NONLINEAR ELECTRONICS (NOEL) PACKAGE 1: LINEAR CIRCUIT FORMULATIONS, N-PORT REPRESENTATIONS, AND STATE EQUATIONS

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/23

13 March 1986

NONLINEAR ELECTRONICS (NOEL) PACKAGE 1: LINEAR CIRCUIT FORMULATIONS, N-PORT REPRESENTATIONS, AND STATE EQUATIONS

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/23

13 March 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

NONLINEAR ELECTRONICS (NOEL) PACKAGE 1: LINEAR CIRCUIT FORMULATIONS, N-PORT REPRESENTATIONS, AND STATE EQUATIONS

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/23

13 March 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

NOEL PACKAGE 1 : LINEAR CIRCUIT FORMULATIONS, N-PORT REPRESENTATIONS AND STATE EQUATIONS†

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 purpose of the programs in this package is to find the characterizations of a linear resistive n-port and a linear dynamic m-port, and to formulate the state equation of a linear dynamic circuit. NOEL will give a ***generalized implicit representation***

$$\mathbf{Pv} + \mathbf{Qi} + \mathbf{s} = 0$$

for the linear resistive n-port, and a ***complex generalized implicit representation*** at frequency $\omega = 2\pi f$

$$\hat{\mathbf{P}}(j\omega)^* \mathbf{v}(j\omega) + \hat{\mathbf{Q}}(j\omega)^* \mathbf{i}(j\omega) + \hat{\mathbf{s}}(j\omega) = 0$$

for the linear dynamic m-port. Various types of explicit representations (impedance, admittance, hybrid, transmission, and scattering) can also be obtained if requested by the user. The linear state equation

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{s}_1 + \mathbf{T}_1\dot{\mathbf{u}}$$

$$\mathbf{y} = \mathbf{Cx} + \mathbf{Du} + \mathbf{s}_2 + \mathbf{T}_2\dot{\mathbf{u}}$$

is given in terms of the parameters $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{T}_1$, and $\mathbf{T}_2$.

March 13, 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 1 :
LINEAR CIRCUIT FORMULATIONS, N-PORT REPRESENTATIONS AND STATE EQUATIONS

1. Introduction

Consider the circuit consisting of an arbitrary interconnection of the circuit elements within the class of allowed elements defined in [2]. The circuit is described by an input file "xx...x.spc" and follows the rules of the input format language defined in [2]. A linear resistive n-port circuit is formed by extracting all the nonlinear and/or dynamic and/or time-varying elements from the circuit, such that it consists of the remaining linear elements or time-invariant independent voltage or current sources as shown in Fig.1. More specifically, the internal part of the linear resistive n-port is made up of :

(1) time-invariant independent voltage sources

(2) time-invariant independent current sources

(3) 2-terminal linear resistors

(4) 4 types of linear controlled sources (VCVS, CCCS, VCCS, CCVS)

and each of the external ports is connected by one of the following elements :

(1) 2-terminal nonlinear resistor

(2) 2-terminal piecewise-linear resistor

(3) 2-terminal capacitor (linear or nonlinear)

(4) 2-terminal inductor (linear or nonlinear)

(5) time-varying voltage source

(6) time-varying current source

(7) nonlinear controlled source

(8) nonlinear 2-port or 3-terminal resistor

(9) piecewise-linear 2-port or 3-terminal resistor

The port number n is determined by the total number of the above class of circuit elements in the circuit; namely, nonlinear/dynamic/time-varying elements.

Since all the internal elements, except the independent sources, are linear and memoryless, the resistive n-port can be characterized by a conductance representation

$$\mathbf{i} = \mathbf{G}\mathbf{v} + \mathbf{s} \tag{1}$$

a resistance representation

$$\mathbf{v} = \mathbf{R}\mathbf{i} + \mathbf{s} \tag{2}$$

or a hybrid representation

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{s} \tag{3}$$

where the constant vector $\mathbf{s}$ is contributed by the independent sources. However, the conductance, resistance, and some particular combination of hybrid representations may not exist in the general case. For example, the conductance representation does not exist if some of the port voltages form a loop. The non-existence of the above explicit representations, Eqs.(1), (2), and (3), will impose limitations for some circuit analysis algorithms which require an explicit relation among the port variables. Given this defect of the explicit representations, we are motivated to discuss only the *generalized implicit representation* for the linear resistive n-port; namely,

$$\mathbf{P}\mathbf{v} + \mathbf{Q}\mathbf{i} + \mathbf{s} = 0 \tag{4}$$

where $\mathbf{P} \in \mathbf{R}^{n \times n}$, $\mathbf{Q} \in \mathbf{R}^{n \times n}$ and $\mathbf{s} \in \mathbf{R}^{n}$.

This representation exists for most circuits except for some very pathologically degenerate circuits[1], and can be used to obtain any other representation, when it exists. (See Eqs.(7)-(13) below.)

Combining the frequency-domain characteristics

$$v_L(j\omega) = j\omega L * i_L(j\omega), \quad i_C(j\omega) = j\omega C * v_C(j\omega) \tag{5}$$

of the linear inductors or capacitors with the linear resistive n-port equation (4), we can further expand the resistive n-port to a dynamic m-port, as shown in Fig.2, but with reduced port dimension m, where n-m totals the number of linear dynamic elements. The dynamic m-port is characterized by an implicit relation, called *complex generalized implicit representation*

$$\hat{\mathbf{P}}(j\omega) * \mathbf{v}(j\omega) + \hat{\mathbf{Q}}(j\omega) * \mathbf{i}(j\omega) + \hat{\mathbf{s}}(j\omega) = 0 \tag{6}$$

at frequency $\omega = 2\pi f$, where $\hat{\mathbf{P}} \in \mathbf{C}^{m \times m}$, $\hat{\mathbf{Q}} \in \mathbf{C}^{m \times m}$, and $\hat{\mathbf{s}} \in \mathbf{C}^{m}$.

Several explicit m-port representations can be easily obtained by taking the matrix inverse operations; namely,

the impedance representation

$$\mathbf{v}(j\omega) = \mathbf{Z}(j\omega) * \mathbf{i}(j\omega) + \mathbf{c}(j\omega) \tag{7}$$

the admittance representation

$$\mathbf{i}(j\omega) = \mathbf{Y}(j\omega) * \mathbf{v}(j\omega) + \mathbf{c}(j\omega) \tag{8}$$

the hybrid I representation

$$\begin{bmatrix} \mathbf{v}_a \\ \mathbf{i}_b \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{aa} & \mathbf{H}_{ab} \\ \mathbf{H}_{ba} & \mathbf{H}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{i}_a \\ \mathbf{v}_b \end{bmatrix} + \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_b \end{bmatrix} \tag{9}$$

the hybrid II representation

$$\begin{bmatrix} \mathbf{i}_a \\ \mathbf{v}_b \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{aa} & \mathbf{H}_{ab} \\ \mathbf{H}_{ba} & \mathbf{H}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{v}_a \\ \mathbf{i}_b \end{bmatrix} + \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_b \end{bmatrix} \tag{10}$$

the transmission representation

$$\begin{bmatrix} \mathbf{v}_a \\ \mathbf{i}_a \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{aa} & \mathbf{H}_{ab} \\ \mathbf{H}_{ba} & \mathbf{H}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{v}_b \\ \mathbf{i}_b \end{bmatrix} + \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_b \end{bmatrix} \tag{11}$$

the scattering representation

$$\mathbf{y}(j\omega) = \mathbf{S}(j\omega) * \mathbf{x}(j\omega) + \mathbf{c}(j\omega) \tag{12}$$

the generalized representation

$$\xi(j\omega) = \mathbf{G}(j\omega) * \eta(j\omega) + \mathbf{c}(j\omega) \tag{13}$$

where

$$\begin{bmatrix} \mathbf{v} \\ \mathbf{i} \end{bmatrix} = \begin{bmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} \tag{14}$$

Substituting the time-domain characteristics

$$v_L(t) = L\frac{di_L}{dt}, \quad i_C(t) = C\frac{dv_C}{dt} \tag{15}$$

of the linear inductors or capacitors into Eq.(4), we can easily obtain the linear implicit state equation

$$\begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{21} & \mathbf{P}_{22} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}(t) \\ \mathbf{v}(t) \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{i}(t) \end{bmatrix} + \begin{bmatrix} \mathbf{s}_1(t) \\ \mathbf{s}_2(t) \end{bmatrix} = 0 \tag{16}$$

and the linear explicit state equation

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{s}_1 + \mathbf{T}_1\dot{\mathbf{u}} \tag{17a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{s}_2 + \mathbf{T}_2\dot{\mathbf{u}} \tag{17b}$$

for a linear dynamic circuit, where

$$\mathbf{A}\in\mathbf{R}^{n\times n}\ ,\ \mathbf{B}\in\mathbf{R}^{n\times r}\ ,\ \mathbf{C}\in\mathbf{R}^{r\times n}\ ,\ \mathbf{D}\in\mathbf{R}^{r\times r}\ ,\ \mathbf{s}_1\in\mathbf{R}^{n}\ ,\ \mathbf{s}_2\in\mathbf{R}^{r}\ ,\ \mathbf{T}_1\in\mathbf{R}^{n\times r}\ ,\ \mathbf{T}_2\in\mathbf{R}^{r\times r}$$

and $\mathbf{x}\in\mathbf{R}^n$ consists of linear capacitor voltages and linear inductor currents, $\mathbf{u}\in\mathbf{R}^r$ consists of the voltages (resp.; currents) of time-varying voltage sources (resp.; current sources), $\mathbf{y}\in\mathbf{R}^r$ consists of the currents (resp.; voltages) of time-varying voltage sources (resp.; current sources). The matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ are zero when there exists no loop of capacitors and time-varying sources, or cutset of inductors and time-varying current sources.

In addition to the above representations for linear circuits, Eq.(4) is also a preliminary step for nonlinear circuit formulations :

Combining the characteristics of the elements connected across the external ports with the generalized implicit equation (4), the equations describing the circuit can be formulated as a nonlinear algebraic equation

$$\mathbf{f}(\mathbf{x}) = 0 \tag{18}$$

for a resistive circuit, or an implicit differential equation

$$\mathbf{f}(\mathbf{x}\ ,\ \dot{\mathbf{x}}\ ,\ t\ ) = 0 \tag{19}$$

for a dynamic circuit, and can be solved by nonlinear DC analysis[3], nonlinear transient analysis[4], canonical piecewise-linear DC analysis[5], and canonical piecewise-linear transient analysis[6].

2. Algorithm

2.1. Linear resistive n-port

The algorithm for finding the generalized implicit equation (4) is summarized in the following sequential steps. For details of the algorithm, refer to Section 6.6, pp.259-265 of [1].

Step 1.

Find the reduced incidence matrix **A**.

Step 2.

Choose a tree T by reducing **A** to

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{I} & \mathbf{D} \end{bmatrix}$$

Step 3.

Divide the branches into four categories :

(i) port branches in the tree

(ii) port branches in the cotree

(iii) nonport branches in the tree

(iv) nonport branches in the cotree

such that the nonport branches are either linear resistive elements or time-invariant independent sources.

Step 4.

Find the KCL (Eq.(6.55) in [1]) and KVL (Eq.(6.56) in [1]).

Step 5.

Find the linear characterization (Eq.(6.65) in [1]) for nonport branches.

Step 6.

Combine KCL, KVL, and the characterization of nonport branches into the tableau form (Eq.(6.66) in [1]).

Step 7.

Substitute the nonport tree currents (resp.; nonport link voltages) in terms of link currents (resp.; tree voltages) into the tableau equation to yield a simpler form (Eq.(6.67) in [1])

Step 8.

Apply elementary row operations to reduce the simplified tableau equation (Eq.(6.67) in [1]) to the row-echelon form (Eq.(6.69) in [1]); and then extract the constraint matrix equation

$$\mathbf{P}' \mathbf{v}' + \mathbf{Q}' \mathbf{i}' = 0$$

from the right bottom corner, where $\mathbf{v}'$ (resp.; $\mathbf{i}'$) denotes the voltages (resp.; currents) of the independent sources and the port elements (nonlinear and/or dynamic and/or time-varying elements).

Step 9.

Substitute the independent voltage or current source values into the constraint matrix equation to yield the generalized implicit equation (4).

2.2. Linear dynamic m-port

Step 1.

Follow the algorithm of Section 2.1 to find the generalized implicit equation (4) for the linear resistive n-port.

Step 2.

Find the linear implicit state equation (16) by substituting the linear time-domain relations Eq.(15) of inductor or capacitor into Eq.(4), where $\mathbf{x}$ consists of the linear capacitor voltages and the linear inductor currents.

Step 3.

Solve the upper part of Eq.(16) in the frequency domain to get

$$\mathbf{x}(j\omega) = -(\mathbf{rp}_{12} + j\,\mathbf{ip}_{12})\mathbf{v} - (\mathbf{rq}_{12} + j\,\mathbf{iq}_{12})\mathbf{i} - (\mathbf{rs}_1 + j\,\mathbf{is}_1) \tag{20}$$

where

$$\mathbf{rp}_{12} + j\,\mathbf{ip}_{12} = (j\omega\mathbf{P}_{11} + \mathbf{Q}_{11})^{-1} * \mathbf{P}_{12} \tag{21a}$$

$$\mathbf{rq}_{12} + j\,\mathbf{iq}_{12} = (j\omega\mathbf{P}_{11} + \mathbf{Q}_{11})^{-1} * \mathbf{Q}_{12} \tag{21b}$$

$$\mathbf{rs}_1 + j\,\mathbf{is}_1 = (j\omega\mathbf{P}_{11} + \mathbf{Q}_{11})^{-1} * \mathbf{s}_1 \tag{21c}$$

Step 4.

Substitute Eq.(20) into the bottom part of Eq.(16) to get the complex generalized implicit equation (6), where

$$\hat{\mathbf{P}}(j\omega) = (\mathbf{P}_{22} + \omega\mathbf{P}_{21}\mathbf{ip}_{12} - \mathbf{Q}_{21}\mathbf{rp}_{12}) + j\,(-\omega\mathbf{P}_{21}\mathbf{rp}_{12} - \mathbf{Q}_{21}\mathbf{ip}_{12}) \tag{22a}$$

$$\hat{\mathbf{Q}}(j\omega) = (\mathbf{Q}_{22} + \omega\mathbf{P}_{21}\mathbf{iq}_{12} - \mathbf{Q}_{21}\mathbf{rq}_{12}) + j\,(-\omega\mathbf{P}_{21}\mathbf{rq}_{12} - \mathbf{Q}_{21}\mathbf{iq}_{12}) \tag{22b}$$

$$\hat{\mathbf{s}}(j\omega) = (\mathbf{s}_2 + j\,\omega\mathbf{P}_{21}\mathbf{is}_1 - \mathbf{Q}_{21}\mathbf{rs}_1) + j\,(-\omega\mathbf{P}_{21}\mathbf{rs}_1 - \mathbf{Q}_{21}\mathbf{is}_1) \tag{22c}$$

Step 5.

Re-arrange the port indices or interchange the port variables, and then perform the matrix inverse operation to reduce the implicit equation (6) to various types of explicit representations Eqs.(7)-(13).

2.3. Linear state equation

Step 1.

Follow Steps 1 and 2 of Section 2.2 to find the implicit state equation (16).

Step 2.

Re-arrange Eq.(16) to

$$\begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12}' \\ \mathbf{P}_{21} & \mathbf{P}_{22}' \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{y} \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12}' \\ \mathbf{Q}_{21} & \mathbf{Q}_{22}' \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} + \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} = 0 \tag{23}$$

such that $\mathbf{u}$ (resp.; $\mathbf{y}$) consists of the input variables (resp.; output variables).

Step 3.

Perform the matrix inverse operation to reduce Eq.(23) to the explicit state equation (17).

3. User's Instruction

3.1. Linear resistive n-port

Step 1.

Create a file "xx...x.spc" which describes the 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. All the circuit elements given in [2] can be included in "xx...x.spc"; namely,

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

'C' : 2-terminal capacitor (linear or nonlinear)

'L' : 2-terminal inductor (linear or nonlinear)

'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

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

Step 2.

Type the command

nport xx...x

for standard output from the screen or

nport xx...x >yy...y

if the output is to be redirected to the file "yy...y", where "xx...x" is the filename for the input file "xx...x.spc" which contains the circuit description.

3.2. Linear dynamic m-port

Step 1.

Same as Step 1 of Section 3.1 except the nonlinear elements (2-terminal or 2-port) are not included. The linear dynamic m-port characterization for a nonlinear circuit can be done by replacing each nonlinear element with an independent time-varying source, and both circuits exhibit the same m-port characterization.

Step 2. (complex generalized implicit equation (6))

Type the command

cnport xx...x

to obtain the complex generalized implicit equation (6). It proceeds interactively with the following procedures :

enter the frequency

After the frequency $f = \omega/2\pi$ is entered, the complex generalized implicit equation (6) is printed and a message is shown to ask the user whether to continue :

continue with a new frequency? y/n

If 'y' then repeat the procedures; else stop.

Step 3. (complex explicit representations Eqs.(7)-(13))

Type the command

```
cnportx xx...x
```

to obtain various types of explicit representations. It proceeds interactively in the following procedures :

```
enter the representation type number
*****************************************************************
1 : impedance matrix representation
2 : admittance matrix representation
3 : hybrid I matrix representation
4 : hybrid II matrix representation
5 : transmission matrix representation
6 : scattering matrix representation
7 : generalized matrix representation
*****************************************************************
```

(i) type 1 : impedance matrix representation

If '1', then the impedance matrix representation Eq.(7) at the given frequency is found and will repeat at other frequencies if the user wishes to continue :

```
enter the frequency
continue with a new frequency? y/n
```

(ii) type 2 : admittance matrix representation

If '2', then the admittance matrix representation Eq.(8) at the given frequency is found which proceeds similarly as case (i).

(iii) types 3,4 : hybrid matrix representation

If '3' or '4', then the hybrid matrix representation is found by the following procedures :

```
enter the frequency
is port j (NAME) a v-controlled or i-controlled port? v/i
```

Enter 'v' (resp.; 'i') if the voltage (resp.; current) of the j-th port is the input variable in that port, where NAME is the name of the element connected across port j. After the v, i controlling information for each port is entered, the explicit hybrid representation is re-arranged to Eq.(9) for hybrid I representation and Eq.(10) for hybrid II representation.

(iv) type 5 : transmission matrix representation

```
enter the frequency
is port j (NAME) an input port? y/n
```

enter 'y' if both the voltage and current of the j-th port are the input variables.

(v) type 6 : scattering matrix representation

```
enter the frequency
enter R_j for port j (NAME)
```

Enter the reference resistance for the j-th port.

(vi) type 7 : generalized matrix representation

enter the frequency

enter row j of the n by n matrix Re(a)

enter row j of the n by n matrix Im(a)

enter row j of the n by n matrix Re(b)

enter row j of the n by n matrix Im(b)

enter row j of the n by n matrix Re(c)

enter row j of the n by n matrix Im(c)

enter row j of the n by n matrix Re(d)

enter row j of the n by n matrix Im(d)

Enter the coordinate transformation matrices **a**, **b**, **c**, and **d** of Eq.(14).

Step 4.

In addition to the standard output from the screen, both the implicit and the explicit representations at each given frequency are written to the output file "xx...x.eq", where "xx...x.spc" is the input file with the circuit description.

3.3. Linear state equation

Step 1.

Same as Step 1 of Section 3.2.

Step 2.

Type the command

staeq xx...x

for standard output from the screen or

staeq xx...x >yy...y

if the output is to be redirected to the file "yy...y", where "xx...x" is the filename for the input file "xx...x.spc".

4. Output Format

The output format for

(i) the generalized implicit equation (4) of the linear resistive n-port

(ii) the complex generalized implicit equation (6) of the linear dynamic m-port at a given frequency

(iii) the explicit representations Eqs.(7)-(13) of the linear dynamic m-port at a given frequency

(iv) the linear state equation (17)

are given by

(i) the **P** , **Q** matrices and the **s** vector

(ii) the $\hat{\mathbf{P}}(j\omega)$, $\hat{\mathbf{Q}}(j\omega)$ matrices and $\hat{\mathbf{s}}(j\omega)$ vector at the given frequency ω

(iii) the $\mathbf{c}(j\omega)$ vector and the $\mathbf{Z}(j\omega)$, $\mathbf{Y}(j\omega)$, $\mathbf{H}(j\omega)$, $\mathbf{T}(j\omega)$, $\mathbf{S}(j\omega)$, and $\mathbf{G}(j\omega)$ at the given frequency ω

(iv) the matrices **A** , **B** , **C** , **D** , $\mathbf{T}_1$, $\mathbf{T}_2$ and the vectors $\mathbf{s}_1$, $\mathbf{s}_2$ respectively. Each port is identified by the node numbers of the port terminals (with the convention that the first node number denotes the terminal with positive polarity) and the external element connected across this port. The standard output format for each type is shown below :

4.1. Linear resistive n-port

```
***************** SPICE  INPUT *****************

        { input file listing }

**************************************************

GENERALIZED IMPLICIT EQUATION OF THE
      LINEAR RESISTIVE n-PORT

          P*v + Q*i + s = 0

        { port definition }


******************** P matrix **********************

        { P[i,j] = ...... }

******************** Q matrix **********************

        { Q[i,j] = ...... }

********************** s vector ********************

        {  s[i]  = ...... }
```

4.2. Linear dynamic m-port

A. Implicit representation

********** SPICE INPUT **********

{input file listing}

****

COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
LINEAR DYNAMIC m-PORT

{port definition}

AT FREQUENCY = fff...ff
********** P MATRIX **********

{ Re(P[i,j]) = ... Im(P[i,j]) = ... }

********** Q MATRIX **********

{ Re(Q[i,j]) = ... Im(Q[i,j]) = ... }

********** s VECTOR **********

{ Re(s[i]) = ... Im(s[i]) = ...}

B. Explicit representation

********** SPICE INPUT **********

{ input file listing }

****

EXPLICIT LINEAR DYNAMIC m-PORT REPRESENTATION

{ port definition }

AT FREQUENCY = fff..ff
********** G MATRIX **********

{ Re(G[i,j]) = ... Im(G[i,j]) = ... }

********** c VECTOR **********

{ Re(c[i]) = ... Im(c[i]) = ... }

4.3. Linear state equation

```
***************** SPICE INPUT *****************

      { input file listing }

************************************************

******** LINEAR STATE EQUATION ********

            x' = A*x + B*u + s_1 + T_1*u'
            y  = C*x + D*u + s_2 + T_2*u'

***** state variables *****

    { state variable definitions }

***** input variables *****

    { input variable definitions }

***** output variables *****
    { output variable definitions }


***************** A MATRIX *****************

            { A[i,j] = ..... }

***************** B MATRIX *****************

            { B[i,j] = ..... }

***************** C MATRIX *****************

            { C[i,j] = ..... }


***************** D MATRIX *****************

            { D[i,j] = ..... }
**************** s_1 VECTOR ****************

            { s_1[i] = ..... }

**************** s_2 VECTOR ****************

            { s_2[i] = ..... }
```

********** T_1 MATRIX **********

{ T_1[i,j] = ... }

********** T_2 MATRIX **********

{ T_2[i,j] = ... }

Remark :

The port current direction in each of the explicit representations Eqs.(7)-(13) follows the conventional direction; namely, current flows into the n-port, or from the negative node to the positive node in each port. The direction, however, is opposite in the implicit representations Eqs.(4) and (6); namely, it flows from the positive node to the negative node. The unconventional direction in the implicit representations provides the convenient compatibility when combining the characteristic of each external-port element, whose current direction is from the positive node to the negative node, with the implicit linear n-port equation to formulate Eqs.(18) and (19).

5. Examples

5.1. Linear resistive n-port

Example 1 : in file "ex1.spc"

A single transistor type-S negative resistance circuit[7] (Fig.3).

Example 2 : in file "ex2.spc"

Schmitt-Trigger circuit (Fig.4).

Example 3 : in file "ex3.spc"

A dynamic circuit with a piecewise-linear resistor (Fig.5).

Example 4 : in file "ex4.spc"

A single transistor amplifier (Fig.6).

5.2. Linear dynamic m-port

Example 5 : in file "ex5.spc"

A linear RLC circuit (Fig.7).

Example 6 : in file "ex6.spc"

A linear LC ladder circuit (Fig.8).

5.3. Linear state equation

Example 7 : in file "ex7.spc"

A linear dynamic circuit with capacitor loop (Fig.9).

6. Diagnosis

6.1. Linear resistive n-port

1. NPORT SPICE_FILE

   Bad command line, the correct one should be

   nport xx...x

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

2. CAN'T OPEN SPICE_FILE xx...x.spc

   The input file "xx...x.spc" does not exist in the current directory.

3. EMPTY INPUT FILE

   The input file "xx...x.spc" contains no circuit description.

4. CAN'T ALLOCATE SPACE

   Not enough disk space for allocation of variables.

5. INSUFFICIENT SPACE FOR ELEMENT LINE ARRAY

   Number of element lines in the input file exceeds the given upper bound.

6. INSUFFICIENT SPACE FOR MODEL LINE ARRAY

   Number of model/graphic control/declaration lines (lines starting with '.') in the input file exceeds the given upper bound.

7. MISSING .END IN THE LAST LINE

   The input file is not terminated by ".end" or ".END".

8. UNDEFINED ELEMENT TYPE 'x'

   The first character 'x' of an element name is not in the class of allowed elements defined in [2].

9. UNDEFINED MODEL "mm...m"

   The model name "mm...m" is not defined in a separate model line

   .model mm...m {.....}

10. MISSING '{' IN THE MODEL LINE

    The model description in a model line is not included within the brackets "{.....}"

11. ZERO ROW IN THE INCIDENCE MATRIX

    The incidence matrix has a zero row due to bad circuit connections.

12. MISMATCHED CONTROLLED SOURCE

    Bad circuit connections for linear controlled sources.

13. DEGENERATE RESISTIVE N-PORT

The generalized implicit equation (4) does not exist.

14. WARNING MESSAGE : UNDERFLOW PROBLEM FOR nn...n

A very small number 'nn...n' (less than 1E-14) appears in the equation parameters due to numerical underflow in the intermediate calculations. This number will be reset to zero.

6.2. Linear dynamic m-port

1. See messages 2-14 of Section 6.1.

2. CNPORT SPICE_FILE

Bad command line for finding the complex generalized implicit equation; the correct one should be

cnport xx...x

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

3. CAN'T OPEN THE OUTPUT EQUATION FILE xx...x.eq

Unable to open the output file "xx...x.eq".

4. RESISTIVE N-PORT; SHOULD USE "nport xx...x"

Use resistive n-port formulation routine for the resistive circuit.

5. THE COMPLEX IMPLICIT EQUATION DOESN'T EXIST

Can't find the complex generalized implicit equation (6) due to inverse operation on the singular matrix in Eq.(21).

6. CNPORTX SPICE_FILE

Bad command line for finding the explicit dynamic m-port representations Eqs.(7)-(13).

7. THE IMPEDANCE REPRESENTATION DOESN'T EXIST
8. THE ADMITTANCE REPRESENTATION DOESN'T EXIST
9. THE HYBRID REPRESENTATION DOESN'T EXIST
10. THE TRANSMISSION REPRESENTATION DOESN'T EXIST
11. THE SCATTERING REPRESENTATION DOESN'T EXIST
12. THE GENERALIZED REPRESENTATION DOESN'T EXIST

Messages 7-12 are due to inverse operation on singular matrices in reducing the implicit representation to various types of explicit representation.

13. ODD PORT NUMBER

The transmission representation does not exist for a linear n-port, where n is an odd number.

6.3. Linear state equation

1. See messages 2-14 of Section 6.1.

2. STAEQ SPICE_FILE

Bad command line for finding the linear state equation.

3. ILLEGAL CIRCUIT ELEMENT TYPE IN LINEAR DYNAMIC CIRCUIT

   Try to find the linear state equation for a circuit containing nonlinear elements.

4. WARNING MESSAGE : CAPACITOR LOOP OR INDUCTOR CURRENT

   Capacitor loop or inductor cutset exists in the circuit such that $\mathbf{T}_1$ or $\mathbf{T}_2$ will be nonzero in Eq.(17).

References

[1] L.O.Chua and P.M.Lin, *Computer Aided Analysis of Electronic Circuits : Algorithms and Computational Techniques* , Englewood Cliffs, NJ : Prentice-Hall, 1975.

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

[3] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 3 : nonlinear DC analysis," ERL Memo. UCB/ERL M86, University of California, Berkeley, 1986.

[4] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 4 : nonlinear transient analysis," ERL Memo. UCB/ERL M86, University of California, Berkeley, 1986.

[5] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 6 : canonical piecewise-linear DC analysis," ERL Memo. UCB/ERL M86, University of California, Berkeley, 1986.

[6] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 7 : canonical piecewise-linear transient analysis," ERL Memo. UCB/ERL M86, University of California, Berkeley, 1986.

[7] L.O.Chua and A.C.Deng, "Negative resistance devices; part II," *Int. J. Circuit Theory and Applications* , Vol.12, pp.337-373.

Figure Captions

Fig.1 A linear resistive n-port formed by extracting all the nonlinear and/or dynamic and/or time-varying elements from the circuit.

Fig.2 A linear dynamic m-port formed by extracting all the nonlinear and/or time-varying elements from the circuit.

Fig.3 A single transistor type-S negative resistance circuit.

Fig.4 Schmitt-trigger circuit.

Fig.5 A dynamic circuit with a piecewise-linear resistor.

Fig.6 A single transistor amplifier.

Fig.7 A linear RLC circuit.

Fig.8 A linear LC ladder circuit.

Fig.9 A linear dynamic circuit with a capacitor loop.

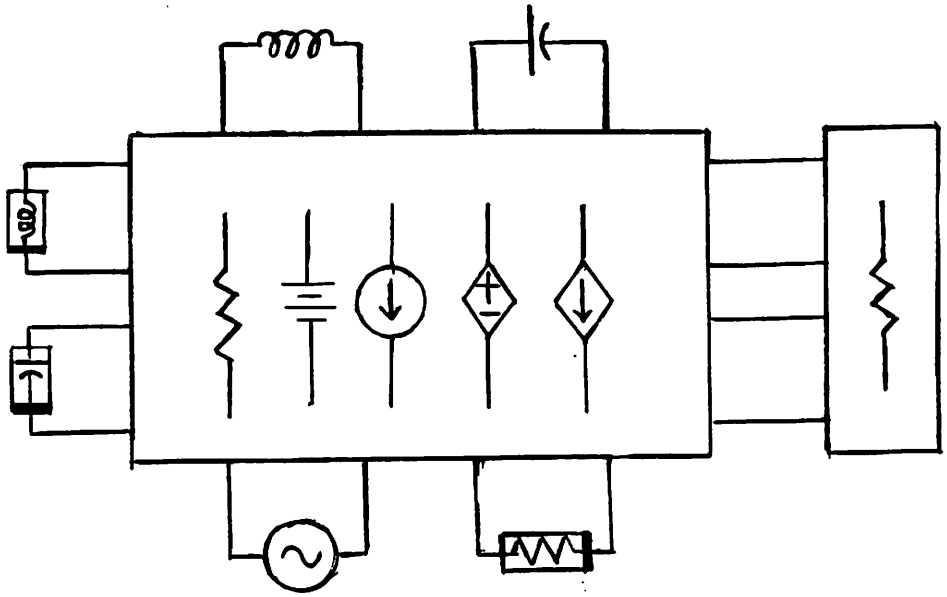


Fig.1

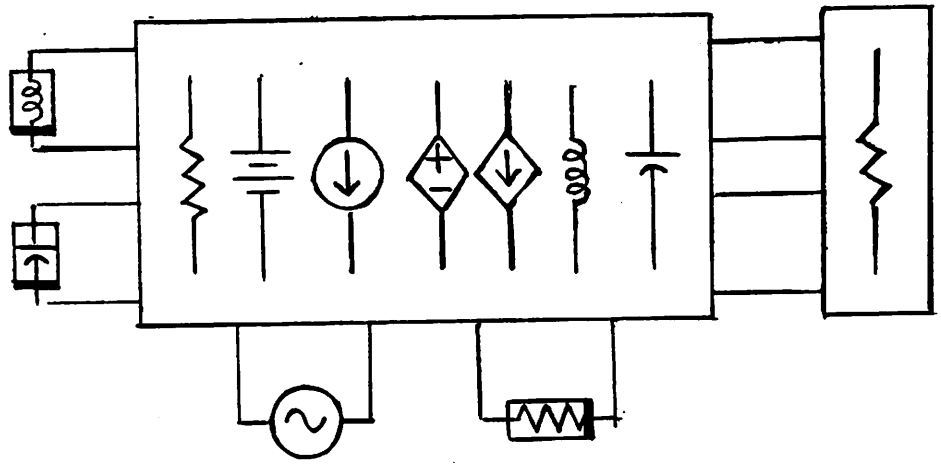
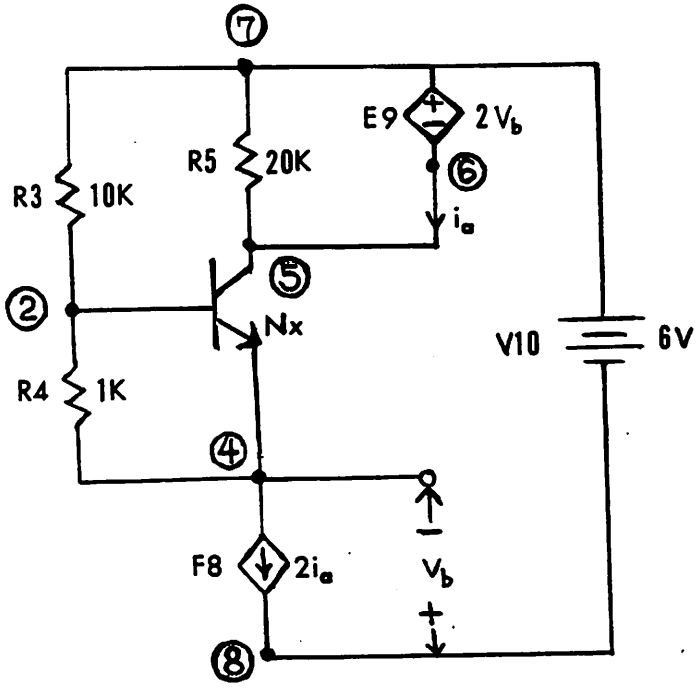
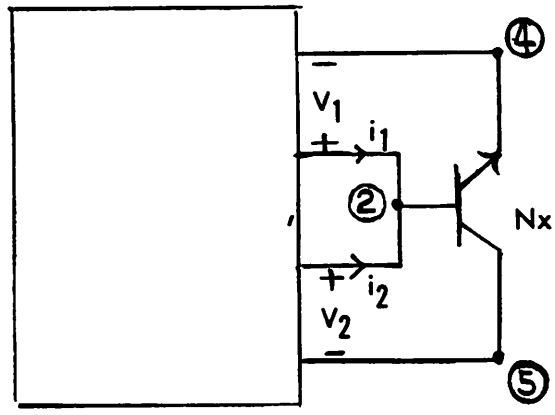


Fig.2

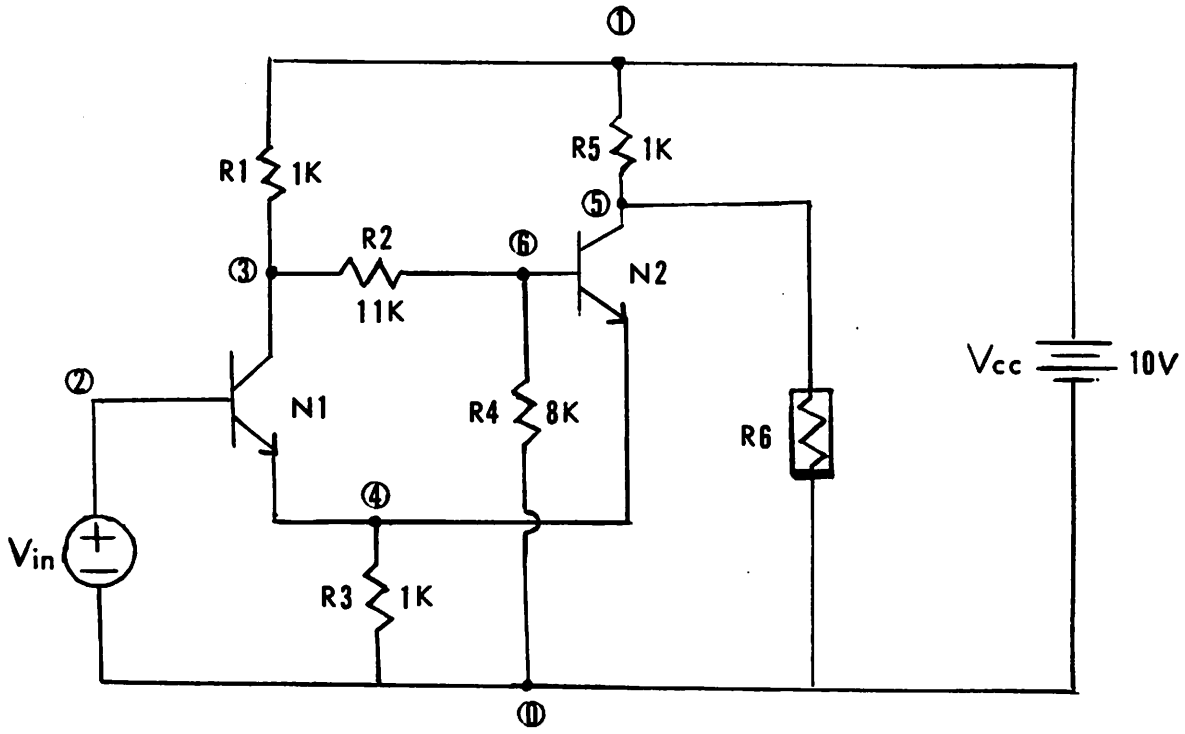


(a)

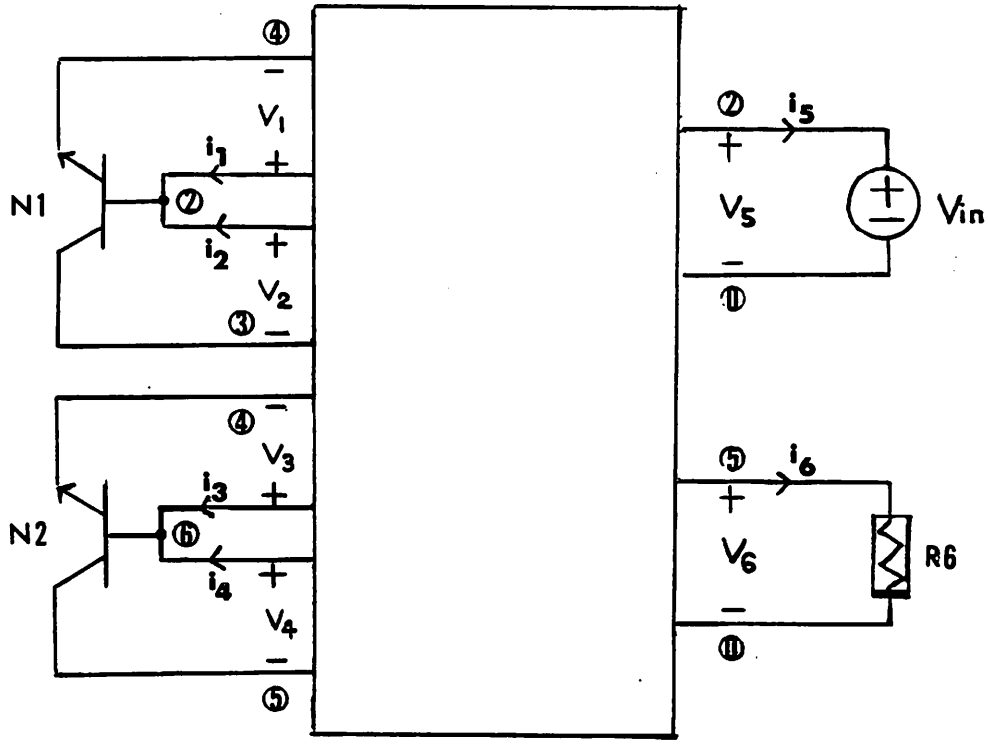


(b)

Fig.3


①
R1 1K
R5 1K
⑤
R2
⑥
③
N2
11K
Vcc
10V
②
N1
R4 8K
R6
④
Vin
R3 1K
⑪

(a)


④
V1
i1
N1
②
i2
V2
③
④
V3
i3
N2
⑥
i4
V4
⑤
②
i5
V5
Vin
⑪
⑤
i6
V6
R6
⑪

(b)

Fig. 4

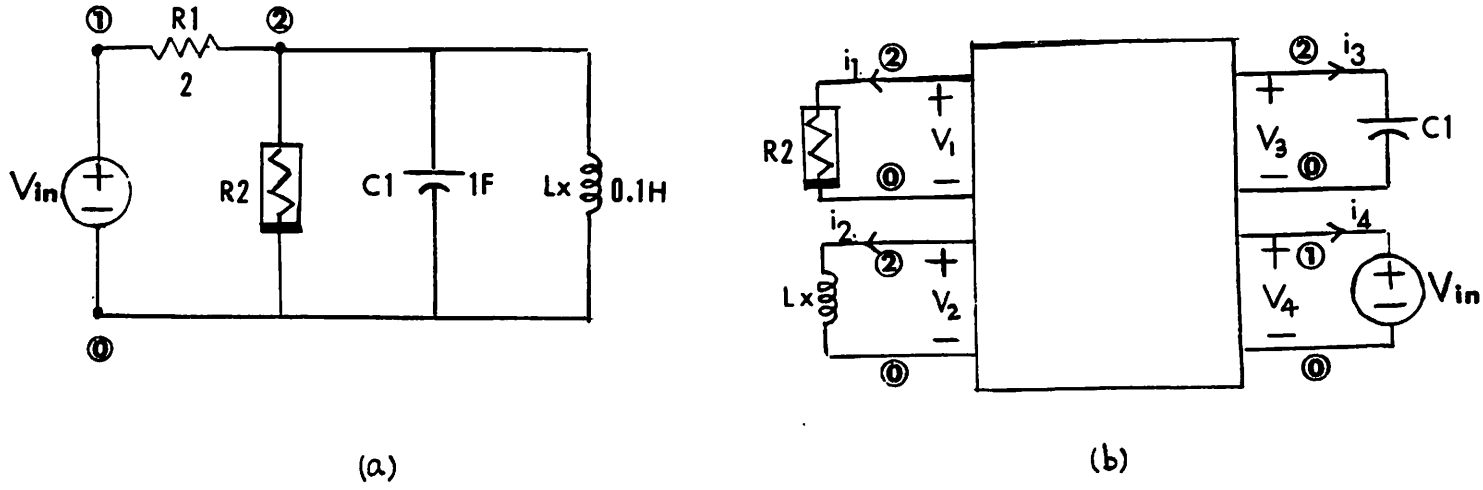

①
R1
②
2
Vin
R2
C1
1F
Lx
0.1H
⓪
(a)
i1
②
R2
V1
⓪
i2
②
Lx
V2
⓪
②
i3
V3
C1
⓪
i4
①
V4
Vin
⓪
(b)

Fig. 5

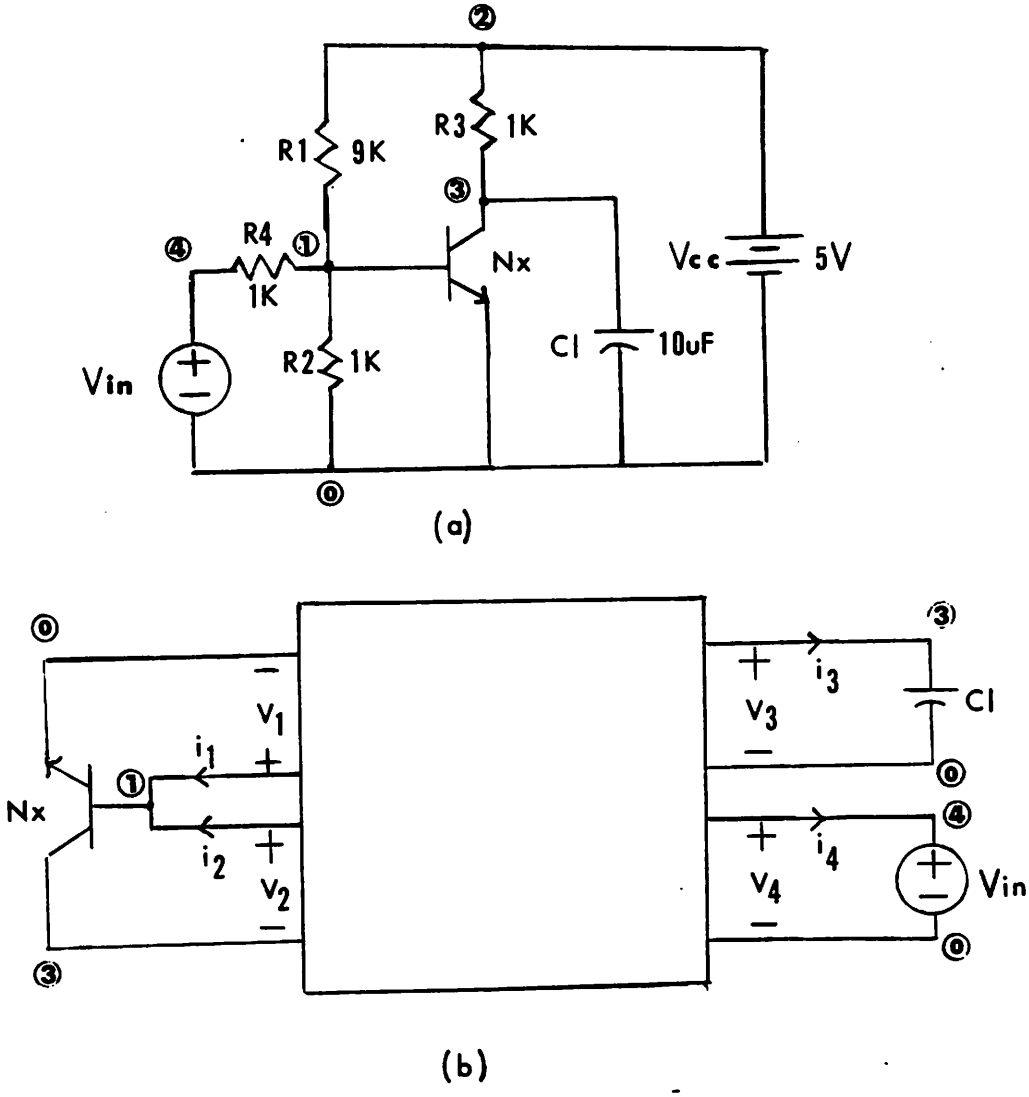

②
R1
9K
R3
1K
③
④
R4
①
1K
Nx
Vcc
5V
Vin
R2
1K
C1
10uF
⓪
(a)
⓪
V1
i1
①
Nx
i2
V2
③
③
i3
V3
C1
⓪
④
i4
V4
Vin
⓪
(b)

Fig. 6

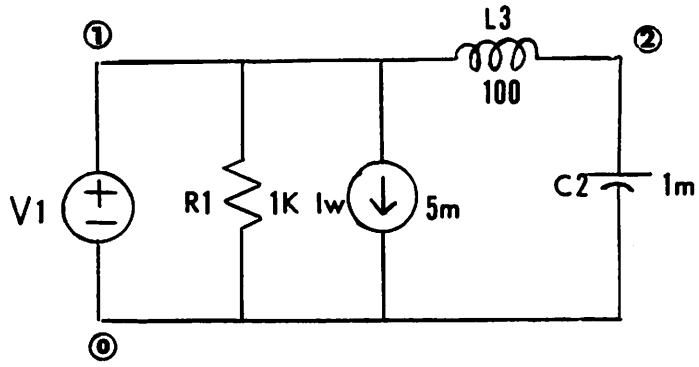


Fig. 7

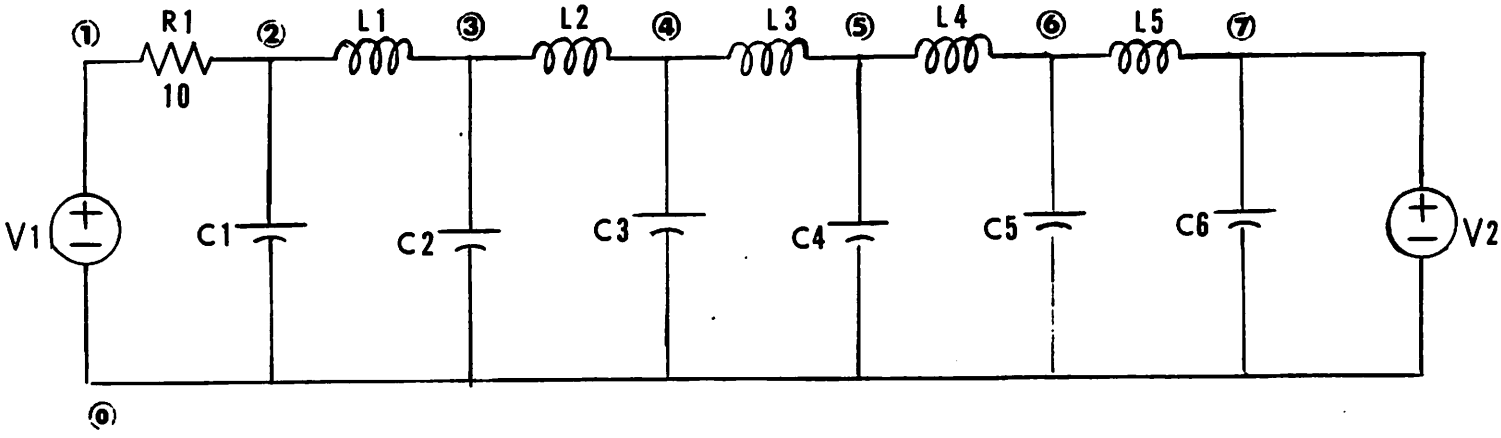


Fig. 8

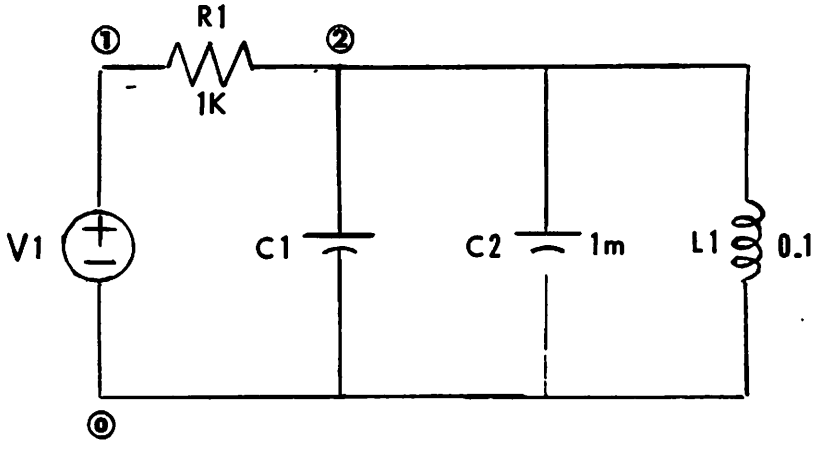


Fig. 9

```
* 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
*********************************************

GENERALIZED IMPLICIT EQUATION OF THE
      LINEAR RESISTIVE 2-PORT

                P*v + Q*i + s = 0

Nx(branch 1) is connected across port-1 (between nodes 2 and 4)
        v[1]=v[2,4]     i[1]=i[2,4]

Nx(branch 2) is connected across port-2 (between nodes 2 and 5)
        v[2]=v[2,5]     i[2]=i[2,5]


********** P matrix **********
P(1,1) = P[0] = 4.000e-04
P(1,2) = P[1] = -3.000e-04
P(2,1) = P[2] = -1.200e+00
P(2,2) = P[3] = 1.000e-01

********** Q matrix **********
Q(1,1) = Q[0] = 0.000e+00
Q(1,2) = Q[1] = -1.000e+00
Q(2,1) = Q[2] = -1.000e+03
Q(2,2) = Q[3] = -1.000e+03

********** s vector **********
s(1) = s[0] = -2.400e-03
s(2) = s[1] = 1.200e+00
```

```
* Example 2
*
* 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
*
* the output nodes , nodes 5 and 0, are connected by an open-circuit resistor
R6 5 0 {i=(0,0)(0.5,0)}
*
* graphic control lines
.x_name Vin
.y_name Vout
.title Schmitt Trigger TC
.x_axis 0 10
.y_axis 0 12
*
* driving voltage source
Vin 2 0 {(0,10):+}
*
* the bipolar transistor is modelled by 2-dimensional canomical pwl function
.model bpmod {(i,i):P1=(-1.85935e-3,0.265431,-0.262774,1.115e-3,1.8786e-2,
$6.885e-2,0.17668,-1.104e-3,-1.86e-2,-6.817e-2,-0.1749);
$P2=(-0.18389,-0.262804,0.525658,
$-1.104e-3,-1.86e-2,-6.817e-2,-0.17493,
$2.208e-3,3.721e-2,0.13634,0.3499);Bd=(1,0,0.5297,1,0,0.6362,1,0,0.6817,
$1,0,0.7144,0,1,0.5297,0,1,0.6362,0,1,0.6817,0,1,0.7144)}
*
.end
```

```
*************** SPICE INPUT *****************

* Example 2
*
* 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
*
* the output nodes , nodes 5 and 0, are connected by an open-circuit resistor
R6 5 0 {i=(0,0)(0.5,0)}
*
* graphic control lines
.x_name Vin
.y_name Vout
.title Schmitt Trigger TC
.x_axis 0 10
.y_axis 0 12
*
* driving voltage source
Vin 2 0 {(0,10):+}
*
* the bipolar transistor is modelled by 2-dimensional canomical pwl function
.model bpmod {(i,i):P1=(-1.85935e-3,0.265431,-0.262774,1.115e-3,1.8786e-2,
$6.885e-2,0.17668,-1.104e-3,-1.86e-2,-6.817e-2,-0.1749);
$P2=(-0.18389,-0.262804,0.525658,
$-1.104e-3,-1.86e-2,-6.817e-2,-0.17493,
$2.208e-3,3.721e-2,0.13634,0.3499);Bd=(1,0,0.5297,1,0,0.6362,1,0,0.6817,
$1,0,0.7144,0,1,0.5297,0,1,0.6362,0,1,0.6817,0,1,0.7144)}
*
.end
*******************************************

GENERALIZED IMPLICIT EQUATION OF THE
      LINEAR RESISTIVE 6-PORT

                P*v + Q*i + s = 0

N1(branch 1) is connected across port-1 (between nodes 2 and 4)
        v[1]=v[2,4]     i[1]=i[2,4]

N1(branch 2) is connected across port-2 (between nodes 2 and 3)
        v[2]=v[2,3]     i[2]=i[2,3]

N2(branch 1) is connected across port-3 (between nodes 6 and 4)
        v[3]=v[6,4]     i[3]=i[6,4]

N2(branch 2) is connected across port-4 (between nodes 6 and 5)
```

```
        v[4]=v[6,5]     i[4]=i[6,5]

Vin is connected across port-5 (between nodes 2 and 0)
        v[5]=v[2,0]     i[5]=i[2,0]

R6 is connected across port-6 (between nodes 5 and 0)
        v[6]=v[5,0]     i[6]=i[5,0]


********** P matrix **********
P(1,1) = P[0] = 1.000e+00
P(1,2) = P[1] = 0.000e+00
P(1,3) = P[2] = -1.000e+00
P(1,4) = P[3] = 1.000e+00
P(1,5) = P[4] = -1.000e+00
P(1,6) = P[5] = 1.000e+00
P(2,1) = P[6] = 0.000e+00
P(2,2) = P[7] = 0.000e+00
P(2,3) = P[8] = 0.000e+00
P(2,4) = P[9] = 0.000e+00
P(2,5) = P[10] = 0.000e+00
P(2,6) = P[11] = 0.000e+00
P(3,1) = P[12] = 1.000e+00
P(3,2) = P[13] = 8.000e+00
P(3,3) = P[14] = -1.000e+00
P(3,4) = P[15] = 0.000e+00
P(3,5) = P[16] = -9.000e+00
P(3,6) = P[17] = 0.000e+00
P(4,1) = P[18] = -1.000e+00
P(4,2) = P[19] = 0.000e+00
P(4,3) = P[20] = 1.000e+00
P(4,4) = P[21] = -1.000e+00
P(4,5) = P[22] = 1.000e+00
P(4,6) = P[23] = 0.000e+00
P(5,1) = P[24] = 1.000e+00
P(5,2) = P[25] = 0.000e+00
P(5,3) = P[26] = 0.000e+00
P(5,4) = P[27] = 0.000e+00
P(5,5) = P[28] = -1.000e+00
P(5,6) = P[29] = 0.000e+00
P(6,1) = P[30] = -1.000e+00
P(6,2) = P[31] = 1.200e+01
P(6,3) = P[32] = 1.000e+00
P(6,4) = P[33] = 0.000e+00
P(6,5) = P[34] = -1.100e+01
P(6,6) = P[35] = 0.000e+00

********** Q matrix **********
Q(1,1) = Q[0] = 0.000e+00
Q(1,2) = Q[1] = 0.000e+00
Q(1,3) = Q[2] = 0.000e+00
Q(1,4) = Q[3] = 0.000e+00
Q(1,5) = Q[4] = 0.000e+00
Q(1,6) = Q[5] = 0.000e+00
Q(2,1) = Q[6] = 1.000e+00
Q(2,2) = Q[7] = 1.000e+00
```

```
Q(2,3) = Q[8] = 0.000e+00
Q(2,4) = Q[9] = 0.000e+00
Q(2,5) = Q[10] = 1.000e+00
Q(2,6) = Q[11] = 0.000e+00
Q(3,1) = Q[12] = -8.000e+03
Q(3,2) = Q[13] = 0.000e+00
Q(3,3) = Q[14] = -8.000e+03
Q(3,4) = Q[15] = -8.000e+03
Q(3,5) = Q[16] = -8.000e+03
Q(3,6) = Q[17] = 0.000e+00
Q(4,1) = Q[18] = 0.000e+00
Q(4,2) = Q[19] = 0.000e+00
Q(4,3) = Q[20] = 0.000e+00
Q(4,4) = Q[21] = -1.000e+03
Q(4,5) = Q[22] = 0.000e+00
Q(4,6) = Q[23] = 1.000e+03
Q(5,1) = Q[24] = 1.000e+03
Q(5,2) = Q[25] = 0.000e+00
Q(5,3) = Q[26] = 1.000e+03
Q(5,4) = Q[27] = 0.000e+00
Q(5,5) = Q[28] = 0.000e+00
Q(5,6) = Q[29] = 0.000e+00
Q(6,1) = Q[30] = -1.100e+04
Q(6,2) = Q[31] = 0.000e+00
Q(6,3) = Q[32] = 0.000e+00
Q(6,4) = Q[33] = 0.000e+00
Q(6,5) = Q[34] = -1.100e+04
Q(6,6) = Q[35] = 0.000e+00

********** s vector **********
s(1) = s[0] = 0.000e+00
s(2) = s[1] = 0.000e+00
s(3) = s[2] = 8.000e+01
s(4) = s[3] = -1.000e+01
s(5) = s[4] = 0.000e+00
s(6) = s[5] = 1.100e+02
```

```
* Example 3
*
* dynamic circuit with piecewise-linear resistor
*
* a nonlinear resistor characterized by pwl model
R2 2 0 {i=4*v-1.75*fabs(v+1)+1.75*fabs(v-1)}
*
C1 2 0 1
Lx 2 0 0.1
R1 1 2 2
*
* sinusoidal input source
Vin 1 0 {sin(t)}
*
* include the file with mathematical declarations
.include "math.h"
.end
```

```
*************** SPICE INPUT *****************

* Example 3
*
* dynamic circuit with piecewise-linear resistor
*
* a nonlinear resistor characterized by pwl model
R2 2 0 {i=4*v-1.75*fabs(v+1)+1.75*fabs(v-1)}
*
C1 2 0 1
Lx 2 0 0.1
R1 1 2 2
*
* sinusoidal input source
Vin 1 0 {sin(t)}
*
* include the file with mathematical declarations
.include "math.h"
.end
**********************************************

GENERALIZED IMPLICIT EQUATION OF THE
      LINEAR RESISTIVE 4-PORT

                P*v + Q*i + s = 0

R2 is connected across port-1 (between nodes 2 and 0)
        v[1]=v[2,0]     i[1]=i[2,0]

Lx is connected across port-2 (between nodes 2 and 0)
        v[2]=v[2,0]     i[2]=i[2,0]

C1 is connected across port-3 (between nodes 2 and 0)
        v[3]=v[2,0]     i[3]=i[2,0]

Vin is connected across port-4 (between nodes 1 and 0)
        v[4]=v[1,0]     i[4]=i[1,0]


********** P matrix **********
P(1,1) = P[0] = -1.000e+00
P(1,2) = P[1] = 1.000e+00
P(1,3) = P[2] = 0.000e+00
P(1,4) = P[3] = 0.000e+00
P(2,1) = P[4] = -1.000e+00
P(2,2) = P[5] = 0.000e+00
P(2,3) = P[6] = 1.000e+00
P(2,4) = P[7] = 0.000e+00
P(3,1) = P[8] = 0.000e+00
P(3,2) = P[9] = 0.000e+00
P(3,3) = P[10] = 0.000e+00
P(3,4) = P[11] = 0.000e+00
P(4,1) = P[12] = 1.000e+00
P(4,2) = P[13] = 0.000e+00
P(4,3) = P[14] = 0.000e+00
P(4,4) = P[15] = -1.000e+00
```

```
********** Q matrix **********
Q(1,1) = Q[0] = 0.000e+00
Q(1,2) = Q[1] = 0.000e+00
Q(1,3) = Q[2] = 0.000e+00
Q(1,4) = Q[3] = 0.000e+00
Q(2,1) = Q[4] = 0.000e+00
Q(2,2) = Q[5] = 0.000e+00
Q(2,3) = Q[6] = 0.000e+00
Q(2,4) = Q[7] = 0.000e+00
Q(3,1) = Q[8] = 1.000e+00
Q(3,2) = Q[9] = 1.000e+00
Q(3,3) = Q[10] = 1.000e+00
Q(3,4) = Q[11] = 1.000e+00
Q(4,1) = Q[12] = 0.000e+00
Q(4,2) = Q[13] = 0.000e+00
Q(4,3) = Q[14] = 0.000e+00
Q(4,4) = Q[15] = -2.000e+00

********** s vector **********
s(1) = s[0] = 0.000e+00
s(2) = s[1] = 0.000e+00
s(3) = s[2] = 0.000e+00
s(4) = s[3] = 0.000e+00
```

```
* Example 4
*
* single transistor amplifier
Nx 1 0 1 3 bpmod
C1 3 0 10u
R4 1 4 1K
R1 1 2 9K
R2 1 0 1K
R3 2 3 1K
Vcc 2 0 5
*
* sinusoidal input
Vin 4 0 {sin(1000*t)}
*
* bipolar transistor model
.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 "math.h"
.include "math.h"
.end
```

```
*************** SPICE INPUT *****************

* Example 4
*
* single transistor amplifier
Nx 1 0 1 3 bpmod
C1 3 0 10u
R4 1 4 1K
R1 1 2 9K
R2 1 0 1K
R3 2 3 1K
Vcc 2 0 5
*
* sinusoidal input
Vin 4 0 {sin(1000*t)}
*
* bipolar transistor model
.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 "math.h"
.include "math.h"
.end
**********************************************

GENERALIZED IMPLICIT EQUATION OF THE
      LINEAR RESISTIVE 4-PORT

                P*v + Q*i + s = 0

Nx(branch 1) is connected across port-1 (between nodes 1 and 0)
        v[1]=v[1,0]     i[1]=i[1,0]

Nx(branch 2) is connected across port-2 (between nodes 1 and 3)
        v[2]=v[1,3]     i[2]=i[1,3]

C1 is connected across port-3 (between nodes 3 and 0)
        v[3]=v[3,0]     i[3]=i[3,0]

Vin is connected across port-4 (between nodes 4 and 0)
        v[4]=v[4,0]     i[4]=i[4,0]


********** P matrix **********
P(1,1) = P[0] = -1.000e+00
P(1,2) = P[1] = 0.000e+00
P(1,3) = P[2] = 0.000e+00
P(1,4) = P[3] = 1.000e+00
P(2,1) = P[4] = -1.000e+00
P(2,2) = P[5] = 1.000e+00
P(2,3) = P[6] = 1.000e+00
P(2,4) = P[7] = 0.000e+00
P(3,1) = P[8] = 1.000e+00
P(3,2) = P[9] = -1.000e+00
P(3,3) = P[10] = 0.000e+00
P(3,4) = P[11] = 0.000e+00
P(4,1) = P[12] = -1.111e+00
```

```
P(4,2) = P[13] = 0.000e+00
P(4,3) = P[14] = 0.000e+00
P(4,4) = P[15] = 0.000e+00

********** Q matrix **********
Q(1,1) = Q[0] = 0.000e+00
Q(1,2) = Q[1] = 0.000e+00
Q(1,3) = Q[2] = 0.000e+00
Q(1,4) = Q[3] = 1.000e+03
Q(2,1) = Q[4] = 0.000e+00
Q(2,2) = Q[5] = 0.000e+00
Q(2,3) = Q[6] = 0.000e+00
Q(2,4) = Q[7] = 0.000e+00
Q(3,1) = Q[8] = 0.000e+00
Q(3,2) = Q[9] = -1.000e+03
Q(3,3) = Q[10] = 1.000e+03
Q(3,4) = Q[11] = 0.000e+00
Q(4,1) = Q[12] = -1.000e+03
Q(4,2) = Q[13] = -1.000e+03
Q(4,3) = Q[14] = 0.000e+00
Q(4,4) = Q[15] = -1.000e+03

********** s vector **********
s(1) = s[0] = 0.000e+00
s(2) = s[1] = 0.000e+00
s(3) = s[2] = -5.000e+00
s(4) = s[3] = 5.556e-01
```

```
* Example 5
*
* Linear RLC 1-port circuit
R1 1 0 1K
L3 1 2 100
C2 2 0 1m
Iw 1 0 5m
V1 1 0 (100*sin(1000*t))
.end
```

```
******************** SPICE INPUT ********************
* Example 5
*
* Linear RLC 1-port circuit
R1 1 0 1K
L3 1 2 100
C2 2 0 1m
Iw 1 0 5m
V1 1 0 {100*sin(1000*t)}
.end
*****************************************************


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

COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
        LINEAR DYNAMIC 1-PORT

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[1,0]

*************************************************************

FREQUENCY = 1.000000e+00
***********************  P MATRIX  ***********************
Re(P[1,1]) = -1.000e+00    Im(P[1,1]) = 2.131e+00
***********************  Q MATRIX  ***********************
Re(Q[1,1]) = -1.000e+03    Im(Q[1,1]) = 0.000e+00
***********************  s VECTOR  ***********************
Re(s[1])    = -5.000e+00    Im(s[1])    = 0.000e+00

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



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

COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
        LINEAR DYNAMIC 1-PORT

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[1,0]

*************************************************************

FREQUENCY = 1.000000e+01
***********************  P MATRIX  ***********************
Re(P[1,1]) = -1.000e+00    Im(P[1,1]) = 1.596e-01
***********************  Q MATRIX  ***********************
Re(Q[1,1]) = -1.000e+03    Im(Q[1,1]) = 0.000e+00
***********************  s VECTOR  ***********************
Re(s[1])    = -5.000e+00    Im(s[1])    = 0.000e+00

-------------------------------------------------------------
```

```
* Example 6
*
* Linear LC ladder 2-port circuit
*
V1 1 0 {sin(t)}
R1 1 2 10
C1 2 0 0.1
L1 2 3 1
C2 3 0 0.1
L2 3 4 1
C3 4 0 0.1
L3 4 5 1
C4 5 0 0.1
L4 5 6 1
C5 6 0 0.1
L5 6 7 1
C6 7 0 0.1
V2 7 0 {cos(t)}
.end
```

```
******************** SPICE INPUT ********************
* Example 6
*
* Linear LC ladder 2-port circuit
*
V1 1 0 {sin(t)}
R1 1 2 10
C1 2 0 0.1
L1 2 3 1
C2 3 0 0.1
L2 3 4 1
C3 4 0 0.1
L3 4 5 1
C4 5 0 0.1
L4 5 6 1
C5 6 0 0.1
L5 6 7 1
C6 7 0 0.1
V2 7 0 {cos(t)}
.end
*****************************************************
```

```
COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
        LINEAR DYNAMIC 2-PORT

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[1,0]

port 2 is connected by V2 between nodes 7 and 0
        v[2]=v[7,0]     i[2]=i[7,0]

*********************************************************

FREQUENCY = 1.000000e+00
***********************  P MATRIX  **********************
Re(P[1,1]) = 0.000e+00    Im(P[1,1]) = 0.000e+00
Re(P[1,2]) = 1.000e+00    Im(P[1,2]) = 0.000e+00
Re(P[2,1]) = -1.000e+00    Im(P[2,1]) = 0.000e+00
Re(P[2,2]) = 0.000e+00    Im(P[2,2]) = 0.000e+00
***********************  Q MATRIX  **********************
Re(Q[1,1]) = 0.000e+00    Im(Q[1,1]) = 3.683e-01
Re(Q[1,2]) = 0.000e+00    Im(Q[1,2]) = -3.069e+00
Re(Q[2,1]) = -1.000e+01    Im(Q[2,1]) = 3.069e+00
Re(Q[2,2]) = 3.553e-15    Im(Q[2,2]) = -3.683e-01
***********************  s VECTOR  **********************
Re(s[1])    = 0.000e+00    Im(s[1])    = 0.000e+00
Re(s[2])    = 0.000e+00    Im(s[2])    = 0.000e+00
```

```
COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
        LINEAR DYNAMIC 2-PORT

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[1,0]

port 2 is connected by V2 between nodes 7 and 0
        v[2]=v[7,0]     i[2]=i[7,0]

*********************************************************

FREQUENCY = 1.000000e+03
***********************  P MATRIX  **********************
Re(P[1,1]) = 0.000e+00    Im(P[1,1]) = 0.000e+00
Re(P[1,2]) = 1.000e+00    Im(P[1,2]) = 0.000e+00
Re(P[2,1]) = -1.000e+00    Im(P[2,1]) = 0.000e+00
Re(P[2,2]) = 0.000e+00    Im(P[2,2]) = 0.000e+00
***********************  Q MATRIX  **********************
Re(Q[1,1]) = 0.000e+00    Im(Q[1,1]) = -5.170e-26
Re(Q[1,2]) = 0.000e+00    Im(Q[1,2]) = -1.592e-03
Re(Q[2,1]) = -1.000e+01    Im(Q[2,1]) = 1.592e-03
Re(Q[2,2]) = 0.000e+00    Im(Q[2,2]) = -2.444e-20
***********************  s VECTOR  **********************
Re(s[1])    = 0.000e+00    Im(s[1])    = 0.000e+00
Re(s[2])    = 0.000e+00    Im(s[2])    = 0.000e+00

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


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

COMPLEX GENERALIZED IMPLICIT EQUATION OF THE
        LINEAR DYNAMIC 2-PORT

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[1,0]

port 2 is connected by V2 between nodes 7 and 0
        v[2]=v[7,0]     i[2]=i[7,0]

*********************************************************

FREQUENCY = 1.000000e-03
***********************  P MATRIX  **********************
Re(P[1,1]) = 0.000e+00    Im(P[1,1]) = 0.000e+00
Re(P[1,2]) = 1.000e+00    Im(P[1,2]) = 0.000e+00
Re(P[2,1]) = -1.000e+00    Im(P[2,1]) = 0.000e+00
Re(P[2,2]) = 0.000e+00    Im(P[2,2]) = 0.000e+00
***********************  Q MATRIX  **********************
Re(Q[1,1]) = 0.000e+00    Im(Q[1,1]) = -2.653e+02
Re(Q[1,2]) = 0.000e+00    Im(Q[1,2]) = -2.652e+02
Re(Q[2,1]) = -1.000e+01    Im(Q[2,1]) = 2.652e+02
Re(Q[2,2]) = 3.553e-15    Im(Q[2,2]) = 2.653e+02
***********************  s VECTOR  **********************
```

```
Re(s[1])    = 0.000e+00    Im(s[1])    = 0.000e+00
Re(s[2])    = 0.000e+00    Im(s[2])    = 0.000e+00

-------------------------------------------------------------
```

```
************************ SPICE INPUT **********************
* Example 5
*
* Linear RLC 1-port circuit
R1 1 0 1K
L3 1 2 100
C2 2 0 1m
Iw 1 0 5m
V1 1 0 (100*sin(1000*t))
.end
***********************************************************
```

EXPLICIT LINEAR DYNAMIC 1-PORT
IMPEDANCE MATRIX REPRESENTATION

port 1 is connected by V1 between nodes 1 and 0
v[1]=v[1,0] i[1]=i[0,1]

```
***********************************************************
```

FREQUENCY = 1.000000e+00

```
************************ Z MATRIX *************************
```

Re(Z[1,1]) = 1.804e+02 Im(Z[1,1]) = 3.845e+02

```
************************ c VECTOR *************************
```

Re(c[1]) = -9.020e-01 Im(c[1]) = -1.923e+00

EXPLICIT LINEAR DYNAMIC 1-PORT
IMPEDANCE MATRIX REPRESENTATION

port 1 is connected by V1 between nodes 1 and 0
v[1]=v[1,0] i[1]=i[0,1]

```
***********************************************************
```

FREQUENCY = 1.000000e+01

```
************************ Z MATRIX *************************
```

Re(Z[1,1]) = 9.752e+02 Im(Z[1,1]) = 1.556e+02

```
************************ c VECTOR *************************
```

Re(c[1]) = -4.876e+00 Im(c[1]) = -7.780e-01

```
EXPLICIT LINEAR DYNAMIC 1-PORT
IMPEDANCE MATRIX REPRESENTATION

port 1 is connected by V1 between nodes 1 and 0
        v[1]=v[1,0]     i[1]=i[0,1]

*************************************************************

FREQUENCY = 1.000000e-01
************************ Z MATRIX ************************
Re(Z[1,1]) = 7.003e+02  Im(Z[1,1]) = -4.581e+02

************************ c VECTOR ************************
Re(c[1]) = -3.502e+00   Im(c[1]) = 2.291e+00

-------------------------------------------------------------
```

```
************************ SPICE INPUT **********************
* Example 6
*
* Linear LC ladder 2-port circuit
*
V1 1 0 (sin(t))
R1 1 2 10
C1 2 0 0.1
L1 2 3 1
C2 3 0 0.1
L2 3 4 1
C3 4 0 0.1
L3 4 5 1
C4 5 0 0.1
L4 5 6 1
C5 6 0 0.1
L5 6 7 1
C6 7 0 0.1
V2 7 0 (cos(t))
.end
************************************************************
```

EXPLICIT LINEAR DYNAMIC 2-PORT
SCATTERING MATRIX REPRESENTATION

port 1 is connected by V1 between nodes 1 and 0
v[1]=v[1,0] i[1]=i[0,1]

port 2 is connected by V2 between nodes 7 and 0
v[2]=v[7,0] i[2]=i[0,7]

```
************************************************************

FREQUENCY = 1.000000e+00
************************ S MATRIX *************************
Re(S[1,1]) = 8.314e-01  Im(S[1,1]) = -4.692e-02
Re(S[1,2]) = -1.050e-02 Im(S[1,2]) = 1.641e-02
Re(S[2,1]) = -1.050e-02 Im(S[2,1]) = 1.641e-02
Re(S[2,2]) = -8.264e-01 Im(S[2,2]) = -5.594e-01

************************ c VECTOR *************************
Re(c[1]) = 0.000e+00    Im(c[1]) = 0.000e+00
Re(c[2]) = 0.000e+00    Im(c[2]) = 0.000e+00
```

```
* Example 7
*
* a linear RLC circuit with capacitor loop
*
Vin 1 0 {sin(t)}
R1 1 2 1K
C1 2 0 1m
C2 2 0 2m
L1 2 0 0.1
.end
```

```
************************ SPICE INPUT **********************
* Example 7
*
* a linear RLC circuit with capacitor loop
*
Vin 1 0 (sin(t))
R1 1 2 1K
C1 2 0 1m
C2 2 0 2m
L1 2 0 0.1
.end
***********************************************************

WARNING MESSAGE : CAPACITOR LOOP OR INDUCTOR CUTSET
******************* LINEAR STATE EQUATION *******************

                x' = A*x + B*u + s_1 + T_1*u'
                y  = C*x + D*u + s_2 + T_2*u'

***** state variables *****
x[1]=v[2,0](C2)
x[2]=i[2,0](L1)

***** input variables *****
u[1]=v[1,0](Vin)

***** output variables *****
y[1]=i[1,0](Vin)

-----------------------------------------------------------
************************** A MATRIX **********************
A[1,1] = -3.333e-01
A[1,2] = -3.333e+02
A[2,1] = 1.000e+01
A[2,2] = 0.000e+00
************************** B MATRIX **********************
B[1,1] = 3.333e-01
B[2,1] = 0.000e+00
************************** C MATRIX **********************
C[1,1] = 1.000e-03
C[1,2] = 0.000e+00
************************** D MATRIX **********************
D[1,1] = -1.000e-03
************************* s_1 VECTOR *********************
s_1[1] = 0.000e+00
s_1[2] = 0.000e+00
************************* s_2 VECTOR *********************
s_2[1] = 0.000e+00
************************* T_1 MATRIX *********************
T_1[1,1] = 0.000e+00
T_1[2,1] = 0.000e+00
************************* T_2 MATRIX *********************
T_2[1,1] = 0.000e+00
```

APPENDIX : SOURCE CODE LISTINGS

1. Linear resistive n-port

nport.c, nport1.c, nport2.c, nport3.c, aclib.c

2. Linear dynamic m-port

2.1. Implicit representation

cnport.c, cnport1.c, cnport2.c

2.2. Explicit representation

porteq.c, porteq1.c, porteq2.c

3. Linear state equation

staeq.c, staeq1.c, staeq2.c

```
#define NODE 50
struct INLINE
{
   char name[9];
   int port;
   int node1,node2,node3,node4,node5,node6,node7,node8;
   char *relation;
};
struct B_VECTOR
{
   int a,b;
};
typedef char *STRING;
```

```
#include <stdio.h>
#include "nport.h"

main(argc,argv)
int argc;
char *argv[];
{
    char line[20];
    char *model[30];
    struct INLINE branch[70];
    struct B_VECTOR branch_vector[70];
    double *P,*Q,*s;
    int i,j,n,mn,n1,n2;
    FILE *fp,*fopen();

    /* open the input file "xx...x.spc" */
    if (argc!=2)
        exit_message("NPORT SPICE_FILE");
    sprintf(line,"%s.spc",*++argv);
    if ((fp=fopen(line,"r"))==NULL)
    {
        printf("CAN'T OPEN SPICE_FILE %s\n",line);
        exit();
    }

    printf("*************** SPICE INPUT *****************\n\n");
    n=70;
    mn=30;
    n_port(fp,model,branch,branch_vector,&P,&Q,&s,&n,&mn);
    fclose(fp);
    printf("*********************************************\n\n");
    printf("GENERALIZED IMPLICIT EQUATION OF THE\n");
    printf("      LINEAR RESISTIVE %d-PORT\n\n",n);
    printf("\t\tP*v + Q*i + s = 0\n\n");
    for (i=0;i<n;i++)
    {
        j=(branch_vector+i)->a;
        if ((branch+j)->port>1)
        {
            sprintf(line,"%s(branch %d)",(branch+j)->name,
                          (branch_vector+i)->b);
            switch((branch_vector+i)->b)
            {
                case 1 : {
                             n1=(branch+j)->node1;
                             n2=(branch+j)->node2;
                             break;
                         }
                case 2 : {
                             n1=(branch+j)->node3;
                             n2=(branch+j)->node4;
                             break;
                         }
                case 3 : {
                             n1=(branch+j)->node5;
                             n2=(branch+j)->node6;
```

```
                                break;
                            }
                    case 4 : {
                                n1=(branch+j)->node7;
                                n2=(branch+j)->node8;
                                break;
                            }
                }
            }
            else
            {
                n1=(branch+j)->node1;
                n2=(branch+j)->node2;
                sprintf(line,"%s",(branch+j)->name);
            }
            printf("%s is connected across port-%d (between nodes %d and %d)\n",
                    line,i+1,n1,n2);
            printf("\tv[%d]=v[%d,%d]\ti[%d]=i[%d,%d]\n\n",i+1,n1,n2,i+1,n1,n2);
        }
        printf("\n********** P matrix **********\n");
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                printf("P(%d,%d) = P[%d] = %.3e\n",i+1,j+1,i*n+j,P[i*n+j]);
        printf("\n********** Q matrix **********\n");
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                printf("Q(%d,%d) = Q[%d] = %.3e\n",i+1,j+1,i*n+j,Q[i*n+j]);
        printf("\n********** s vector **********\n");
        for (i=0;i<n;i++)
            printf("s(%d) = s[%d] = %.3e\n",i+1,i,s[i]);
}
```

```
#include <stdio.h>
#include "nport.h"

STRING *st;                     /* array of input lines              */
int ln;                         /* # of input lines (excluding the   */
                                /* last line ".end")                 */
int mbran=0;                    /* total number of branches          */
int *D_ab,*D_ay,*D_zb,*D_zy;    /* fundamental cutset matrices (see  */
                                /* Eq.(6.55) in Chua and Lin's book) */
double *F_iy,*F_vy,*F_iz,*F_vz; /* element behavior equation for all */
                                /* linear elements (see Eq.(6.65) in */
                                /* Chua and Lin's book)              */


/*************************************************************************/
/* Formulate the generalized implicit equation                           */
/*              P*v + Q*i + s = 0                                        */
/* for a linear resistive n-port.                                        */
/*************************************************************************/

n_port(fp,moda,branch,branch_vector,P,Q,s,n,mn)

FILE *fp;                         /* pointer to the spice file          */
char *moda[];                     /* array of model/title/comment lines */
struct INLINE branch[];           /* element information structure      */
struct B_VECTOR branch_vector[];  /* branch index structure :           */
                                  /* the (branch_vector[i]->b)-th branch */
                                  /* of the (branch_vector[i]->a)-th    */
                                  /* element is connected across the    */
                                  /* i-th port of the linear n-port     */
double **P,**Q,**s;               /* parameters of hybrid equation :    */
                                  /*            P*v + Q*i + s = 0       */
int *n;                           /* port number of the linear n-port   */
int *mn;                          /* # of the model/title/comment lines */

{
    char *calloc();
    STRING *bst;                  /* array of element lines             */
    int bn,xn;                    /* # of element lines                 */

    bst= (STRING *) calloc(*n,sizeof(STRING));
    xn = *mn;
    bn = *n;

    /* read the circuit from input file */
    read_ckt(fp,bst,moda,branch,&bn,mn);

    if (*n<bn)
        exit_message("INSUFFICIENT SPACE FOR INPUT LINE ARRAY");
    if (xn<*mn)
        exit_message("INSUFFICIENT SPACE FOR MODEL LINE ARRAY");

    /* formulate the n-port equation */
    get_n_port(branch,branch_vector,P,Q,s,n,bn);
}
```

```
/*************************************************************************/
/* Read the input file "xx...x.spc" and separate each input line into     */
/* the element line (stored in the array bst) or the model/graphic control */
/* /declaration line (stored in the array moda).                          */
/*************************************************************************/

read_ckt(fp,bst,moda,branch,bn,mn)
FILE *fp;
char *bst[];
char *moda[];
struct INLINE branch[];
int *bn;
int *mn;
{
    /* read the input file and store each input line in the array st */
    ln=read_file(fp,(*bn)+(*mn));

    /* separate input lines into element lines and */
    /* model/graphic control/declaration lines     */
    branch_model(bst,moda,bn,mn);

    /* identify the name, nodes, and relation of each   */
    /* element and store them in the structure "branch" */
    store_branch(bst,*bn,branch);

    /* replace the model name by a real model description */
    model_name(branch,moda,*bn,*mn);
}

/*************************************************************************/
/* Get the generalized implicit equation                                 */
/*              Pv + Qi + s = 0                                          */
/* for the linear resistive n-port whose citcuit topology and element    */
/* characteristics are stored in the structure arrays "branch[]" which is */
/* found by calling read_ckt().                                          */
/*************************************************************************/

get_n_port(branch,branch_vector,P,Q,s,n,bn)
struct INLINE branch[];
struct B_VECTOR branch_vector[];
double **P,**Q,**s;
int *n,bn;
{
    int *A;                     /* incidence matrix                    */
    int *node_vector;           /* array of each node_number           */
    int node_number;            /* total # of nodes in the circuit     */
    int branch_number;          /* total # of branches in the circuit  */
    int tp;                     /* # of port branches in the tree      */
    int tn;                     /* # of non-port branches in the tree  */
    int cp;                     /* # of port branches in the cotree    */
    int cn;                     /* # of non-port branches in the cotree*/
    int mx;                     /* row # of constraint matrix          */
    int px;                     /* column # of constraint matrix       */
    double *F;                  /* linear matrix characterization of   */
                                /* all linear elements                 */
    double fabs();
```

```
    /* allocates spaces for the incidence matrix and */
    /* the node vector with roughly estimated sizes  */
    pre_alloc(&A,&node_vector);

    /* find the incidence matrix A */
    incidence_matrix(branch,&A,branch_vector,&node_vector,&node_number,
                     &branch_number,bn);

    /* Extract the nonlinear elements to form a linear n-port */
    tree_port(branch,A,branch_vector,branch_number,
              node_number,&tp,&tn,&cp,&cn);

    /* construct the branch_relation matrix for linear elements */
    l_tree(branch,branch_vector,branch_number,node_number,tp,tn,cp,cn);
    l_cotree(branch,branch_vector,branch_number,node_number,tp,tn,cp,cn);

    /* construct the tableau matrix F */
    F_matrix(&F,tp,tn,cp,cn);

    /* perform row operation on F for hybrid matrix equation */
    F_reduce(F,P,Q,branch_vector,tp,tn,cp,cn,&mx,&px);

    /* remove the independent dc sources */
    rm_source(branch,P,Q,s,branch_vector,&mx,&px);

    *n=mx;      /* # of nonlinear elements */
    /* reset small parameter values to zero */
    underflow(*n,*P,*Q,*s);
}

/*************************************************************************/
/* Reset small parameter values of P, Q, and s ( due to numerical underflow*/
/* in the intermediate computations ) to zero.                           */
/*************************************************************************/

underflow(n,P,Q,s)
int n;
double *P,*Q,*s;
/* the ij-th entry of the matrices (i) P (ii) Q is expressed as */
/*                        P[i*n+j]                              */
/*                        Q[i*n+j]                              */

{
    int i,j;

    for (i=0;i<n;i++)
    {
        if (fabs(s[i])<1.0E-12 && s[i]!=0.0)
        {
            printf("WARNING MESSAGE : UNDERFLOW PROBLEM FOR");
            printf(" s[%d]=%.3e\n",i,s[i]);
            s[i]=0.0;
        }
        for (j=0;j<n;j++)
        {
```

```
            if (fabs(P[i*n+j])<1.0E-12 && P[i*n+j]!=0.0)
            {
                printf("WARNING MESSAGE : UNDERFLOW PROBLEM FOR");
                printf(" P[%d]=%.3e\n",i*n+j,P[i*n+j]);
                P[i*n+j]=0.0;
            }
            if (fabs(Q[i*n+j])<1.0E-12 && Q[i*n+j]!=0.0)
            {
                printf("WARNING MESSAGE : UNDERFLOW PROBLEM FOR");
                printf(" Q[%d]=%.3e\n",i*n+j,Q[i*n+j]);
                Q[i*n+j]=0.0;
            }
        }
    }
}

/*************************************************************************/
/* Replace the model_name xxx..x in the element_relation field with the    */
/* real description (yyyy...yy) defined in the model line                  */
/*              .model xxx..x (yyyy...yy)                                  */
/*************************************************************************/

model_name(branch,moda,bn,mn)
struct INLINE branch[];
char *moda[];
int bn,mn;
{
    int i,j,k;
    char s[500],sx[10],*calloc();

    for (i=0;i<bn;i++)
    {
        strcpy(s,(branch+i)->relation);

        /* element_relation is defined by a model_name */
        if (s[0]!='(' && (s[0]<'0' || s[0]>'9') && s[0]!='-' && s[0]!='+')
        {
            j=0;

            /* search for the model_name definition in the input lines */
            while (j<mn)
            {
                if (find_index(".model",moda[j])==0)
                {
                    find_word(moda[j],sx,2);
                    if (find_index(s,sx)==0 && find_index(sx,s)==0) break;
                    else j++;
                }
                else j++;
            }
            if (j>=mn)  /* can't find the model definition */
            {
                printf("MODEL %s IS NOT AVAILABLE\n",s);
                exit();
            }
            else        /* defined in the j-th model line */
```

```
            {
                if ((k=find_index("(",moda[j]))>0)
                {   /* replace the model_name by its definition */
                    strcpy(s,moda[j]);
                    strdel(s,0,k);
                    (branch+i)->relation=calloc(strlen(s)+1,sizeof(char));
                    strcpy((branch+i)->relation,s);
                }
                else
                {
                    printf("MISSING '(' IN THE MODEL LINE\n");
                    printf("%s\n",moda[j]);
                    exit();
                }
            }
        }
    }
}

/*************************************************************************/
/* parse each element line and store the element information on the      */
/* structure "branch" with the following subfields : (i) element name    */
/* (ii) node number (iii) element relation (iv) port number ( 1 for 1-port */
/* or 2-terminal element; 2 for 2-port element ).                        */
/*************************************************************************/

store_branch(bst,bn,branch)
char *bst[];
int bn;
struct INLINE branch[];
{
    int i;
    char px[80],ch,*calloc();

    for (i=0;i<bn;i++,branch++)
    {
        find_word(bst[i],branch->name,1);       /* branch name  */
        find_word(bst[i],px,2);
        branch->node1=atoi(px);                 /* 1st node     */
        find_word(bst[i],px,3);
        branch->node2=atoi(px);                 /* 2nd node     */
        ch=branch->name[0];                     /* element type */

        switch (ch)
        {
            case 'R' :
            case 'L' :
            case 'C' :
            case 'V' :
            case 'I' : one_port(bst[i],branch); break;
            case 'E' :
            case 'F' :
            case 'G' :
            case 'H' : lin_2_port(bst[i],branch); break;
            case 'K' :
            case 'N' : multi_port(bst[i],branch); break;
```

```
            default  : {
                            printf("UNDEFINED ELEMENT TYPE '%c'\n",ch);
                            exit();
                        }
        }
        mbran+=branch->port;
    }
}

/*************************************************************************/
/* Get the branch information for the linear 1-port (2-terminal) elements. */
/*************************************************************************/

one_port(line,p)
char *line;
struct INLINE *p;
{
    char px[80],*clloc();

    find_word(line,px,4);       /* element relation */
    if ((p->relation=calloc(strlen(px)+1,sizeof(char)))==NULL)
        exit_message("CAN'T ALLOCATE SPACE");
    strcpy(p->relation,px);
    p->port=1;                  /* port # = 1 */
}

/*************************************************************************/
/* Get the branch information for the linear 2-port elements.            */
/*************************************************************************/

lin_2_port(line,p)
char *line;
struct INLINE *p;
{
    char px[80],*calloc();

    find_word(line,px,4);
    p->node3=atoi(px);          /* 3rd node */
    find_word(line,px,5);
    p->node4=atoi(px);          /* 4th node */
    find_word(line,px,6);
    if ((p->relation=calloc(strlen(px)+1,sizeof(char)))==NULL)
        exit_message("CAN'T ALLOCATE SPACE");
    strcpy(p->relation,px);
    p->port=2;                  /* port # = 2 */
}

/*************************************************************************/
/* Get the branch information for the nonlinear multi-port elements.     */
/*************************************************************************/

multi_port(line,p)
char *line;
struct INLINE *p;
{
    char px[80],py[80],*calloc();
```

```
    find_word(line,px,4);
    p->node3=atoi(px);          /* 3rd node */
    find_word(line,px,5);
    p->node4=atoi(px);          /* 4th node */
    find_word(line,px,6);
    if (*px=='(' || *px>'9' || *px<'0')
    {
        /* nonlinear 2-port */
        find_word(line,py,6);
        if ((p->relation=calloc(strlen(py)+1,sizeof(char)))==NULL)
            exit_message("CAN'T ALLOCATE SPACE");
        strcpy(p->relation,py);
        p->port=2;              /* port # = 2 */
    }
    else
    {   /* port # >= 3 */
        p->node5=atoi(px);      /* 5th node */
        find_word(line,px,7);
        p->node6=atoi(px);      /* 6th node */
        find_word(line,px,8);
        if (*px=='(' || *px>'9' || *px<'0')
        {
            /* nonlinear 3-port */
            find_word(line,py,8);
            p->relation=calloc(strlen(py)+1,sizeof(char));
            strcpy(p->relation,py);
            p->port=3;          /* port # = 3 */
        }
        else
        {   /* port # >= 4 */
            p->node7=atoi(px);  /* 7th node */
            find_word(line,px,9);
            p->node8=atoi(px);  /* 8th node */
            find_word(line,py,10);
            p->relation=calloc(strlen(py)+1,sizeof(char));
            strcpy(p->relation,py);
            p->port=4;          /* port # = 4 */
        }
    }
}

/*************************************************************************/
/* read the spice file and store each input line in the array "st".      */
/*************************************************************************/

read_file(fp,n)
FILE *fp;
int n;
{
    int j,k,l;
    char line[81],*p,*calloc();

    l=0;
    st=(STRING *)calloc(n,sizeof(STRING));
    while (fgets(line,80,fp)!=NULL)
```

```
    {
        if (line[0]=='*')       /* comment line */
            continue;

        /* get rid of unnecessary spaces in the tail of the input line */
        j=strlen(line);
        while (--j>=0)
            if (line[j]!=' ' && line[j]!='\t' && line[j]!='\n')
                break;
        line[j+1]='\0';

        /* append the line starting with '$' to the previous line */
        if (line[0]=='$')
        {
            strdel(line,0,1);
            if ((p=calloc(strlen(line)+k+1,sizeof(char)))==NULL)
                exit_message("CAN'T ALLOCATE SPACE");
            strcpy(p,st[--l]);
            strcat(p,line);
            free(st[l]);
            st[l]=p;
        }

        else
        {
            if ((st[l]=calloc(strlen(line)+1,sizeof(char)))==NULL || l>n)
                exit_message("INSUFFICIENT SPACE FOR INPUT LINE ARRAY");
            strcpy(st[l],line);
        }
        k=strlen(st[l]);
        if (find_index(".end",st[l])==0 || find_index(".END",st[l])==0)
            break;      /* last line is reached */
        l++;
    }
    if (l==0)
        exit_message("EMPTY INPUT FILE");
    if (find_index(".end",st[l])!=0 && find_index(".END",st[l])!=0)
        exit_message("MISSING .END IN THE LAST LINE");
    return(l);          /* l = total # of input lines in the spice */
                        /* file (excluding the comment lines and    */
                        /* the last line ".end")                    */
}

/*************************************************************************/
/* partition each input line in the spice file to the class of (i) element */
/* lines (ii) model/graphic control/comment/declaration lines.            */
/*************************************************************************/

branch_model(bst,moda,bp,mp)
char *bst[],*moda[];
int *bp,*mp;
{
    int i;
    char *p,*calloc();

    *bp=0;
```

```
    *mp=0;
    for (i=0;i<ln;i++)
    {
        if((p=calloc(strlen(st[i])+1,sizeof(char)))==NULL)
            exit_message("CAN'T ALLOCATE SPACE");
        strcpy(p,st[i]);
        if (st[i][0]=='.')
            moda[(*mp)++]=p;
        else bst[(*bp)++]=p;
    }
}

/*************************************************************************/
/* Preliminary allocation for the incidence matrix A and the node vector.  */
/*************************************************************************/

pre_alloc(A,node_vector)
int **A,**node_vector;
{
    calloci(NODE*mbran,A,"A");
    calloci(NODE,node_vector,"node_vector");
}
```

```
#include <stdio.h>
#include "nport.h"

/* see variable definitions in file "nport1.c" */
extern int *D_ay,*D_ab,*D_zb,*D_zy;
extern int mbran;

/*************************************************************************/
/* Find the incidence matrix A and choose a tree by reducing A to the     */
/* row-echelon form.                                                       */
/*************************************************************************/

incidence_matrix(branch,A,branch_vector,node_vector,node_number,
                 branch_number,bn)

struct INLINE branch[];
struct B_VECTOR branch_vector[];
int **A,**node_vector,*node_number,*branch_number,bn;

{
    int i,j;
    struct B_VECTOR *pv;

    *branch_number=0;
    *node_number=0;
    pv=branch_vector;
    for (i=0;i<bn;i++,branch++)
    {
        /* find incidence matrix A and node_vector */
        assign_node(branch,*A,*node_vector,node_number,branch_number);

        if (NODE<*node_number)
        {
            printf("NODE NUMBER > %d\n",NODE);
            exit();
        }

        /* find the branch_vector which stores the index */
        /* for each branch within the spice file         */
        for (j=1;j<=branch->port;j++)
        {
            pv->a=i;    /* the branch is in the i-th element */
                        /* line of the spice file            */
            pv->b=j;    /* the branch is in the j-th port of */
                        /* the element                       */
            pv++;
        }
    }

    /* re-allocate spaces for A and node_vector */
    new_alloc(A,node_vector,*branch_number,*node_number);

    pv=branch_vector;

    /* reduce A to the row-echelon form and choose a tree */
    r_echlon(*A,pv,*node_number,*branch_number);
```

```
}


/*************************************************************************/
/* Reduce A to the row-echelon form.                                     */
/*************************************************************************/

r_echlon(A,pv,node,bran)
int *A;         /* the incidence matrix with ij-th entry expressed as */
                /*              A[i*bran+j]                           */
                /* where bran is the total # of branches              */
struct B_VECTOR *pv;
int node,bran;

{
    int c,i,j,k,m;

    for (i=0;i<(node-1);i++)
    {
        /* choose a nonzero pivot */
        if (A[i*bran+i]==0)
        {
            for (k=i;k<(node-1);k++)
                if (A[k*bran+i]!=0) break;
            if (k==(node-1))
            {
                /* no other nonzero element in the same column */

                /* find a nonzero element in the same row      */
                for (j=i+1;j<bran && A[i*bran+j]==0;j++);

                if (j>=bran)
                    exit_message("ZERO ROW IN THE INCIDENCE MATRIX");

                /* column interchange */
                for (m=0;m<(node-1);m++)
                    switch_i(&A[m*bran+i],&A[m*bran+j]);
                switch_i(&((pv+i)->a),&((pv+j)->a));
                switch_i(&((pv+i)->b),&((pv+j)->b));
            }
            else
                /* row interchange */
                for (m=i;m<bran;m++)
                    switch_i(&A[i*bran+m],&A[k*bran+m]);
        }

        /* choose a positive pivot */
        if (A[i*bran+i]==-1)
            for (m=i;m<bran;m++)
                A[i*bran+m] = -1*A[i*bran+m];

        /* elementary row operations */
        for (k=0;k<(node-1);k++)
            if ((k!=i) && (A[k*bran+i]!=0))
        {
            c=A[k*bran+i];
```

```
            for (m=i;m<bran;m++)
                A[k*bran+m]=A[k*bran+m]-c*A[i*bran+m];
        }
    }
}

/*************************************************************************/
/* Assign 1 or -1 to the incidence matrix for an element and fill the     */
/* corresponding nodes to node_vector.                                    */
/*************************************************************************/

assign_node(pb,A,node_vector,node,bran)
struct INLINE *pb;
int *A,*node_vector,*node,*bran;
{
    /* process the 1st branch */
    s_node(A,node_vector,pb->node1,pb->node2,node,bran);

    /* when port # > 1, process the 2nd branch */
    if (pb->port>1)
        s_node(A,node_vector,pb->node3,pb->node4,node,bran);

    /* when port # > 2, process the 3rd branch */
    if (pb->port>2)
        s_node(A,node_vector,pb->node5,pb->node6,node,bran);

    /* when port # > 3, process the 4th branch */
    if (pb->port>3)
        s_node(A,node_vector,pb->node7,pb->node8,node,bran);
}

/*************************************************************************/
/* Assign 1 or -1 to the incidence matrix A for a branch connected across */
/* two nodes with node number pnode and nnode.                            */
/*************************************************************************/

s_node(A,node_vector,pnode,nnode,nn,bn)

int *A;         /* incidence matrix, the ij-th entry is expressed */
                /* as A[i*mbran+j]                                */
int *node_vector,pnode,nnode,*nn,*bn;
{
    int k,nx[2],i;

    nx[0]=pnode;
    nx[1]=nnode;
    for (i=0;i<2;i++)
    {
        /* search node_vector to determine whether the present node */
        /* has been processed and stored in node_vector             */
        for (k=0;k<*nn;k++)
            if (node_vector[k]==nx[i]) break;

        /* a new node */
        if (k==*nn)
        {
```

```
            node_vector[*nn]=nx[i];
            (*nn)++;    /* increment the node # */
        }

        A[k*mbran+(*bn)]=(i==0)? 1:-1;  /* fill 1 or -1 to A */
    }
    (*bn)++;            /* increment the branch # */
}

/*************************************************************************/
/* Re-allocate spaces for A and node_vector for a more exact and compact   */
/* allocation when total # of nodes and branches are found.                */
/*************************************************************************/

new_alloc(A,node_vector,bran,node)
int **A;        /* pointer to the incidence matrix, the ij-th entry is */
                /*      (*)A[i*mbran+j] ( before new allocation )       */
                /*      (*)A[i*bran+j] ( after  new allocation )       */
                /* where bran is the total # of branches               */
int **node_vector,bran,node;
{
    int *new,i,j;

    /* re-allocate space for A */
    calloci(node*bran,&new,"new");
    for (i=0;i<node;i++)
        for (j=0;j<bran;j++)
            new[i*bran+j]=(*A)[i*mbran+j];
    cfree(*A);
    *A=new;

    /* re-allocate space for node_vector */
    calloci(node,&new,"new");
    for (i=0;i<node;i++)
        new[i]=(*node_vector)[i];
    cfree(*node_vector);
    *node_vector=new;
}

/*************************************************************************/
/* Partition each branch into (i) port branch in the tree ( # = tp )       */
/* (ii) nonport branch in the cotree ( # = tn ) (iii) port branch in the   */
/* cotree ( # = cp ) (iv) nonport branch in the cotree ( # = cn ).         */
/*************************************************************************/

tree_port(branch,A,branch_vector,bran,node,
          tp,tn,cp,cn)
struct INLINE branch[];
struct B_VECTOR branch_vector[];
int *A,node,bran,*tp,*tn,*cp,*cn;

{
    int *t;

    calloci(mbran,&t,"t");
    /* partition branches into port and nonport branches */
```

```
    sep_port(branch,branch_vector,t,bran);

    /* partition the tree branches */
    tree_sep(A,branch_vector,t,node,bran,tp);

    *tn=node-1-*tp;     /* # of nonport branches in the tree */

    /* partition the cotree branches */
    link_sep(A,branch_vector,t,node,bran,cp);

    *cn=bran-*tp-*tn-*cp;       /* # of nonport branches in the cotree */

    /* find the fundamental cutset matrix */
    d_matrix(A,*tp,*cp,*tn,*cn,bran,node);
}

/*************************************************************************/
/* Find the matrices (i) D_ab (ii) D_ay (iii) D_zb (iv) D_zy in the      */
/* fundamental cutset matrix Eq.(6.55) in Chua and Lin's book            */
/*              /                                  \                     */
/*              |  I_aa   0_az   D_ay   D_ab  |                          */
/*              |                              |                          */
/*              |  0_za   I_zz   D_zy   D_zb  |                          */
/*              \                              /                          */
/* where        a = port branches in the tree                            */
/*              b = port branches in the cotree                          */
/*              z = nonport branches in the tree                         */
/*              y = nonport branches in the cotree                       */
/*************************************************************************/

/* the ij-th entry in the matrices (i) A (ii) D_ab (iii) D_ay (iv) D_zb  */
/* (v) D_zy is expressed as                                              */
/*                                                                       */
/* (i)          A[i*bran+j]                                              */
/* (ii)         D_ab[i*cp+j]                                             */
/* (iii)        D_ay[i*cn+j]                                             */
/* (iv)         D_zb[i*cp+j]                                             */
/* (v)          D_zy[i*cn+j]                                             */

d_matrix(A,tp,cp,tn,cn,bran,node)
int *A;
int tp,cp,tn,cn,node;

{
    int i,j;

    calloci(tp*cp,&D_ab,"D_ab");
    calloci(tp*cn,&D_ay,"D_ay");
    calloci(tn*cn,&D_zy,"D_zy");
    calloci(tn*cp,&D_zb,"D_zb");

    for (i=0;i<tp;i++)
    {
        for (j=0;j<cp;j++)
            D_ab[i*cp+j]=A[i*bran+j+node-1];
        for (j=0;j<cn;j++)
```

```
            D_ay[i*cn+j]=A[i*bran+j+node-1+cp];
    }
    for (i=0;i<tn;i++)
    {
        for (j=0;j<cp;j++)
            D_zb[i*cp+j]=A[(i+tp)*bran+j+node-1];
        for (j=0;j<cn;j++)
            D_zy[i*cn+j]=A[(i+tp)*bran+j+node-1+cp];
    }
}

/*************************************************************************/
/* Partition link branches into (i) port branches (ii) nonport branches    */
/* and re-arrange them such that the first cp branches in the cotree are   */
/* port branches.                                                          */
/*************************************************************************/

link_sep(A,branch_vector,t,node,bran,cp)
int *A,node,bran,t[],*cp;
struct B_VECTOR branch_vector[];

{
    int i,j,k;
    struct B_VECTOR *pv;

    pv=branch_vector;
    *cp=0;

    /* if the i-th branch in the cotree is a nonport branch,   */
    /* then interchange with a port branch with a larger index */
    for (i=node-1;i<bran;i++)
    {
        if (t[i]==0)
        {
            k=bran-1;
            while (t[k]==0 && i<k)
                k--;
            if (i<k)
            {
                for (j=0;j<(node-1);j++)
                    switch_i(&A[j*bran+i],&A[j*bran+k]);
                switch_i(&((pv+i)->a),&((pv+k)->a));
                switch_i(&((pv+i)->b),&((pv+k)->b));
                t[k]=0;
                t[i]=1;
                (*cp)++;
            }
        }
        else (*cp)++;
    }
}

/*************************************************************************/
/* Partition tree branches into (i) port branches (ii) nonport branches    */
/* and re-arrange them such that the first tp branches in the tree are     */
/* the port branches and the others are the nonport branches.              */
```

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

tree_sep(A,branch_vector,t,node,bran,tp)
int *A,node,bran,t[],*tp;
struct B_VECTOR branch_vector[];

{
    int i,j,k;
    struct B_VECTOR *pv;

    pv=branch_vector;
    *tp=0;
    for (i=0;i<(node-1);i++)
    {
        /* if the i-th branch in the tree is a nonport branch,   */
        /* then interchange with a port branch with larger index */
        if (t[i]==0)
        {
            k=node-2;
            while (t[k]==0 && k>i)
                k--;
            if (k>i)
            {
                switch_i(&((pv+i)->a),&((pv+k)->a));
                switch_i(&((pv+i)->b),&((pv+k)->b));
                for (j=(node-1);j<bran;j++)
                    switch_i(&A[i*bran+j],&A[k*bran+j]);
                t[k]=0;
                t[i]=1;
                (*tp)++;
            }
        }
        else (*tp)++;
    }
}

/*************************************************************************/
/* partition each branch to classes of (i) port branches (non_R, C, L, V   */
/* and I ) (ii) nonport branches (lin_R, E, F, G, H).                      */
/*************************************************************************/

sep_port(branch,branch_vector,t,bran)
struct INLINE branch[];
struct B_VECTOR branch_vector[];
int t[];        /* i-th branch is a port (resp.; nonport) branch */
                /* if t[i]=1 (resp.; t[i]=0)                     */
int bran;

{
    int i,j;
    struct B_VECTOR *pv;
    char ch,cg;

    pv=branch_vector;
    for (i=0;i<bran;i++,pv++)
    {
```

```
        j=pv->a;                    /* branch index in the element lines */
        ch=branch[j].name[0];       /* element type */
        cg=branch[j].relation[0]; /* characteristic type : linear or   */
                                    /* nonlinear                         */
        t[i]=1;
        switch(ch)
        {
            case 'E' :
            case 'F' :
            case 'G' :
            case 'H' : t[i]=0; break;
            case 'R' : if (cg!='(') t[i]=0; break;
            default  : t[i]=1; break;
        }
    }
}
```

```
#include <stdio.h>
#include "nport.h"

/* see variable definitions in the file "nport1.c" */
extern int *D_ab,*D_ay,*D_zb,*D_zy;
extern double *F_iz,*F_vy,*F_iy,*F_vz;

/*************************************************************************/
/* Find the linear characterization for the linear elements in the tree.   */
/*************************************************************************/

l_tree(branch,branch_vector,bran,node,tp,tn,cp,cn)
struct INLINE branch[];
struct B_VECTOR branch_vector[];
int bran,node,tp,tn,cp,cn;

{
    struct INLINE *bv;
    struct B_VECTOR *pv;
    int i,j,k;
    char *d,ch,*calloc();
    double x,stof();

    /* allocate spaces for the matrices (i) F_iz (ii) F_vz    */
    /* (iii) F_iy (iv) F_vy whose ij-th entry is expressed as */
    /*          F_iz[i*tn+j]                                  */
    /*          F_vz[i*tn+j]                                  */
    /*          F_iy[i*cn+j]                                  */
    /*          F_vy[i*cn+j]                                  */
    /*                                                        */
    callocd((tn+cn)*tn,&F_iz,"F_iz");
    callocd((tn+cn)*tn,&F_vz,"F_vz");
    callocd((tn+cn)*cn,&F_iy,"F_iy");
    callocd((tn+cn)*cn,&F_vy,"F_vy");
    bv=branch;
    pv=branch_vector;
    for (i=tp;i<(node-1);i++)
    {
        j=(pv+i)->a;            /* the branch is in the j-th element line */
        ch=(bv+j)->name[0];     /* element type */
        if ((d=calloc(strlen((bv+j)->relation)+1,sizeof(char)))==NULL)
            exit_message("CAN'T ALLOCATE SPACE");
        strcpy(d,(bv+j)->relation);
        x=stof(d);              /* value for the linear characteristic */
        switch(ch)
        {
            case 'R' : {   /* linear resistor */
                           F_iz[(i-tp)*tn+i-tp] = x;
                           F_vz[(i-tp)*tn+i-tp] = -1;
                           break;
                       }
            case 'E' : {   /* linear voltage-controlled voltage source */
                           if ((pv+i)->b!=1)    /* controlled branch */
                               F_iz[(i-tp)*tn+i-tp]=1;
                           else                 /* controlling branch */
                           {
```

```
                                F_vz[(i-tp)*tn+i-tp] = -1;
                                k=search(i,j,branch_vector,bran);
                                if (k<(node-1)) F_vz[(i-tp)*tn+k-tp]=x;
                                else F_vy[(i-tp)*cn+k-cp-node+1]=x;
                            }
                            break;
                        }
            case 'F' : {    /* linear current-controlled current source */
                            if ((pv+i)->b!=1)    /* controlled branch */
                                F_vz[(i-tp)*tn+i-tp]=1;
                            else                 /* controlling branch */
                            {
                                F_iz[(i-tp)*tn+i-tp] = -1;
                                k=search(i,j,branch_vector,bran);
                                if (k<(node-1)) F_iz[(i-tp)*tn+k-tp]=x;
                                else F_iy[(i-tp)*cn+k-cp-node+1]=x;
                            }
                            break;
                        }
            case 'G' : {    /* voltage-controlled current source */
                            if ((pv+i)->b!=1)    /* controlled branch */
                                F_iz[(i-tp)*tn+i-tp]=1;
                            else                 /* controlling branch */
                            {
                                F_iz[(i-tp)*tn+i-tp] = -1;
                                k=search(i,j,branch_vector,bran);
                                if (k<(node-1)) F_vz[(i-tp)*tn+k-tp]=x;
                                else F_vy[(i-tp)*cn+k-node-cp+1]=x;
                            }
                            break;
                        }
            case 'H' : {    /* current-controlled voltage source */
                            if ((pv+i)->b!=1)    /* controlled branch */
                                F_vz[(i-tp)*tn+i-tp]=1;
                            else                 /* controlling branch */
                            {
                                F_vz[(i-tp)*tn+i-tp] = -1;
                                k=search(i,j,branch_vector,bran);
                                if (k<(node-1)) F_iz[(i-tp)*tn+k-tp]=x;
                                else F_iy[(i-tp)*cn+k-node-cp+1]=x;
                            }
                            break;
                        }
        }
        cfree(d);
    }
}

/*************************************************************************/
/* Find the linear characterization of the linear elements in the cotree. */
/*************************************************************************/

l_cotree(branch,branch_vector,bran,node,tp,tn,cp,cn)
struct INLINE branch[];
struct B_VECTOR branch_vector[];
int bran,node,tp,tn,cp,cn;
```

```
{
    int i,j,k;
    struct INLINE *bv;
    struct B_VECTOR *pv;
    char *d,ch;
    double x,stof();

    bv=branch;
    pv=branch_vector;
    for (i=node+cp-1;i<bran;i++)
    {
        j=(pv+i)->a;            /* j-th element line */
        ch=(bv+j)->name[0];     /* element type */
        if ((d=calloc(strlen((bv+j)->relation)+1,sizeof(char)))==NULL)
            exit_message("CAN'T ALLOCATE SPACE");
        strcpy(d,(bv+j)->relation);
        x=stof(d);              /* value for the linear characteristic */
        switch(ch)
        {
            case 'R' : {   /* linear resistor */
                           F_iy[(i-tp-cp)*cn+i-tp-cp-tn]=x;
                           F_vy[(i-tp-cp)*cn+i-tp-cp-tn] = -1;
                           break;
                       }
            case 'E' : {   /* linear voltage-controlled voltage source */
                           if ((pv+i)->b!=1)    /* controlled branch */
                               F_iy[(i-tp-cp)*cn+i-tp-cp-tn]=1;
                           else                 /* controlling branch */
                           {
                               F_vy[(i-tp-cp)*cn+i-tp-cp-tn] = -1;
                               k=search(i,j,branch_vector,bran);
                               if (k>=(node-1))
                                   F_vy[(i-tp-cp)*cn+k-tp-cp-tn]=x;
                               else F_vz[(i-tp-cp)*tn+k-tp]=x;
                           }
                           break;
                       }
            case 'F' : {   /* linear currrent-controlled current source */
                           if ((pv+i)->b!=1)    /* controlled branch */
                               F_vy[(i-tp-cp)*cn+i-tp-cp-tn]=1;
                           else                 /* controlling branch */
                           {
                               F_iy[(i-tp-cp)*cn+i-tp-cp-tn] = -1;
                               k=search(i,j,branch_vector,bran);
                               if (k>=(node-1))
                                   F_iy[(i-tp-cp)*cn+k-tp-cp-tn]=x;
                               else F_iz[(i-tp-cp)*tn+k-tp]=x;
                           }
                           break;
                       }
            case 'G' : {   /* linear voltage-controlled current source */
                           if ((pv+i)->b!=1)    /* controlled branch */
                               F_iy[(i-tp-cp)*cn+i-tp-cp-tn]=1;
                           else                 /* controlling branch */
                           {
                               F_iy[(i-tp-cp)*cn+i-tp-cp-tn] = -1;
```

```
                                k=search(i,j,branch_vector,bran);
                                if (k)=(node-1))
                                    F_vy[(i-tp-cp)*cn+k-tp-cp-tn]=x;
                                else F_vz[(i-tp-cp)*tn+k-tp]=x;
                            }
                            break;
                        }
            case 'H' : {    /* linear current-controlled voltage source */
                            if ((pv+i)->b!=1)    /* controlled branch */
                                F_vy[(i-tp-cp)*cn+i-tp-cp-tn]=1;
                            else                 /* controlling branch */
                            {
                                F_vy[(i-tp-cp)*cn+i-tp-cp-tn] = -1;
                                k=search(i,j,branch_vector,bran);
                                if (k)=(node-1))
                                    F_iy[(i-tp-cp)*cn+k-tp-cp-tn]=x;
                                else F_iz[(i-tp-cp)*tn+k-tp]=x;
                            }
                            break;
                        }
        }
        free(d);
    }
}

/*************************************************************************/
/* Search the other branch (k-th branch) of the linear controlled source   */
/* which, together with the i-th branch, is in the j-th element line.      */
/*************************************************************************/

int
search(i,j,branch_vector,bran)
int i,j,bran;
struct B_VECTOR branch_vector[];

{
    int k;

    for (k=0;k<bran;k++)
        if ((branch_vector+k)->a==j && k!=i)
            break;
    if (k>bran)
        exit_message("MISMATCH CONTROLLED SOURCE");
    else
        return(k);
}

/*************************************************************************/
/* Find the linear characterization matrix F for all the linear elements. */
/* (see Chua and Lin's book Eq.(6.67) in p.262)                            */
/*************************************************************************/

F_matrix(F,tp,tn,cp,cn)
double **F;
int tp,tn,cp,cn;
{
```

```
    int l,m;

    /* allocate space for F which is an m by l matrix */
    m=tp+cp+tn+cn;
    l=m+tp+cp;
    callocd(m*l,F,"F");

    /* find the top block of F */
    F_top(tn+cn,l,tp,cp,cn,*F);

    /* find the middle block of F */
    F_middle(tn+cn,l,tp,cp,cn,*F);

    /* find the bottom block of F */
    F_bottom(tn+cn,l,tp,cp,tn,cn,*F);
}

/*************************************************************************/
/* Find the top block of F. (see Chua and Lin's book Eq.(6.67) p.262)    */
/*************************************************************************/

/* the ij-th entry in the matrices (i) F (ii) D_ay (iii) D_ab is expressed */
/* as                                                                      */
/* (i)          F[i*l+j]                                                   */
/* (ii)         D_ay[i*cn+j]                                               */
/* (iii)        D_ab[i*cp+j]                                               */

F_top(n,l,tp,cp,cn,F)
int n,l,tp,cp,cn;
double *F;
{
    int i,j;

    for (i=0;i<tp;i++)
    {
        for (j=0;j<cn;j++)
            F[i*l+j]=D_ay[i*cn+j];
        F[i*l+i+n]=1;
        for (j=n+tp+cp;j<(n+tp+2*cp);j++)
            F[i*l+j]=D_ab[i*cp+j-n-tp-cp];
    }
}

/*************************************************************************/
/* Find the middle bolck of F. (see chua and Lin's book Eq.(6.67) p.262) */
/*************************************************************************/

/* the ij-th entry of the matrices (i) F (ii) D_zb (iii) D_ab is expressed */
/* as                                                                      */
/* (i)          F[i*l+j]                                                   */
/* (ii)         D_zb[i*cp+j]                                               */
/* (iii)        D_ab[i*cp+j]                                               */

F_middle(n,l,tp,cp,cn,F)
int n,l,tp,cp,cn;
double *F;
```

```
{
    int i,j;

    for (i=tp;i<(tp+cp);i++)
    {
        for (j=cn;j<n;j++)
            F[i*l+j] = -D_zb[(j-cn)*cp+i-tp];   /* -transpose(D_zb) */
        F[i*l+i+n]=1;
        for (j=n+tp+2*cp;j<(n+2*(tp+cp));j++)
            F[i*l+j] = -D_ab[(j-(n+tp+2*cp))*cp+i-tp]; /* -transpose(D_ab) */
    }
}

/*************************************************************************/
/* Find the bottom block of F. (see Chua and Lin's book Eq.(6.68) p.262)   */
/*************************************************************************/

/* the ij-th entry of the matrices (i) F (ii) D_zy (iii) D_ay (iv) D_zb    */
/* (v) F_iz (vi) F_iy (vii) F_vz (viii) F_vy is expressed as               */
/*                                                                         */
/* (i)          F[i*l+j]                                                   */
/* (ii)         D_zy[i*cn+j]                                               */
/* (iii)        D_ay[i*cn+j]                                               */
/* (iv)         D_zb[i*cp+j]                                               */
/* (v)          F_iz[i*tn+j]                                               */
/* (vi)         F_iy[i*cn+j]                                               */
/* (vii)        F_vz[i*tn+j]                                               */
/* (viii)       F_vy[i*cn+j]                                               */

F_bottom(n,l,tp,cp,tn,cn,F)
int n,l,tp,cp,tn,cn;
double *F;
{
    int i,j,k;
    double c;

    for (i=(tp+cp);i<(tp+cp+n);i++)
    {
        for (j=0;j<cn;j++)
        {
            c=0;
            for (k=0;k<tn;k++)
                c=c+F_iz[(i-tp-cp)*tn+k]*D_zy[k*cn+j];
            F[i*l+j]=F_iy[(i-tp-cp)*cn+j]-c;
        }
        for (j=cn;j<n;j++)
        {
            c=0;
            for (k=0;k<cn;k++)
                c=c+F_vy[(i-tp-cp)*cn+k]*D_zy[(j-cn)*cn+k];
            F[i*l+j]=F_vz[(i-tp-cp)*tn+j-cn]+c;
        }
        for (j=n;j<(n+tp);j++)
        {
            c=0;
            for (k=0;k<cn;k++)
```

```
                c=c+F_vy[(i-tp-cp)*cn+k]*D_ay[(j-n)*cn+k];
            F[i*l+j+2*cp+tp]=c;
        }
        for (j=n+tp;j<n+tp+cp;j++)
        {
            c=0;
            for (k=0;k<tn;k++)
                c=c+F_iz[(i-tp-cp)*tn+k]*D_zb[k*cp+j-n-tp];
            F[i*l+j+cp] = -1*c;
        }
    }
}

/*************************************************************************/
/* Reduce F to yield the constraint matrix equation for the linear       */
/* elements. (see Chua and Lin's book, Eq.(6.69) p.262)                  */
/*************************************************************************/

F_reduce(F,P,Q,bv,tp,tn,cp,cn,mx,px)
double *F,**P,**Q;
int tp,tn,cp,cn,*mx,*px;
struct B_VECTOR bv[];
{
    int i=0,j=0,n;

    *mx=tp+cp+tn+cn;
    n=tn+cn+2*(tp+cp);
    r_ech_F(&i,&j,mx,n,tn,cn,F);
    callocd((tp+cp)*(tp+cp),P,"P");
    callocd((tp+cp)*(tp+cp),Q,"Q");
    P_and_Q(i,j,n,mx,tp,cp,F,*P,*Q);
    *mx = *mx-i;
    *px=tp+cp;
    if ((*mx)!=(*px))
        exit_message("DEGENERATE CIRCUIT");
    for (i=tp;i<tp+cp;i++)
    {
        (bv+i)->a=(bv+i+tn)->a;
        (bv+i)->b=(bv+i+tn)->b;
    }
    cfree(F);
}

/*************************************************************************/
/* Reduce F to the row-echelon form as Eq.(6.69) in Chua and Lin's book  */
/* and return the indices ii and jj which are the row and column indices */
/* for the (0,0) entry of the constraint matrix in the reduced F in      */
/* Eq.(6.69).                                                            */
/*************************************************************************/

/* the ij-th entry of the matrix is expressed as */
/*                                               */
/*              F[i*n+j]                         */

r_ech_F(ii,jj,mx,n,tn,cn,F)
int *ii,*jj,*mx,n,tn,cn;
```

```
double *F;
{
    int i,j,k,l;
    double c;

    i = *ii;
    j = *jj;
    while ((i<*mx) && (j<(cn+tn)))
    {
        if (F[i*n+j]==0)
        {
            /* if zero pivot, then choose a nonzero one from */
            /* the same column by row interchange            */
            k=i+1;
            while (k<*mx && F[k*n+j]==0)
                k++;
            if (k<*mx)
                for (l=j;l<n;l++)
                    switch_d(&F[i*n+l],&F[k*n+l]);
        }
        if (F[i*n+j]!=0)
        {
            /* elementary row operation */
            for (k=0;k<*mx;k++)
                if ((k!=i) && (F[k*n+j]!=0))
                {
                    c=F[k*n+j];
                    for (l=j;l<n;l++)
                        F[k*n+l]=F[k*n+l]-F[i*n+l]*c/F[i*n+j];
                }
            i++;
            j++;
        } else j++;
    }
    *ii=i;
    *jj=j;
}

/*************************************************************************/
/* Find the constraint matrix equation for the linear n-port             */
/*              P*v + Q*i =.0                                             */
/* ( the right-bottom corner in Eq.(6.69))                               */
/*************************************************************************/

/* the ij-th entry in the matrices (i) F (ii) P (iii) Q is expressed as  */
/*                                                                       */
/* (i)          F[i*l+j]                                                 */
/* (ii)         P[i*(tp+cp)+j]                                           */
/* (iii)        Q[i*(tp+cp)+j]                                           */

P_and_Q(i,j,l,mx,tp,cp,F,P,Q)
int i,j,l,*mx,tp,cp;
double *F,*P,*Q;
{
    int ix,jx;
```

```
    for (ix=i;ix<*mx;ix++)
    {
        for (jx=j;jx<(j+tp);jx++)
        {
            Q[(ix-i)*(tp+cp)+jx-j]=F[ix*l+jx];
            P[(ix-i)*(tp+cp)+jx-j]=F[ix*l+jx+tp+2*cp];
        }
        for (jx=(j+tp);jx<(j+tp+cp);jx++)
        {
            P[(ix-i)*(tp+cp)+jx-j]=F[ix*l+jx];
            Q[(ix-i)*(tp+cp)+jx-j]=F[ix*l+jx+cp];
        }
    }
}

/*************************************************************************/
/* Remove the independent voltage and current sources from the constrained */
/* matrix equation      P*v + Q*i = 0                                     */
/* by substituting the source values into the equation to yield a new     */
/* equation (called generalized hybrid equation)                          */
/*              P'*v + Q'*i + s' = 0                                      */
/* with smaller dimension. (differ by the # of independent sources)       */
/*************************************************************************/

rm_source(branch,PP,QQ,ss,bv,mx,px)
struct INLINE branch[];
double **PP,**QQ,**ss;
struct B_VECTOR bv[];
int *mx,*px;

{
    int i=0,j,m;
    double *P,*Q,*s;
    double x,stof();
    char ch,*d,*calloc();

    P = *PP;
    Q = *QQ;
    s = *ss;
    m = *mx;
    callocd(m,&s,"s");
    while (i<*px)
    {
        j=(bv+i)->a;
        if ((d=calloc(strlen((branch+j)->relation)+1,sizeof(char)))==NULL)
            exit_message("CAN'T ALLOCATE SPACE");
        strcpy(d,(branch+j)->relation);
        ch=(branch+j)->name[0];
        if (((ch=='V') || (ch=='I')) && (d[0]!='('))
        {
            /* time-invariant independent voltage or current source */
            x=stof(d);          /* source value */

            /* choose a nonzero element in the column as a pivot */
            nonzero_pivot(ch,i,mx,px,m,P,Q,s);
```

```
            /* reduce the P or Q matrix such that there is only one */
            /* nonzero element in the column which corresponds to   */
            /* the variable of the independent source               */
            reduce(ch,i,mx,px,m,P,Q,s);

            /* re-arrange P, Q, and s for a more compact */
            /* constrained equation                      */
            re_arrange(ch,i,mx,px,m,bv,x,P,Q,s);
            (*mx)--;
            (*px)--;

        }
        else i++;
        free(d);
    }

    /* re-allocate spacs for P, Q, and s for a more compact allocation */
    new_P_Q_s(m,mx,P,Q,s,PP,QQ,ss);
}

/*************************************************************************/
/* If the top entry of the column corresponding to the independent source  */
/* variable is zero, then perform row interchange to yield a nonzero entry */
/* in the top position of that column.                                     */
/*************************************************************************/

nonzero_pivot(ch,i,mx,px,m,P,Q,s)
char ch;
int i,*mx,*px,m;
double *P,*Q,*s;

/* the ij-th entry in the matrices (i) P (ii) Q is expressed as */
/*                                                              */
/* (i)          P[i*m+j]                                        */
/* (ii)         Q[i*m+j]                                        */

{
    int k,l;
    double fabs();

    if ((Q[0*m+i]==0 && ch=='V') || (P[0*m+i]==0 && ch=='I'))
    {
        k=1;
        if (ch=='V')
            while (fabs(Q[k*m+i])<=1.0e-14 && k<(*mx) k++;
        else
            while (fabs(P[k*m+i])<=1.0e-14 && k<(*mx) k++;
        if (k<(*mx)
        {
            for (l=0;l<(*px;l++)
            {
                switch_d(&P[0*m+l],&P[k*m+l]);
                switch_d(&Q[0*m+l],&Q[k*m+l]);
            }
            switch_d(s,s+k);
        }
```

```
    }
}

/*************************************************************************/
/* Reduce the column vector of P or Q which corresponds to the independent */
/* source variable such that each entry except the top one in that column  */
/* is zero.                                                                */
/*************************************************************************/

/* the ij-th entry in the matrices (i) P (ii) Q is expressed as */
/* (i)          P[i*m+j]                                        */
/* (ii)         Q[i*m+j]                                        */

reduce(ch,i,mx,px,m,P,Q,s)
char ch;
int i,*mx,*px,m;
double *P,*Q,*s;
{
    int k,l;
    double c,c1;

    if ((Q[0*m+i]!=0 && ch=='V') || (P[0*m+i]!=0 && ch=='I'))
    {
        /* reduce the column of P (resp.; Q) if current */
        /* (resp.; voltage) source                      */
        if (ch=='V') c1=Q[0*m+i];
        else c1=P[0*m+i];
        for (k=1;k<*mx;k++)
            if ((Q[k*m+i]!=0 && ch=='V') || (P[k*m+i]!=0 && ch=='I'))
            {
                if (ch=='V') c=Q[k*m+i];
                else c=P[k*m+i];
                for (l=0;l<*px;l++)
                {
                    Q[k*m+l]=Q[k*m+l]-c*Q[0*m+l]/c1;
                    P[k*m+l]=P[k*m+l]-c*P[0*m+l]/c1;
                }
                s[k]=s[k]-c*s[0]/c1;
            }
    }
}

/*************************************************************************/
/* Get rid of the independent source variable in the constrained equation  */
/* and yield the generalized hybrid equation with a more compact dimension */
/*************************************************************************/

/* the ij-th entry in the matrices (i) P (ii) Q is expressed as */
/* (i)          P[i*m+j]                                        */
/* (ii)         Q[i*m+j]                                        */

re_arrange(ch,i,mx,px,m,bv,x,P,Q,s)

char ch;
int i,*mx,*px,m;
struct B_VECTOR bv[];
```

```
double x,*P,*Q,*s;
{
    int k,l;

    for (k=0;k<(*mx)-1;k++)
    {
        if (ch=='V')
            s[k]=s[k+1]+x*P[(k+1)*m+i];
        else
            s[k]=s[k+1]+x*Q[(k+1)*m+i];
        for (l=0;l<i;l++)
        {
            P[k*m+l]=P[(k+1)*m+l];
            Q[k*m+l]=Q[(k+1)*m+l];
        }
        for (l=i;l<(*px)-1;l++)
        {
            P[k*m+l]=P[(k+1)*m+l+1];
            Q[k*m+l]=Q[(k+1)*m+l+1];
        }
    }
    for (l=i;l<(*px)-1;l++)
    {
        (bv+l)->a=(bv+l+1)->a;
        (bv+l)->b=(bv+l+1)->b;
    }
}

/*************************************************************************/
/* Re-allocate spaces for the hybrid parameters P, Q, and s which need    */
/* less space than originally allocated.                                  */
/*************************************************************************/

/* the ij-th entry in the matrices (i) P (ii) Q is expressed as */
/* (i)          P[i*(*mx)+j]                                    */
/* (ii)         Q[i*(*mx)+j]                                    */

new_P_Q_s(m,mx,P,Q,s,PP,QQ,ss)
int m,*mx;
double *P,*Q,*s,**PP,**QQ,**ss;
{
    int j,k;
    double *Px,*Qx;

    /* re-allocate s */
    rallocd(&s,*mx,m);
    *ss=s;

    /* re-allocate P and Q */
    callocd((*mx)*(*mx),&Px,"Px");
    callocd((*mx)*(*mx),&Qx,"Qx");
    for (j=0;j<(*mx);j++)
        for (k=0;k<(*mx);k++)
        {
            /* move data of P and Q to the new allocated area */
            Px[j*(*mx)+k]=P[j*m+k];
```

```
            Qx[j*(*mx)+k]=Q[j*m+k];
        }
    cfree(*PP);
    cfree(*QQ);
    *PP=Px;
    *QQ=Qx;
}
```

```
#include <stdio.h>

/*************************************************************************/
/* Convert a real number expression terminated by a unit character to a    */
/* real number with double precision.                                      */
/*************************************************************************/

double stof(s)
char *s;        /* input string expression */
{
    char *d;    /* number field expression */
    char ch;    /* unit character */
    char *calloc();
    double x,atof();

    d=calloc(strlen(s)+1,sizeof(char));
    ch = *(s+strlen(s)-1);      /* extract the last character */
    if (ch<'0' || ch>'9')       /* if it is a unit character */
    {
        strcpy(d,s);
        strdel(d,strlen(s)-1,1);        /* extract the number field */
        x=atof(d);
        switch(ch)
        {
            case 'K' : x=x*1e3;   break;        /* Kilo */
            case 'M' : x=x*1e6;   break;        /* Mega */
            case 'G' : x=x*1e9;   break;        /* Giga */
            case 'T' : x=x*1e12;  break;        /* Tera */
            case 'm' : x=x*1e-3;  break;        /* milli */
            case 'u' : x=x*1e-6;  break;        /* micro */
            case 'n' : x=x*1e-9;  break;        /* naro */
            case 'p' : x=x*1e-12; break;        /* pico */
            case 'f' : x=x*1e-15; break;        /* femptl */
            default  : {
                            printf("UNDEFINED UNIT CHARACTER %c\n",ch);
                            exit();
                        }
                        break;
        }
    }
    else x=atof(s);
    return(x);
}

/*************************************************************************/
/* Give the sign of a real number x; 1 if x>=0 and -1 if x<0.              */
/*************************************************************************/

sgn(x)
double x;
{
    if (x>=0) return(1);
    else return(-1);
}

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

```
/* Switch two real number.                                                  */
/****************************************************************************/

switch_d(a,b)
double *a,*b;
{
    double c;

    c = *a;
    *a = *b;
    *b = c;
}

/****************************************************************************/
/* Switch two integer number.                                               */
/****************************************************************************/

switch_i(a,b)
int *a,*b;
{
    int c;

    c = *a;
    *a = *b;
    *b = c;
}

/****************************************************************************/
/* Find the i-th word w from the string s.                                  */
/****************************************************************************/

find_word(s,w,i)
char *s,*w;
int i;
{
    int k=1,m=0,n=0,j;
    char *ps;

    j=strlen(s)+1;
    ps=s;       /* starting position in the input string s */
    while (*s==' ' && m++<j)    /* delete leading blanks in s */
        s++;
    if (m<j)
    {
        while (k<i && n++<j)
        {
            /* search the starting position for the i-th word */
            if (*s==' ' && *(s+1)!=' ')
                k++;
            s++;
        }
        while (*s!=' ' && *s!='\t' && *s!='\0' && n<j)
            *w++ = *s++;        /* copy the i-th word to w */
        *w='\0';
    }
    if (m>=j || n>=j)
```

```
    {
        printf("FAIL TO FIND A WORD IN %s\n",ps);
        exit();
    }
}

/*************************************************************************/
/* Find the position of the string t within the string s; -1 is returned   */
/* if t is not found within s.                                             */
/*************************************************************************/

find_index(t,s)
char *s,*t;
{
    int i,j,k;

    for (i=0;s[i]!='\0';i++)
    {
        for (j=i,k=0;t[k]!='\0' && s[j]==t[k];j++,k++);
        if (t[k]=='\0')
            return(i);
    }
    return(-1);
}

/*************************************************************************/
/*. Delete n characters from the n1-th position of string s.              */
/*************************************************************************/

strdel(s,n1,n)
char *s;
int n1,n;
{
    int i,j,k=0;

    j=strlen(s)+1;
    for (i=0;k<=j,s[i+n1+n]!='\0';k++,i++)
        s[i+n1]=s[i+n1+n];
    if (k>j)
    {
        printf("ERROR IN STRDEL IN STRING\n");
        printf("%s\n",s);
        exit();
    }
    else s[i+n1]='\0';
}

/*************************************************************************/
/* Get rid of unnecessary spaces in the string.                            */
/*************************************************************************/

sqeez(s)
char *s;
{
    char *sx,*tx,*t,*calloc();
    int i=0,k;
```

```
    sx=s;
    k=strlen(s);
    t=calloc(k+1,sizeof(int));
    tx=t;
    while (*sx != '\0' && i++<k)
    {
        if (*sx != ' ' && *sx != '\t' && *sx != '\n')
            *tx++ = *sx++;
        else sx++;
    }
    *tx='\0';
    if (i>k)
        exit_message("ERROR IN SQEEZ");
    else
        strcpy(s,t);
}

/*************************************************************************/
/* Print a message and exit.                                             */
/*************************************************************************/

exit_message(message)
char *message;
{
    printf("%s\n",message);
    exit();
}

/*************************************************************************/
/* allocate space for integer.                                           */
/*************************************************************************/

calloci(n,pt,s)
int n,**pt;
char *s;
{
    char *calloc();
    int *p;

    p =(int *) calloc(n,sizeof(int));
    if (p == NULL)
    {
        printf("CAN'T ALLOCATE SPACE FOR %s\n",s);
        exit();
    }
    else
        *pt = p;
}

/*************************************************************************/
/* Allocate space for double.                                            */
/*************************************************************************/

callocd(n,pt,s)
int n;
```

```
double **pt;
char *s;
{
    char *calloc();
    double *p;

    p = (double *) calloc(n,sizeof(double));
    if ( p == NULL )
    {
        printf("CAN'T ALLOCATE SPACE FOR %s\n",s);
        exit();
    }
    else
        *pt = p;
}

/*************************************************************************/
/* Re-allocate oldsize of integers to a new area with newsize spaces.     */
/*************************************************************************/

ralloci(ip,newsize,oldsize)
int **ip,newsize,oldsize;
{
    char *calloc();
    int i,size,*pt;

    /* allocate a new space */
    pt = *ip;
    if ((*ip= (int *) calloc(newsize,sizeof(int)))==NULL)
        exit_message("CAN'T RE_ALLOCATE");

    if (newsize<oldsize)
        size=newsize;
    else
        size=oldsize;

    /* move the data to the new area */
    for (i=0;i<size;i++)
        (*ip)[i]=pt[i];

    cfree(pt);
}

/*************************************************************************/
/* Re-allocate oldsize of doubles to a new area with newsize spaces.      */
/*************************************************************************/

rallocd(dp,newsize,oldsize)
int newsize,oldsize;
double **dp;
{
    char *calloc();
    int i,size;
    double *pt;

    /* allocate the a new space */
```

```
    pt = *dp;
    if ((*dp= (double *) calloc(newsize,sizeof(double)))==NULL)
        exit_message("CAN'T RE_ALLOCATE");

    if (newsize<oldsize) size=newsize;
    else size=oldsize;

    /* move the data to the new area */
    for (i=0;i<size;i++)
        (*dp)[i]=pt[i];

    cfree(pt);
}

/*************************************************************************/
/* Transform a double precision number to its ASCII code.                */
/*************************************************************************/

f_to_ax(x,d)
double x;
char d[];
{
    int i;
    char e[20],e1[8],e2[10];
    double fabs();

    sprintf(e,"%.5e",fabs(x));  /* e = ASCII code of |x| */
    strncpy(e1,e,7);            /* e1 = number field of e */
    e1[7]='\0';

    /* delete unnecessary '0' in e1 */
    for (i=6;i>0;i--)
    {
        if (e1[i]=='0' || e1[i]=='.') e1[i]='\0';
        else break;
    }
    strcpy(e2,e+7);             /* e2 = exponent of e */

    /* delete unnecessary '0' in e2 */
    i=0;
    while (i++<2 && e2[2]=='0')
        strdel(e2,2,1);
    if (strlen(e2)==2) e2[0]='\0';
    if (e2[1]=='+') strdel(e2,1,1);

    /* concate e1 (number) with e2 (exponent) */
    if (x<0)
        sprintf(d,"-%s%s",e1,e2);
    else
        sprintf(d,"%s%s",e1,e2);
}
```

```
typedef struct INLINE
{
   char name[9];
   int port;
   int node1,node2,node3,node4,node5,node6,node7,node8;
   char *relation;
} STI;
typedef struct B_VECTOR
{
   int a,b;
} STJ;
typedef struct CXPORT
{
    int np;
    double *cpr,*cpi,*cqr,*cqi,*csr,*csi;
};
typedef struct PORTDEF
{
    char element[9];
    int type;
    int node1,node2;
    char *relation;
} PD;
typedef char *STRING;
#define NODE 50
#define NELEM 70
#define NMODEL 30
```

```
#include <stdio.h>
#include "cnport.h"
#include "complex.h"

/*************************************************************************/
/* This is the main program of the routine "cnport" which finds the complex */
/* generalized implicit equation                                          */
/*              P(jw)*v + Q(jw)*i + s(jw) = 0                             */
/* for a linear dynamic m-port, which is obtained by extracting all the   */
/* nonlinear elements and time-varying independent sources from the circuit */
/* such that the remaining elements within the m-port are composed of either*/
/* linear resistive or dynamic elements, and time-invariant independent   */
/* sources. It consists of three files : "cnport.c", cnport1.c", "cnport2.c"*/
/* and the library routines "nport.lib", "staeq.lib", "complex.lib",      */
/* and "clnpack.lib".                                                     */
/*************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    struct CXPORT cport;
    struct PORTDEF *pdef1;
    FILE *fp,*hp,*fopen();
    double freq;

    /* open the input spice file "xx...x.spc" and the output equation */
    /* file "xx...x.eq"                                               */
    open_spc_eq(argc,argv,&fp,&hp);

    /* find the complex implicit linear dynamic n-port equation */
    cn_port(fp,hp,freq,&cport,&pdef1);

    fclose(fp);
    fclose(hp);
}

/*************************************************************************/
/* print the complex n-port equation on the screen and the output file    */
/* "xx...x.eq", where "xx...x.spc" is the input file.                     */
/*************************************************************************/

output_result(hp,freq,cport,pdef1)
FILE *hp;       /* output file pointer */
double freq;    /* frequency */
struct CXPORT cport;    /* complex n-port parameters */
struct PORTDEF *pdef1;  /* port definitions */
{
    int i,j,n1,n2;
    char lx[100],ly[100];

    /* print the headlines */
    sprintf(lx,"-------------------------------------------------------------");
    fprintf(hp,"\n\n%s\n\n",lx);
    printf("\n\n%s\n\n",lx);
    sprintf(ly,"COMPLEX GENERALIZED IMPLICIT EQUATION OF THE\n");
```

```
    sprintf(lx,"%s\tLINEAR DYNAMIC %d-PORT\n\n",ly,cport.np);
    printf("%s",lx);
    fprintf(hp,"%s",lx);

    /* print the port definitions */
    for (i=0;i<cport.np;i++)
    {
        n1=(pdef1+i)->node1;
        n2=(pdef1+i)->node2;
        sprintf(lx,"port %d is connected by %s between nodes %d and %d\n",
                i+1,(pdef1+i)->element,n1,n2);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
        sprintf(lx,"\tv[%d]=v[%d,%d]\ti[%d]=i[%d,%d]\n\n",i+1,n1,n2,i+1,n1,n2);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }
    sprintf(lx,"*********************************************************");
    printf("\n%s\n\n",lx);
    fprintf(hp,"%s\n\n",lx);

    /* print the complex generalized implicit equation */
    printf("FREQUENCY = %.6e\n",freq);
    fprintf(hp,"FREQUENCY = %.6e\n",freq);
    sprintf(lx,"***********************  P MATRIX  ***********************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<cport.np;i++)
        for (j=0;j<cport.np;j++)
        {
            sprintf(lx,"Re(P[%d,%d]) = %.3e    Im(P[%d,%d]) = %.3e\n",i+1,j+1,
                   cport.cpr[i*cport.np+j],i+1,j+1,cport.cpi[i*cport.np+j]);
            printf("%s",lx);
            fprintf(hp,"%s",lx);
        }
    sprintf(lx,"***********************  Q MATRIX  ***********************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<cport.np;i++)
        for (j=0;j<cport.np;j++)
        {
            sprintf(lx,"Re(Q[%d,%d]) = %.3e    Im(Q[%d,%d]) = %.3e\n",i+1,j+1,
                   cport.cqr[i*cport.np+j],i+1,j+1,cport.cqi[i*cport.np+j]);
            printf("%s",lx);
            fprintf(hp,"%s",lx);
        }
    sprintf(lx,"***********************  s VECTOR  ***********************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<cport.np;i++)
    {
        sprintf(lx,"Re(s[%d])    = %.3e    Im(s[%d])    = %.3e\n",
                i+1,cport.csr[i],i+1,cport.csi[i]);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }
```

```
    sprintf(lx,"-------------------------------------------------------------");
    printf("\n%s\n\n",lx);
    fprintf(hp,"\n%s\n\n",lx);
}

/*********************************************************************/
/* Open the input spice file and the table file.                     */
/*********************************************************************/

open_spc_eq(argc,argv,fp,hp)
int argc;
char *argv[];
FILE **fp,**hp;
{
    char fname[30];
    FILE *fopen();

    if (argc != 2)
        exit_message("CNPORT SPICE_FILE");
    sprintf(fname,"%s.eq",*++argv);
    if ((*hp=fopen(fname,"w")) == NULL)
    {
        printf("CAN'T OPEN THE OUTPUT EQUATION FILE %s\n",fname);
        exit();
    }
    sprintf(fname,"%s.spc",*argv);
    print_spc(fname,*hp);
    if ((*fp=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE INPUT SPICE FILE %s\n",fname);
        exit();
    }
}

/**********************************************************************/
/* Print each line of the input spice file to the output file "xx...x.eq". */
/**********************************************************************/

print_spc(fname,hp)
char fname[];           /* output file filename "xx...x.eq" */
FILE *hp;               /* output file pointer */
{
    FILE *gp,*fopen();
    char line[81];

    sprintf(line,"******************** SPICE INPUT ********************\n");
    printf("%s",line);
    fprintf(hp,"%s",line);
    if ((gp=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE INPUT SPICE FILE %s\n",fname);
        exit();
    }
    while (fgets(line,80,gp) != NULL)
    {
        fprintf(hp,"%s",line);
```

```
        printf("%s",line);
        if (find_index(".end",line) == 0 || find_index(".END",line) == 0)
            break;
    }
    fclose(gp);
    sprintf(line,"*******************************************************\n");
    fprintf(hp,"%s",line);
    printf("%s",line);
}
```

```
#include <stdio.h>
#include "cnport.h"
#include "complex.h"

int ma;         /* number of dynamic elements in the circuit */
int na;         /* # of dynamic elements + # of input sources */

/* parameters of the implicit linear resistive na-port equation */
/*                      P*v + Q*i + s = 0                       */
double *P,*Q,*s;

/* parameters of the implicit state equation            */
/*      p_11*x' + p_12*v + q_11*x + q_12*i + s_1 = 0    */
/*      p_21*x' + p_22*v + q_21*x + q_22*i + s_2 = 0    */
double *p_11,*p_12,*p_21,*p_22,*q_11,*q_12,*q_21,*q_22,*s_1,*s_2;

double *rp_12,*ip_12;   /* rp_12 + j*ip_12 = inv(jw*p_11+q_11)*p_12 */
double *rq_12,*iq_12;   /* rq_12 + j*iq_12 = inv(jw*p_11+q_11)*q_12 */
double *rc_1,*ic_1;     /* rc_1  + j*ic_1  = inv(jw*p_11+q_11)*c_1  */

/* parameters of the complex implicit dynamic (na-ma)-port equation */
/*      (cpr+j*cpi)*v + (cqr+j*cqi)*i + (csr+j*csi) = 0             */
double *cpr,*cpi,*cqr,*cqi,*csr,*csi;

COMPLEX *a;     /* a = jw*p_11+q_11 */
COMPLEX w;      /* w = 2*pi*f, where f is frequency */
COMPLEX *cp_12; /* cp_12 = rp_12 + j*ip_12 */
COMPLEX *cq_12; /* cq_12 = rq_12 + j*iq_12 */
COMPLEX *c_1;   /* c_1   = rc_1  + j*ic_1  */

/* structure for the external port element              */
/* (pdef+j)->element : element name of the element      */
/*                     connected across the j-th port   */
/* (pdef+j)->node1   : positive node of the j-th port   */
/* (pdef+j)->node2   : negative node of the j-th port   */
/* (pdef+j)->relation: element characteristic           */
/* (pdef+j)->type    : element type; 1 (resp.; 2) for   */
/*                     current (resp.; voltage) source  */
struct PORTDEF *pdef;

/*************************************************************************/
/* Find the complex generalized implicit equation                        */
/*              P(jw)*v(jw) + Q(jw)*i(jw) + s(jw) = 0                    */
/* for a linear dynamic m-port, where                                    */
/*      P(jw) = cport.cpr + j*cport.cpi                                  */
/*      Q(jw) = cport.cqr + j*cport.cqi                                  */
/*      s(jw) = cport.csr + j*cport.csi                                  */
/* and  m = cport.np                                                     */
/*************************************************************************/

cn_port(fp,hp,freq,cport,pdef1)
FILE *fp,*hp;
double freq;
struct CXPORT *cport;
struct PORTDEF **pdef1;
{
```

```
    char ch[2],*calloc();
    int i,j,k;
    FILE *gp,*fopen();

    /* get the implicit state equation */
    na=NELEM;
    gp=fopen("table.tbl","w");
    imp_eq(fp,gp,&ma,&na,&P,&Q,&s);
    if (ma == 0)
        exit_message("RESISTIVE N-PORT; SHOULD USE  nport \"xx...x\"");
    fclose(gp);

    /* read the external port element information */
    /* from the table file "table.tbl"            */
    gp=fopen("table.tbl","r");
    pdef=(PD *)calloc(na-ma,sizeof(PD));
    read_tbl(na-ma,pdef,gp);
    fclose(gp);

    /* re-arrange the implicit resistive n-port equation */
    re_order(pdef);

    /* dynamic allocation for the equation parameters */
    alloc_1();

    *pdef1=pdef;

    /* find the dynamic n-port equation for each frequency */
    ch[0]='y';
    while (ch[0] == 'y')
    {
        printf("\nenter the frequency\n");
        scanf("%lf",&freq);

        /* repeat the computations for each frequency */
        repeat_f(freq,cport);

        /* print the parameters of the dynamic n-port equation */
        output_result(hp,freq,*cport,pdef);

        printf("\ncontinue with a new frequency? y/n\n");
        scanf("%1s",ch);
    }
}

/*************************************************************************/
/* This routine is within a do loop in cn_port() for repeating the       */
/* computations of finding the complex generalized implicit equation     */
/*              P(jw)*v(jw) + Q(jw)*i(jw) + s(jw) = 0                    */
/* for a linear dynamic m-port at each given frequency w=2*pi*f.         */
/*************************************************************************/

repeat_f(freq,cport)
double freq;
struct CXPORT *cport;
{
```

```
    /* find w=2*pi*freq */
    get_w(freq);

    /* find the complex matrix a(jw)=jw*p_11+q_11 */
    get_mx();

    /* find inv(a)*p_12, inv(a)*q_12, and inv(a)*s_1 */
    if (find_inv() == -1)
        exit_message("THE COMPLEX IMPLICIT EQUATION DOESN'T EXIST");

    /* compute the parameters of the complex n-port equation */
    cpx_port();

    /* organize the n-port equation parameters into */
    /* the structure cport                          */
    cport->np=na-ma;
    cport->cpr=cpr;
    cport->cpi=cpi;
    cport->cqr=cqr;
    cport->cqi=cqi;
    cport->csr=csr;
    cport->csi=csi;
}

/*********************************************************************/
/* Allocate spaces for the parameters of implicit  state equation and    */
/* the complex generalized implicit linear dynamic n-port equation.      */
/*********************************************************************/

alloc_1()
{
    int i,j;
    char *calloc();

    /* allocate spaces for the impliciit state equation */
    p_11=(double *)calloc(ma*ma,sizeof(double));
    p_12=(double *)calloc((na-ma)*ma,sizeof(double));
    p_21=(double *)calloc((na-ma)*ma,sizeof(double));
    p_22=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    q_11=(double *)calloc(ma*ma,sizeof(double));
    q_12=(double *)calloc((na-ma)*ma,sizeof(double));
    q_21=(double *)calloc((na-ma)*ma,sizeof(double));
    q_22=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    s_1=(double *)calloc(ma,sizeof(double));
    s_2=(double *)calloc(na-ma,sizeof(double));

    /* partition the implicit state equation as
    /*          /          \ /    \   /          \ /   \   /   \        */
    /*          |p_11  p_12| | x' |   |q_11  q_12| | x |   |s_1|        */
    /*          |          |*|    | + |          |*|   | + |   | = 0    */
    /*          |p_21  p_22| | v  |   |q_21  q_22| | i |   |s_2|        */
    /*          \          / \    /   \          / \   /   \   /        */
    for (i=0;i<ma;i++)
    {
        for (j=0;j<ma;j++)
        {
```

```
            p_11[i*ma+j]=P[i*na+j];
            q_11[i*ma+j]=Q[i*na+j];
        }
        for (j=0;j<(na-ma);j++)
        {
            p_12[i*(na-ma)+j]=P[i*na+j+ma];
            q_12[i*(na-ma)+j]=Q[i*na+j+ma];
        }
        s_1[i]=s[i];
    }
    for (i=0;i<na-ma;i++)
    {
        for (j=0;j<ma;j++)
        {
            p_21[i*ma+j]=P[(i+ma)*na+j];
            q_21[i*ma+j]=Q[(i+ma)*na+j];
        }
        for (j=0;j<na-ma;j++)
        {
            p_22[i*(na-ma)+j]=P[(i+ma)*na+j+ma];
            q_22[i*(na-ma)+j]=Q[(i+ma)*na+j+ma];
        }
        s_2[i]=s[i+ma];
    }

    /* allocate spaces for the parameters of the complex n-port equation */
    cpr=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    cpi=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    cqr=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    cqi=(double *)calloc((na-ma)*(na-ma),sizeof(double));
    csr=(double *)calloc(na-ma,sizeof(double));
    csi=(double *)calloc(na-ma,sizeof(double));

    /* allocate space for the complex a */
    a=(COMPLEX *)calloc(ma*ma,sizeof(COMPLEX));

    /* allocate spaces for rp_12, ip_12, rq_12, iq_12, rc_1, and ic_1 */
    rp_12=(double *)calloc((na-ma)*ma,sizeof(double));
    ip_12=(double *)calloc((na-ma)*ma,sizeof(double));
    rq_12=(double *)calloc((na-ma)*ma,sizeof(double));
    iq_12=(double *)calloc((na-ma)*ma,sizeof(double));
    rc_1=(double *)calloc(ma,sizeof(double));
    ic_1=(double *)calloc(ma,sizeof(double));
}

/*************************************************************************/
/* Compute the matrix  a = jw*p_11 + q_11  for a given frequency w.      */
/*************************************************************************/

get_mx()
{
    char *calloc();
    int i;
    COMPLEX *cp_11,*cq_11,*cmplx(),*cadd(),*cmult();

    cp_11=(COMPLEX *)calloc(ma*ma,sizeof(COMPLEX));
```

```
    cq_11=(COMPLEX *)calloc(ma*ma,sizeof(COMPLEX));
    for (i=0;i<ma*ma;i++)
    {
        cmplx(cp_11+i,p_11[i],0.0);
        cmplx(cq_11+i,q_11[i],0.0);
        cmult(a+i,cp_11+i,&w);
        cadd(a+i,a+i,cq_11+i);
    }
    cfree(cp_11);
    cfree(cq_11);
}

/*************************************************************************/
/* Compute the angular frequency   w = 2*pi*f.                           */
/*************************************************************************/

get_w(freq)
double freq;
{
    double atan2(),pi;
    COMPLEX *cmplx();

    pi=atan2(1.0,1.0)*4;
    freq=2*pi*freq;
    cmplx(&w,0.0,freq);
}

/*************************************************************************/
/* Find the complex matrices                                             */
/*              cp_12 = rp_12 + j*ip_12 = inv(jw*p_11+q_11)*p_12         */
/*              cq_12 = rq_12 + j*iq_12 = inv(jw*p_11+q_11)*q_12         */
/*              c_1   = rc_1  + j*ic_1  = inv(jw*p_11+q_11)*s_1          */
/*************************************************************************/

find_inv()
{
    char *calloc();
    int *ipvt;
    double rcond;
    COMPLEX *zq,*cmplx();
    int i,j;
    double creal(),cimag();

    /* temperatory allocation for matrix computations */
    ipvt=(int *)calloc(ma,sizeof(int));
    cp_12=(COMPLEX *)calloc(ma*(na-ma),sizeof(COMPLEX));
    cq_12=(COMPLEX *)calloc(ma*(na-ma),sizeof(COMPLEX));
    c_1=(COMPLEX *)calloc(ma,sizeof(COMPLEX));
    zq=(COMPLEX *)calloc(ma,sizeof(COMPLEX));

    /* LU decomposition of a(jw) */
    cgeco(a,ma,ipvt,&rcond,zq);

    if (rcond < 1.0e-12)
        return(-1);
```

```
    /* find inv(a(jw))*cp_12, inv(a(jw))*cq_12, inv(a(jw))*c_1 */
    /* and store them in cp_12, cq_12, and c_1 respectively    */
    /* the ji-th entries of cp_12, cq_12 are cp_12[i*ma+j] and */
    /* cq_12[i*ma+j] respectively                              */
    for (i=0;i<(na-ma);i++)
    {
        for (j=0;j<ma;j++)
        {
            cmplx(cp_12+i*ma+j,p_12[j*(na-ma)+i],0.0);
            cmplx(cq_12+i*ma+j,q_12[j*(na-ma)+i],0.0);
        }
        cgesl(a,ma,ipvt,cp_12+i*ma,0);
        cgesl(a,ma,ipvt,cq_12+i*ma,0);
    }
    for (j=0;j<ma;j++)
        cmplx(c_1+j,s_1[j],0.0);
    cgesl(a,ma,ipvt,c_1,0);

    /* cp_12 = rp_12 + j*ip_12                                 */
    /* cq_12 = rq_12 + j*iq_12                                 */
    /* c_1   = rc_1  + j*ic_1                                  */
    /* the ij-th entries of rp_12, ip_12, rq_12, iq_12         */
    /* are rp_12[i*(na-ma)+j], ip_12[i*(na-ma)+j],             */
    /* rq_12[i*(na-ma)+j], and iq_12[i*(na-ma)+j] respectively */
    for (i=0;i<ma;i++)
        for (j=0;j<(na-ma);j++)
        {
            rp_12[i*(na-ma)+j]=creal(cp_12+j*ma+i);
            ip_12[i*(na-ma)+j]=cimag(cp_12+j*ma+i);
            rq_12[i*(na-ma)+j]=creal(cq_12+j*ma+i);
            iq_12[i*(na-ma)+j]=cimag(cq_12+j*ma+i);
        }
    for (j=0;j<ma;j++)
    {
        rc_1[j]=creal(c_1+j);
        ic_1[j]=cimag(c_1+j);
    }

    cfree(ipvt);
    cfree(zq);
    cfree(cp_12);
    cfree(cq_12);
    cfree(c_1);
}

/*************************************************************************/
/* Read the port definition in the table file "xx...x.tbl"; the j-th port  */
/* is described by the line                                                 */
/*              u[j]=v[n1,n2](Vname):(f(t))                                 */
/* (resp.;      u[j]=i[n1,n2](Iname):(f(t))                               ) */
/* when a voltage source "Vname" (resp.; current source "Iname") is         */
/* connected across two nodes n1, n2 of the j-th port, and is characterized*/
/* by a time function f(t). The port information is organized into a        */
/* structure "pdef" with the subfields :                                    */
/*              (pdef+j)->type = 1 iff a current source is connected        */
/*                                     across the j-th port                 */
```

```
/*              (pdef+j)->n1 = positive node of the j-th port              */
/*              (pdef+j)->n2 = negative node of the j-th port              */
/*              (pdef+j)->element = voltage or current source name         */
/*              (pdef+j)->relation = source characteristic                 */
/*****************************************************************************/

read_tbl(k,pdef,gp)
int k;
struct PORTDEF *pdef;
FILE *gp;
{
    int i=0,k1;
    char *calloc(),e1[300],line[300];

    while (fgets(line,80,gp) != NULL)
    {
        if (line[0] == 'u')
        {
            strdel(line,0,5);   /* delete the preceding "u[j]=" */

            /* read the positive node n1 */
            k1=find_index("[",line);
            strdel(line,0,k1+1);
            k1=find_index(",",line);
            strcpy(e1,line);
            strdel(e1,k1,strlen(e1)-k1);
            (pdef+i)->node1=atoi(e1);

            /* read the negative node n2 */
            strdel(line,0,k1+1);
            k1=find_index("]",line);
            strcpy(e1,line);
            strdel(e1,k1,strlen(e1)-k1);
            (pdef+i)->node2=atoi(e1);

            /* read the excitation source name across the port */
            k1=find_index("(I",line);
            if (k1>0)
                (pdef+i)->type=1;
            else
            {
                k1=find_index("(V",line);
                (pdef+i)->type=2;
            }
            strdel(line,0,k1+1);
            strcpy(e1,line);
            k1=find_index(")",e1);
            strdel(e1,k1,strlen(e1)-k1);
            strcpy((pdef+i)->element,e1);

            /* read the source characteristic */
            strdel(line,0,k1+2);
            k1=find_index("}",line);
            (pdef+i)->relation=calloc(strlen(line),sizeof(char));
            strcpy((pdef+i)->relation,line);
            strdel((pdef+i)->relation,k1+1,strlen(line)-k1-1);
```

```
            i++;
        }
    }
    if (i != k)
        exit_message("MISMATCHED PORT NUMBER IN THE TABLE FILE");
}

/*************************************************************************/
/* Re-arrange the implicit state equation                                */
/*                        /   \     /   \                                */
/*                        | x'|     | x |                                */
/*                      P*|   | + Q*|   | + s = 0                        */
/*                        | y |     | u |                                */
/*                        \   /     \   /                                */
/* to                                                                    */
/*                        /   \     /   \                                */
/*                        | x'|     | x |                                */
/*                      P*|   | + Q*|   | + s = 0                        */
/*                        | v |     | i |                                */
/*                        \   /     \   /                                */
/* where the j-th component of y is the voltage (resp.; current) of the  */
/* input current source (resp.; voltage source) and each component of u is */
/* complement to that of y.                                              */
/*************************************************************************/

re_order(pdef)
struct PORTDEF *pdef;
{
    int i,j;

    for (j=ma;j<na;j++)
        if ((pdef+j-ma)->type != 1)     /* connected by a voltage source */
            for (i=0;i<na;i++)
                switch_d(P+i*na+j,Q+i*na+j);
}
```

```
#include <stdio.h>
#include "cnport.h"
#include "complex.h"

/* the following variables are defined in the file "cnport1.c" */
extern int ma,na;
extern double *P,*Q,*s;
extern double *p_11,*p_12,*p_21,*p_22,*q_11,*q_12,*q_21,*q_22,*s_1,*s_2;
extern double *rp_12,*ip_12,*rq_12,*iq_12,*rc_1,*ic_1;
extern double *cpr,*cpi,*cqr,*cqi,*csr,*csi;
extern COMPLEX *a,w;
extern COMPLEX *cp_12,*cq_12,*c_1;

/*********************************************************************/
/* Perform the matrix multiplication                                 */
/*              cc = aa * bb                                         */
/* where aa, bb, cc are mm by nn, nn by kk, and mm by kk matrices    */
/* respectively.                                                     */
/*********************************************************************/

mult_mx(cc,aa,bb,mm,nn,kk)
int mm,nn,kk;
double *cc,*aa,*bb;
{
    int i,j,k;

    for (i=0;i<mm;i++)
        for (j=0;j<kk;j++)
        {
            cc[i*kk+j]=0.0;
            for (k=0;k<nn;k++)
                cc[i*kk+j]+=aa[i*nn+k]*bb[k*kk+j];
        }
}

/*********************************************************************/
/* Find the product aa = v * bb, where aa and bb are 1-dimensional arrays*/
/* with dimension mm, and v is a real scalar.                        */
/*********************************************************************/

prod_mx(aa,bb,v,mm)
int mm;
double *aa,*bb,v;
{
    int i;

    for (i=0;i<mm;i++)
        aa[i]=v*bb[i];
}

/*********************************************************************/
/* Find the addition (resp. subtraction) of two 1-dimensional arrays */
/* cc = aa + bb (resp.; cc = aa - bb) when ix=1 (resp.; ix=-1), where */
/* aa, bb, and cc have the same dimension mm.                        */
/*********************************************************************/
```

```
add_mx(cc,aa,bb,mm,ix)
int mm,ix;
double *aa,*bb,*cc;
{
    int i;

    if (ix==1)
        for (i=0;i<mm;i++)
            cc[i]=aa[i]+bb[i];
    else
        for (i=0;i<mm;i++)
            cc[i]=aa[i]-bb[i];
}

/*************************************************************************/
/* Substitute                                                            */
/*      x(jw) = -(rp_12*v+rq_12*i+rc_1)-j*(ip_12*v+iq_12*i+ic_1)         */
/* into                                                                  */
/*      p_21*x' + p_22*v + q_21*x + q_22*i + s_2 = 0                     */
/* and find the complex n-port equation                                  */
/*      [(p_22+w*p_21*ip_12-q_21*rp_12)+j*(-w*p_21*rp_12-q_21*ip_12)]*v(jw) */
/*    + [(q_22+w*p_21*iq_12-q_21*rq_12)+j*(-w*p_21*rq_12-q_21*iq_12)]*i(jw) */
/*    + [(s_2+w*p_21*ic_1-q_21*rc_1)+j*(-w*p_21*rc_1-q_21*ic_1)] = 0     */
/*************************************************************************/

cpx_port()
{
    char *calloc();
    int i,j,k;
    double *temp1,*temp2,rw,cimag();

    k=na-ma;
    rw=cimag(&w);
    temp1=(double *)calloc(k*k,sizeof(double));
    temp2=(double *)calloc(k*k,sizeof(double));

    /* find Re(cp)=p_22+w*p_21*ip_12-q_21*rp_12 */
    mult_mx(temp1,p_21,ip_12,k,ma,k);
    prod_mx(temp1,temp1,rw,k*k);
    add_mx(temp1,p_22,temp1,k*k,1);
    mult_mx(temp2,q_21,rp_12,k,ma,k);
    add_mx(cpr,temp1,temp2,k*k,-1);

    /* find Im(cp)=-w*p_21*rp_12-q_21*ip_12 */
    mult_mx(temp1,p_21,rp_12,k,ma,k);
    prod_mx(temp1,temp1,-1.0*rw,k*k);
    mult_mx(temp2,q_21,ip_12,k,ma,k);
    add_mx(cpi,temp1,temp2,k*k,-1);

    /* find Re(cq)=q_22+w*p_21*iq_12-q_21*rq_12 */
    mult_mx(temp1,p_21,iq_12,k,ma,k);
    prod_mx(temp1,temp1,rw,k*k);
    add_mx(temp1,q_22,temp1,k*k,1);
    mult_mx(temp2,q_21,rq_12,k,ma,k);
    add_mx(cqr,temp1,temp2,k*k,-1);
```

```
    /* find Im(cq)=-w*p_21*rq_12-q_21*iq_12 */
    mult_mx(temp1,p_21,rq_12,k,ma,k);
    prod_mx(temp1,temp1,-1.0*rw,k*k);
    mult_mx(temp2,q_21,iq_12,k,ma,k);
    add_mx(cqi,temp1,temp2,k*k,-1);

    /* find Re(cs)=w*p_21*ic_1-q_21*rc_1+s_2 */
    mult_mx(temp1,p_21,ic_1,k,ma,1);
    prod_mx(temp1,temp1,rw,k);
    add_mx(temp1,temp1,s_2,k,1);
    mult_mx(temp2,q_21,rc_1,k,ma,1);
    add_mx(csr,temp1,temp2,k,-1);

    /* find Im(cs)=-w*p_21*rc_1-q_21*ic_1 */
    mult_mx(temp1,p_21,rc_1,k,ma,1);
    prod_mx(temp1,temp1,-1.0*rw,k);
    mult_mx(temp2,q_21,ic_1,k,ma,1);
    add_mx(csi,temp1,temp2,k,-1);

    cfree(temp1);
    cfree(temp2);
}
```

```
typedef struct INLINE
{
   char name[9];
   int port;
   int node1,node2,node3,node4,node5,node6,node7,node8;
   char *relation;
} STI;
typedef struct B_VECTOR
{
   int a,b;
} STJ;
typedef struct CXPORT
{
    int np;
    double *cpr,*cpi,*cqr,*cqi,*csr,*csi;
};
typedef struct PORTDEF
{
    char element[9];
    int type;
    int node1,node2;
    char *relation;
} PD;
typedef char *STRING;
#define NODE 50
#define NELEM 70
#define NMODEL 30
```

```
#include <stdio.h>
#include "cnport.h"
#include "complex.h"

/*************************************************************************/
/* This is the main program of the routine "cnportx" which finds various     */
/* types of the explicit representations of a linear dynamic m-port : (1)    */
/* impedance (2) admittance (3) hybrid I and II (4) transmission (5)         */
/* scattering (6) generalized, matrix representations                        */
/*              y(jw) = mat_rep(jw)*x(jw) + s_rep(jw)                        */
/* It consists of three files : "porteq.c", "porteq1.c", "porteq2.c" and     */
/* the library routines "nport.lib", "staeq.lib", "cnport1.lib",             */
/* "complex.lib", and "clnpack.lib".                                         */
/*************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    struct CXPORT cport;
    struct PORTDEF *pdef;
    int i,j;
    FILE *fp,*hp,*fopen();
    double freq;

    /* open the input file and the output file */
    open_spc_eq(argc,argv,&fp,&hp);

    /* find the explicit n-port representation */
    cn_port1(fp,hp,freq,&cport,&pdef);

    fclose(fp);
}

/*************************************************************************/
/* Open the input file "xx...x.spc" and the output file "xx...x.eq".      */
/*************************************************************************/

open_spc_eq(argc,argv,fp,hp)
int argc;
char *argv[];
FILE **fp,**hp;
{
    char fname[30];
    FILE *fopen();

    /* bad command line */
    if (argc != 2)
        exit_message("CNPORTX SPICE_FILE");

    /* open the output file "xx...x.eq" */
    sprintf(fname,"%s.eq",*++argv);
    if ((*hp=fopen(fname,"w")) == NULL)
    {
        printf("CAN'T OPEN THE OUTPUT EQUATION FILE %s\n",fname);
        exit();
```

```
    }

    /* print the input lines of "xx...x.spc" to the output file */
    sprintf(fname,"%s.spc",*argv);
    print_spc(fname,*hp);

    /* open the input file "xx...x.spc" */
    if ((*fp=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE SPICE FILE %s\n",fname);
        exit();
    }
}

/*************************************************************************/
/* Copy each line of the input file "xx...x.spc" to the output file      */
/* "xx...x.eq".                                                          */
/*************************************************************************/

print_spc(fname,hp)
char fname[];
FILE *hp;
{
    FILE *gp,*fopen();
    char line[81];

    sprintf(line,"********************** SPICE INPUT ********************\n");
    printf("%s",line);
    fprintf(hp,"%s",line);
    gp=fopen(fname,"r");
    while (fgets(line,80,gp) != NULL)
    {
        fprintf(hp,"%s",line);
        printf("%s",line);
        if (find_index(".end",line)==0 || find_index(".END",line)==0)
            break;
    }
    fclose(gp);
    sprintf(line,"*********************************************************\n");
    printf("%s",line);
    fprintf(hp,"%s",line);
```

```
#include <stdio.h>
#include "cnport.h"
#include "/usr/include/local/complex.h"

/* parameters of the explicit n-port representation */
/*              y = mat_rep*x + s_rep                */
COMPLEX *mat_rep,*s_rep;

/* parameters of the implicit n-port representation */
/*              ax*y = bx*x + cs                     */
COMPLEX *ax,*bx;

/* controlling parameters for interactive input; neglect */
/* the interactive questions when nonzero and using the  */
/* previous inputs                                       */
int readx=0,readh=0;

int m;  /* port dimension */

/* parameters of implicit dynamic n-port equation  */
/*      (pr+j*pi)*V(jw) + (qr+j*qi)*I(jw) + cs = 0 */
double *pr,*pi,*qr,*qi;

/* parameters of the coordinate transformation  */
/*      v = (ar+j*ai)*y + (br+j*bi)*x           */
/*      i = (cr+j*ci)*y + (dr+j*di)*x           */
double *ar,*ai,*br,*bi,*cr,*ci,*dr,*di;

/* parameters of the implicit equation    */
/*      (axr+j*axi)*y = (bxr+j*bxi)*x + s */
double *axr,*axi,*bxr,*bxi;

/* chx[j][0]='v' (resp.; 'i') iff the j-th port is v-controlled */
/* chy[j][0]=          ...................                     */
char chx[10][2],chy[10][2];

/*************************************************************************/
/* Reduce the implicit dynamic m-port equation                           */
/*              P(jw)*v(jw) + q(jw)*i(jw) + s(jw) = 0                    */
/* to the explicit representation                                        */
/*              y(jw) = mat_rep(jw)*x(jw) + s_rep(jw)                    */
/*************************************************************************/

gen_port(hp,k,freq,cport,pdef)
FILE *hp;
int k;
double freq;
struct CXPORT cport;
struct PORTDEF *pdef;
{
    char *calloc();
    int i;

    m=cport.np;
    switch(k)
    {
```

```
        case 1 : impedance(cport,pdef); break;
        case 2 : admittance(cport,pdef); break;
        case 3 :
        case 4 : hybrid(k,cport,pdef); break;
        case 5 : transmission(cport,pdef); break;
        case 6 : scattering(cport,pdef); break;
        case 7 : generalized(cport,pdef); break;
        default : exit_message("UNDEFINED TYPE OF REPRESENTATION");
                  break;
    }

    /* write the explicit representation to the output file "xx...x.eq" */
    list_mat_s(hp,k,freq,cport.np,pdef);
}

/*************************************************************************/
/* Input the type of port equation from the user:                        */
/*      1 : impedance matrix representation                              */
/*      2 : admittance matrix representation                             */
/*      3 : hybrid I matrix representation                               */
/*      4 : hybrid II matrix representation                              */
/*      5 : transmission matrix representation                           */
/*      6 : scattering matrix representation                             */
/*      7 : generalized matrix representation                            */
/*************************************************************************/

get_opt(k)
int *k;
{
    printf("\nenter the equation type number\n");
    printf("\n*****************************************\n");
    printf("1 : impedance matrix representation\n");
    printf("2 : admittance matrix representation\n");
    printf("3 : hybrid I matrix representation\n");
    printf("4 : hybrid II matrix representation\n");
    printf("5 : transmission matrix representation\n");
    printf("6 : scattering matrix representation\n");
    printf("7 : generalized matrix representation\n");
    printf("*****************************************\n");
    scanf("%d",k);
}

/*************************************************************************/
/* Find the complex m-port representation                                */
/*              ax*v(jw) = bx*i(jw) + s(jw)                              */
/* and the impedance matrix presentation                                 */
/*              v(jw) = Z(jw)*i(jw) + c(jw)                              */
/* where                                                                 */
/*              Z = inv(ax)*bx                                           */
/*              c = inv(ax)*s                                            */
/*************************************************************************/

impedance(cport,pdef)
struct CXPORT cport;
struct PORTDEF *pdef;
{
```

```
    int i,j;
    double creal(),cimag();

    /* find ax and bx */
    for (j=0;j<m;j++)
        for (i=0;i<m;i++)
        {
            cmplx(ax+i*m+j,-1.0*cport.cpr[i*m+j],-1.0*cport.cpi[i*m+j]);
            cmplx(bx+i*m+j,-1.0*cport.cqr[i*m+j],-1.0*cport.cqi[i*m+j]);
        }

    /* find the inverse operations inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE IMPEDANCE REPRESENTATION DOESN'T EXIST");
}

/*********************************************************************/
/* Find the complex m-port representation                            */
/*              ax*i(jw) = bx*v(jw) + s(jw)                          */
/* and the admittance matrix representation                          */
/*              i(jw) = Y*v(jw) + c(jw)                              */
/* where                                                             */
/*              Y = inv(ax)*bx                                       */
/*              c = inv(ax)*s                                        */
/*********************************************************************/

admittance(cport,pdef)
struct CXPORT cport;
struct PORTDEF *pdef;
{
    int i,j;

    /* find ax and bx */
    for (j=0;j<m;j++)
        for (i=0;i<m;i++)
        {
            cmplx(ax+i*m+j,cport.cqr[i*m+j],cport.cqi[i*m+j]);
            cmplx(bx+i*m+j,cport.cpr[i*m+j],cport.cpi[i*m+j]);
        }

    /* find the inverse operations inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE ADMITTANCE REPRESENTATION DOESN'T EXIST");
}

/*********************************************************************/
/* Find the complex m-port representation                            */
/*              ax*y(jw) = bx*x(jw) + s(jw)                          */
/* and the hybrid matrix representation                              */
/*              y(jw) = H(jw)*x(jw) + c(jw)                          */
/* where                                                             */
/*              H = inv(ax)*bx                                       */
/*              c = inv(ax)*s                                        */
/*********************************************************************/

hybrid(k,cport,pdef)
```

```
struct CXPORT cport;
struct PORTDEF *pdef;
{
    char chv='v',chi='i';
    int i,j;

    for (j=0;j<m;j++)
    {
        /* interactive input for hybrid combination */
        if (readx < m)
        {
            printf("is port %d (%s) a v-controlled or i-controlled port? v/i\n",
                j+1,(pdef+j)->element);
            scanf("%1s",chx[j]);
            readx++;
        }

        /* find ax and bx for each combination */
        if (chx[j][0]=='i')
            for (i=0;i<m;i++)
            {
                cmplx(bx+i*m+j,-1.0*cport.cqr[i*m+j],-1.0*cport.cqi[i*m+j]);
                cmplx(ax+i*m+j,-1.0*cport.cpr[i*m+j],-1.0*cport.cpi[i*m+j]);
            }
        if (chx[j][0]=='v')
            for (i=0;i<m;i++)
            {
                cmplx(ax+i*m+j,cport.cqr[i*m+j],cport.cqi[i*m+j]);
                cmplx(bx+i*m+j,cport.cpr[i*m+j],cport.cpi[i*m+j]);
            }
    }

    /* find the inverse operations inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE HYBRID REPRESENTATION DOESN'T EXIST");

    if (k==3)   /* hybrid I representation */
        hyb_1(pdef,chv,chi);
    else        /* hybrid II representation */
        hyb_1(pdef,chi,chv);
}

/*************************************************************************/
/* Re-arrange the port indices such that the hybrid representation exhibits */
/* the block form                                                          */
/*              /     \   /           \ /     \   /     \                 */
/*              | v_a |   | H_aa  H_ab | | i_a |   | s_a |                 */
/*              |     | = |           |*|     | + |     |                 */
/*              | i_b |   | H_ba  H_bb | | v_b |   | s_b |                 */
/*              \     /   \           / \     /   \     /                 */
/* for type I hybrid representation; or                                    */
/*              /     \   /           \ /     \   /     \                 */
/*              | i_a |   | H_aa  H_ab | | v_a |   | s_a |                 */
/*              |     | = |           |*|     | + |     |                 */
/*              | v_b |   | H_ba  H_bb | | i_b |   | s_b |                 */
/*              \     /   \           / \     /   \     /                 */
```

```
/* for type II hybrid representation.                                    */
/*************************************************************************/

hyb_1(pdef,chv,chi)
struct PORTDEF *pdef;
char chv,chi;   /* chv=='v' and chi=='i' iff type I rep. */
{
    int i,j,k;
    COMPLEX temp;
    char temch[100];

    /* copy the old information */
    for (j=0;j<m;j++)
        strcpy(chy[j],chx[j]);

    for (j=0;j<m;j++)
    {
        if (chy[j][0] == chv)
        {
            k=j;

            /* looking for i-port (resp.; v-port) if hybrid I (resp.; II) */
            while (k++<m)
                if (chy[k][0] == chi)
                /* the k-th port is an i-port (resp.; v-port) */
                /* in hybrid I (resp.; II)                    */
                {
                    /* interchange the j-th and the k-th columns of mat_rep */
                    for (i=0;i<m;i++)
                    {
                        ccopy(&temp,mat_rep+i*m+j);
                        ccopy(mat_rep+i*m+j,mat_rep+i*m+k);
                        ccopy(mat_rep+i*m+k,&temp);
                    }

                    /* interchange the j-th and the k-th rows of mat_rep */
                    for (i=0;i<m;i++)
                    {
                        ccopy(&temp,mat_rep+j*m+i);
                        ccopy(mat_rep+j*m+i,mat_rep+k*m+i);
                        ccopy(mat_rep+k*m+i,&temp);
                    }

                    /* interchange the j-th and the k-th components of s_rep */
                    ccopy(&temp,s_rep+j);
                    ccopy(s_rep+j,s_rep+k);
                    ccopy(s_rep+k,&temp);

                    chy[k][0]=chv;
                    if (readh ==0)
                    {
                        /* interchange the port definitions */
                        switch_i(&((pdef+j)->node1),&((pdef+k)->node1));
                        switch_i(&((pdef+j)->node2),&((pdef+k)->node2));
                        strcpy(temch,(pdef+j)->element);
                        strcpy((pdef+j)->element,(pdef+k)->element);
```

```
                        strcpy((pdef+k)->element,temch);
                        readh=1;
                    }
                }
            }
        }
}

/*********************************************************************/
/* Find the transmission matrix representation                       */
/*              y(jw) = T(jw)*x(jw) + c(jw)                          */
/*********************************************************************/

transmission(cport,pdef)
struct CXPORT cport;
struct PORTDEF *pdef;
{
    int i;

    /* define the coordinate matrices a, b, c, and d */
    if (readx == 0)
    {
        t_abcd(pdef);
        readx=1;
    }

    /* find ax and bx for the implicit representation */
    /*          ax*X(jw) = bx*Y(jw) + s(jw)           */
    get_ax_bx(cport);
    for (i=0;i<m*m;i++)
    {
        cmplx(ax+i,axr[i],axi[i]);
        cmplx(bx+i,bxr[i],bxi[i]);
    }

    /* find the inverse operations inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE TRANSMISSION REPRESENTATION DOESN'T EXIST");
}

/*********************************************************************/
/* Define the coordinate transformation matrices a, b, c, and d for the  */
/* transmission matrix representation.                               */
/*********************************************************************/

t_abcd(pdef)
struct PORTDEF *pdef;
{
    int i,j,m2;
    char ch[2];

    if (m%2 != 0)
        exit_message("ODD PORT NUMBER");
    m2=m/2;
    for (i=0;i<m;i++)
    {
```

```
        printf("is port %d (%s) an input port? y/n\n",i+1,(pdef+i)->element);
        scanf("%1s",ch);
        if (ch[0] == 'n')
        {
            ar[i*m+i]=1.0;
            br[(i+m2)*m+i]=1.0;
            cr[i*m+m2+i] = -1.0;
            dr[(i+m2)*m+i+m2] = 1.0;
        }
        else
        {
            ar[(i+m2)*m+i]=1.0;
            br[i*m+i]=1.0;
            cr[(i+m2)*m+i+m2] = -1.0;
            dr[i*m+i+m2] = 1.0;
        }
    }
}

/*************************************************************************/
/* Find the scattering matrix representation                             */
/*              y(jw) = S(jw)*x(jw) + c(jw)                              */
/*************************************************************************/

scattering(cport,pdef)
struct CXPORT cport;
struct PORTDEF *pdef;
{
    int i;

    /* define the coordinate matrices a, b, c, and d */
    if (readx == 0)
    {
        s_abcd(pdef);
        readx=1;
    }

    /* find ax and bx for the implicit scattering representation */
    /*          ax*Y(jw) = bx*X(jw) + s(jw)                          */
    get_ax_bx(cport);
    for (i=0;i<m*m;i++)
    {
        cmplx(ax+i,axr[i],axi[i]);
        cmplx(bx+i,bxr[i],bxi[i]);
    }

    /* inverse operations for inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE SCATTERING REPRESENTATION DOESN'T EXIST");
}

/*************************************************************************/
/* Define the coordinate transformation matrices a, b, c, and d for the  */
/* scattering matrix representation.                                     */
/*************************************************************************/
```

```
s_abcd(pdef)
struct PORTDEF *pdef;
{
    int i;
    double rx,sqrt();

    for (i=0;i<m;i++)
    {
        printf("enter R_%d for the %d-th port (%s)\n",
                i+1,i+1,(pdef+i)->element);
        scanf("%lf",&rx);
        rx=sqrt(rx);
        ar[i*m+i]=rx;
        br[i*m+i]=rx;
        cr[i*m+i]= -1.0/rx;
        dr[i*m+i]=1.0/rx;
    }
}

/*********************************************************************/
/* Find the explicit generalized representation                      */
/*              y(jw) = G(jw)*x(jw) + c(jw)                          */
/* where                                                             */
/*              v(jw) = a*y(jw) + b*x(jw)                            */
/*              i(jw) = c*y(jw) + d*x(jw)                            */
/*********************************************************************/

generalized(cport,pdef)
struct CXPORT cport;
struct PORTDEF *pdef;
{
    int i,j;

    /* read the coordinate matrices a, b, c, and d */
    if (readx ==0)
        read_abcd();

    /* find ax and bx for the implicit representation */
    /*          ax*Y(jw) = bx*X(jw) + s(jw)          */
    get_ax_bx(cport);
    for (i=0;i<m*m;i++)
    {
        cmplx(ax+i,axr[i],axi[i]);
        cmplx(bx+i,bxr[i],bxi[i]);
    }

    /* inverse operations for inv(ax)*bx and inv(ax)*s */
    if (get_inv(m,cport) == -1)
        exit_message("THE GENERALIZED REPRESENTATION DOESN'T EXIST");
}

/*********************************************************************/
/* Substitute the coordinate relation                                */
/*              v(jw) = a*y(jw) + b*x(jw)                            */
/*              i(jw) = c*y(jw) + d*x(jw)                            */
/* into the implicit linear dynamic n-port equation                  */
```

```
/*              (pr+j*pi)*v(jw) + (qr+j*qi)*i(jw) + (sr+j*si(jw)) = 0     */
/* and get the new implicit equation                                      */
/*              ax*y(jw) = bx*x(jw) + s(jw)                               */
/*************************************************************************/

get_ax_bx(cport)
struct CXPORT cport;
{
    int i,j;
    char *calloc();
    double *temp1,*temp2;

    temp1=(double *)calloc(m*m,sizeof(double));
    temp2=(double *)calloc(m*m,sizeof(double));
    pr=cport.cpr;
    pi=cport.cpi;
    qr=cport.cqr;
    qi=cport.cqi;

    /* find Re(bx)=pr*br+qi*di-pi*bi-qr*dr */
    mult_mx(temp1,pr,br,m,m,m);
    mult_mx(temp2,qi,di,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,1);
    mult_mx(temp2,pi,bi,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,-1);
    mult_mx(temp2,qr,dr,m,m,m);
    add_mx(bxr,temp1,temp2,m*m,-1);

    /* find Im(bx)=pr*bi+pi*br-qr*di-qi*dr */
    mult_mx(temp1,pr,bi,m,m,m);
    mult_mx(temp2,pi,br,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,1);
    mult_mx(temp2,qr,di,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,-1);
    mult_mx(temp2,qi,dr,m,m,m);
    add_mx(bxi,temp1,temp2,m*m,-1);

    /* find Re(ax)=-pr*ar+pi*ai+qr*cr-qi*ci */
    mult_mx(temp1,pi,ai,m,m,m);
    mult_mx(temp2,pr,ar,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,-1);
    mult_mx(temp2,qr,cr,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,1);
    mult_mx(temp2,qi,ci,m,m,m);
    add_mx(axr,temp1,temp2,m*m,-1);

    /* find Im(ax)=-pi*ar-pr*ai+qr*ci+qi*cr */
    mult_mx(temp1,qi,cr,m,m,m);
    mult_mx(temp2,qr,ci,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,1);
    mult_mx(temp2,pr,ai,m,m,m);
    add_mx(temp1,temp1,temp2,m*m,-1);
    mult_mx(temp2,pi,ar,m,m,m);
    add_mx(axi,temp1,temp2,m*m,-1);

    cfree(temp1);
```

```
    cfree(temp2);
}

/*************************************************************************/
/* Read the coordinate matrices a, b, c, d for the new variables y and x   */
/* such that                                                               */
/*              v = (ar + j*ai)*y + (br + j*bi)*x                          */
/*              i = (cr + j*ci)*y + (dr + j*di)*x                          */
/*************************************************************************/

read_abcd()
{
    int i,j;

    for (i=0;i<m;i++)
    {
        printf("enter row %d of the %d by %d matrix Re(a)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",ar+i*m+j);
        printf("enter row %d of the %d by %d matrix Im(a)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",ai+i*m+j);
        printf("enter row %d of the %d by %d matrix Re(b)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",br+i*m+j);
        printf("enter row %d of the %d by %d matrix Im(b)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",bi+i*m+j);
        printf("enter row %d of the %d by %d matrix Re(c)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",cr+i*m+j);
        printf("enter row %d of the %d by %d matrix Im(c)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",ci+i*m+j);
        printf("enter row %d of the %d by %d matrix Re(d)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",dr+i*m+j);
        printf("enter row %d of the %d by %d matrix Im(d)\n",i+1,m,m);
        for (j=0;j<m;j++)
            scanf("%lf",di+i*m+j);
    }
    readx=1;
}
```

```
#include <stdio.h>
#include "cnport.h"
#include "/usr/include/local/complex.h"

/* see the declarations in the file "porteq1.c" */
extern COMPLEX *ax,*bx,*mat_rep,*s_rep;
extern double *pr,*pi,*qr,*qi;
extern double *ar,*ai,*br,*bi,*cr,*ci,*dr,*di;
extern double *axr,*axi,*bxr,*bxi;

int *ipvt;      /* pivoting vector in LU decomposition */
COMPLEX *zq;    /* working vector for matrix operation */

/*************************************************************************/
/* Dynamic allocation for the parameters used in various representation.  */
/*************************************************************************/

alloc_3(m)
int m;
{
    char *calloc();

    ax=(COMPLEX *)calloc(m*m,sizeof(COMPLEX));
    bx=(COMPLEX *)calloc(m*m,sizeof(COMPLEX));
    ipvt=(int *)calloc(m,sizeof(int));
    zq=(COMPLEX *)calloc(m,sizeof(COMPLEX));
    mat_rep=(COMPLEX *)calloc(m*m,sizeof(COMPLEX));
    s_rep=(COMPLEX *)calloc(m,sizeof(COMPLEX));
    pr=(double *)calloc(m*m,sizeof(double));
    pi=(double *)calloc(m*m,sizeof(double));
    qr=(double *)calloc(m*m,sizeof(double));
    qi=(double *)calloc(m*m,sizeof(double));
    ar=(double *)calloc(m*m,sizeof(double));
    ai=(double *)calloc(m*m,sizeof(double));
    br=(double *)calloc(m*m,sizeof(double));
    bi=(double *)calloc(m*m,sizeof(double));
    cr=(double *)calloc(m*m,sizeof(double));
    ci=(double *)calloc(m*m,sizeof(double));
    dr=(double *)calloc(m*m,sizeof(double));
    di=(double *)calloc(m*m,sizeof(double));
    axr=(double *)calloc(m*m,sizeof(double));
    axi=(double *)calloc(m*m,sizeof(double));
    bxr=(double *)calloc(m*m,sizeof(double));
    bxi=(double *)calloc(m*m,sizeof(double));
}

/*************************************************************************/
/* Find the inverse operations                                           */
/*              mat_rep = inv(ax)*bx                                     */
/*              s_rep = inv(ax)*cs                                       */
/*************************************************************************/

get_inv(m,cport)
struct CXPORT cport;
{
    int i,j;
```

```
    char *calloc();
    double rcond;
    double creal(),cimag();
    COMPLEX *cz;

    cz=(COMPLEX *)calloc(m,sizeof(COMPLEX));

    /* LU decomposition on ax */
    cgeco(ax,m,ipvt,&rcond,zq);

    if (rcond < 1.0e-12)
        return(-1);
    for (i=0;i<m;i++)
    {
        for (j=0;j<m;j++)
            ccopy(cz+j,bx+j*m+i);
        cgesl(ax,m,ipvt,cz,0);
        for (j=0;j<m;j++)
            ccopy(mat_rep+j*m+i,cz+j);
    }
    for (j=0;j<m;j++)
        cmplx(s_rep+j,cport.csr[j],cport.csi[j]);
    cgesl(ax,m,ipvt,s_rep,0);
    cfree(cz);
}

/*************************************************************************/
/* Print various types of representation                                 */
/*              y(jw) = mat_rep(jw)*x(jw) + s_rep(jw)                    */
/* on the standard output and the output file "xx...x.eq".               */
/*************************************************************************/

list_mat_s(hp,k,freq,dim,pdef)
FILE *hp;
int k,dim;
double freq;
struct PORTDEF *pdef;
{
    char ch;
    int i,j,n1,n2;
    char lx[100],ly[50],lz[50];

    sprintf(lx,"-------------------------------------------------------------");
    printf("\n\n%s\n\n",lx);
    fprintf(hp,"\n\n%s\n\n",lx);
    switch(k)
    {
        case 1 : {
                    sprintf(ly,"IMPEDANCE MATRIX");
                    ch='Z';     break;
                 }
        case 2 : {
                    sprintf(ly,"ADMITTANCE MATRIX");
                    ch='Y';     break;
                 }
        case 3 : {
```

```
                    sprintf(ly,"HYBRID I MATRIX");
                    ch='H';     break;
                }
        case 4 : {
                    sprintf(ly,"HYBRID II MATRIX");
                    ch='H';     break;
                }
        case 5 : {
                    sprintf(ly,"TRANSMISSION MATRIX");
                    ch='T';     break;
                }
        case 6 : {
                    sprintf(ly,"SCATTERING MATRIX");
                    ch='S';     break;
                }
        case 7 : {
                    sprintf(ly,"GENERALIZED MATRIX");
                    ch='G';     break;
                }
    }

    /* print the headings */
    sprintf(lz,"EXPLICIT LINEAR DYNAMIC %d-PORT\n",dim);
    sprintf(lx,"%s%s REPRESENTATION\n\n",lz,ly);
    printf("%s",lx);
    fprintf(hp,"%s",lx);

    /* print the port definitions */
    for (i=0;i<dim;i++)
    {
        n1=(pdef+i)->node1;
        n2=(pdef+i)->node2;
        sprintf(lx,"port %d is connected by %s between nodes %d and %d\n",
                i+1,(pdef+i)->element,n1,n2);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
        sprintf(lx,"\tv[%d]=v[%d,%d]\ti[%d]=i[%d,%d]\n\n",i+1,n1,n2,i+1,n2,n1);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }
    sprintf(lx,"*************************************************************\n");
    printf("%s\n",lx);
    fprintf(hp,"%s\n",lx);

    /* print the explicit representation at the given frequency */
    sprintf(lx,"FREQUENCY = %.6e\n",freq);
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,"************************ %c MATRIX *************************\n",
            ch);
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<dim;i++)
        for (j=0;j<dim;j++)
        {
            sprintf(lx,"Re(%c[%d,%d]) = %.3e\tIm(%c[%d,%d]) = %.3e\n",
```

```
                    ch,i+1,j+1,creal(mat_rep+i*dim+j),
                    ch,i+1,j+1,cimag(mat_rep+i*dim+j));
            printf("%s",lx);
            fprintf(hp,"%s",lx);
        }
    printf("\n");
    fprintf(hp,"\n");
    sprintf(lx,"********************** c VECTOR **********************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<dim;i++)
    {
        sprintf(lx,"Re(c[%d]) = %.3e\tIm(c[%d]) = %.3e\n",i+1,creal(s_rep+i),
                i+1,cimag(s_rep+i));
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }
    sprintf(lx,"------------------------------------------------------------");
    printf("\n%s\n\n",lx);
    fprintf(hp,"\n%s\n\n",lx);
}
```

```
typedef struct INLINE
{
   char name[9];
   int port;
   int node1,node2,node3,node4,node5,node6,node7,node8;
   char *relation;
} STI;
typedef struct B_VECTOR
{
   int a,b;
} STJ;
typedef char *STRING;
#define NODE 50
#define NELEM 70
#define NMODEL 30
```

```
#include <stdio.h>
#include "stateq.h"

extern struct INLINE *branch;
extern struct B_VECTOR *branch_vector;

/*************************************************************************/
/* This is the main program of the routine "staeq" which finds the linear   */
/* state equation                                                           */
/*              x' = A*x + B*u + s_1 + T_1*u'                               */
/*              y  = C*x + D*u + s_2 + T_2*u'                               */
/* for a linear dynamic circuit, where the constant vectors s_1, s_2 are    */
/* contributed by time-invariant independent sources, and T_1, T_2 are due  */
/* to the loop of capacitors and input voltage sources, or cutset of        */
/* inductors and input current sources. It consists of three files :        */
/* "staeq.c", "staeq1.c", "staeq2.c" and the library routines "nport.lib"   */
/* "linpack.lib".                                                           */
/*************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    int i,j,k;
    double *A,*B,*t1,*t2,*C,*D,*s1,*s2,*P,*Q,*s;
    FILE *fp,*gp,*hp,*fopen();
    int n,m,l;

    /* open the input file */
    open_spc_tbl(argc,argv,&fp,&gp);

    /* find the state equation */
    n=NELEM;
    state_eq(fp,gp,&n,&m,&l,&A,&B,&C,&D,&s1,&s2,&t1,&t2);
    fclose(fp);
    fclose(gp);

    /* print the state equation */
    print_eq(m,n);      /* print the headings and the variable definitions */
    printf("************************* A MATRIX **********************\n");
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            printf("A[%d,%d] = %.3e\n",i+1,j+1,A[i*n+j]);
    printf("************************* B MATRIX **********************\n");
    for (i=0;i<n;i++)
        for (j=0;j<m;j++)
            printf("B[%d,%d] = %.3e\n",i+1,j+1,B[i*m+j]);
    printf("************************* C MATRIX **********************\n");
    for (i=0;i<m;i++)
        for (j=0;j<n;j++)
            printf("C[%d,%d] = %.3e\n",i+1,j+1,C[i*n+j]);
    printf("************************* D MATRIX **********************\n");
    for (i=0;i<m;i++)
        for (j=0;j<m;j++)
            printf("D[%d,%d] = %.3e\n",i+1,j+1,D[i*m+j]);
    printf("************************* s_1 VECTOR ********************\n");
```

```
    for (i=0;i<n;i++)
        printf("s_1[%d] = %.3e\n",i+1,s1[i]);
    printf("************************ s_2 VECTOR *********************\n");
    for (i=0;i<m;i++)
        printf("s_2[%d] = %.3e\n",i+1,s2[i]);
    printf("************************ T_1 MATRIX *********************\n");
    for (i=0;i<n;i++)
        for (j=0;j<m;j++)
            printf("T_1[%d,%d] = %.3e\n",i+1,j+1,t1[i*m+j]);
    printf("************************ T_2 MATRIX *********************\n");
    for (i=0;i<m;i++)
        for (j=0;j<m;j++)
            printf("T_2[%d,%d] = %.3e\n",i+1,j+1,t2[i*m+j]);

}

/*****************************************************************************/
/* Open the input file "xx...x.spc and the table file "xx...x.tbl".          */
/*****************************************************************************/

open_spc_tbl(argc,argv,fp,gp)
int argc;
char *argv[];
FILE **fp,**gp;
{
    FILE *fopen();
    char fname[30];

    /* bad command line */
    if (argc != 2)
        exit_message("STAEQ SPICE_FILE");

    /* open the spice file */
    sprintf(fname,"%s.spc",*++argv);
    print_spc(fname);   /* print each line of the input file */
    if ((*fp=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE SPICE FILE %s\n",fname);
        exit();
    }

    /* open the table file */
    sprintf(fname,"%s.tbl",*argv);
    if ((*gp=fopen(fname,"w")) == NULL)
    {
        printf("CAN'T OPEN THE TABLE FILE %s\n",fname);
        exit();
    }
}

/*****************************************************************************/
/* Print the headings of the linear state equation.                          */
/*****************************************************************************/

print_eq(m,n)
int m,n;
```

```
{
    int i,j;

    printf("******************* LINEAR STATE EQUATION *******************\n");
    printf("\n\t\tx' = A*x + B*u + s_1 + T_1*u' \n");
    printf("\t\ty  = C*x + D*u + s_2 + T_2*u' \n\n");
    printf("***** state variables *****\n");
    for (i=0;i<n;i++)
    {
        j=(branch_vector+i)->a;
        if ((branch+j)->name[0] == 'C')
            printf("x[%d]=v[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        else
            printf("x[%d]=i[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
    }
    printf("\n***** input variables *****\n");
    for (i=n;i<n+m;i++)
    {
        j=(branch_vector+i)->a;
        if ((branch+j)->name[0] == 'V')
            printf("u[%d]=v[%d,%d](%s)\n",i-n+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        else
            printf("u[%d]=i[%d,%d](%s)\n",i-n+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
    }
    printf("\n***** output variables *****\n");
    for (i=n;i<n+m;i++)
    {
        j=(branch_vector+i)->a;
        if ((branch+j)->name[0] == 'V')
            printf("y[%d]=i[%d,%d](%s)\n",i-n+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        else
            printf("y[%d]=v[%d,%d](%s)\n",i-n+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
    }
    printf("\n");
    printf("-----------------------------------------------------------\n");
}

/*************************************************************************/
/* Copy each line of the input file "xx...x.spc".                        */
/*************************************************************************/

print_spc(fname)
char fname[];
{
    FILE *gp,*fopen();
    char line[81];

    printf("********************** SPICE INPUT *********************\n");
    gp=fopen(fname,"r");
    while (fgets(line,80,gp) != NULL)
```

```
    {
        printf("%s",line);
        if (find_index(".end",line)==0 || find_index(".END",line)==0)
            break;
    }
    fclose(gp);
    printf("*************************************************************\n\n");
}
```

```
#include <stdio.h>
#include "stateq.h"

struct INLINE *branch;
struct B_VECTOR *branch_vector;
char *model[NMODEL];
double *d;

/*************************************************************************/
/* Find the state equation                                               */
/*              x' = A*x + B*u + s_1 + T_1*u'                            */
/*              y  = C*x + D*u + s_2 + T_2*u'                            */
/* for a linear dynamic circuit.                                         */
/*************************************************************************/

state_eq(fp,hp,n1,m1,r1,A,B,C,D,s1,s2,t1,t2)
FILE *fp;       /* input file pointer */
FILE *hp;       /* pointer of table file */
int *n1,*m1,*r1; /* A is n1xn1; B is n1xm1; C is r1xn1; D is r1xr1 */
double **A,**B,**C,**D,**s1,**s2,**t1,**t2;
{
    char *calloc();
    FILE *gp;
    double *P,*Q,*s;
    int m,n;

    gp=NULL;
    n = *n1;
    imp_eq1(fp,gp,&m,&n,&P,&Q,&s);
    state(&m,&n,A,B,C,D,P,Q,s,s1,s2,t1,t2);
    cfree(d);
    get_table(hp,m,n);
    cfree(P);
    cfree(Q);
    cfree(s);
    *n1=m;
    *m1=n-m;
    *r1=n-m;
}

/*************************************************************************/
/* Find the implicit equation                                            */
/*              P*w  + Q*z + s = 0                                       */
/* where z consists of                                                   */
/*              1. capacitor voltage                                     */
/*              2. inductor current                                      */
/*              3. voltage of input voltage source                       */
/*              4. current of input current source                       */
/* and w consists of                                                     */
/*              1. capacitor current                                     */
/*              2. inductor voltage                                      */
/*              3. current of input voltage source                       */
/*              4. voltage of input current source                       */
/*************************************************************************/

imp_eq1(fp,hp,m,n,P,Q,s)
```

```
FILE *fp,*hp;
int *m,*n;
double **P,**Q,**s;
{
    char *calloc();
    int mn,i,j;

    *m=0;
    mn=NMODEL;
    branch=(STI *)calloc(*n,sizeof(STI));
    branch_vector=(STJ *)calloc(*n,sizeof(STJ));
    n_port(fp,model,branch,branch_vector,P,Q,s,n,&mn);
    d=(double *)calloc(*n,sizeof(double));
    decode(*n,m,*P,*Q,*s);

    if (hp!=NULL)
        get_table(hp,*m,*n);
}

/*************************************************************************/
/* Find the implicit state equation                                      */
/*              /   \       /   \                                        */
/*            P*| x'|  + Q*| x | + s = 0                                 */
/*              | y |       | u |                                        */
/*              \   /       \   /                                        */
/*                                                                       */
/* where x consists of                                                   */
/*              1. capacitor voltage                                     */
/*              2. inductor current                                      */
/* and u (resp.; y) consists of                                          */
/*              1. voltage (resp.; current) of input voltage source      */
/*              2. current (resp.; voltage) of input current source      */
/*************************************************************************/

imp_eq(fp,hp,m,n,P,Q,s)
FILE *fp,*hp;
int *m,*n;
double **P,**Q,**s;
{
    int i,j;

    imp_eq1(fp,hp,m,n,P,Q,s);

    /* multiply the capacitance or inductance */
    for (i=0;i<*n;i++)
        for (j=0;j<*m;j++)
            (*P)[i*(*n)+j] *= d[j];
}

/*************************************************************************/
/* process each element across the linear resistive n-port.              */
/*************************************************************************/

decode(n,m,P,Q,s)
int n,*m;
double *P,*Q,*s;
```

```
{
    char ch;
    int i,j,k;
    double stof();

    for (i=0;i<n;i++)
    {
        j=(branch_vector+i)->a;         /* element index */
        ch=(branch+j)->name[0]; /* element type */
        if (ch=='V' || ch=='I') /* input source */
            arrange(&ch,i,&j,n,P,Q,s);
        switch(ch)
        {
            case 'L' : {
                            /* inductance */
                            d[(*m)++]=stof((branch+j)->relation);
                            break;
                        }
            case 'C' : {
                            /* capacitance */
                            d[(*m)++]=stof((branch+j)->relation);
                            for (k=0;k<n;k++)
                                switch_d(&P[k*n+i],&Q[k*n+i]);
                            break;
                        }
            case 'V' : {
                            for (k=0;k<n;k++)
                            switch_d(&P[k*n+i],&Q[k*n+i]);
                            break;
                        }
            case 'I' : break;
            default  : {
                            printf("ILLEGAL CIRCUIT ELEMENT TYPE %c\n",ch);
                            exit();
                            break;
                        }
        }
    }
}

/*************************************************************************/
/* List each state variable and input-output source in the table file.    */
/*************************************************************************/

get_table(gp,m,n)
FILE *gp;
int m,n;
{
    char line[50];
    int i,j;

    for (i=0;i<n;i++)
    {
        j=(branch_vector+i)->a;
        if (i<m)
        {
```

```
            if ((branch+j)->name[0]=='C')
            sprintf(line,"x[%d]=v[%d,%d](%s)\n",i,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
            else sprintf(line,"x[%d]=i[%d,%d](%s)\n",i,(branch+j)->node1,
                         (branch+j)->node2,(branch+j)->name);
        }
        else
        {
            if ((branch+j)->name[0]=='V')
               sprintf(line,"u[%d]=v[%d,%d](%s):%s\n",i-m,(branch+j)->node1,
                      (branch+j)->node2,(branch+j)->name,(branch+j)->relation);
            else
                sprintf(line,"u[%d]=i[%d,%d](%s):%s\n",i-m,(branch+j)->node1,
                      (branch+j)->node2,(branch+j)->name,(branch+j)->relation);
        }
        fputs(line,gp);
    }
}

/*************************************************************************/
/* Re-arrange the row vectors such that the input-output ports are in the  */
/* bottom position of the n-port equation.                                 */
/*************************************************************************/

arrange(c,i,m,n,P,Q,s)
int i,*m,n;
char *c;
double *P,*Q,*s;
{
    int l,k,j;
    char ch;

    for (l=i+1;l<n;l++)
    {
        j=(branch_vector+l)->a;
        ch=(branch+j)->name[0];
        if (ch=='C' || ch=='L')
        {
            *c=ch;
            *m=j;
            switch_i(&((branch_vector+l)->a),&((branch_vector+i)->a));
            switch_i(&((branch_vector+l)->b),&((branch_vector+i)->b));
            for (k=0;k<n;k++)
            {
                switch_d(&P[k*n+i],&P[k*n+l]);
                switch_d(&Q[k*n+i],&Q[k*n+l]);
            }
            break;
        }
    }
}

/*************************************************************************/
/* Get the state equation.                                                 */
/*************************************************************************/
```

```
state(m2,n2,A,B,C,D,P,Q,s,s1,s2,t1,t2)
int *m2,*n2;
double **A,**B,**C,**D,*P,*Q,s[],**s1,**s2,**t1,**t2;
{
    char *calloc();
    int i,j,m,n,*ipvt,job=0;
    double rcond,*zq,*f,*aa,fabs(),*b1;

    m = *m2;
    n = *n2;
    v_alloc(n,&ipvt,&zq,&f,&aa,P);
    sgeco(aa,n,ipvt,&rcond,zq);
    b1=(double *)calloc(n*(n-m),sizeof(double));

    if (fabs(rcond)<1.0E-10)
    {
        printf("WARNING MESSAGE : CAPACITOR LOOP OR INDUCTOR CUTSET\n");
        /* capacitor loop or inductor cutset */
        loop_cs(P,Q,b1,s,d,&m,&n); /* get the independent variables */
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                aa[i*n+j]=P[i*n+j];
        sgeco(aa,n,ipvt,&rcond,zq);
        for (j=0;j<m;j++)
            d[j]=1.0;
    }

    *A=(double *)calloc(m*m,sizeof(double));
    *B=(double *)calloc(m*(n-m),sizeof(double));
    *C=(double *)calloc(m*(n-m),sizeof(double));
    *D=(double *)calloc((n-m)*(n-m),sizeof(double));
    *s1=(double *)calloc(m,sizeof(double));
    *s2=(double *)calloc(n-m,sizeof(double));
    *t1=(double *)calloc(m*(n-m),sizeof(double));
    *t2=(double *)calloc((n-m)*(n-m),sizeof(double));

    /* get A, B, C, D, s1, s2, t1, and t2 */
    for (i=0;i<n;i++)
    {
        for (j=0;j<n;j++)
            f[j] = -Q[j*n+i];
        sgesl(aa,n,ipvt,f,job);
        if (i<m)
        {
            for (j=0;j<m;j++)
                (*A)[j*m+i]=f[j]/d[j];
            for (j=0;j<n-m;j++)
                (*C)[j*m+i]=f[m+j];
        }
        else
        {
            for (j=0;j<m;j++)
                (*B)[j*(n-m)+i-m]=f[j]/d[j];
            for (j=0;j<n-m;j++)
                (*D)[j*(n-m)+i-m]=f[m+j];
        }
```

```
    }
    sgesl(aa,n,ipvt,s,job);
    for (i=0;i<n;i++)
    {
        if (i<m)
            (*s1)[i] = -s[i]/d[i];
        else
            (*s2)[i-m] = -s[i];
    }
    for (j=0;j<(n-m);j++)
    {
        for (i=0;i<n;i++)
            f[i] = -b1[i*(n-m)+j];
        sgesl(aa,n,ipvt,f,job);
        for (i=0;i<n;i++)
        {
            if (i<m)
                (*t1)[i*(n-m)+j]=f[i];
            else
                (*t2)[(i-m)*(n-m)+j]=f[i];
        }
    }
    *m2=m;
    *n2=n;
    v_free(ipvt,zq,f,aa);
}

/*************************************************************************/
v_alloc(n,ipvt,zq,f,aa,P)
int n,**ipvt;
double **zq,**f,**aa,*P;
{
    char *calloc();
    int i;

    *ipvt=(int *)calloc(n,sizeof(int));
    *zq=(double *)calloc(n,sizeof(double));
    *f=(double *)calloc(n,sizeof(double));
    *aa=(double *)calloc(n*n,sizeof(double));
    for (i=0;i<n*n;i++)
        (*aa)[i]=P[i];
}

/*************************************************************************/
v_free(ipvt,zq,f,aa)
int *ipvt;
double *zq,*f,*aa;
{
    cfree(ipvt);
    cfree(zq);
    cfree(f);
    cfree(aa);
}
```

```
#include <stdio.h>
#include "stateq.h"

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

extern struct INLINE *branch;
extern struct B_VECTOR *branch_vector;
extern char *model[];

/*************************************************************************/
/* Reduce the P matrix to row-elchelon form.                             */
/*************************************************************************/

elchelon(n,ii,P,Q,s)
int *n,*ii;
double *P,*Q,*s;
{
    int i=0,l= -1,j,k;
    double c;

    while (++l<*n)
    {
        if (P[i*(*n)+l]==0.0)
        {   /* select a nonzero pivot */
            for (j=i+1;P[j*(*n)+l]==0.0 && j<*n;j++);
            if (j<*n)
            {
                switch_d(&s[i],&s[j]);
                for (k=0;k<*n;k++)
                {
                    switch_d(&P[i*(*n)+k],&P[j*(*n)+k]);
                    switch_d(&Q[i*(*n)+k],&Q[j*(*n)+k]);
                }
            }
        }
        if (P[i*(*n)+l]!=0.0)
        {
            for (j=i+1;j<*n;j++)
                if (P[j*(*n)+l]!=0.0)
                {
                    c=P[j*(*n)+l]/P[i*(*n)+l];
                    for (k=l;k<*n;k++)
                        P[j*(*n)+k]-=c*P[i*(*n)+k];
                    for (k=0;k<*n;k++)
                        Q[j*(*n)+k]-=c*Q[i*(*n)+k];
                    s[j]-=c*s[i];
                }
            i++;
        }
    }
    *ii=i;
}
```

```
/*************************************************************************/
/* Choose independent variables from capacitor voltages which form a loop  */
/* or inductor currents which form a cutset.                               */
/*************************************************************************/

loop_cs(P,Q,bl,s,d,m,n)
int *m,*n;
double *P,*Q,*s,*bl,*d;
{
    char *calloc();
    int i,*lx;

    /* reduce the P matrix to the row echelon form */
    elchelon(n,&i,P,Q,s);
    lx=(int *)calloc((*n)-i+1,sizeof(int));
    lx[(*n)-i] = -1;

    /* get the linear relationship for capacitor voltages (resp.; inductor */
    /* currrents) within a capacitor loop (resp.; inductor cutset)         */
    red_Q(m,n,i,lx,Q,s);

    /* replace the rudundent capacitor voltages (resp.; inductor currents) */
    /* by other variables                                                  */
    comp_PQ(m,n,i,lx,P,Q,s,d,bl);

    /* kick out the redundent variables and get a more compact equation */
    new_PQ(m,n,i,lx,P,Q);
}

/*************************************************************************/
/* get the linear equation for capacitor voltages in a loop or inductor    */
/* currents in a cutset                                                    */
/*************************************************************************/

red_Q(m,n,i,lx,Q,s)
int *m,*n,i,lx[];
double *Q,*s;
{
    int j,k,l,l1;
    double c,fabs();

    for (j=i;j<*n;j++)
    {
        /* select the largest coefficient in the linear equation */
        c=0.0;
        for (l1=0;l1<*m;l1++)
            if (fabs(Q[j*(*n)+l1])>c)
            {
                c=fabs(Q[j*(*n)+l1]);
                lx[j-i]=l1;
            }

        /* a linear equation on voltage source and current source variables */
        if (fabs(Q[j*(*n)+lx[j-i]])<1.0E-10)
        {
            printf("VOLTAGE_SOURCE LOOP OR CURRENT_SOURCE CUT_SET\n");
```

```
            exit();
        }

        /* get the row_echelon form for the linear equations involving */
        /* capacitor loop voltages or inductor cutset currents         */
        for (k=0;k<*n;k++)
            if (k!=lx[j-i])
                Q[j*(*n)+k]=Q[j*(*n)+k]/Q[j*(*n)+lx[j-i]];
        s[j]=s[j]/Q[j*(*n)+lx[j-i]];
        Q[j*(*n)+lx[j-i]]=1.0;
        for (l=i;l<*n;l++)
            if (l!=j && Q[l*(*n)+lx[j-i]]!=0.0)
            {
                c=Q[l*(*n)+lx[j-i]];
                for (k=0;k<*n;k++)
                {
                    if (k==lx[j-i])
                        Q[l*(*n)+k]=0.0;
                    else
                        Q[l*(*n)+k]-=c*Q[j*(*n)+k];
                }
            }
    }
    for (j=0;j<*n-i;j++)
        for (k=j;k<*n-i;k++)
            if (lx[j]>lx[k])
                switch_i(&lx[j],&lx[k]);
}

/*************************************************************************/
/* Replace the redundant variables.                                      */
/*************************************************************************/

comp_PQ(m,n,i,lx,P,Q,s,d,b1)
int *m,*n,i,lx[];
double *P,*Q,*s,*d,*b1;
{
    int j,k,l;

    for (j=0;j<i;j++)
        for (k=0;k<*m;k++)
            P[j*(*n)+k]=P[j*(*n)+k]*d[k];
    for (j=0;j<*n-i;j++)
        for (l=0;l<i;l++)
        {
            s[l]-=s[i+j]*Q[l*(*n)+lx[j]];
            for (k=0;k<*n;k++)
                if (k!=lx[j])
                    Q[l*(*n)+k]-=Q[l*(*n)+lx[j]]*Q[(i+j)*(*n)+k];
            for (k=0;k<*m;k++)
                if (k!=lx[j])
                    P[l*(*n)+k]-=P[l*(*n)+lx[j]]*Q[(i+j)*(*n)+k];
            for (k=0;k<*n-*m;k++)
                b1[l*(*n-*m)+k]-=P[l*(*n)+lx[j]]*Q[(i+j)*(*n)+k+*m];
        }
}
```

```
/*************************************************************************/
/* Get new P and Q matrices by deleting redundant variables.             */
/*************************************************************************/

new_PQ(m,n,i,lx,P,Q)
int *m,*n,i,*lx;
double *P,*Q;
{
    char *calloc();
    int j,k,jx,kx,l;
    double *px,*qx;

    px=(double *)calloc(i*i,sizeof(double));
    qx=(double *)calloc(i*i,sizeof(double));
    l = *n - i;
    jx=0;
    for (j=0;j<i;j++)
        for (k=0;k<*n;k++)
        {
            for (kx=0;kx<(*n)-i;kx++)
                if (k==lx[kx])
                    break;
            if (kx==(*n-i))
            {
                px[jx]=P[j*(*n)+k];
                qx[jx++]=Q[j*(*n)+k];
            }
        }
    j=0;
    k=0;
    while (j<*n)
    {
        if (j==lx[k])
        {
            k++;
            for (jx=j;jx<(*n-1);jx++)
            {
                (branch_vector+jx)->a=(branch_vector+jx+1)->a;
                (branch_vector+jx)->b=(branch_vector+jx+1)->b;
            }
            for (jx=k;jx<l;jx++)
                lx[jx]--;
            (*m)--;
            (*n)--;
        }
        else
            j++;
    }
    for (jx=0;jx<(*n)*(*n);jx++)
    {
        P[jx]=px[jx];
        Q[jx]=qx[jx];
    }
    cfree(px);
    cfree(qx);
```