SNAP: A COMPUTER PROGRAM FOR GENERATING SYMBOLIC NETWORK FUNCTIONS
125 p CSCL 09B

Reproduced for NASA
by the
NASA Scientific and Technical Information Facility
SNAP
A COMPUTER PROGRAM FOR GENERATING
SYMBOLIC NETWORK FUNCTIONS

P. M. Lin
G. E. Alderson

School of Electrical Engineering
Purdue University
Lafayette, Indiana

Supported by
National Aeronautics and Space Administration
Grant NGL 15-005-021
# TABLE OF CONTENTS

## I. INTRODUCTION ................................................................. 1

## II. A GENERAL DISCUSSION OF THE BASIC ALGORITHMS ......................... 6
   II-1. Formulating the Signal-Flow Graph (SFG) ................................ 6
      a. Data Required ................................................. 6
      b. Finding a Tree .............................................. 7
      c. Rules for Formulating the Compact SFG ....................... 8
      d. The Gain Formula for "Closed" SFG ......................... 11
   II-2. Manipulating SFG Branch Weights .................................... 12
   II-3. Generating First Order Loops ...................................... 16
      a. General Description ......................................... 16
      b. A detailed Description of the Path-Finding Algorithm ........ 18
   II-4. Generating Nontouching Loops of Order Two or More ................... 21

## III. USER'S GUIDE ............................................................... 28
   III-1. Information Needed by User ....................................... 28
      Appendix A (A Sorting Technique for Handling Multi-
      Input, Multi-Output Networks) .................................. 36
      Appendix B (A Brief List of Limitations on the
      Size and Type of Network Allowed) ............................ 39
      Appendix C (Selecting a "Good" Tree) ........................... 40
   III-2. Modifying the Dimension of Arrays ................................ 42

## IV. PROGRAMMER'S GUIDE .......................................................... 47
   IV-1. Definitions ........................................................ 47
   IV-2. Flow Charts ...................................................... 50
   IV-3. Program Listing .................................................. 100

## REFERENCES ............................................................................ 124
I. INTRODUCTION

The majority of computer aided circuit analysis programs belong to the class of "numerical programs", that is, the output is some numerical value. At the time of our research a few programs, most notably ANPl(8)*, NASAP(9), and CORNAP(10) could generate network functions as rational functions of s but did not allow the value of any element to be left in symbol form. The research project reported here represents, we believe, the first effort to generate symbolic network functions. By a symbolic network function we mean \( \frac{V_{\text{out}}}{V_{\text{in}}}, \frac{I_{\text{out}}}{I_{\text{in}}}, \text{or} \frac{1_{\text{out}}}{1_{\text{in}}} \) as a ratio of two polynomials of one of the following types:

1. All network element values are represented by symbols (the symbols need not all be different)

   Examples: 
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2 LRC}{s^2 2LRC + s(L+R^2 C) + R}
   \]
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{Z Y R}{2Z Y R + Z + R^2 Y + R}
   \]

2. Some element values are specified numerically, some symbolically,

   Example: 
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2 R}{s^2 2R + s(.5x10^6+150R^2) + .75x10^8 R}
   \]
   or

3. All element values are given numerically,

   Example: 
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2}{2s^2 + 2x10^4 s + .75x10^8}
   \]

There are many reasons why one may be interested in totally or partially symbolic network functions. The following presents a few of the more important ones.

1. Insight. To illustrate the added "insight" symbolic programs can provide
in comparison to numerical type programs, suppose we have been asked to verify that the network in Fig. 1 is a negative impedance converter for large $\beta$, i.e.,

$$Z_{in}(s) \rightarrow -Z_L(s) \text{ as } \beta \rightarrow \infty$$

(1)

Figure 1

To verify (1) with some degree of certainty using a numerical program would require evaluating $Z_{in}(s)$ (and $Z_L(s)$) for many different values of $\beta$ and frequency $\omega$, a time consuming process at best since most programs must completely re-evaluate the network response for every relatively large change in parameter values. Furthermore, the resulting verification would only be valid for the particular structure and component values chosen for $Z_L(s)$. With a symbolic program, one computer run gives the symbolic transfer function

$$Z_{in}(s) = Z_L(s) \left( \frac{2 + \beta}{2 - \beta} \right)$$

from which (1) follows immediately.

(2) Error Control. To demonstrate how a symbolic program can be used to effectively control round-off error, consider the differential amplifier\(^{(1)}\)
shown in Figures 1 and 2 of Section II-1. If network branches 1, 2, 5, and 11 are chosen as the tree for deriving the signal-flow graph (SFG), then the set of nontouching loops (all orders with sign attached) which belong to the numerator of the low frequency transfer function

\[ \frac{I_{\text{out}}}{V_{\text{in}}} \bigg|_{s=0} = \frac{N}{\Delta} \]

is given by

\[ \left\{ \frac{(R_1 R_3 A_2)}{(R E)^2 R_2}, \frac{-(R_1 R_3 A_1)}{(R E)^2 R_2}, \frac{R_1 R_3 A_1}{(R E)^2 R_2}, \frac{-(R_1 A_1)}{R E R_2}, \frac{-(R_3 A_2)}{(R E)^2 R_2}, \frac{R_1 A_2}{R E R_2} \right\} \]

and the corresponding set for the denominator is given by

\[ \left\{ \frac{R_1}{R_2}, \frac{R_1}{R_2}, \frac{R_3}{R_2}, \frac{R_1 R_3}{R_2 R_2}, \frac{R_1 R_3}{R_2 R_2}, \frac{R_1 R_3}{R_2 R_2}, \frac{R_1 R_3}{R_2 R_2} \right\} \]

Letting \( A_2 = A_1, R_1 = 5K, R_2 = 15K, R_3 = 10K, \) and \( R E = 25, \) evaluate \( N \) and \( \Delta \) by summing the terms in the order given in the above sets keeping each number generated to 8 significant digits. Then

\[
N = A_1 \left[ 5.3333333-5.3333333+5.3333333-.013333333-5.3333333+0.13333333 \right] \\
= 3.3 \times 10^{-8} A_1
\]

and \( \Delta = 1335 \)

Thus

\[
\frac{I_{\text{out}}}{V_{\text{in}}} \bigg|_{s=0} = \frac{3.3 \times 10^{-8}}{1335} A_1
\]

which is incorrect since \( N = 0. \) Although the above transfer function was derived using SFG theory, round-off errors which cause erroneous results can occur in any computer program restricted to numerical evaluation, and are generally very difficult to predict or control. Because round-off error enhancement in the evaluation of network functions often occurs as a result of widely separated values of some of the network elements, one method of error control would be to leave such element values in
symbolic form. This technique can be applied to the above example by noting that \( R_E \) should be kept as a symbol since its value is considerably less than the other resistance values. Thus, keeping \( R_E \) as a symbol and re-evaluating \( N \) gives

\[
N = A_1 \left[ 3333.3333 \left( \frac{R_E}{R_E} \right)^2 - \frac{3333.3333}{(R_E)^2} + \frac{3333.3333}{(R_E)^2} - \frac{33333333}{R_E} - \frac{3333.3333}{(R_E)^2} + \frac{33333333}{R_E} \right] = 0
\]

That is,

\[
\left. \frac{I_{out}}{V_{in}} \right|_{s=0} = 0
\]

(3) Sensitivity Analysis. Sensitivity of the network function to changes in a particular network parameter can be found using a symbolic program by keeping this parameter as a symbol and then performing the required differentiation. Although there exist powerful numerical techniques for sensitivity analysis, the above procedure using a symbolic program has the particularly desirable feature of being less susceptible to round-off errors.

(4) Parameter Variation. Suppose we wanted to evaluate the network function for many different values of one or more network parameters. Using a symbolic program, we could leave these parameters in symbol form and then efficiently and accurately perform the large number of required evaluations on the resulting symbolic network function. On the other hand numerical programs now available must re-derive the transfer function for every relatively large parameter change.

(5) Iterative Piecewise Linear Analysis of Resistive Nonlinear Networks. (11)

Part of this powerful analysis technique requires the solution of a
resistive linear network where some resistances and some d-c sources are kept in symbol form.

The primary objective* of this project has been the development of some new or improved concepts needed to make a symbolic network analysis program efficient with respect to program storage and execution time. The project culminated in the program SNAP (Symbolic Network Analysis Program) which finds symbolic network functions for networks containing R, L, and C type elements and all four types of controlled sources. SNAP contains the following unique features:

(1) The extensive use of a path-finding algorithm in place of matrix operations,
(2) Efficient techniques for finding all loops of the SFG and for enumerating all higher order loops,
(3) The use of the "compact signal-flow graph" instead of the "primitive signal-flow graph", and
(4) A simple coding technique which is used
   (a) manipulate symbols thereby allowing the complete program to be written in Fortran (another important aspect of the coding scheme is that it permits repeated symbols to be treated as one symbol), and
   (b) determine whether or not loop sets touch in the algorithm for enumerating higher order loops.

New techniques for handling multi-inputs and multi-outputs are also presented in this report although they have not yet been incorporated into the program SNAP.

* At about the same time the results of this project were disseminated, (3) another symbolic program (by coincidence also called SNAP) whose primary concern is the on-line use for design purposes made its appearance.
II. A GENERAL DISCUSSION OF THE BASIC ALGORITHMS

II-1. Formulating the Signal-Flow Graph (SFG)

a. Data Required

A SFG is generated by SNAP (Symbolic Network Analysis Program) from data specifying the topological structure of the network, the input-output variables, and the characteristics of each network branch. The network topology is described by

(a) A unique number for each branch, and

(b) the initial and terminal nodes of each branch as determined by the assigned current direction.

The input to the network must be a single independent voltage or current source and the output requested must be the voltage or current associated with a network branch or the voltage between any two nodes of the network. Finally, each branch is characterized by

(a) a symbol which specifies its type, i.e.,

   passive branches: R, G, L, C, Y, Z  
   control sources: VV, VC, CV, CC  
   independent sources: E, I

(b) a symbol representing the branch name together with the branch value if specified, and

(c) the branch number of the control (for dependent sources only)

As an example, consider the network given below (from a paper by A. DeMari\(^{(1)}\)).

---

*Refer to Appendix A of section III-1 for a discussion on how to handle multi-inputs and multi-outputs.*
Table 1 (Network Data)

<table>
<thead>
<tr>
<th>Branch Type</th>
<th>Branch Number</th>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Symbol</th>
<th>Value</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>R1</td>
<td>$5 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>R1</td>
<td>$5 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>R2</td>
<td>$15 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>A1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>A2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>RE</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>RE</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>R3</td>
<td>$10 \times 10^3$</td>
<td></td>
</tr>
</tbody>
</table>

b. Finding a Tree

The formulation of a SFG starts with the choice of a network tree. The selection of network branches to be used in the tree is made as follows: Independent voltage sources and controlled voltage sources are the first ones to be used. Then come the passive RLC elements in any order. In choosing the $(J+1)$th branch, the undirected graph formed by the $J$ branches already selected is tested to determine whether a path exists between the two terminal nodes of the $(J+1)$th branch. If so, the branch under consideration is disqualified. If not, the $(J+1)$th branch is added to the tree. Let $n$ be the number of nodes.
of the network graph. When \( n-1 \) branches are successfully chosen by the above process, we have obtained a tree.

As an example, consider the network of Fig. 2. If, following the selection of the voltage source, the passive branches are examined in the order by which they are listed, the tree shown in Fig. 3 results

![Diagram of network graph with labeled branches and links]

Tree branches:
- \( b_1 = 1 \), \( b_2 = 2 \), \( b_3 = 5 \), \( b_4 = 9 \)

Links:
- \( l_1 = 3 \), \( l_2 = 4 \), \( l_3 = 6 \), \( l_4 = 7 \)
- \( l_5 = 8 \), \( l_6 = 10 \), \( l_7 = 11 \)

Figure 3

It is important to note that the complexity of the SPG and consequently the time required to evaluate the transfer function depends on the tree selected. (2) A brief summary of the rules for choosing a "good" tree is given in Appendix C at the end of Section III-1.

c. Rules for Formulating the Compact SFG

A "compact SFG" is a signal-flow graph whose node variables consist only of tree branch voltages and link currents except when additional nodes are needed for control sources or for the output variable. This type of SFG can be more efficiently evaluated than the so-called primitive SFG which contains one node for the branch voltage and another for the branch current.

The compact SFG is constructed according to the following rules: (An example as derived from Fig. 2, Fig. 3, and Table 1 is given in Fig. 4).

Rule (1): For each link \( l_k \), the unique fundamental circuit \( C_k \) containing
Figure 4: SFG

Table 2 (SFG DATA)

<table>
<thead>
<tr>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Exponent of s</th>
<th>Branch Value</th>
<th>Branch Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>FB</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-5x10^3</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-5x10^3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>-5x10^3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>-1/15x10^3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5x10^3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1/15x10^3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>-5x10^3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>A1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>5x10^3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>0</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>A2</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>0</td>
<td>5x10^3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>0</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>0</td>
<td>-25</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>0</td>
<td>-10^-4</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>0</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0</td>
<td>10^-4</td>
<td></td>
</tr>
</tbody>
</table>
branches $b_i, i=1,2,...,m$ is found. Two sets of SFG branches can then be created.

Set (a): For each passive branch in the tree branch set $b_i, i=1,2,...,m$, a directed branch in the SFG is formed from node $V_{b_i}$ to node $I_k$, with weight equal to the impedance of branch $b_i$, prefixed with the proper sign (positive, if the directions of $b_i$ and $b_i$ concur in $C_k$, and negative otherwise).

Set (b): If the link $l_k$ is a passive branch, a directed branch in the SFG is formed from each node $V_{b_i}, i=1,2,...,m$, to node $I_k$, having weight equal to the admittance of link $l_k$, prefixed with the proper sign (negative, if the directions of $l_k$ and $b_i$ concur in $C_k$, positive otherwise).

Rule (2): If any of the four types of controlled sources are present, a directed branch is created in the SFG from the controlling variable to the controlled source, having weight equal to the constant of proportionality ($g_m$, beta, etc.). If the controlling variable is a link voltage or a tree branch current, one more node is added to the SFG to represent this controlling variable $X$ (node $X_{12}$ in Fig. 4 is a node of this type). $X$ is then expressed in terms of the tree branch voltage or link current through a simple immittance relationship.

Rule (3): If the desired output $Y$ is neither a tree branch voltage nor a link current, then one node is added to the SFG to represent $Y$. $Y$ is then expressed in terms of tree branch voltages or in terms of a link current through a simple immittance relationship.

Rule (4): Finally, the SFG is "closed" by adding a branch with a symbolic weight (FB), directed from the output to the input node.
d. The Gain Formula for "Closed" SFG

The purpose for introducing the closed SFG is because only all orders of non-touching loops need be found as opposed to the evaluation of Mason's formula which required enumerating certain paths as well as loops.

To derive the gain expression for the closed SFG consider first Mason's equation for the transfer function.

\[ T = \frac{\sum_{i=1}^{m} P_i \Delta_i}{\Delta} \]

where

\[ \Delta = 1 + \sum_{j=1}^{\infty} (-1)^j L_{k,j} \]

\[ L_{k,j} \] is the product of the transmittances of the \( k \)th set of non-intersecting loops of order \( j \).

\[ P_i \] is the transmittance product of the \( i \)th path between \( X_i \) and \( X_0 \).

\[ \Delta_i \] is the partial determinant obtained from \( \Delta \) after removal of all loops intersecting the \( i \)th path between \( X_i \) and \( X_0 \).

Let \( \Delta_c \) be the determinant of the closed SFG. It is then noted that since \( \{P_i\}_{i=1}^{m} \) is the set of all paths from \( X_i \) to \( X_0 \), the loops present in the closed SFG not present in the original SFG will be precisely \( \{(FB)P_i\}_{i=1}^{m} \) where \( FB \) is the symbol assigned to the added branch. Further, since the path \( FB \) contains only nodes \( X_i \) and \( X_0 \) which, in turn, are present in every path \( P_i, i=1,2,\ldots,m \), it follows that the non-intersecting loop combinations that do not touch the loops \( (FB)P_i, i=1,2,\ldots,m \) will be precisely those combinations which do not touch the path \( P_i, i=1,2,\ldots,m \). It follows that

\[ \Delta_c = (FB) \sum_{i=1}^{m} P_i \Delta_i + \Delta \]

Thus, the transfer function can be found by simply sorting the terms of the determinant of the closed SFG.
II-2. Manipulating SFG Branch Weights

Each branch weight in the SFG is of the form

\[ \text{Constant} \cdot \text{Symbol} \cdot s^n \]

If an arbitrary branch has an initial node \( X_i \) and a final node \( X_f \), then the three parameters

\[ C(X_i, X_f) = \text{constant} \]
\[ S(X_i, X_f) = \text{symbol} \]
\[ E(X_i, X_f) = \text{exponent of } s \]

completely define the weight of the branch. After a loop or a set of nontouching loops has been found in the SFG, say by some path-finding technique, it is desirable to combine the weight parameters of each branch in the loop set to form a composite loop set weight. The loop set constant may be easily formed by taking the product of the constants associated with each branch. Similarly, the loop set exponent parameter is readily found by summing the exponents assigned to each branch. However, because computers are not particularly adept at symbol manipulation, it is inefficient with respect to both time and storage to form directly a composite loop set symbol. A much better technique is to convert each branch symbol into a numeric code. These codes are assigned to the SFG branches as follows: Each distinct symbol in the SFG is stored in the array \( S(j) \) and assigned a code \( B^j \) where \( B \) is some base \( B \in \{2, 4, \ldots, 2^m\} \). Now for an arbitrary SFG branch having initial node \( X_i \) and final node \( X_f \) which contains the symbol \( S(n) \), the code

\[ K(X_i, X_f) = B^n \]

is assigned.
The real value of this coding technique stems from the fact that the composite loop set code formed by summing the codes representing the individual branch symbols can be uniquely decoded provided the number of identical symbols combined into any code is less than B.

As an example of the above concepts for manipulating the SFG branch weights, refer to the SFG shown in Fig. 4. Consider, in particular, the loop defined by the node sequences

\( V_2 - I_3 - V_2 \) and \( V_4 - I_5 - V_4 \)

Then

\[
\text{composite loop set constant} = (-5 \times 10^3)(1)(-5 \times 10^3)(1) = 25 \times 10^6
\]

and

\[
\text{composite loop set power} = 0 + 1 + 0 + 1 = 2
\]

To find the loop set code, an array of distinct symbols of the SFG and their corresponding codes must be set up.

- no symbol \( \leftrightarrow 0 \)
- \( S(1) = F_B \leftrightarrow 4^0 \)
- \( S(2) = C \leftrightarrow 4^1 \)
- \( S(3) = A_1 \leftrightarrow 4^2 \)
- \( S(4) = A_2 \leftrightarrow 4^3 \)

Note that because there will be at most two identical symbols in any code, the base 4 was chosen. Using the above codes gives

\[
\text{composite set code} = K(V_2, I_3) + K(I_3, V_2)
\]

\[
+ K(V_4, I_5) + K(I_5, V_4)
\]

\[
= 4^2 + 0 + 0 + 4^2
\]

\[
= 8
\]

Now to decode this number, say in the output, we would write

\[
8 = 2(4^1)
\]

\[
= (S(2))^2
\]

\[
= C^2
\]
which is indeed the symbol associated with the loop impedance product. The above coding scheme for manipulating symbols is easily adapted to the computer by incorporating the masking operation \texttt{AND}. To determine the number of S(1) type symbols contained in a given code, the \texttt{AND} operation is applied to the code and B-1. In general, the number of S(J) symbols is found by dividing (using integer division so as to truncate the remainder) the code used to determine the number of S(J-1) symbols by B and then applying the \texttt{AND} operation. For example, consider the loop set previously discussed.

\begin{verbatim}
loop set code = \texttt{8} = (00000000100)\texttt{2}
B -1 = 3 = (00000000011)\texttt{2}
\text{(loop set code).AND.}(B-1) = \texttt{8}.AND.3
\end{verbatim}

\begin{verbatim}
= 0
\end{verbatim}

Thus, the symbol S(1) = FB is not present. Now divide the loop code by B and repeat the above procedure

\begin{verbatim}
new code = \frac{\texttt{8}}{\texttt{4}} = 2
\end{verbatim}

\begin{verbatim}
(new code).AND.(B-1) = \texttt{2}.AND.3
\end{verbatim}

\begin{verbatim}
= (00000000010)\texttt{2}
\end{verbatim}

\begin{verbatim}
= 2
\end{verbatim}

This implies C is contained in the code 2 and that its exponent is 2, i.e. \(C^2\). The process stops when the code is reduced to zero.

Each loop set (of any order) contributes to a term in the network function. As each loop set (of any order) is generated and coded, it is compared with existing terms. If a term with the same symbol code and power of 5 exists, then the constant of the term is updated by adding to it the constant of the new loop set. Otherwise, a new term is created. Note that the above process is an important step towards reducing the storage requirements.
After all loop sets have been found, the transfer function is complete, and it remains only to transform the symbol code of each term into its corresponding symbol set by the .AND. operation previously described.
II-3. Generating First Order Loops

a. General Description

Let the nodes of the SFG be labelled 1, 2, ..., N. All first order loops which contain node J (J=1 initially) can be found by conceptually splitting node J into two nodes, one node containing all incoming branch and the other containing all outgoing branches, and then enumerating all paths between these two nodes. All branches going into node J are then removed and the process repeated for node J+1. Clearly, this procedure will produce all circuits with no duplications.

The problem of efficiently finding all circuits now becomes one of finding paths. The path-finding algorithm utilized by SNAP is based on a routing technique which conceptually resembles that proposed by Kroft\(^{(5)}\). However, because our ultimate objective is a flexible user-oriented program, we have chosen to use FORTRAN instead of SNOBOL as Kroft did. A general description of the concepts contained in the algorithm will be given here in addition to a rigorous step-by-step description presented at the end of this section.

Consider the SFG given in Fig. 4. The topological structure of the SFG can be completely described by the following routing table where the entries in the J\(^{th}\) row are the set of all nodes of distance one from node J. Note that the entries of each row are made to decrease as the column subscript M increases. This facilitates modifying the table after all paths through a particular node, say node J, have been found because only the right most non-zero entry of each row must be tested, i.e. if that entry equals J, it is a set to zero. As an example in using the routing table, the following two circuits can easily be shown to compose the complete set of circuits containing node 1.

1 - 11 - 9 - 12 - 8 - 5 - 6 - 1
1 - 11 - 9 - 10 - 7 - 2 - 6 - 1
Routing Table

A particularly important feature of the path-finding algorithm is the method by which each new node generated from the routing table must be tested to prevent loops from being formed. Rather than comparing the prospective node to each node already in the path, it is much more efficient to define the function

\[
F(I) = \begin{cases} 
1 & \text{if } I \text{ is contained in the path node sequence} \\
0 & \text{if } I \text{ is not contained in the path node sequence} 
\end{cases}
\]

on which only one logic test need be made.

Additional insight may be obtained by viewing the path-finding technique graphically. That is, the process by which paths are generated can be observed by applying the following two rules directly to the SFG.

(1) Let node J be the last node added to the path node sequence (initially J = input node). To select the next node, traverse that branch connected to node J that goes to the highest numbered node satisfying both the following requirements:
We did not just back up from this node while applying rule 2, and

b. this node is not included in the path node sequence.

Repeat this process until the output node is reached (then store the node sequence and go to rule 2) or until no new node can be found that satisfies (a) and (b) (then go to rule 2).

(2) Back up along the path just found (this is always possible unless we are at the input node in which case all paths have been found) until a new route can be taken according to rule 1.

![Diagram](image)

**Figure 5**

The heavy lines of Fig. 5 show the path which results from applying rule 1 when node 1 is considered both the initial and terminal node. Generating a second path requires backtracking to node 9, then continuing the sequence 10-7-2-6-1. Note that the above graphical technique for listing all paths can be helpful when solving problems by hand.

b. A Detailed Description of the Path-Finding Algorithm

Algorithm PF* (Path-finding): This algorithm finds all paths between two nodes of a directed graph (without parallel edges) whose nodes are labelled 1, 2, ..., N. The only modification necessary to adapt the algorithm to finding

*The format used to describe the path-finding algorithm follows the style of Knuth(6).
all circuits thru node L is to set I ← L where L and I are defined below.

Notations:

I: Initial path node
L: Last path node
N: Number of nodes in graph
E_J: Number of branches leaving node J
R(J,M): Routing table
C_J: Column counter for the J^{th} row of the table R
P(V,W): The V^{th} node in the node sequence of path W
W: Number of nodes in path W
F(K): A function used to test whether node K is repeated, and whether the last node is reached.

PF1. (Preliminary)

Set R(J,1), R(J,2),...,R(J,E_J) to the group of E_J nodes of distance one from node J. When using the algorithm to find circuit, made the entries of each row decrease as M increases.

Set R(J,M) ← \{ -1 for M = E_J+1 and J=I
0 for M = E_J+1 and J≠I

Set F(K) ← \{ 1 for K=I
0 for K=J and J≠I,L
-1 for K=L

Set C_J ← 1 for J=1,2,...,N
Set W ← 1, V ← 2, J ← I, P(1,1) ← I

PF2. (Find the next node)

Set P(V,W) ← R(J,C_J)

PF3. (Test R)

\{ < 0 stop; all paths have been found
IF R(J,C_J) = 0 set F(J) ← 0, go to step PF6
> 0 go to step PF4
PF4. (Test F)

\[
\begin{cases}
< 0 & \text{path completed; go to step PF7} \\
0 & \text{go to step PF5} \\
> 0 & \text{set } C_J \leftarrow C_J + 1; \text{ go to step PF2}
\end{cases}
\]

PF5. (Prepare for next node)

Set \( J \leftarrow P(V,W), F(J) \leftarrow 1, V \leftarrow V+1 \), go to step PF2

PF6. (Back step)

Set \( C_J \leftarrow 1, J \leftarrow P(V-2,W), C_J \leftarrow C_J + 1, V \leftarrow V-1 \), go to step PF2

PF7. (Finish path)

Set \( C_J \leftarrow C_J + 1, P(K,W+1) \leftarrow P(K,W), K=1,2,...,U_W-1, W \leftarrow W+1 \), go to step PF2.
II-4. Generating Nontouching Loops of Order Two or More

Preliminary results from SNAP indicate that of the following subprograms, (1) finding a SFG, (2) coding and de-coding, (3) enumerating first order loops, and (4) finding all higher order nontouching loops, the last will generally require the most time unless the network contains many distinct symbols in which case subprogram (2) may dominate. It is therefore necessary to exercise considerable care in developing an algorithm for finding all orders of nontouching loops.

In general, to find loop sets of all orders, some comparison between the node sequences of the different loops must be made. A brute force technique is simply to store all the node sequences of the first order loops and to find nontouching loops by direct comparison of the nodes contained in the loops. Of course, storage is also needed to indicate the loops contained in some of the higher order combinations, but this storage is necessary even in the more efficient techniques which follow.

The above method is improved considerably if instead of directly comparing the nodes of loop A and loop B to determine if they touch, a function $F(I)$ is defined as

$$F(I) = \begin{cases} 1 & \text{if } I \in \{\text{nodes in loop A}\} \\ 0 & \text{otherwise} \end{cases}$$

and then tested as follows:

$$\text{If } F(J) \neq 0 \text{ all } J \in \{\text{nodes in loop B}\} \Rightarrow \text{loops do not touch}$$

$$\text{If } F(J) = 1 \text{ any } J \in \{\text{nodes in loop B}\} \Rightarrow \text{loops touch}$$

For those computers which can accommodate the .AND. operation (or equivalent), the following coding technique reduces the number of logic

*Although the program was correct, the algorithm was incorrectly described in reference (3).
comparisons needed to determine if two loops touch to one and, perhaps what is even more important, requires only a single code be stored for each first order loop instead of the complete node sequence. As each first order loop is generated, it is assigned an integer code whose binary representation shows the set of nodes in the loop. For example, if loop A contains the nodes \{11, 9, 12, 8, 5, 6, 1\} and loop B contains the nodes \{2, 6\}, then the codes are

\[
A = (110110110001)_2 = 3505 \\
B = (00000100010)_2 = 34
\]

To determine whether the two loops touch or not, the masking operation .AND. is used. Thus,

\[
(A) \text{ .AND. (B)} = (000000100000)_2 \neq 0
\]

The result is not zero, indicating that loops A and B touch.

Using the coding scheme the complete set of nontouching pairs of loops is found and stored in the one dimension array N. Let \(n\) = number of first order of loops. Then

\[
[N(1), N(2), \ldots, N[P(1)], N[P(1)+1], \ldots, N[P(2)], N[P(2)+1], \ldots, N[P(3)], \\
N[P(3)+1], \ldots, N[P(n -1)], N[P(n -1)+1], \ldots, N[P(n )]]
\]

is the complete set of nontouching pairs of loops, where the

\[
[N(1), N(2), \ldots, N[P(1)] = \text{set of loops numbered higher than 1 which do not touch loop 1.} \\
[N[P(1)+1], N[P(1)+2], \ldots, N[P(2)] = \text{set of loops numbered higher than 2 which do not touch loop 2.} \\
\vdots \\
[N[P(i-1)+1], N[P(i-1)+2], \ldots, N[P(i)] = \text{set of loops numbered higher than } i \text{ which do not touch loop } i \\
\vdots \\
[N[P(n -1)+1], N[P(n -1)+2], \ldots, N[P(n )] = \text{empty set because there are no loops numbered higher than } n
\]

Note that the array P is simply used to partition the array N such that the set

\[
[N[P(i-1)+1], N[P(i-1)+2], \ldots, N[P(1)]
\]

does not touch loop i.
Example: (Consider the SFG of Fig. 4)

The first order loops are

<table>
<thead>
<tr>
<th>loop</th>
<th>node sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-11-9-12-8-5-6-1</td>
</tr>
<tr>
<td>2</td>
<td>1-11-9-10-7-2-6-1</td>
</tr>
<tr>
<td>3</td>
<td>2-6-2</td>
</tr>
<tr>
<td>4</td>
<td>2-3-2</td>
</tr>
<tr>
<td>5</td>
<td>4-5-4</td>
</tr>
<tr>
<td>6</td>
<td>5-6-5</td>
</tr>
<tr>
<td>7</td>
<td>9-11-9</td>
</tr>
<tr>
<td>8</td>
<td>9-10-9</td>
</tr>
</tbody>
</table>

To find the array of nontouching pairs P, SNAP codes the above loops and proceeds to use the .AND. operator. The results are

\[ N = \{4,5,5,7,8,5,6,7,8,7,8\} \]

and \[ P(1)=1, P(2)=2, P(3)=5, P(4)=9, P(5)=11, P(6)=13, P(7)=13, P(8)=13. \]

To systematically continue the process, an array \( S \) is created from which all higher order loop sets (2 or more) not touching loop \( l \) can be found. By incrementing \( l \) from 1 to \( n \), all higher order loops will then be enumerated.

Let

\[
S(1,1)S(1,2)\ldots S[1,U(1)] \ldots S[1,J(1)],0,\ldots
\]

\[
S(2,1)S(2,2)\ldots S[2,U(2)+1]\ldots S[2,J(2)],0,\ldots
\]

\[
\vdots
\]

\[
S(K,1)S(K,2)\ldots S[K,U(K)] \ldots S[K,J(K)],0,\ldots
\]

where

\[
S(1,1) = N[P(L-1)+1]
\]

\[
S(1,2) = N[P(L-1)+2]
\]

\[
\vdots
\]

\[
S[1,J(1)] = N[P(L)]
\]
and where the entire (loop numbers) of row \(M (M \geq 2)\) are those loops in the set 

\[S[M-1, U(M-1)+1], S[M-1, U(M-1)+2], \ldots, S[M-1, J(M-1)]\]

which do not touch the loop \(S[M-1, U(M-1)]\).

The arrows shown in the array \(S\) given above are referred to as "pointers". Note that \(U(J)\) indicates the position of "pointer" of the \(J\)th row. Example: \(U(3) = 5\) means the pointer of row 3 is currently located at the 5th column.

The procedure for finding all higher order loop combinations is given in the following flow chart:

```
[Flowchart]
```

Preliminary
- \(n\): number of first order loops
- \(l\): first order loop under consideration
- \(U(i)\): pointer position for row \(i\) in \(S\)
- \(J(i)\): number of loops in row \(i\) of \(S\)
- \(K\): the row counter indicating that row \(K\) of \(S\) is being scanned to generate a set of \(K+2\) order loops

Set \(l = 0\)
- \(U(i) = 1\) \(i = 1, 2, \ldots, n\)
- \(J(i) = 0\) \(i = 1, 2, \ldots, n\)

\[\ldots\]

Set \(l = l + 1\)

Have all first order loops been used to generate higher-order combinations? i.e. Is \(l = n\)?

Yes \(\rightarrow\) Stop (all higher order loops have been found)

No

Insert into row 1 of \(S\) the numbers corresponding to the loops numbered higher than \(l\) which do not touch loop \(l\).

Set \(K = 1\)
- \(J(1)\): number of these loops

\[\leq 1\]

Test \(J(1)\)

\[> 1\]
Generate row $K+1$ of $S$ as follows: Insert those loops of the set $\{S[K,K+U(K)], \ldots, S[K,K+J(K)]\}$ that do not touch loop $S[K,K(U(K))]$ into row $K+1$ of $S$. Set $J(K+1) = \text{number of these loops}$. When a new element, say $S(K+1,X)$, is generated, the weight parameters corresponding to the symbol code, constant term, and power of $s$ are stored (or when possible combined with other similar type terms) for the loop set loop for the loop set loop

```latex
\text{loop} L \cdot \text{loop} S[1,U(1)] \cdot \ldots \cdot \text{loop} S(K+1,X)
```

\begin{itemize}
  \item TEST $J(K+1) > 1$ \quad $K \leftarrow K+1$
  \item \text{TEST} $J(K+1) \leq 1$
    \begin{itemize}
      \item Can we generate row $K+1$ by incrementing the pointer of row $K$? \ i.e. \ Is $U(K) < J(K)-1$?
      \begin{itemize}
        \item No
          \item Can we back up one row? \ i.e. \ Is $K > 1$
      \end{itemize}
      \begin{itemize}
        \item Yes
          \item $U(K) = 1$
        \item $J(K) = 0$
        \item $K = K - 1$
      \end{itemize}
    \end{itemize}
  \end{itemize}
Example:

From the preceding example,

\[ N = \{4, 5, 7, 8, 5, 6, 7, 8, 7, 8, 7, 8\} \]

and \[ P(1) = 1, P(2) = 2, P(3) = 5, P(4) = 9, P(5) = 11 \]
\[ P(6) = 13, P(7) = 13, P(8) = 13 \]

Arrays \( N \) & \( P \) are more easily interpreted by setting up the following table:

<table>
<thead>
<tr>
<th>loop J</th>
<th>loops numbered higher than J that do not touch loop J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5, 7, 8</td>
</tr>
<tr>
<td>4</td>
<td>5, 6, 7, 8</td>
</tr>
<tr>
<td>5</td>
<td>7, 8</td>
</tr>
<tr>
<td>6</td>
<td>7, 8</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
</tr>
</tbody>
</table>

The sequence for producing the higher order loops is as follows:

**loop 1**

loops not touching loop 1 are inserted into first row of \( S \) (see Table 3)

\[ S \text{ array} \]

\[
\begin{bmatrix}
4 & 0 & \ldots \\
0 & \ddots & \ddots \\
\end{bmatrix}
\]

Output

no 3\textsuperscript{rd} order loops

**loop 2**

\[ S \text{ array} \]

\[
\begin{bmatrix}
5 & 0 & \ldots \\
0 & \ddots & \ddots \\
\end{bmatrix}
\]

Output

no 3\textsuperscript{rd} order loops
loop 3

S array

<table>
<thead>
<tr>
<th>5</th>
<th>7</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output

loop 5 does not touch loop 7 or loop 8 (this is determined by comparing loop codes—see section II-4)

<table>
<thead>
<tr>
<th>5</th>
<th>7</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

loop 7 touches loop 8; thus, there is no 3rd row. Further if the pointer of row 1 is incremented by 1, no new 2nd row can be created. Thus, we are done with loop 3.

loop 4

S array

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output

loop 5 does not touch loop 7 or loop 8

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

loop 7 touches loop 8; thus, there is no 3rd row. Increment pointer of row 1.
loop 6 does not touch loop 7 or loop 8

\[
\begin{bmatrix}
5 & 6 & 7 & 8 \\
0 & 0 & 0 & 0 \\
\vdots & & \ddots & \\
7 & 8 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

loop 4*loop 6*loop 7
loop 4*loop 6*loop 8

incrementing pointers give no additional third order loops

loop 5

S array
\[
\begin{bmatrix}
7 & 8 & 0 & \ldots \\
0 & \vdots & & \\
\end{bmatrix}
\]
Output
no 3rd order loops

loop 7 touches loop 8

loop 6

S array
\[
\begin{bmatrix}
7 & 8 & 0 & 0 \\
0 & \vdots & & \\
\end{bmatrix}
\]
Output
no 3rd order loops

loop 7 touches loop 8

loop 7

S array
[0]
Output
no 3rd order loops

loop 8

S array
[0]
Output
no 3rd order loops
III. USER'S GUIDE

III-1. Information Needed by User

Program: SNAP (Symbolic Network Analysis Program)
Purpose: To obtain the network functions* $\frac{V_{out}}{V_{in}}$, $\frac{V_{out}}{I_{in}}$, $\frac{I_{out}}{V_{in}}$, or $\frac{I_{out}}{I_{in}}$ as a ratio of two polynomials of the following type:

1. all network element values are represented by symbols (the symbols need not all be different),
   
   Examples:
   
   $$\frac{V_{out}}{V_{in}} = \frac{s^2LRC}{s^22LRC + s(L+R^2C) + R}$$
   $$\frac{V_{out}}{V_{in}} = \frac{Z\text{YR}}{2Z\text{YR} + Z + R^2Y + R}$$

2. some element values are specified numerically, some symbolically,

   Example:
   
   $$\frac{V_{out}}{V_{in}} = \frac{s^2R}{s^22R + s(0.5\times10^6 + 150R^2) + 0.75\times10^8R}$$

3. all element values are given numerically,

   Example:
   
   $$\frac{V_{out}}{V_{in}} = \frac{s^2}{2s^2 + 2\times10^4s + 0.75\times10^8}$$

Description: Program SNAP is designed to handle lumped, linear, time invariant networks** containing the following type components:

1. two-terminal circuit elements -- resistance, inductance, and capacitance.

2. two-terminal networks described by an admittance or impedance parameter.

---

* Refer to Appendix A at end of this section for a technique of handling multi-output functions.

** See Appendix B for a brief list of additional limitations on the size and type of network allowed.
(3) all four types of controlled sources (Note: Mutual inductance, ideal transformers, gyrators, etc., can be modeled with elements in (1) and (3))

(4) one independent source; see Appendix A for a technique of handling multi-input networks.

Network Data Required: After the network components have been modeled by the type elements allowed, the branches and nodes are to be numbered consecutively starting with 1 and reference directions for each branch current are to be chosen. The following gives the sequence of data cards needed to describe the network.

**CARD 1**

Columns | Contents
--- | ---
1-72 | Title card (all 72 columns are reproduced in output)

**CARD 2**

Columns | Contents
--- | ---
1-5 (adjusted) | Number of nodes in the network
6-10 (adjusted) | Number of branches in the network

The following three entries are optional.

11-15 (adjusted) | Number base of symbol codes (automatically set to 8 if left blank)
21 | 1 if a description of the SFG is to be listed, blank otherwise
22 | 1 if all loops (circuits) in the SFG are to be listed (node sequence), blank otherwise
CARD 3

Contents

1-5 right (adjusted)  Network branch number of source

6-10 right (adjusted)  Network branch number associated with output (leave blank if output is a voltage across more than one branch)

11-15 right (adjusted)  Node number corresponding to the positive output voltage terminal (these columns can be left blank if columns 6-10 are not blank)

16-20 right (adjusted)  Node number corresponding to the negative output voltage terminal (these columns can be left blank if columns 6-10 are not blank)

CARDS 4 thru (b+3)

(b = number of network branches)

Note 1: Each card describes one network branch (element).

Note 2: If output is a voltage (current) associated with a particular branch, then the data card describing this branch should be entered first (last) among the branch data cards (cards 4 thru (b+3)) to insure that this branch will be chosen as part of the tree (cotree).

Note 3: When a large number of branches share one common terminal, it is better to place these branches first starting with card 4 (card 5 if note 2 applies). The reason is given in Appendix C at the end of this section.

Columns

Element number--all elements of the network must be assigned a distinct number (positive integer). For greatest efficiency, the numbering should be consecutive.

Initial node--this is relative to the arbitrarily chosen current direction.

Terminal node--this is relative to the arbitrarily chosen current direction.

Element symbol--the element's value, if not specified, is represented by this symbol.

Equal sign (=) if element is to be assigned a value. Leave blank if element value is to be represented in symbolic form.

Element value (if known)--Format is E12.5. Units should be compatible with element type as specified in columns 1-2; for example, R is expressed in ohms, G in mhos.

If element is a dependent source, enter the element number of its control.
An Example: We wish to find $\frac{I_{out}}{V_{in}}$, keeping $A_1$, $A_2$, and $C$ as symbols.

FIGURE 1. ORIGINAL NETWORK.

FIGURE 2. MODELED NETWORK.
TABLE 1. Input Data as Reproduced in Program Output.

---

**DIFFERENTIAL AMPLIFIER**

---

<table>
<thead>
<tr>
<th>NUMBER OF NODES</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF BRANCHES</td>
<td>11</td>
</tr>
<tr>
<td>ELEMENT NUMBER OF SOURCE</td>
<td>1</td>
</tr>
<tr>
<td>ELEMENT NUMBER ASSOCIATED WITH OUTPUT</td>
<td>6</td>
</tr>
<tr>
<td>BASE FOR SYMBOL CODES</td>
<td>8</td>
</tr>
</tbody>
</table>

---

**NETWORK**

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>TYPE</th>
<th>NUMBER</th>
<th>NODE</th>
<th>INITIAL NODE</th>
<th>TERMINAL NODE</th>
<th>SYMBOL</th>
<th>VALUE OF CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E 1</td>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-0.</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 2</td>
<td>R</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-0.</td>
<td></td>
</tr>
<tr>
<td>C 3</td>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>C -0.</td>
<td></td>
</tr>
<tr>
<td>R 5</td>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R1=5.00000E+03</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td>C</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>C -0.</td>
<td></td>
</tr>
<tr>
<td>R 6</td>
<td>R</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>R2=1.50000E+04</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>CC 7</td>
<td>CC</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>A1 -0.</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CC 8</td>
<td>CC</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>A2 -0.</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>R 9</td>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>RE=2.50000E+01</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 10</td>
<td>R</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>RE=2.50000E+01</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 11</td>
<td>R</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>R3=1.00000E+04</td>
<td>-0</td>
<td></td>
</tr>
</tbody>
</table>

---

**TREE SELECTED**

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>TYPE</th>
<th>NUMBER</th>
<th>NODE</th>
<th>INITIAL NODE</th>
<th>TERMINAL NODE</th>
<th>SYMBOL</th>
<th>VALUE OF CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E 1</td>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-0.</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 2</td>
<td>R</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>R1=5.00000E+03</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 5</td>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R1=5.00000E+03</td>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>R 9</td>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>RE=2.50000E+01</td>
<td>-0</td>
<td></td>
</tr>
</tbody>
</table>
TABLE 2. Program Output Information Showing Signal-Flow Graph, Circuits, and Execution Times.

**SFG**

<table>
<thead>
<tr>
<th>INITIAL TERMINAL EXPONENT NODE</th>
<th>TERMINAL EXPONENT NODE</th>
<th>BRANCH OF S VALUE</th>
<th>BRANCH SYMBOL</th>
<th>1 IF SYMBOL IS INVERTED</th>
<th>1 IF SYMBOL IS USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>-1.000000E+00</td>
<td>R₈</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.000000E+00</td>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-5.000000E+03</td>
<td>R₈</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1.000000E+00</td>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-5.000000E+03</td>
<td>R₈</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>-6.666667E-05</td>
<td>R₂</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5.000000E+03</td>
<td>R₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6.666667E-05</td>
<td>R₂</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>-5.000000E+03</td>
<td>R₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>1.000000E+00</td>
<td>A₁</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>5.000000E+03</td>
<td>R₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>4.000000E-02</td>
<td>Rₑ</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>1.000000E+00</td>
<td>A₂</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>5.000000E+03</td>
<td>R₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>4.000000E-02</td>
<td>Rₑ</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>-2.500000E+01</td>
<td>Rₑ</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>-1.000000E-04</td>
<td>R₃</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>2.500000E+01</td>
<td>Rₑ</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>1.000000E-04</td>
<td>R₃</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TIME FOR FORMULATING SIGNAL FLOW GRAPH IN SECONDS** • 252

**CIRCUITS**

<table>
<thead>
<tr>
<th>NO.</th>
<th>NODE LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 11 9 12 8 5 6 1</td>
</tr>
<tr>
<td>2</td>
<td>1 11 9 10 7 2 6 1</td>
</tr>
<tr>
<td>3</td>
<td>2 6 2</td>
</tr>
<tr>
<td>4</td>
<td>2 3 2</td>
</tr>
<tr>
<td>5</td>
<td>4 5 4</td>
</tr>
<tr>
<td>6</td>
<td>5 6 5</td>
</tr>
<tr>
<td>7</td>
<td>9 11 9</td>
</tr>
<tr>
<td>8</td>
<td>9 10 9</td>
</tr>
</tbody>
</table>

**TIME FOR FINDING 8 FIRST ORDER LOOPS IN SECONDS** • 0.46

**TIME FOR FINDING 19 SETS OF NONTOUCHING LOOPS, IN SECONDS** • 0.27

**TIME FOR DECODING SYMBOLS IN SECONDS** • 0.124
### TABLE 3. Network Transfer Function and Total Execution Time.

#### NUMERATOR POLYNOMIAL

\[ = (3.3333 \times 10^{-5} + 1.6667 \times 10^{-5} s)(A_2 - A_1) \]

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>SYMBOL FOR GIVEN COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( A_2 ) / 1</td>
</tr>
<tr>
<td>2</td>
<td>( A_1 ) / 1</td>
</tr>
<tr>
<td>3</td>
<td>( C ) / ( A_2 ) / 1</td>
</tr>
<tr>
<td>4</td>
<td>( C ) / ( A_1 ) / 1</td>
</tr>
</tbody>
</table>

#### POWER OF S

<table>
<thead>
<tr>
<th>COLUMN 1</th>
<th>COLUMN 2</th>
<th>COLUMN 3</th>
<th>COLUMN 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.3333E-05</td>
<td>-3.3333E-05</td>
<td>0.</td>
</tr>
<tr>
<td>1</td>
<td>0.</td>
<td>0.</td>
<td>1.6667E-01</td>
</tr>
<tr>
<td>2</td>
<td>0.</td>
<td>0.</td>
<td>0.</td>
</tr>
</tbody>
</table>

#### CONSTANT COEFS. IN THE POLYNOMIAL

#### DENOMINATOR POLYNOMIAL

\[ = 3.3375 + 2.67 \times 10^4 s + 5.00625 \times 10^7 s^2 \]

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>SYMBOL FOR GIVEN COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 / 1</td>
</tr>
<tr>
<td>2</td>
<td>( C ) / 1</td>
</tr>
<tr>
<td>3</td>
<td>( C \times 2 ) / 1</td>
</tr>
</tbody>
</table>

#### POWER OF S

<table>
<thead>
<tr>
<th>COLUMN 1</th>
<th>COLUMN 2</th>
<th>COLUMN 3</th>
<th>COLUMN 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.33750E+00</td>
<td>0.</td>
<td>0.</td>
</tr>
<tr>
<td>1</td>
<td>0.</td>
<td>2.67000E+04</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>0.</td>
<td>0.</td>
<td>5.00625E+07</td>
</tr>
</tbody>
</table>

#### EXECUTION TIME IN SECONDS

\[ .509 \]

#### AUGUST 1970 VERSION OF SNAP
APPENDIX A

A Sorting Technique for Handling Multi-Input, Multi-Output Networks.

Multi-Inputs

Program SNAP (the August 1970 revision) permits only one independent source branch. However, networks containing more than one source can easily be handled with the following technique. Let \( W_i \) \( i = 1, 2, \ldots, n \) represent a set of \( n \) independent sources, either voltage or current. Assign \( W_1 \) as the permitted independent source and make \( W_2, W_3, \ldots, W_n \) dependent sources which are dependent on \( W_1 \) with proportionality factors

\[
    k_2 = \frac{W_2}{W_1}, k_3 = \frac{W_3}{W_1}, \ldots, k_n = \frac{W_n}{W_1}
\]

respectively.

Only the numerator polynomial in the output will contain these parameters thus permitting the user to easily put the output function into the form

\[
    \frac{W_{\text{out}}}{W_1} = \frac{P_1 + P_2 k_2 + P_3 k_3 + \ldots + P_n k_n}{\Delta}
\]

where \( \Delta \) and \( P_i \) \( i = 1, 2, \ldots, n \) are polynomials. The output function can then be written

\[
    W_{\text{out}} = \frac{P_1 W_1 + P_2 W_2 + \ldots + P_n W_n}{\Delta}
\]

(1)

Although at present SNAP does not give the output function in the form of Eq. (1) directly, only a few program modifications are necessary to effect such a result. For example, the program could internally create a new input node, \( I_{\text{new}} \), of the SFG and then make each independent source, \( W_i \), dependent on \( I_{\text{new}} \) with weight \( P_i \) as shown in Fig. 1 below.
Multi-Outputs

The following technique can be used to obtain more than one output function in a single computer run: Augment the original network by appending one end of a series connection of dependent voltage sources to the given network such that

(a) to each branch current, \( I_j \), desired as an output, there corresponds a dependent voltage source which depends on \( I_j \) and has symbolic weight \( I_{oj} \),

and (b) to each voltage \( V_{AB} \) desired as an output, there corresponds a set of the dependent voltage sources each dependent upon a voltage across one of the branches in the path between nodes A and B and all having symbolic weight \( V_{OAB} \).

By specifying the output to be the voltage across the entire series connection of dependent voltage sources, outputs \( I_j \) and \( V_{AB} \) will be those output terms which contain \( I_{oj} \) and \( V_{OAB} \) respectively. Only a few modifications of the present version of SNAP would be necessary to have the program internally perform the network augmentation described above (at present, the user must do the augmenting).

As an example, Fig. 2 illustrates the network augmentation needed to find the voltage \( V_{14} \) and current \( I_5 \) for the given bridge network in one computer run.
\[ V_{47} = V_{014} (V_{12} + V_{24}) + I_{05} I_{5} \]
## Appendix B

### A Brief List of Limitations on the Size and Type of Network Allowed

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of network branches</td>
<td>35</td>
<td>SNAP cannot handle all networks having 35 branches or less. Other factors such as time considerations, SFG characteristics (number of higher order loops, for example), and number of network symbols to name a few can further limit the size of the network.</td>
<td></td>
</tr>
<tr>
<td>Maximum number of elements that can be represented by the same symbol</td>
<td>7</td>
<td>This number can be increased to $2^n - 1$ by increasing the symbol code base used to $2^n$, $n &gt; 3$, on the input data card 2.</td>
<td></td>
</tr>
<tr>
<td>Number of different powers of s</td>
<td>15</td>
<td>Sufficient for networks containing no more than 15 reactive elements.</td>
<td></td>
</tr>
<tr>
<td>Estimate of the maximum number of distinct network symbols permitted</td>
<td>12</td>
<td>This restriction results from the fact that SNAP can contain no more than 150 different symbol combinations in the output.</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C

Selecting a "Good" Tree

The network tree used to generate the SFG has a very significant effect on the number of loops and higher order loops present in the SFG. The loop enumeration and evaluation, in turn, often determines the time and storage needed by a computer to solve a given network. The ladder network of Figure 1 together with Table 1 illustrated the interrelationship between the tree selected, the number of loops (all orders), computer execution time, and computer storage.

![Figure 1](image)

Table 1

<table>
<thead>
<tr>
<th>Tree Branches</th>
<th>Number of loops</th>
<th>Number of higher order loops</th>
<th>Time required to find $V_o/I_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star tree: 1, 3, 5, 7, 9, 11, 13, 15, 17</td>
<td>17</td>
<td>2567</td>
<td>1.55 seconds</td>
</tr>
<tr>
<td>1, 2, 4, 7, 9, 11, 13, 15, 17</td>
<td>38</td>
<td>8096</td>
<td>3.93 seconds</td>
</tr>
<tr>
<td>1, 2, 4, 6, 9, 11, 13, 15, 17</td>
<td>117</td>
<td>19719</td>
<td>9.42 seconds</td>
</tr>
<tr>
<td>1, 2, 4, 6, 8, 11, 13, 15, 17</td>
<td>476</td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

(storage for first order exceeded)

Unfortunately, choosing the "best" tree, that is, a tree which will minimize the number of loop combinations of all orders is a very involved process. See reference 2 and 7 for a detailed discussion of this problem.
For most networks, however, a tree that will result in a reasonable amount of execution time and computer storage can be selected by applying one of the following rules (rule 2 results in a better tree than rule 1)

Rule 1: Select a tree in which as many branches as possible form a star, that is, the branches share a common node. Modify this tree, if necessary, to include any branch which has two or more branches in parallel with it.

Rule 2: Let $T_k$ be some tree (not necessarily the best) of the network graph. For each link $e_i$ of the graph, define $B_{e_i}$ as the number of tree branches which form a circuit with $e_i$. Then form the sum

$$S_{T_k} = \sum_{i=1}^{L} B_{e_i}$$

where $L = \text{number of links in the graph having } T_k$ as a tree.

Select that tree, say $T_j$, which satisfies the inequality

$$S_{T_j} < S_{T_k} \quad k = 1, 2, \ldots, N$$

where $N = \text{number of trees}$

The example given in section III-1 uses a tree, $T_j$, having $S_{T_j} = 11$.

The tree generated internally by program SNAP includes all voltage sources together with those passive branches read in first (starting with input data card 4) which complete the tree. Thus, to have SNAP select the tree that has been chosen by the user, it is necessary that the user's tree include all voltage sources and that all its passive branches be listed first starting with input data card 4.
III-2. **Modifying the Dimension of Arrays**

In order to make SNAP applicable to many different type networks, a flexible yet simple procedure is needed for modifying the dimension of the arrays. For example, storage requirements for networks containing many symbols will be determined by the number of symbols, symbol codes, etc., whereas the storage needed for networks having no symbols will be determined by the number of loops, nontouching loops of all orders, and related network characteristics. Because it is not possible to determine apriori reasonable bounds for all the network characteristics, error diagnostics have been built into the program to inform the user as to which arrays have been inadequately dimensioned. As a result, the technique for adjusting the array dimension, in SNAP can be outlined as follows:

1. Check that those network characteristics which can be determined before running the program are within the specified limits. These limits are listed following the dimension statements of the main program for convenient reference.

2. Run the program. If an array dimension is exceeded an error message will result which specifies the network characteristic involved. For example, if the SFG of a given network has an excessive number of circuits, the message "No. of circuits exceeds limit--increase dimensions containing NPAC" will result. The definition of NPAC (number of paths and circuits) are found immediately following the array dimensions in the main program. It is important to point out that a computer run may continue to completion even if the dimension of some arrays have been exceeded (an error message is still given, however). In this situation, the results cannot be considered reliable.
(3) Once it has been ascertained by (1) and (2) that dimension modifications are in order, refer to the next few pages to determine the arrays associated with the network characteristics of interest. Increase the dimension of all the arrays indicated by say 20% (several runs may be necessary to achieve adequate program dimensions). Then update the value of the parameter (NPAC, for example) corresponding to the network characteristic involved. This parameter is used throughout the program (as limits on DO loops etc.) thereby making it unnecessary to do any additional program modifications.

NBN = Number of Network Branches (Presently 35)

PROGRAM MAIN

IG(NBN), KODES(NBN), KODE(NBN,NBN)
SHBOL(NBN), KONC(NBN), IXPO(NBN,NBN)
IFLOW(NBN), N(NBN,NBN), CONS(NBN,NBN)
LT(NBN), NP(NBN),

SUBROUTINE SFG

JROW(NBN), TYPB(NBN), IQUALX(NBN), JBX(NBN)
NP(NBN), JB(NBN), VALX(NBN), LBX(NBN)
IVV(NBN), LB(NBN), NUMLX(NBN), IB(NBN,NBN)
NUML(NBN), MSYM(NBN), INTRE(NBN), NS(NBN,NBN)
ICV(NBN), IQUAL(NBN), NOTREE(NBN), NF(NBN,NBN)
INTREE(NBN), VAL(NBN), TYPX(NBN),
LINC(NBN), SYM(NBN), NUMX(NBN),

SUBROUTINE FTREE

TYPX(NBN), INTRE(NBN), NF(NBN,NBN)
JBX(NBN), NOTREE(NBN),
LBX(NBN), NP(NBN)

SUBROUTINE TREP

JX(NBN), JMEM(NBN), NF(NBN,NBN)
NP(NBN), KMEM(NBN)
NBG = Number of Branches in SFG (Presently 100)

PROGRAM MAIN

NFIRST(NBG), SYMBUL(NBG), NEST(NBG)
NLAST(NBG), MIX(NBG), TYPE(NBG)
IXPON(NBG), CVAL(NBG),
WEIGHT(NBG), KONSO(NBG),

SUBROUTINE SFG

NFIRST(NBG), MAPY(NBG), SYMBUL(NBG)
NLAST(NBG), KONSO(NBG), MIX(NBG)
IXPON(NBG), NEST(NBG), CVAL(NBG)
WEIGHT(NBG), TYPE(NBG),

NPAC = Number of Paths Plus Circuits (Presently 300)

PROGRAM MAIN

CONST(NPAC), MAPO(NPAC), JAC(NPAC)
KODET(NPAC), NOCTOT(NPAC), NPCODE(NPAC)
IXPOT(NPAC), NUP(NPAC),
NTO = Number of Terms in Output (Presently 150)

PROGRAM MAIN

NA(NTO), POLYU(NEXPS,NTO), SEMPON(NTO,NSPT/2)
NB(NTO), POLY(NEXPS,NTO), SEMPOD(NTO,NSPT/2)
KSORT(NTO), SIMBON(NTO,NSPT/2),
ITOP(NTO), SIMBOD(NTO,NSPT/2)

SUBROUTINE ARRAY

KSORT(NTO), POLY(NEXPS,NTO)

SUBROUTINE DECODE

ITOP(NTO)

NSPT = Number of Symbols per term in Output (Presently 20)

PROGRAM MAIN

KONS(NSPT), KODF(NSPT), SEMPON(NTO,NSPT/2)
KODI(NSPT), SIMBON(NTO,NSPT/2), SEMPOD(NTO,NSPT/2)
SEMBOL(NSPT), SIMBOLD(NTO,NSPT/2),

SUBROUTINE FTREE

KCOL(NSPT)

SUBROUTINE DECODE

SEMBOL(NSPT), KODF(NSPT), KODI(NSPT)
NEXPS = Number of Different Powers of s (Presently 15)

PROGRAM MAIN

MSORT(NEXPS), POLYU(NEXPS, NTO), POLY(NEXPS, NTO)

SUBROUTINE ARRAY

MSORT(NEXPS), POLY(NEXPS, NTO)

NRI = Maximum Number of Nontouching Loops (Presently 15)

PROGRAM MAIN

ISET(NRI, NCI)

NCI = Maximum Number of Loops Not Touching any Given Loop (Presently 100)

PROGRAM MAIN

ISET(NRI, NCI)

NEON = Number of Nontouching Pairs of Loops (Presently 1200)

PROGRAM MAIN

NOTCH(NEON)

NRS = Number of Repeated Symbols (Presently 9)

PROGRAM MAIN

STAR(NRS)
IV. PROGRAMMER'S GUIDE

IV-1. Definitions

\[ \text{CONST}(J,L) = \text{WEI}G(T)i \text{ FOR BRANCH } I \text{ OF THE SFG WHERE} \]
\[ J = \text{FIRST}(I), \ L = \text{LAST}(I) \]
\[ \text{CONST}(I) = \text{COMPOSITE CONSTANT ASSOCIATED WITH CIRCUIT } I. \text{ IT IS FOUND BY} \]
\[ \text{TAKING THE PRODUCT OF THE CONSTANT VALUES OF EVERY SFG BRANCH} \]
\[ \text{IN CIRCUIT } I \]
\[ \text{CVAL}(\text{NUMC}) = \text{VALX}(\text{LINK}) \text{ WHERE } \text{NUMC} = \text{NUMX}(\text{LINK}) \]
\[ (\text{USED ONLY FOR NETWORK BRANCHES NOT IN THE TREF}) \]
\[ \text{IR}(\text{LF}, \text{JF}) = \text{IB}(\text{JF}, \text{LF}) = \text{NUMC WHERE } JF = \text{JB}(\text{NUMC}) \text{ AND } I = \text{LB}(\text{NUMC}) \]
\[ \text{AND NUMC IS A NETWORK TREE BRANCH NUMBER (ASSIGNED BY USER)} \]
\[ \text{IFLOW}(K) = \text{A FLAG FOR THE PURPOSE OF CHECKING WHETHER NODE } K \text{ IS REPEATED} \]
\[ \text{AND WHETHER THE LAST NODE IS REACHED} \]
\[ \text{IT}(L) = \text{SYMBOL CODE ASSIGNED TO THE SFG BRANCHES HAVING} \]
\[ \text{TERMINAL NO. L} \]
\[ \text{INTRF}(K) = I. \text{ THE } I-\text{TH NETWORK BRANCH IN THE DATA BRANCH LIST IS} \]
\[ \text{CHOOSEN AS THE } K-\text{TH BRANCH OF THE NETWORK TREE} \]
\[ \text{INTRF}(\text{NUMC}) = 1 \text{ IF THE NETWORK BRANCH NUMBERED } \text{NUMC BY THE USER IS} \]
\[ \text{SELECTED FOR THE TREE, 0 OTHERWISE} \]
\[ \text{IVAL}(\text{NUMC}) = \text{IVALX}(I) \text{ WHERE } \text{NUMC} = \text{NUMX}(I) \]
\[ (\text{USED ONLY FOR NETWORK TREE BRANCHES}) \]
\[ \text{IVALX}(I) = \text{EQUAL SIGN(=) IF } I-\text{TH NETWORK BRANCH IN THE DATA BRANCH} \]
\[ \text{LIST HAS A NUMERICAL VALUE. LEFT BLANK IF } I-\text{TH BRANCH IS} \]
\[ \text{TO BE REPRESENTED BY A SYMBOL} \]
\[ \text{ISET}(J, I) = \text{THE INTEGER ARRAY WHICH TOGETHER WITH THE ARRAY NOTCH CAN} \]
\[ \text{BE USED TO FIND ALL SETS OF NONTOUCHING LOOPS OF ORDER GREATER} \]
\[ \text{THAN 2} \]
\[ \text{ITOP}(J, C) = 1 \text{ IF THE TERMS IN COLUMN JC OF THE ARRAY POLY BELONG TO} \]
\[ \text{THE NUMERATOR OF THE OUTPUT TRANSFER FUNCTION, 0 IF THEY} \]
\[ \text{BELONG TO THE DENOMINATOR} \]
\[ \text{IVV}(M) = \text{NETWORK BRANCH NUMBER OF THE } M-\text{TH VOLTAGE CONTROLLED} \]
\[ \text{VOLTAGE SOURCE IN THE DATA BRANCH LIST} \]
\[ \text{IXPO}(J, L) = \text{IXPON}(I) \text{ FOR BRANCH } I \text{ OF THE SFG WHERE} \]
\[ J = \text{FIRST}(I), \ L = \text{LAST}(I) \]
\[ \text{IXPON}(I) = \text{EXPONENT OF S ASSOCIATED WITH THE VALUE OF THE SFG BRANCH } I \]
\[ \text{IXPOT}(I) = \text{COMPOSITE EXPONENT OF S FOR CIRCUIT } I. \text{ IT IS FOUND BY ADDING} \]
\[ \text{THE } S \text{ POWERS ASSOCIATED WITH EACH BRANCH IN CIRCUIT } I \]
\[ \text{JAC}(J) = \text{NUMBER OF NONZERO ENTRIES IN ROW } J \text{ OF ISET} \]
\[ JH(\text{NUMC}) = \text{JHX}(I) \text{ WHERE } \text{NUMC} = \text{NUMX}(I) \]
\[ (\text{USED ONLY FOR NETWORK TREE BRANCHES}) \]
JAX(I)=INITIAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST
JMEM(I)=THE ROW OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN
JROW(LF)=THE NUMBER OF NON-ZERO ENTRIES IN ROW LF OF THE ARRAY NF
JX(I+1)=NP(I)
KBASIS=NUMBER BASE OF THE SYMBOL CODES. THAT IS, THE SFG CONTAINS KOD DISTINCTION SYMBOLS (SEMBO(K), K=1,2,...,KOD), NOT INCLUDING THE LAST VARIABLE (WHERE SEMBO(K) IS ASSIGNED THE CODE KBASIS+K)
KHOL=COUNTER USED TO FIND THE NUMBER OF LOOPS OF ORDER > OR GREATER THAN A ROW COUNTER OF THE MATRIX POLY
KMEM(I)=THE COLUMN OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN
KODE(JL)=CODE REPRESENTING THE SYMBOL OF THE SFG BRANCH HAVING J AS AN INITIAL NODE AND L AS THE TERMINAL NODE
KODES(J)=2**J···(J-1) WHERE J IS A NODE OF THE SFG
KODE(T)=COMPOSITE CODE ASSOCIATED WITH CIRCUIT I. THIS CODE REPRESENTS THE SET OF SYMBOLS CORRESPONDING TO THE SET OF SFG BRANCHES CONTAINED IN CIRCUIT I
KODE(NZ) IS THE MULTIPLICITY OF THE SYMBOL CORRESPONDING TO THE CODE KOD(NZ)
KODI(NZ)=NZ1,2,...,IZ IS THE SET OF INDIVIDUAL SYMBOL CODES THAT MAKE UP THE COMPOSITE CODE KORS(JZ)
KONC(J)=COLUMN COUNTER FOR ROW J OF THE ROUTING MATRIX N(J,K)
KONS(KOZY)=1 IF THE SYMBOL HAVING CODE KOZY IS NOT AN INVERSE SYMBOL A 0 IF THE SET OF SYMBOLS CORRESPONDING TO THE COMPOSITE CODE KORS(J) BELONGS TO THE DENOMINATOR POLYNOMIAL
KONS0(I)=1 IF SYMBOL OF THE SFG BRANCH I=SYMBOL(I), 0 IF SYMBOL OF THE SFG BRANCH I=SYMBOL(I+1)
KORS(K)=THE CODE ASSIGNED TO COLUMN K OF THE MATRIX POLY
LH(NUMC)=LHX(1) WHERE NUMC=NUMX(1) (USED ONLY FOR NETWORK TREE BRANCHES)
LX(I)=TERMINAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST
LINC(NUMC)=1 IF THE NETWORK BRANCH NUMBERED NUMC BY THE USER IS NOT IN THE TREE, 0 OTHERWISE
LIST=NUMBER OF DIRECTED BRANCHES IN THE SFG
LISTC=1 IF ALL CIRCUITS OF THE SFG ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LISTG=1 IF SFG INFORMATION (BRANCH SYMBOLS, WEIGHTS ETC.) ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LISTP=1 IF ALL PATHS FROM NODE NIN TO NODE NOUT ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LT(J)=NUMBER OF POSITIVE ENTRIES IN ROW J OF N(J,K)
MAP0(NIP)=NOCTOT(NIP)-NOCTOT(NIP-1) WHICH EQUALS THE NUMBER OF LOOPS NOT TOUCHING LOOP NIP
MIX(I)=MAPPING OF THE SFG BRANCH LIST INTO A LIST SATISFYING ONE OF THE FOLLOWING CONDITIONS
NFIRST(J).GT.NFIRST(K) FOR J.GT.K OR NFIRST(J)=NFIRST(K), NLAST(J).LT.NLAST(K) FOR J.GT.K
MSORT(K)=THE EXPONENT OF S ASSIGNED TO ROW K OF THE MATRIX POLY
N(J,K)=WHERE K=I,2,...,LT(J) IS THE TERMINAL NODE OF SFG BRANCH HAVING J AS ITS INITIAL NODE. THE VALUE OF EACH NONZERO ENTRY IN A GIVEN ROW IS MADE TO DECREASE AS K INCREASES. THE ADDITION ENTRY N(NF+LT(NIN)+1)=1 IS ALSO MADE
NA(J)=NUMBER OF SYMBOLS (NOT COUNTING INVERSE SYMBOLS) IN THE CODE KORS(J)
NB(J)=NUMBER OF INVERSE SYMBOLS IN THE CODE KORS(J)
NCIR=1 IF CIRCUITS ARE TO BE FOUND, AND 0 IF CIRCUITS ARE NOT TO BE FOUND
NEST(1)=1 IF THE SFG BRANCH 1 CONTAINS A SYMBOL IN ADDITION TO THE LAPLACE VARIABLE S; 0 IF THE SFG BRANCH 1 CONTAINS NO SYMBOL EXCEPT POSSIBLY FOR THE LAPLACE VARIABLE S.

NF(LF,JROJ)=ROUTING TABLE FOR THE NETWORK COMPOSED ONLY OF BRANCHES BELONGING TO THE TREE.

NFIR=1 IF PATHS ARE TO BE FOUND (NFIR SET TO 1 IF LISP=1), AND 0 IF PATHS ARE NOT TO BE FOUND.

NFIRST(I)=INITIAL NODE OF THE DIRECTED SFG BRANCH I.

NIN=NETWORK BRANCH NUMBER OF THE SOURCE. THIS BECOMES THE SOURCE NODE OF THE SFG.

NLAST(I)=TERMINAL NODE OF THE DIRECTED SFG BRANCH I.

NO=NUMBER OF BRANCHES IN NETWORK.

NODA=0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK ELEMENT. IN THIS CASE IT DESIGNATES THE POSITIVE TERMINAL OF THE OUTPUT VOLTAGE.

NOOA=0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK ELEMENT. IN THIS CASE IT DESIGNATES THE NEGATIVE TERMINAL OF THE OUTPUT VOLTAGE.

NO=NUMBER OF CIRCUITS (LOOPS).

NPO=NUMBER OF PATHS FROM NODE NIN TO NODE NOUT IN THE SFG.

NOUT=NETWORK BRANCH NUMBER ASSOCIATED WITH THE OUTPUT (VOLTAGE ACROSS CURRENT TIME). THIS BECOMES THE SFG NODE CORRESPONDING TO 1 OUTPUT VARIABLE.

NOTCH(OC) AND NOCTOT(K), CONSIDER THE INTEGER SET (I) = (1, 2, ..., N2) WHERE N2=NUMBER OF NONTOUCHING PAIRS OF LOOPS.

NOW CONSIDER THE FOLLOWING SUBSET OF (I).

S(I) = (NOCTOT(K-1) + 1 + NOCTOT(K-1) + 2 +... + NOCTOT(K)) WHDF

NOCTOT(0)=0. THEN THE SET (NOTCH(J), J IN S(I)) IS THE SET OF LOOPS THAT DO NOT TOUCH LOOP K.

NOTREE(I)=1 IF THE I-TH NETWORK BRANCH IN THE DATA LIST IS CHOSEN FOR THE TREE; 0 OTHERWISE.

NP(I)=THE NODE SEQUENCE OF A PATH BETWEEN NOUVE NIN AND NODE NOUT OF THE SFG. IF VIN=NOUT THIS IS THE NODE SEQUENCE FOR A CIRCUIT.

NPCODE(K)=COMPOSITE CODE USED TO IDENTIFY CIRCUIT K, FOUND BY SUMMING THE CODES, COUES(J), ALLOTTED TO EACH NODE, J, IN THE CIRCUIT.

NUML(NUMC)=NUML(A(I)) WHERE NUMC=NUMX(I) (USED ONLY FOR NETWORK TREE BRANCHES).

NS(LF,JF)=1 IF THE NETWORK TREE BRANCH IN (LF,JF) HAS INITIAL NODE LF AND TERMINAL NODE JF AND EQUALS = I IF THE NETWORK TREE BRANCH HAS INITIAL NODE JF AND TERMINAL NODE LF.

NUMX(I)=THE NETWORK BRANCH NUMBER ASSIGNED BY THE USER TO THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST.

NUP(J) DESIGNATES THE LOOP ISET(J, NUP(J)),* OF ROW J WHICH IS NOT TOUCHED BY THE LOOPS ENTERED IN ROW J+1 OF ISET.

POLY(K,L)=MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A TERM IN THE NUMERATOR OR DENOMINATOR OUTPUT POLYNOMIAL HAVING THE S POWER OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L.

POLYU(K,L)=MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A TERM IN THE NUMERATOR OF THE OUTPUT POLYNOMIAL HAVING THE S POWER OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L.

SEMHR(KO)=SYMBOL CORRESPONDING TO THE CODE KBASIS** (KO=1).

SEMPON(J1,J2), J2=1,2,...,NA(J1), AND SEMPON(J1, J3), J3=1,2,...,NA(J1) ARE RESPECTIVELY THE MULTIPLES OF THE SYMBOLS SIMON(J1, J2), J2=1,...,NA(J1), AND SIMHOD(J1, J3), J3=1,...,NA(J1), SIMHOD(J1, J2), J2=1,...,NA(J1), AND SIMHOD(J1, J3), J3=1,...,NA(J1) ARE RESPECTIVELY THE SYMBOLS AND INVERSE SYMBOLS CORRESPONDING TO THE SYMBOL CODE KSOX(T1).
SMBOL(K) = SMBUL(I) FOR THE SFG BRANCH I WHERE I=MIX(K)
STAR(I) = **I THIS ARRAY IS GENERATED FROM DATA STATEMENTS AND IS
USED IN FORMING THE ARRAYS SEMPO AND SEMPOD
SYM(NUMC) = SYMX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
SMBRL(I) = SYMBOL ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I
SYMX(I) = SYMBOL(3 CHARACTERS AT MOST) ASSIGNED BY USER TO THE I-TH
NETWORK BRANCH IN THE DATA BRANCH LIST. THE ELEMENTS VALUE IF NOT SPECIFIED IS REPRESENTED BY THIS SYMBOL
SYMRLC(I) = SYMX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
SYMRL(I) = SYMRLC(I) WHERE NUMC=NUMX(I)
TYPE(NUMC) = TYPX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES NOT IN THE TREE)
VAL(NUMC) = VALX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
VALX(I) = ELEMENT VALUE(E12.5) OF I-TH NETWORK BRANCH IN THE DATA
BRANCH LIST
WEIGT(I) = CONSTANT TERM ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I
IV-2. Flow Charts

Program SNAP is divided into the following sections:

- Program MAIN (Subprograms 1 thru 12)
- Subroutine SFG (Subprograms A thru J)
- Subroutine FTREE
- Subroutine TREP
- Subroutine ARRAY
- Subroutine DECODE

As indicated above, program MAIN is further broken down into 12 subprograms and subroutine SFG is divided into 10 subprograms.
Subprogram MAIN-1

This program reads in some preliminary network data.

Read in
(a) problem name
(b) NOD, NOB, KBASIS, LISTC,
    LISTC, LISTP, NIN, NOUT,
    NODA, NODB
Set KBASIS to 8 if a zero valve has been read in.
Write out the above information for reference purposes.

To Subprogram MAIN-2
Subprogram MAIN-2

This program generates the SFG routing matrix, creates a code for each symbol (excluding s), and sets up arrays for the constants and powers of s associated with the branch weights.

Call subroutine SFG.
Transfer the following data into subroutine SFG: NIN, NOUT, NOD, NOB, LISTG, NODA, NODB.
(see subroutine SFG for additional data read in)
Subroutine SFG returns the following information to program MAIN-2:
LIST, NFIRST(I), NLAST(I), IXPON(I), WEIGT(I), SYMBUL(I), KONSO(I), NEST(I), MIX(I), I = 1, LIST
MG = KBASIS*MG
initially MG=1

IBO = IBO + 1
initially IBO = 0

Compare IBO to LIST

all branches have been evaluated

To Subprogram MAIN-3

if IBO > LIST

LOB = MIX(IBO)
J = NFIRST(LOB)
L = NLAST(LOB)

LT(J) = LT(J) + 1
N(J,LT(J)) = NLAST(LOB)
After the Jth row is completed set
N(J,LT(J)+1) = \[ \begin{cases} 0 & \text{if } J \neq NIN \\ -1 & \text{if } J = NIN \end{cases} \]

The mapping LOB = MIX(IBO) reorders the SFG branch list so that (1) the nonzero entries of a given row of the routing matrix N(J,I) decreases as I increases and (2) the Jth row is completed before elements are entered into the J+1st row.

CONST(J,L) = WEGT(LOB)
IXPO(J,L) = IXPON(LOB)
where J = NFIRST(LOB), L = NLAST(LOB)
SMBOL(IBO) = SYMBOL(LOB)

Test NEST(LOB)

KODE(J,L) = 0

= 0

SFG branch LOB does not contain a symbol

Test IG(L) where L = NLAST(LOB)

KODE(J,L) = IG(L)

= 1

A code has already been assigned to branches entering node L

KP = KP + 1
Initially KP = 0

Compare SMBOL(IBO) to SMBOL(KP)

not equal

952

equal

Compare KONSO(LOB) and KONSO(LOBX) where LOBX = MIX(KP)

not equal

the symbol associated with branch IBO is the inverse of the symbol associated with branch KP

not equal

Compare KP to IBO - 1

952

equal
MG = KBASIS * MG  
(Initially MG = 1)  
IG(L) = MG  
KODE(JL) = IB(L)

KOO = KOO + 1  
Initially KOO = 0

SEMBOL(KOO) = SYMBOL(IBO)  
This establishes a direct correspondence  
between a symbol and its code  
SEMBOL(KOO) \iff \text{KBASIS}^{**}KOO

Test KONSO(LOB)  
\begin{align*}  
\text{SFG branch LOB} & \text{ contains the symbol} \\
\frac{1}{SEMBOL(KOO)} & \rightarrow \text{KONS(KOO)=1} \\
\text{SFG branch LOB} & \text{ contains the symbol} \\
SEMBOL(KOO) & \rightarrow \text{KONS(KOO)=0}
\end{align*}
Subprogram MAIN-3

This program codes the nodes of the SFG, and prepares the counters for finding all paths and/or circuits.

```
Set POLY(J,K) = 0, J = 1, NEXPS; K = 1, NTO
MSORT(J) = 0, J = 1, NEXPS
KODI(J) = 0, J = 1, NSPT
KSORT(J) = 0, J = 1, NTO
IR = 1, NFIR = 1, KNO = 0

Code node JS of the SFG as follows
KODES(JS) = 2 * KODES(JS-1)
JS = 2, NNG
where KODES(1) = 1
```

```
TEST

= 1
Write out
LISTP
NIN & NOUT

= 0
Prepare to
find circuits
thru node 1

N[NIN, LT(NIN) + 1] = 0
N[1, LT(1) + 1] = -1
NIN = 1
NOUT = 1
KLAS = 0

NFIR = 0

To Subprogram MAIN-4
```

(Subprogram MAIN-4)
Subprogram MAIN-4

This program finds all paths from node NIN to node NOUT and/or all circuits of the SFG.

Preliminary (PF1-1)
IFLOW(I1)=0  I1=1,NNG
KONC(I1)=0  I1=1,NNG
NIP=KLAS
KLAS=0

I = 2

Preliminary (PF1-2)
JX(1)=NIN
J=NIN
NP(1)=NIN
IFLOH(NIN)=1
IFLOW(OUT)=-1

K = KONC(J)

PF2(FIND NEXT NODE)
NP(I)=N(J,K)

PF3 (TEST ROUTING MATRIX)
N(J,K)

< 0
All loops thru a particular node found. Go to subprogram MAIN-5 to eliminate this node.

> 0
continue to flower check

100
(Subprogram MAIN-5)

All paths out of node J checked. Return to previous node.
(Prepare for next node) set IFLOW(J)=1 I=I+1

FLOW(J)=1

"'0

Node J acceptable

> 0

Node J is the final node

PF7
(Loop Completed)

(1) Find the composite code for the circuit node list.

I=1
NPCODE(IR) = \sum KODE [NP(IS)]
IS=1

(2) Find composite code, exponent, and constant associated with branch weight.

I
KODET(IR) = \sum KODE [NP(KEW-1), NP(KEW)]
KEW=2

I
CONST(IR) = \prod CONS [NP(KEW-1), NP(KEW)]
KEW=2

I
IXPOT(IR) = \sum IXP0 [NP(KEW-1), NP(KEW)]
KEW=2

Set CONEW=CONST(IR), IXNEW=IXPOT(IR), KONEW=KODET(IR)
CALL ARRAY (1, CONEW, IXNEW, KONEW, POLY, L1, LK, KIK)

IR=IR+1

Look for another loop
Subprogram MAIN-5

This program determines if circuits are to be found and if so modifies the SFG by eliminating node J.

Write out time to find paths

TEST NCIR

 Only paths between nodes NIN and NOUT to be listed

STOP

=0

TEST NFIR

=1 The user has specified that circuits be formed

Paths have been found - circuits thru node 1 are to be found

Write out time to find paths. Set N[1,LT(1)+1]=1

NIN=1

NOUT=1

All circuits found thru node J have been found. The SFG must be modified by eliminating node J from the routing table.

NIN=J+1

NOUT=J+1

N[J,LT(J)+1]=0

Test last nonzero entry in each row of N(I1,I2). If this entry equals J set N(I1,I2)=0 and LT(I1)=LT(I1)-1

N[NIN,LT(NIN)+1]=-1

(Subprogram MAIN-4)
Subprogram MAIN-6

This program finds and stores all 2nd order nontouching loops

Set
NOCTOT(K1)=0  K1=1,NPAC
LOOP(M1)=0  M1=1,NNG

DO 203 LIR1=NOP+1,NOL-1

DO 202 LIR2=LIR1+1,NOL

*AND* together the node code for loop LIR1 and the node code for loop LIR2
NAN=NPCODE(LIR1).AND.NPCODE(LIR2)

CALL ARRAY (LIR2,LIR1,KXPO2,KSYM2,LIR1,LIR2)

CALL ARRAY (2,TCONS2,KXPO2,KSYM2,POLY,LIL,KIK)
The following defines arrays NOTCH and NOCTOT:

Consider the integer set

\[ \{I\} = \{1, 2, \ldots, N_2\} \]

where \( N_2 \) = number of nontouching pairs of loops

Now consider the following subset of \( \{I\} \)

\[ \{I_{g}\} = \{NOCTOT(K-1)+1, NOCTOT(K-1)+2, \ldots, NOCTOT(K)\} \]

Then the set

\[ \{NOTCH(J): J \in \{I_{g}\}\} \]

is the set of loops that do not touch loop \( K \).
Subprogram MAIN-7

This program finds all nontouching loops of order greater than 2, and stores the associated code, power of s, and constant term.

The matrix ISET is given below in its general form to aid in understanding the flow chart of subprogram MAIN-7.

\[
\begin{bmatrix}
\text{ISET}(1,1)\text{ISET}(1,2)\cdots\text{ISET}[1,\text{NUP}(1)]\cdots\text{ISET}[1,\text{JAC}(1)] \\
\text{ISET}(2,1)\text{ISET}(2,2)\cdots\text{ISET}[2,\text{NUP}(2)]\cdots\text{ISET}[2,\text{JAC}(2)] \\
\vdots \\
\text{ISET}(\text{KAP},1)\text{ISET}(\text{KAP},2)\cdots\text{ISET}[\text{KAP,\text{NUP}(\text{KAP})]}\cdots\text{ISET}[\text{KAP,\text{JAC}(\text{KAP})}] \\
\text{ISET}(\text{KAP},\text{NUP}(\text{KAP}+1))\cdots
\end{bmatrix}
\]

where \(\text{ISET}(J,I), I=1,2,\ldots,\text{JAC}(J)\)
is the subset of

\[\{\text{ISET}[J-1,\text{NUP}(J-1)+1], \text{ISET}[J-1,\text{NUP}(J-1)+2],\ldots,\text{ISET}[J-1,\text{JAC}(J-1)]\}\]

which does not touch the loop

\(\text{ISET}[J-1,\text{NUP}(J-1)]\)
DO 490 NIP=NOP+1,NOL

Generate the first row of ISET as follows:
ISET(1,1)=NOTCH(NOCTOT(NIP-1)+1)
ISET(1,2)=NOTCH(NOCTOT(NIP-1)+2)
...
ISET[1,JAC(1)]=NOTCH(NOCTOT(NIP))
The first row of ISET is seen to be the set of loops which do not touch loop NIP

Set JAC(K)=0 \{K=1,NPAC
NUP(K)=0 \}

KAP=2

\[\text{KAP is the row counter of ISET}\]

KAP=KAP-1

All higher order loops not touching loop NIP have been found

\(\leq 0\)

490

\(> 0\)

TEST KAP

KAP=KAP+1

425

JAC(KAP+1)=0
NUP(KAP)=NUP(KAP)+1

429
ISAT = ISET(KAP, NUP(KAP))

DO 435 MAPI = NUP(KAP) + 1, JAC(KAP)

AND, together the node code of loop ISAT and the node code of loop ISOT
KAN = NPCODE(ISAT).AND. NPCODE(ISOT)
where ISOT = ISET(KAP, MAPI)

TEST KAN ≠ 0

= 0 [Loops ISAT and ISOT do not touch]

Find constant, code, and exponent for the nontouching loops of order KAP+1 just found.

\[
\begin{align*}
TCONSG &= \text{CONST}(\text{NIP}) \times \text{CONST}(\text{ISOT}) \times \prod_{L=1}^{\text{KAP}} \text{CONST}[\text{ISET}(L, \text{NUP}(L))] \\
KSYM &= \text{KODET}(\text{NIP}) \times \text{KODET}(\text{ISOT}) + \sum_{L=1}^{\text{KAP}} \text{KODET}[\text{ISET}(L, \text{NUP}(L))] \\
KXP &= \text{IXPOT}(\text{NIP}) + \text{IXPOT}(\text{ISOT}) + \sum_{L=1}^{\text{KAP}} \text{IXPOT}[\text{ISET}(L, \text{NUP}(L))]
\end{align*}
\]

CALL ARRAY (KAP+2, TCONSG, KXP, KSYM, POLY, LIL, KIK)
Update column counter of row KAP+1 of ISET and insert the last loop found into ISET

\[ JAC(KAP+1) = JAC(KAP+1) + 1 \]
\[ ISET[KAP+1, JAC(KAP+1)] = ISET(KAP, MAPI) \]

435 \[ \text{End of DO loop} \]

430 \[ \text{End of loop} \]

Row KAP+1 of ISET has now been found and it has more than 1 non-zero entries. Thus, proceed to find the entries of row KAP+2.

\[ JAC(KAP+1) - 2 \geq 0 \]

425

\[ JAC(KAP+1) - 2 < 0 \]

Row KAP+1 of ISET has now been found but it does not contain more than one non-zero entry. Since not all loops of row KAP have been exhausted, increment NUP(KAP) by one and re-evaluate row KAP+1.

\[ JAC(KAP) - NUP(KAP) - 1 > 0 \]

429

\[ JAC(KAP) - NUP(KAP) - 1 \leq 0 \]

Row KAP+1 of ISET has now been found but it does not contain more than one non-zero element. Further, all loops of row KAP have been exhausted. Thus, it is necessary to back up one row and re-evaluate row KAP概况：
CALL ARRAY (2,1.,0,0,POLY,LIL,KIK)

Print out number of loops found and time required in seconds.

To Subprogram MAIN-8
Subprogram MAIN-8

This program decodes composite codes representing nontouching loops and sets up tags for use in printing out the symbolic transfer function.

Set

\[
\begin{align*}
\text{POLYU}(J1,J2) &= 0 \quad J1 = 1, \text{HEXPS}; \quad J2 = \text{NTO} \\
\text{SEMPON}(J3,J4) &= \text{STAR}(1) \\
\text{SEMPOD}(J3,J4) &= \text{STAR}(1) \\
\text{SIMBON}(J3,J4) &= \text{SB} \\
\text{SIMBOD}(J3,J4) &= \text{SB} \\
\text{NA}(J5) &= 0 \\
\text{NB}(J5) &= 0
\end{align*}
\]

\(J3 = 1, \text{NTO}; \quad J4 = \text{NSPTU}\)

SB and STAR(1) are obtained from "data" statements

\[\text{DO } 646 \quad JZ = 1, \text{LIL}-1\]

(LIL-1 different composite symbol codes have been found in Subprogram MAIN-7)

\[
\begin{align*}
\text{KODY} &= \text{KSORT}(JZ) \\
\text{ITOP}(JZ) &= 0
\end{align*}
\]

There is no symbol associated with this value of JZ

\[646\]
CALL DECODE (KOO, KBASIS, KODY, IZ, FB, JZ, SEMBOL, KODF, KODI, ITOP)

This subroutine (a) sets

\[ ITOP(JZ) = \begin{cases} 
1 & \text{if terms having the code } KSORT(JZ) \text{ belong in numerator of output polynomial} \\
0 & \text{if terms belong in denominator} 
\end{cases} \]

(b) finds the set

\[ KODI(I), I=1,IZ \]

where \( SEMBOL[KODI(I)], I=1,IZ \) are the corresponding set of symbols, and

(c) finds the multiplicity of each individual symbol, \( SEMBOL[KODI(I)] \), and records these values in the array \( KODF(I), I=1,IZ \)

\[
\text{DO 645 } NZ=1, IZ
\]

\[
\text{TEST } KONS[KODI(NZ)] = 0
\]

The symbol corresponding to \( KODI(NZ) \) is not to be inverted in the output

\[
\text{NAK=NAK+1}
\]
\[
\text{SIMBON(JZ,NAK)=SEMBOL[KODI(NZ)]}
\]
\[
\text{SEMPON(JZ,NAK)=STAR[KODF(NZ)]}
\]
\[
\text{NA(JZ)=NA(JZ)+1}
\]

\[
\text{TEST } KONS[KODI(NZ)] = 1
\]

The symbol corresponding to \( KODI(NZ) \) is to appear inverted in the output. i.e.

\[
\frac{1}{SEMBOL[KODI(NZ)]}
\]

\[
\text{NAT=NAT+1}
\]
\[
\text{SIMBOD(JZ,NAT)=SEMBOL[KODI(NZ)]}
\]
\[
\text{SEMPOD(JZ,NAT)=STAR[KODF(NZ)]}
\]
\[
\text{NB(JZ)=NB(JZ)+1}
\]
increment
NZ

```
645
```

[End of DO loop]

continue

```
646
```

[End of DO loop]

increment
JZ

Subprogram MAIN-9
Subprogram MAIN-9

This program separates POLY into the arrays POLYU and POLY for use in printing out the constant terms of the transfer function.

```plaintext
DO 755 JA=1,KIK-1
    There are KIK-1 rows in POLY (KIK-1 different powers of s)

DO 755 JC=1,LIL-1
    There are LIL-1 columns in POLY (LIL-1 different composite symbol codes)

The entry POLY(JA,JC) belongs in the denominator of the transfer function

JIB=JIB+1
POLYU(JA,JIB)=POLY(JA,JC)

The entry POLY(JA,JC) belongs in the numerator of the transfer function

JD=JD+1
POLY(JA,JD)=POLY(JA,JC)

[End of DO loop]

Increment JC until JC=LIL-1 then increment JA

Print out time for decoding symbols

Subprogram MAIN-10
```
Subprogram MAIN-10

This program normalizes the transfer function so as to have all positive powers of $s$

```
DO 522 KAR=1, KIK-1
```

- **TEST**
  - **MSORT(KAR)**
    - **> 0**
      - The power of $s$ corresponding to row KAR of POLYU or POLY is negative
    - **< 0**
      - MAXIM is to be the absolute value of the most negative power of $s$
```
MAXIM+MSORT(KAR)
```

```
END of DO loop
```

```
Using MAXIM make powers of $s$ either positive or zero.
i.e.
MSORT(K)=MAXIM+MSORT(K)
K=1,2,...,KIK-1
```

Subprogram MAIN-11
Subprogram MAIN-11

Write out the matrix of constant coefficients for the numerator polynomial of the transfer function. Also write out the symbols and s powers that correspond respectively to the columns and rows of the array of constants.

Subprogram MAIN-12

Write out the matrix of constant coefficients for the denominator polynomial of the transfer function. Also write out the symbols and s powers that correspond respectively to the columns and rows of the array of constants.
Subroutine SFG(NFIRST, NLAST, IXPON, WEIGHT, SYMBSL, KONS0, MIX, HIST, LIST, NIN,
NOUT, MOD, KOB, LISTG, NODA, NODB)

This subroutine generates a signal flow-graph (SFG) for the given network.
The program is subdivided into subprograms A thru J.
Subprogram "A"

This program uses DATA statements to define certain variables, nulls arrays, creates the SFG feedback branch, reads in network branch information and calls FTREE to find a tree.

Use DATA statement to define the following variables: Y, G, C, IQ, R, CL, Z, E, CI, CC, CV, VV, VC, FB, ONE.

Set NS(IC, IK) = 0  IC = 1, NNG; IK = 1, NNG
NF(IC, IK) = 0
NEST(IG) = 0  IG = 1, NBG
KONSO(IG) = 0
INTREE(I1) = 0  I1 = 1, NNG
JROW(I1) = 0
MO = 0, LO = 0, LIST = 1, LINK = 0

Create SFG feedback branch used to make the SFG "closed".
NLAST(1) = NIN
IXPON(1) = 0
WEIGT(1) = -1
SYMBUL(1) = FB
KONSO(1) = 0
NEST(1) = 1
Note: NFIRST(1) is determined in Subprogram "I"
Read in the following network branch information:
TYPX(I), NUMX(I), JBX(I), LBX(I), SYMX(I),
IQUALX(I), VALX(I), NUMLX(I)
I=1, NOB

Choose a tree of the network for use in finding the SFG
CALL FTREE(TYPX, JBX, LBX, INTRE, NOTREE, NOD, NOB)

Subprogram "B"
Subprogram "B"

This program sets up tree branch information and creates a routing matrix and sign matrix for the tree.

DO 21 NU=1,NOD-1

Put network tree branch information in terms of the branch labels specified by user.
Let IO=INTRE(NU), NUMC=NUNX(IO)
Then TYPB(NUMC)=TYPX(IO)
   JB(NUMC)=JBUX(IO)
   LB(NUMC)=LBX(IO)
   SYM(NUMC)=SYMX(IO)
   IQUAL(NUMC)=IVALX(IO)
   VAL(NUMC)=VALX(IO)
   NUML(NUMC)=NUMLX(IO)
   INTREE(NUMC)=1

WRITE TYPB(NUMC), NUMC, JB(NUMC), LB(NUMC),
   SYM(NUMC), IQUAL(NUMC), VAL(NUMC), NUML(NUMC)

Compare TYPB(NUMC) to VV
   .EQ.  MO=MO+1
   .NE.

   Compare TYPB(NUMC) to CV
   .EQ.  LO=LO+1
   .NE.
The following generates a routing matrix, $NF$, and a sign matrix, $NS$, from the chosen network tree for use in finding the SFG.

- $IB(JF, LF) = NUMC$
- $IB(LF, JF) = NUMC$

```
JF = JB(NUMC)
LF = LB(NUMC)
JROW(JF) = JROW(JF) + 1
NF[JF, JROW(JF)] = LF
NS[JF, LF] = 1
JROW(LF) = JROW(LF) + 1
NF[LF, JROW(LF)] = JF
NS[LF, JF] = 1
```

Increment NU

```
[End of 21 DO loop]
```

Continue

```
Subprogram "C"
```
Subprogram "C"

This program generates SFG information from branch node to link node.

```
NES=0
NOBY=NOB

Continue

TEST KLU-NOB

=0

< 0

LINK=LINK+1
Initially 0

TEST NOTREE(LINK)

=1

=0 Network branch NUMX(LINK) is not in tree

```

All passive network branches together with VC and CC type sources have been converted to SFG variables. Now all CV and VV type sources must be accounted for.

```

Put network link information in terms of the branch labels specified by user.
Let NUMC=NUMX(LINK)
then TYPE(NUMC)=TYPX(LINK)
    JK=JBX(LINK)
    LK=LBX(LINK)
    SYM(NUMC)=SYMX(LINK)
    IQUAL(NUMC)=IQUALX(LINK)
    CVAL(NUMC)=VALX(LINK)
    NUMB=NUMX(LINK)

```

```

LINK(NUMC)=1
This records which network branches are not in the tree.

```
CALL TREP(JK, LX, HF, HP, NPL)
TREP finds the path in the network tree which forms a circuit with the link NUMC.

IFIN=NUMC

LON=LON+1

Counter LON will increment over the number of tree branches contained in the fundamental circuit defined by the link NUMC. For each tree branch in this list, a SPG branch will be formed from NUMC to the given tree branch.
INIT=IB[EP(LON),NP(LON+1)]
SIGN=NS[NP(LON),NP(LON+1)]

169 Subprogram "D"

KDEPS

=1

=0 [tree branch INIT is passive]

COMPARE IQUAL(NUMC) and IQ

.NE. WES=1 CONST=SIGN

.EQ. [Network branch NUMC has no symbol]

CONS=SIGN*CUAL(NUMC)

LIST=LIST+1

TEST WES

=0

=1

HES(LIST)=1

Specify all branch information for LIST
KONSO(LIST)=KANSO
NFIRST(LIST)=INIT
LAST(LIST)=IFIN
SYMBUL(LIST)=SYH(IFIN)
IXPOS(LIST)=IXPS
TEST KONSO(LIST)  = 0 \rightarrow WEIGHT(LIST) = CONST

= 1 \quad \text{Link weight is given in impedance units and thus must be inverted.}

WEIGHT(LIST) = \frac{1}{\text{CONST}}

MAPY(NUMC) = LIST

Subprogram "D"
Subprogram "D"

This program generates SFG information from link node to branch node.

```
 TEST TYPB(INIT)  .EQ. E,I,VV, or CV
                 .NE. E,I,VV, or CV 201
                 LIST=LIST+1

 TEST TYPB(INIT)
 .EQ. R or Z
 IXPON(LIST)=0
 KONSO(LIST)=1

 .EQ. C
 IXPON(LIST)=-1
 KONSO(LIST)=1

 .EQ. L
 IXPON(LIST)=1

 .EQ. G or Y
 IXPON(LIST)=0
 KONSO(LIST)=1

 COMPARE
 IQUAL(INIT) to IQ
 .EQ.

 TEST KONSO(LIST)  .NE. NEST(LIST)=1
                      WEIGHT(LIST)=-SIGN
                      WEIGHT(LIST)=-SIGN*VAL(INIT)

                      WEIGHT(LIST)=-SIGN/VAL(INIT)
```
Subprogram tiC

NFIRST(LIST)=IFIN
NLAST(LIST)=INIT
SYMBUL(LIST)=SYM(INIT)

NPLA=NPL-1-LON

TEST

NPLA

149
Subprogram "C"

151
Subprogram "C"
Subprogram "E"

This program sets up SFG information for VC type control sources.

165 Numb was found from
NUMB=NUMLX(LINK)
Thus, Numb is the branch
number of the control
for the VC type source.

NUNO=NUMB

TEST
INTR(E)(NUMB)

=1

=0

The network branch whose
current is the controlling
variable for the VC source
is not in the tree. Thus a
new node must be created in
the SFG to represent the
current controlling variable.

Create a SFG branch from node
NUMB (current thru controlling
network branch) to new node
NOBY (voltage across controlling
network branch)
LIST=LIST+1
NFIRST(LIST)=NUMB
NOBY=NOBY+1 (initially
NOBY=NOB)
NLAST(LIST)=NOBY
SYMBUL(LIST)=SYM(NUMB)

NUNO=NOBY
Network branch NUMB contains a symbol.

Using the newly created SFG node representing the controlling voltage, a SFG node can now be generated for the dependent current source.
Subprogram "C"

Network branch NUMC contains a symbol

NE

NEQ.

NE)

NE.

NE.

Network branch NUMC contains a symbol

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.

NE.
Subprogram "F"

This program sets up SFG information for CC type control sources.

265

The structure of subprogram "F" is completely analogous to subprogram "E".

123 Subprogram "C"

Subprogram "G"

This program sets up SFG information for VV type control sources.

360

This program is cycled over all voltage controlled voltage sources in the network. A list of these sources is maintained in the array

IVV(MI),MI=1,M0

Other than the above, the structure of subprogram "G" is completely analogous to subprogram "E".

460 Subprogram "H"
Subprogram "H"

This program sets up SFG information for CV type control sources.

This program is cycled over all current controlled voltage sources in the network. A list of these sources is maintained in the array ICV(MI), MI=1,LO

Other than the above, the structure of subprogram "H" is completely analogous to subprogram "E"

515 Subprogram "I"

The SFG is now complete except for the output node.
Subprogram "I"

This program generates the output node of the SFG.

User has specified that output is between nodes NODA and NODB of the network. The SFG output node NOUT is made a function of the tree branch voltages existing in the path between NODA and NODB.

CALL TREP (NODA,NODB,NP, NP, NPL)

NOUT=NODB+1

DO 510 KOP=1, NPL-1

LIST=LIST+1

510 [End of DO loop]

Generate a SFG branch between node IB[NP(KOP), NP(KOP+1)] and NOUT
NFIRST(LIST)=IB[NP(KOP), NP(KOP+1)]
NLAST(LIST)=NOUT
SYMBUL(LIST)=ONE
IXCON(LIST)=0
KNSO(LIST)=0
NEST(LIST)=0
WEIGHT(LIST)=NS[NP(HOP), NP(MOP+1)]
514

NFIRST(1) = NOUT
This is the initial node of the SFG feedback branch

= 0
TEST LISTG

= 1
Write out the following SFG information:
NFIRST(J), NLAST(J), IXPON(J), WEIGHT(J),
SYMBUL(J), KONSO(J), NEST(J)
J = 1, LIST

Continue

Subprogram "J"
Subprogram "J"

This program orders SFG information for input to main program. That is, a mapping function MIX is found which reorders the SFG information so that the routing matrix N(J,K) as calculated in MAIN will automatically have its entries decrease as K increases.

- Flowchart -

1. SET MIX(J)=J J=1,NBG
2. DO 80 KON=1,LIST-1
   - IU=KON+1
   - IL=KON
3. TEST
   - NFIRST[MIX(IU)]<NFIRST[MIX(IL)]
     - [MIX(IU) and MIX(IL) must be interchanged]
     - MXL=MIX(IL)
     - MIX(IL)=MIX(IU)
     - MIX(IU)=MXL
4. 83
5. 80
mX(IU) and 
mIX(IL) must be 
interchanged

mXL=mXIL(IL)
mIX(IL)=mIX(IU)
mIX(IU)=mXL

IL=IL-1
IU=IU-1

TEST
IL

TEST
mLAST[mIX(IU)]-mLAST[mIX(IL)]

<= 0

> 0

mIX(IU) and 
mIX(IL) must be 
interchanged

mXL=mXIL(IL)
mIX(IL)=mIX(IU)
mIX(IU)=mXL

IL=IL-1
IU=IU-1

> 0 82

<= 0

mX([mIX(IU)]-mX([mIX(IL)]

#0

Continue
Subroutine FTREE(TYPX,JBX, LBX,INTRE,NOTREE,NOD,NOB)

This program finds a tree of the network to use in generating a SFG.

1. Null the arrays
   NF(I2,I3),KCOL(I4),NOTREE(I5)

2. \( I = I + 1 \)
   Initially \( I = 0 \)

3. TEST
   \( \text{TYPE}(I) \)

4. \( = E, VV, \text{or CV} \)

5. \( K = K + 1 \)

6. TEST
   \( \text{INTRE}(K) = I \)
   \( \text{NOTREE}(I) = 1 \)

The branch just selected for the tree must now be included in a routing matrix. This matrix will be used later to test passive network branches for use in the tree.

\[
\text{NF}[\text{JBX}(I),\text{KCOL}(\text{JBX}(I))] = \text{LBX}(I) \\
\text{NF}[\text{LBX}(I),\text{KCOL}(\text{LBX}(I))] = \text{JBX}(I)
\]

7. TEST
   \( K = \text{NOD} + 1 \)
   \( \geq 0 \) [Tree filled]
     \( < 1 \)

8. \( \geq 0 \)
   TEST
   \( = 0 \)

9. \( < 0 \)
   TEST
   \( I = \text{NOB} \)

10. Passive network branches are now being considered

11. The list of network branches are still being examined for voltage sources

12. All voltage sources have been chosen. The list of network branches are again examined and the necessary number of passive branches selected.
A passive branch has been tentatively accepted for the tree. It is now necessary to go to the path finding subroutine TREP to see if the branch just selected will form a loop.

CALL TREP

The passive branch just selected for the tree must now be entered into the routing matrix.

Continue

Return
Subroutine TREP(NIN, NOUT, NP, NPL)

Given the routing matrix for the network tree with input and output nodes specified, this subroutine finds a node list representing the path between these two nodes.

PRELIMINARY
Set
- JX(I5) = 0
- NP(I5) = 0
- JMEM(I5) = 0
- KMEM(I5) = 0

NPL = 0, JX(1) = JX(2) = NIN,
I = 1, J = NIN, NP(1) = NIN

20

K = 0

25

K = K + 1

[Path finished]

TEST
NF(J, K) = NOUT ≠ 0

50

TEST
NF(J, K) = 0

100

J - NIN ≠ 0

Stop no path found

60

[Backstep]

25

≥ 0

[Flower check]

NF(J, K) - JX(I) = 0

45

[Flower formed]

≠ 0

60
45

Store and remember vertex
I=I+1
NP(I)=NP(J,K)
JMEM(I)=J
JX(I+1)=NF(J,K)
J=NF(J,K)
KMEM(I)=K

25

Backstep
J=JMEM(I)
K=KMEM(I)
I=I-1

50

Final path vertex and path length
NP(I+1)=NOUT
NPL=I+1

60

Return

100
Subroutine ARRAY(JSIG,XCON,JXPO,JKOD,POLY,LIL,KIK)

This subroutine takes the constant associated with each loop or non-touching combination of loops in the SFG and stores it in the matrix POLY. It does this by comparing the code and exponent of the given loop combination with the codes and exponents assigned to the columns and rows of POLY respectively.

```
If MMX=0
  NNX=0
  TEST KIK-1
  If < 0
    22
  If > 0
    DO 2 MMX=1, KIK-1
    MMX=MMX+1
    TEST JXPO-MSORT(IS4)
    If # 0
      10
      2 [End of DO loop]
      Continue
    If 0
      MSORT(KIK)=JXPO
      MMX=KIK
      KIK=KIK+1
    22
```
DO 12 NN=1,LIL-1

NNX=NNX-1

= 0

TEST JKOD=KSORT(NN)

= 0

End of DO loop

Continue

KSORT(LIL)=JKOD
NNX=LIL
LIL=LIL+1

POLY(MMX,NNX)=POLY(MMX,NNX) +XCON*(-1.)**JSIG

Return
Subroutine DECODE(KOO,KODY,IZ,FB,JZ,SEMBOL,KODF,KODI,ITOP,KBASIS)

This program decodes the composite symbol codes.

IZ=0
M=KBASIS-1

DO 3 J=1,KOO

IPOWER=M.AND.KODY

TEST IPOWER

= 0

3

≠ 0

KODY contains code for SEMBOL(J)

COMPARE SEMBOL(J) to FB

. EQ.

The term having code KODY belongs in numerator of transfer function

. NE.

IZ=IZ+1
KODF(IZ)=IPOWER
KODI(IZ)=J

ITOP(JZ)=1

increment J

KODY=KODY/KBASIS

3 [End of DO loop]
IV-3. Program Listing

***SNAP***

THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK

C CHARACTERISTIC NAM (DEFINED IN PROGRAM MAIN-1)
DIMENSION LT(35), LG(35), SMBOI(35)
DIMENSION ILST(35), NP(35), KODES(35), KONC(35)
DIMENSION N(35,35), CONS(35,35), KONE(35,35), IAPO(35,35)

C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK

C CHARACTERISTIC NAM
DIMENSION N1HST(100), NLAST(100), IXPON(100), WGT(100)
DIMENSION SYBMUL(100), Mix(100), CVAL(100)
DIMENSION KONS(100), NEST(100), TYPE(100)

C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK

C CHARACTERISTIC NAM
DIMENSION C(300), NPLT(300), XPOT(300), IAPO(300)
DIMENSION N(LTU(300), NUP(300), JAC(300)
DIMENSION NDCODE(300)

C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK

C CHARACTERISTIC NAM
DIMENSION NA(150), NR(150)
DIMENSION KNUV(20), KOU(20), SEMBOL(20), KUUF(20)
DIMENSION MSORT(15), KSORT(150), POLY(15,150)
DIMENSION POLY(15,150), ITO(150)
DIMENSION SIMHON(150,10), SIMHOO(150,10)
DIMENSION SEMPO(150,10), SEMPOU(150,10)
The following arrays are associated with the network:

- Dimension: `iset(15,100)`, `notch(1200)`, `star(9)`

```c
COMMON SEMPN, SEMPOD, PULY
COMMON/C1/MSORT, KSORT
COMMON/C2/NGS, NBG
COMMON/C3/NEPS, NTO
COMMON/C4/NSPT
EQUIVALENCE (1APU(1,1), NOTCH(1), SIMSON(1,1))
EQUIVALENCE (CONS(1,1), ISET(1,1), SIMOD(1,1))
EQUIVALENCE (KOU(1,1), PULYU(1,1))
DATA NASH/2H /
DATA F$H/3H F$H/3H /
DATA STAR(1), STAR(2), STAR(3)/3H, 3H*2, 3H*3/
DATA STAR(4), STAR(5), STAR(6), 3H*4, 3H*5, 3H*6/
DATA STAR(7), STAR(8), STAR(9), 3H*7, 3H*8, 3H*9/
DATA ONE/3H /
```

**Program Main**

**Preliminary Input Information**

- `nbn`=number of branches in network
- `nbg`=number of branches of SFG
- `nto`=number of terms in output
- `nspt`=number of symbols per term
- `neps`=number of different powers of S
- `npac`=number of paths plus circuits
- `nri`=maximum number of non-touching loops
- `nci`=maximum number of loops not touching any given loop
- `neon`=number of non-touching pairs of loops
- `nws`=number of repeated symbols (number of network elements assigned same symbol)

```
NBN=35
NFG=100
NTO=150
NSPT=20
NEPS=15
NPAC=300
NRI=15
NCI=100
NEON=1200
NPS=9
```

- `nsptu`=number of symbols in numerator of each term
- `nrg`=number of branches in tree of SFG
- `nng`=number of nodes in SFG

```
NNG=NBN
NSPTU=NSPT/2
NRTG=NBN
CALL SECOND (TCOMP)
WRITE(6,160A) TCOMP
1600 FORMAT (6, 160A) TCOMP
```

1600 FORMAT (1X, 6E15.8) CUMULATION TIME IN SECONDS, F15.8/
111 CONTINUE
```
WRITE (6, 519)
519 FORMAT(1H1)
```
```
CALL SECOND (TSTART)
```
```
1600 FORMAT (1X, 6E15.8) CUMULATION TIME IN SECONDS, F15.8/
111 CONTINUE
```
READ(S,1150) (WEIGHT(J), J=1,72)
1150 FORMAT (72A1)
IF (EOF(S)) 11111, 11112
11111 STOP
11112 CONTINUE
WRITE (6,1160) (WEIGHT(J), J=1,71)
1160 FORMAT (1X,71A1//)
DO 1151 J=1,72
1151 WEIGHT(J)=0.
READ(S,1240) NOU,NOD,KHASIS,LISTG,LISTC,LISTP
1240 FORMAT (3I5,5X,3I1)
IF (KHASIS) 1357, 1358
1357 KHASIS=8
1358 CONTINUE
READ(S, 1) NIN,NOUT,NOUA,NODA
1 FORMAT (4I5)
WRITE(6,720) NOUT
720 FORMAT (2X,1GHNUMBER OF NODES=, I3)
WRITE(6,721) NIN
721 FORMAT (2X,1GHNUMBER OF BRANCHES=, I3)
IF (LISTG) 723, 724, 725
722 CONTINUE
C LIST SFG
723 IF (LISTG) 729, 725, 726
724 CONTINUE
C LIST ALL CIRCUITS
725 IF (LISTP) 729, 126, 127
726 WRITE(6,729) NOUT
727 CONTINUE
WRITE (6,729) NOUT
728 FORMAT (2X,3HELEMENT NUMBER ASSOCIATED WITH OUTPUT=, I3)
GO TO 1806
1804 CONTINUE
WRITE(6,730) NOUT
1802 CONTINUE
WRITE(6,731) NOUA
1801 CONTINUE
WRITE (6,731) NOUA
1300 CONTINUE
WRITE(S,732) NOUT, NODA
1301 FORMAT (2X,3HELEMENT NUMBER OF SOURCE=, I3)
WRITE (6,732) NOUA
1302 CONTINUE
WRITE (6,732) NOUA
1303 CONTINUE
WRITE (6,733) NOUA
1304 CONTINUE
WRITE(6,733) NOUA
1305 CONTINUE
WRITE (6,734) NOUA
1306 CONTINUE
WRITE (6,734) NOUA
1307 FORM T (S,22PHASE FOR SYMBOL CODES=, I4)

C PROGRAM MAIN--2
C TAKE SFG BRANCH INFORMATION AS FOUND
C BY SUBROUTINE SFG AND GENERATE
C (1) ROUTING MATRIX INFORMATION
C N(J,K), AND L(J)
C (2) SFG BRANCH VALUES IXPO(J,L), CONS(J,L),
C KODE(J,L) WHERE J=FIRST(I), L=LAST(I), AND
C I=BRANCH NUMBER
C TOGETHER WITH THE SYMBOL SEMbol(K), K=1,2,...,M1
C CALL SUBROUTINE TO FORMULATE THE SIGNAL FLOW GRAPH, SFG
CALL SFG (FIRST,NLAST,IXPON,WEIGT,SYMBOl,KONSO,MIX,LIST, LISTG, LISTC, LISTP, NIN, NOUT, NOU, NOA, NOB)
IF(NOR)=1111,1111,1920

1920 CONTINUE
CALL SECOND(T2)
WRITE (6,93A)
TSFG=T2-TSTART
WRITE (6,1602) TSFG

1602 FORMAT(I1X,2/"TIME FOR FORMULATING SIGNAL/
122H FLOW GRAPH IN SECONDS;F15.3/)}
IHO=0
K0=0
MIC=1
K=0
MG=1
JLAS=1
NCIR=1
ININ=NIN
INOUT=NOUt
KOO=0
DO 301 INK=INK+NSPT

301 KON=INK)
C 300 IG(INK)=0

300 IG(INK)=0

C FIND IXPO(J,L),CONS(J,L)
C G0 TO 307
305 MG=MBASIS*MG
MIC=MIC+1
307 IH0=IHO+1
IF(LIST(IHO))14,4,4
4 CONTINUE
LOB=MIX(IHO)
J=NFIRST(LOB)
L=LAST(LOB)
IXPO(J,L)=IXPO(LOB)
CONS(J,L)=WEIGHT(LOB)

C FIND ROUTING MATRIX
8 IF(J.EQ.JLAM) GO TO 10
LT(JLAS)=K
K1=K+1
IF(JLAS=NIN)24,27,28
27 N(JLAS,K1)=1
GO TO 29
28 N(JLAS,K1)=0
29 JLAS=JLAS+1
K=0
GO TO 8
10 K=K+1
N(J,K)=L

C FIND KOOE(J,L) AND SEMBOL(K00)
SMHOL(IHO)=SMHOL(LOB)
MODE=MODE(LOB)
IF(MODE)=335,316,335
335 IF(I<10)5+g60+5
5 CODE(J,L)=IG(L)
GO TO 307
960 CONTINUE
KPU=IHO-1
IF(KPU)=953,953,315
315 DO 952 KP=1,KPU

PROGRAM MAIN--3
NULL CERTAIN ARRAYS, SET COUNTERS, AND DEFINE
A CODE FOR EACH NODE OF THE SGU

MPL=0
KTK=1
LIL=1
DO 601 KAM=i,NEXPS
DO 601 KIM=i,NTO
601 POLY(KAM,KIM)=0
DO 602 KP1=1,NEXPS
602 MSORT(KP1)=0
DO 950 K02=1,NSPT
950 KODI(K02)=0
DO 603 KP2=1,NTO
603 KSORT(KP2)=0

IR=1
NFIR=1
KNO=0
KODES(1)=1
DO 2000 JS=2,NNG
2000 KODES(JS)=2,KODES(JS-1)
IF(LISTP)175,175,1116
1116 WRITE(6,170)ININ,NOUT
170 FORMAT(17H GATHS FROM NOVE 12.9H TO NOUE 127/) WRITE(6,1905)
1905 FORMAT(5X,17HNOE, NODE LIST)
175 CONTINUE
IF(LISTP)1113,1113,23
1113 K3=LT(NIN)+1
N(NIN+K3)=0
K2=LT(1)+1
N(1,K2)=-1
NIN=1
NOUT=1
KLAS=0
24 NFIR=0
PROGRAM MAIN

PATH-FINDING ALGORITHM
IN ADDITION, STEP PF7 CALCULATES THE COMPOSITE CODE, CONSTANT, AND EXPONENT OF THE PATH.

C

PF1 (PRELIMINARY)

DO 1112 IZ0=1,NNG
1112 IFLOW(IZ0)=0
   DO 31 NI=1,NNG
   KONC(I)=1
   NOP=KLAS
   KLAS=0
   23 I=2
   J=NIN
   NP(I)=NIN
   IFLOW(NIN)=1
   IFLOW(NOUT)=1

C

25 K=KONC(J)

C

PF2 (FINishes NEXT NODE)

NP(I)=N(J,K)

C

PF3 (TEST ROUTING MATRIX)

IF(N(J,K))>100,60,34

C

PF4 (TEST FOR FLOWER)

34 NJK=N(J,K)
   IF(IFLOW(NJK))>50,38,26
   26 KONC(J)=KONC(J)+1
   GO TO 25

C

PF5 (PREPARE FOR NEXT NODE)

38 J=NP(I)
   IFLOW(J)=1
   I=I+1
   GO TO 25

C

PF6 (BACKSTEP)

60 IFLOW(J)=0
   KONC(J)=1
   J=NP(I-2)
   KONC(J)=KONC(J)+1
   I=I-1
   GO TO 25

C

PF7 (FINISH PATH)

50 KONC(J)=KONC(J)+1
   KLAS=KLAS+1

C

FIND CODE FOR NODE PATH
program main=-5
modify the spa by removing every branch connected
to the node through which all circuits have just
been found

100 t3=0.
if(ncir=1)2610,102,210
102 continue
if(nfih=1)104,2010,104
103 k4=lt(nin)+1
n(nin,k4)=0
k5=lt(1)+1
n(1,k5)=-1
nin=1
nout=1
go to 24

write(b,1362)
format(1x,3y,n0,of circuits exceeds limit-increase,
126 dimensions containing npac)
1361 continue
go to 25
104 IF(NIN-JLAS)105,200,200
105 NIN=J+1
   NOUT=N+1
   KONC(J)=1
   NY=LT(J)+1
   N(J,NY)=0
   DO 109 JC=1,N,JLAS
      LCOL=LT(JC)
      IF(LCOL.EQ.0) GO TO 109
      IF(N(JC,LCOL)-J)109,107,109
107 N(JC,LCOL)=0
   LT(JC)=LT(JC)-1
109 CONTINUE
   NZ=LT(NIN)+1
   N(NIN,NZ)=-1
   NOUT=NIN
   GO TO 23
2010 CALL SECON(N(T3)
   TPATH=T3-T2
   WRITE(6,202)NO,NOP,TPATH
202 FORMAT(6,2012)NOC,TPATH
2012 FORMAT(6,2012)NOC,TIME FOR FINDING,110,17H PATHS; IN SECONDS,F15.3/
   IF(NCIR-1)250,103,250
200 CONTINUE
   NOL=KLA5
   CALL SECON(N(T4)
   IF(T3)2014,2020,2014
2014 TCI=2014,T3
   GO TO 2016
2020 TCI=2016,T2
2016 WRITE(6,160)NO,TCH
160 FORMAT(6,160)NOC,TCH
1603 FORMAT(6,1603)NOC,TIME FOR FINDING,110,18H FIRST ORDER LOOPS; /
   11H IN SECONDS,F15.3/
C PROGRAM MAIN--6
C FIND SECON ORDER LOOPS

   NOL=KLAS
   KMOL=0
   DO 257 KOW=1,NPAC
257 NOCTOT(KOW)=0
   LOW1=NOX+1
   NOC=0
   NOL1=NOL-1
   DO 203 LIR1=LOW1,NOL1
      LOW2=LOW1+1
   DO 202 LIR2=LOW2,NOL
      NAN=NPCODE(LIR1) .AND. NPCODE(LIR2)
      IF(NAN)202,201,202
201 CONTINUE
   TCONS2=CONST(LIR1) .AND. TCONS(LIR2)
   KXPOZ=1XPOX(LIR1)+1XPOX(LIR2)
   KSYM2=KSOFT(LIR1) .AND. KSOFT(LIR2)
   CALL ARRAY(2,TCONS2,KXPOZ,KSYM2,POLY,L1L,KIK)
   KMOL=KMOL+1
   NOC=NOX+1
   IF(NOC=NEUN)1395,1396,1395
1395 WRITE(6,1397)
1397 FORMAT(5,46,JHINCREASE NEUN--THE DIMENSION OF THE ARRAY NOTCH)
1396 CONTINUE
NOTCH(NOC)=LH2
202 CONTINUE
203 NOCTOT(LH1)=NOC
NOCTOT(NOL)=NOC

C PROGRAM MAIN--7
C FIND ALL LOOPS OF ORDER
C GREATER THAN 2

C GENERATE THE FIRST ROW OF ISET
NIPL=NOP+1
KAPMAX=1
INK0=1
DO 1170 ISC=NIPL*NOL
INK1=NOCTOT(ISC)
IF (ISC=1) 1171,1172
1172 INK2=NOCTOT(ISC-1)+1
GO TO 1173
1171 INK2=1
1173 IF (INK1=INK2=INK0) 1170,1170,1175
1175 INK0=INK1=INK2
1170 CONTINUE
IF (INK0=NC1) 1391,1391,1390
1390 WRITE(*,1392) INK0
1392 FORMAT(1X,52HINCREASE NC1--THE NO. OF COLUMNS IN DIMENSION OF ISET)
1391 CONTINUE
DO 490 NIP=NIPL*NOL
INKU=NOCTOT(NIP)
IF (NIP=1) 211,211
211 INKL=NOCTOT(NIP-1)+1
GO TO 212
210 INKL=1
212 CONTINUE
IF (INKU=INK1) 490,490,410
410 JIP=0
DO 480 INK=INKL,INKU
JIP=JIP+1
480 ISET(JIP,JIP)=NOTCH(INK)
MAPO(NIP)=INKU-INKL+1
C
C INITIATE PROCESS FOR FINDING
C HIGHER ORDER LOOPS
DO 430 KAT=1,NPAC
JAC(KAT)=0
430 NUP(KAT)=0
JAC(1)=MAPO(NIP)
KAP=2
440 KAP=KAP+1
IF (KAP) 490,490,429
429 KAP=KAP+1
IF (KAP=NHI) 1351,1351,1350
1350 CONTINUE
429 KAP=KAP+1
JAC(KAP)=0
NUP(KAP)=NUP(KAP)+1
C מסוים

C LABEL Loop of FIRST CKT
NAP=NUPI(AAPI)
IF (KAPMAX= KAP) 1347, 1348, 1348
1347 KAPMAX=KAP
1348 CONTINUE
ISAT=ISET(KAP, NAP)
C
C TEST Loop of REMAINING CKTS
MAPU=JAC(KAP)
M API=NUPI(KAP) + 1
DO 435 MAPI=MAPI+MAPU
ISOT=ISET(KAP, MAPI)
KAN=NPCUE(ISAT) .AND. NPCODE(ISOT)
IF (KAN) 435, 455, 435
455 CONTINUE
C
C WRITE
TCONSG=CONST(NIP)
K XPOG=IXPOT(NIP)
KSYM=KODET (NIP)
DO 477 LP0=1 KAP
ITIC=NUPI(LPU)
ITUCH=ISET (LP0, ITIC)
TCONSG=TCONSG*CONST (ITUCH)
KXPOG=KXPOG*IPOT (ITUCH)
477 KSYM=KSYM*KODET (ITUCH)
TCONSG=TCONSG*CONST (ISOT)
KXPOG=KXPOG*IPOT (ISOT)
KSYM=KSYM* KODET (ISOT)
KAPP=KAP+2
CALL ARHAY(KAPP, TCONSG, KXPOG, KSYM, POLY, LIL, KIK)
KHOL=KHOL+1
C SET COUNTERS
423 KAPI=KAP+1
JAC(KAPI)=JAC(KAPI)+1
JACK=JAC(KAPI)
ISET(KAPI, JACK)=ISET(KAP, MAPI)
435 CONTINUE
JACK=JAC(KAPI)
IF (JACK=2) 425, 431, 425
431 IF (JAC(KAPI)=NUPI(KAP) - 1) 440, 440, 429
490 CONTINUE
CALL ARHAY (2, 1, 0, 0, POLY, LIL, KIK)
CALL SECOND (15)
TNLT=T5-T4
WRITE (6, 1694) KHOL, TNLT
1604 FORMAT (1X, 18, (TIME FOR FINDING ,110, RH SETS OF /
130H NONTOUCHING LOOPS, IN SECONDS, F15.3/)
C
C PROGRAM MAIN--R
C DECODE COMPOSITE SYMBOL CODE
C AND ISOLATE SYMBOLS FROM
C INVERSE SYMBOLS

NANU=LIL=1
DO 691 J1=1, NEXPS
DO 691 J2=1, NTO
691 POLYU(J1, J2) = 0
DO 693 J1=1, NTO
693
DO 693 J2=1,NSPTU
SEPON(J1,J2)=STAR(1)
SEMP00(J1,J2)=STAR(1)
SIMB0N(J1,J2)=SH
693 SIMBO(J1,J2)=SH
DO 951 J4=1,NT0
NA(J4)=0
951 NRJ4)=0
C
C DECODE KS0RT(J7) AND RECPNU TEAMS
CONTAINING FEEXBACK SYMBOL #F3#
JZU=1#1-1
DO 646 JZ=1,JZU
KODY=K0DI(JZ)
ITP(JZ)=0
IF(KOY)715,646,715
715 CALL DECODE(K00,KODY,JZ,FH,JZ,SEMPOL,K0DF,K0UJ,ITP,KHAST)
C
C ISOLATE NUM. SYMBOLS AND INVERSE SYMBOLS
OF KSORT(JZ)
637 NAK=0
NAT=0
IF(JZ)646,646,647
647 CONTINUE
DO 645 NZ=1,JZ
KOZY=K0DI(NZ)
IARG=K0DF(NZ)
IF(IARG=NHS)1340,1340,1341
1341 WRITE(6,1342)
1342 FORMAT(1X,30HINCREASE THE DIMENSION OF STAR)
1340 CONTINUE
IF(KONS(K0ZY))657,657,659
657 NAK=NAK+1
IF(NAK=NSPTU)11376,1375,1375
1375 WRITE(6,1377)
1377 FORMAT(1X,40HNSPT EXCEEDS LIMIT INCREASE DIMENSIONS ,
NSPT CONTAINING NSPT)
1376 CONTINUE
SIMB0N(JZ,NAK)=SEMP00(KOZY)
SEPON(JZ,NAK)=STAR(IARG)
NA(JZ)=NA(J7)+1
GO TO 645
659 NAT=NAT+1
IF(NAT=NSPTU-1)1381,1380,1380
1380 WRITE(6,1382)
1382 FORMAT(1X,40HNSPT EXCEEDS LIMIT INCREASE DIMENSIONS ,
NSPT CONTAINING NSPT)
1381 CONTINUE
SIMB0D(JZ,NAT)=SEMP00(KOZY)
SEP00(JZ,NAT)=STAR(IARG)
NB(JZ)=NB(J7)+1
645 CONTINUE
646 CONTINUE
C
C PROGRAM MAIN--9
C SEPARATE POLY INTU ARRAYS FOR THE
C NUMERATOR AND DENOMINATOR OF THE
C TRANSFER FUNCTION
THE CONSTANT COEFFICIENTS IN THE TRANSFER FUNCTION
ARE SEPARATED INTO ARRAYS FOR THE NUMERATOR
AND DENOMINATOR

KIKA=1
DO 755 JA=1,KIKA
JIRA=0
JOA=0
DO 755 JC=1,KIKA
IF(ITOP(JC))753,753,751
751 JIR=JIR+1
POLYU(JA,JIR)=POLY(JA,JC)
GO TO 755
753 JO=J0+1
POLY(JA,JO)=POLY(JA,JC)
755 CONTINUE
CALL SECONDIT6)
TDECO2=T6-T5
WRITE (6,1605) TDECOD
1605 FORMAT (1X,*6HTIME FOR DECODING SYMBOLS IN SECONDS,F15.3/)

PROGRAM MAIN--10
MAKE POWERS OF S IN OUTPUT
TRANSFER FUNCTION POSITIVE

MAXIM=0
KARU=KIKA=1
DO 522 KAR=1,KARU
IF(MSOT(KAR))521,522,522
521 MAXIM=MAXIM+MSOT(KAR)
523 MAXIM=MAXIM+MSOT(KAR)
522 CONTINUE
DO 524 K=1,KARU
524 MSOT(K)=MAXIM+MSOT(K)

PROGRAM MAIN--11
PRINT OUT NUMERATOR OF THE TRANSFER FUNCTION

LUK=0
IKU=LIL=1
WRITE (6,931)
WRITE (6,936)
WRITE (6,92A)
920 FORMAT (25X*20HNUMERATOR POLYNOMIAL//)
WRITE (6,921)
921 FORMAT (1X,9HCOLUMN,12X,9HSYMBOL FOR GIVEN COLUMN)
DO 905 IK=1,IKU
IF(ITOP(IK));905,905,901
901 ILU=NA(IK)
IF(1LU)710,710,711
710 ILU=1
711 JLU=NB(IK)
IF(JLU)712,712,713
712 JLU=1
713 CONTINUE
LUK=LUK+1
WRITE (6,902) LUK,(SIMBON(IK,IL),SEMPON(IK,IL),
1  IL=1, ITL), DASH, (SIMBOU (IK, JU), SEMPOU (IK, JU), JU = 1, JLU)  
903 FORMAT (9X, 15*20X, 30A3)  
905 CONTINUE  
WRITE (6, 930)  
930 FORMAT (/)  
WRITE (6, 1H21)  
1A21 FORMAT (1X, 7H POWER)  
WRITE (6, 925)  
422 FORMAT (1X, 8H UF S 17X, 33H CONSTANT COEFS. IN THE POLYNOMIAL)  
LML = 1  
LMU = 7  
IF (JIH = LMU) 820, 818, B18  
820 LMU = JIH  
818 WRITE (6, 806) (LO, LD = LML, LMU)  
806 FORMAT (2X, 8H COLUMN, 12)  
KROWU = IK = 1  
DO 808 KROW = 1, KHOWU  
WRITE (6, 812) MSRT (KROW), (POLY (KROW, LM), LM = LML, LMU)  
810 FORMAT (15, 9H 17(E12.5))  
808 CONTINUE  
IF (JIH = LMU) 814, 814, 812  
812 LML = LML + 7  
LMU = LMU + 7  
IF (JIH = LMU) 816, 818, 818  
816 LMU = JIH  
GO TO 818  
A14 CONTINUE

C PROGRAM MAIN-12  
C PRINT OUT DENOMINATOR OF  
C THE TRANSFER FUNCTION

LUK = 0  
IKU = IL = 1  
WRITE (6, 930)  
931 FORMAT (50H*********************************************************************************)  
WRITE (6, 930)  
WRITE (6, 925)  
923 FORMAT (25X, 22H DENOMINATOR POLYNOMIAL///)  
WHITE (6, 924)  
924 FORMAT (1X, 6H COLUMN, 12X, 23H SYMBOL FOR GIVEN COLUMN)  
DO 705 IK = 1, IKU  
IF (ITOP (IK), 701, 701, 705  
701 IK = NA (IK)  
LUK = LUK + 1  
IF (IKU = 915, 915, 916  
915 IKU = 1  
416 JLU = NA (IK)  
IF (JLU = 917, 917, 918  
917 JLU = 1  
918 CONTINUE  
WRITE (6, 703) LUK, (SIMBON (IK, IL), SEMPON (IK, IL),  
1IL = 1, ILU), DASH, (SIMBON (IK, JU), SEMPON (IK, JU), JU = 1, JLU)  
703 FORMAT (1X, 15*20X, 30A3)  
705 CONTINUE  
WRITE (6, 930)  
WRITE (6, 1H22)  
1822 FORMAT (1X, 7H POWER)  
WRITE (6, 925)
925 FORMAT(1X,8H OF S 17X,33H CONSTANT COEFS. IN THE POLYNOMIAL)
LML=1
LMU=7
IF(JD-LMU)520,518,518
520 LMU=JD
518 WRITE (6,506) (L0+LD=LML+LMU)
506 FORMAT(2X,7(8X,6H COLUMN,12))
KROW=KI-K-1
DO 508 KROW=1,KROW
WRITE (6,511) M(ROW)=POLY(KROW+LM)+LM=LML+LMU
510 FORMAT(15,3H 7(E12.5))
508 CONTINUE
IF(JD-LMU)520,518,518
520 LMU=LMU+7
512 LMU=LMU+7
508 CONTINUE
IF(JD-LMU)514,514,512
512 LMU=LMU+7
510 CONTINUE
GO TO 518
WRITE (6,930)
CALL SECOND(TEND)
TEXEC=TEND-START
WRITE (6,1101) TEXEC
1101 FORMAT(1X,7H EXECUTION TIME IN SECONDS 15.3//
128H AUGUST 1970 VERSION OF SNAP)
250 GO TO 1111
END
SUBROUTINE SFN(NFIRST,NLAST,IXPON,WEIGT,SYMUL,KON,MTX,NEST,
LIST,NIN,NOUT,NUD,NOH,LISTG,NUA,NUR)
C********************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NBG(DEFINED IN PROGRAM MAIN-1)
DIMENSION JQ(35),NP(35),IVV(35),NUM(35),ICV(35),INTPE(35)
DIMENSION LNAC(35)
DIMENSION NF(35,35),LB(35,35),NS(35,35)
DIMENSION TPX(35),JM(35),LB(35),MSYM(35)
DIMENSION IUAL(35),VAL(35),SYM(35)
DIMENSION IUALX(35),VALX(35),NUMXX(35),INTPE(35),NOFRF(35)
DIMENSION TPX(35),NUM(35),JMX(35),IHX(35),SYM(35)
C********************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NBG
DIMENSION CV(100)
DIMENSION NF(100),NL(100),IX(100),WEIGT(100)
DIMENSION MAPY(100),KON(100),NEST(100),TYPE(100)
DIMENSION SYMUL(100),MIX(100)
C********************************************************************
COMMON NF,NS,IB
COMMON/C2/NBG
C
C SUBPROGRAM #A#
DATA Y,GC,CTUHRC,2HY,2MG,2HC,1H=2MR,2HL,2HZ/
DATA E,CT,CCCV,VC,2HL,2HI,2HCC,2MV,2HV,2VHC/
DATA FB/3H FB/
DATA ONE/3H 1/
DO 710 IC=1,NNB
DO 710 IK=1,NNG
IB=0
710 NF(IC,IK)=0
LINK=0
DO 152 IG=1,NRG
NEST(IG)=0
152 KONO(IG)=0
WRITE (6,266)
260 FORMAT (/)
WRITE(6,717)
717 FORMAT(30X*THNETWORK/)
NLAST(I)=NIN
IXPON(I)=0
WEIGT(I)=-1.
SYMUL(I)=FU
KONSO(I)=0
NEST(I)=1
MO=0
LO=0
LIST=1
DO 5 II=1,NMG
NJ=NEST(I)
JL=KONSO(I)
LINK=O
DO 152 IG=1,NRG
NEST(IG)=NJ
WRITE (6,266)
260 FORMAT (30X*13HTREE SELECTED)
NUMU=NOD-1
10=INTRE(NUMU)
NUMC=NUMX(IO)
TYPB(NUMC)=TYPX(IO)
JB(NUMC)=JBX(IO)
LB(NUMC)=LBX(IO)
SYM(NUMC)=SYMX(IO)
IQAL(NUMC) = IQAUX(IO)
VAL(NUMC) = VAXX(IO)
NUML(NUMC) = NULX(IO)
INTREE(NUMC) = 1
WRITE(6, 517) TYPB(NUMC) * NUMC, JH(NUMC), LB(NUMC), SYM(NUMC),
IQAL(NUMC), VAL(NUMC), NUML(NUMC)

517 FORMAT (4X, A2, 6X, (12, 6X, 12, 6X, 12, 6X, 6X, A3, A1, E12.5, 2X, 12)
KLU = KLU + 1
LINC(NUMC) = 0
IF (TYPB(NUMC) .NE. VV) GO TO 3
M0 = M0 + 1
IVV(M0) = NUMC
3 IF (TYPB(NUMC) .NE. CV) GO TO 4
L0 = L0 + 1
ICV(L0) = NUMC
4 JF = JH(NUMC)
LF = LB(NUMC)
IR(JF, LF) = NULC
IR(LF, JF) = NULC
JROW(JF) = JROW(JF) + 1
JRO(JF) = JROW(JF)
NF(JF, JRO(JF)) = LF
NS(JF, LF) = 1
JROW(LF) = JROW(LF) + 1
JR(LF) = JROW(LF)
NF(LF, JR(LF)) = JF
NS(LF, JF) = -1
21 CONTINUE
WRITE(6, 260)
WRITE(6, 715)
715 FORMAT (30X, 9HSFG/)
DO 13 ILL = 1, N00
JRO(J) = JROW(ILL) + 1
13 NF(ILL, JRO(J)) = 0
C
C          SUBPROGRAM #C#
C          THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
C          FROM BRANCH NODE TO LINK NODE
C          NOB=NOB
C
151 CONTINUE
NES = 0
LON = 0
IF (KLU = NOB) = 32, 360, 532
532 LINK = LINK + 1
IF (NOTREELINK) = 534, 534, 532
534 NUMC = NUMX(LINK)
TYPE(NUMC) = TYPB(LINK)
JK = JH(X(LINK))
LK = LBX(LINK)
SYM(NUMC) = SYM(X(LINK))
IQAL(NUMC) = IQALX(LINK)
CVAL(NUMC) = VALZ(LINK)
NUMB = NUMLX(LINK)
TYP2 = TYPB(NUMC)
CVALU = CVAL(NUMC)
KLU = KLU + 1
LINC(NUMC) = 1
KDEP = 0
KANS = 0
IF (TYPE(NUMC) .EQ. CL) GO TO 117
IF (TYPE(NUMC) .EQ. C) GO TO 119
IF (TYPE (NUMC) .EQ. Y) GO TO 119
IF (TYPE (NUMC) .EQ. R) GO TO 120
IF (TYPE (NUMC) .EQ. Z) GO TO 120
IF (TYPE (NUMC) .EQ. C) GO TO 121
KDEPS = 1
IF (TYPE (NUMC) .EQ. E) GO TO 123
IF (TYPE (NUMC) .EQ. C1) GO TO 123
IF (TYPE (NUMC) .EQ. VC) GO TO 165
IF (TYPE (NUMC) .EQ. CC) GO TO 265
117 IXPS = -1
KANSO = 1
GO TO 123
119 IXPS = 0
GO TO 123
700 IXPS = 0
KANSO = 1
GO TO 123
121 IXPS = 1
123 CALL TREP (JK, LK, NF, NP, NPL)
IN=NUMC
149 LON=LON+1
NP1=NP(LON)
NP2=NP(LON+1)
107 INIT=IF(NP1, NP2)
109 SIGN=NS(NP1, NP2)
IF (KEPS) 167, 167, 169
167 IF (EQUAL (NUMC) .EQ. 10) GO TO 111
NES = 1
CONST=SIGN
GO TO 125
111 CONST=SIGN*CALU
125 LIST=LIST+1
IF (NES) 502, 503, 502
502 NFST(LIST)=
503 KONS(LIST) = KANSO
NFIRST(LIST) = INIT
NLAST(LIST) = IN=
SYMUL(LIST) = SYM(IFN)
IXPON(LIST)=IXPS
IF (KONS(LIST)) 505, 505, 504
504 WEGT(LIST)=1./CONST
GO TO 506
505 WEGT(LIST)=CONST
506 MAPY(NUMC)= 'LIST
127 FORMAT (315, E12.5)
129 FORMAT (A4)
C C SUBPROGRAM #1!
C THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
C FROM LINK NODE TO BRANCH NODE
169 CONTINUE
IF (TYPE(INIT) .EQ. E) GO TO 201
IF (TYPE(INIT) .EQ. C1) GO TO 201
IF (TYPE(INIT) .EQ. VV) GO TO 201
IF (TYPE(INIT) .EQ. CV) GO TO 201
LIST=LIST+1
IF (TYPE(INIT) .EQ. R) GO TO 133
IF (TYPE(INIT) .EQ. Z) GO TO 133
IF (TYPE(INIT) .EQ. G) GO TO 702
IF (TYPE(INIT) .EQ. Y) GO TO 702
IF (TYPE(INIT) .EQ. CL) GO TO 135
IF (TYPH(INIT).EQ.0) GO TO 137
133 IXPON(LIST) = 0
GO TO 141
702 IXPON(LIST) = 0
KONSO(LIST) = 1
GO TO 141
135 IXPON(LIST) = 1
GO TO 141
137 IXPON(LIST) = -1
KONSO(LIST) = 1
141 IF (IQUAL(INIT).EQ.10) GO TO 139
NEST(LIST) = 1
WEIGT(LIST) = -1*SIGN
GO TO 147
139 IF (KONSO(LIST)) .NE. ,608,608,607
607 WEIGT(LIST) = -SIGN/VAL(INIT)
GO TO 147
608 WEIGT(LIST) = -SIGN*VAL(INIT)
147 NFIRST(LIST) = IFIN
NLAST(LIST) = INIT
SYMVL(LIST) = SYM(INIT)
201 NPLA = NPL - 1 - LUN
IF (NPLA) 151, 151, 149
C THIS PROGRAM SETS UP SFG INFO. FOR VC
C TYPE CONSOL SOURCES
165 NUMO = NUMH
IF (INTHEE(NUMH)) 163, 163, 161
163 LIST = LIST + 1
NFIRST(LIST) = NUMH
NOHY = NOHY + 1
NLAST(LIST) = NOHY
SYMVL(LIST) = SYM(NUMH)
NUMO = NOHY
IF (TYPE(NUMH).EQ.Y) GO TO 912
IF (TYPE(NUMH).EQ.6) GO TO 912
IF (TYPE(NUMH).EQ.C) GO TO 914
IF (TYPE(NUMH).EQ.CL) GO TO 916
NUMO = 0
IXPON(LIST) = 0
GO TO 918
912 IXPON(LIST) = 0
KUNO = 1
GO TO 918
914 IXPON(LIST) = -1
KUNO = 1
GO TO 918
916 IXPON(LIST) = 1
KUNO = 0
918 IF (IQUAL(NUMH).EQ.10) GO TO 920
NEST(LIST) = 1
WEIGT(LIST) = -1,
GO TO 209
920 IF (KUNO) 922, 922, 924
922 WEIGT(LIST) = CVAL(NUMH)
GO TO 209
924 WEIGT(LIST) = CVAL(NUMH)
209 KONSO(LIST) = 1
161 LIST = LIST + 1
NFIRST(LIST) = NUMO
C
C SUBPROGRAM #F#
C THIS PROGRAM SETS UP SFG INFO. FOR CC
C TYPE CONTROL SOURCES
265 MUNO=NUMH
620 IF(INTKEE(NUMH))GO TO 21
621 LIST=LIST+1
622 NFIRST(LIST)=NUMH
623 NOBY=NOBY+1
624 NLAST(LIST)=NOBY
625 SYMBUL(LIST)=SYM(NUMH)
626 MUNO=NOBY
627 IF(TYPH(NUMH),EQ.2)GO TO 233
628 IF(TYPH(NUMH),EQ.3)GO TO 233
629 IF(TYPH(NUMH),EQ.4)GO TO 235
6210 IF(TYPH(NUMH),EQ.5)GO TO 237
6211 KUNO=0
6212 IXPON(LIST)=0
6213 GO TO 241
6214 233 IXPON(LIST)=0
6215 KUNO=1
6216 GO TO 241
6217 235 IXPON(LIST)=1
6218 KUNO=1
6219 GO TO 241
6220 237 IXPON(LIST)=1
6221 KUNO=0
6222 IF(IQUAL(NUMH),EQ.9)GO TO 239
6223 NEST(LIST)=1
6224 WEIGHT(LIST)=1
6225 GO TO 247
6226 239 IF(KUNO)GO TO 200
6227 900 WEIGHT(LIST)=VAL(NUMH)
6228 GO TO 247
6229 902 WEIGHT(LIST)=VAL(NUMH)
6230 247 KUNO=KUNO+1.
6231 LIST=LIST+1
6232 NFIRST(LIST)=MUNO
6233 NLAST(LIST)=NUMH
6234 SYMBUL(LIST)=SYM(NUMH)
6235 IXPON(LIST)=0
6236 IF(IQUAL(NUMH),EQ.10)GO TO 271
6237 NEST(LIST)=1
6238 WEIGHT(LIST)=1
6239 GO TO 281
6240 271 WEIGHT(LIST)=VAL NUMH
6241 GO TO 281
6242 281 CONTINUE
6243 GO TO 123
C
C SUBPROGRAM #G#
NOBY=NUBY+1
NLAST(LIST)=NUBY
SYMBUL(LIST)=SYM(NUNO)
IF (TYPH(NUNO).EQ.Z) GO TO 433
IF (TYPH(NUNO).EQ.R) GO TO 433
IF (TYPH(NUNO).EQ.CL) GO TO 435
IF (TYPH(NUNO).EQ.C) GO TO 437
KUNO=0
IXPON(LIST)=0
GO TO 441
433 IXPON(LIST)=0
KUNO=1
GO TO 441
435 IXPON(LIST)=1
KUNO=1
GO TO 441
437 IXPON(LIST)=1
KUNO=0
441 IF (TUAL(NUNO).EQ.10) GO TO 439
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 448
439 IF (KUNO)=908,908,910
908 WEIGT(LIST)=VAL(NUNO)
GO TO 448
910 WEIGT(LIST)=VAL(NUNO)
448 KONSO(LIST)=1
447 CONTINUE
NUNO=NUBY
461 LIST=LIST+1
NIFIRST(LIST)=NUNO
NLAST(LIST)=LI
SYMBUL(LIST)=SYM(LI)
IXPON(LIST)=0
IF (IQUAL(LI,.EQ.10)) GO TO 471
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 403
471 WEIGT(LIST)=VAL(LI)
403 CONTINUE
405 CONTINUE
SUBPROGRAM F12
GENERATING OUTPUT NODE OF SFG
515 CONTINUE
IF (NOUT)=514,512,514
512 CALL THEP(NOUTA,NOUTB,NF,NP,NPL)
NOUT=NUBY+1
MOPU=NPL+1
DO 510 MOP=1,MOPU
N1=NP(MOP)
N2=NP(MOP+1)
LIST=LIST+1
NIFIRST(LIST)=IH(N1,N2)
NLAST(LIST)=NOUT
SYMBUL(LIST)=ONE
IXPON(LIST)=0
KONSO(LIST)=0
NEST(LIST)=0
510 WEIGT(LIST)=NS(N1,N2)
C THIS PROGRAM SETS UP SFG INFO. FOR CV
C TYPE CONTROL SOURCES
360 IF (LINC(NUNO) > 361) GO TO 364
364 DD 305 MI = 1, LO
   KI = ICV (MI)
   NUNO = NUML (KI)
IF (LINC(NUNO)) 361:361:363
363 LIST = LIST + 1
   NFIRST (LIST) = NUML (KI)
   NOBY = NOBY + 1
   NLAST (LIST) = NOBY
   SYMBUL (LIST) = SYM (NUNO)
IF (TYPE (NUNO) .EQ. Y) GO TO 333
IF (TYPE (NUNO) .EQ. G) GO TO 333
IF (TYPE (NUNO) .EQ. C) GO TO 335
IF (TYPE (NUNO) .EQ. CL) GO TO 337
   KUNO = 0
   IXPON (LIST) = 0
   GO TO 341
333 IXPON (LIST) = 0
   KUNO = 1
   GO TO 341
335 IXPON (LIST) = -1
   KUNO = 0
   GO TO 341
337 IXPON (LIST) = 1
   KUNO = 0
   GO TO 341
341 IF (IQUAL (NUNO) .EQ. IO) GO TO 339
   NEST (LIST) = 1
   WEGT (LIST) = 1.
   GO TO 348
339 IF (KUNO) 904, 904, 906
404 WEGT (LIST) = CVAL (NUNO)
   GO TO 348
906 WEGT (LIST) = 1./CVAL (NUNO)
348 KONSO (LIST) = 1
347 CONTINUE
   NUNO = NOBY
361 LIST = LIST + 1
   NFIRST (LIST) = NUNO
   NLAST (LIST) = KI
   SYMBUL (LIST) = SYM (KI)
   IXPON (LIST) = 0
IF (IQUAL (KI) .EQ. IO) GO TO 371
   NEST (LIST) = 1
   WEGT (LIST) = 1.
   GO TO 303
371 WEGT (LIST) = VAL (KI)
303 CONTINUE
305 CONTINUE
C C SUBPROGRAM $H$
C C THIS PROGRAM SETS UP SFG INFO. FOR CV
C TYPE CONTROL SOURCES
460 IF (LINC(NUNO)) 451, 451, 464
464 DD 405 MI = 1, LO
   LI = ICV (MI)
   NUNO = NUML (LI)
IF (LINC(NUNO)) 463:463:461
463 LIST = LIST + 1
   NFIRST (LIST) = NUML (LI)
THE FOLLOWING AKHAYS ARE ASSOCIATED WITH THE NETWORK

C CHARACTERISTICS NNH, AND NSPT (DEFINED IN PROGRAM MAIN-1)

DIMENSION TYPX(35), JBX(35), LMX(35), INTRE(35), NOTPFE(35)

DATA E*VV,CV/2HE,2HV++]HCV/
DATA R+CL,C,Y,2/2HR,2HC,2HY,2HZ/
DATA G/2H2G/
00 40 I2=1*NNG
00 40 I3=1*NNG
40 NF(I2,13)=0
   M=0
   K=0
   KCO=0
   DO 1 I=1*NNO
1   KCOL(I)=0
   DO 3 I7=1*NNO
3   NOTREE(I7)=n
   I=0
5   I=I+1
   IF(TYPX(I).*EU.*E) GO TO 10
   IF(TYPX(I).*E.*V.*V) GO TO 10
   IF(TYPX(I).*E.*C.*V) GO TO 10
10   K=K+1
14   INTRE(K)=I
   JAX(I)=JBX(I)
   KCOL(JAX(I))=KCOL(JBX(I))+1
   KCOL1=KCOL(JBX(I))
   NF(JBX(I),KCOL(I))=LHX(I)
   IBX(I)=LHX(I)
   KCOL(IHX(I))=KCOL(IBX(I))+1
   KCOL2=KCOL(IBX(I))
   NF(IHX(I),KCOL2)=JAX(I)
   NOTREE(I)=-1
   IF(K-NOU+1)Z922.22
2   IF(M)4,4,12
4   IF(I-NOH)6,12
12   M=M+1
   IF(TYPX(M).*E.*H) GO TO 16
   IF(TYPX(M).*E.*G) GO TO 16
16   NF(I,J)2,22.22
   IF(M)4,4,12
17   IF(TYPX(M).*E.*C) GO TO 16
18   IF(TYPX(M).*E.*C) GO TO 16
19   IF(TYPX(M).*E.*Y) GO TO 16
20   IF(TYPX(M).*E.*Z) GO TO 16
21   IF(I-NOH)12,22.22
16   NINX=JBX(M)
   NOUTX=LHX(M)
   CALL TREP(NINX,NOUTX,NF,NP,NPL)
   IF(NPL)21,21,12
21   I=M
   GO TO 10
22   CONTINUE
   RETURN
END
SUBROUTINE TREP(NIN,NOUT,NF,NP,NPL)
C******************************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NFIWORK
C CHARACTERISTIC NFI (DEFINED IN PROGRAM MAIN=1)
DIMENSION JX(35),NP(35),JIME(35),KME(35)
DIMENSION NF(35,35)
C******************************************************************************
COMMON/C2/NN,N,NNG
DO 80 15=1,NNG
   JX(15)=0
   NP(15)=0
   JIME(15)=0
80   KME(15)=0
NPL=0
JX(1)=NIN
JX(2)=NIN
I=1
J=NIN
NP(I)=NIN
20 K=0
25 K=K+1
IF(NF(J,K)=NOUT)30,50,30
30 IF(NF(J,K))34,32,34
32 IF(J=NIN)60,100,60
C C FLOW CK CHECK
34 NJK=NF(J,K)
IF(NJK=JX(I))45,25,45
C C STORE AND REMEMBER VERTEX
45 I=I+1
NP(I)=NF(J,K)
JMEM(I)=J
IA=I+1
JX(IA)=NF(J,K)
42 J=NF(J,K)
KMFM(I)=K
GO TO 20
C C BACKSTEP
60 J=JMFM(I)
K=KMEM(I)
I=I-1
GO TO 25
C C FINAL PATH VERTEX AND PATH LENGTH
50 II=I+1
NP(II)=NOUT
62 NPL=II
100 CONTINUE
RETURN
END
SUBROUTINE AHAY(JSIG,XCONDXPO,JKON,POLY,LIL,KIK)
C*************************************************************
COMMON/C1/MSORT,KSORT
COMMON/C3/NEXPS,NTO
MMX=0
NNX=0
IF(KIK=1)3,22,3
3 MMU=KIK-1
00 2 M=1,MMU
MMX=MMX+1
IF(JXPO=MSOR1(MM))2,10,2
2 CONTINUE
22 MSORT(KIK)=JXPO
MMX=KIK
KIK=KIK+1
IF(KIK=NEXPS-1)1386,1385,1385
1385 WRITE(6,1387)
1387 FORMAT(1x,42HS-POWER EXCEEDS LIMIT-INCREASE DIMENSIONS ,
116HCONTAINING NEAPS)
1386 CONTINUE
10 IF(LIL=1)11 11,24+11
11 NNU=LIL=1
DO 12 NN=1,NNU
NNX=NNX+1
IF(JKOU-KSORT(NN))12,20,12
12 CONTINUE
24 KSORT(LIL)=JKOU
NNX=LIL
LIL=LIL+1
IF(LIL=NTO-1)1367,1365,1365
1365 WRITE(6,1366)
1366 FORMAT(1x,4HNO. OF TERMS IN OUTPUT EXCEEDS LIMIT-INCREASE ,
125HDIMENSIONS CONTAINING NTO)
1367 CONTINUE
20 POLY(MMX,NNX)=POLY(MMX,NNX)*CON*(14)*JSIG
RETURN
END

SUBROUTINE DECIDE(KON,KOUY,IZ,FB,JZ,SEMBUL,KOEF,KODI,ITOP,KRASIS)
C*******************************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS N5PT; AND NTO (DEFINED IN PROGRAM MAIN=1)
C DIMENSION ITOP(150),SEMBUL(20),KOEF(20),KODI(20)
C*******************************************************************************
COMMON/C4/N5PT
IZ=0
M=KRASIS=1
DO 3 J=1,KOUD
IPower=M,ANV,KOUY
IF(IPower)3,3,2
2 IF(SEMBUL(J),NE,FB)GO TO 10 4
IZ=IZ+1
IF(IZ=N5PT-1)1371,1370,1370
1370 WRITE(6,1372)
1372 FORMAT(1x,4HNO. OF SYMBOLS PER TERM EXCEEDS OUTPUT-INCREMENT ,
125HDIMENSIONS CONTAINING N5PT)
1371 CONTINUE
KOEF(IZ)=IPower
KODI(IZ)=J
GO TO 3
4 ITOP(JZ)=1
3 KOUD=KOUD/KRASIS
RETURN
END
REFERENCES


