NONLINEAR ELECTRONICS (NOEL) PACKAGE 10 SUBST: A PACKAGE FOR THE STEADY-STATE ANALYSIS OF NONLINEAR CIRCUITS DRIVEN BY MULTI-FREQUENCY-COMPONENT SIGNALS

by

Akio Sakamoto and Leon O. Chua

Memorandum No. UCB/ERL M87/54

31 July 1987

NONLINEAR ELECTRONICS (NOEL) PACKAGE 10 SUBST: A PACKAGE FOR THE STEADY-STATE ANALYSIS OF NONLINEAR CIRCUITS DRIVEN BY MULTI-FREQUENCY-COMPONENT SIGNALS

by

Akio Sakamoto and Leon O. Chua

Memorandum No. UCB/ERL M87/54

31 July 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This research is supported in part by the Office of Naval Research under Contract N00014-86-K-0351 and by the National Science Foundation Grant MIP-8614000.

NONLINEAR ELECTRONICS (NOEL) PACKAGE 10 SUBST: A PACKAGE FOR THE STEADY-STATE ANALYSIS OF NONLINEAR CIRCUITS DRIVEN BY MULTI-FREQUENCY-COMPONENT SIGNALS

by

Akio Sakamoto and Leon O. Chua

Memorandum No. UCB/ERL M87/54

31 July 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This research is supported in part by the Office of Naval Research under Contract N00014-86-K-0351 and by the National Science Foundation Grant MIP-8614000.

NOnlinear ELectronics (NOEL) Package 10 SUBST: A Package for the Steady-State Analysis of Nonlinear Circuits Driven by Multi-Frequency-Component Signals

Akio Sakamoto† and Leon. O. Chua‡

ABSTRACT

The programs in this package implements the substitution algorithm [1] which finds the steady-state responses of nonlinear circuits driven by multi-frequency-component signals. The present version accepts linear circuit elements including all four types of linear controlled sources, and 2-terminal nonlinear resistors whose $v-i$ characteristics are described by an explicit relation, an implicit relation, or a piecewise-linear representation. The present version can be easily modified to include other types of nonlinear elements.

† Department of Electrical Engineering, Faculty of Engineering, Tokushima University, 2 Minami-josanjima, Tokushima, 770 Japan

‡ Department of Electrical Engineering and Computer Sciences and Electronics Research Laboratories, University of California, Berkeley, CA 94720, USA

1. Introduction

This package is a set of circuit simulation programs written in C programming language for calculating the steady-state response of nonlinear circuits driven by multi-frequence-component signals. The algorithm implemented in this package is a frequency domain approach and is called the *substitution method*, which is derived in a paper by Ushida, Chua and Sugawara [1]. The basic idea is to find the steady-state responses of the circuit by solving a series of associated time-invariant sensitivity circuit derived by the relaxation method at each frequency component until the residual error is acceptable.

This package uses libraries written in the Nonlinear Electronics Lab by several authors, namely,

(1) COMPLEX -- complex number crunching library; by T. Sugawara

(2) CLNPACK -- LINPACK complex C version; by T. Sugawara

(3) LINPACK -- LINPACK C version; by T. Parker

(4) FFT -- Fast Fourier Transform library; by T. Sugawara & Y. Liao

(5) NPORT -- library for N-port circuit; by A. Deng

(6) SYM -- symbolic differentiation library; by T. Parker

2. Description of the Circuit

The circuit to be analyzed is described in a file with its name ended by ".spc". From now on we call a file describing the circuit a spc-file. An spc-file must follow the (SPICE-like) NOEL circuit description specified by [2]. The available elements types for this package are the following:

(1) Time-varying independent voltage and current sources:

The characteristic for each independent source must be in one of the forms {A*sin(B*t)} or {A*cos(B*t)}, where A and B are constants.

(2) Time-invariant independent voltage and current sources.

(3) 2-terminal linear capacitors and inductors.

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

(5) 2-terminal linear resistors.

(6) 2-terminal nonlinear resistors:

The $v-i$ characteristics of the resistors must be defined in one of the following six forms;

(a) voltage-controlled explicit form $\{i=f(v)\}$.

Example: $\{i=v*v+2*v+1\}$.

(b) current-controlled explicit form $\{v=f(i)\}$.

Example: $\{v=\sin(i)*i+\exp(i/2)\}$.

(c) voltage-controlled implicit form $\{g(v,i)=0;v\}$.

Example: $\{i-(\exp(v-i)-1)=0;v\}$.

(d) current-controlled implicit form $\{g(i,v)=0;i\}$.

Example: $\{\sin(i+3)*v-i=0;i\}$.

(e) voltage-controlled piecewise-linear representation,

$\{v=(i_0,v_0)(i_1,v_1)\cdots(i_n,v_n)\}$.

(f) current-controlled piecewise-linear representation,

$\{i=(v_0,i_0)(v_1,i_1)\cdots(v_n,i_n)\}$.

Where "voltage- (resp., current-) controlled" means that the value of 'i' (resp., 'v') is uniquely determined whenever the value of 'v' (resp.; 'i') is given.

3. Algorithm

3.1. Stage 1: Making intermediate file

Step 1. Extract all the capacitors, inductors, independent sources and nonlinear resistors from the circuit to be analyzed and treat the remaining circuit as a linear resistive n-port. Find the representation for the linear resistive n-port, where n is the total number of the elements of time-varying independent sources, linear capacitors, linear inductors and nonlinear resistors. That is, find two $n \times n$ real matrices $\mathbf{P}$ and $\mathbf{Q}$, and a real column n-

vector **s** such that

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

where **v** and **i** are the port voltage and port current vectors, respectively, for the n-ports (see Note 3.1).

Step 2. Make the intermediate sub-file (file with its name ended by ".sub") that consist of the values for **P**, **Q**, **s** and the information for each port (see Note 3.2), and the frequencies to take into account in FFT analysis at the next stage (see Note 3.3).

[Note 3.1] Step 1 is mainly done by the "linear resistive n-port" formulation program of NOEL package 1 [4].

[Note 3.2] The information differs from the type of port element. For independent source, the frequency and the magnitude of phaser representation are to be included. For L or C, only the value of capacitance or inductance is necessary. For nonlinear resistor defined by explicit or implicit form, the function of definition and its derivative are put in sub-file as character strings. For piecewise-linear elements, the breakpoints are stored in the sub-file.

[Note 3.3] The frequencies to be considered in the next stage are determined from four values, that is, the maximum and the minimum frequencies in the time-varying independent sources and two parameters **m** and **k**. Parameter **m** is the data size of FFT calculation performed at the next stage and parameter **k** is the highest harmonic for each basic frequency component. The default values for **m** and **k** are 128 and 7, respectively, and they can be changed by the user.

3.2. Stage 2: Computing steady-state response

Step 1. Get the information of the circuit to be analyzed from sub-file generated by the previous stage.

Step 2. Set the estimated value for each nonlinear port to zero, and repeat the following substeps:

(i) Analyze the n-port linear circuit for each frequency, where the port of voltage- (resp., current-) controlled nonlinear elements are substituted by voltage (resp., current) sources whose values are equal to the estimated (or initialized) values of voltage (resp. current) on the terminating nonlinear resistors.

(ii) Compute the response of estimated value for each nonlinear element (see Note 3.4).

(iii) Calculate the difference between the response of linear n-port and that of the nonlinear elements for each terminating nonlinear element. At this step, the computation may be interrupted by the user for putting results into rsl-file (file whose name ended with the character string ".rsl") or changing parameters or quitting the program; we shall explain this in detail in the next section.

(iv) Compute the new estimated value for each port terminated with nonlinear resistors by means of relaxation method.

[Note 3.4] When nonlinear element is defined by implicit form, the response for each controlling variable is calculated by Newton-Raphson method.

3.3. Stage 3: Display waveform

Step 1. Read the information for display from rsl-file; for example, the smallest frequency to take into account in FFT analysis and the position of the frequencies to be displayed, etc.

Step 2. Read a set of the spectrum data of voltage and current on each nonlinear element and display the waveform and the spectrum.

4. User's Instruction

4.1. Stage 1: Making intermediate file by program sub1

sub1 is the program to prepare the intermediate file for the next analysis stage. The input file describing the circuit in the NOEL input format [2] must be named "filename.spc", where "filename" is a valid filename chosen by the user. To run the program, type

sub1 filename [CR]

where [CR] means the return key.

The number of model lines and the number of branches are limited to 30 and 70, respectively. These numbers are defined MACRO's in header file "subst.h", and may be changed to appropriate values.

The program requires two parameters to proceed. One is the data size **m** for FFT analysis, and the another is the highest harmonic **k** for each basic frequency component. The default values for them are 128 and 7, respectively. The user can change them by typing the name of the parameter(s) followed by its value. For example

>> Enter as [@m?? @k??] => @k5 [CR]

where parameter **k** is changed to 5 and **m** is not changed.

The program produces two output files. One is the intermediate file named "filename.sub" to be used by **sub2.** The other output file is printed to "stdout" and it contains the information of the circuit and the intermediate file.

4.2. Stage 2: Computing steady-state response by "sub2"

The input file "filename.sub" for **sub2** must be the one generated by **sub1.** The command line for running **sub2** is

sub2 filename [CR]

The program produces two output files. One named "filename.rsl" contains the results from the analysis to be used as the input for the next display stage. The other is printed to "stdout" and contains the information of the convergence ratio.

The program requires three parameters from the user as input in addition to the sub-file. The parameters must be set before and during the computation. Two of them are related to the convergence of the algorithm and the other is for setting interruption.

(1) **p** -- the constant for relaxation algorithm. $0 \le p \le 1$. Its initial value is 0.3. We usually expect a faster convergence ratio for a greater **p**, but the algorithm may diverge if **p** is too large.

(2) **q** -- the scaling constant for Newton-Raphson algorithm. $0 \leq q \leq 1$. The initial value is 1.0. For stiff equations, the value of **q** should be small, for example, 0.3.

(3) **s** -- the number of iterations for Newton-Raphson algorithm between interruptions, and the initial value is 10.

When setting the parameters, the current value for each parameter is displayed as the default values. You enter only those parameters that you want to change. For example,

>> Enter as [@p?? @q?? @s??] => @s3 @p.5 [CR]

where parameter **p** is changed to 0.5, **s** is changed to 3 and **q** remains unchanged.

At the interruption you can decide to output the information of the voltage and the current values of each nonlinear elements (expressed in form of real and imaginary parts for each frequency component) to the rsl-file and/or change the parameters or quit the program. The interruption occurs when one of the following conditions holds; (a) current error is smaller than one thousandth of the initial error, (b) current error is greater than ten times of the initial error, or (c) number of iteration is a multiple of **s**, where the initial error is obtained by the first iteration.

4.3. Stage 3: Display waveform by sub3

The command line is similar to the previous stages; namely,

sub3 filename [CR]

Several questions are asked by the program.

(1) Do you need this graph ? (y/n) >

This is asked whenever a new variable saved in the rsl-file is to be displayed.

(2) [wave] Neglect D.C. component ? (y/n) >

If the answer is **y**, The dc component is neglected in the displayed time domain waveform.

(3) [wave] Y-axis maximum number ? (#.##) >

If you cannot decide the proper value, enter reasonable guess. The maximum value of waveform is displayed after drawing and you can re-draw with the proper value.

(4) [wave] X-axis from #.#T ? (0, 0.5) >

The value of X-axis means the number of multiples of period T to be displayed. The minimum value for X-axis must be either 0 or 0.5.

(5) [wave] X-axis to #.#T ? (0.5, 1, 1.5, 2) >

This is the maximum value of the X-axis.

(6) [wave] X-axis step size ? (1..6, 10) >

Enter 1, if you want the finest drawing. Enter 10, if you merely need the maximum value.

(7) [wave] Do you need hard copy ? (y/n) >

For hard copy of the displayed waveform.

(8) [spectrum] Y-axis minimum number ? (10^#) >

Spectrum is displayed by bar graph. The unit on X-axis is the minimum angular frequency of FFT analysis. Y-axis is in logarithmic scale. The answer to this should be the relative lower bound for the display with respect to the frequency component having the maximum magnitude. For example, if -3 is entered, then all the frequency components whose magnitudes are larger than one thousandth of the maximum value (60 dB below) is displayed.

(9) [spectrum] Do you need hard copy ? (y/n) >

For hard copy.

(10) Do you need re-drawing ? (y/n) >

For rescaling the drawing.

5. Diagnosis

5.1. sub1

1. ILLEGAL R-PORT

This message appears when the $v-i$ characteristic of nonlinear resistor is not legally defined. (See Section 2.1 for valid definitions).

2. DERIVATION IS TOO LONG

   The definition for nonlinear resistor is legal, but the expression is too long to be stored in the character buffer of the program.

3. ILLEGAL C-PORT

4. ILLEGAL L-PORT

   For C and L, this version permits linear characteristic only. This message appears when the relation field for L or C begins with "{".

5. ILLEGAL PORT TYPE

   Illegal terminating port element encountered. (See Section 2.1 for allowed elements).

6. NO INDEPENDENT SOURCE

   This version requires at least one time-varying independent source.

7. ILLEGAL SOURCE

8. ILLEGAL SOURCE AMPLITUDE

9. ILLEGAL SOURCE RELATION

   Messages 7-9 are due to illegal definition of independent source. (See Section 2.1 for detail).

5.2. sub2

1. CAN'T ALLOCATE SPACE FOR nptr->xxx

   Too many nonlinear elements.

2. ABNORMAL VALUE = nn FROM 'cgefa' in solve_ln

3. ABNORMAL VALUE = nn FROM 'cgefa' in solve_sn

   Function "cgefa" solves the linear system with complex numbers. "solve_ln" and "solve_sn" are functions used in substeps (i) and (iv) of Step 2, respectively. Check the port structure whether it forms particular cut-sets or loops.

4. Floating exception

   The residual error may be two large.

5.3. sub3

1. CAN'T allocate space --> xxxxx

   This massage rarely happens if the rsl-file is legal as long as the computer have enough memory.

5.4. Others

1. CAN'T ALLOCATE SPACE FOR xxxxx

   Out of memory.

2. CANNOT OPEN FILE xxxxx

   Check the existence of file "xxxxx".

3. Messages from libraries listed in Introduction may be displayed. If it is the case, see [4] for the library NPORT or appropriate documents.

References

[1] A. Ushida, L. O. Chua and T. Sugawara, "A substitution algorithm for solving nonlinear circuits with multi-frequency components," in press.

[2] A. C. Deng and L. O. Chua, "Nonlinear Electronics package 0: general description," ERL Memo., UCB/ERL M86/22, University of California, Berkeley, 1986.

[3] A. Ushida and L. O. Chua, "A relaxation method for frequency-domain approach," Internal memo.

[4] A. C. Deng and L. O. Chua, "NOnlinear Electronics package 1: linear circuit formulations, n-port representations, and state equations," ERL Memo., UCB/ERL M86/23, University of California, Berkeley, 1986.

Appendix I.

We list some examples in this appendix to provide more detailed information for the user. Each example is designed to emphasize one specific aspects of the program. The following are discussions for the examples along with the input and output listed for each example.

Example 1.

The algorithm implemented in this package is essentially a frequency domain approach. The frequency components to be calculated by the program for each nonlinear element are determined by that of the time-varying independent sources included in the circuit. Therefore, the circuit to be analyzed must include at least one (sinusoidal) time-varying independent source. This is an example of a circuit having only one sinusoidal source.

The first list for Example 1 is its spc-file "example1.spc". It then shows the sub-file "example1.sub" saved by **sub1** with **m** set to 128 and **k** to 15.

The third list of this example is the rsl-file "example1.rsl" from execution of **sub2.** All parameters remains as default values. After 7 iterations, the residual error = 5.77E-01 which is smaller than one thousandth of initial error. Therefore, the solution at that time is forced to be saved into rsl-file "example1.rsl" and the computation halts.

Fig. 1 (a) shows the spectrum for the voltage across the nonlinear resistor R_g and Fig. 1 (b) is the corresponding time domain waveform.

```
* Example 1:  Simple example with unique exciting frequency
*
*           +---+ 2 +---+ 3 +---+
*   1 *-----+ R +---+ L +---+ C +----*------* 4
*     |     +---+   +---+   +---+    |      |
*    +++                              +++   +++++ nonlinear
*    |e| Vin=5*cos(4.44*t)          Rp| |   |   |  resistor
*    +++                              +++   ----- i=v+v*v+v*v*v
*     |                                |      |
*   0 *--------------------------------*------* 0
*
*  sinusoidal voltage source
Vin    1     0     {5*cos(4.44*t)}
*  next three elements are linear
R      1     2      1
L      2     3      1
C      3     4      1
Rp     4     0      5
*  voltage-controlled nonlinear resistor
Rg     4     0     {i=v+v*v+v*v*v}
.end
```

```
n = 4; indp = 1; nonl = 1;
matrix P:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00

matrix Q:
   1.0000000000e+00    0.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   1.0000000000e+00    1.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -1.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -5.0000000000e+00

vector s:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00


>type = 1; var1 = i; var2 = 0; name = Vin; node1 = 1; node2 = 0;
relat = 5.00e+00*cos(4.440e+00*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   4.4400000000e+00;
>type = 4; var1 = v; var2 = 0; name = C; node1 = 3; node2 = 4;
value =   1.0000000000e+00;
>type = 3; var1 = i; var2 = 0; name = L; node1 = 2; node2 = 3;
value =   1.0000000000e+00;
>type = 5; var1 = v; var2 = i; name = Rg; node1 = 4; node2 = 0;
relat = v*(1.00e+00 + v*(1.00e+00 + v))
deriv = 1.00e+00 + v*(2.00e+00 + 3.00e+00*v)

<omega =   4.4400000000e+00; M = 8; nindx = 9;
vector indx:
    0     1     2     3     4     5     6     7     8
```

```
    4.4400000000e+00
1
1
9
0 1 2 3 4 5 6 7 8
Rg
v
    5.1652986901e-04
   -1.3787458934e-01     0.0000000000e+00
    3.0227442553e-01     7.3337610036e-01
    6.0368430653e-02    -6.7470198319e-02
    4.8660231133e-02     1.8791990922e-02
   -2.2649841832e-03    -3.1713597116e-02
    4.2881175706e-03    -1.4107341575e-03
   -8.4382362939e-03    -7.2564217814e-03
    3.8710879853e-04     6.9320666738e-04
   -5.8935537315e-03     2.8323663275e-04

Rg
i
    5.1652986901e-04
    2.7902717730e-02     0.0000000000e+00
    3.5477518868e-01     8.6974678656e-01
   -2.0586288851e-02     7.7665465732e-03
   -9.0902725702e-03    -7.6589249872e-03
   -1.3559582911e-03     6.3485668439e-03
   -9.7243963431e-04     1.0185952250e-04
    1.6029252785e-03     1.9523372661e-03
   -1.0003845953e-04    -4.6600106348e-05
    1.4168857193e-03     0.0000000000e+00
```

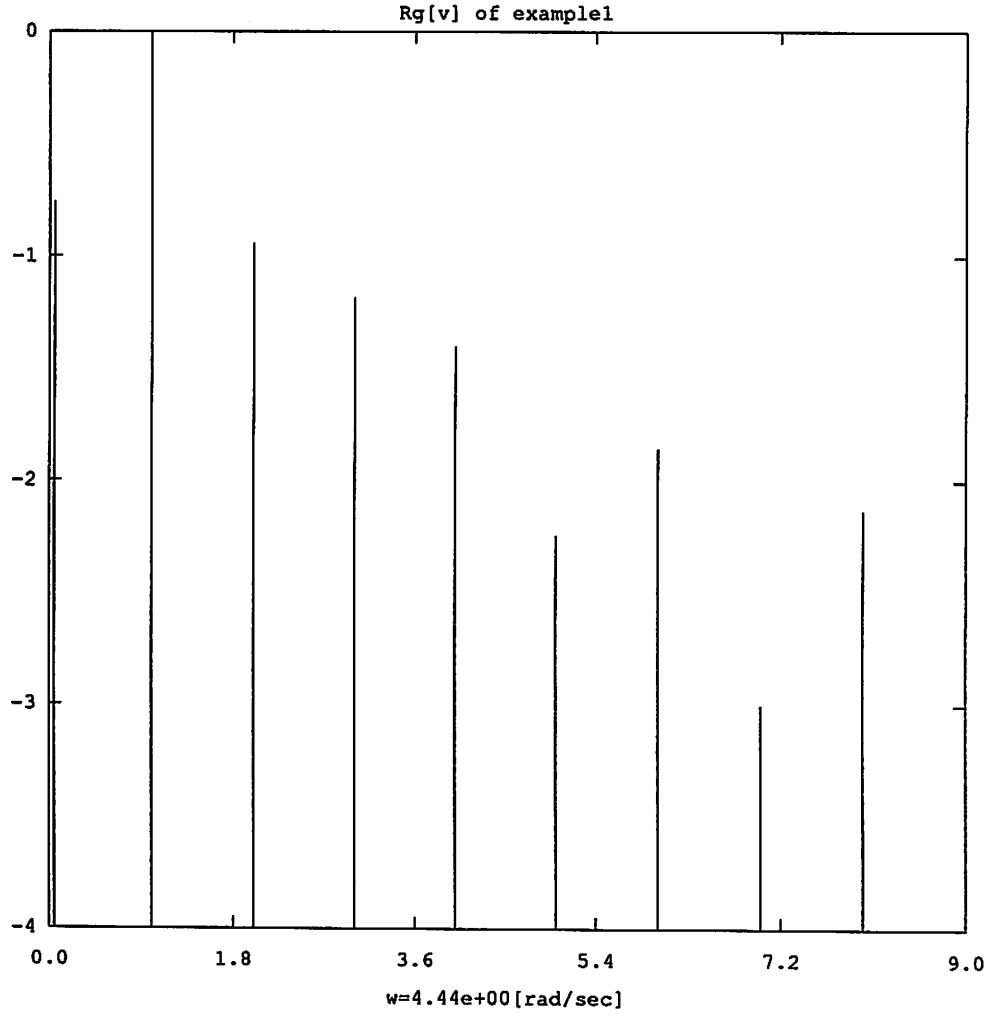


Fig. 1 (a)

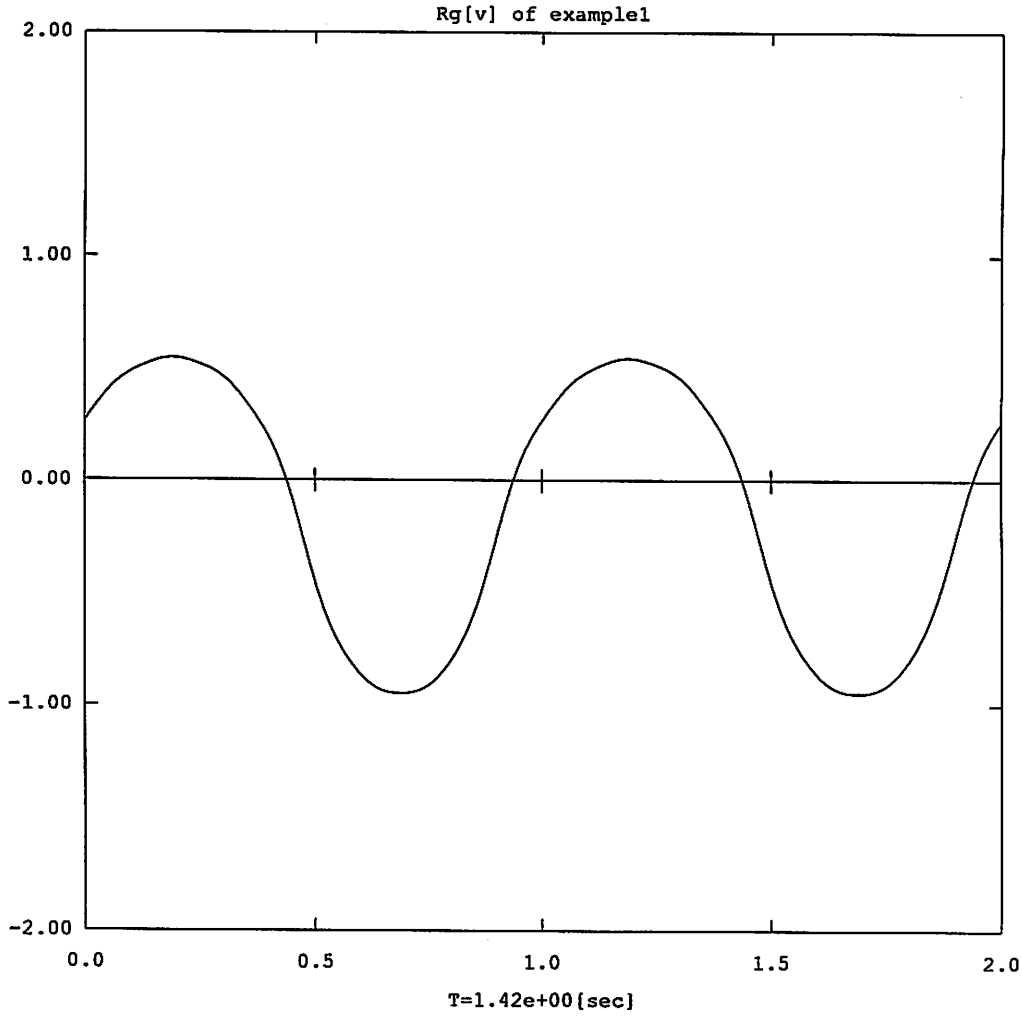


Fig. 1 (b)

Example 2.

One of the features of this program is the capability of analyzing a circuit driven by multi-frequency-component signals. The circuit for Example 2 is very similar to that of Example 1, the only difference is that it has two voltage sources with different frequencies. The angular frequencies for the two independent sources V_1 and V_2 are 4.44 and 35.5, respectively. The program decides the lowest frequency to be analyzed is 2.2188.

We used the default values for all parameters of the programs. Files "example2.spc", "example2.sub" and "example2.rsl" are listed in the following and Fig. 2 (a) and (b) show the spectrum and waveform of the voltage across R_g, respectively.

```
* Example 2:  Example with 2 frequencies.
*
*           +---+ 2 +---+ 3 +---+
*   1 *-----+ R +---+ L +---+ C +----*------* 4
*     |     +---+   +---+   +---+    |      |
*    +++                             +++   +++++
*    |e| V1=5*cos(4.44*t)             | |   |   |  nonlinear
*    +++                          Rp | |   |   |  resistor
*     |                             | |   |   |    Rg
*    +++                            +++   ----- i=v+v*v+v*v*v
*    |e| V2=5*cos(35.5*t)             |      |
*    +++                             |      |
*     |                              |      |
*   0 *------------------------------*------* 0
*
*
*
*  sinusoidal voltage source is  5*cos(4.44*t) + 5*cos(35.5*t)
V1    10     0     {5*cos(4.44*t)}
V2     1    10     {5*cos(35.5*t)}
R      1     2      1
L      2     3      1
C      3     4      1
Rp     4     0      5
Rg     4     0     {i=v+v*v+v*v*v}
.end
```

```
n = 5; indp = 2; nonl = 1;
matrix P:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -1.0000000000e+00
matrix Q:
   1.0000000000e+00    0.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    1.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   1.0000000000e+00    0.0000000000e+00   -1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00
vector s:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00

>type = 1; var1 = i; var2 = 0; name = V1; node1 = 10; node2 = 0;
relat = 5.00e+00*cos(4.440e+00*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   4.4400000000e+00;
>type = 1; var1 = i; var2 = 0; name = V2; node1 = 1; node2 = 10;
relat = 5.00e+00*cos(3.550e+01*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   3.5500000000e+01;
>type = 4; var1 = v; var2 = 0; name = C; node1 = 3; node2 = 4;
value =   1.0000000000e+00;
>type = 3; var1 = i; var2 = 0; name = L; node1 = 2; node2 = 3;
value =   1.0000000000e+00;
>type = 5; var1 = v; var2 = i; name = Rg; node1 = 4; node2 = 0;
relat = v*(1.00e+00 + v*(1.00e+00 + v))
deriv = 1.00e+00 + v*(2.00e+00 + 3.00e+00*v)

<omega =   2.2187500000e+00; M = 128; nindx = 64;
vector indx:
    0     2     4     6     8    10    12    14    16    18
   20    22    24    26    28    30    32    34    36    38
   40    42    44    46    48    50    52    54    56    58
   60    62    64    66    68    70    72    74    76    78
   80    82    84    86    88    90    92    94    96    98
  100   102   104   106   108   110   112   114   116   118
  120   122   124   126

.
```

```
    2.2187500000e+00
2
2 16
64
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36
38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72
74 76 78 80 82 84 86 88 90 92 94 96 98 100 102 104 106
108 110 112 114 116 118 120 122 124 126
Rg
v
    2.5931375995e-04
   -1.3787510050e-01     0.0000000000e+00
    3.0315946328e-01     7.2981504141e-01
    5.9420464137e-02    -6.6241802515e-02
    4.8831314967e-02     1.3076938130e-02
   -5.3221793964e-03    -2.9237927550e-02
   -1.1117910851e-03    -1.0236241743e-02
   -1.8092551288e-02     6.7530541197e-03
   -1.2501128723e-02    -2.4101892290e-03
    1.3836858179e-03     8.4939707317e-02
    1.2396698157e-02    -4.5628245366e-03
    1.1798838094e-02     1.2021517117e-02
    3.8543216014e-03    -9.6992559291e-03
    3.0414227610e-03     8.9536862071e-04
   -1.7579551079e-03    -5.3902106627e-03
    1.9174992694e-04     1.2254899666e-03
   -1.6404654807e-03    -2.0184466015e-03
    7.4685577554e-05     1.2680068003e-03
    2.5244416576e-04     1.4228648557e-03
    1.2063225056e-03    -2.2801916311e-04
    7.4542750369e-04     5.4374727668e-05
    1.6246341429e-04    -1.1225898484e-03
   -1.0339987565e-04    -1.8895725753e-04
   -5.5784901457e-04    -4.6432754942e-04
   -1.6626934060e-04     2.8671217813e-04
   -3.0291318502e-04     3.1013095980e-05
    1.3577916947e-04     2.9492835127e-04
    3.2450264824e-05     7.3609396210e-05
    2.1244511993e-04    -2.0038478311e-05
    2.5465097122e-05    -6.7127254862e-05
    1.4497871103e-05    -1.3667481121e-04
   -7.6386622780e-05    -1.3892853568e-05
   -7.2209265151e-05    -3.0317296028e-05
   -2.8875822999e-05     7.5578411255e-05
   -1.4780654984e-05     2.1314877706e-05
    4.2879930900e-05     4.8567484142e-05
    1.6679421363e-05    -1.0593473166e-05
    3.4576008995e-05    -1.0522611095e-05
   -9.0197968889e-06    -2.4943517764e-05
   -3.1168406532e-06    -1.5481092745e-05
   -2.1366057131e-05     7.0989311796e-07
   -7.4076684339e-06     4.2126499291e-06
   -3.1598004000e-06     1.4999730377e-05
    4.8892438228e-06     3.9547163100e-06
    8.8637966645e-06     4.9593722062e-06
    3.8242556985e-06    -5.8475120496e-06
    3.5182311271e-06    -3.4973718877e-06
   -4.1283833887e-06    -5.0773504272e-06
   -1.8989751752e-06    -7.1451500079e-07
   -4.3067138640e-06     1.3849751947e-06
    1.1699751583e-07     2.2490418219e-06
    2.2450114207e-07     2.5333057119e-06
    2.1511011525e-06     2.0229888328e-07
    1.3758175111e-06    -6.4691862672e-08
```

```
 5.2179760016e-07   -1.6901920659e-06
-1.9366277689e-07   -7.2891292313e-07
-1.1007590858e-06   -7.1675226385e-07
-5.4858601375e-07    3.8727236152e-07
-6.2062670791e-07    5.4172563033e-07
 3.3870468483e-07    6.2398651578e-07
 2.9007013956e-07    3.3949356926e-07
 5.7763118891e-07   -1.2037904435e-07
 1.9806640769e-07   -2.7186374467e-07
 1.8795558541e-08   -4.1092895240e-07
-2.7861533671e-07   -1.8464207284e-07

Rg
i
 2.5931375995e-04
 2.7506062854e-02    0.0000000000e+00
 3.5385274219e-01    8.7047634366e-01
-1.9955092584e-02    7.2327412675e-03
-8.9741562052e-03   -6.3599300300e-03
-4.6695656324e-04    6.3400882738e-03
-3.1379848374e-04    2.2415413825e-03
 3.8334688247e-03   -6.1738965771e-04
 2.3100882935e-03    8.4431111133e-04
 6.1190656515e-03    1.2361761680e-01
-2.4973273968e-03    5.4074703119e-04
-2.0016837547e-03   -2.6294525138e-03
-1.0033581634e-03    1.9382013986e-03
-6.6160321671e-04   -2.2055041324e-04
 2.2229050868e-04    1.1002159938e-03
-3.4223250877e-05   -2.6479742782e-04
 3.2825529139e-04    4.0750919670e-04
 3.9481567343e-05   -2.5922352999e-04
 8.3399762003e-07   -2.6787068852e-04
-2.6937631521e-04    4.4167845788e-05
-1.6704766012e-04   -3.2583303080e-05
-5.9970781244e-05    2.1209015339e-04
 2.1470561771e-05    4.0024941164e-05
 1.0621403536e-04    1.0584237195e-04
 4.5607981166e-05   -5.2428544779e-05
 6.9405129976e-05    1.6437502414e-06
-2.4536778435e-05   -6.2226023923e-05
-6.4417620855e-06   -2.1183144025e-05
-4.7495500048e-05   -3.0807621280e-06
-6.9707627123e-06    1.3846948719e-05
-9.9925841914e-06    2.9680940055e-05
 1.7463477285e-05    5.8066688977e-06
 1.5256945122e-05    9.4212890220e-06
 1.0797098631e-05   -1.4625005841e-05
 4.0037040428e-06   -5.2027001491e-06
-7.5804173935e-06   -1.2774187294e-05
-5.2370845887e-06    8.7421386383e-07
-9.2718743335e-06    9.7733239260e-07
 8.9826799897e-07    6.1923117187e-06
 6.6913007359e-08    4.4920758589e-06
 5.7973433060e-06    1.3674826982e-06
 2.0778518750e-06   -5.3893195499e-07
 2.0741009707e-06   -3.6523679552e-06
-1.3365617140e-06   -1.7091067871e-06
-1.9108455112e-06   -2.0169245542e-06
-1.7189434447e-06    1.1604576897e-06
-1.1492349636e-06    9.2668219340e-07
 6.5980962745e-07    1.8351773881e-06
 5.9604085260e-07    4.8765998553e-07
 1.4327416322e-06   -5.4228218800e-08
```

1.9723049621e-07	-7.4800273778e-07
2.0678153935e-07	-8.6052340442e-07
-7.2932159514e-07	-3.3096912372e-07
-4.3901649425e-07	-8.4982288187e-08
-4.3445730467e-07	5.1303610331e-07
4.2283963094e-08	3.2163796641e-07
2.8052656404e-07	3.7480591086e-07
3.0797480688e-07	-7.9684489257e-08
2.7464701980e-07	-1.5554335685e-07
-5.2063707572e-08	-3.0729170598e-07
-1.0556842100e-07	-1.5635086266e-07
-2.7434680230e-07	-1.7133699510e-08
-9.1587336858e-08	1.3717763492e-07
-5.8934214326e-08	1.7868950144e-07
1.8558677781e-07	1.1215348716e-07

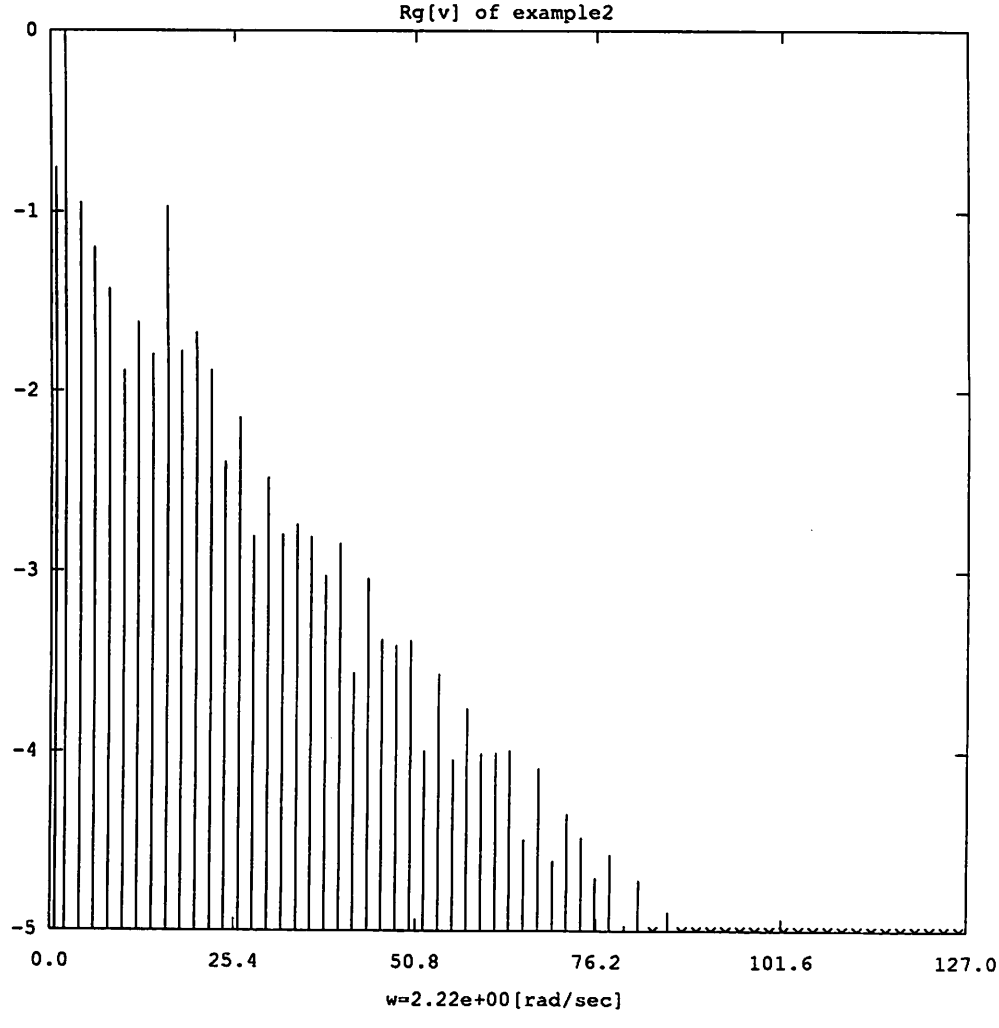

Rg[v] of example2
0
-1
-2
-3
-4
-5
0.0
25.4
50.8
76.2
101.6
127.0
w=2.22e+00[rad/sec]

Fig. 2 (a)

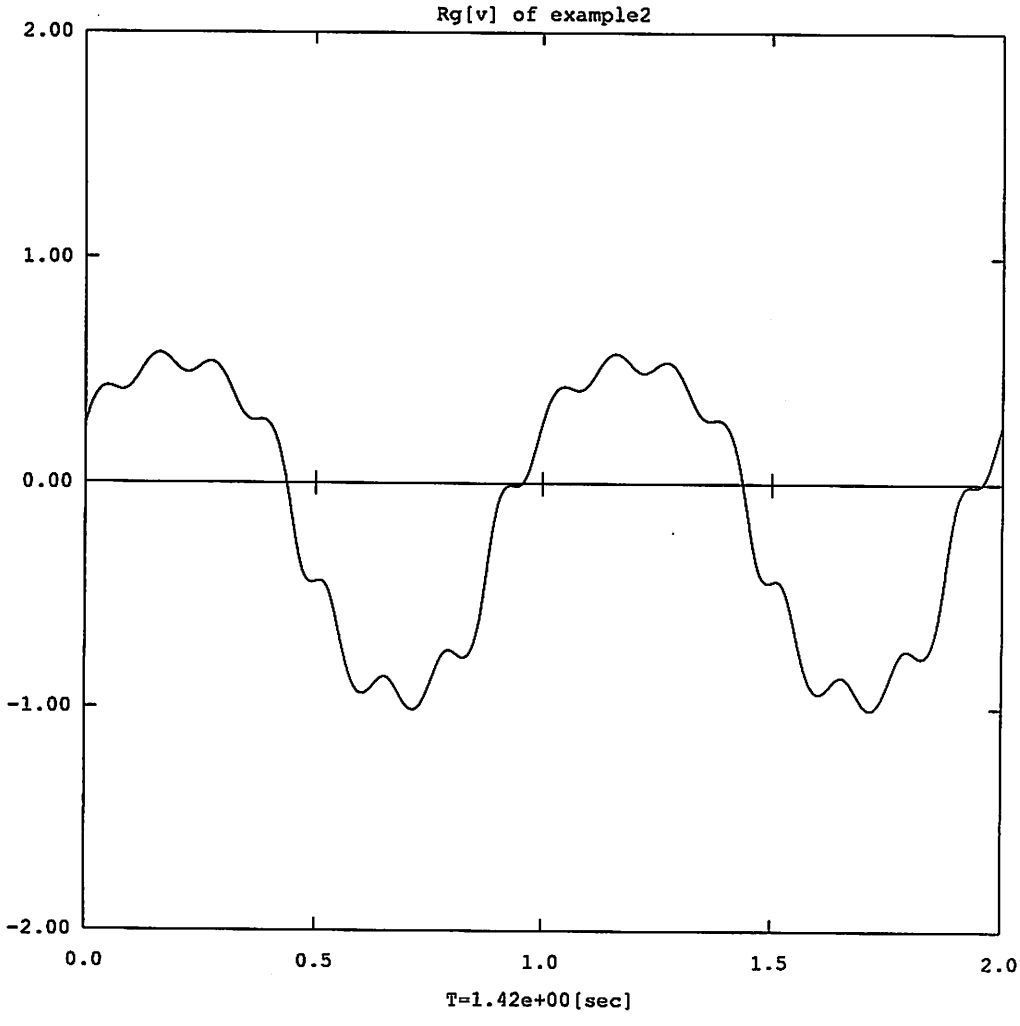


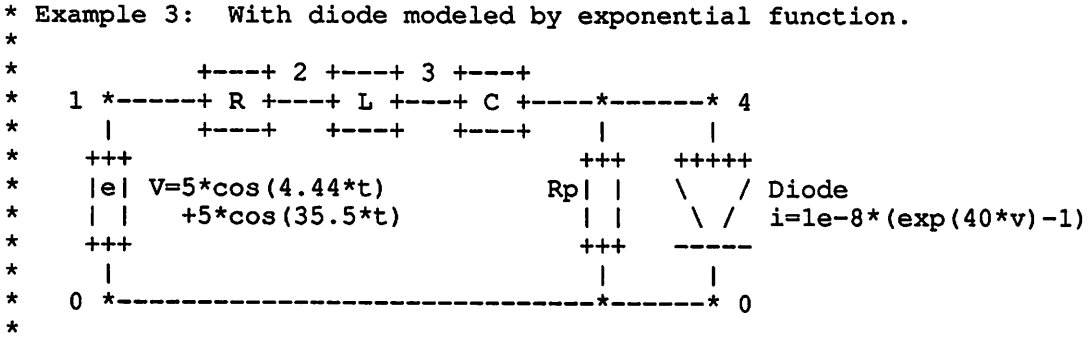
Fig. 2 (b)

Example 3. (A non-convergent case.)

For this example, we replace the nonlinear resistor in Example 1 by a P-N junction diode which is modeled by the exponential P-N junction law as shown in the spc-file for this example. It is well known that the value of the exponential function increases rapidly as the value of its argument increases. This kind of stiffness may cause divergence of the algorithm or prolonged computation time as is the case of this example. Our program fails to find the solution for this example.

We list files "example2.spc" and "example2.sub" in the following where for **sub1** we set the two parameters as **m** = 128 and **k** = 5.

For **sub2,** at first we used default parameter values as shown in the next list of "example3.out2" which was the saved information displayed on the computer terminal. The algorithm diverged at the first iteration. Then We tried smaller values for **p** and **q**. As seen from the list, although the algorithm did not diverge, the error reduced very slowly, the simulation became very inefficient.




```
V1    10     0     {5*cos(4.44*t)}
V2     1    10     {5*cos(35.5*t)}
R      1     2      1
L      2     3      1
C      3     4      1
Rp     4     0      5
*  diode (voltage-controlled nonlinear resistor)
Rd     4     0     {i=1e-8*(exp(40*v)-1)}
.end
```

```
n = 5; indp = 2; nonl = 1;
matrix P:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -1.0000000000e+00
matrix Q:
   1.0000000000e+00    0.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    1.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   1.0000000000e+00    0.0000000000e+00   -1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00
vector s:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00

>type = 1; var1 = i; var2 = 0; name = V1; node1 = 10; node2 = 0;
relat = 5.00e+00*cos(4.440e+00*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   4.4400000000e+00;
>type = 1; var1 = i; var2 = 0; name = V2; node1 = 1; node2 = 10;
relat = 5.00e+00*cos(3.550e+01*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   3.5500000000e+01;
>type = 4; var1 = v; var2 = 0; name = C; node1 = 3; node2 = 4;
value =   1.0000000000e+00;
>type = 3; var1 = i; var2 = 0; name = L; node1 = 2; node2 = 3;
value =   1.0000000000e+00;
>type = 5; var1 = v; var2 = i; name = Rd; node1 = 4; node2 = 0;
relat = -1.00e-08*(1.00e+00 - exp(4.00e+01*v))
deriv = 4.00e-07*exp(4.00e+01*v)

<omega =   1.6904761905e+00; M = 128; nindx = 41;
vector indx:
    0     3     6     9    12    15    18    21    24    27
   30    33    36    39    42    45    48    51    54    57
   60    63    66    69    72    75    78    81    84    87
   90    93    96    99   102   105   108   111   114   117
  120
```

```
************  SUBSTITUTION ALGORITHM  ************ <ckts/example3>
  M = 128,  nindx = 41
  # of ports = 5 (# of nonlinear = 1, # of indp. source = 2)

 ** parameters: p = 0.30, q = 1.00, d = 5.00, s = 10  and  r = 0 **
=== iteration  0 ===
        error(Rd) = 5.07e-01 //  ratio = 1.000
=== iteration  1 ===
        error(Rd) = 4.24e+54


************  SUBSTITUTION ALGORITHM  ************ <ckts/example3>
  M = 128,  nindx = 41
  # of ports = 5 (# of nonlinear = 1, # of indp. source = 2)

 ** parameters: p = 0.20, q = 1.00, d = 5.00, s = 5  and  r = 0 **
=== iteration  0 ===
        error(Rd) = 5.07e-01 //  ratio = 1.000
=== iteration  1 ===
        error(Rd) = 4.24e+54


************  SUBSTITUTION ALGORITHM  ************ <ckts/example3>
  M = 128,  nindx = 41
  # of ports = 5 (# of nonlinear = 1, # of indp. source = 2)

 ** parameters: p = 0.10, q = 0.20, d = 5.00, s = 5  and  r = 0 **
=== iteration  0 ===
        error(Rd) = 5.07e-01 //  ratio = 0.200
=== iteration  1 ===
        error(Rd) = 6.29e+03


************  SUBSTITUTION ALGORITHM  ************ <ckts/example3>
  M = 128,  nindx = 41
  # of ports = 5 (# of nonlinear = 1, # of indp. source = 2)

 ** parameters: p = 0.05, q = 0.10, d = 5.00, s = 5  and  r = 0 **
=== iteration  0 ===
        error(Rd) = 5.07e-01 //  ratio = 0.100
=== iteration  1 ===
        error(Rd) = 4.55e-01 //  ratio = 0.100
=== iteration  2 ===
        error(Rd) = 5.95e+01 //  ratio = 0.100
=== iteration  3 ===
        error(Rd) = 3.32e+01 //  ratio = 0.100
=== iteration  4 ===
        error(Rd) = 1.85e+01 //  ratio = 0.100
=== iteration  5 ===
        error(Rd) = 1.02e+01 //  ratio = 0.100
=== iteration  6 ===
        error(Rd) = 5.74e+00 //  ratio = 0.100
=== iteration  7 ===
        error(Rd) = 3.35e+00 //  ratio = 0.100
=== iteration  8 ===
        error(Rd) = 2.04e+00 //  ratio = 0.100
=== iteration  9 ===
        error(Rd) = 1.29e+00 //  ratio = 0.100
=== iteration 10 ===
        error(Rd) = 8.68e-01
 ** parameters: p = 0.10, q = 0.20, d = 5.00, s = 5  and  r = 0 **  //  ratio = 0.200
=== iteration 11 ===
        error(Rd) = 5.29e-01 //  ratio = 0.200
```

```
=== iteration 12 ===
       error(Rd) = 4.13e-01 //  ratio = 0.200
=== iteration 13 ===
       error(Rd) = 3.78e-01 //  ratio = 0.200
=== iteration 14 ===
       error(Rd) = 3.65e-01 //  ratio = 0.200
=== iteration 15 ===
       error(Rd) = 3.57e-01
 ** parameters: p = 0.20, q = 0.50, d = 5.00, s = 5  and  r = 0 **  //  ratio = 0.500
=== iteration 16 ===
       error(Rd) = 3.37e-01 //  ratio = 0.500
=== iteration 17 ===
       error(Rd) = 3.28e-01 //  ratio = 0.500
=== iteration 18 ===
       error(Rd) = 3.23e-01 //  ratio = 0.500
=== iteration 19 ===
       error(Rd) = 3.18e-01 //  ratio = 0.500
=== iteration 20 ===
       error(Rd) = 3.13e-01
 ** parameters: p = 0.40, q = 0.70, d = 5.00, s = 5  and  r = 0 **  //  ratio = 0.700
=== iteration 21 ===
       error(Rd) = 3.05e-01 //  ratio = 0.700
=== iteration 22 ===
       error(Rd) = 2.97e-01 //  ratio = 0.700
=== iteration 23 ===
       error(Rd) = 2.90e-01 //  ratio = 0.700
=== iteration 24 ===
       error(Rd) = 2.83e-01 //  ratio = 0.700
=== iteration 25 ===
       error(Rd) = 2.76e-01
 ** parameters: p = 0.60, q = 1.00, d = 5.00, s =
```

Example 4. (Compensation resistor modeling.)

For the circuit used in Example 3, to model the P-N junction diode there is a practical technique that transform the stiff diode model into an equivalent weakly nonlinear element [3]. That is, by introducing compensation resistors R_c and $-R_c$ in series with the diode, we can model the diode as a series circuit made of a negative resistor $-R_c$ and a nonlinear resistor whose $v-i$ characteristic is $i = I_s\,(\exp((v-R_c\,i)/V_T) - 1))$, which can be formulated by an implicit function as shown in the spc-file listing for Example 4.

The two parameters for **sub1** are **m** = 128 and **k** = 5 and the parameters for **sub2** are default values. Files "example4.sub" and "example4.rsl" are listed after the spc-file "example4.spc". Fig. 3 (a) and (b) are the spectrum and the waveform of the current on the diode in the circuit.

```
* Example 4:  New diode modeling with compensation resistors.
*
*
V1     10      0      {5*cos(4.44*t)}
V2      1     10      {5*cos(35.5*t)}
R       1      2       1
L       2      3       1
C       3      4       1
Rp      4      0       5
*  compensation negative resistor
Rc      4      5      -1
*  modified diode (voltage-controlled implicit form)
Rd      5      0      {i-1e-8*(exp(40*(v-i))-1)=0;v}
.end
```

```
n = 5; indp = 2; nonl = 1;
matrix P:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
  -1.0000000000e+00   -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00
matrix Q:
   1.0000000000e+00    0.0000000000e+00    0.0000000000e+00    1.0000000000e+00
   0.0000000000e+00   -1.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   1.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -6.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00
vector s:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00

>type = 1; var1 = i; var2 = 0; name = V1; node1 = 10; node2 = 0;
relat = 5.00e+00*cos(4.440e+00*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   4.4400000000e+00;
>type = 1; var1 = i; var2 = 0; name = V2; node1 = 1; node2 = 10;
relat = 5.00e+00*cos(3.550e+01*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   3.5500000000e+01;
>type = 4; var1 = v; var2 = 0; name = C; node1 = 3; node2 = 4;
value =   1.0000000000e+00;
>type = 3; var1 = i; var2 = 0; name = L; node1 = 2; node2 = 3;
value =   1.0000000000e+00;
>type = 7; var1 = v; var2 = i; name = Rd; node1 = 5; node2 = 0;
relat = i + 1.00e-08*(1.00e+00 - exp(-4.00e+01*(i - v)))
dffrn = 1.00e+00 + 4.00e-07*exp(-4.00e+01*(i - v))
deriv = 4.00e-07*exp(-4.00e+01*(i - v))/(1.00e+00 + 4.00e-07*exp(-4.00e+01*(i - v)

<omega =   1.6904761905e+00; M = 128; nindx = 41;
vector indx:
    0     3     6     9    12    15    18    21    24    27
   30    33    36    39    42    45    48    51    54    57
   60    63    66    69    72    75    78    81    84    87
   90    93    96    99   102   105   108   111   114   117
  120
```

```
    1.6904761905e+00
2
3 21
41
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51
54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
102 105 108 111 114 117 120
Rd
v
    5.7126377356e-04
   -8.3696322381e-01     0.0000000000e+00
    1.6490649997e+00     1.9535757526e+00
   -2.0972532284e-02    -5.9587773380e-01
    9.3096971815e-02    -2.4425288448e-02
   -9.3699477052e-02    -7.6618749685e-02
   -1.5929532909e-02    -3.8116609248e-02
   -1.7145326314e-01    -4.4248320052e-02
    3.7052241989e-02     4.3105032769e-01
    1.2199043881e-01    -1.2381076444e-01
    1.3594688406e-02     1.8680843050e-02
   -2.9085119907e-02    -4.3317193739e-02
    1.5778927368e-02    -9.6469752416e-03
   -2.2627036798e-02     1.3609969686e-02
   -6.1392451821e-03    -1.2569278553e-02
    9.4035212000e-03     1.3531692664e-02
   -8.9426697118e-03     4.4187543822e-03
    9.5041331438e-03    -7.3320657005e-03
    4.0686185288e-03     6.2849159106e-03
   -5.5366591549e-03    -7.4764256414e-03
    4.8678948804e-03    -4.3638157458e-03
   -5.8696367642e-03     3.7075855856e-03
   -4.4246571453e-03    -4.4653250077e-03
    2.3330854031e-03     4.1648070967e-03
   -4.3644849581e-03     3.9474981123e-03
    2.5761154899e-03    -1.6418390104e-03
    3.1243251026e-03     4.0898876179e-03
   -1.5409848185e-03    -1.4142278946e-03
    3.4731201440e-03    -2.3428383393e-03
   -8.4695770589e-04     1.6643244099e-03
   -1.9092429156e-03    -2.7162036213e-03
    1.6489389932e-03     7.0290698425e-04
   -2.1187669607e-03     1.7991776640e-03
    6.8270456347e-04    -1.4008573591e-03
    1.8014612398e-03     1.8204573245e-03
   -1.0457792008e-03    -5.7203484847e-04
    1.7430900803e-03    -1.6927050495e-03
   -3.9037402029e-04     7.9109525362e-04
   -1.4585554394e-03    -1.6921476480e-03
    6.3263781299e-04     3.1722596728e-04
   -1.6581367945e-03     1.1381306740e-03
    8.1187099382e-04    -4.5175285010e-04

Rd
i
    5.7126377356e-04
    2.0910192806e-01     0.0000000000e+00
    1.7386394968e-01     2.9492547862e-01
   -8.6815927349e-02     1.4964967612e-01
   -2.5194231412e-02    -3.2185808541e-03
    1.7579900227e-02     2.6558233968e-02
    1.7534474218e-03     1.0638990069e-02
    4.0667901396e-02     2.0000183653e-02
    8.6084633947e-03     6.7127524525e-02
   -3.5238595149e-02     2.6418622610e-02
```

-2.8089019914e-03	-5.0327778580e-03
5.9126031818e-03	1.1754222575e-02
-4.2042171031e-03	1.9907171322e-03
6.0262280966e-03	-2.8105610392e-03
1.2774685378e-03	3.2950775889e-03
-2.0149280301e-03	-3.5649695367e-03
2.3433072734e-03	-8.9948929312e-04
-2.5017414731e-03	1.6626349321e-03
-8.9336335704e-04	-1.6273582475e-03
1.2572607241e-03	1.9680442601e-03
-1.2706927063e-03	1.0036772669e-03
1.5474422591e-03	-8.3773959530e-04
1.0565620088e-03	1.1718363416e-03
-5.0144285276e-04	-1.0879819857e-03
1.1608260605e-03	-9.3161956145e-04
-6.5846983181e-04	3.6933657355e-04
-7.1619024633e-04	-1.0647916442e-03
3.7565226387e-04	3.8011221686e-04
-8.9110837067e-04	5.4490096771e-04
2.4458038893e-04	-4.0566784863e-04
4.5242631105e-04	7.0418345508e-04
-4.0209517954e-04	-2.0215637416e-04
5.5933526825e-04	-4.3074409359e-04
-1.8430597706e-04	3.4777317216e-04
-4.2784361174e-04	-4.7649080150e-04
2.7226728993e-04	1.6091254375e-04
-4.4391379843e-04	4.1849867012e-04
1.1773502845e-04	-1.8904530922e-04
3.5493200948e-04	4.5512193556e-04
-1.7584517833e-04	-7.8983610593e-05
4.1764843946e-04	-2.7998565652e-04
-2.0486090622e-04	1.1663811991e-04

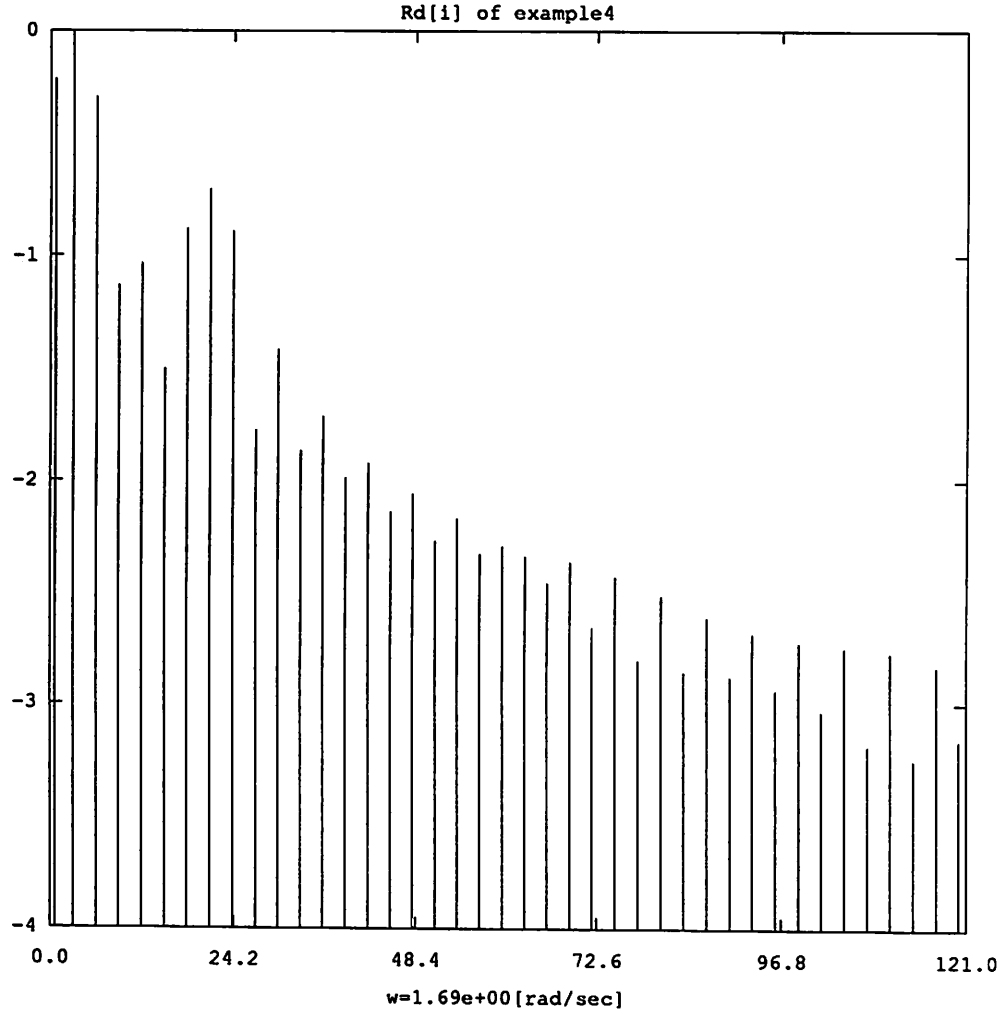


Fig. 3 (a)

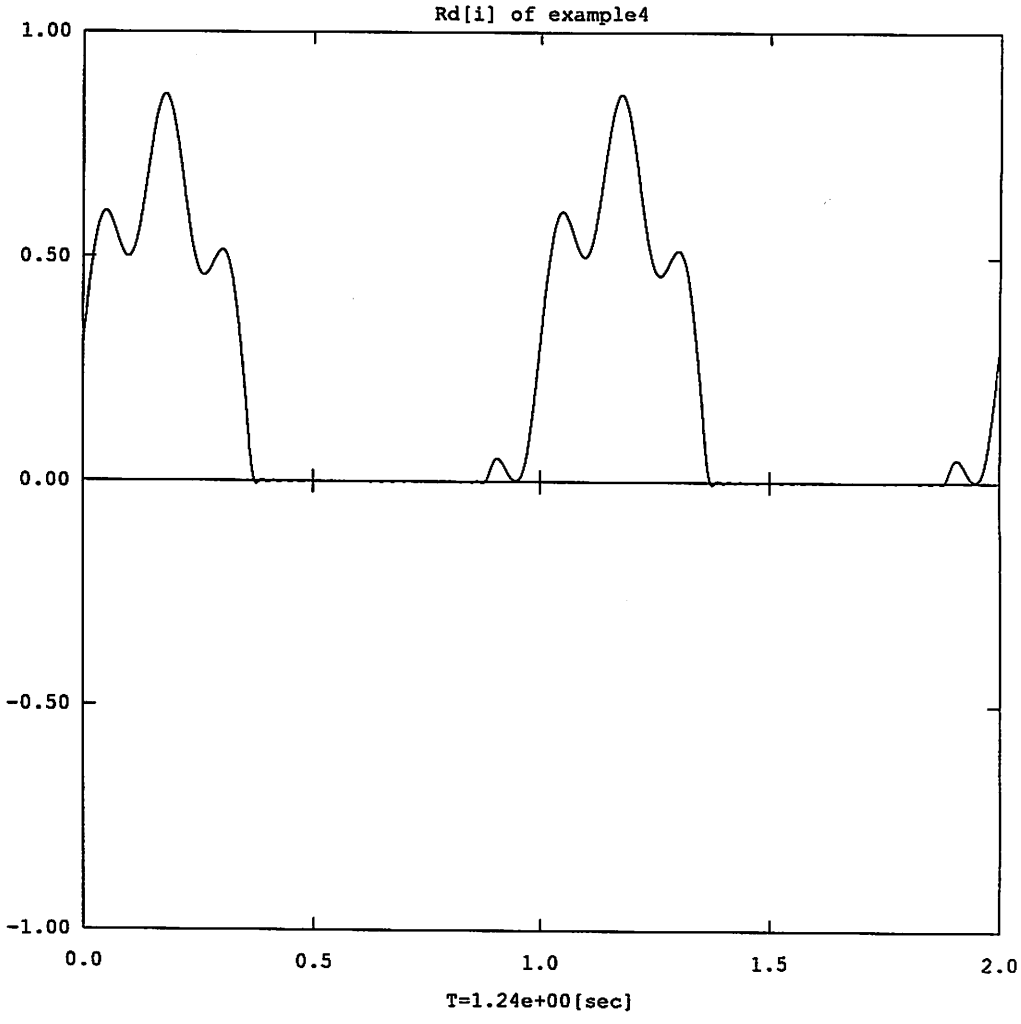


Fig. 3 (b)

Example 5 (Piecewise-linear resistor.)

This example demonstrates the use of piecewise-linear representation for nonlinear resistors. The circuit used in this example is almost the same as the one in the previous example except that the diode in the circuit is modeled by a piecewise-linear characteristic. One usually expects shorter execution time for circuits whose nonlinear resistors are modeled by the piecewise-linear representations.

The convergence ratios for this example and for Example 4 are similar. And since the $v-i$ characteristic for the nonlinear resistor R_d for the two examples are very close, we expect the two results to be close. This is verified by comparing Fig. 4 (a) and (b), the spectrum and the waveform of the current on the diode, with Fig. 3 (a) and (b), respectively.

```
* Example 5:  New diode modeling with compensation resistors.
*
*
V1     10      0      {5*cos(4.44*t)}
V2      1     10      {5*cos(35.5*t)}
R       1      2       1
L       2      3       1
C       3      4       1
Rp      4      0       5
Rc      4      5       -1
*  modified diode (voltage-controlled piecewise-linear form)
Rd      5      0      {v=(0,0)(0.2,0)(0.3,1.528058e-3)
$(0.5,9.76330e-2)(2,1.52887)}
.end
```

```
n = 5; indp = 2; nonl = 1;
matrix P:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
  -1.0000000000e+00   -1.0000000000e+00    1.0000000000e+00    1.0000000000e+00
   0.0000000000e+00
matrix Q:
   1.0000000000e+00    0.0000000000e+00    0.0000000000e+00    1.0000000000e+00
   0.0000000000e+00   -1.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    1.0000000000e+00    0.0000000000e+00
   1.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00   -1.0000000000e+00
  -6.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
  -5.0000000000e+00
vector s:
   0.0000000000e+00    0.0000000000e+00    0.0000000000e+00    0.0000000000e+00
   0.0000000000e+00

>type = 1; var1 = i; var2 = 0; name = V1; node1 = 10; node2 = 0;
relat = 5.00e+00*cos(4.440e+00*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   4.4400000000e+00;
>type = 1; var1 = i; var2 = 0; name = V2; node1 = 1; node2 = 10;
relat = 5.00e+00*cos(3.550e+01*t)
phasor = (  5.0000000000e+00,  0.0000000000e+00)  omega =   3.5500000000e+01;
>type = 4; var1 = v; var2 = 0; name = C; node1 = 3; node2 = 4;
value =   1.0000000000e+00;
>type = 3; var1 = i; var2 = 0; name = L; node1 = 2; node2 = 3;
value =   1.0000000000e+00;
>type = 9; var1 = v; var2 = i; name = Rd; node1 = 5; node2 = 0;
pln = 5
  0.0000000000e+00   0.0000000000e+00     2.0000000000e-01   0.0000000000e+00
  3.0000000000e-01   1.5280580000e-03     5.0000000000e-01   9.7633000000e-02
  2.0000000000e+00   1.5288700000e+00

<omega =   1.6904761905e+00; M = 128; nindx = 41;
vector indx:
    0     3     6     9    12    15    18    21    24    27
   30    33    36    39    42    45    48    51    54    57
   60    63    66    69    72    75    78    81    84    87
   90    93    96    99   102   105   108   111   114   117
  120
```

```
    1.6904761905e+00
2
3 21
41
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57
60 63 66 69 72 75 78 81 84 87 90 93 96 99 102 105 108 111
114 117 120
Rd
v
    2.9338524785e-04
   -8.4708888042e-01     0.0000000000e+00
    1.6406815449e+00     1.9539464116e+00
   -2.0261573774e-02    -5.9631629404e-01
    9.1333583925e-02    -2.0600546549e-02
   -9.4243543105e-02    -7.7451939485e-02
   -1.2959523128e-02    -3.7369993909e-02
   -1.7092475626e-01    -4.3763184945e-02
    3.8309022280e-02     4.2921422392e-01
    1.2289989914e-01    -1.2438638069e-01
    1.2817005327e-02     1.7335598778e-02
   -2.8946813442e-02    -4.4507679921e-02
    1.5396308864e-02    -9.4441465585e-03
   -2.3005433423e-02     1.3416911074e-02
   -5.7194932053e-03    -1.2279250653e-02
    9.4612174598e-03     1.3636304511e-02
   -8.3619458450e-03     3.8379791773e-03
    9.9573266281e-03    -7.6267377880e-03
    4.0278478230e-03     5.5207122401e-03
   -5.3493245965e-03    -8.0489629292e-03
    4.5429775050e-03    -4.5804467944e-03
   -5.9709427568e-03     3.2157779010e-03
   -4.2557139013e-03    -4.4689525311e-03
    2.1539925595e-03     3.8346661229e-03
   -4.1009702955e-03     3.2836610591e-03
    2.3930285793e-03    -2.0136949989e-03
    2.4729217526e-03     3.1766363222e-03
   -2.0156311935e-03    -1.9154551515e-03
    2.2832483676e-03    -2.3302612069e-03
   -1.7878145386e-03     1.5649580707e-03
   -2.4396223588e-03    -1.9422141201e-03
    9.0052028001e-04     1.4173941835e-03
   -2.0019172754e-03     2.2466822555e-03
    8.1086915701e-04    -5.6131678540e-04
    1.7264328175e-03     1.9152405578e-03
   -6.1326371356e-04    -4.2904204203e-04
    1.5299405457e-03    -1.2953003445e-03
   -4.6816118201e-04     6.7335087106e-04
   -1.1768533748e-03    -1.1787184356e-03
    5.1706466844e-04     6.6663279674e-04
   -1.0445751962e-03     1.1551440652e-03
    1.1552263836e-03    -2.2360287371e-04

Rd
i
    2.9338524785e-04
    2.1184835066e-01     0.0000000000e+00
    1.7581424449e-01     2.9814265488e-01
   -8.7939347563e-02     1.5000138558e-01
   -2.5071029214e-02    -4.3280650453e-03
    1.7367463438e-02     2.6438101871e-02
    8.4121788246e-04     1.0088908763e-02
    4.0263407767e-02     1.9644053737e-02
    8.0051209404e-03     6.7207110639e-02
   -3.5528651967e-02     2.6227949863e-02
```

-2.6210509047e-03	-4.8466827171e-03
5.8580536999e-03	1.1974204472e-02
-4.1317183824e-03	1.9074860770e-03
6.0717062257e-03	-2.7844364221e-03
1.1075746133e-03	3.1854046582e-03
-2.0971852524e-03	-3.6573723898e-03
2.1654436812e-03	-8.2934451775e-04
-2.6364798093e-03	1.6957240882e-03
-9.0866139013e-04	-1.4723130768e-03
1.2094597453e-03	2.0970111096e-03
-1.2101351359e-03	1.0725262953e-03
1.5514332401e-03	-7.2231198828e-04
1.0211165853e-03	1.1878577580e-03
-4.7960034801e-04	-9.7002182808e-04
1.0722067255e-03	-7.5226719540e-04
-6.2769633485e-04	4.9796464741e-04
-5.9730134880e-04	-8.1095256698e-04
4.8023325048e-04	5.0640382596e-04
-5.9425875528e-04	5.7446516527e-04
4.5990234414e-04	-3.5977587284e-04
5.9322876110e-04	5.2279958552e-04
-2.1831337984e-04	-3.4538458419e-04
5.0909164017e-04	-5.3843350589e-04
-2.0977245012e-04	1.3833798964e-04
-4.2260290501e-04	-4.8013674525e-04
1.3857229358e-04	1.1924121644e-04
-4.0099721043e-04	3.1745551373e-04
1.0711734984e-04	-1.6506774460e-04
2.7543634788e-04	2.8703593177e-04
-1.2159809648e-04	-1.8576342963e-04
2.7253224836e-04	-2.8764083562e-04
-2.8773249357e-04	4.5722268397e-05

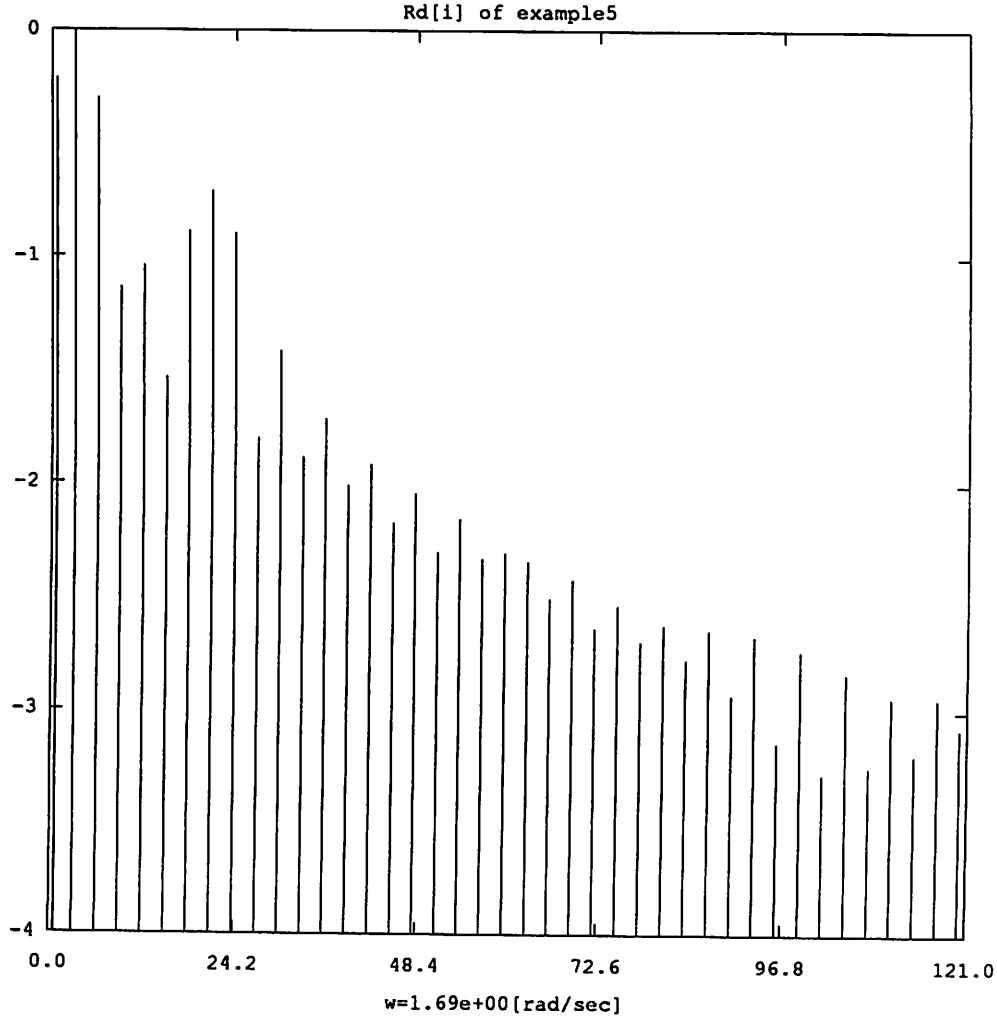


Fig. 4 (a)

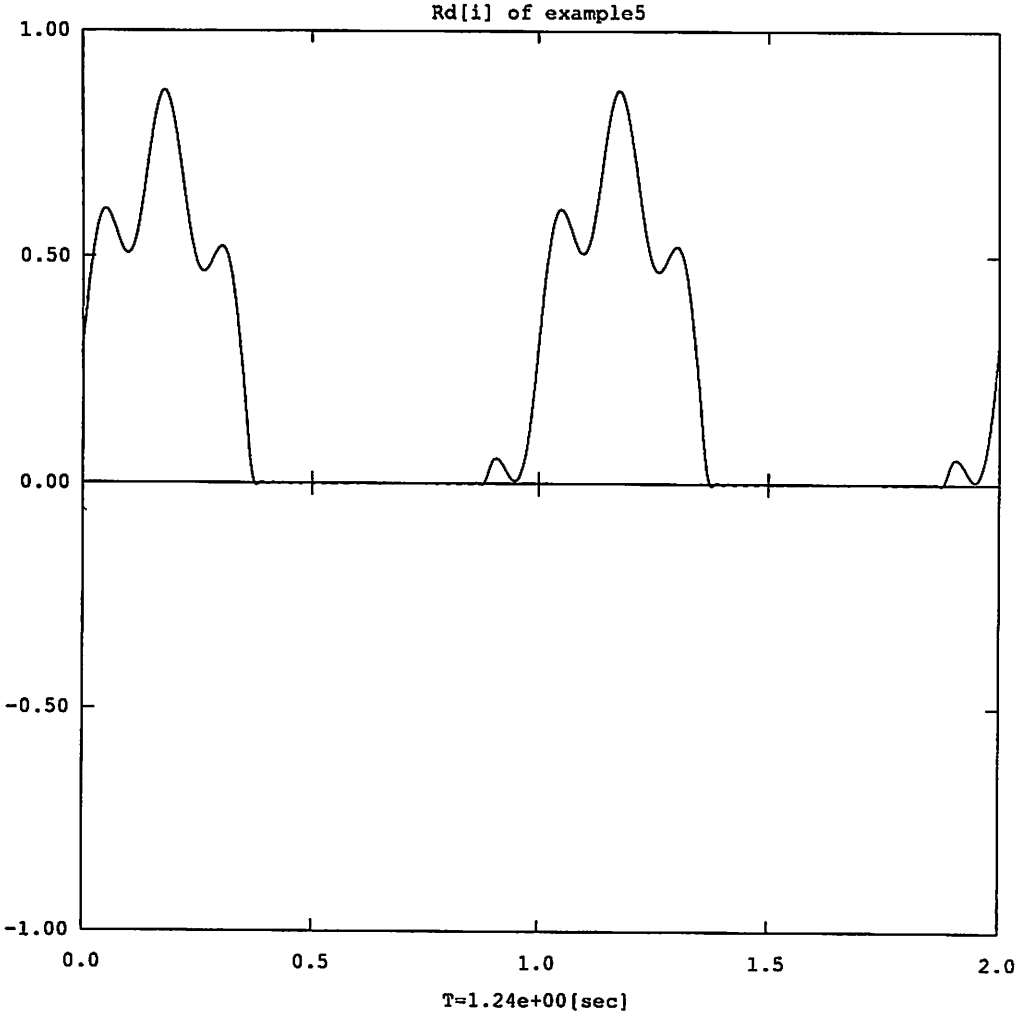
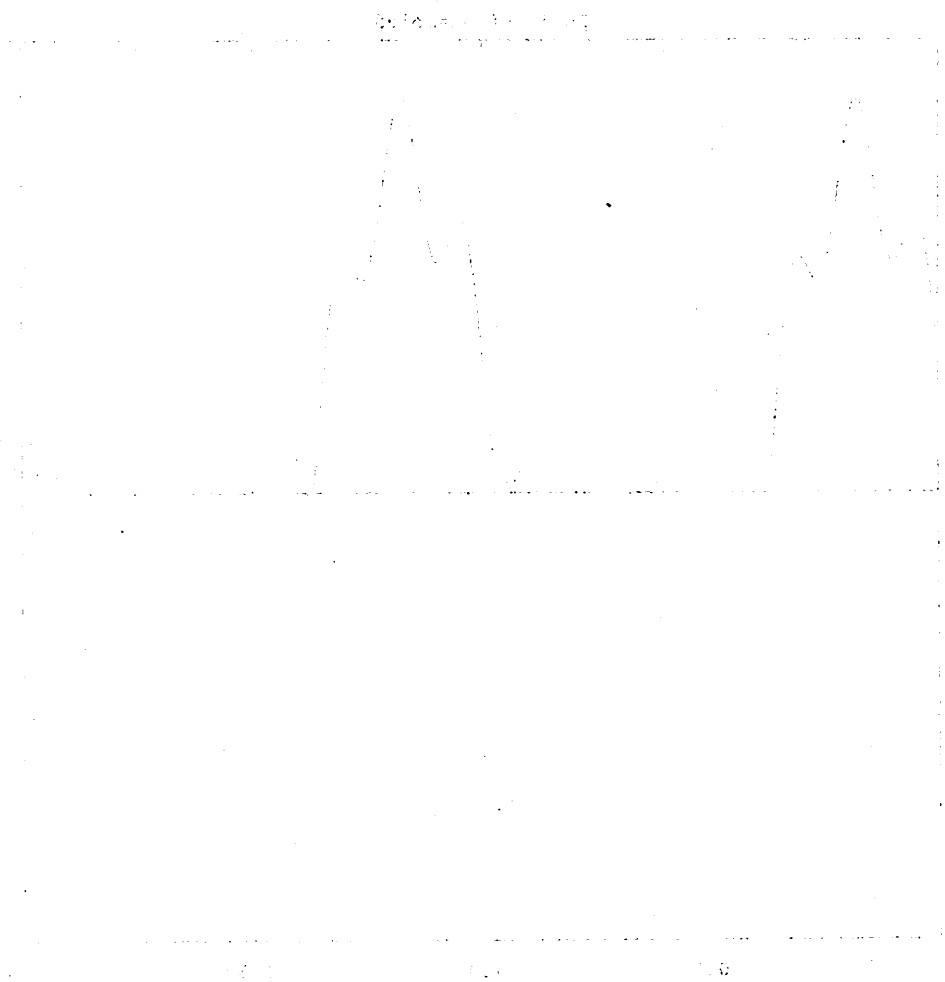


Fig. 4 (b)

Appendix II.

Source code listing.



```
/*********************************************************************/
/* Header for substitution algorithm  <subst.h>            Mar. 1987  */
/*                          for NEC PC-98XA          by Sakamoto Akio  */
/*********************************************************************/


typedef struct {
    int     type;
    double  value;
            COMPLEX source;
}               PORT;

typedef struct {
    char   *var1,
           *var2,
           *p_name;
    int     pnum,
            pln;
            EQN_TREE * relat, *deriv, *dffrn;
    double *plx,
           *ply;
            COMPLEX * src, *res, *dlt, *eps, *gmm, *drv;
            PORT * prtn;
}               NONLNR;

typedef struct {
    int     mltply;
            COMPLEX phasor;
    double  omg;
            PORT * prti;
}               INDPND;

#define  indp_V_S   1          /*  independent voltage source */
#define  indp_C_S   2          /*  independent current source */
#define  linear_L   3          /*  linear inductor */
#define  linear_C   4          /*  linear capacitor */
#define  v_ctrl_R   5          /*  voltage controlled resistor */
#define  c_ctrl_R   6          /*  current controlled resistor */
#define  v_R_impl   7          /*  voltage ctrl'ed resistor def'ed by
                                   implicit eq. */
#define  c_R_impl   8          /*  current ctrl'ed resistor def'ed by
                                   implicit eq. */
#define  v_R_pwln   9          /*  voltage ctrl'ed resistor def. by
                                   piecewise-lin. */
#define  c_R_pwln  10          /*  current ctrl'ed resistor def. by
                                   piecewise-lin. */
#define  f_ctrl_L  11          /*  flux controlled inductor */
#define  c_ctrl_L  12          /*  current controlled inductor */
#define  v_ctrl_C  13          /*  voltage controlled capacitor */
#define  c_ctrl_C  14          /*  charge controlled capacitor */

#define  BRLIMIT  70
#define  MDLIMIT  30
```

```
/* ************************************************************************ */
/*   Library for substitution algorithm                                     */
/*                      for NEC PC-98XA          by Akio SAKAMOTO           */
/* ************************************************************************ */

#include <stdio.h>
#include <ctype.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"


/* ************************************************************************ */
/*   Restrictions for circuit elements:                                     */
/*     independent voltage or current source                      [V, I]    */
/*     2-terminal linear resistor, inductor, capacitor          [R, L, C]   */
/*     linear dependent sorce                                 [E, F, G, H]  */
/*     2-terminal nonlinear resistor of the form;                     [R]   */
/*        a) explicit form such as  {i=f(v)} or {v=f(i)}                    */
/*        b) implicit form such as  {f(v,i)=0;v} or {f(i,v)=0;i}             */
/*        c) piecewise linear form  {v=(v0,i0)(v1,i1) ... (vn,in)}          */
/*                                  or {i=(i0,v0)(i1,v1) ... (in,vn)}       */
/* ************************************************************************ */


/*************************************************************************/
/* Opens the file <name.ext> by the mode 'mode', and returns the file    */
/* pointer.  If it can't open, the message is printed to 'stderr' and exit */
/* the program.                                                          */
/*************************************************************************/
FILE *
open_file(name, ext, mode)
char  *name, *ext, *mode;
{
        char  fname[64];
        FILE  *fp;

        sprintf(fname, "%s.%s", name, ext);
        if ((fp = fopen(fname, mode)) == NULL) {
                printf("\7\7");
                fprintf(stderr, "\nCANNOT OPEN FILE %s\n", fname);
                exit(1);
        }
        return(fp);
}


/*************************************************************************/
/* Inputs the parameter changing line, and chages them, if any.          */
/* Returns  0  if input string is only carrage return.                   */
/*************************************************************************/
int
change_para(val, pname, n, mark)
double  **val;
char    *pname, mark;
int     n;
{
        char    para[100], sub[20], *ptr, *get_substr(), *strcat();
        int     k;
        double  atof();

        fprintf(stderr, " >> Enter as [");
```

```
        for (k = 0; k < n; k++)
                fprintf(stderr, "%c%c??%c", mark, pname[k], (k == n - 1)? ']': ' ');
        fprintf(stderr, " => ");
        fgets(para, 100, stdin);
        if (para[0] == '\n')
                return(0);
        sub[0] = ' ';
        sub[1] = '\0';
        strcat(para, sub);
        ptr = get_substr(para, mark, ' ', sub);
        while (*ptr != '\0') {
                for (k = 0; k < n && sub[0] != pname[k]; k++)
                        ;
                if (k < n)
                        *val[k] = atof(sub + 1);
                else
                        fprintf(stderr, "\7?? Parameter [%c%s] is ignored!\n", mark, sub);
                ptr = get_substr(ptr, mark, ' ', sub);
        }
        return(1);
}


/*************************************************************************/
/* Prints out the spc-file.                                              */
/*************************************************************************/
void
prnt_spc(fp, filename)
FILE  *fp;
char  *filename;
{
        char  *line, *malloc();

        line = malloc(256);
        printf("\n\n***************  SPICE INPUT  ***************");
        printf(" <%s.spc>\n", filename);
        while (fgets(line, 256, fp) != NULL)
                if (line[0] != '\n')
                        printf("%s", line);
        printf("*****************************************************\n\n");
        free(line);
}


/*************************************************************************/
/* Prints out the arrays 'P', 'Q' and 's'.                               */
/*************************************************************************/
void
prnt_PQs(P, Q, s, n, filename)
double  *P, *Q, *s;
int     n;
char    *filename;
{
        void   check_matrix(), check_vector();

        printf("\nGENERALIZED IMPLICT EQUATION OF THE LINEAR RESISTIVE");
        printf(" %d-PORT  <%s.spc>\n", n, filename);
        printf("\t\t P*v + Q*i + s = 0\n\n");
        check_matrix(P, n, "P");
        check_matrix(Q, n, "Q");
        check_vector(s, n, "s");
        printf("\n");
        printf("*************************************************\n\n");
}
```

```
/*************************************************************************/
/* Prints out the vector 'x' of order 'n' with message in 's'.           */
/*************************************************************************/
void
check_vector(x, n, s)
double  *x;
int     n;
char    *s;
{
        int  i;

        printf("**check_vector <<%s>>", s);
        for (i = 0; i < n; i++)
                printf("%c%15.5e", i % 5 == 0 ? '\n': ' ', x[i]);
        printf("\n");
}


/*************************************************************************/
/* Prints out the matrix 'x' of order 'n' with message in 's'.           */
/*************************************************************************/
void
check_matrix(x, n, s)
double  *x;
int     n;
char    *s;
{
        int  i, j;

        printf("**check_matrix <<%s>>", s);
        for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                        printf("%c%15.5e", j % 5 == 0 ? '\n': ' ', x[i * n + j]);
        printf("\n");
}


/*************************************************************************/
/* Prints out the complex vector 'a' of order 'n' with message in 's'.   */
/*************************************************************************/
void
check_cmplx(a, n, s)
COMPLEX  *a;
int      n;
char     *s;
{
        int  i;

        printf("**check_cmplx <<%s>>", s);
        for (i = 0; i < n; i++, a++)
                printf("%c  (%12.4e, %12.4e)  ", i % 2 == 0 ? '\n': ' ', a->r, a->i);
        printf("\n");
}


/*************************************************************************/
/* Gets substring of 's_in' that starts with the char 'startc' and ends  */
/* with the char 'endc'.  The substring is put into 's_out', where two end */
/* characters are not included and any blank is omitted.  Returns the    */
/* pointer of the character 'endc'.                                      */
/*************************************************************************/
char *
```

```
get_substr(s_in, startc, endc, s_out)
char  *s_in, startc, endc, *s_out;
{
        for (; *s_in != startc && *s_in != '\0'; s_in++)
                ;
        if (*s_in != '\0')
                for (s_in++; *s_in != endc && *s_in != '\0'; s_in++)
                        if (! isspace(*s_in))
                                *s_out++ = *s_in;
        *s_out = '\0';
        return(s_in);
}


/*************************************************************************/
/* Return simplified eqation tree of 'eqn'.                              */
/*************************************************************************/
EQN_TREE *
enc_simp(eqn)
char  *eqn;
{
        EQN_TREE  *tree, *sym_enc(), *sym_simp();

        tree = sym_enc(eqn);
        if (sym_stat != OK)
                exit_message("CAN'T ENCODE THE RELATION");
        tree = sym_simp(tree);
        if (sym_stat != OK)
                exit_message("CAN'T SIMPLIFY THE RELATION");
        return(tree);
}


/*************************************************************************/
/* Return simplified derivation of 'relation'.                           */
/*************************************************************************/
EQN_TREE *
deri_simp(relation, var)
EQN_TREE  *relation;
char      *var;
{
        EQN_TREE  *deriv, *sym_deriv(), *sym_simp();

        deriv = sym_deriv(relation, var);
        if (sym_stat != OK)
                exit_message("CAN'T ENCODE THE DERIVATIVE");
        deriv = sym_simp(deriv);
        if (sym_stat != OK)
                exit_message("CAN'T SIMPLIFY THE DERIVATIVE");
        return(deriv);
}


/*************************************************************************/
/* Return string of the name of 'type'.                                  */
/*************************************************************************/
char *
porttype(type)
int  type;
{
        switch (type) {
        case indp_V_S:  return("indp_V_S");
        case indp_C_S:  return("indp_C_S");
        case linear_L:  return("linear_L");
```

```
        case linear_C:  return("linear_C");
        case v_ctrl_R:  return("v_ctrl_R");
        case c_ctrl_R:  return("c_ctrl_R");
        case v_R_impl:  return("v_R_impl");
        case c_R_impl:  return("c_R_impl");
        case v_R_pwln:  return("v_R_pwln");
        case c_R_pwln:  return("c_R_pwln");
        case f_ctrl_L:  return("f_ctrl_L");
        case c_ctrl_L:  return("c_ctrl_L");
        case v_ctrl_C:  return("v_ctrl_C");
        case c_ctrl_C:  return("c_ctrl_C");
        }
        return (NULL);
}
```

```
/* ************************************************************************ */
/*   A substitution algorithm <sub11.c>                    Apr. 1987        */
/*   Libraries for <sub1m.c>                                                */
/*                                                  by Akio SAKAMOTO        */
/* ************************************************************************ */

#include <stdio.h>
#include <ctype.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"


/*************************************************************************/
/* Makes and puts the information for each port.                         */
/*************************************************************************/
void
put_port_info(fp, n, branch, branch_vector, M, K)
FILE            *fp;
int             n, M, K;
struct INLINE   *branch;
struct B_VECTOR *branch_vector;
{
        void    put_source(), put_resistor(), put_L_C();
        int     k, j, indp = 0, nindx, *indx, *mltply, *dum, *m, ind;
        double  *omgs, omin = 1.0e+100, omax = 0.0, omega;

/* Make and put the information of each port */

        callocd(n, &omgs, "omgs");
        for (k = 0; k < n; k++) {
                printf("\n >> port-%d", k + 1);
                j = branch_vector[k].a;
                switch (branch[j].name[0]) {
                case 'V':
                        put_source(fp, branch + j, 'V', &omin, &omax, omgs + indp++);
                        break;
                case 'I':
                        put_source(fp, branch + j, 'I', &omin, &omax, omgs + indp++);
                        break;
                case 'R':
                        put_resistor(fp, branch + j);
                        break;
                case 'L':
                        if (branch[j].relation[0] != '{')
                                put_L_C(fp, branch + j, linear_L, "i");
                        else
                                exit_message("ILLEGAL L-PORT");
                        break;
                case 'C':
                        if (branch[j].relation[0] != '{')
                                put_L_C(fp, branch + j, linear_C, "v");
                        else
                                exit_message("ILLEGAL C-PORT");
                        break;
                default:
                        exit_message("ILLEGAL PORT TYPE");
                }
        }
        printf("\n*********************************************************\n");

/* Make and put the information for array 'indx' */
```

```
        switch (indp) {
        case 0:
                exit_message("\nINDPENDENT SOURCE IS NOTHING");
        case 1:
                omega = omin;
                for (M = 4; M <= K; M *= 2)
                        ;
                printf("\n  >> Independent source is only one,");
                printf("\n\t  so  M  is set to  %d (omega = %12.4e).\n", M, omega);
                nindx = M + 1;
                calloci(nindx, &indx, "indx");
                for (k = 0; k < nindx; k++)
                        indx[k] = k;
                break;
        default:
                omega = omax / (M / (K + 1));
                if (omin / omega < 0.6)
                        omega = omin;
                printf("\n  >> The number of independent sources is %d,", indp);
                printf(" and  omega = %12.4e.", omega);
                calloci(indp, &mltply, "mltply");
                for (k = 0; k < indp; k++) {
                        mltply[k] = omgs[k] / omega + 0.5;
                        printf("\n    ... ang. freq. %12.4e  => %4d * omega  = %12.4e",
                                omgs[k], mltply[k], mltply[k] * omega);
                }
                calloci(M + 1, &dum, "dum");
                for (k = 0; k <= M; k++)
                        dum[k] = 0;
                calloci(indp, &m, "m");
                for (k = 0; k < indp; k++)
                        m[k] = - K;
                do {
                        for (ind = k = 0; k < indp; k++)
                                ind += m[k] * mltply[k];
                        if (- M <= ind && ind <= M)
                                dum[(ind < 0)? -ind: ind] = 1;
                } while (next(m, K, m + indp));
                for (nindx = k = 0; k <= M; k++)
                        nindx += dum[k];
                calloci(nindx, &indx, "indx");
                for (ind = k = 0; k <= M; k++)
                        if (dum[k] == 1)
                                indx[ind++] = k;
                free(mltply);
                free(dum);
                free(m);
                break;
        }
        fprintf(fp, "\n\n<omega = %18.10e; M = %d; nindx = %d;", omega, M, nindx);
        printf("\n\n\t M = %d,  K = %d,  nindx = %d\n\n", M, K, nindx);
        fprintf(fp, "\nvector indx:\n");
        for (k = 0; k < nindx; k++)
                fprintf(fp, "%5d%c", indx[k], k % 10 == 9? '\n': ' ');
        fprintf(fp, "\n");
        free(indx);
        free(omgs);
}

/*************************************************************************/
/* Returns the status of next during calculating the values of 'indx'.   */
/* Status is  0  if it is the end; otherwise status is  1.               */
```

```
/*****************************************************************************/
int
next(m, K, last)
int  *m, K, *last;
{
        if (m == last)
                return(0);
        if (++(*m) <= K)
                return(1);
        *m = - K;
        return(next(m + 1, K, last));
}


/*****************************************************************************/
/* Makes and puts the information of [indp_V_S] or [indp_C_S].               */
/*****************************************************************************/
void
put_source(fp, branch, type, omin, omax, omgs)
FILE           *fp;
struct INLINE  *branch;
char           type;
double         *omin, *omax, *omgs;
{
        char      eqn[256], *get_substr();
        EQN_TREE  *relat, *enc_simp();
        double    omega, amp, get_phasor(), sym_eval();
        COMPLEX   phasor;

/* Puts the first line. */

        switch (type) {
        case 'V':
                fprintf(fp, "\n>type = %d; var1 = i; ", indp_V_S);
                printf(" [indp_V_S] sol. is current");
                break;
        case 'I':
                fprintf(fp, "\n>type = %d; var2 = v; ", indp_C_S);
                printf(" [indp_C_S] sol. is voltage");
                break;
        default:
                exit_message("ILLEGAL SOURCE");
        }
        fprintf(fp, "var2 = 0; name = %s; node1 = %d; node2 = %d;",
                branch->name, branch->node1, branch->node2);
        printf(" (between nodes %2d  and %2d)  name = %s",
                branch->node1, branch->node2, branch->name);

/* Put the relation */

        get_substr(branch->relation, '{', '}', eqn);
        relat = enc_simp(eqn);
        sym_dec(eqn, 256, relat);
        fprintf(fp, "\nrelat = %s", eqn);
        printf("\n\t%c(t) = %s", type == 'V'? 'v': 'i', eqn);
        if (relat->op == MULTIPLY) {
                amp = sym_eval(relat->lp);
                if (sym_stat != OK)
                        exit_message("ILLEGAL SOURCE AMPLITUDE");
                omega = get_phasor(relat->rp, &phasor);
        }
        else {
                amp = 1.0;
                omega = get_phasor(relat, &phasor);
```

```
        }
        if (omega == 0.0)
                exit_message("ILLEGAL SOURCE RELATION");
        phasor.r *= amp;
        phasor.i *= amp;

/* Put phasor and omega */

        fprintf(fp, "\nphasor = (%18.10e,%18.10e)  omega = %18.10e;",
                phasor.r, phasor.i, omega);
        printf("\n\t  phasor = (%13.5e, %13.5e),  omega = %e",
                phasor.r, phasor.i, omega);
        if(*omin > omega)
                *omin = omega;
        if(*omax < omega)
                *omax = omega;
        *omgs = omega;
}


/*************************************************************************/
/* Gets the value of phasor, puting into 'val', and returns the angular  */
/* frequency of the relation describing in 'x'.                          */
/*************************************************************************/
double
get_phasor(x, val)
EQN_TREE  *x;
COMPLEX   *val;
{
        int      operation;
        double   phi, sym_eval(), sin(), cos();
        COMPLEX  *cmplx();

        operation = x->op;
        if (operation != COSINE && operation != SINE)
                return(0.0);
        x = x->lp;
        if (x->op == ADD) {
                if (x->rp->op == REAL) {
                        phi = sym_eval(x->rp);
                        x = x->lp;
                }
                else if (x->lp->op == REAL) {
                        phi = sym_eval(x->lp);
                        x = x->rp;
                }
                else
                        return(0.0);
        }
        else
                phi = 0.0;
        if (operation == COSINE)
                cmplx(val, cos(phi), sin(phi));
        else
                cmplx(val, sin(phi), -cos(phi));
        if (x->op != MULTIPLY)
                return((x->op == VARIABLE)? 1.0: 0.0);
        else if (x->lp->op == REAL && x->rp->op == VARIABLE)
                return(sym_eval(x->lp));
        else if (x->rp->op == REAL && x->lp->op == VARIABLE)
                return(sym_eval(x->rp));
        return(0.0);
}
```

```
/*************************************************************************/
/* Makes and puts the information of nonlinear resistor port.            */
/*************************************************************************/
void
put_resistor(fp, branch)
FILE           *fp;
struct INLINE  *branch;
{
        void      make_plxy();
        char      eqn[256], eqn2[256], *var1, *var2, *br_relation;
        char      *strcpy(), *malloc(), *strcat(), *porttype(), *get_substr();
        EQN_TREE  *relat, *deriv, *dffrn, *enc_simp(), *deri_simp();
        int       type, pln, k, check_pwln();
        double    *plx, *ply;

        strcpy(var1 = malloc(2), "i");
        strcpy(var2 = malloc(2), "v");

/* Find the type of resistor */

        br_relation = get_substr(branch->relation, '{', '=', eqn);
        if (eqn[0] == 'i' && eqn[1] == '\0') {
                get_substr(br_relation, '=', '}', eqn);
                if (pln = check_pwln(eqn)) {
                        type = c_R_pwln;
                }
                else {
                        type = v_ctrl_R;
                        var1 = "v";
                        var2 = "i";
                }
        }
        else if (eqn[0] == 'v' && eqn[1] == '\0') {
                get_substr(br_relation, '=', '}', eqn);
                if (pln = check_pwln(eqn)) {
                        type = v_R_pwln;
                        var1 = "v";
                        var2 = "i";
                }
                else {
                        type = c_ctrl_R;
                }
        }
        else {
                br_relation = get_substr(br_relation, '=', ';', eqn2);
                if (eqn2[0] != '0' || eqn2[1] != '\0')
                        exit_message("ILLEGAL R-PORT");
                get_substr(br_relation, ';', '}', eqn2);
                switch (eqn2[0]) {
                case 'v':
                        type = v_R_impl;
                        var1 = "v";
                        var2 = "i";
                        break;
                case 'i':
                        type = c_R_impl;
                        break;
                default:
                        exit_message("ILLEGAL R-PORT");
                }
        }

/* Put the first line. */
```

```
        fprintf(fp, "\n>type = %d; var1 = %s; var2 = %s;", type, var1, var2);
        fprintf(fp, " name = %s; node1 = %d; node2 = %d;",
                branch->name, branch->node1, branch->node2);
        printf(" [%s] sol. is %s",
                porttype(type), *var1 == 'v'? "voltage": "current");
        printf(" (between nodes %2d  and %2d)  name = %s",
                branch->node1, branch->node2, branch->name);

/* Make and put the relation */

        switch (type) {
        case v_ctrl_R:
        case c_ctrl_R:
                relat = enc_simp(eqn);
                        sym_dec(eqn, 256, relat);
                        fprintf(fp, "\nrelat = %s", eqn);
                        printf("\n\t %c(%c)  = %s", *var2, *var1, eqn);
                deriv = deri_simp(relat, var1);
                        sym_dec(eqn, 256, deriv);
                        fprintf(fp, "\nderiv = %s", eqn);
                        printf("\n\t d%c/d%c = %s", *var2, *var1, eqn);
                break;
        case v_R_impl:
        case c_R_impl:
                relat = enc_simp(eqn);
                        sym_dec(eqn, 256, relat);
                        fprintf(fp, "\nrelat = %s", eqn);
                        printf("\n\tf(%c,%c) = %s", *var1, *var2, eqn);
                dffrn = deri_simp(relat, var2);
                        sym_dec(eqn, 256, dffrn);
                        fprintf(fp, "\ndffrn = %s", eqn);
                        printf("\n\t df/d%c = %s", *var2, eqn);
                sym_dec(eqn2, 256, deri_simp(relat, var1));
                eqn[0] = '\0';
                strcat(strcat(strcat(eqn, "-("), eqn2), ")/(");
                sym_dec(eqn2, 256, dffrn);
                strcat(strcat(eqn, eqn2), ")");
                if (strlen(eqn) > 256)
                        exit_message("DERIVATION IS TOO LONG");
                deriv = enc_simp(eqn);
                        sym_dec(eqn, 256, deriv);
                        fprintf(fp, "\nderiv = %s", eqn);
                        printf("\n\t d%c/d%c = %s", *var2, *var1, eqn);
                break;
        case v_R_pwln:
        case c_R_pwln:
                callocd(pln, &plx, "plx");
                callocd(pln, &ply, "ply");
                make_plxy(pln, plx, ply, eqn);
                fprintf(fp, "\npln = %d\n", pln);
                for (k = 0; k < pln; k++)
                        fprintf(fp, "%18.10e %18.10e  %c",
                                plx[k], ply[k], k % 2 == 1? '\n': ' ');
                printf("\n\t(%c, %c) = ", *var1, *var2);
                for (k = 0; k < pln; k++)
                        printf("(%g, %g)  ", plx[k], ply[k]);
                free(ply);
                free(plx);
                break;
        }
}
```

```
/*************************************************************************/
/* Returns the number of the coordinates of the form "(xxx,xxx)", where  */
/* xxx is a constant of double.  If there is any character between co-   */
/* ordinates, 0  is returned.                                            */
/*************************************************************************/
int
check_pwln(eqn)
char  *eqn;
{
        char  buffc[100], *get_substr();
        int   num = 0, check_double();

        while (*eqn != '\0') {
                eqn = get_substr(eqn, '(', ',', buffc);
                if (check_double(buffc) == 0)
                        return(0);
                eqn = get_substr(eqn, ',', ')', buffc);
                if (check_double(buffc) == 0)
                        return(0);
                num++;
                eqn++;
        }
        return(num);
}


/*************************************************************************/
/* Returns  1  if 'buffc' consists the characters of digits, 'e', '.',   */
/* '-', '+' and 'E'; otherwise returns  0.                               */
/*************************************************************************/
int
check_double(buffc)
char  *buffc;
{

        if (*buffc == '\0')
                return(0);
        while (*buffc != '\0') {
                if (isdigit(*buffc) == 0 && *buffc != 'e' && *buffc != '.'
                        && *buffc != '-' && *buffc != '+' && *buffc != 'E')
                        return(0);
                buffc++;
        }
        return(1);
}


/*************************************************************************/
/* Makes the arrays 'plx' and 'ply' for piecewise linear resistor.       */
/*************************************************************************/
void
make_plxy(pln, plx, ply, eqn)
int     pln;
double  *plx, *ply;
char    *eqn;
{
        char    val[100], *get_substr();
        double  atof();

        while (pln-- > 0) {
                eqn = get_substr(eqn, '(', ',', val);
                *plx++ = atof(val);
                eqn = get_substr(eqn, ',', ')', val);
                *ply++ = atof(val);
```

```
        }
}


/*************************************************************************/
/* Makes and puts the information of linear inductor or capasitor.       */
/*************************************************************************/
void
put_L_C(fp, branch, type, var1)
FILE            *fp;
struct INLINE   *branch;
int             type;
char            *var1;
{
        double  val, stof();

/* Puts the first line. */

        fprintf(fp, "\n>type = %d; var1 = %s; var2 = 0; name = %s;",
                type, var1, branch->name);
        fprintf(fp, " node1 = %d; node2 = %d;", branch->node1, branch->node2);
        printf(" [%s] sol. is %s",
                porttype(type), *var1 == 'v'? "voltage": "current");
        printf(" (between nodes %2d  and %2d)  name = %s",
                branch->node1, branch->node2, branch->name);

/* Put the value */

        fprintf(fp, "\nvalue = %18.10e;", val = stof(branch->relation));
        printf("\n\tvalue = %13.5e [%c]", val, type == linear_L? 'H': 'F');
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm  "sub1m"     Version 2.0      Apr. 1987       */
/*   Preprocessor for  <sub2>                                               */
/*                                              by Akio SAKAMOTO           */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

/* ********************************************************************** */
/*   Restrictions for circuit elements:                                     */
/*     independent voltage or current source                        [V, I] */
/*     2-terminal linear resistor, inductor, capacitor           [R, L, C] */
/*     linear dependent sorce                                 [E, F, G, H] */
/*     2-terminal nonlinear resistor of the form;                      [R] */
/*        a) explicit form such as  {i=f(v)} or {v=f(i)}                   */
/*        b) implicit form such as  {f(v,i)=0;v} or {f(i,v)=0;i}            */
/*        c) piecewise linear form  {v=(v0,i0)(v1,i1) ... (vn,in)}          */
/*                                  or {i=(i0,v0)(i1,v1) ... (in,vn)}       */
/* ********************************************************************** */


main(argc, argv)
int  argc;
char *argv[];
{
        void            prnt_PQs(), put_port_info(), prnt_spc();
        char            mark = '@', *model[MDLIMIT], fname[50], pname[2];
        struct INLINE   branch[BRLIMIT];
        struct B_VECTOR branch_vector[BRLIMIT];
        double          *P, *Q, *s, *ptrd[2], double_m, double_k;
        int             n, mn, k, nonl, indp;
        FILE            *fp, *open_file();

/* Command line */

        if (argc != 2) {
                fprintf(stderr, "\7Usage: sub1 file_name\n");
                exit(0);
        }
        sprintf(fname, "%s", *(++argv));

/* Makes linear circuit using "n_port" */

        fp = open_file(fname, "spc", "r");
        prnt_spc(fp, fname);
        fclose(fp);
        fp = open_file(fname, "spc", "r");
        n = BRLIMIT;
        mn = MDLIMIT;
        n_port(fp, NULL, model, branch, branch_vector, &P, &Q, &s, &n, &mn);
        fclose(fp);

/* Puts 'n', 'indp', 'nonl', 'P', 'Q' and 's' to file "fname.sub" [<-fp]. */

        prnt_PQs(P, Q, s, n, fname);
        fp = open_file(fname, "sub", "w");
        for (nonl = indp = k = 0; k < n; k++)
                switch(branch[branch_vector[k].a].name[0]) {
                case 'V':
```

```
                case 'I':
                        indp++;
                        break;
                case 'R':
                        nonl++;
                        break;
                }
        fprintf(fp, "n = %d; indp = %d; nonl = %d;", n, indp, nonl);
        fprintf(fp, "\nmatrix P:\n");
        for (k = 0; k < n * n; k++)
                fprintf(fp, "%19.10e%c", P[k], k % 4 == 3? '\n': ' ');
        fprintf(fp, "\nmatrix Q:\n");
        for (k = 0; k < n * n; k++)
                fprintf(fp, "%19.10e%c", Q[k], k % 4 == 3? '\n': ' ');
        fprintf(fp, "\nvector s:\n");
        for (k = 0; k < n; k++)
                fprintf(fp, "%19.10e%c", s[k], k % 4 == 3? '\n': ' ');
        fprintf(fp, "\n");

/* Gets parameters */

        double_m = 128.0;    ptrd[0] = &double_m;    pname[0] = 'm';
        double_k = 7.0;      ptrd[1] = &double_k;    pname[1] = 'k';
        fprintf(stderr, " == parameters for the number of harmonics ==\n");
        do {
                fprintf(stderr, "\torder of FFT calculation (m) = %d\n",
                        (int) double_m);
                fprintf(stderr, "\trequired harmonics       (k) = %d\n",
                        (int) double_k);
        }  while (change_para(ptrd, pname, 2, mark));

/* Makes and puts the information of ports into file [<-fp]. */

        printf("\n**************  PORTS INFORMATION  ***************");
        printf(" <%s.sub>", fname);
        put_port_info(fp, n, branch, branch_vector,
                (int) double_m, (int) double_k);
        fclose(fp);
        return (1);
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm <sub2lckt>                    Apr. 1987      */
/*   Libraries for <sub2m>                                                 */
/*                                                  by Akio SAKAMOTO       */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

extern int     *indx, nindx, M;
extern NONLNR  *nonlnr, *nptr, *nlast;
extern INDPND  *indpnd, *iptr, *ilast;


/*************************************************************************/
/* Computes the linear circuit response for each port.                   */
/*************************************************************************/
void
ln_ckt_analysis(port, P, Q, s, n, omega)
PORT    *port;
double  *P, *Q, *s, omega;
int     n;
{
        void     callocC(), solve_ln();
        int      k, *ipvt;
        COMPLEX  *A, *b, *czero(), *ccopy(), *cmplxr_c(), *zero_c();

        callocC(n * n, &A, "A");
        callocC(n, &b, "b");
        calloci(n, &ipvt, "ipvt");

        for (k = 0; k < nindx; k++) {
                for (nptr = nonlnr; nptr < nlast; nptr++)
                        ccopy(&nptr->prtn->source, nptr->src + k);
                if (k == 0) {
                        cmplxr_c(b, s, n);
                        for (iptr = indpnd; iptr < ilast; iptr++)
                                czero(&iptr->prti->source);
                }
                else {
                        zero_c(b, n);
                        for (iptr = indpnd; iptr < ilast; iptr++)
                                if (iptr->mltply == indx[k])
                                        ccopy(&iptr->prti->source, &iptr->phasor);
                                else
                                        czero(&iptr->prti->source);
                }
                solve_ln(port, P, Q, n, indx[k]*omega, A, b, ipvt);
                for (nptr = nonlnr; nptr < nlast; nptr++)
                        ccopy(nptr->eps + k, b + nptr->pnum);
        }

        free(ipvt);
        free(b);
        free(A);
}


/*************************************************************************/
/* Solves linear circuit by using 'cgefa' and 'cgesl'.                   */
```

```
/*****************************************************************************/
void
solve_ln(port, P, Q, n, omega, A, b, ipvt)
PORT     port[];
double   *P, *Q, omega;
int      n, *ipvt;
COMPLEX  *A, *b;
{
        int      i, j, info;
        COMPLEX  dummy, *ccopy(), *csub(), *cmult(), *cmplx(), *cmplxr();

        for (j = 0; j < n; j++)
                switch (port[j].type) {
                case linear_L:
                        for (i = 0; i < n; i++)
                                cmplx(A + i*n+j, Q[i*n+j],
                                        omega*(port[j].value)*P[i*n+j]);
                        break;
                case linear_C:
                        for (i = 0; i < n; i++)
                                cmplx(A + i*n+j, P[i*n+j],
                                        omega*(port[j].value)*Q[i*n+j]);
                        break;
                case indp_V_S:  case v_ctrl_R:  case v_R_impl:  case v_R_pwln:
                        for (i = 0; i < n; i++) {
                                cmplxr(A + i*n+j, Q[i*n+j]);
                                csub(b+i, b+i, cmult(&dummy, cmplxr(&dummy, P[i*n+j]),
                                        &port[j].source));
                        }
                        break;
                case indp_C_S:  case c_ctrl_R:  case c_R_impl:  case c_R_pwln:
                        for (i = 0; i < n; i++) {
                                cmplxr(A + i*n+j, P[i*n+j]);
                                csub(b+i, b+i, cmult(&dummy, cmplxr(&dummy, Q[i*n+j]),
                                        &port[j].source));
                        }
                        break;
                }

        cgefa(A, n, ipvt, &info);
        if (info != n)
                printf("ABNORMAL VALUE = %d  FROM 'cgefa' in solve_ln\n", info);
        else
                cgesl(A, n, ipvt, b, 0);
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm  <sub2m>       Version 3.0     Apr. 1987      */
/*         Stored only essential values for each nonlinear element          */
/*                                                 by Akio SAKAMOTO         */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

/* ********************************************************************** */
/*   Restrictions for circuit elements:                                   */
/*     independent voltage or current source                      [V, I] */
/*     2-terminal linear resistor, inductor, capacitor         [R, L, C] */
/*     linear dependent sorce                                [E, F, G, H] */
/*     2-terminal nonlinear resistor of the form;                    [R] */
/*        a) explicit form such as  {i=f(v)} or {v=f(i)}                  */
/*        b) implicit form such as  {f(v,i)=0;v} or {f(i,v)=0;i}           */
/*        c) piecewise linear form  {v=(v0,i0)(v1,i1) ... (vn,in)}         */
/*                                  or {i=(i0,v0)(i1,v1) ... (in,vn)}      */
/* ********************************************************************** */

int     *indx, nindx, M;
NONLNR  *nonlnr, *nptr, *nlast;
INDPND  *indpnd, *iptr, *ilast;

main(argc, argv)
int   argc;
char  *argv[];
{
void  check_cmplx();
        void     get_data(), get_parameter();
        void     ln_ckt_analysis(), nonl_ckt_response(), relaxation();
        char     fname[50];
        int      n, k, ite;
        double   *P, *Q, *s, omega;
        double   p = 0.3, q = 1.0, step = 10.0;
        double   upper, lower, norm_eps();
        FILE     *fp, *open_file();
        PORT     *port;

/* Command line */

        if (argc != 2) {
                fprintf(stderr, "\7Usage: sub2 file_name");
                exit(1);
        }
        sprintf(fname, "%s", *(++argv));

/* Gets data from sub-file [<-fp], and prepare the structures 'indpnd'     */
/* and 'nonlnr'.                                                           */

        fp = open_file(fname, "sub", "r");
        get_data(fp, &P, &Q, &s, &n, &port, &omega);
        fclose(fp);
        for (k = 0; k < n; k++)
                s[k] = - s[k];

/* Puts data to rsl-file [<-fp]. */

        fp = open_file(fname, "rsl", "w");
```

```
        fprintf(fp, "%20.10e\n", omega);
        fprintf(fp, "%d\n", ilast - indpnd);
        for (iptr = indpnd; iptr < ilast; iptr++)
                fprintf(fp, "%d ", iptr->mltply);
        fprintf(fp, "\n%d\n", nindx);
        for (k = 0; k < nindx; k++)
                fprintf(fp, "%d ", indx[k]);
        fprintf(fp, "\n");

/* Substitution algorithm */

        printf("\n************  SUBSTITUTION ALGORITHM  ************");
        printf(" <%s>\n  M = %d,  nindx = %d\n", fname, M, nindx);
        printf("  # of ports = %d (# of nonlinear = %d,", n, nlast - nonlnr);
        printf(" # of indp. source = %d)\n", ilast - indpnd);
        get_parameter(&p, &q, &step);
        printf("\n=== iteration %2d ===\n", ite = 0);
        fprintf(stderr, "\7");
        ln_ckt_analysis(port, P, Q, s, n, omega);
        nonl_ckt_response();
        upper = 10.0 * norm_eps();
        lower = upper / 10000.0;
        relaxation(port, P, Q, n, omega, 0.0, q);

        while ( 1 ) {
                printf("\n=== iteration %2d ===\n", ++ite);
                fprintf(stderr, "\7");

                ln_ckt_analysis(port, P, Q, s, n, omega);
                        /* Analyzes linear circuit for each frequency. */

                nonl_ckt_response();
                        /* Calculates the response of each nonlinear element */

                if (quit(fp, upper, lower, ite, &p, &q, &step))
                        /* Checks the value epsilon and determines to quit */
                        break;

                relaxation(port, P, Q, n, omega, p, q);
                        /* Calculates new value 'src' for each nonlinear element */
        }
        fclose(fp);
        printf("\n\n");
        return (1);
}
```

```
/* ************************************************************************ */
/*   A substitution algorithm <sub2nckt>                    Apr. 1987       */
/*   Libraries for <sub2m>                                                  */
/*                                                  by Akio SAKAMOTO        */
/* ************************************************************************ */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

extern int     *indx, nindx, M;
extern NONLNR  *nonlnr, *nptr, *nlast;
extern INDPND  *indpnd, *iptr, *ilast;

#define  CON_RES  1.0e-4
        /* convergence value for calculating the response of explicit eq. */


/*************************************************************************/
/* Computes the response for each nonlinear element.                     */
/*************************************************************************/
void
nonl_ckt_response()
{
        void      callocC(), do_irfft(), do_rfft();
        char      *var1, *var2, *porttype();
        EQN_TREE  *relat, *deriv, *dffrn;
        int       k, count, j, jlast;
        double    *bufd_src, *bufd_res, *bufd_drv, err, del, sol;
        double    src, drv, *plx, *ply;
        double    fabs(), sym_eval();
        COMPLEX   *bufferC, *a, *b, *czero();

        callocd(2 * M, &bufd_src, "bufd_src");
        callocd(2 * M, &bufd_res, "bufd_res");
        callocd(2 * M, &bufd_drv, "bufd_drv");
        callocC(M + 1, &bufferC, "bufferC");

        for (nptr = nonlnr; nptr < nlast; nptr++) {
                var1 = nptr->var1;
                relat = nptr->relat;
                deriv = nptr->deriv;
                do_irfft(nptr->src, bufd_src, bufferC);
                switch (nptr->prtn->type) {
                case v_ctrl_R:  case c_ctrl_R:
                        for (k = 0; k < 2 * M; k++) {
                                sym_set(var1, bufd_src[k]);
                                bufd_res[k] = sym_eval(relat);
                                bufd_drv[k] = sym_eval(deriv);
                        }
                        break;
                case v_R_impl:  case c_R_impl:
                        var2 = nptr->var2;
                        dffrn = nptr->dffrn;
                        for (sol = 0.0, k = 0; k < 2 * M; k++) {
                                sym_set(var1, bufd_src[k]);
                                for (count = 0; count < 60; count++) {
                                        sym_set(var2, sol);
                                        if (fabs(err = sym_eval(relat)) < CON_RES)
                                                break;
                                        del =  - err / sym_eval(dffrn);
```

```
#define  LIMIT    0.5
                                        if (del < - LIMIT)
                                                sol += - LIMIT;
                                        else if (del < LIMIT)
                                                sol += del;
                                        else
                                                sol += LIMIT;
                                }
                                bufd_res[k] = sol;
                                bufd_drv[k] = sym_eval(deriv);
                        }
                        break;
                case v_R_pwln:  case c_R_pwln:
                        plx = nptr->plx;
                        ply = nptr->ply;
                        jlast = nptr->pln - 1;
                        for (k = 0; k < 2 * M; k++) {
                                src = bufd_src[k];
                                for (j = 1; src > plx[j] && j < jlast; j++)
                                        ;
                                drv = bufd_drv[k]
                                    = (ply[j] - ply[j-1]) / (plx[j] - plx[j-1]);
                                bufd_res[k] = drv * (src - plx[j]) + ply[j];
                        }
                        break;
                }
                do_rfft(bufd_res, nptr->res, bufferC);
                do_rfft(bufd_drv, nptr->drv, bufferC);
                for (a = nptr->eps, b = nptr->res, k = 0; k < nindx; k++, a++, b++) {
                        a->r -= b->r;
                        a->i -= b->i;
                }
                nptr->prtn->value = nptr->drv->r;
                czero(nptr->drv);
        }

        free(bufferC);
        free(bufd_drv);
        free(bufd_res);
        free(bufd_src);
}


/*************************************************************************/
/* Applies inverse FFT to 'from_C' of type COMPLEX, and results are put  */
/* into 'to_d' of type double.                                           */
/*************************************************************************/
void
do_irfft(from_C, to_d, buf_C)
COMPLEX  *from_C, *buf_C;     /* from_C[nindx], buf_C[M + 1] */
double   *to_d;               /* to_d[2 * M]                 */
{
        int      k;
        double   *irfft();
        COMPLEX  *ccopy(), *zero_c();

        zero_c(buf_C, M + 1);
        for (k = 0; k < nindx; k++, from_C++)
                ccopy(buf_C + indx[k], from_C);
        irfft(2 * M, buf_C, to_d);
}


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

```
/* Applies FFT to 'from_d' of type double, and results are put into 'to_C' */
/* of type COMPLEX.                                                        */
/*************************************************************************/
void
do_rfft(from_d, to_C, buf_C)
double   *from_d;          /* from_d[2 * M]              */
COMPLEX  *to_C, *buf_C;    /* to_C[nindx], buf_C[M + 1] */
{
        int      k;
        COMPLEX  *rfft(), *ccopy();

        rfft(2 * M, from_d, buf_C);
        for (k = 0; k < nindx; k++, to_C++)
                ccopy(to_C, buf_C + indx[k]);
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm <sub2out>                    Apr. 1987      */
/*   Libraries for <sub2m>                                                */
/*                                                  by Akio SAKAMOTO      */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

extern int     *indx, nindx, M;
extern NONLNR  *nonlnr, *nptr, *nlast;
extern INDPND  *indpnd, *iptr, *ilast;


/*************************************************************************/
/* Checks the value epsilon and determine to quit.                       */
/* Returns  1  to quit; 0  to continue.                                  */
/*************************************************************************/
int
quit(fp, upper, lower, ite, p, q, step)
FILE    *fp;
double  upper, lower, *p, *q, *step;
int     ite;
{
        void    get_parameter(), fput_s();
        char    instr[10];
        double  eps, norm_eps(), nrm, norm();

        eps = norm_eps();
        if (eps > lower && (ite == 0 || ite % (int) *step != 0) && eps < upper)
                return(0);
        fprintf(stderr, "\7\7\7\n>> Current error = %12.4e  (at iteration %d)\n",
                eps, ite);
        while (1) {
                fprintf(stderr, "  Enter p(put on file) or n(not) => ");
                fgets(instr, 10, stdin);
                if (instr[0] == 'n' || instr[0] == 'N')
                        break;
                if (instr[0] == 'p' || instr[0] == 'P') {
                        for (nptr = nonlnr; nptr < nlast; nptr++) {
                                fput_s(fp, nptr->p_name, nptr->var1,
                                        nrm = norm(nptr->eps, nindx), nptr->src);
                                fput_s(fp, nptr->p_name, nptr->var2, nrm, nptr->res);
                        }
                        break;
                }
        }
        while (1) {
                fprintf(stderr,
                        "  Enter c(continue) or p(parameter) or h(halt) => ");
                fgets(instr, 10, stdin);
                if (instr[0] == 'c' || instr[0] == 'C')
                        return(0);
                if (instr[0] == 'p' || instr[0] == 'P') {
                        get_parameter(p, q, step);
                        return(0);
                }
                if (instr[0] == 'h' || instr[0] == 'H')
                        return(1);
        }
```

```
}


/*****************************************************************************/
/* Gets the new parameter, if any.                                           */
/*****************************************************************************/
void
get_parameter(p, q, step)
double  *p, *q, *step;
{
        char    mark = '@', pname[3];
        double  *ptrd[3];

        pname[0] = 'p';   ptrd[0] = p;
        pname[1] = 'q';   ptrd[1] = q;
        pname[2] = 's';   ptrd[2] = step;
        fprintf(stderr, " == parameters for convergence ==\n");
        do {
                fprintf(stderr, "\tconstant for relaxation alg.  (p) = %4.2f\n", *p);
                fprintf(stderr, "\tconstant for Newton algorithm (q) = %4.2f\n", *q);
                fprintf(stderr, "\tinterruption step for iteration (s) = %d\n",
                        (int) *step);
        }  while (change_para(ptrd, pname, 3, mark));

        printf("\n ** parameters: p = %4.2f, q = %4.2f  and  s = %d **",
                *p, *q, (int) *step);
}


/*****************************************************************************/
/* Returns the maximum value of the norm of 'eps' of nonlinear elements.     */
/*****************************************************************************/
double
norm_eps()
{
        double  norm_m = 0.0, norm_j, norm();

        for (nptr = nonlnr; nptr < nlast; nptr++) {
                norm_j = norm(nptr->eps, nindx);
                printf("\terror(%s) =%9.2e ", nptr->p_name, norm_j);
                if (norm_m < norm_j)
                        norm_m = norm_j;
        }
        return(norm_m);
}


/*****************************************************************************/
/* Returns the norm of 'eps' of nonlinear elements [<-ptr].                  */
/*****************************************************************************/
double
norm(ptr, n)
COMPLEX *ptr;
int     n;
{
        double sum = 0.0, sqrt();

        while (n-- > 0) {
                sum += ptr->r * ptr->r + ptr->i * ptr->i;
                ptr++;
        }
        return(sqrt(sum));
}
```

```
/*************************************************************************/
/* Puts the data of nonliner resistor into file [<-fp].                  */
/*************************************************************************/
void
fput_s(fp, name, var, nrm, val)
FILE     *fp;
char     *name, *var;
double   nrm;
COMPLEX  *val;
{
        int  k;

        fprintf(fp, "%s\n", name);
        fprintf(fp, "%s\n", var);
        fprintf(fp, "%20.10e\n", nrm);
        fprintf(fp, "%20.10e %20.10e\n", val->r, 0.0);
        for (val++, k = 1; k < nindx; k++, val++)
                fprintf(fp, "%20.10e %20.10e\n", 2 * val->r, -2 * val->i);
        fprintf(fp, "\n");
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm <sub2pre>                    Apr. 1987      */
/*   Libraries for <sub2m>                                                */
/*                                                  by Akio SAKAMOTO      */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

extern int     *indx, nindx, M;
extern NONLNR  *nonlnr, *nptr, *nlast;
extern INDPND  *indpnd, *iptr, *ilast;


/*************************************************************************/
/* Memory allocation for COMPLEX                                         */
/*************************************************************************/
void
callocC(n, pt, s)
int      n;
COMPLEX  **pt;
char     *s;
{
        char     *calloc();
        COMPLEX  *p;

        p = (COMPLEX *) calloc(n, sizeof(COMPLEX));
        if (p == NULL) {
                printf("CAN'T ALLOCATE SPACE FOR %s\n", s);
                exit(1);
        }
        else
                *pt = p;
}


/*************************************************************************/
/* Gets the data from sub-file [<-fp].                                   */
/*************************************************************************/
void
get_data(fp, P, Q, s, n, port, omega)
FILE    *fp;
double  **P, **Q, **s, *omega;
int     *n;
PORT    **port;
{
        void      callocC(), set_nonl();
        int       k, j, nn, nonl, indp;
        char      eqn[256], line[30], *ptrc, var1[2], var2[2], p_name[10];
        char      *malloc(), *calloc(), *get_substr();
        double    atof();
        EQN_TREE  *enc_simp();
        COMPLEX   *zero_c();
        PORT      *pptr;

/* Gets the data of 'n', 'indp', 'nonl', 'P', 'Q' and 's'. */

        ptrc = get_substr(fgets(eqn, 256, fp), '=', ';', line);
        *n = atoi(line);
        ptrc = get_substr(ptrc, '=', ';', line);
```

```
        indp = atoi(line);
        get_substr(ptrc, '=', ';', line);
        nonl = atoi(line);
        nn = *n * *n;
        callocd(nn, P, "P");
        callocd(nn, Q, "Q");
        callocd(*n, s, "s");
        *port = (PORT *) calloc(*n, sizeof(PORT));
        indpnd = (INDPND *) calloc(indp, sizeof(INDPND));
        ilast = indpnd + indp;
        nonlnr = (NONLNR *) calloc(nonl, sizeof(NONLNR));
        nlast = nonlnr + nonl;
        fscanf(fp, "\nmatrix P:");
        for (k = 0; k < nn; k++)
                fscanf(fp, "%lf", *P + k);
        fscanf(fp, "\nmatrix Q:");
        for (k = 0; k < nn; k++)
                fscanf(fp, "%lf", *Q + k);
        fscanf(fp, "\nvector s:");
        for (k = 0; k < *n; k++)
                fscanf(fp, "%lf", *s + k);

/* Gets the data for each port. */

        for (indp = nonl = k = 0; k < *n; k++) {
                do {
                        ptrc = fgets(eqn, 256, fp);
                } while (ptrc[0] != '>');
                pptr = *port + k;
                ptrc = get_substr(ptrc, '=', ';', line);
                pptr->type = atoi(line);
                ptrc = get_substr(ptrc, '=', ';', var1);
                ptrc = get_substr(ptrc, '=', ';', var2);
                get_substr(ptrc, '=', ';', p_name);
                switch (pptr->type) {
                case indp_V_S:  case indp_C_S:
                        iptr = indpnd + indp;
                        fgets(eqn, 256, fp);  /* scans the relation */
                        ptrc = fgets(eqn, 256, fp);
                        ptrc = get_substr(ptrc, '(', ',', line);
                        iptr->phasor.r = atof(line) / 2.0;
                        ptrc = get_substr(ptrc, ',', ')', line);
                        iptr->phasor.i = atof(line) / 2.0;
                        get_substr(ptrc, '=', ';', line);
                        iptr->omg = atof(line);
                        iptr->prti = pptr;
                        indp++;
                        break;
                case linear_L:  case linear_C:
                        fscanf(fp, "\nvalue = %lf", &pptr->value);
                        break;
                case v_ctrl_R:  case c_ctrl_R:
                        nptr = nonlnr + nonl;
                        get_substr(fgets(eqn, 256, fp), '=', '\0', eqn);
                        nptr->relat = enc_simp(eqn);
                        get_substr(fgets(eqn, 256, fp), '=', '\0', eqn);
                        nptr->deriv = enc_simp(eqn);
                        set_nonl(nptr, k, var1, var2, p_name, pptr);
                        nonl++;
                        break;
                case v_R_impl:  case c_R_impl:
                        nptr = nonlnr + nonl;
                        get_substr(fgets(eqn, 256, fp), '=', '\0', eqn);
                        nptr->relat = enc_simp(eqn);
```

```
                        get_substr(fgets(eqn, 256, fp), '=', '\0', eqn);
                        nptr->dffrn = enc_simp(eqn);
                        get_substr(fgets(eqn, 256, fp), '=', '\0', eqn);
                        nptr->deriv = enc_simp(eqn);
                        set_nonl(nptr, k, var1, var2, p_name, pptr);
                        nonl++;
                        break;
                case v_R_pwln:  case c_R_pwln:
                        nptr = nonlnr + nonl;
                        fscanf(fp, "pln = %d", &nptr->pln);
                        nptr->plx = (double *) calloc(nptr->pln, sizeof(double));
                        nptr->ply = (double *) calloc(nptr->pln, sizeof(double));
                        for (j = 0; j < nptr->pln; j++)
                                fscanf(fp, "%lf%lf", &nptr->plx[j], &nptr->ply[j]);
                        set_nonl(nptr, k, var1, var2, p_name, pptr);
                        nonl++;
                        break;
                }
        }

/* Gets the data of 'omega', 'M' and 'nindx', and makes the array 'indx'. */

        do {
                ptrc = fgets(eqn, 256, fp);
        } while (ptrc[0] != '<');
        ptrc = get_substr(ptrc, '=', ';', line);
        *omega = atof(line);
        ptrc = get_substr(ptrc, '=', ';', line);
        M = atoi(line);
        get_substr(ptrc, '=', ';', line);
        nindx = atoi(line);
        calloci(nindx, &indx, "indx");
        fscanf(fp, "vector indx:");
        for (k = 0; k < nindx; k++)
                fscanf(fp, "%d", indx + k);

/* Makes the arrays for nonlinear resistors.  */

        for (nptr = nonlnr; nptr < nlast; nptr++) {
                callocC(nindx, &nptr->src, "nptr->src");
                zero_c(nptr->src, nindx);
                callocC(nindx, &nptr->res, "nptr->res");
                callocC(nindx, &nptr->eps, "nptr->eps");
                callocC(nindx, &nptr->dlt, "nptr->dlt");
                callocC(nindx, &nptr->gmm, "nptr->gmm");
                callocC(nindx, &nptr->drv, "nptr->drv");
        }

/* Calculates the value 'mltply' for each independent source. */

        for (iptr = indpnd; iptr < ilast; iptr++)
                iptr->mltply = iptr->omg / *omega + 0.5;
}


/*************************************************************************/
/* Sets the information, 'pnum', 'var1', 'var2', 'p_name' and 'prtn',    */
/* for nonlnear resistor [<-nptr].                                       */
/*************************************************************************/
void
set_nonl(npt, k, var1, var2, p_name, pptr)
NONLNR  *npt;
int     k;
char    *var1, *var2, *p_name;
```

```
PORT    *pptr;
{
        char  *malloc(), *strcpy();

        npt->pnum = k;
        strcpy(npt->var1 = malloc(strlen(var1) + 1), var1);
        strcpy(npt->var2 = malloc(strlen(var2) + 1), var2);
        strcpy(npt->p_name = malloc(strlen(p_name) + 1), p_name);
        npt->prtn = pptr;
}
```

```
/* ********************************************************************** */
/*   A substitution algorithm <sub2rlx>                    Apr. 1987      */
/*   Libraries for <sub2m>                                                */
/*                                                  by Sakamoto Akio      */
/* ********************************************************************** */

#include <stdio.h>
#include "nport.h"
#include "symtype.h"
#include "sym.h"
#include "complex.h"
#include "subst.h"

extern int     *indx, nindx, M;
extern NONLNR  *nonlnr, *nptr, *nlast;
extern INDPND  *indpnd, *iptr, *ilast;

#define  RELAX_ITE  3
        /* iteration times for relaxation algorithm */


/*************************************************************************/
/* Relaxation algorithm                                                  */
/*************************************************************************/
void
relaxation(port, P, Q, n, omega, p, q)
PORT    *port;
double  *P, *Q, omega, p, q;
int     n;
{
        void     callocC(), solve_sn(), make_gmm(), add_src_dlt();
        int      k, m, *ipvt;
        COMPLEX  *A, *b, *ccopy(), *csub();

        callocC(n * n, &A, "A");
        callocC(n, &b, "b");
        calloci(n, &ipvt, "ipvt");

        for (k = 0; k < nindx; k++) {
                for (nptr = nonlnr; nptr < nlast; nptr++)
                        ccopy(&nptr->prtn->source, nptr->eps + k);
                solve_sn(port, P, Q, n, indx[k]*omega, A, b, ipvt);
                for (nptr = nonlnr; nptr < nlast; nptr++)
                        ccopy(nptr->dlt + k, b + nptr->pnum);
        }
        for (m = 0; p > 0.0 && m < RELAX_ITE; m++) {
                make_gmm(p);
                for (k = 0; k < nindx; k++) {
                        for (nptr = nonlnr; nptr < nlast; nptr++)
                                csub(&nptr->prtn->source, nptr->eps + k, nptr->gmm + k);
                        solve_sn(port, P, Q, n, indx[k]*omega, A, b, ipvt);
                        for (nptr = nonlnr; nptr < nlast; nptr++)
                                ccopy(nptr->dlt + k, b + nptr->pnum);
                }
        }
        add_src_dlt(q);

        free(ipvt);
        free(b);
        free(A);
}


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

```
/* Solves the sensitivity circuit by using 'cgefa' and 'cgesl'.            */
/*************************************************************************/
void
solve_sn(port, P, Q, n, omega, A, b, ipvt)
PORT     port[];
double   *P, *Q, omega;
int      n, *ipvt;
COMPLEX  *A, *b;
{
        int      i, j, info;
        COMPLEX  dummy, *ccopy(), *cadd(), *cmult(), *cmplx(), *cmplxr(), *zero_c();

        zero_c(b, n);
        for (j = 0; j < n; j++)
                switch (port[j].type) {
                case linear_L:
                        for (i = 0; i < n; i++)
                                cmplx(A + i*n+j, Q[i*n+j],
                                        omega*(port[j].value)*P[i*n+j]);
                        break;
                case linear_C:
                        for (i = 0; i < n; i++)
                                cmplx(A + i*n+j, P[i*n+j],
                                        omega*(port[j].value)*Q[i*n+j]);
                        break;
                case indp_V_S:
                        for (i = 0; i < n; i++)
                                cmplxr(A + i*n+j, Q[i*n+j]);
                        break;
                case indp_C_S:
                        for (i = 0; i < n; i++)
                                cmplxr(A + i*n+j, P[i*n+j]);
                        break;
                case v_ctrl_R:  case v_R_impl:  case v_R_pwln:
                        for (i = 0; i < n; i++) {
                                cmplxr(A + i*n+j, P[i*n+j] + port[j].value*Q[i*n+j]);
                                cadd(b+i, b+i, cmult(&dummy, cmplxr(&dummy, Q[i*n+j]),
                                        &port[j].source));
                        }
                        break;
                case c_ctrl_R:  case c_R_impl:  case c_R_pwln:
                        for (i = 0; i < n; i++) {
                                cmplxr(A + i*n+j, Q[i*n+j] + port[j].value*P[i*n+j]);
                                cadd(b+i, b+i, cmult(&dummy, cmplxr(&dummy, P[i*n+j]),
                                        &port[j].source));
                        }
                        break;
                }

        cgefa(A, n, ipvt, &info);
        if (info != n)
                printf("ABNORMAL VALUE = %d  FROM 'cgefa' in solve_sn\n", info);
        else
                cgesl(A, n, ipvt, b, 0);
}


/*************************************************************************/
/* Makes 'gmm' for each nonlinear resistor.                              */
/*************************************************************************/
void
make_gmm(p)
double  p;
{
```

```
        void     callocC(), copy_conj();
        int      i, j, k, ind, M2, M3;
        COMPLEX  dummyC, *buff1, *buff2, *zero_c(), *cmult();

        M2 = 2 * M;
        M3 = 3 * M;
        callocC(M2 + 1, &buff1, "buff1 in make_gmm");
        callocC(M2 + 1, &buff2, "buff2 in make_gmm");

        for (nptr = nonlnr; nptr < nlast; nptr++) {
                copy_conj(buff1, nptr->drv);
                copy_conj(buff2, nptr->dlt);
                zero_c(nptr->gmm, nindx);
                for (i = 0; i <= M2; i++) {
                        if ((buff1 + i)->r == 0.0 && (buff1 + i)->i == 0.0)
                                continue;
                        ind = 0;
                        for (j = M2 - i; j <= M2 && j <= M3 - i; j++) {
                                if ((buff2 + j)->r == 0.0 && (buff2 + j)->i == 0.0)
                                        continue;
                                k = i + j - M2;
                                while (indx[ind] < k && ind < nindx)
                                        ind++;
                                if (indx[ind] == k) {
                                        cmult(&dummyC, buff1 + i, buff2 + j);
                                        (nptr->gmm + ind)->r += p * dummyC.r;
                                        (nptr->gmm + ind)->i += p * dummyC.i;
                                        ind++;
                                }
                        }
                }
                nptr->gmm->i = 0.0;
        }
        free(buff1);
        free(buff2);
}


/*************************************************************************/
/* Sets complex numbers such as                                          */
/*         buff[0 .. M-1] <= conjugate of val[M+1 .. 1],                 */
/*         buff[M] <= val[0], and                                        */
/*         buff[M+1 .. 2*M+1] <= val[1 .. M+1].                          */
/*************************************************************************/
void
copy_conj(buff, val)
COMPLEX  *buff, *val;
{
        int      i, ind;
        COMPLEX  *zero_c();

        zero_c(buff, 2 * M + 1);
        buff += M;
        for (i = 1; i < nindx; i++) {
                ind = indx[i];
                (buff + ind)->r = (buff - ind)->r = (val + i)->r;
                (buff + ind)->i = (val + i)->i;
                (buff - ind)->i = - (val + i)->i;
        }
        buff->r = val->r;
        buff->i = 0.0;
}
```

```
/*************************************************************************/
/* Adds the value 'dlt' to 'src' for each nonlinear elements.            */
/* That is,  src <= src + ratio * dlt, where 'ratio' is the value to     */
/* prevent the over modifying the value 'src'.                           */
/*************************************************************************/
void
add_src_dlt(ratio)
double  ratio;
{
        int      k;
        COMPLEX  *a, *b;

        for (nptr = nonlnr; nptr < nlast; nptr++) {
                a = nptr->src;
                b = nptr->dlt;
                for (k = 0; k < nindx; a++, b++, k++) {
                        a->r += ratio * b->r;
                        a->i += ratio * b->i;
                }
        }
}
```

```
#define DISPLAY         /* display graphics on terminal */
/*#define COLOR         /* for color terminals only */
#define PLNS    63
#define BKGD    1
#define BOXS    2
#define WVFM    4
#define SPEC    8
```

```
/*============================================================*/
/*===  Graph Drawing  |                          April 1987   ===*/
/*===     Program     |   Using MASSCOMP Graphic Library     ===*/
/*===       sub3g.c   |         (832 * 600 dots screen)      ===*/
/*============================================================*/

#include <stdio.h>
#include "complex.h"
#include "sub3.h"

#define  QM5 "[wave] Neglect D.C. component ? (y/n)"
#define  QM6 "[wave] Y-axis maximum number ? (#.##)"
#define  QM7 "[wave] X-axis from #.#T ? ( 0, 0.5 )"
#define  QM8 "[wave] X-axis to #.#T ? ( 0.5, 1, 1.5, 2 )"
#define  QM9 "[wave] X-axis step size ? ( 1..6, 10 )"
#define  QMA "[wave] Do you need hard copy ? (y/n)"
#define  QMB "[spectrum] Y-axis minimum number ? (10^#)"
#define  QMC "[spectrum] Do you need hard copy ? (y/n)"

#define  XWS   7                 /* X width of 1 symbol  */
#define  YWS   9                 /* Y width of 1 symbol  */
#define  SSS   6                 /*   tick length of Y   */
#define  TTT   6                 /*   tick length of X   */

#define  FX0  50                 /* signal wave graph    */
#define  FY0 250                 /*    (0,0)=(FX0,FY0)   */
#define  FXL 420                 /*    X-axis length     */
#define  FYL 200                 /*    Y-axis length     */

#define  SX0 530                 /* spectrum graph       */
#define  SY0  50                 /*    (0,0)=(SX0,SY0)   */
#define  SXL 300                 /*    X-axis length     */
#define  SYL 400                 /*    Y-axis length     */

#define  ABS(x) ((x) < 0 ? -(x) : (x))

extern int  x0,
            y0;


/*===== Drawing Graph =====*/

s_graph (omega, coef, nindx, indx, eval, var, ttl)
double  omega;
int     coef,
        nindx,
       *indx;
COMPLEX *eval;
char   *var,
       *ttl;
{
    char    str[64],
            unit[8],
            amount[8];
    double  value,
            peak,
            t0,
            ts,
            tx,
            t,
            wt,
            nwt,
            ymax,
            xm,
```

```
            dx,
            dy,
            twopi;
    double  sin (), cos (), log10 (), sqrt (), pow (), atof (), atan2 ();
    int     k,
            ks,
            x,
            y,
            x1,
            x2,
            y1,
            y2,
            step,
            xfr,
            xto,
            xwd;
    FILE   *fp,
           *open_file ();

    twopi = 8.0 * atan2 (1.0, 1.0);

    switch (var[0]) {
        case 'v':
            strcpy (amount, "Vlt");
            strcpy (unit, "V");
            break;
        case 'i':
            strcpy (amount, "Amp");
            strcpy (unit, "A");
            break;
        case 'q':
            strcpy (amount, "Chg");
            strcpy (unit, "C");
            break;
        case 'p':
            strcpy (amount, "Flx");
            strcpy (unit, "Wb");
            break;
        default:
            strcpy (amount, "???");
            strcpy (unit, "??");
            break;
    }

/* [wave] input value */

    ks = y_or_n (QM5);          /* ? neglect D.C. comp. */
    while (1) {                 /* Y-axis maximum number */
        s_mess (QM6, str, 1);
        ymax = atof (str);
        if (ymax > 0.0)
            break;
        fprintf (stderr, "\7\7");
    }
    while (1) {                 /* X-axis from */
        s_mess (QM7, str, 1);
        xfr = atof (str) / 0.5;
        if (xfr == 0 || xfr == 1)
            break;
        fprintf (stderr, "\7\7");
    }
    while (1) {                 /* X-axis to */
        s_mess (QM8, str, 1);
        xto = atof (str) / 0.5;
```

```
        xwd = xto - xfr;
        if (xto >= 1 && xto <= 4 && xwd > 0)
            break;
        fprintf (stderr, "\7\7");
    }
    while (1) {                 /* X-axis step size */
        s_mess (QM9, str, 1);
        step = atoi (str);
        if (step > 0)
            break;
        fprintf (stderr, "\7\7");
    }
    xm = 0.5;
    if (xwd == 1) {
        xm = 0.1;
        xfr = xfr * 5;
        xto = xto * 5;
        xwd = xwd * 5;
    }
    t0 = twopi / (omega * coef);
    ts = t0 * xwd * xm / FXL;
    tx = xfr * xm * t0;
/* [wave] draw box */
    s_cls (2);
    x0 = FX0;
    y0 = FY0;
    s_line (0, FYL, FXL, -FYL, BOXS, 1);
    s_line (0, 0, FXL, 0, BOXS, 0);
/* [wave] draw Y-axis */
    for (k = 2; k >= -2; k--) {
        y = FYL / 2.0 * k;
        s_line (0, y, SSS, y, BOXS, 0);
        s_line (FXL, y, FXL - SSS, y, BOXS, 0);
        sprintf (str, "%5.2f", ymax / 2.0 * k);
        s_symb (-XWS * 5 - TTT, y + YWS / 2, str, BOXS);
    }
    sprintf (str, "%s[%s]", amount, unit);
    s_symb (-XWS * 5, FYL + YWS * 2 + TTT, str, BOXS);
/* [wave] draw X-axis */
    for (k = xfr; k <= xto; k++) {
        x = FXL / xwd * (k - xfr);
        s_line (x, 0, x, SSS, BOXS, 0);
        s_line (x, -FYL, x, -FYL + SSS, BOXS, 0);
        s_line (x, FYL, x, FYL - SSS, BOXS, 0);
        sprintf (str, "%4.1fT", k * xm);
        s_symb (x - XWS * 3, -FYL - TTT, str, BOXS);
    }
    sprintf (str, "T=%8.2e[sec]", t0);
    s_symb (FXL / 2 - XWS * 8, -FYL - YWS - TTT * 2, str, BOXS);
/* [wave] make "dummy.opt" */
    fp = open_file ("dummy", "opt", "w");
    fprintf (fp, "T %s\nH %s\n", ttl, str);
    if (xwd != 5)
        fprintf (fp, "x %g %g %d %d %d\n", xfr / 2.0, xto / 2.0, xwd, 0, 1);
    else
        fprintf (fp, "x %g %g %d %d %d\n", xfr / 10.0, xto / 10.0, 5, 0, 1);
    fprintf (fp, "y %g %g %d %d %d\n", -ymax, ymax, 4, 0, 2);
    fprintf (fp, "a\nb\nS\nd dummy.dat 1 2\n");
    fclose (fp);
/* [wave] draw wave */
    fp = open_file ("dummy", "dat", "w");
    peak = 0.0;
    for (x = 0; x <= FXL; x += step) {
        t = ts * x + tx;
```

```
        wt = omega * t;
        value = 0.0;
        for (k = ks; k < nindx; k++) {
            nwt = (double) indx[k] * wt;
            value += cos (nwt) * eval[k].r + sin (nwt) * eval[k].i;
        }
        if (peak < ABS (value))
            peak = ABS (value);
        if (x == 0) {
            x1 = x;
            y1 = (int) (value / ymax * (double) FYL + 0.5);
        }
        else {
            x2 = x;
            y2 = (int) (value / ymax * (double) FYL + 0.5);
            s_line (x1, y1, x2, y2, WVFM, 0);
            x1 = x2;
            y1 = y2;
        }
        fprintf (fp, "%g\t%g\n", wt * coef / twopi, value);
    }
    fclose (fp);
    if (ks == 0) {
        sprintf (str, "Peak=%8.2e[%s]", peak, unit);
        s_symb (FXL / 2 - XWS * 9, FYL + YWS + TTT, str, BOXS);
    }
    else {
        sprintf (str, "DC=%8.2e[%s]", eval[0].r, unit);
        s_symb (FXL / 2 - XWS * 8, FYL + YWS + TTT, str, BOXS);
    }
    if (y_or_n (QMA))           /* ? need hard copy */
        s_copy ();
    unlink ("dummy.opt");
    unlink ("dummy.dat");

/* [spectrum] input value */

    while (1) {                 /* [spectrum] Y-axis min. no. */
        s_mess (QMB, str, 1);
        step = atoi (str);
        if (step < 0)
            break;
        fprintf (stderr, "\7\7");
    }
    ts = SXL / (indx[nindx - 1] + 4.0);
    ymax = 0.0;
    for (k = 0; k < nindx; k++) {
        value = sqrt (pow (eval[k].r, 2.0) + pow (eval[k].i, 2.0));
        if (value > ymax)
            ymax = value;
    }
/* [spectrum] draw Y-axis */
    x0 = SX0;
    y0 = SY0;
    s_line (0, 0, SXL, SYL, BOXS, 1);
    for (k = 0; k >= step; k--) {
        y = SYL - SYL * k / step;
        s_line (0, y, SSS, y, BOXS, 0);
        s_line (SXL, y, SXL - SSS, y, BOXS, 0);
        s_symb (-XWS * 4 - TTT, y + YWS / 2, "10", BOXS);
        sprintf (str, "%2d", k);
        s_symb (-XWS * 2 - TTT, y + YWS, str, BOXS);
    }
    sprintf (str, "%s[%s]", amount, unit);
```

```
    s_symb (-XWS * 5, SYL + YWS * 2 + TTT, str, BOXS);
/* [spectrum] draw X-axis */
    y = 10;
    if (indx[nindx - 1] > 40)
        y = 20;
    if (indx[nindx - 1] > 100)
        y = 50;
    if (indx[nindx - 1] > 200)
        y = 100;
    for (k = 0; k <= indx[nindx - 1]; k += y) {
        x = ts * (k + 2);
        s_line (x, 0, x, SSS, BOXS, 0);
        s_line (x, SYL, x, SYL - SSS, BOXS, 0);
        sprintf (str, "%4dw", k);
        s_symb (x - XWS * 3, -TTT, str, BOXS);
    }
    sprintf (str, "Max=%8.2e[%s]", ymax, unit);
    s_symb (SXL / 2 - XWS * 8, SYL + YWS + TTT, str, BOXS);
    sprintf (str, "w=%8.2e[rad/sec]", omega);
    s_symb (SXL / 2 - XWS * 9, -YWS - TTT * 2, str, BOXS);
/* [spectrum] make "dummy.opt" */
    fp = fopen ("dummy.opt", "w");
    fprintf (fp, "T %s\nH %s\n", ttl, str);
    fprintf (fp, "x  0 %d  5 0 1\n", indx[nindx - 1] + 1);
    fprintf (fp, "y %d  0 %d 0 0\n", step, -step);
    fprintf (fp, "a\nb\nS\nd dummy.dat 1 2\n");
    fclose (fp);
/* [spectrum] draw spectrum */
    fp = fopen ("dummy.dat", "w");
    dx = (indx[nindx - 1] + 1) / 200.0;
    dy = -step / 200.0;
    for (k = 0; k < nindx; k++) {
        value = sqrt (pow (eval[k].r, 2.0) + pow (eval[k].i, 2.0));
        if (value != 0) {
            value = log10 (value / ymax);
            if (value < step)
                value = step;
        }
        else
            value = step;
        x1 = (indx[k] + 2) * ts;
        y1 = (step - value) / step * SYL;
        s_line (x1, 0, x1, y1, SPEC, 0);
        if (k == 0)
            bar (fp, dx, value, step, dx, dy);
        else
            bar (fp, (double) indx[k], value, step, dx, dy);
    }
    fclose (fp);
    if (y_or_n (QMC))           /* ? need hard copy */
        s_copy ();
    unlink ("dummy.opt");
    unlink ("dummy.dat");
}


bar (fp, x, val, step, dx, dy)
FILE   *fp;
double  x,
        val,
        dx,
        dy;
int     step;
{
```

```
    fprintf (fp, "%g\t%d\n", x, step);
    if (val == (double) step) {
        fprintf (fp, "%g\t%g\n", x - dx, step + dy);
        fprintf (fp, "%g\t%d\n", x, step);
        fprintf (fp, "%g\t%g\n", x + dx, step + dy);
    }
    else
        fprintf (fp, "%g\t%g\n", x, val);
    fprintf (fp, "%g\t%d\n", x, step);
}
```

```
/*================================================================*/
/*===  Graph Drawing  |                          April 1987   ===*/
/*===     Program     |   Using MASSCOMP Graphic Library     ===*/
/*===       sub3l.c   |         (832 * 600 dots screen)      ===*/
/*================================================================*/

#include <stdio.h>
#include "sub3.h"

extern int  x0,
            y0;


/*===== error =====*/

s_error (mess1, mess2)          /* print error message */
char   *mess1,
       *mess2;                  /* & program exit      */
{
    fprintf (stderr, "\7\7 %s ---> %s \7\7\n", mess1, mess2);
    exit (1);
}


/*===== Message =====*/

s_mess (mess, str, m)           /* print question */
int     m;                      /* & scan strings */
char   *mess,
       *str;
{
    if (m == 1)
        s_cls (0);
    fprintf (stderr, "\t\t\t\t %s >", mess);
    scanf ("%s", str);
}


/*===== Clear Screen =====*/

s_cls (m)                       /* m=0 : clear line    */
int     m;                      /* m=1 : clear text    */
{                               /* m=2 : clear graphic */
    switch (m) {                /* m=3 : m=1 & m=2     */
        case 0:
            fprintf (stderr, "\033[1A\033[2K");
            break;
        case 1:
#ifdef DISPLAY
            mgiclearpln (0, 1, 0);
#endif DISPLAY
            fprintf (stderr, "\033[H");
            break;
        case 2:
#ifdef DISPLAY
#ifdef COLOR
            mgiclearpln (0, 31, 0);
#endif COLOR
#endif DISPLAY
            break;
        case 3:
#ifdef DISPLAY
            mgiclearpln (0, -1, 0);
#endif DISPLAY
```

```
            fprintf (stderr, "\033[H");
            break;
    }
}


/*===== graphics start =====*/

s_start () {
#ifdef DISPLAY
    mgiasngp (0, 0);
#ifdef COLOR
    mgipln (PLNS);
    mgicm (BKGD, mgfcns ("background"));
    mgicm (BOXS, mgfcns ("box"));
    mgicm (WVFM, mgfcns ("waveform"));
    mgicm (SPEC, mgfcns ("spectrum"));
#else COLOR
    mgipln (3);
    mgihue (3);
#endif COLOR
    s_cls (3);
    mgifetchgf (1, "7x9");
#endif DISPLAY
}


/*===== graphics end =====*/

s_end () {
    s_cls (3);
#ifdef DISPLAY
    mgideagp ();
#endif DISPLAY
}


/*===== Draw Line =====*/

s_line (x1, y1, x2, y2, c, m)    /* c    : color */
int     x1,
        y1,
        x2,
        y2,
        c,
        m;                       /* m=0 : line  */
{                                /* m=1 : box   */
#ifdef DISPLAY
#ifdef COLOR
    mgihue (c);
#else COLOR
    mgihue (3);
#endif COLOR
    if (m == 0)
        mgil (x0 + x1, y0 + y1, x0 + x2, y0 + y2);
    else {
#ifdef COLOR
        mgihue (BKGD);
        mgibox (x0 + x1, y0 + y2, x0 + x2, y0 + y1);
        mgihue (c);
#endif COLOR
        mgil (x0 + x1, y0 + y1, x0 + x1, y0 + y2);
        mgil (x0 + x1, y0 + y2, x0 + x2, y0 + y2);
        mgil (x0 + x2, y0 + y2, x0 + x2, y0 + y1);
```

```
        mgil (x0 + x2, y0 + y1, x0 + x1, y0 + y1);
    }
#endif DISPLAY
}


/*===== Draw Symbol =====*/
#define  YWS  9

s_symb (x, y, moji, c)          /* c   : color    */
int     x,
        y,
        c;
char   *moji;
{
#ifdef DISPLAY
#ifdef COLOR
    mgihue (c);
#else COLOR
    mgihue (3);
#endif COLOR
    mgigfs (x0 + x, y0 + y - YWS - 1, 0, moji);
#endif DISPLAY
}


/*===== Hard Copy =====*/

s_copy () {
    system ("plot2d < dummy.opt | fred \"tsptoplot | psplot | lpr -o-h\"");
    fprintf (stderr, "\033[6;1H");
}
```

```
/*=============================================================*/
/*===  Graph Drawing  |                          April 1987   ===*/
/*===     Program     |   Using MASSCOMP Graphic Library     ===*/
/*===       sub3m.c   |         (832 * 600 dots screen)      ===*/
/*=============================================================*/

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

#define  ERR "Can't allocate space"

#define  TI1 "Graph Drawing Program for MASSCOMP"/* program title */
#define  TI2 "data file name :"

#define  QM1 "Do you need this graph ? (y/n)"/* question message */
#define  QM2 "Do you need re-drawing ? (y/n)"

int     x0,
        y0;


main (argc, argv)
int     argc;
char   *argv[];
{
    char    fname[40],
            nam[20],
            var[2],
           *calloc (), ttl[80];
    double  omega,
            error;
    int     k,
            ncoef,
           *coef,
            mcoef,
            nindx,
           *indx;
    COMPLEX *eval;
    FILE   *fp,
           *open_file ();

/* get command line */

    if (argc != 2) {
        fprintf (stderr, "Usage: sub3 file_name\n");
        exit (1);
    }

/* input data : rsl-file */

    sprintf (fname, "%s", *++argv);
    fp = open_file (fname, "rsl", "r");
    fscanf (fp, "%lf", &omega);
    fscanf (fp, "%d", &ncoef);
    coef = (int *) calloc (ncoef, sizeof (int));
    if (coef == NULL)
        s_error (ERR, "coef[]");
    for (mcoef = 100, k = 0; k < ncoef; k++) {
        fscanf (fp, "%d", coef + k);
        if (coef[k] < mcoef)
            mcoef = coef[k];
    }
    fscanf (fp, "%d", &nindx);
    indx = (int *) calloc (nindx, sizeof (int));
```

```
    if (indx == NULL)
        s_error (ERR, "indx[]");
    eval = (COMPLEX *) calloc (nindx, sizeof (COMPLEX));
    if (eval == NULL)
        s_error (ERR, "eval[]");
    for (k = 0; k < nindx; k++)
        fscanf (fp, "%d", indx + k);

/* ---- main loop ---- */

    s_start ();
    fscanf (fp, "%s", nam);
    while (feof (fp) == 0) {
        s_cls (3);
    /* read data block */
        fscanf (fp, "%s", var);
        fscanf (fp, "%lf", &error);
        for (k = 0; k < nindx; k++)
            fscanf (fp, "%lf %lf", &eval[k].r, &eval[k].i);
    /* display data */
        fprintf (stderr, "----- %s -----\n", TI1);
        fprintf (stderr, "\t\t\t\t\t\t %s %s\n", TI2, fname);
        fprintf (stderr, "omega = %lE ( ", omega);
        for (k = 0; k < ncoef; k++)
            fprintf (stderr, "%d ", coef[k]);
        fprintf (stderr, ")\n");
        fprintf (stderr, "error = %lE ( %s[%s] )\n\n", error, nam, var);
        sprintf (ttl, "%s[%s] of %s", nam, var, fname);
        if (y_or_n (QM1))       /* ? graph draw */
            do
                s_graph (omega, mcoef, nindx, indx, eval, var, ttl);
            while (y_or_n (QM2));/* ? re-draw */
        fscanf (fp, "%s", nam);
    }
    fclose (fp);
    s_end ();
    return (1);
}


y_or_n (mess)
char   *mess;
{
    char    ans[10];

    s_mess (mess, ans, 1);
    while (ans[0] != 'y' && ans[0] != 'n') {
        fprintf (stderr, "\7\7");
        s_mess (mess, ans, 1);
    }
    return (ans[0] == 'y');
}


FILE   *open_file (name, ext, mode)
char   *name,
       *ext,
       *mode;
{
    char    fname[64];
    FILE   *fp;

    sprintf (fname, "%s.%s", name, ext);
    if ((fp = fopen (fname, mode)) == NULL) {
```

```
        fprintf (stderr, "\7\7");
        fprintf (stderr, "\nCANNOT OPEN FILE %s\n", fname);
        exit (1);
    }
    return (fp);
}
```