC= 0.34700 VJE= 0.67373 VJC= 0.47372

% + TF = 9.693e-10 TR = 380.00e-9 CJE= 2.6734e-11

% + CJC= 1.4040e-11 FC = 0.95000 XCJC= 0.94518

Cje0=2.6734e-11;

Vje=0.67373;

Mje= 0.32339;

Mjs=0;

Cjbe = 10e-11;

Cjbc = Cjbe;

Cjcs = 2\*10e-14;

Cjs0 = 0 ;

Vjs= 0.50000;

Fc=0.95000;

Cjc0=Cj;

Xcjc= 0.94518 ;

Vjc= 0.47372 ;

Mjc= 0.34700 ;

Tr = 380.00e-9;

Tf= 9.693e-10;

VAf=100;

VAr=100;

vdcoef = 1/vt;

rgain = .5;

vcrit = vt\*log( vt/(sqrt(2)\*leakage) );

c=core.Element(t).Node1 ;

b=core.Element(t).Node2 ;

e=core.Element(t).Node3 ;

if (strncmpi(core.Element(t).Type, 'npn',4)==1) polarity=1 ;

else polarity=-1; end

if(c ~= 0) vc = core.vNodeset( c );

else vc = 0; end

if(b ~= 0) vb = core.vNodeset( b );

else vb = 0; end

if(e ~= 0) ve = core.vNodeset( e );

else ve = 0; end

vbe=vb-ve;

vce=vc-ve;

vbc=vb-vc;

core.isNonlinear=1;

if(core.mode==1) %AC small-signal model

w=core.freq;

%interpret zero as infinity for that model parameter

if( Vtf > 0 )

Vtf=1.0 / Vtf ;

else

Vtf=0;

end

%depletion capacitance of base-emitter diode

Cbe = pnCapacitance (vbe, Cje0, Vje, Mje, Fc);

Qbe = pnCharge (vbe, Cje0, Vje, Mje, Fc);

%depletion and diffusion capacitance of base-collector diode

Cbci = pnCapacitance (vbc, Cjc0 \* Xcjc, Vjc, Mjc, Fc) ; %+ Tr \* gir;

Qbci = pnCharge (vbc, Cjc0 \* Xcjc, Vjc, Mjc, Fc) ;%+ Tr \* Ir;

%depletion and diffusion capacitance of external base-collector capacitor

Cbcx = pnCapacitance (vce, Cjc0 \* (1 - Xcjc), Vjc, Mjc, Fc);

Qbcx = pnCharge (vce, Cjc0 \* (1 - Xcjc), Vjc, Mjc, Fc);

%depletion capacitance of collector-substrate diode

Ccs = pnCapacitance2 (vc, Cjs0, Vjs, Mjs);

Qcs = pnCharge2 (vc, Cjs0, Vjs, Mjs);

Ibf = (Is/bF)\*(exp(vbe/vt)-1); %Eq. 13

Ibr = (Is/bR)\*(exp(vbc/vt)-1); %Eq. 14

Icc = bF\*Ibf-bR\*Ibr; %\*(1+vce/va)

gPiF = Ibf/vt ; %Eq. 15, g\_ce

gPiR = Ibr/vt ; %Eq. 16, g\_ec

gmf = bF\*gPiF ; % g\_ee

gmR = bR\*gPiR ; % g\_cc

go = Icc/Va;

%gbe = gbei + gben;

%gbc = gbci + gbcn;

%gmfr = gm;

Cbe=1e-9;

Cbci=1e-9;

Ccs=1e-9;

%compute admittance matrix entries

Ybe = 1i \* 2.0 \* pi \* w \* Cbe ; %+ gbe ;

Ybc = 1i \* 2.0 \* pi \* w \* Cbci;%+gbc;

Ycs = 0.0 + 1i \* 2.0 \* pi \* w \* Ccs;

disp(sprintf('capacitances= %f %f %f , %f %f %f', Cbe, Cbci, Ccs, Ybe, Ybc, Ycs ));

core = stampY(core, b, b, Ybc + Ybe);

core = stampY(core, b, c, -Ybc);

core = stampY(core, b, e, -Ybe);

core = stampY(core, c, b, -Ybc + gmf);

core = stampY(core, c, c, Ybc + Ycs + go);

core = stampY(core, c, e, -gmf - go);

core = stampY(core, e, b, -Ybe - gmf);

core = stampY(core, e, c, -go);

core = stampY(core, e, e, Ybe + gmf + go);

else %DC and Transient Analysis, large-signal model

disp('-------------------');

%power-consumption: (v0-v2)\*ib + (v1-v2)\*ic

gmin = 0;

if (core.dcIteration > 10),

% if we have trouble converging, put a conductance in parallel with all P-N junctions.

% Gradually increase the conductance value for each iteration.

gmin = exp(-9\*log(10)\*(1-core.dcIteration/3000.));

if (gmin > .1)

gmin = .1;

end

end

%vbc = pnp\*limitStep(pnp\*vbc, pnp\*lastvbc);

%vbe = pnp\*limitStep(pnp\*vbe, pnp\*lastvbe);

%lastvbc = vbc;

%lastvbe = vbe;

pcoef = vdcoef\*polarity;

expbc = exp(vbc\*pcoef);

expbe = exp(vbe\*pcoef);

if (expbe < 1)

expbe = 1;

end

ie = polarity\*leakage\*(-(expbe-1)+rgain\*(expbc-1));

ic = polarity\*leakage\*(fgain\*(expbe-1)-(expbc-1));

ib = -(ie+ic);

gee = -leakage\*vdcoef\*expbe;

gec = rgain\*leakage\*vdcoef\*expbc;

gce = -gee\*fgain;

gcc = -gec\*(1/rgain);

disp(sprintf('voltages: %f %f %f', vb, vc, ve));

disp(sprintf('param: beta=%f vcrit=%f fgain=%f', beta, vcrit, fgain ));

disp(sprintf('T: %f %f ', vbc, vbe));

disp(sprintf('gain %f ', ic/ib));

disp(sprintf('T %f %f %f %f ', vbc, vbe, ie, ic));

disp(sprintf('gee= %f ', gee));

disp(sprintf('gec= %f ', gec));

disp(sprintf('gce= %f ', gce));

disp(sprintf('gcc= %f ', gcc));

disp(sprintf('gce+gcc= %f ', gce+gcc));

disp(sprintf('gee+gec= %f ', gee+gec));

%pause

% stamps from page 302 of Pillage. Node 0 is the base,

% node 1 the collector, node 2 the emitter. Also stamp

% minimum conductance (gmin) between b,e and b,c

core = stampY(core, b, b, -gee-gec-gce-gcc + gmin\*2);

core = stampY(core, b, c, gec+gcc - gmin);

core = stampY(core, b, e, gee+gce - gmin);

core = stampY(core, c, b, gce+gcc - gmin);

core = stampY(core, c, c, -gcc + gmin);

core = stampY(core, c, e, -gce);

core = stampY(core, e, b, gee+gec - gmin);

core = stampY(core, e, c, -gec);

core = stampY(core, e, e, -gee + gmin);

% we are solving for v(k+1), not delta v, so we use formula

% 10.5.13, multiplying J by v(k)

core = stampJ(core, b, -ib - (gec+gcc)\*vbc - (gee+gce)\*vbe);

core = stampJ(core, c, -ic + gce\*vbe + gcc\*vbc);

core = stampJ(core, e, -ie + gee\*vbe + gec\*vbc);

end

end

BJT Capacitance

function c = pnCapacitance( Uj, Cj, Vj, Mj, Fc )

%pnCapacitance Computes pn-junction depletion capacitance.

if (Uj <= Fc \* Vj)

c = Cj \* exp (-Mj \* log (1 - Uj / Vj));

else

c = Cj \* exp (-Mj \* log (1 - Fc)) \* (1 + Mj \* (Uj - Fc \* Vj) / Vj / (1 - Fc));

end

end

function c = pnCapacitance2 ( Uj, Cj, Vj, Mj)

% This function computes the pn-junction depletion capacitance with

% no linearization factor given.

if (Uj <= 0)

c = Cj \* exp (-Mj \* log (1 - Uj / Vj));

else

c = Cj \* (1 + Mj \* Uj / Vj);

end

end

BJT Charge

function q = pnCharge ( Uj, Cj, Vj, Mj, Fc)

%Computes pn-junction depletion charge.

if (Uj <= Fc \* Vj),

a = 1 - Uj / Vj;

b = exp ((1 - Mj) \* log (a));

q = Cj \* Vj / (1 - Mj) \* (1 - b);

else

% Approach 1

% a = 1 - Fc;

% b = exp ((1 - Mj) \* log (a));

% a = exp ((1 + Mj) \* log (a));

% c = 1 - Fc \* (1 + Mj);

% d = Fc \* Vj;

% e = Vj \* (1 - b) / (1 - Mj);

% q = Cj \* (e + (c \* (Uj - d) + Mj / 2 / Vj \* (sqr (Uj) - sqr (d))) / a);

%Approach 2: this variant is numerically more stable

a = 1 - Fc;

b = exp (-Mj \* log (a));

f = Fc \* Vj;

c = Cj \* (1 - Fc \* (1 + Mj)) \* b / a;

d = Cj \* Mj \* b / a / Vj;

e = Cj \* Vj \* (1 - a \* b) / (1 - Mj) - d / 2 \* f \* f - f \* c;

q = e + Uj \* (c + Uj \* d / 2);

end

end

function q = pnCharge2(Uj, Cj, Vj, Mj)

% This function computes the pn-junction depletion charge with no

% linearization factor given.

if (Uj <= 0)

q = Cj \* Vj / (1 - Mj) \* (1 - exp ((1 - Mj) \* log (1 - Uj / Vj)));

else

q = Cj \* Uj \* (1 + Mj \* Uj / 2 / Vj);

end

end

# DIODE

function core=stampDiode(core, i)

n1=core.Element(i).Node1;

n2=core.Element(i).Node2;

if(n1~=0)

v1 = core.vNodeset(n1);

else

v1 = 0;

end

if(n2~=0)

v2 = core.vNodeset(n2);

else

v2 = 0;

end

%diode equations:

Is = 10^(-15); % saturation current

Vt = 0.02585126075 ; % 1/40 % thermal voltage

vd=v1-v2;

id = Is \* ( exp(vd/Vt)-1 ); %standard diode equation

geq = id/Vt ; % g\_eq = dId/dVd = IS/Vt \* exp(Vd/vt) = i(v\_n)/V\_t

Ieq = id - geq\*vd ;

core=stampConductance(core, n1, n2, geq);

core=stampCurrentSource(core, n1, n2, Ieq);

core.isNonlinear=1;

end