NONLINEAR ELECTRONICS (NOEL) PACKAGE 3:

NONLINEAR DC ANALYSIS

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/26

21 March 1986

ELECTRONICS RESEARCH LABORATORY

COLLEGE OF ENGINEERING
University of California, Berkeley
94720

NONLINEAR ELECTRONICS (NOEL) PACKAGE 3:

NONLINEAR DC ANALYSIS

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/26

21 March 1986

ELECTRONICS RESEARCH LABORATORY

COLLEGE OF ENGINEERING
University of California, Berkeley
94720

NOEL PACKAGE 3 : NONLINEAR DC ANALYSIS†

An-Chang Deng and Leon O. Chua

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, CA 94720

ABSTRACT

The program in this package performs the DC analysis of a nonlinear resistive circuit. A preprocessor of circuit formulation routine is called to translate the circuit into a C source code, which describes the circuit equation $\mathbf{f}(\mathbf{x}) = 0$ and the corresponding Jacobian matrix $\mathbf{J}_\mathbf{f} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. The Newton-Raphson algorithm is then applied to solve the circuit equation through iterations.

March 17, 1986

† Research supported by the Joint Services Electronics Program under Contract F49620-84-C-0057, and the Semiconductor Research Corporation under Grant SRC 82-11-008.

NOEL PACKAGE 3 : NONLINEAR DC ANALYSIS

1. Introduction

DC analysis of resistive networks is a fundamental problem in circuit simulation since it is the first step for other analyses such as transient or ac analysis. Using the circuit formulation utility program[1], we can formulate any given circuit as a set of circuit equations which are written in the form of a C function

equation(f, x, x_dot, t)

Since no dynamic element is considered, so x_dot will not appear in this function. It is therefore a C function describing an algebraic equation $f(x) = 0$. In addition to the equation routine, the symbolic expression of the Jacobian matrix is also listed in a second C function

jacob(x, x_dot, dx_dot, jf, t)

which directly evaluates the Jacobian matrix.

Hence, with the above C source code for describing the circuit equations and the corresponding Jacobian matrix, the dc operating point of the resistive circuit can be found by applying the Newton-Raphson algorithm to solve the algebraic equation $f(x) = 0$. The initial guess for Newton-Raphson iteration is given by the user or set to zero as default. For the circuit with multiple operating points, the user may start with various sets of initial guesses in order to converge to the different solutions.

2. Algorithm

Step 1.

Find the C source code describing the circuit equation and the corresponding Jacobian matrix.

Step 2.

Apply Newton-Raphson iteration †

$$x^{(l+1)} = x^{(l)} - \lambda[J_f(x^{(l)})]^{-1}f(x^{(l)}) \tag{1}$$

until it converges with

$$||x^{(l+1)} - x^{(l)}||_1 < \epsilon_1 = 10^{-7} \tag{2}$$

and $l < 20$, where $x^{(0)}$ is the initial guess entered by the user or set to zero by default selection.

Step 3.

If $l \geq 20$, go to Step 4; else store the convergent solution x to the output file "xx...x.op" and increment s if x differs from any of the previously found solution in "xx...x.op"; more specifically, if

$$||x - x^{[i]}||_1 > \epsilon_2 = 10^{-6} \tag{3}$$

for i=1,2,...,s, where "xx...x.op" contains s distinct dc operating points $x^{[1]}, x^{[2]}, \ldots, x^{[s]}$.

Step 4.

Stop the dc analysis if the user decides to abort; else go to Step 2 with a new initial guess.

† The scaling factor $0 < \lambda < 1$ is introduced to avoid numerical overflow when f() contains functions which are sensitive to the variation of x. Without λ, the iteration formula Eq.(1) converges very slowly (or not at all) especially when the circuit contains bipolar transistors modeled by exponential functions.

3. User's Instruction

Step 1.

Create a file "xx...x.spc" which describes the resistive circuit to be analyzed and follows the rules of the input format language defined in [2] for each class of circuit elements, where "xx...x" is the filename for the input file with type extension ".spc". Only the resistive elements can be included in "xx...x.spc"; namely

'R' : 2-terminal resistor (linear or general nonlinear char.)

'V' : independent voltage source (time-invariant or time-varying)

'I' : independent current source (time-invariant or time-varying)

'E' : linear voltage-controlled voltage source

'F' : linear current-controlled current source

'G' : linear voltage-controlled current source

'H' : linear current-controlled voltage source

'K' : nonlinear controlled source (at most 2 controlling variables)

'N' : 2-port or 3-terminal resistor (linear or general nonlinear char.)

The nonlinear 1-port or 2-port resistor described by pwl characteristic can appear in "xx...x.spc" only when it is described by absolute function fabs() instead of the numerical expression defined in [2].

Steps 2-5 are combined as a batch process "dcsim.bat" and are executed by typing the command

dcsim xx...x

where "xx...x.spc" is the input file.

Step 2.

Type the command

form xx...x

to produce the C source code "xx...x.c" which includes the circuit equation and the Jacobian matrix routines. It also produces the table file "xx...x.tbl" which contains a mapping table between the circuit equation variables **x** and the element voltages or currents (e.g., v(R1), i(R2)).

Step 3.

Compile the source code "xx...x.c".

Step 4.

Link the object code with the dc simulation routines.

Step 5.

Type the command

nrdc xx...x

to perform dc analysis which proceeds interactively with the user in the following steps :

(a) **default initial guess? y/n**

Type 'y' for default initial guess ($x^{(0)} = 0$) in Newton-Raphson iteration.

(b) If 'n' in (a), the user has to give the initial guess for each controlling variable (see Sec. 4).

v(R1) =
i(R2) =
....

(c) If the iteration converges, the convergent solution will be printed and ask the user whether to try another solution.

convergent solution .....
v(R1) =
i(R2) =
....
would you like to try another solution? y/n

If 'y' then repeat the process from (a) with a new initial guess; else stop the dc analysis.

(d) If the iteration does not converge, or iterates to a point with singular Jacobian matrix, it will stop the iteration and print the last iteration point. It also prompts a message to ask the user whether to continue the analysis :

not convergent; the last iteration .....
v(R1) =
i(R2) =
....
would you like to continue? y/n

If 'y' then repeat the process from (a) with a new initial guess, else stop the dc analysis.

4. Output Format

Each computed operating point is written into the output file "xx...x.op" if it differs from any operating point already existing in "xx...x.op". The operating point, immediately following a line with stars "*****.....**", is given in terms of the controlling variable of each nonlinear or time-varying element; namely

(1) voltage of a 2-terminal voltage-controlled resistor

(2) current of a 2-terminal current-controlled resistor

(3) voltage and current of a 2-terminal implicit-relation resistor

(4) voltage of a time-varying independent current source

(5) current of a time-varying independent voltage source

(6) port-1 voltage and port-2 voltage of a voltage-voltage controlled 2-port resistor

(7) port-1 current and port-2 current of a current-current controlled 2-port resistor

(8) port-1 voltage and port-2 current of a voltage-current controlled 2-port resistor

(9) port-1 current and port-2 voltage of a current-voltage controlled 2-port resistor

(10) controlled-branch voltage (resp.; current) of a nonlinear controlled current source (resp.; voltage source)

voltage (resp.; current) of the i-th controlling port if it is an open-circuit voltage (resp.; short-circuit current)

The output file is terminated with a line "&&&&.....&&" to denote the end of file. A standard output format for the operating point(s) in "xx...x.op" is shown as follows :

```
*************************
v(R1)=......
i(R2)=......
 ...........
*************************
v(R1)=......
i(R2)=......
 ...........
&&&&&&&&&&&&&&&&
```

5. Examples

Example 1 : in file "ex1.spc"

A single transistor type-S negative resistance circuit (Fig.1)

Example 2 : in file "ex2.spc"

A type-N 2-transistor negative resistance circuit (Fig.2)

Example 3 : in file "ex3.spc"

Schmitt-Trigger circuit (Fig.3)

6. Diagnosis

1. NRDC OUTPUT_FILE

   Bad command line, the correct one should be

   nrdc xx...x

   where "xx...x.spc" is the input file.

2. CAN'T OPEN THE TABLE FILE xx...x.tbl

   The table file "xx...x.tbl" does not exist in the current directory.

3. CAN'T OPEN THE DC_OP FILE xx...x.op

   Can't open "xx...x.op" due to insufficient disk space or too many files in the current directory.

4. SINGULAR JACOBIAN MATRIX

   The Jacobian matrix is singular or some of the matrix entries numerically overflow (e.g., 3.0E+256); should try another initial guess.

5. MORE THAN 10 DC OPERATING POINTS

   Number of operating points is beyond the maximal allowable number (equal to 10); should stop the dc analysis and copy the output file "xx...x.op" to a temporary file, and then restart the dc analysis to store more operating points to the file "xx...x.op" until the limit 10 is reached again.

References

[1] A.C.Deng and L.O.Chua, "NOnlinear ELectronics utility programs"

[2] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 0 : general description," ERL Memo. M86, University of California, Berkeley, 1986.

Figure Captions

Fig.1 A single transistor type-S negative resistance circuit.

Fig.2 A type-N 2-transistor negative resistance circuit.

Fig.3 Schmitt-Trigger circuit.

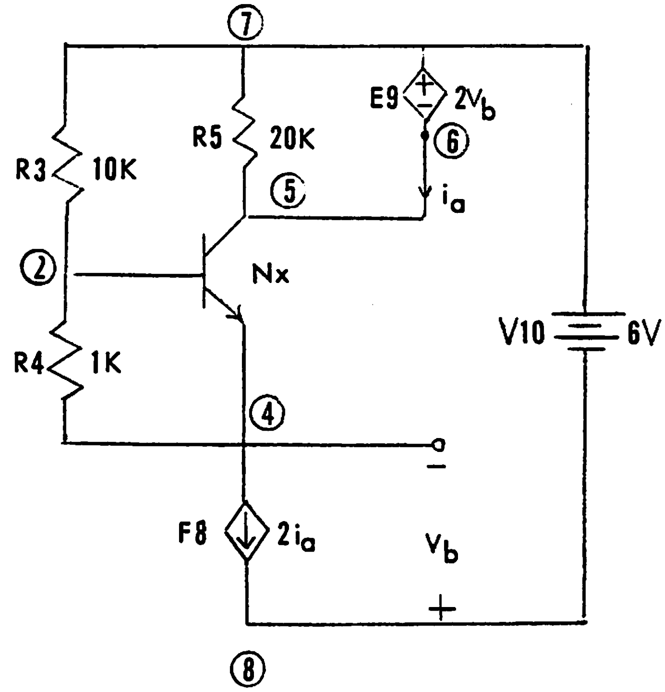
7
R3 10K
R5 20K
E9 2Vb
6
5
ia
2
Nx
V10 6V
R4 1K
4
F8 2ia
Vb
8

Fig.1

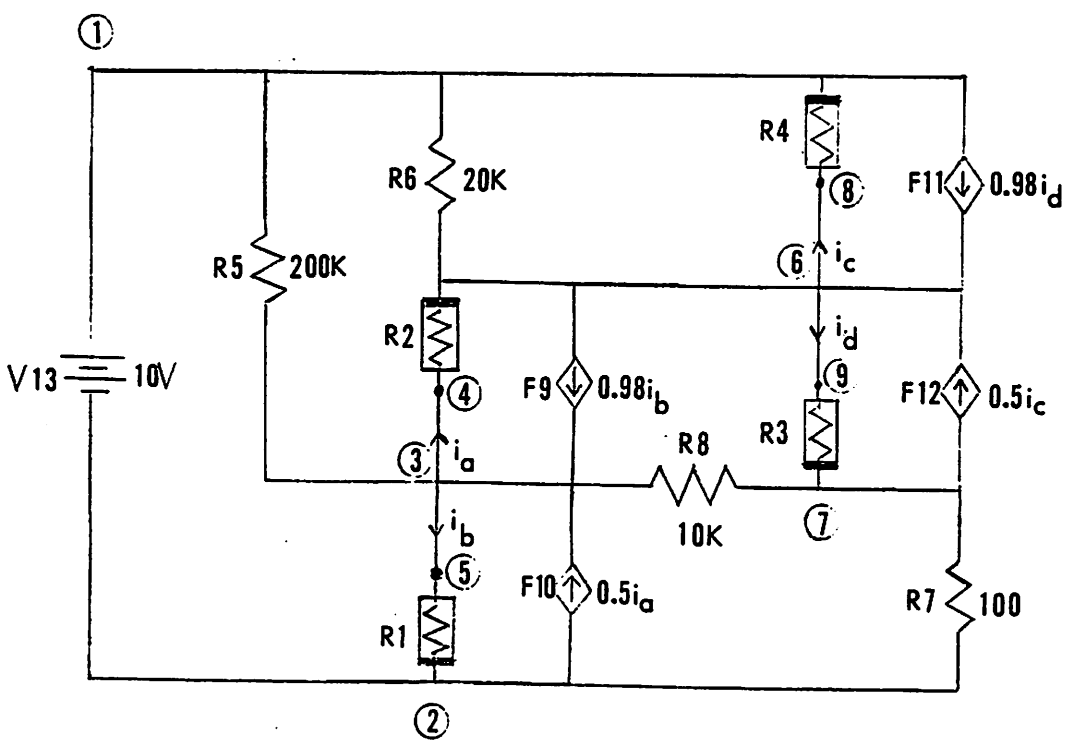
1
R4
8
F11 0.98id
R6 20K
R5 200K
6 ic
id
R2
9
V13 10V
4
F9 0.98ib
F12 0.5ic
R3
3 ia
R8
10K
7
ib
5
F10 0.5ia
R7 100
R1
2

Fig.2

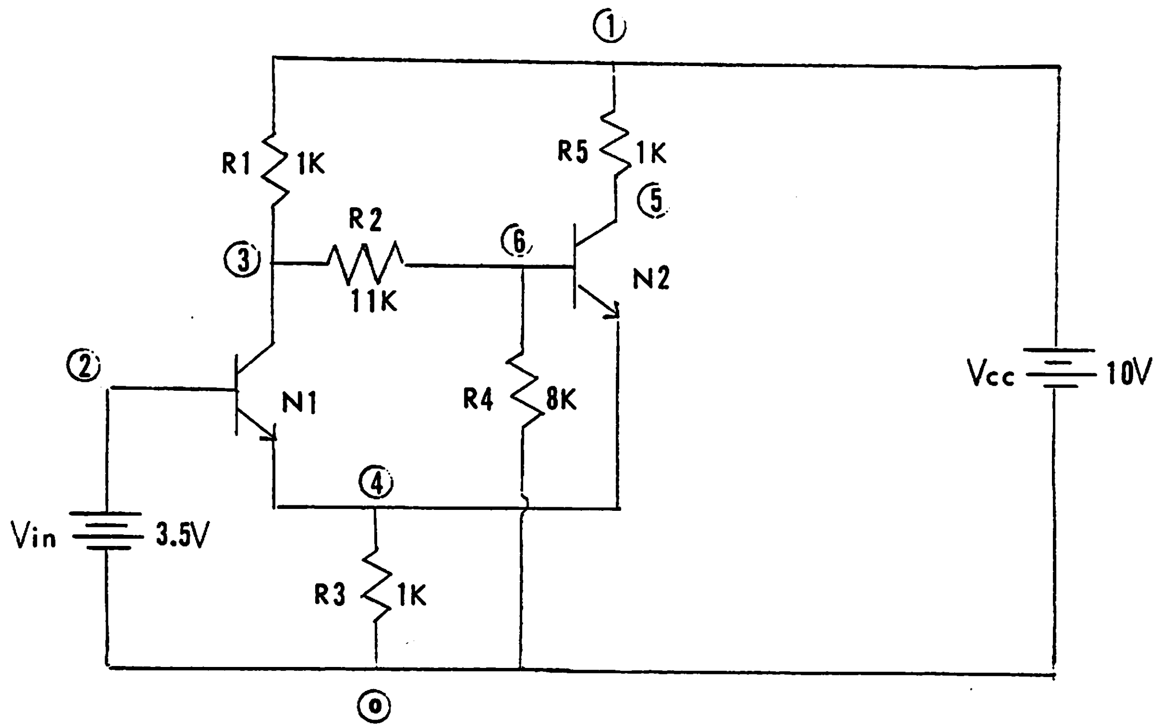

Fig.3

```
* Example 1
*
* single transistor type-S negative resistance device
*
* transistor is treated as a 2-port resistor
Nx 2 4 2 5 bpmod
*
* Ebers-Moll model of bipolar transistor
.model bpmod (i1=1.005e-14*exp(38.46*v1)-1.0e-14*exp(38.46*v2);
$i2=2.0e-14*exp(38.46*v2)-1.0e-14*exp(38.46*v1))
*
R3 2 7 10K
R4 2 4 1K
R5 7 5 20K
F8 4 8 6 5 2
E9 7 6 8 4 2
*
* driving voltage is fixed at 6 Volt
V10 7 8 6
*
* include the file with mathematical function declarations
.include "math.h"
.end
```

```
/********************SPICE    INPUT**********************
* Example 1
*
* single transistor type-S negative resistance device
*
* transistor is treated as a 2-port resistor
Nx 2 4 2 5 bpmod
*
* Ebers-Moll model of bipolar transistor
.model bpmod (i1=1.005e-14*exp(38.46*v1)-1.0e-14*exp(38.46*v2);
$i2=2.0e-14*exp(38.46*v2)-1.0e-14*exp(38.46*v1))
*
R3 2 7 10K
R4 2 4 1K
R5 7 5 20K
F8 4 8 6 5 2
E9 7 6 8 4 2
*
* driving voltage is fixed at 6 Volt
V10 7 8 6
*
* include the file with mathematical function declarations
.include "math.h"
.end
********************************************************


********************VARIABLE TABLE**********************
x[0]=v1(Nx)
x[1]=v2(Nx)
*******************************************************/

#include "math.h"


/******************************************************
******************************************************/

equation(f,x,x_dot,t)
double *f,*x,*x_dot,t;
{
    double y[2];

    y[0] = 1.005e-14*exp(38.46*x[0])-1.0e-14*exp(38.46*x[1]);
    y[1] = 2.0e-14*exp(38.46*x[1])-1.0e-14*exp(38.46*x[0]);
    f[0] = 4e-4*x[0]-3e-4*x[1]-y[1]-2.4e-3;
    f[1] = -1.2*x[0]-1e3*y[0]+1e-1*x[1]-1e3*y[1]+1.2;
}

/*******************************************************
*******************************************************/

jacob(x,x_dot,dx_dot,jf,t)
double *x,*x_dot,*dx_dot,*jf,t;
{
    jf[0] = 4.00e-04 + 3.8460e-13*exp(3.8460e+01*x[0]);
```

```
    jf[1] = -(3.00e-04 + 7.6920e-13*exp(3.8460e+01*x[1]));
    jf[2] = -(1.20e+00 + 1.9230e-12*exp(3.8460e+01*x[0]));
    jf[3] = 1.00e-01 - 3.8460e-10*exp(3.8460e+01*x[1]);
}

/*****************************************************
*****************************************************/

var_alloc(n,f,x,x_dot,dx_dot,jf)
int *n;
double **f,**x,**x_dot,**dx_dot,**jf;
{
    char *calloc();

    *n=2;
    *f=(double *)calloc(2,sizeof(double));
    *x=(double *)calloc(2,sizeof(double));
    *x_dot=(double *)calloc(2,sizeof(double));
    *dx_dot=(double *)calloc(2,sizeof(double));
    *jf=(double *)calloc(4,sizeof(double));
}
```

```
default initial guess? y/n
y
convergent solution ...
v1(Nx) = 3.750e-01
v2(Nx) = -7.500e+00
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(Nx)=0.65
v2(Nx)=-4
convergent solution ...
v1(Nx) = 6.562e-01
v2(Nx) = -4.080e+00
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(Nx)=0.69
v2(Nx)=0.65
convergent solution ...
v1(Nx) = 6.885e-01
v2(Nx) = 6.361e-01
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(Nx)=0.6
v2(Nx)=0.5
convergent solution ...
v1(Nx) = 6.562e-01
v2(Nx) = -4.080e+00
would you like to try another solution? y/n
n
```

```
*******************
v1(Nx)=3.750e-01
v2(Nx)=-7.500e+00
*******************
v1(Nx)=6.562e-01
v2(Nx)=-4.080e+00
*******************
v1(Nx)=6.885e-01
v2(Nx)=6.361e-01
&&&&&&&&&&&&&&&&&&&
```

```
* Example 2
*
* type-N 2-transistor negative resistance circuit
*
* each transistor is modelled by 2 pn-junction diodes and 2 CCCSs
* T1 transistor
R1 5 2 {i=1E-14*exp(38.46*v)}
R2 4 6 {i=1E-14*exp(38.46*v)}
F9 6 3 3 5 0.98
F10 2 3 3 4 0.5
*
* T2 transistor
R3 9 7 {i=1E-14*exp(38.46*v)}
R4 8 1 {i=1E-14*exp(38.46*v)}
F11 1 6 6 9 0.98
F12 7 6 6 8 0.5
*
* biasing resistors
R5 1 3 200K
R6 1 6 20K
R7 7 2 100
R8 3 7 10K
*
* driving voltage source is fixed at 10 Volts
V13 1 2 10
* include the file with mathematical function declaration
.include "math.h"
.end
```

```
/********************SPICE    INPUT**********************
* Example 2
*
* type-N 2-transistor negative resistance circuit
*
* each transistor is modelled by 2 pn-junction diodes and 2 CCCSs
* T1 transistor
R1 5 2 {i=1E-14*exp(38.46*v)}
R2 4 6 {i=1E-14*exp(38.46*v)}
F9 6 3 3 5 0.98
F10 2 3 3 4 0.5
*
* T2 transistor
R3 9 7 {i=1E-14*exp(38.46*v)}
R4 8 1 {i=1E-14*exp(38.46*v)}
F11 1 6 6 9 0.98
F12 7 6 6 8 0.5
*
* biasing resistors
R5 1 3 200K
R6 1 6 20K
R7 7 2 100
R8 3 7 10K
*
* driving voltage source is fixed at 10 Volts
V13 1 2 10
* include the file with mathematical function declaration
.include "math.h"
.end
*********************************************************


********************VARIABLE TABLE**********************
x[0]=v(R1)
x[1]=v(R2)
x[2]=v(R4)
x[3]=v(R3)
*********************************************************/

#include "math.h"


/*******************************************************
*******************************************************/

equation(f,x,x_dot,t)
double *f,*x,*x_dot,t;
{
    double y[4];

    y[0] = 1E-14*exp(38.46*x[0]);
    y[1] = 1E-14*exp(38.46*x[1]);
    y[2] = 1E-14*exp(38.46*x[2]);
    y[3] = 1E-14*exp(38.46*x[3]);
    f[0] = 1.96e4*y[0]-2e4*y[1]+x[2]+1e4*y[2]+4e2*y[3];
    f[1] = x[0]-x[1]-x[2]-1e1;
```

```
    f[2] = -x[0]+1.01e2*x[2]+5e3*y[2]-1.01e2*x[3]-1e4*y[3]+1.01e3;
    f[3] = -5e-6*x[0]-2e-2*y[0]-5e-1*y[1]-1e-2*x[2]-5e-1*y[2]+1e-2*x[3]+y[3]-9.995e-2;
}

/**********************************************************
**********************************************************/

jacob(x,x_dot,dx_dot,jf,t)
double *x,*x_dot,*dx_dot,*jf,t;
{
    jf[0] = 7.538160e-09*exp(3.8460e+01*x[0]);
    jf[1] = -7.6920e-09*exp(3.8460e+01*x[1]);
    jf[2] = 1.00e+00 + 3.8460e-09*exp(3.8460e+01*x[2]);
    jf[3] = 1.53840e-10*exp(3.8460e+01*x[3]);
    jf[4] = 1.00e+00;
    jf[5] = -1.00e+00;
    jf[6] = -1.00e+00;
    jf[7] = 0.00e+00;
    jf[8] = -1.00e+00;
    jf[9] = 0.00e+00;
    jf[10] = 1.010e+02 + 1.9230e-09*exp(3.8460e+01*x[2]);
    jf[11] = -(1.010e+02 + 3.8460e-09*exp(3.8460e+01*x[3]));
    jf[12] = -(5.00e-06 + 7.6920e-15*exp(3.8460e+01*x[0]));
    jf[13] = -1.9230e-13*exp(3.8460e+01*x[1]);
    jf[14] = -(1.00e-02 + 1.9230e-13*exp(3.8460e+01*x[2]));
    jf[15] = 1.00e-02 + 3.8460e-13*exp(3.8460e+01*x[3]);
}

/*******************************************************
*******************************************************/

var_alloc(n,f,x,x_dot,dx_dot,jf)
int *n;
double **f,**x,**x_dot,**dx_dot,**jf;
{
    char *calloc();

    *n=4;
    *f=(double *)calloc(4,sizeof(double));
    *x=(double *)calloc(4,sizeof(double));
    *x_dot=(double *)calloc(4,sizeof(double));
    *dx_dot=(double *)calloc(4,sizeof(double));
    *jf=(double *)calloc(16,sizeof(double));
}
```

```
default initial guess? y/n
y
not convergent; last iteration point ...
v(R1) = 2.347e-04
v(R2) = -4.648e-03
v(R4) = -3.832e-11
v(R3) = 4.880e-03

would you like to continue? y/n
y
default initial guess? y/n
n
enter the initial guess
v(R1)=0.62
v(R2)=-1
v(R4)=-5
v(R3)=0.6
convergent solution ...
v(R1) = 6.355e-01
v(R2) = -2.962e-01
v(R4) = -9.068e+00
v(R3) = 6.820e-01
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v(R1)=0.63
v(R2)=0
v(R4)=0
v(R3)=0.5
not convergent; last iteration point ...
v(R1) = 6.300e-01
v(R2) = -5.488e-03
v(R4) = 8.962e-04
v(R3) = 5.055e-01

would you like to continue? y/n
y
default initial guess? y/n
n
enter the initial guess
v(R1)=0.62
v(R2)=-0.1
v(R4)=-10
v(R3)=0.6
convergent solution ...
v(R1) = 6.355e-01
v(R2) = -2.962e-01
v(R4) = -9.068e+00
v(R3) = 6.820e-01
would you like to try another solution? y/n
n
```

```
******************
v(R1)=6.355e-01
v(R2)=-2.962e-01
v(R4)=-9.068e+00
v(R3)=6.820e-01
&&&&&&&&&&&&&&&&&&
```

```
* Example 3
*
* Schmitt-Trigger Circuit
*
* the transistors are treated as 2-port resistors
N1 2 4 2 3 bpmod
N2 6 4 6 5 bpmod
*
Vcc 1 0 10
R1 1 3 1K
R2 3 6 11K
R3 4 0 1K
R4 6 0 8K
R5 1 5 1K
*
Vin 2 0 3.5
*
* Ebers-Moll model of bipolar transistor
.model bpmod (i1=1.005e-14*exp(38.46*v1)-1.0e-14*exp(38.46*v2);
$i2=2.0e-14*exp(38.46*v2)-1.0e-14*exp(38.46*v1))
*
* include the file with mathematical function declarations
.include "math.h"
.end
```

```
/*********************SPICE    INPUT**********************
* Example 3
*
* Schmitt-Trigger Circuit
*
* the transistors are treated as 2-port resistors
N1 2 4 2 3 bpmod
N2 6 4 6 5 bpmod
*
Vcc 1 0 10
R1 1 3 1K
R2 3 6 11K
R3 4 0 1K
R4 6 0 8K
R5 1 5 1K
*
Vin 2 0 3.5
*
* Ebers-Moll model of bipolar transistor
.model bpmod (i1=1.005e-14*exp(38.46*v1)-1.0e-14*exp(38.46*v2);
$i2=2.0e-14*exp(38.46*v2)-1.0e-14*exp(38.46*v1))
*
* include the file with mathematical function declarations
.include "math.h"
.end
**********************************************************


********************VARIABLE TABLE**********************
x[0]=v1(N1)
x[1]=v2(N1)
x[2]=v1(N2)
x[3]=v2(N2)
*********************************************************/

#include "math.h"


/*******************************************************
*******************************************************/

equation(f,x,x_dot,t)
double *f,*x,*x_dot,t;
{
    double y[4];

    y[0] = 1.005e-14*exp(38.46*x[0])-1.0e-14*exp(38.46*x[1]);
    y[1] = 2.0e-14*exp(38.46*x[1])-1.0e-14*exp(38.46*x[0]);
    y[2] = 1.005e-14*exp(38.46*x[2])-1.0e-14*exp(38.46*x[3]);
    y[3] = 2.0e-14*exp(38.46*x[3])-1.0e-14*exp(38.46*x[2]);
    f[0] = x[0]+1e3*y[0]+1e3*y[2]-3.5;
    f[1] = x[0]+8*x[1]+8e3*y[1]-x[2]-8e3*y[2]-8e3*y[3]+4.85e1;
    f[2] = -x[0]+x[2]-x[3]-1e3*y[3]-6.5;
    f[3] = -x[0]+1.2e1*x[1]+1.1e4*y[1]+x[2]+7.15e1;
}
```

```
/**********************************************************
**********************************************************/

jacob(x,x_dot,dx_dot,jf,t)
double *x,*x_dot,*dx_dot,*jf,t;
{
    jf[0] = 1.00e+00 + 3.865230e-10*exp(3.8460e+01*x[0]);
    jf[1] = -3.8460e-10*exp(3.8460e+01*x[1]);
    jf[2] = 3.865230e-10*exp(3.8460e+01*x[2]);
    jf[3] = -3.8460e-10*exp(3.8460e+01*x[3]);
    jf[4] = 1.00e+00 - 3.07680e-09*exp(3.8460e+01*x[0]);
    jf[5] = 8.00e+00 + 6.15360e-09*exp(3.8460e+01*x[1]);
    jf[6] = -(1.00e+00 + 1.53840e-11*exp(3.8460e+01*x[2]));
    jf[7] = -3.07680e-09*exp(3.8460e+01*x[3]);
    jf[8] = -1.00e+00;
    jf[9] = 0.00e+00;
    jf[10] = 1.00e+00 + 3.8460e-10*exp(3.8460e+01*x[2]);
    jf[11] = -(1.00e+00 + 7.6920e-10*exp(3.8460e+01*x[3]));
    jf[12] = -(1.00e+00 + 4.23060e-09*exp(3.8460e+01*x[0]));
    jf[13] = 1.20e+01 + 8.46120e-09*exp(3.8460e+01*x[1]);
    jf[14] = 1.00e+00;
    jf[15] = 0.00e+00;
}

/**********************************************************
**********************************************************/

var_alloc(n,f,x,x_dot,dx_dot,jf)
int *n;
double **f,**x,**x_dot,**dx_dot,**jf;
{
    char *calloc();

    *n=4;
    *f=(double *)calloc(4,sizeof(double));
    *x=(double *)calloc(4,sizeof(double));
    *x_dot=(double *)calloc(4,sizeof(double));
    *dx_dot=(double *)calloc(4,sizeof(double));
    *jf=(double *)calloc(16,sizeof(double));
}
```

```
default initial guess? y/n
y
not convergent; last iteration point ...
v1(N1) = 1.709e-03
v2(N1) = -2.930e-03
v1(N2) = 1.953e-03
v2(N2) = -2.930e-03

would you like to continue? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(N1)=0.68
v2(N1)=-3.3
v1(N2)=0.3
v2(N2)=-6.8
convergent solution ...
v1(N1) = 6.853e-01
v2(N1) = -3.339e+00
v1(N2) = 6.508e-02
v2(N2) = -7.120e+00
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(N1)=0.67
v2(N1)=-4.2
v1(N2)=0.65
v2(N2)=-5.4
convergent solution ...
v1(N1) = 6.615e-01
v2(N1) = -4.932e+00
v1(N2) = 6.724e-01
v2(N2) = -4.785e+00
would you like to try another solution? y/n
y
default initial guess? y/n
n
enter the initial guess
v1(N1)=0.17
v2(N1)=-6
v1(N2)=0.68
v2(N2)=-2.6
convergent solution ...
v1(N1) = 2.662e-01
v2(N1) = -5.994e+00
v1(N2) = 6.890e-01
v2(N2) = -2.859e+00
would you like to try another solution? y/n
n
```

```
******************
v1(N1)=6.853e-01
v2(N1)=-3.339e+00
v1(N2)=6.508e-02
v2(N2)=-7.120e+00
******************
v1(N1)=6.615e-01
v2(N1)=-4.932e+00
v1(N2)=6.724e-01
v2(N2)=-4.785e+00
******************
v1(N1)=2.662e-01
v2(N1)=-5.994e+00
v1(N2)=6.890e-01
v2(N2)=-2.859e+00
&&&&&&&&&&&&&&&&&&&
```

APPENDIX

SOURCE CODE LISTINGS

```
#include <stdio.h>

/*****************************************************************************/
/* This is the main program of the routine "nrdc" which uses Newton-Raphson  */
/* iteration                                                                 */
/*                x_k+1 = x_k - inv(J(x_k))*f(x_k)                           */
/* to find the dc operating point(s) of a nonlinear resistive circuit. It    */
/* consists of three files : "nrdc.c", "nrdc1.c", "xx...x.c" (the equation   */
/* file), and the library routine "linpack.lib".                             */
/*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    FILE *fp,*hp;

    /* open the variable-table file */
    open_tbl_op(argc,argv,&fp,&hp);

    /* dc analysis */
    dc_simu(fp,hp);
}
```

```
#include <stdio.h>

char *var_name[10];
int n,ns;
double *x,*f,*jf,*xp[10],*dummy1,*dummy2;

/***************************************************************************/
/* Open the file containing the mapping table between the circuit variables*/
/* (voltages and currents) and the equation variables (independent        */
/* variables x and dependent variables y)                                  */
/***************************************************************************/

open_tbl_op(argc,argv,fp,hp)
int argc;
char *argv[];
FILE **fp,**hp;
{
    FILE *fopen();
    char line[12];

    if (argc != 2 )
        exit_message("NRDC OUTPUT_FILE");
    sprintf(line,"%s.tbl",*++argv);
    if ((*fp=fopen(line,"r"))==NULL)
    {
        printf("CAN'T OPEN THE TABLE FILE %s\n",line);
        exit();
    }
    sprintf(line,"%s.op",*argv);
    if ((*hp=fopen(line,"w"))==NULL)
    {
        printf("CAN'T OPEN OUTPUT DC_OP FILE %s\n",*argv);
        exit();
    }
}

/***************************************************************************/
/* DC simulation for DC operating point(s)                                 */
/***************************************************************************/

dc_simu(fp,hp)
FILE *fp,*hp;
{
    int iter=1;

    ns=0;
    /* allocate spaces for the variables used in */
    /* equation and Jacobian matrix routines     */
    var_alloc(&n,&f,&x,&dummy1,&dummy2,&jf);

    /* read the variable-table */
    get_tbl(fp);
    fclose(fp);

    while(iter==1)       /* while iteration has't converged */
    {
```

```
        /* get the initial guess */
        get_ini_gs();

        if (newton()==-1)        /* not convergent */
            no_cg(&iter);
        else                     /* convergent */

            /* print dc the operating point in the output file */
            dc_pt(&iter,hp);
    }
    fputs("&&&&&&&&&&&&&&&&&&&\n",hp);
    fclose(hp);
}

/******************************************************************************/
/* Get the initial guess for Newton-Raphson iteration; zero for default     */
/* initial guess.                                                             */
/******************************************************************************/

get_ini_gs()
{
    int i;
    char ch[2];

    printf("default initial guess? y/n\n");
    scanf("%1s",ch);
    if (ch[0]=='n')
    {
        printf("enter the initial guess\n");
        for (i=0;i<n;i++)
        {
            printf("%s=",var_name[i]);
            scanf("%lf",x+i);
        }
    } else
        for (i=0;i<n;i++) x[i]=0.0;
}

/******************************************************************************/
/* Print the computed dc operating point in the output file "xx...x.$op".  */
/******************************************************************************/

dc_pt(iter,hp)
int *iter;
FILE *hp;
{
    char ch[2],line[30];
    int i;

    /* print the dc operating point if it is a fresh solution */
    if (new_sol()==1)
    {
        fputs("******************\n",hp);
        for (i=0;i<n;i++)
        {
            sprintf(line,"%s=%.3e\n",var_name[i],x[i]);
```

```
            fputs(line,hp);
        }
    }

    printf("would you like to try another solution? y/n\n");
    scanf("%1s",ch);
    if (ch[0]=='n') *iter=0;
}

/****************************************************************************/
/* Check whether the convergent solution x differs any of the previously   */
/* found solution(s) xp in the output file "xx...x.$op". Return -1 if the  */
/* computed solution has been found before.                                */
/****************************************************************************/

new_sol()
{
    int i,k;
    double dif,fabs();

    printf("convergent solution ...\n");
    for (i=0;i<n;i++)
        printf("%s = %.3e\n",var_name[i],x[i]);

    /* check whether the solution x exists in the output file */
    for (k=0;k<ns;k++)
    {
        dif=0;
        for (i=0;i<n;i++)
            dif+=fabs(x[i]-xp[k][i]);
        if (dif<1.0e-6)
            return(-1);
    }

    if (ns>=10)
        exit_message("MORE THAN 10 DC OPERATING POINTS");
    else
        callocd(n,&xp[ns],"xp");

    /* store the convergent solution to the output file  and */
    /* increment the # of solution(s) in the output file     */
    for (i=0;i<n;i++)
        xp[ns][i]=x[i];
    ns++;

    return(1);
}

/****************************************************************************/
/* Print the last iteration point when Newton-Raphson iteration does not   */
/* converge.                                                               */
/****************************************************************************/

no_cg(iter)
int *iter;
{
```

```
    int i;
    char ch[2];

    printf("not convergent; last iteration point ...\n");
    for (i=0;i<n;i++)
        printf("%s = %.3e\n",var_name[i],x[i]);
    printf("\nwould you like to continue? y/n\n");
    scanf("%1s",ch);
    if (ch[0]=='n')
        *iter=0;
}

/******************************************************************************/
/* Newton-Raphson iteration for solving the nonlinear equation                */
/*                f(x) = 0                                                    */
/* Return -1 if the iteration is not convergent.                              */
/******************************************************************************/

newton()
{
    int k=0,*ipvt;
    double *zq,tx=0.0;
    double rcond,max,*z,fabs(),*ff;

    /* allocate spaces for the variables used in calling */
    /* Linpack routines sgeco and sgesl                  */
    alloc_1(&ipvt,&zq,&z,&ff);

    /* Newton-Raphson iteration */
    while (++k<20)                  /* limited to 20 iterations */
    {
        /* evaluate f(x) */
        equation(f,x,dummy1,tx);

        /* prevent overflow */
        if (pre_ovfl(k,&max,z,ff)==-1)
            return(-1);

        if (max<1.0e-7)             /* convergent */
            break;

        /* evaluate the Jacobian matrix jf */
        jacob(x,dummy1,dummy2,jf,tx);

        /* iteration formula : x_k+1 = x_k - inv(jf)*f(x_k) */
        sgeco(jf,n,ipvt,&rcond,zq);     /* LU decomposition for jf */
        if (fabs(rcond)<1.0e-16)        /* singular Jacobian matrix */
        {
            printf("SINGULAR JACOBIAN MATRIX\n");
            return(-1);
        } else
            next_iter(ipvt,z,ff);       /* find next iteration pt x_k+1 */
    }
    free_1(ipvt,zq,z,ff);
    if (k>=20) return(-1);
    else return(1);
```

```
}

/*****************************************************************************/
/* If f(x) numerically overflows, reduce the distance between sequential   */
/* iteration points to avoid overflow (especially due to exp function).    */
/*****************************************************************************/

pre_ovfl(k,max,z,ff)
int k;
double *max,*ff,*z;
{
    int i,j=0;
    double norm(),tx=0.0;

    /* reduce iteration distance if f(x))100 for k-th iteration with */
    /* k)1 (iteration distance |x_k - x_k-1| is undefined for k=1)   */
    while ((*max=norm(n,f))>100 && k>1)
    {
        for (i=0;i<n;i++)
        {
            ff[i]=0.5*ff[i];
            x[i]=z[i]-ff[i];
        }
        equation(f,x,dummy1,tx);        /* re-evaluate f(x) */
        if (++j>10)
            return(-1);
    }
    return(1);
}

/*****************************************************************************/
/* Find the next iteration point x_k+1 = x_k - inv(jf)*f(x_k) and save     */
/* the previous iteration point x_k and the iteration distance in case     */
/* the distance has to be reduced due to the overflow at the new iteration */
/* point x_k+1.                                                             */
/*****************************************************************************/

next_iter(ipvt,z,ff)
int *ipvt;
double *z,*ff;
{
    int i;

    sgesl(jf,n,ipvt,f,0);       /* find inv(jf)*f */
    for (i=0;i<n;i++)
    {
            z[i]=x[i];          /* save previous point  */
            ff[i]=f[i];         /* iteration distance   */
            x[i]=z[i]-f[i];     /* next iteration point */
    }
}

/*****************************************************************************/
/* Find the norm-1 of a vector                                              */
/*****************************************************************************/
```

```
double
norm(m,y)
int m;          /* vector dimension */
double y[];     /* vector */
{
    int i;
    double max=0.0,fabs();

    for (i=0;i<m;i++)
        max+=fabs(y[i]);
    return(max);
}

/*************************************************************************/
/* Read the file with the mapping table (between the circuit variables v,i */
/* and the equation variables x,y).                                        */
/*************************************************************************/

get_tbl(fp)
FILE *fp;
{
    int j,k;
    char ex[3],line[30],*calloc();

    while (fgets(line,30,fp)!=NULL)
    {
        /* get the variable index j from "x[j]=xxxx..xxx" */
        strdel(line,0,2);
        k=find_index("]=",line);
        strncpy(ex,line,k);
        j=atoi(ex);

        /* extract the name of circuit element; e.g., Rcc, Ri */
        k=find_index("=",line);
        strdel(line,0,k+1);
        line[strlen(line)-1]='\0';
        var_name[j]=calloc(10,sizeof(char));
        strcpy(var_name[j],line);
    }
}

/*************************************************************************/
/* Allocate spaces for the variables used in Linpack routines sgeco and  */
/* sgesl.                                                                */
/*************************************************************************/

alloc_1(ipvt,zq,z,ff)
int **ipvt;
double **zq,**z,**ff;
{
    char *calloc();

    *ipvt=(int *)calloc(n,sizeof(int));
    *zq=(double *)calloc(n,sizeof(double));
    *z=(double *)calloc(n,sizeof(double));
    *ff=(double *)calloc(n,sizeof(double));
```

```
}

/******************************************************************************/
/* Free the spaces for the variables used in Linpack routines sgeco and     */
/* sgesl when finishing Newton-Raphson iteration.                             */
/******************************************************************************/

free_1(ipvt,zq,z,ff)
int *ipvt;
double *zq,*z,*ff;
{
    cfree(ipvt);
    cfree(zq);
    cfree(z);
    cfree(ff);
}
```