NONLINEAR ELECTRONICS (NOEL) PACKAGE 2:

COMPUTER GENERATION OF SYMBOLIC TRANSFER FUNCTION

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/ 24

21 March 1986

NONLINEAR ELECTRONICS (NOEL) PACKAGE 2:

COMPUTER GENERATION OF SYMBOLIC TRANSFER FUNCTION

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/24

21 March 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

NONLINEAR ELECTRONICS (NOEL) PACKAGE 2:

COMPUTER GENERATION OF SYMBOLIC TRANSFER FUNCTION

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/24

21 March 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

NOEL PACKAGE 2 : COMPUTER GENERATION OF SYMBOLIC TRANSFER FUNCTION†

An-Chang Deng and Leon O. Chua

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

ABSTRACT

The program in this package implements the Souriau-Frame algorithm and the frequency interpolation algorithm for finding the symbolic transfer function $\mathbf{H}(s)$ of a linear circuit. The computed transfer function is given by the coefficients $\mathbf{H}_0$, $\mathbf{H}_1$, ..., $\mathbf{H}_{n-1}$, q_0, q_1, ..., q_{n-1} in the explicit representation

$$\mathbf{H}(s) = \frac{\mathbf{H}_0 s^{n-1} + \mathbf{H}_1 s^{n-2} + ... + \mathbf{H}_{n-2} s + \mathbf{H}_{n-1}}{s^n + q_0 s^{n-1} + ... + q_{n-2} s + q_{n-1}}$$

which provides an efficient way for computing the frequency response of a linear circuit at various frequencies.

March 16, 1986

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

NOEL PACKAGE 2 :
COMPUTER GENERATION OF SYMBOLIC TRANSFER FUNCTION

1. Introduction

A linear dynamic circuit can be described by one of the various types of n-port representations in the frequency domain[1]. Each n-port representation is the preliminary and the fundamental step for circuit analysis as the circuit is characterized by the combination of the linear n-port equation and the external port element characteristics, which are either driving sources or output variables. The n-port representation, obtained by the routines in [1], provides the information on the circuit behavior at each frequency. However, it may need a lot of sampled frequencies in practical circuit design in order to accurately predict a wide range of frequency responses of the circuit, and will take considerable amount of cpu time for computing the n-port representation at all frequencies. This motivates us to develop the symbolic n-port representation; namely

$$\mathbf{y}(s) = (\mathbf{H}(s) + \mathbf{D})\mathbf{u}(s) + (\mathbf{c}(s) + \mathbf{s}_2)/s \tag{1}$$

for the linear circuit characterized by the explicit state equation[1] :

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{s}_1 \tag{2a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{s}_2 \tag{2b}$$

where

$$\mathbf{H}(s) = \mathbf{C}(s\,\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} = \frac{\mathbf{H}_0 s^{n-1} + \mathbf{H}_1 s^{n-2} + \ldots\ldots + \mathbf{H}_{n-2} s + \mathbf{H}_{n-1}}{s^n + q_0 s^{n-1} + \ldots\ldots + q_{n-2} s + q_{n-1}} \tag{3}$$

$$\mathbf{c}(s) = \mathbf{C}(s\,\mathbf{I} - \mathbf{A})^{-1}\mathbf{s}_1 = \frac{\mathbf{c}_0 s^{n-1} + \mathbf{c}_1 s^{n-2} + \ldots\ldots + \mathbf{c}_{n-2} s + \mathbf{c}_{n-1}}{s^n + q_0 s^{n-1} + \ldots\ldots + q_{n-2} s + q_{n-1}} \tag{4}$$

and $\mathbf{H}(s)$ is an $m \times m$ matrix such that each entry of $\mathbf{H}(s)$ is a rational function; namely

$$h_{ij}(s) = \frac{h_{0,ij} s^{n-1} + h_{1,ij} s^{n-2} + \ldots\ldots + h_{n-2,ij} s + h_{n-1,ij}}{s^n + q_0 s^{n-1} + \ldots\ldots + q_{n-2} s + q_{n-1}} \tag{5}$$

where $h_{k,ij}$ is the ij-th entry of the matrix $\mathbf{H}_k$ in Eq.(3) for k=0,1,..,n-1. Similarly, $\mathbf{c}(s)$ is an m-dimensional vector such that each of its entry is a rational function of s.

Two distinct algorithms are included in the routine for finding the symbolic n-port representation in Eq.(1); namely, the Souriau-Frame algorithm[2] and the frequency interpolation algorithm[3]. Both algorithms give the same results, except for circuits with widely separated parameters. In this case, the equation parameters span over a wide range of numerical values. This discrepancy is caused by the machine accuracy ϵ where

$$1 + \epsilon = 1 \tag{6}$$

In most computers, $\epsilon \approx 1.0 \times 10^{-16}$. It follows that if w is the number with largest absolute value in the equation parameters, then any number x such that $x \in S = \{z \mid abs(z) < w \times 10^{-16}\}$ is numerically indistinguishable from any other y in S.

2. Algorithm

2.1. Souriau-Frame algorithm

Step 1.

Find the linear state equation

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{s}_1 \tag{7a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{s}_2 \tag{7b}$$

for the linear dynamic circuit (see [1]), where $\mathbf{x}$ consists all linear capacitor voltages and linear inductor currents; $\mathbf{u}$ consists of the voltages (resp.; currents) of the driving voltage sources (resp.; driving current sources); $\mathbf{y}$ consists of the currents (resp.; voltages) of the driving voltage sources (resp.; driving current sources).

Step 2.

Follow the formula

$$(s\mathbf{I}-\mathbf{A})^{-1} = \frac{adj\,(s\mathbf{I}-\mathbf{A})}{|\,s\mathbf{I}-\mathbf{A}\,|} = \frac{\mathbf{T}_0 s^{n-1} + \mathbf{T}_1 s^{n-2} + \vdots\ldots + \mathbf{T}_{n-2}s + \mathbf{T}_{n-1}}{s^n + q_0 s^{n-1} + \ldots\ldots + q_{n-2}s + q_{n-1}} \tag{8}$$

to find $\mathbf{F}(s) = (s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}$, and $\mathbf{r}(s) = (s\mathbf{I}-\mathbf{A})^{-1}\mathbf{s}_1$, where

$$\mathbf{T}_0 = \mathbf{I}$$

$$q_0 = -tr\,(\mathbf{A})$$

$$\mathbf{T}_1 = \mathbf{T}_0\mathbf{A} + r_0\mathbf{I}$$

$$q_1 = -\frac{1}{2}tr\,(\mathbf{T}_1\mathbf{A})$$

.....

.....

$$\mathbf{T}_k = \mathbf{T}_{k-1}\mathbf{A} + r_{k-1}\mathbf{I}$$

$$q_k = -\frac{1}{k}tr\,(\mathbf{T}_k\mathbf{A})$$

Step 3.

Find the transfer function

$$\mathbf{H}(s) = \mathbf{C} * \mathbf{F} \tag{9a}$$

$$\mathbf{c}(s) = \mathbf{C} * \mathbf{r} \tag{9b}$$

such that

$$\mathbf{y}(s) = (\mathbf{H}(s) + \mathbf{D})\mathbf{u}(s) + (\mathbf{c}(s) + \mathbf{s}_2)/s \tag{10}$$

2.2. Frequency interpolation algorithm

Step 1.

Same as Step 1 of Section 2.1.

Step 2.

Let $\mathbf{F}(s) = (s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}$, then $\mathbf{F}(s)$ is a rational function matrix

$$\mathbf{F}(s) = \frac{\mathbf{F}_0 s^{n-1} + \mathbf{F}_1 s^{n-2} + \ldots\ldots + \mathbf{F}_{n-2} s + \mathbf{F}_{n-1}}{s^n + q_0 s^{n-1} + \ldots\ldots + q_{n-2} s + q_{n-1}} \tag{11}$$

where $q(s)$ is the characteristic polynomial

$$q(s) = \det(s\mathbf{I} - \mathbf{A}) \tag{12}$$

and $\mathbf{F}_i \in \mathbf{R}^{n \times m}$ for i=0,1,...,n-1.

Step 3.

Find $\mathbf{F}^{(i)} = (s_i \mathbf{I} - \mathbf{A})^{-1} * \mathbf{B}$ at $s_i = e^{j\frac{2\pi i}{n+1}}$ for i=0,1,...,n-1.

Step 4.

Use discrete Fourier transformation to find

$$\mathbf{F}_k = \frac{1}{n+1} \sum_{t=0}^{n} \mathbf{F}^{(t)} e^{-j\frac{2\pi tk}{n+1}} \tag{13}$$

Step 5.

Use discrete Fourier transformation to find

$$q_k = \frac{1}{n+1} \sum_{t=0}^{n} d^{(t)} e^{-j\frac{2\pi tk}{n+1}} \tag{14}$$

where

$$d^{(t)} = \det(s_t \mathbf{I} - \mathbf{A}) \tag{15}$$

3. User's Instruction

Step 1.

Create a file "xx...x.spc" which describes the circuit to be analyzed and follows the rules of the input format language defined in [4] for each class of circuit elements, where "xx...x" is the filename. All the linear elements and the time-varying sources can be included in "xx...x.spc"; namely

'R' : 2-terminal linear resistor

'C' : 2-terminal linear capacitor

'L' : 2-terminal linear inductor

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

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

'E' : linear voltage-controlled voltage source

'F' : linear current-controlled current source

'G' : linear voltage-controlled current source

'H' : linear current-controlled voltage source

Step 2.

Type the command

symtrf xx...x

to find the symbolic transfer function with three options :

enter the option number

1 : Souriau-Frame algorithm

2 : Frequency interpolation algorithm

3 : Both

It is recommanded to use option 1 when the circuit has no more than 10 dynamic elements since it saves the computation time. The accuracy of the Souriau-Frame algorithm, however, will deteriorate with larger circuit size and should use the frequency interpolation algorithm. Option 3 for choosing both algorithms provides a comparison on the computed solutions.

Remark :

The symbolic transfer function obtained by **symtrf** includes the impedance, admittance, and any possible combination of hybrid representation. Connect a driving voltage source (resp.; current source) across a port which is to be a voltage-controlled port (resp.; current-controlled port). In particular, the admittance representation of the linear dynamic n-port can be found by connecting each port with a driving voltage source.

4. Output Format

In addition to the standard output from the screen, the symbolic transfer function is also written to the output file "xx...x.trf" with the following format :

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

        { input file listing }

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

USING SOURIAU-FRAME ALGORITHM
```

or

```
USING FREQUENCY INTERPOLATION ALGORITHM

********** SYMBOLIC TRANSFER FUNCTION **********

          y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s
          H(s) = h(s)/q(s) ,   c(s) = g(s)/q(s)

dimension m = ..

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

   { input variable definitions }

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

***** output variables *****

   { output variable definitions }

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

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

D[i,j] = ......

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

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

s_2[i] = ......

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

********** q(s) **********

q(s) = s^n + q[0]*s^(n-1) + ... + q[n-2]*s + q[n-1]

order n = ..
q[0] = ...
q[1] = ...
....
....
q[n-1] = ...

********** h(s) **********

h(s) = h[0]*s^(n-1) + h[1]*s^(n-2) + ... + h[n-2]*s + h[n-1]

h_ij[0] = ...
h_ij[1] = ...
....
....
h_ij[n-1] = ...
i=1,2,...,m and j=1,2,...,m

********** g(s) **********

g(s) = g[0]*s^(n-1) + g[1]*s^(n-2) + ... + g[n-2]*s + g[n-1]

g_i[0] = ...
g_i[1] = ...
....
....
g_i[n-1] = ...
i=1,2,...,m

Remark :

q(s) is the characteristic polynomial of the matrix A and each q[i], i=0,1,...,n-1, is a scalar; h[i], i=0,1,...,n-1, is an m by m matrix, where m is the number of input variables (or output variables) in the circuit, h_ij[k] denotes the ij-th entry of of the matrix h[k]; $c(s) = g(s)/q(s)$ is contributed by the constant sources in the circuit, and each g[i] is an m-dimensional vector.

5. Examples

Example 1 : in file "ex1.spc"

A linear RLC 1-port circuit (Fig.1)

Example 2 : in file "ex2.spc"

An LC ladder 2-port circuit (Fig.2)

6. Diagnosis

1. See Section 6.1 of [1].

2. SYMTRF SPICE_FILE

   Incorrect command line, the correct one should be

   symtrf xx...x

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

3. CAN'T OPEN THE INPUT FILE xx...x.spc

   Unable to open and read the input file "xx...x.spc".

4. CAN'T OPEN THE TRANSFER FUNCTION FILE xx...x.trf

   Unable to open the output file "xx...x.trf" which is to store the parameters of the symbolic transfer function.

5. CAPACITOR LOOP OR INDUCTOR CUTSET

   There exists a capacitor loop or inductor cutset in the circuit.

References

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

[2] L.O.Chua and P.M.Lin, *Computer aided analysis of electronic circuits : algorithms and computational techniques* , Englewood Cliffs, NJ : Prentice-Hall, 1975.

[3] J.Vlach and K.Singhal, *Computer methods for circuit analysis and design* , Van Nostrand Reinhold, 1983.

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

Figure Captions

Fig.1 A linear RLC 1-port circuit.

Fig.2 An LC ladder 2-port circuit.

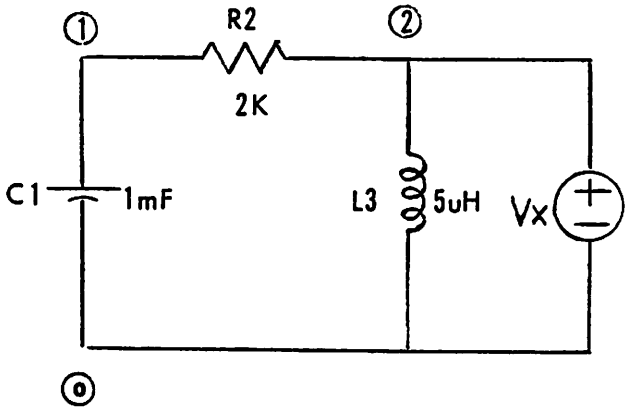
①
R2
②
2K
C1
1mF
L3
5uH
Vx
⓪

Fig.1

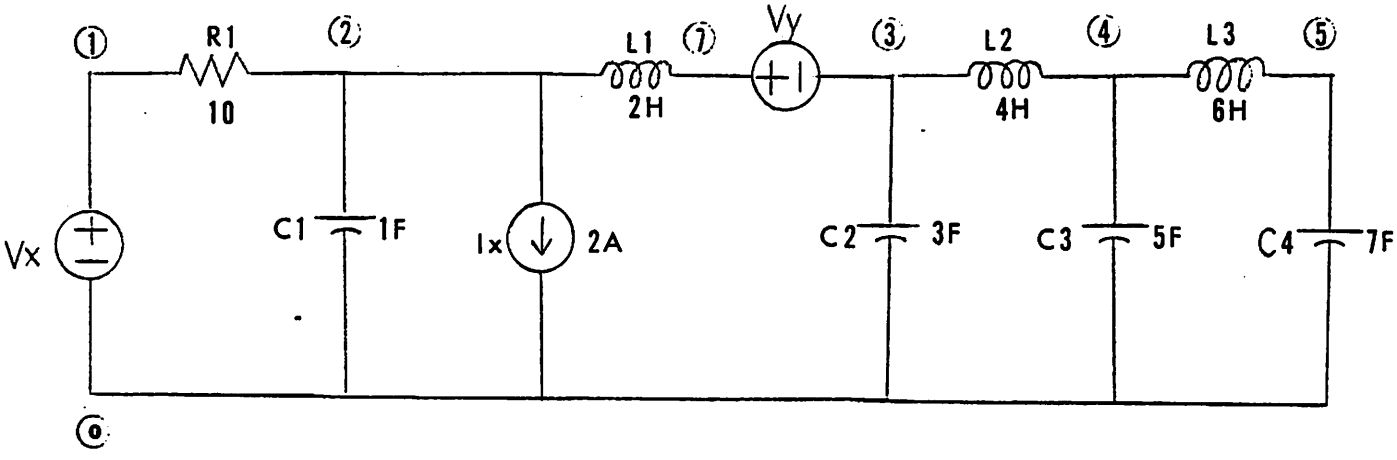
①
R1
②
10
L1
⑦
2H
Vy
③
L2
④
4H
L3
⑤
6H
Vx
C1
1F
Ix
2A
C2
3F
C3
5F
C4
7F
⓪

Fig.2

```
* Example 1
*
* linear RLC 1-port circuit
*
C1 1 0 1m
R2 1 2 2K
Vx 2 0 {sin(1000*t)}
L3 2 0 5u
.end
```

```
********************** SPICE INPUT **************************
* Example 1
*
* linear RLC 1-port circuit
*
C1 1 0 1m
R2 1 2 2K
Vx 2 0 (sin(1000*t))
L3 2 0 5u
.end
*************************************************************
```

USING SOURIAU-FRAME ALGORITHM .....

```
************** SYMBOLIC TRANSFER FUNCTION ***************
          y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s
          H(s) = h(s)/q(s) ,  c(s) = g(s)/q(s)
dimension m = 1


***** input variables *****
u[1] = v[2,0](Vx)


***** output varisbles *****
y[1] = i[2,0](Vx)


****************** D matrix ****************
D[1,1] = -5.000000e-04


***************** s_2 vector ***************
s_2[1] = 0.000000e+00


******************** q(s) ******************
q(s) = s^n + q[0]*s^(n-1) + ..... + q[n-2]*s + q[n-1]

order n = 2

q[0] = 5.000000e-01
q[1] = 0.000000e+00


******************** h(s) ******************
h(s) = h[0]*s^(n-1) + h[1]*s^(n-2) + ..... + h[n-2]*s + h[n-1]

h_11[0] = -2.000000e+05
h_11[1] = -1.000000e+05
```

```
******************** g(s) ******************
g(s) = g[0]*s^(n-1) + g[1]*s^(n-2) + ..... + g[n-2]*s + g[n-1]

g_1[0] = 0.000000e+00
g_1[1] = 0.000000e+00



USING FREQUENCY INTERPOLATION ALGORITHM .....

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

************** SYMBOLIC TRANSFER FUNCTION ***************
          y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s
          H(s) = h(s)/q(s) ,  c(s) = g(s)/q(s)
dimension m = 1


***** input variables *****
u[1] = v[2,0](Vx)


***** output variables *****
y[1] = i[2,0](Vx)


****************** D matrix ****************
D[1,1] = -5.000000e-04


***************** s_2 vector ***************
s_2[1] = 0.000000e+00


******************** q(s) ******************
q(s) = s^n + q[0]*s^(n-1) + ..... + q[n-2]*s + q[n-1]

order n = 2

q[0] = 5.000000e-01
q[1] = 1.480297e-16


******************** h(s) ******************
h(s) = h[0]*s^(n-1) + h[1]*s.^(n-2) + ..... + h[n-2]*s + h[n-1]

h_11[0] = -2.000000e+05
h_11[1] = -1.000000e+05


******************** g(s) ******************
g(s) = g[0]*s.^(n-1) + g[1]*s.^(n-2) + ..... + g[n-2]*s + g[n-1]

g_1[0] = 0.000000e+00
```

g_1[1] = 0.000000e+00

```
* Example 2
*
* LC ladder circuit
*
Vx 1 0 (sin(t))
R1 1 2 10
C1 2 0 1
Ix 2 0 2
L1 2 7 2
Vy 7 3 (cos(t))
C2 3 0 3
L2 3 4 4
C3 4 0 5
L3 4 5 6
C4 5 0 7
.end
```

```
********** SPICE INPUT **********
* Example 2
*
* LC ladder circuit
*
Vx 1 0 (sin(t))
R1 1 2 10
C1 2 0 1
Ix 2 0 2
L1 2 7 2
Vy 7 3 (cos(t))
C2 3 0 3
L2 3 4 4
C3 4 0 5
L3 4 5 6
C4 5 0 7
.end
**********
```

USING SOURIAU-FRAME ALGORITHM .....

```
********** SYMBOLIC TRANSFER FUNCTION **********
          y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s
          H(s) = h(s)/q(s) ,  c(s) = g(s)/q(s)
dimension n = 2


***** input variables *****
u[1] = v[7,3](Vy)
u[2] = v[1,0](Vx)


***** output variables *****
y[1] = i[7,3](Vy)
y[2] = i[1,0](Vx)


********** D matrix **********
D[1,1] = 0.000000e+00
D[1,2] = 0.000000e+00
D[2,1] = 0.000000e+00
D[2,2] = -1.000000e-01


********** s_2 vector **********
s_2[1] = 0.000000e+00
s_2[2] = 0.000000e+00


********** q(s) **********
q(s) = s^n + q[0]*s^(n-1) + ..... + q[n-2]*s + q[n-1]
```

order n = 7

q[0] = 1.000000e-01
q[1] = 8.571429e-01
q[2] = 3.571429e-02
q[3] = 1.190476e-01
q[4] = 2.380952e-03
q[5] = 3.174603e-03
q[6] = 1.984127e-05

****************** h(s) ******************
h(s) = h[0]*s^(n-1) + h[1]*s^(n-2) + ... + h[n-2]*s + h[n-1]

h_11[0] = -5.000000e-01
h_11[1] = -5.000000e-02
h_11[2] = -9.523810e-02
h_11[3] = -9.523810e-03
h_11[4] = -2.976190e-03
h_11[5] = -2.976190e-04
h_11[6] = 1.084202e-18

h_12[0] = 0.000000e+00
h_12[1] = 5.000000e-02
h_12[2] = 9.251859e-20
h_12[3] = 9.523810e-03
h_12[4] = -1.635596e-19
h_12[5] = 2.976190e-04
h_12[6] = 3.263621e-20

h_21[0] = 2.775558e-17
h_21[1] = 5.000000e-02
h_21[2] = 2.973812e-18
h_21[3] = 9.523810e-03
h_21[4] = 3.304235e-20
h_21[5] = 2.976190e-04
h_21[6] = -1.084202e-19

h_22[0] = 1.000000e-02
h_22[1] = -2.775558e-18
h_22[2] = 3.571429e-03
h_22[3] = -2.973812e-19
h_22[4] = 2.380952e-04
h_22[5] = -2.498828e-20
h_22[6] = 1.984127e-06

****************** g(s) ******************
g(s) = g[0]*s^(n-1) + g[1]*s^(n-2) + ... + g[n-2]*s + g[n-1]

g_1[0] = 0.000000e+00
g_1[1] = -1.000000e+00
g_1[2] = 0.000000e+00
g_1[3] = -1.904762e-01
g_1[4] = 3.469447e-18
g_1[5] = -5.952381e-03

g_1[6] = -6.505213e-19

g_2[0] = -2.000000e-01
g_2[1] = 5.551115e-17
g_2[2] = -7.142857e-02
g_2[3] = 5.947623e-18
g_2[4] = -4.761905e-03
g_2[5] = 4.997656e-19
g_2[6] = -3.968254e-05

USING FREQUENCY INTERPOLATION ALGORITHM .....

********** SYMBOLIC TRANSFER FUNCTION **********
y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s
H(s) = h(s)/q(s) , c(s) = g(s)/q(s)
dimension m = 2

***** input variables *****
u[1] = v[7,3](Vy)
u[2] = v[1,0](Vx)

***** output variables *****
y[1] = i[7,3](Vy)
y[2] = i[1,0](Vx)

********** D matrix **********
D[1,1] = 0.000000e+00
D[1,2] = 0.000000e+00
D[2,1] = 0.000000e+00
D[2,2] = -1.000000e-01

********** s_2 vector **********
s_2[1] = 0.000000e+00
s_2[2] = 0.000000e+00

********** q(s) **********
q(s) = s^n + q[0]*s^(n-1) + ... + q[n-2]*s + q[n-1]

order n = 7

q[0] = 1.000000e-01
q[1] = 8.571429e-01
q[2] = 3.571429e-02
q[3] = 1.190476e-01
q[4] = 2.380952e-03

q[5] = 3.174603e-03
q[6] = 1.984127e-05

****************** h(s) ******************
h(s) = h[0]*s^(n-1) + h[1]*s^(n-2) + ... + h[n-2]*s + h[n-1]

h_11[0] = -5.000000e-01
h_11[1] = -5.000000e-02
h_11[2] = -9.523810e-02
h_11[3] = -9.523810e-03
h_11[4] = -2.976190e-03
h_11[5] = -2.976190e-04
h_11[6] = -1.387779e-17

h_12[0] = 1.301043e-18
h_12[1] = 5.000000e-02
h_12[2] = -3.469447e-18
h_12[3] = 9.523810e-03
h_12[4] = 4.336809e-19
h_12[5] = 2.976190e-04
h_12[6] = 7.806256e-18

h_21[0] = 2.914335e-17
h_21[1] = 5.000000e-02
h_21[2] = 2.279922e-18
h_21[3] = 9.523810e-03
h_21[4] = 3.304235e-20
h_21[5] = 2.976190e-04
h_21[6] = 5.551115e-18

h_22[0] = 1.000000e-02
h_22[1] = -2.255141e-18
h_22[2] = 3.571429e-03
h_22[3] = -2.106450e-19
h_22[4] = 2.380952e-04
h_22[5] = 1.731419e-18
h_22[6] = 1.984127e-06

****************** g(s) ******************
g(s) = g[0]*s^(n-1) + g[1]*s^(n-2) + ... + g[n-2]*s + g[n-1]

g_1[0] = 4.163336e-17
g_1[1] = -1.000000e+00
g_1[2] = 6.938894e-17
g_1[3] = -1.904762e-01
g_1[4] = 0.000000e+00
g_1[5] = -5.952381e-03
g_1[6] = -9.714451e-17

g_2[0] = -2.000000e-01
g_2[1] = 5.273559e-17
g_2[2] = -7.142857e-02
g_2[3] = 5.947623e-18
g_2[4] = -4.761905e-03

```
g_2[5] = -1.658726e-17
g_2[6] = -3.968254e-05
```

APPENDIX

SOURCE CODE LISTINGS

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

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

/*****************************************************************************/
/* This is the main program of the routine "symtrf" which finds the symbolic */
/* transfer function                                                         */
/*              y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s                      */
/* of a linear dynamic circuit described by the state equation               */
/*              x' = A*x + B*u + s_1                                         */
/*              y  = C*x + D*u + s_2                                         */
/*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    char *calloc(),lx[100];
    int i,j,k;
    double *A,*B,*t1,*t2,*C,*D,*s1,*s2,*p,*q,*r,*h,*g,*K;
    FILE *fp,*gp,*hp;
    int n,m,l,opt;

    /* open the input file "xx...x.spc" and the table file "xx...x.tbl" */
    open_spc_tbl(argc,argv,&fp,&gp,&hp);
    n=NELEM;
    state_eq(fp,gp,&n,&m,&l,&A,&B,&C,&D,&s1,&s2,&t1,&t2);
    fclose(fp);
    fclose(gp);

    /* allocate spaces for the parameters of symbolic transfer function */
    alloc_p(n,m,&p,&q,&r,&h,&g,&K);

    printf("enter the option number of the algorithm\n");
    printf("1 : Souriau-Frame algorithm\n");
    printf("2 : Frequency interpolation algorithm\n");
    printf("3 : Both\n");
    scanf("%d",&opt);

    /* Souriau-Frame algorithm */
    if (opt == 1 || opt == 3)
    {
        sprintf(lx,"\n\nUSING SOURIAU-FRAME ALGORITHM .....\n\n");
        printf("%s",lx);
        fprintf(hp,"%s",lx);

        /* find inv(s*I-A)*B and inv(s*I-A)*s_1 */
        s_f_alg(n,m,A,B,s1,p,q,r);

        /* find C*inv(s*I-A)*B and C*inv(s*I-A)*s_1 */
        get_hg(n,m,C,p,r,h,g);

        /* print the symbolic representation */
        print_trf(hp,n,m,D,s2,p,q,r,h,g);
```

```
    }

    /* frequency interpolation algorithm */
    if (opt == 2 || opt == 3)
    {
        sprintf(lx,"\n\nUSING FREQUENCY INTERPOLATION ALGORITHM .....\n\n");
        printf("\n\n%s",lx);
        fprintf(hp,"\n\n%s",lx);

        /* choose K=I */
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
            {
                if (i==j)
                    K[i*n+j]=1.0;
                else
                    K[i*n+j]=0.0;
            }

        /* find inv(s*I-A)*B and inv(s*I-A)*s_1 */
        trsf(n,m,K,A,B,s1,p,q,r);

        /* find C*inv(s*I-A)*B and C*inv(sI-A)*s_1 */
        get_hg(n,m,C,p,r,h,g);

        /* print the symbolic representation */
        print_trf(hp,n,m,D,s2,p,q,r,h,g);
    }
    fclose(hp);
}

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

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

    if (argc != 2)
        exit_message("SYMTRF SPICE_FILE");
    sprintf(fname,"%s.tbl",*++argv);
    if ((*gp=fopen(fname,"w")) == NULL)
    {
        printf("CAN'T OPEN THE TABLE FILE %s\n",fname);
        exit();
    }
    sprintf(fname,"%s.trf",*argv);
    if ((*hp=fopen(fname,"w")) == NULL)
    {
        printf("CAN'T OPEN THE TRANSFER_FUNCTION FILE %s\n",fname);
        exit();
```

```
    }
    sprintf(fname,"%s.spc",*argv);
    print_spc(fname,*hp);
    if ((*fp=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE INPUT FILE %s\n",fname);
        exit();
    }
}

/*****************************************************************************/
/* Print the input file "xx...x.spc" on the screen and the output file       */
/* "xx...x.trf".                                                             */
/*****************************************************************************/

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

    sprintf(line,"********************* SPICE INPUT ************************");
    printf("%s\n",line);
    fprintf(hp,"%s\n",line);
    if ((ip=fopen(fname,"r")) == NULL)
    {
        printf("CAN'T OPEN THE INPUT FILE %s\n",fname);
        exit();
    }
    while (fgets(line,80,ip) != NULL)
    {
        printf("%s",line);
        fprintf(hp,"%s",line);
        if (find_index(".end",line) == 0 || find_index(".END",line) == 0)
            break;
    }
    sprintf(line,"*****************************************************************");
    printf("%s\n\n",line);
    fprintf(hp,"%s\n\n",line);
    fclose(ip);
}

/*****************************************************************************/
/* Allocate spaces for the parameters of the symbolic transfer function.     */
/*****************************************************************************/

alloc_p(n,m,p,q,r,h,g,K)
int n,m;
double **p,**q,**r,**h,**g,**K;
{
    char *calloc();

    *p=(double *)calloc((n+1)*n*m,sizeof(double));
    *q=(double *)calloc(n+1,sizeof(double));
    *r=(double *)calloc((n+1)*n,sizeof(double));
```

```
    *h=(double *)calloc(n*m*m,sizeof(double));
    *g=(double *)calloc(n*m,sizeof(double));
    *K=(double *)calloc(n*n,sizeof(double));
}

/*****************************************************************************/
/* Find the transfer function                                                */
/*              H(s) = C*F(s)                                                */
/*              c(s) = C*r(s)                                                */
/* where                                                                     */
/*              F(s) = inv(sI-A)*B                                           */
/*              r(s) = inv(sI-A)*s_1                                         */
/*****************************************************************************/

get_hg(n,m,C,p,r,h,g)
int n,m;
double *C,*p,*r,*h,*g;
{
    int i,ix,j,k;

    for (ix=0;ix<n;ix++)
       for (i=0;i<m;i++)
       {
            for (j=0;j<m;j++)
            {
                h[ix*m*m+i*m+j] = 0.0;
                for (k=0;k<n;k++)
                    h[ix*m*m+i*m+j]+=C[i*n+k]*p[ix*n*m+k*m+j];
            }
            g[ix*m+i] = 0.0;
            for (k=0;k<n;k++)
                g[ix*m+i]+=C[i*n+k]*r[ix*n+k];
       }
}

/*****************************************************************************/
/* Print the symbolic transfer function to the output file "xx...x.trf"      */
/* and the standard output.                                                  */
/*****************************************************************************/

print_trf(hp,n,m,D,s2,p,q,r,h,g)
FILE *hp;
int n,m;
double *D,*s2,*p,*q,*r,*h,*g;
{
    int i,j,k;
    char lx[100];

    /* print the headings */
    sprintf(lx,"-------------------------------------------------------------");
    printf("%s\n\n",lx);
    fprintf(hp,"%s\n\n",lx);
    sprintf(lx,"************* SYMBOLIC TRANSFER FUNCTION *************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,"          y(s) = (H(s) + D)*u(s) + (c(s) + s_2)/s\n");
```

```
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,"          H(s) = h(s)/q(s) ,  c(s) = g(s)/q(s)\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);

    /* print the order of the transfer function */
    sprintf(lx,"dimension m = %d\n",m);
    printf("%s",lx);
    fprintf(hp,"%s",lx);

    /* print the definitions of the input variables */
    sprintf(lx,"\n\n***** input variables *****\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<m;i++)
    {
        j=(branch_vector+i+n)->a;
        if ((branch+j)->name[0] == 'V')
            sprintf(lx,"u[%d] = v[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        else
            sprintf(lx,"u[%d] = i[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }

    /* print the definitions of the output variables */
    sprintf(lx,"\n\n***** output variables *****\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<m;i++)
    {
        j=(branch_vector+i+n)->a;
        if ((branch+j)->name[0] == 'V')
            sprintf(lx,"y[%d] = i[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        else
            sprintf(lx,"y[%d] = v[%d,%d](%s)\n",i+1,(branch+j)->node1,
                    (branch+j)->node2,(branch+j)->name);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }

    /* print the constant matrix D */
    sprintf(lx,"\n\n**************** D matrix ***************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<m;i++)
        for (j=0;j<m;j++)
        {
            sprintf(lx,"D[%d,%d] = %.6e\n",i+1,j+1,D[i*m+j]);
            printf("%s",lx);
            fprintf(hp,"%s",lx);
        }
```

```
    /* print the source vector s_2 */
    sprintf(lx,"\n\n*************** s_2 vector **************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<m;i++)
    {
        sprintf(lx,"s_2[%d] = %.6e\n",i+1,s2[i]);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }

    /* print the characteristic polynomial q(s) */
    sprintf(lx,"\n\n****************** q(s) ****************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,"q(s) = s^n + q[0]*s^(n-1) + ..... + q[n-2]*s + q[n-1]\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,"\norder n = %d\n\n",n);
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    for (i=0;i<n;i++)
    {
        sprintf(lx,"q[%d] = %.6e\n",i,q[i]);
        printf("%s",lx);
        fprintf(hp,"%s",lx);
    }

    /* print the nominator h(s) of the transfer function */
    sprintf(lx,"\n\n******************* h(s) ****************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,
            "h(s) = h[0]*s^(n-1) + h[1]*s^(n-2) + ..... + h[n-2]*s + h[n-1]");
    printf("%s\n\n",lx);
    fprintf(hp,"%s\n\n",lx);
    for (i=0;i<m;i++)
        for (j=0;j<m;j++)
        {
            for (k=0;k<n;k++)
            {
                sprintf(lx,"h_%d%d[%d] = %.6e\n",i+1,j+1,k,h[k*m*m+i*m+j]);
                printf("%s",lx);
                fprintf(hp,"%s",lx);
            }
            printf("\n");
            fprintf(hp,"\n");
        }

    /* print the nominator g(s) of the constant source vector */
    sprintf(lx,"\n******************* g(s) ****************\n");
    printf("%s",lx);
    fprintf(hp,"%s",lx);
    sprintf(lx,
            "g(s) = g[0]*s^(n-1) + g[1]*s^(n-2) + ..... + g[n-2]*s + g[n-1]");
```

```
    printf("%s\n\n",lx);
    fprintf(hp,"%s\n\n",lx);
    for (i=0;i<m;i++)
    {
        for (k=0;k<n;k++)
        {
            sprintf(lx,"g_%d[%d] = %.6e\n",i+1,k,g[k*m+i]);
            printf("%s",lx);
            fprintf(hp,"%s",lx);
        }
        printf("\n");
        fprintf(hp,"\n");
    }
}
```

```
#include <stdio.h>

/*****************************************************************************/
/* Find the trace of a matrix.                                               */
/*****************************************************************************/

double trace(n,A)
int n;
double *A;
{
   int i;
   double y=0.0;

   for (i=0;i<n;i++)
      y+=A[i*n+i];
   return(y);
}

/*****************************************************************************/
/* Find the symbolic expression of the transfer function matrix by using    */
/* Souriau-Frame Algorithm (see pp.378 in Chua and Lin's book).              */
/*****************************************************************************/

s_f_alg(n,m,a,b,s,p,q,r)
int n,m;
double *a,*b,*p,*q,*s,*r;
{
    char *calloc();
    int i,j;
    double trace();
    double *c,*e;

    /* temporary storage */
    c=(double *)calloc(n*n,sizeof(double));
    e=(double *)calloc(n*n,sizeof(double));

    /* initialization of the algorithm */
    for (i=0;i<n;i++)
    {
        r[i]=s[i];
        for (j=0;j<m;j++)
            p[i*m+j]=b[i*m+j];
        for (j=0;j<n;j++)
            if (j==i)
                c[i*n+j]=1.0;
    }

    /* Souriau-Frame algorithm */
    for (i=1;i<=n;i++)
    {
        multiply(n,n,n,c,a,e);
        q[i-1] = -1*trace(n,e)/i;
        add_diah(n,e,c,q[i-1]);
        if (i < n)
        {
            multiply(n,n,m,c,b,p+i*n*m);
```

```
            multiply(n,n,1,c,s,r+i*n);
        }
    }
}

/*****************************************************************************/
/* Find    b = a + q*I .                                                     */
/*****************************************************************************/

add_diah(n,a,b,q)
int n;
double *a,*b,q;
{
    int i;

    for (i=0;i<n*n;i++)
        b[i]=a[i];
    for (i=0;i<n;i++)
        b[i*n+i]+=q;
}

/*****************************************************************************/
/* Find the matrix multiplication  c = a*b  where a is an n by m matrix,     */
/* b is an m by k matrix, and c is an n by k matrix.                         */
/*****************************************************************************/

multiply(n,m,k,a,b,c)
int n,m,k;
double *a,*b,*c;
{
    int i,j,l;

    for (i=0;i<n;i++)
        for (j=0;j<k;j++)
        {
            c[i*k+j]=0.0;
            for (l=0;l<m;l++)
                c[i*k+j]+=a[i*m+l]*b[l*k+j];
      }
}
```

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

int *ipvt;
COMPLEX *aa,*cc,*tt,*zq,*z,*zz,*dd,*qx;

/*****************************************************************************/
/* Using the unit circuit interpolation algorithm to find the symbolic       */
/* transfer functions F(s) and r(s) such that                                */
/*                                -1                              -1         */
/*              F(s) = (sK-A)   * B   and   r(s) = (sK-A)   * s_1             */
/* and the circuit is assumed to be described by the implicit state          */
/* equation                                                                  */
/*              K*x' = A*x + B*u + s_1                                       */
/*****************************************************************************/

trsf(n,m,c,a,b,s1,P,q,r)
int n,m;
double *a,*b,*c,s1[],P[],q[],r[];
{
int i,j,k,job=0;
double cx,pi,atan2();
COMPLEX s;
COMPLEX *cmplx(),*cexp(),*cmult(),*ccopy();
double creal(),cimag(),rcond;

    /* allocate spaces for the complex variables */
    cpx_alloc(n,m,a,b,c);

    pi=atan2(1.0,1.0)*4;

    /* interpolation of each frequency */
    for (i=0;i<=n;i++)
    {
        cx=2*i*pi/(n+1);
        cmplx(&s,0.0,cx);
        cexp(&s,&s);

        /* compute the matrix tt=(sK-A) at the interpolated frequency */
        get_matrix(n,cc,aa,tt,&s);

        /* LU decomposition of tt */
        cgeco(tt,n,ipvt,&rcond,zq);

        /* find the determinant of tt */
        cgedi(tt,n,ipvt,&dd[i],z,10);

        /* compute inv(tt)*B and inv(tt)*d */
        for (k=0;k<=m;k++)
        {
            for (j=0;j<n;j++)
            {
                if (k==m)
                    cmplx(&z[j],s1[j],0.0);
                else
                    cmplx(&z[j],b[j*m+k],0.0);
```

```
            }
            cgesl(tt,n,ipvt,z,job);
            for (j=0;j<n;j++)
                cmult(&zz[i*n*(m+1)+j*(m+1)+k],&dd[i],&z[j]);
        }
    }

    /* recover the transfer function parameters from each */
    /* interpolated frequency                             */
    for (k=0;k<n;k++)
        for (j=0;j<=m;j++)
        {
            for (i=0;i<=n;i++)
                ccopy(&z[i],&zz[i*n*(m+1)+k*(m+1)+j]);
            dft(n,qx,z);
            for (i=0;i<n;i++)
            {
                if (j<m)
                    P[i*n*m+k*m+j]=creal(&qx[n-i-1]);
                else
                    r[i*n+k]=creal(&qx[n-i-1]);
            }
        }
    dft(n,qx,dd);
    for (i=0;i<n;i++)
        q[i]=creal(&qx[n-i-1]);

    /* free the spaces for the complex variables which are no */
    /* longer required                                        */
    cpx_free();
}

/*************************************************************************/
/* Allocate spaces for complex variables.                                */
/*************************************************************************/

cpx_alloc(n,m,a,b,c)
int n,m;
double *a,*b,*c;
{
    char *calloc();
    int i;
    COMPLEX *cmplx();

    ipvt=(int *)calloc(n,sizeof(int));
    zq=(COMPLEX *)calloc(n,sizeof(COMPLEX));
    z=(COMPLEX *)calloc(n+1,sizeof(COMPLEX));
    dd=(COMPLEX *)calloc(n+1,sizeof(COMPLEX));
    qx=(COMPLEX *)calloc(n+1,sizeof(COMPLEX));
    zz=(COMPLEX *)calloc((n+1)*n*(m+1),sizeof(COMPLEX));
    aa=(COMPLEX *)calloc(n*n,sizeof(COMPLEX));
    cc=(COMPLEX *)calloc(n*n,sizeof(COMPLEX));
    tt=(COMPLEX *)calloc(n*n,sizeof(COMPLEX));
    for (i=0;i<n*n;i++)
    {
        cmplx(aa+i,a[i],0.0);
```

```
        cmplx(cc+i,c[i],0.0);
    }
}

/*************************************************************************/
/* Free the spaces occupied by the complex variables which are no linger */
/* required.                                                             */
/*************************************************************************/

cpx_free()
{
    cfree(ipvt);
    cfree(zq);
    cfree(z);
    cfree(dd);
    cfree(qx);
    cfree(zz);
    cfree(aa);
    cfree(cc);
    cfree(tt);
}

/*************************************************************************/
/* Compute the matrix c=s*a-b  at a given frequency s.                   */
/*************************************************************************/

get_matrix(n,a,b,c,s)
int n;
COMPLEX *a,*b,*c,*s;
{
   int i;
   COMPLEX *csub(),*cmult();

    for (i=0;i<n*n;i++)
    {
        cmult(c+i,a+i,s);
        csub(c+i,c+i,b+i);
    }
}

/*************************************************************************/
/* Compute the discrete Fourier transform                                */
/*                          jk                                           */
/*            y_k = sum a_j*w        j from 0 to n; for k=0,1,...,n      */
/* where                                                                 */
/*              w = exp(2*pi*i/(n+1)) and i*i = -1                       */
/*************************************************************************/

dft(n,a,y)
int n;
COMPLEX *a,*y;
{
    int i,j;
    double cx,pi,atan2(),creal(),cimag();
    COMPLEX w,*cmplx(),*cexp(),*cadd();
```

```
    pi=4*atan2(1.0,1.0);
    for (i=0;i<=n;i++)
    {
        cmplx(&a[i],0.0,0.0);
        for (j=0;j<=n;j++)
        {
            cx = -2*pi*i*j/(n+1);
            cmplx(&w,0.0,cx);
            cexp(&w,&w);
            cmult(&w,&w,&y[j]);
            cadd(&a[i],&a[i],&w);
        }
        cmplx(&w,1.0/(n+1),0.0);
        cmult(&a[i],&a[i],&w);
    }
}
```