

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 7656	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) NRL MODIFIED VERSION OF CINDA-3G PROGRAM		5. TYPE OF REPORT & PERIOD COVERED This is a final report on the problem.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Mary E. Gealy	8. CONTRACT OR GRANT NUMBER(s)	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS None
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Naval Research Laboratory Washington, D.C. 20375		12. REPORT DATE August 22, 1974
		13. NUMBER OF PAGES 207
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) CINDA-3G program for CDC-3800 Numerical differencing analysis Heat transfer		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A programming manual has been prepared for the thermal analyzing program, CINDA. The program's options offer the user a variety of methods for solution of thermal analog models presented to it in a network format. The network representation of the thermal problem is unique in that it has a one-to-one correspondence to both the physical model and the mathematical model. This analogy enables engineers quickly to construct mathematical models of complex		

(Continued)

thermophysical problems and prepare them for program input. In addition, the program contains numerous subroutines for handling interrelated complex phenomena such as sublimation; diffuse radiation within enclosures; simultaneous, one-dimensional, incompressible fluid flow including valving and transport delay effects; etc. It can handle all three types of heat transfer—conduction, convection, and radiation. The optional combinations of these capabilities, in conjunction with the model size allowable (4000 nodes on a 65k-core machine), make CINDA an extremely potent analytical tool for thermal systems analysis in the hands of a competent engineer analyst. Its uses include determining temperatures of structures such as bridges, rockets, and buildings; finding cooling requirements for electric circuits; and studying the thermal properties of adverse thermal systems such as nuclear reactors and automobile engines.

The programs on pp. 104, 105, 106 are adaptations of similar programs published in "MITAS User Information Manual (Martin Marietta Thermal Analyzer System)," CYBERNET Publications Division, Control Data Corporation, Copyright 1972.

CONTENTS

I. INTRODUCTION	1
Background	1
Overview	1
II. DISCUSSION.....	2
Lumped-Parameter Representation	2
Basics of Finite Differencing	4
Iterative Techniques.....	6
Pseudo-Compute Sequence	8
Data Logistics	10
Order of Computation	11
Systems Programming	11
III. DATA INPUT REQUIREMENTS	13
Title Block	13
Node Data Block	14
Conductor Data Block	16
Constants Data Block.....	19
Array Data Block	20
Program Control	21
Execution Operations Block	23
Variables 1 Operations Block	25
Variables 2 Operations Block	26
Output Calls Operations Block	28
Parameter Runs	29
Store and Recall Problem Options	30
Data Printing Option	31
IV. ERROR MESSAGES	31
V. OPERATING SYSTEM DESCRIPTION.....	33
General	33
Preprocessor.....	35
VI. EXTERNAL PLOTTING PACKAGE.....	37
Data Set.....	37
Diagnostics.....	39

VII. TAPE USAGE AND DECK SETUPS	39
CINDA Operating System	39
Plot Package	44
VIII. ALPHABETIC LISTING OF AVAILABLE SUBROUTINES	48
Execution	50
Interpolation	57
Arithmetic	67
Output	83
Matrix	86
Special	99
Internal	108
ACKNOWLEDGMENTS	108
APPENDIX A — Sample Problem 1A	109
APPENDIX B — Sample Problem 1B	161

NRL MODIFIED VERSION OF CINDA-3G PROGRAM

I. INTRODUCTION

Background

When it was recognized that a thermal analyzing system was needed at the Naval Research Laboratory, the Chrysler Numerical Differencing Analyzer for third-generation computers (CINDA-3G)* was obtained for use on the CDC-3800 computer. CINDA-3G was developed by the Thermodynamic Section of the Aerospace Physics Branch of the Chrysler Corporation Space Division at NASA's Michoud Assembly Facility. A major portion of this work was done as a NASA-funded project from the Manned Spacecraft Center in Houston, Texas.

CINDA was programmed in FORTRAN V for the Univac 1108. Because of differences between the computers, modifications to the program were necessary before it could be usable on the CDC-3800. The task of conversion required compensating for compiler differences (FORTRAN V vs 3800 FORTRAN), rewriting some routines in COMPASS, and in general, adapting CINDA to the Drum SCOPE system. Documentation supplied with the Univac version of CINDA was a most useful aid in the conversion process. This NRL report represents the CDC-3800 version of CINDA-3G, and is partly a rewrite of the original Chrysler document. While most of the sections have been only slightly modified, the section on tape usage and deck setups is strictly applicable to CDC-3800 software. The external plotting package in Section VI for the CALCOMP plotter replaces plotting routines used by the Univac 1108 for the SC-4020.

Overview

This programming manual deals with the CINDA system in two general categories—the logic and data needed in setting up the problem data deck, and the actual structure of the operating system.

The logic in constructing a problem for CINDA involves developing a lumped-parameter representation of the physical problem. This model simulates the elements of heat transfer, and the user must supply the corresponding network data, which will be used by one or more routines selected from a large subroutine library. The user must determine which routines are needed and the order in which they are to be activated. This information and the other related logic are entered in a modified FORTRAN language. The major routines involved use various iterative techniques for solutions, with the program-formed compute sequences minimizing the required operations.

Note: Manuscript submitted September 5, 1973.

*D.R. Lewis, J.D. Gaski, and L.R. Thompson, "Chrysler Improved Numerical Differencing Analyzer for 3rd Generation Computers," Technical Note TN-AP-67-287, Chrysler Corporation Space Division, New Orleans, La., 1967.

CINDA is not merely a single execution program, but is an operating system in itself. It consists of two software packages of its own (the *preprocessor* and *subroutine library file*) and is also quite dependent on the computer's system software (at NRL, the CDC-3800's Drum SCOPE system). This dependence is largely due to language and allocating differences of FORTRAN compilers, as well as computer work size and assembly language routines. The preprocessor reads and processes the two types of user data—the network data and the logic data. From the former, its output is a binary data file, and from the latter, it generates five FORTRAN subroutines which, when compiled, are referred to as the processor. The processor makes user-requested calls to subroutines in the library file, executing the binary data output from the preprocessor. The user's logic controls the processor until the end of the job. For any particular problem the arrays in the processor are dimensioned exactly as needed by the preprocessor. This feature, together with user-controlled output, saves computer time and money.

II. DISCUSSION

Lumped-Parameter Representation

The key to utilizing a network-type analysis program lies in the users' ability to develop a lumped-parameter representation of the physical problem. Once this is done, superimposing the network mesh is a mechanical task at most and the numbering of the network elements is simple although perhaps tedious. It might be said that the network representation is a "crutch" for the engineer, but it does simplify the data logistics and allow easy preparation of data input to the program. In addition, it allows the user to identify uniquely any element in the network and modify its value or function during the analysis as well as sense any potential or current flow in the network. Another feature of the network is that it has a one-to-one correspondence to the mathematical model as well as the physical model.

Perhaps the most critical aspect of the lumped-parameter approach is determining the lump size. There are methods for optimizing the lump size, but they usually involve more analytical effort and computer time than the original analysis. One must also keep in mind that for a transient problem, time is being lumped as well as space. Of prime importance is what information is being sought from the analysis. If spot temperatures are being sought, nodes must at least fall on the spots and not include much more physically than would be expected to exist at a relatively similar temperature. Nodes must fall at end points when a temperature gradient is sought. Of necessity, lumping must be fairly fine where isotherms are sought. Lumping should be coarse in areas of high thermal conductivity. When nonlinear properties are being evaluated, the lumping should be fine enough so that extreme gradients are not encountered. The lumping is also dependent on the severity of the nonlinearity.

To reduce round-off error, the explicit stability criteria of the lump (the capacitance value divided by the summation of conductor values into the node) should be held fairly constant. The value $C/\Sigma G$ is directly proportional to the square of the distance between nodes. Although refining the lumped-parameter representation will yield more accurate answers, halving the distance between nodes decreases the stability criteria by a factor of four and increases the number of nodes by a factor of two, four, or eight depending upon whether the problem is one, two, or three dimensional. For the explicit case,

halving the distance between nodes increases the machine time for transient analysis by a factor of 8, 16, or 32, respectively. The increase in solution time for the implicit methods is somewhat less but proportional.

When lumping the time space, consideration must be given to the frequency of the boundary conditions. A time step must not step over boundary excitation points or they will be missed. Do not step over pulses; rather, rise and fall with them. Generally the computation interval for the explicit methods is sufficiently small so that frequency effects can be ignored. However, care must be exercised when specifying the time step for implicit methods. If only a small portion of a transient analysis involves frequency considerations, the time step used may be selectively restricted for that interval. By setting the maximum time step allowed as a function of time, we may utilize an interpolation call to vary it accordingly.

One must also realize that the problem being solved is linearized over the time step. Heating rate calculations are usually computed for a time point and then applied to a time space. If the rates are nonlinear, a certain amount of error is introduced, particularly so with radiation. These nonlinear effects may cause almost any method of solution to diverge. A brute force method for forcing convergence is to limit the temperature change allowed over the time space. Consideration of the factors mentioned above, coupled with some experience in using the program, will aid the observant analyst in choosing lump sizes that will yield answers of sufficient engineering accuracy with a reasonable amount of computer time.

Figure 1 shows the lumped-parameter representation and network superposition of a one-dimensional heat transfer problem.

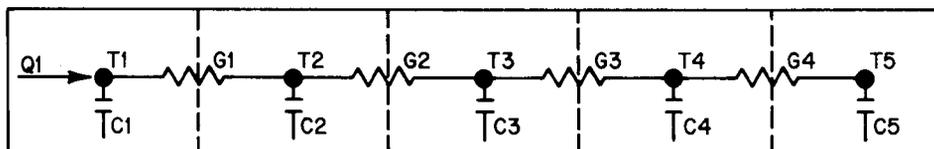


Fig. 1—One-dimensional network

The "node" points are centered within the lumps, and temperatures at the nodes are considered uniform throughout the lump. The capacitors hung from the nodes indicate the ability of the lump to store thermal energy. Capacitance values are calculated as lump volume times density times specific heat. The conductors (electrical symbol G) represent the capability for transmitting thermal energy from one lump to another. Conductor values for energy transmission through solids are calculated as thermal conductivity times the energy cross-sectional flow area divided by path length (distance between nodes). Conductor values for convective heat transfer are calculated as the convection coefficient times the energy cross-sectional flow area. Conductors representing energy transfer by radiation are usually indicated by crossed arrows over the conductor symbol. Radiation transfer is nonlinear; it is proportional to the difference of the absolute temperatures raised to the fourth power. Utilization of the Fahrenheit system allows easy automation of this nonlinear transfer function by the program and reduces the radiation conductor value to the product of the Stephan-Boltzmann constant times the surface area times the net radiant interchange factor (script F).

Basics of Finite Differencing

The concept of network superposition on the lumped-parameter representation of a physical system is easy to grasp. Describing the network to the program is also quite straightforward. Having described a network to the program, what information have we really supplied and what does the program do with it? Basically, we desire the solution to a simultaneous set of partial differential equations of the diffusion type; i.e.,

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + S, \quad \nabla^2 \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \quad (1)$$

That the diffusivity ($\alpha = k/\rho C_p$) may be temperature varying or nonlinear radiation transfer occurring is immaterial at this point. Of importance is how Eq. (1) is finite differenced and its relationship to the network and energy flow equations more commonly utilized by the engineer. The partial of the T-state variable with respect to time is finite differenced across the time space as follows:

$$\frac{\partial T}{\partial t} \approx \frac{T' - T}{\Delta t}, \quad (2)$$

where the prime indicates the new T value after passage of the Δt time step.

The right side of Eq. (1) could be written with T primed to indicate implicit "backward" differencing or unprimed to indicate explicit "forward" differencing. The following equation is illustrative of how "backward" and "forward" combinations may be obtained.

$$\frac{\partial T}{\partial t} = \beta(\alpha \nabla^2 T + S) + (1 - \beta)(\alpha' \nabla^2 T' + S'); \quad 0 \leq \beta \leq 1. \quad (3)$$

Any value of β less than 1 yields an implicit set of equations which must be solved in a simultaneous manner (more than one unknown exists in each equation). Any value of β equal to or less than 1/2 yields an unconditionally stable set of equations or in other words, any time step desired may be used. Values of β greater than 1/2 invoke stability criteria or limitations on the magnitude of the time step. A value of β equal to 1/2 yields an unconditionally stable implicit set of equations commonly known as "forward-backward" differencing or the Crank-Nicholson method. Various transformations or first order integration applied to Eq. (1) generally yield an implicit set of equations similar to Eq. (3) with β equal to 1/2. The following finite difference approach generally applies to transformed equations.

Let us consider the right side of Eq. (3) with $\beta = 1$ and rewrite it as follows:

$$\alpha \nabla^2 T + S \approx \frac{\alpha}{\Delta x} \left(\frac{\partial T}{\partial x^-} - \frac{\partial T}{\partial x^+} \right) + \frac{\alpha}{\Delta y} \left(\frac{\partial T}{\partial y^-} - \frac{\partial T}{\partial y^+} \right) + \frac{\alpha}{\Delta z} \left(\frac{\partial T}{\partial z^-} - \frac{\partial T}{\partial z^+} \right) + S. \quad (4)$$

The minus or plus signs on the first partial terms indicate that they are taken on the negative or positive side, respectively, of the point under consideration and always in the

same direction. If we consider three consecutive points (1, 2, and 3) ascending in the x direction we can complete the finite difference of the x portion of Eq. (4) as follows:

$$\frac{\alpha}{\Delta x} \left(\frac{\partial T_2}{\partial x^-} - \frac{\partial T_2}{\partial x^+} \right) \approx \frac{\alpha}{\Delta x} \left(\frac{T_1 - T_2}{\Delta x^-} + \frac{T_3 - T_2}{\Delta x^+} \right). \quad (5)$$

Applying the above step to the y and z portions of Eq. (4) yields the common denominator of volume ($V = \Delta x * \Delta y * \Delta z$). Using Eq. (3) with $\beta = 1$, finite differencing with the steps used for Eqs. (3), (4), and (5), substituting $\alpha = k/\rho C_p$, and multiplying both sides by $\rho V C_p$ yield

$$\begin{aligned} \frac{\rho V C_p}{\Delta t} (T_0' - T_0) &= \frac{k A_x}{\Delta x^-} (T_1 - T_0) + \frac{k A_x}{\Delta x^+} (T_2 - T_0) \\ &+ \frac{k A_y}{\Delta y^-} (T_3 - T_0) + \frac{k A_y}{\Delta y^+} (T_4 - T_0) \\ &+ \frac{k A_z}{\Delta z^-} (T_5 - T_0) + \frac{k A_z}{\Delta z^+} (T_6 - T_0) + Q, \end{aligned} \quad (6)$$

where $A_x = \Delta y \Delta z$, $A_y = \Delta x \Delta z$, $A_z = \Delta x \Delta y$ and $Q = \rho V C_p S$.

x, y, and z correspond to the coordinates of Fig. 2a.

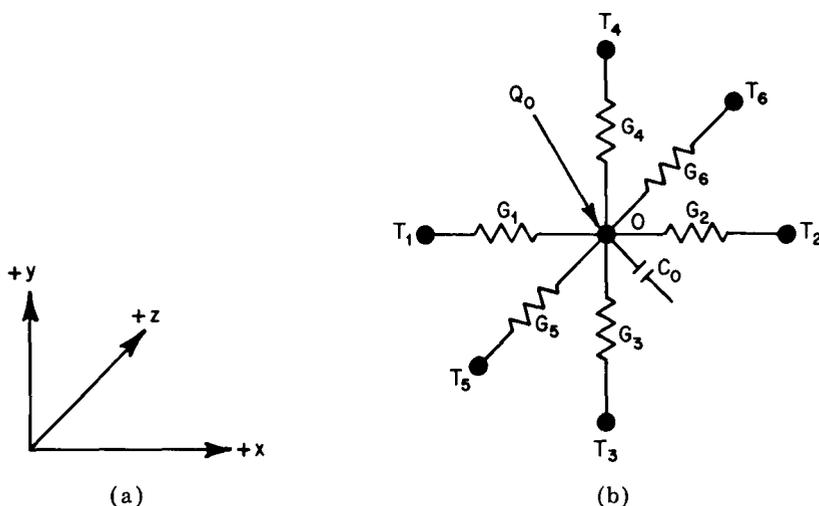


Fig. 2—Network of a three-dimensional system

The numbering system corresponds to a three-dimensional network, shown in Fig. 2b. It should be obvious that the network capacitance value is ρVC_p , that the G_1 value is $kAx/\Delta x$, etc. Equation (6) may then be written as

$$C_0(T'_0 - T_0)/\Delta t = G_1(T_1 - T_0) + G_2(T_2 - T_0) + G_3(T_3 - T_0) + G_4(T_4 - T_0) + G_5(T_5 - T_0) + G_6(T_6 - T_0) + Q_0, \quad (7)$$

or in engineering terminology, the rate of change of temperature with respect to time is proportional to the summation of heat flows into the node.

It should be noted that Fig. 2 is essentially superimposed on a lumped-parameter cube of a physical system and is the network representation of Eq. (1). Since Eq. (7) is written in explicit form, only one unknown (T'_0) exists and all of the information necessary for its solution is contained in the network description. If it had been formulated implicitly, it would have to be solved in a simultaneous manner. No matter what method of solution is requested of the program, the information necessary has been conveyed by the network description. When an implicit set is used with $\beta > 0$, the energy flows based on old temperatures are added to the Q term and the equations are then treated in the same manner as for $\beta = 0$;

$$\alpha \nabla^2 T + S = 0. \quad (8)$$

The solution of Poisson's equation (8) is the solution utilized for steady state analysis. It is extremely important because virtually all of the unconditionally stable implicit methods reduce to it. If Eq. (7) had all the right side values primed and the left side was subtracted from both sides, we could think of $C_0/\Delta t$ as a G_0 term and T_0 (old) would then become a boundary node. In a manner of speaking, the capacitor we look at in three dimensions becomes a conductor in four dimensions. We could draw a four-dimensional network, but since there is no feedback in time it is senseless to take more than one time step at a time. However, various time-space transformations can be utilized such that a one-dimensional "transient" yields the solution to a two-dimensional steady state problem, etc. This is analogous to the "Particle in Cell" method developed in the nuclear field for following shock-wave propagation.

Iterative Techniques

Now that we have discussed the correlation between the physical, network, and mathematical models, let's investigate the commonality of the various methods of solution. By describing the network of Fig. 1 to the program, we have supplied it with five temperatures, five capacitors, five sources (four not specified and therefore zero), four conductors, and the adjoining node numbers of the conductors. An explicit formulation such as Eq. (6) has only one unknown. Its solution is easily obtainable as long as any associated stability criteria are continuously satisfied. A more interesting formulation would be a set of implicit equations as follows:

$$\begin{aligned}
 (T'_1 - T_1)C_1/\Delta t &= Q'_1 + G_1(T'_2 - T'_1) \\
 (T'_2 - T_2)C_2/\Delta t &= Q'_2 + G_1(T'_1 - T'_2) + G_2(T'_3 - T'_2) \\
 (T'_3 - T_3)C_3/\Delta t &= Q'_3 + G_2(T'_2 - T'_3) + G_3(T'_4 - T'_3) \\
 (T'_4 - T_4)C_4/\Delta t &= Q'_4 + G_3(T'_3 - T'_4) + G_4(T'_5 - T'_4) \\
 (T'_5 - T_5)C_5/\Delta t &= Q'_5 + G_4(T'_4 - T'_5) .
 \end{aligned} \tag{9}$$

If the above had been formulated as partly explicit and implicit, the known explicit portion would have been calculated and added to the Q terms, then the β factor would have been divided into the Q terms and multiplied by the Δt term.

If we divide the Δt term into the C terms and indicate this by priming C, we can reformulate Eq. (9) as follows:

$$\begin{aligned}
 (C'_1 + G_1) T'_1 &= Q'_1 + C'_1 T_1 + G_1 T'_2 \\
 (C'_2 + G_1 + G_2) T'_2 &= Q'_2 + C'_2 T_2 + G_1 T'_1 + G_2 T'_3 \\
 (C'_3 + G_2 + G_3) T'_3 &= Q'_3 + C'_3 T_3 + G_2 T'_2 + G_3 T'_4 \\
 (C'_4 + G_3 + G_4) T'_4 &= Q'_4 + C'_4 T_4 + G_3 T'_3 + G_4 T'_5 \\
 (C'_5 + G_4) T'_5 &= Q'_5 + C'_5 T_5 + G_4 T'_4 .
 \end{aligned} \tag{10}$$

This equation can be generalized as

$$T'_i = \frac{C'_i T_i + \sum G_a T'_a + Q'_i}{C'_i + \sum G_a} , \tag{11}$$

where the subscript a indicates connection to adjoining nodes. A C' value of zero yields the standard steady state equation, the conductor weighted mean of all the surrounding nodes. We see here that the C' can be thought of as a conductor to the old temperature value and therefore Eq. (11), although utilized to obtain transient solutions, can be considered as a steady state equation in four dimensions. By rewriting Eqs. (10) in the form of Eq. (11) we are in a position to discuss iterative techniques. By assuming all old values on the right hand side of Eq. (10), we could calculate a new set of temperatures on the left which, although wrong, are closer to the correct answer. This single set of calculations is termed an iteration. By replacing all of the old temperatures with those just calculated, we can then perform another iteration. This process is called "block" iteration. A faster method is to utilize only one location for each temperature. This way, the newest temperature available is always utilized. This method is termed "successive point" iteration and is generally 25% faster than "block" iteration. The iterative process is continued a fixed (set by user) number of times or until the maximum absolute difference between the new and old temperature values is less than some prespecified value (set by user).

Although the above operations are similar to a relaxation procedure, there is a slight difference. We are performing a set of calculations in a fixed sequence. A relaxation procedure would continuously seek the node with the maximum temperature difference between old and new and calculate it. Programmingwise, as much work is required in the seeking operation, which must be consecutive, as in the calculation. For this reason it would be wasteful to code a true relaxation method.

In addition to the iterative approach, several solution subroutines utilize an acceleration feature and/or a different convergence criteria. Once it can be determined that the temperatures are approaching the steady state value, an extrapolation is applied in an attempt to accelerate convergence. This convergence criterion is the maximum absolute temperature change allowed between iterations. This criteria, however, is generally one sided and any associated errors are accumulative. In order to obtain greater accuracy, some subroutines are coded to perform an energy balance on the entire system (a type of Green's function) and apply successively more severe convergence criteria until the system energy balance (energy in minus energy out) is within some prespecified tolerance.

Pseudo-Compute Sequence

A set of simultaneous equations such as Eqs. (10) is quite often treated by matrix methods and formulated as follows:

$$[A] \{T'\} = \{B\}, \quad (12)$$

where

$$\{A\} = \begin{bmatrix} (C'_1 + G_1) & -G_1 & 0 & 0 & 0 \\ -G_1 & (C'_2 + G_1 + G_2) & -G_2 & 0 & 0 \\ 0 & -G_2 & (C'_3 + G_2 + G_3) & -G_3 & 0 \\ 0 & 0 & -G_3 & (C'_4 + G_3 + G_4) & -G_4 \\ 0 & 0 & 0 & -G_4 & (C'_5 + G_4) \end{bmatrix}$$

and

$$(13)$$

$$\{T'\} = \begin{Bmatrix} T'_1 \\ T'_2 \\ T'_3 \\ T'_4 \\ T'_5 \end{Bmatrix} \quad \{B\} = \begin{Bmatrix} Q'_1 + C'_1 T'_1 \\ Q'_2 + C'_2 T'_2 \\ Q'_3 + C'_3 T'_3 \\ Q'_4 + C'_4 T'_4 \\ Q'_5 + C'_5 T'_5 \end{Bmatrix}$$

The inverse of [A] is then calculated and the solution obtained by matrix multiplication;

$$\{T'\} = [A]^{-1} \{B\}. \quad (14)$$

It should be noted that the one-dimensional problem has no more than three finite values in any row or column of the coefficient matrix $[A]$. A three-dimensional problem would generally have no more than seven finite values in any row or column. It is easy to see that a 1000-node, three-dimensional problem would require one million data locations, of which approximately 993,000 would contain zero. The inverse might require an additional one million data locations. Aside from exceeding computer core area, the computer time required to calculate the inverse is proportional to the cube of the problem size, and large problems soon become uneconomical to solve.

The explicit and iterative implicit methods previously discussed are well suited for optimizing the data storage area required. Note the adjoining node numbers associated with the conductors of Fig. 1;

$$\begin{aligned} 1,1,2 &\rightarrow G1 \text{ between nodes 1 and 2} \\ 2,2,3 &\rightarrow G2 \text{ between nodes 2 and 3} \\ 3,3,4 &\rightarrow G3 \text{ between nodes 3 and 4} \\ 4,4,5 &\rightarrow G4 \text{ between nodes 4 and 5.} \end{aligned} \quad (15)$$

Note also the row and column position of conductor values off the main diagonal in the $[A]$ coefficient matrix, Eq. (13). By retaining the adjoining node numbers for each conductor we are able to identify its element position in the coefficient matrix. As a consequence we need store only the finite values. The main diagonal term in a composite of the node capacitance and conductor values off the main diagonal.

The program operates on the adjoining node numbers to form what is termed the pseudo-compute sequence (PCS). The nodes are to be calculated sequentially in ascending order, so the adjoining nodes are searched until the number 1 is found. When this occurs the conductor number and the adjoining node number are stored as a doublet value. The search is continued until all nodes one are located and the conductor number for the last receives a minus sign. The process is then continued for node two, etc., until all the node numbers have been processed. The pseudo-compute sequence formed (LPCS) is shown below left. A slight variation to this operation is to place a minus sign on the original other adjoining node number so that it is not recognized when it is searched for. The resulting pseudo-compute sequence thus formed (SPCS) is shown below right.

LPCS	SPCS
-1,2	-1,2
1,1	-2,3
-2,3	-3,4
2,2	-4,5
-3,4	-0,0
3,3	
-4,5	
-4,4	

The above pseudo-compute sequences are termed long (LPCS) and short (SPCS), respectively. By starting at the top of the pseudo-compute sequence, we are operating on node 1. The two values identify the conductor into the node (the position of the conductor value in an array of conductor values) and the adjoining node (the position of the temperature, capacitor, and source values in arrays of temperature, capacitor, and source values, respectively). The node being operated on starts as one and is advanced by one each time a negative conductor number is passed.

It is easy to see that the LPCS identifies the element position and value locations of all the off-diagonal elements of the row being operated on. It takes complete advantage of the sparsity of the coefficient matrix. It is well suited for "successive point" iteration of the implicit equations because all elements in a row are identified. When a row is processed and the new T value obtained, the new T can then be used in the calculation procedure of succeeding rows.

The SPCS identifies each conductor only once and in this manner takes advantage of the symmetry of the coefficient matrix as well as the sparsity. It is well suited for explicit methods of solution. The node being operated on and the adjoining node number reveal their temperature value locations and their source value locations. The explicit solution subroutines calculate the energy flow through the conductor, add it to the source location of the node being worked on, and subtract it from the source location for the adjoining node. However, if the short pseudo-compute sequence were utilized for implicit methods of solution, they would require the use of slower "block" iterative procedures. The succeeding rows do not have all of the elements defined, and the energy rates passed ahead were based on old temperature values.

Data Logistics

The pseudo-compute sequence formulated as shown above allow the program to store only the finite values in the coefficient matrix, thereby taking advantage of its sparsity. In addition, the SPCS takes advantage of any symmetry which may exist. Multiply-connected conductors which will be covered in the next section allow the user to take advantage of similarity as well. The foregoing is fairly easy to follow, especially if the nodes and conductors start with the number 1 and continue sequentially with no missing numbers. This restriction is too limiting for general use on large network models. To overcome this restriction the program assigns relative numbers (sequential and ascending) to the incoming node data, conductor data, constants data, and array data in the order received. Any numbers missing in the actual numbering system set up by the user are packed out, thereby requiring only as much core space as is actually necessary.

All solution (Execution) subroutines require three locations for diffusion node data (temperature, capacitance, and source) and one location for each conductor value. They also may require from zero to three extra locations per node for intermediate data storage. Each node in a three-dimensional network has essentially six conductors connected to it but only three are unique; i.e., each additional node requires only three more conductors. Hence, each node in a three-dimensional system requires from six to nine storage locations for data values (temperature, capacitance, source, three conductors, and up to three intermediate locations). The two integer values that make up a doublet of the PCS are packed into a single core location. Hence, for a three-dimensional network,

each node requires approximately three locations for data addressing for the short and six locations for the long pseudo-compute sequence. The number of core locations required for node can vary from 9 to 15.

The program requires the user to allocate an array of data locations to be used for intermediate data storage and initialize array start and length indicators. Each subroutine that requires intermediate storage area has access to this array and the start and length indicators. They check to see that there is sufficient space, update the start and length indicators, and continue with their operations. If they call upon another subroutine requiring intermediate storage, the secondary subroutine repeats the check and update process. Whenever any subroutine terminates its operations it returns the start and length indicators to their entry values. This process is termed *dynamic storage allocation* and allows subroutines to share a common working area.

Order of Computation

A problem data deck consists of data and operations blocks which are preprocessed by CINDA and passed on to the system FORTRAN compiler. The operations blocks are named EXECUTION, VARIABLES 1, VARIABLES 2 and OUTPUT CALLS. The FORTRAN compiler constructs these blocks as individual subroutines with the entry names EXECTN, VARBL1, VARBL2 and OUTCAL, respectively. After a successful compilation, control is passed to the EXECTN subroutine. Therefore, the order of computation depends on the sequence of subroutine calls placed in the EXECUTION block by the program user. No other operations blocks are performed unless called upon by the user either directly by name or indirectly from some subroutine which internally calls upon them. The network execution subroutines listed on p. internally call upon VARBL1, VARBL2, and OUTCAL. Their internal order of computation is quite similar, the primary difference being the analytical method by which they solve the network. Figure 3 represents a flow diagram of all the network solution subroutines; the subroutine writeups contain the comparisons made at the various check points and the routings taken.

Systems Programming

CINDA is actually an operating system rather than an applications program. Two programs are run and executed, the second program being the product of the first. The initial program, the *preprocessor*, operates in an integral fashion with a large library of assorted subroutines which can be called in any sequence desired. It reads all of the input data, packs them, assigns relative numbers, forms the pseudo-compute sequence, and writes the data on two different peripheral units. One unit contains FORTRAN source language generated from the operations blocks, with all of the data values dimensioned exactly in name COMMON. This program, the *processor*, is then compiled and executed, using as input the data from the first-mentioned peripheral unit. The FORTRAN allocator has access to the CINDA subroutine library and loads only those subroutines referred to by the problem being processed.

Due to this type of operation, CINDA is extremely dependent on the system software supplied. However, once the program has been made operational on a particular machine, the problem data deck prepared by the user can be considered as machine independent.

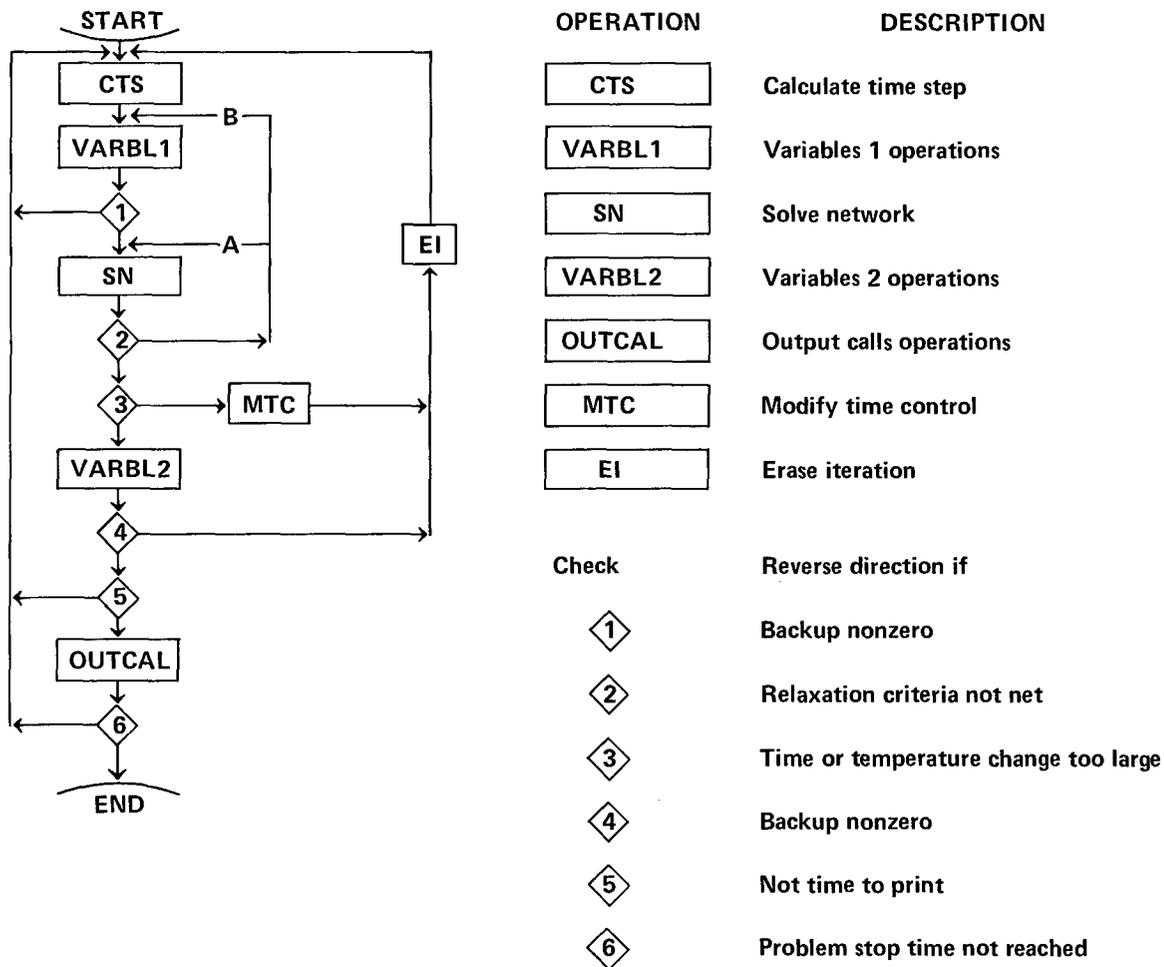


Fig. 3—Basic flowchart for network solution subroutines

III. DATA INPUT REQUIREMENTS

A CINDA problem data deck consists of both data and instruction cards. The card-reading subroutines for CINDA do not utilize a fixed format type of input; but instead use a free-form format. The type of data is designated by a mnemonic code in columns 8, 9, and 10. This is followed by the data field which consists of columns 12-80 or the instruction field which consists of columns 12-72. Although blanks are allowed before or after numerical data they may not be contained within. The number 1.234 is fine, but 1. 234 will cause the program to abort. The program processes the problem data into FORTRAN common data and reforms instructions into FORTRAN source language which are then passed on to the system FORTRAN compiler. Instruction cards which contain an F in column 1 are passed on exactly as received. Any instruction card with or without an F in column 1 may contain a statement or sequence number in columns 2-5 which is passed on to and used by the FORTRAN compiler.

The most frequently used mnemonic code is three blanks. The data following this blank mnemonic code may be one or more integers, floating-point numbers (with or without the E exponent designation) or alphanumeric words of up to six characters each. The reading of a word or number continues until a comma is encountered and then the next word or number is read. As many numbers or words as desired may be placed on a card, but they may not be broken between cards. A new card is equivalent to starting with a comma and therefore no continuation designation is required. All blanks are ignored and reading continues until the terminal column is reached or a dollar sign encountered. Comments pertinent to a data card may be placed after a dollar sign and are not processed by the program. If sequential commas are encountered, floating-point zero values are placed between them.

The next most frequently used code is BCD (for binary-coded decimal) which must be followed by an integer 1 through 9 in column 12. The integer designates the number of 6-character words immediately following it. Blanks are retained and only the designated number of 6-character words are read from the card. The mnemonic code END is utilized to designate the end of a block of input to the program. The code REM serves the same function as a FORTRAN comment card; it is not processed by the program but allows the user to insert nondata for clarification purposes. The special codes CGS, CGD, and GEN will be discussed later in this section.

The data deck prepared by a program user consists of various input blocks containing either data or instructions. A fixed sequence of block input is required, and each block must start with a BCD 3 header card and terminate with an END card (mnemonic codes). Specific details about these blocks follows.

Title Block

The first card of a problem data deck is the title block header card. It conveys information to the program as to the type of problem, which data blocks follow, and how they should be processed. The three options presently available are

```

      8
      ↓
      BCD 3GENERAL
or   BCD 3THERMAL SPCS
or   BCD 3THERMAL LPCS

```

The GENERAL indicates that a nonnetwork problem follows and therefore no node or conductor data is present. The THERMAL cards indicate that a conductor-capacitor (CG) network description follows and that either a short (SPCS) or long (LPCS) pseudo-compute sequence should be constructed. The title block header card may be followed by as many BCD cards as desired. However, the first 20 words (six characters each) are retained by the program and used as a page heading by the user-designated output routines. The block must be terminated by an END card and is then followed by node data for a CG network problem or constants data for a nonnetwork problem.

Node Data Block

As discussed in Section II, there are three types of nodes; diffusion, arithmetic, and boundary. Diffusion nodes are those nodes with a positive capacitance and have the ability to store energy. Their future values are calculated by a finite difference representation of the diffusion partial differential equation. Arithmetic nodes are designated by a negative capacitance value; they have no physical capacitance and are unable to store energy. Their future values are calculated by a finite difference representation of Poisson's partial differential equation. This is a steady state calculation which always utilizes the latest diffusion node values available. Boundary nodes are designated by a minus sign on the node number; they refer to the mathematical boundary, not necessarily the physical boundary. Their values are not changed by the network solution subroutines but may be modified as desired by the user.

A diffusion node causes three core locations to be utilized, one each for temperature, capacitance, and a source location. An arithmetic node receives core locations only for temperature and source and a boundary node receives only a temperature location. The program user is required to group his node data into the above three classes and submit them in that order. Node data input with the three-blank mnemonic code always consists of three values—the integer node number followed by the floating-point initial temperature and capacitance values. A negative capacitance value is used to designate an arithmetic node while a negative node number designates a boundary node. Although the capacitance value of a boundary node is meaningless, it must be included so as to maintain the triplet format.

All nodes are renumbered sequentially (from one on) in the order received. The user input number is termed the actual node number while the program assigned number is termed the relative node number. This relative numbering system allows sequential packing of the data and does not require a sequential numbering system on the part of the program user. It is worth noting that the pseudo-compute sequence is based on the relative numbering system. Hence, the computational sequence of the nodes is identical with their input sequence. If a user desired to reorder the computations in order to aid boundary propagation, he needs merely to reorder his nodal input data.

The mnemonic codes CGS, CGD, and GEN may be used. The CGS and CGD codes are used when one or two materials, respectively, with temperature-varying properties are to be considered. For a single material the node number and initial temperature remain the same but instead of a capacitance value, the user may input the starting location (integer count) or a doublet array of the temperature-varying property followed by the actual (literal) multiplying factor value to complete the calculation or a constants location containing it. For a node consisting of two materials, the node number and initial temperature remain the same but the user would use two array addresses and multiplying factors with a CGD code. These codes would look as follows:

```

8
↓
CGS N#,TI,A1,F1
or CGD N#,TI,A1,F1,A2,F2
    
```

where N# is the integer node number and Ti is the floating-point initial temperature. The A arguments refer to doublet arrays of temperature-varying Cp or ρ^*Cp , and the F arguments may be or refer to a constant location containing the weight or volume, respectively. The CGS code causes the product of the interpolated value times the F factor to be used as the capacitance value. The CGD code uses the sum of the separate interpolations times the factor products as the capacitance value.

To input a group of sequential nodes, the following code is available:

```

8
↓
GEN N#,#N,IN,TI,X,Y,Z,W
    
```

where

- N# is the starting node number
- #N is the total number of nodes desired (integer)
- IN is an increment for the generated nodes (integer)
- Ti is the initial temperature for all nodes,

and the capacitance value is calculated as the produce of X times Y times Z times W. If this product is negative, arithmetic nodes will be generated. If N# is negative, boundary nodes will be generated. A sample node data block could be as follows:

```

8  12
↓  ↓
BCD 3NODE DATA
    1,80.,1.2,2,80.,1.3           $TWO DIFFUSION NUDES
CGS 3,80.,A1,4.63                 $SINGLE MATERIAL NODE
CGD 4,80.,A1,2.31,A2,4.76         $DOUBLE MATERIAL NODE
GEN 5,10,1,80.,4.63,1.,1.,1.     $GENERATE 10 NODES,5-14
    15,80.,-1.,16,80.,-1.        $TWO ARITHMETIC NUDES
    -18,-460.,0                   $ONE BOUNDARY NODE
END
    
```

The above does not correspond to a problem; it just represents data input. Note that the nodes are input in the order: diffusion, arithmetic and boundary. The factor portion of the CGS and CGD codes may be a literal (actual value) as shown or reference a constant's location containing the value. Either one (not both) of the array arguments on the CGD code may be a literal if the property is constant. Both codes set up linear interpolation calls which utilize the node temperature as the independent variable and interpolate a dependent value which is then multiplied by the factor to obtain the capacitance value. The CGD call causes two interpolations and multiplications to be performed and sums the products to obtain the capacitance value. These interpolations are performed each iteration during the transient analysis.

The GEN code expects values in the following order; starting node number, number of nodes to be generated, an increment for indexing the generated node numbers, the initial temperature for all nodes, and four floating point numbers, the product of which is the capacitance value.

Conductor Data Block

Two basic types of conductors may be used, regular or radiation, and either may utilize temperature-varying properties in calculating the conductance value. When utilizing the blank mnemonic code a regular conductor consists of the integer conductor number followed by two integer adjoining node numbers and the floating-point conductance value. If more than one conductor has the same constant value, they may share the same conductor number and value. This is accomplished by placing two or more pairs of integer adjoining node numbers between the conductor number and value. The CGS and CGD mnemonic codes may also be utilized for conductors. They would appear as follows:

```

      8
      ↓
CGS  G# ,NA ,NB ,A1 ,F1
or   CGD G# ,NA ,NB ,A1 ,F1 ,A2 ,F2
    
```

where

G# is the integer conductor number
 NA is one adjoining node number
 NB is the other adjoining node number.

The A arguments refer to doublet arrays of temperature-varying thermal conductivity $k(T)$, and the F arguments may be or refer to a constant location containing the cross sectional area divided by path length.

For CGS with F1 positive

$$G = k1(Tm)*F1, Tm = (Ta + Tb)/2.0. \quad (16)$$

For CGS with F1 negative

$$G = k1(Ta)*|F1|. \quad (17)$$

For CGD

$$G = \frac{1.0}{\frac{1.0}{k_1(T_a)*F_1} + \frac{1.0}{k_2(T_b)*F_2}} \quad (18)$$

The CGS mnemonic code may be utilized for either regular or radiation conductors. The data consist of the integer conductor number and one pair only of integer adjoining node numbers, and are followed by an array address and multiplying factor. A regular conductor would normally utilize the CGS code where the addressed array would be thermal conductivity vs temperature, and the multiplying factor would consist of the cross-sectional area divided by path length. A surface radiation conductor would utilize the CGS code for a temperature-varying array of emissivity with the multiplying factor being the product of surface area times the Stephan-Boltzmann constant ($F = 1.0$).

The CGD code may be utilized for regular conductors passing through two materials. In this case two temperature-varying property arrays and multiplying factors are input. Two conductance values are calculated and one over the summation of their inverses is returned as the conductor value. Either of the array addresses may be a literal if one of the properties is a constant. The GEN code is also available for conductors and is input as follows:

```

8
↓
GEN G#, #G, IG, NA, INA, NB, INB, X, Y, Z, W

```

where

G# is the starting conductor number
 #G is the total number of conductors desired (integer)
 IG is an increment for the generated conductors (integer)
 NA and NB are initial adjoining node numbers (integers)
 INA and INB are increments for the generated adjoining nodes (integers),

and all generated conductors receive the same conductance value of X times Y times Z divided by W. A negative G# will cause radiation conductors to be generated.

The GEN code may be used to generate sequential conductors, either radiation or regular. The data consist of the integer conductor number, an integer for the number of conductors to be generated, an integer increment for indexing the generated conductors, the first integer adjoining node number, an integer increment for indexing the first adjoining node number, the second integer adjoining node number, an integer increment for indexing the second adjoining node number, and finally four floating-point numbers; the product of the first three divided by the fourth is the constant conductance value. For example:

```

8
↓
GEN 1,2,1,1,1,2,1,2.,2.,2.,2.
GEN -3,3,0,1,1,10,0,1.,1.,1.,1.E+15

```

is equivalent to

12
 ↓
 1,1,2,4.,2,2,3,4.
 -3,1,10,2,10,3,10,1.E-15.

An additional feature of the program is the one-way conductor. This is a conductor value which enters into the temperature calculation of only one of its adjoining nodes and is indicated by placing a minus sign on the unaffected node. The CGS, CGD, and GEN codes may be used for one-way conductors. Physically this occurs in incompressible fluid flow, and therefore the upstream node would receive the minus sign.

A program idiosyncrasy which should be mentioned is that while a single-valued conductor with as many adjoining node pairs as desired may be used, extending several cards if necessary, an adjoining node pair must not be split between cards. In addition, the CGS, CGD, and GEN card may have more than one set of data on a card, but a set of data may not be broken between cards. All regular conductors must be entered prior to any radiation conductors. The following is illustrative of the various conductor input options.

```

8
↓
BCD 3CONDUCTOR DATA
    1,1,2,1.2,2,2,3,1.7
    3,3,4,4,5,5,6,1.5
    4,-7,8,-8,9,7.6
CGS 5,10,11,A3,4.6
CGD 6,12,13,A3,4.1,A4,7.6
GEN 7,3,1,1,1,9,1,1.6,4.0,1.,1.
    -10,1,99,1.E-15
CGS -11,2,99,A5,1.E-14
GEN -12,4,1,3,1,99,0,1.E-14,1.,1.,1.
END
    $TWO REGULAR CONDUCTORS
    $TRIPLE PLACED CONDUCTOR
    $DOUBLE PLACED ONE-WAY COND.
    $VARIABLE CONDUCTOR, SINGLE
    $VARIABLE CONDUCTOR, DOUBLE
    $GENERATE THREE CONDUCTORS
    $RADIATION CONDUCTOR
    $VARIABLE EMISSIVITY RADIATION
    $GENERATE FOUR RADIATION COND.
    
```

The first GEN card is equivalent to the following:

12
 ↓
 7,1,9,6.4,8,2,10,6.4,9,3,11,6.4

and the second GEN card is equivalent to

12
 ↓
 -12,3,99,1.E-14,-13,4,99,1.E-14
 -14,5,99,1.E-14,-15,5,99,1.E-14

If the second GEN card had incremented the conductor number by zero, it would have been equivalent to

12

↓

-12,3,99,4,99,5,99,6,99,1.E-14

Once the node and conductor data have been read by the program, construction of the pseudo-compute sequence is performed. Any errors encountered cause an appropriate error message to be printed and a "do not execute" switch to be set. However, the program will continue to process input data and attempt to discover any and all recognizable errors. Items checked for are no duplicate node or conductor numbers, all conductor adjoining nodes must have been specified in node data, and all diffusion and arithmetic nodes must have at least one conductor into them. A missing comma will dislocate the data input sequence causing pages of error messages. If over 200 error messages are printed, the program gives up and immediately terminates.

Constants Data Block

Constants data are always input as doublets, the constant name or number followed by its value. They are divided into two types, control constants and user constants, and may be intermingled within the block. Control constants (≈ 50) have alphanumeric names while user constants receive a number. User constants are simply data storage locations which may contain integers, floating-point numbers, or up to 6-character alphanumeric words. It is up to the program user to place data in user constant locations as needed and supply the location addresses to subroutines as arguments.

Control constant values are communicated through program COMMON to specific subroutines which require them. However, any control constant name desired can be used as a subroutine argument. Wherever possible, control constant values not specified are set to some acceptable value. If a required control constant value is not specified, an appropriate error message is printed and the program terminated. It is up to the user to check the writeups of subroutines he is using to determine control constant requirements. A list of control constant names and brief description of each follows; check subroutine writeups for exact usage.

ARLXCA	The maximum arithmetic relaxation change allowed.	
ARLXCC	The maximum arithmetic relaxation change calculated.	
ATMPCA	The maximum arithmetic temperature change allowed.	
ATMPCC	The maximum arithmetic temperature change calculated.	
BACKUP	If nonzero, the time step just done is erased and redone.	
BALENG	User-specified system energy balance to be maintained.	
CSGFAC	Stability criteria multiplication/division factor.	
CSGMAX	Maximum stability criteria for the network.	} (C/ Σ G) max and min.
CSGMIN	Minimum stability criteria for the network.	
CSGRAL	Stability criteria range allowed.	
CSGRCL	Stability criteria range calculated.	
DAMPA	Arithmetic node damping factor.	
DAMPD	Diffusion node damping factor.	
DRLXCA	The maximum diffusion relaxation change allowed.	
DRLXCC	The maximum diffusion relaxation change calculated.	
DTIMEH	Highest time step allowed (maximum).	
DTIMEI	Input time step for implicit solutions.	

DTIMEL Lowest time step allowed (minimum).
 DTIMEU Time step used for all transient network problems.
 DTMPCA The maximum diffusion temperature change allowed.
 DTMPCC The maximum diffusion temperature change calculated.
 ENGBAL The calculated energy balance of the system.
 IDCNT A counter for STOREP and RECALL identification (integer).
 LAXFAC Number of iterations before radiation conductor is linear (integer).
 LINECT A line counter location for program output.
 LØØPCT Program count of iteration loops performed (integer).
 NLØØP User input number of iteration loops desired (integer).
 ØPEITR Causes output each iteration if set nonzero.
 ØOUTPUT Time interval for activating OUTPUT CALLS.
 PAGECT A page counter location for program output.
 TIMEM Mean time for the computation interval.
 TIMEN New time at the end of the computation interval.
 TIMEND Problem stop time for transient analysis.
 TIMEØ Old time at the start of the computation interval, also used as problem start time, may be negative.

ITEST,JTEST,KTEST,LTEST,MTEST are dummy control constants with integer names.

RTEST,STEST,TTEST,UTEST,VTEST are dummy control constants with noninteger names.

The following is representative of a constants data block:

```

8
↓
BCD 3CONSTANTS DATA
    TIMEND,10.0,OUTPUT,1.0           $CONTROL CONSTANTS
    1,10,2,3,3,7,4,8                $INTEGERS
    5,1.,6,1.E3,7,1.E-3             $FLOATING POINT
    8,TEMP,9,ALPHA                   $ALPHANUMERIC
END
    
```

Array Data Block

Array data are exceedingly simple to enter. The user inputs an array number, sequentially lists his information, and terminates it with an END (data END, not mnemonic). Numerous subroutines (interpolation, matrix, etc.) require that the exact number of values in an array be specified as an integer. In order to reduce the number of subroutine arguments and chance of error, the CINDA preprocessor counts the number of values in an array and supplies this integer count as the first value in the array. The writeup of any subroutine whose array arguments require the array integer count will list the array argument as A(IC). Subroutines whose array arguments require the first data value rather than the integer count will list the array argument as A(DV). When a user inputs the array number as positive, the integer count is calculated by the preprocessor and supplied as the first value in the array. For example,

```

12
↓
1,1.6,2.4,3.8,END

```

The above array (Array 1) contains three data values and was input as a positive array. By addressing A1 as a subroutine argument the integer count 3 would be the first value followed by 1.6, 2.4 and 3.8. If the user wanted the 1.6 data value to be addressed the argument should be A1+1. The user has the option of placing a minus sign on the input array number. In this event the integer count of data values in the array is not calculated or stored and addressing the array as A1 obtains the first data value. For example:

```

12
↓
-2,1.6,2.4,3.8,END

```

Entering the argument A2 would address the 1.6 data value; the integer count is not available. The following is an example of various types of arrays.

```

8
↓
BCD 3ARRAY DATA
    1,1.6,2.4,3.8,END          $FLOATING POINT NUMBERS
    2,TEMP1,TEMP2,END         $ALPHANUMERIC
    3                          $ALPHANUMERIC
BCD 3TEMPERATURE STUDY
END
    -4,SPACE,100,END          $SPACE OPTION
END

```

Two types of alphanumeric inputs are shown above. The first allows each word to be separated by a comma, requires each word to start with a letter, and does not allow the use of blanks. The second requires use of the BCD mnemonic code and the integer word count. It allows use of letters, numbers, or characters anywhere and retains blanks. The space option is an easy way for the user to specify a large number of locations which are initialized by the preprocessor as floating-point zeros. The space option requires the word SPACE followed by the number of locations to be initialized. It may be used anywhere in an array and as many times as desired as long as total available core space is not exceeded.

Program Control

Aside from the title block, there are either two or four data blocks depending upon whether the problem is GENERAL or THERMAL, respectively. No matter which, there are also four operations blocks entitled EXECUTION, VARIABLES 1, VARIABLES 2, and OUTPUT CALLS. The operations or instructions called for in these blocks determine the program control. They are preprocessed by CINDA and passed on to the system FORTRAN compiler as four separate subroutines entitled EXECTN, VARBL1, VARBL2, and OUTCAL, respectively. When the FORTRAN compilation is successfully completed, control is passed to the EXECTN subroutine which sequentially performs the operations

in the same order as entered by the user in the EXECUTION block. None of the operations specified in the other three blocks will be performed unless they are called for, either directly by name in the EXECUTION block or internally by some other called-for subroutine.

No operations will be performed unless requested by the user, and, no control constants will be utilized unless some subroutine calls for them. Network solution subroutines internally call upon VARBL1, VARBL2, and OUTCAL (see Fig. 3). They also use numerous control constants, but their individual writeups in Section VIII must be consulted to determine which ones and their exact usage. Network solution subroutines require no arguments but most others do. These arguments may be addresses which refer to the location of data or they may be literals; i.e., the actual data value. All of the input data can be addressed by using alphanumeric arguments of the following form.

TN for the temperature location of node N
 CN for the capacitance location of node N
 QN for the source location of node N
 GN for the conductance location of conductor N
 KN for the value location of constant N
 AN for the starting location of array N

and control constants utilize their individual names.

When addressing arrays the user must be cautious as to the use of positive or negative arrays and address them accordingly. However, the user may uniquely address any item in an array. For instance, the one-hundredth value in a positive array ten could be uniquely addressed as A10+100. This plus option is available only for arrays. If perhaps a user desired to address the 20 BCD words for the title block which were retained for output page headings, he could do so by using the argument H1.

Dynamic Storage Allocation is a unique feature of the CINDA-3G program. Although not carried to the ultimate, all subroutines which require working space generally obtain it from a common working array. However, it is up to the user to specify information about this array to the program. To do so, the user must place three FORTRAN cards at the start of the Execution block, the first of which must come *before* the BCD 3EXECUTION card. For example,

```

1      7          21  25
↓      ↓          ↓   ↓
F      DIMENSION X( 100)
        BCD 3EXECUTION
F      NDIM = 100
F      NTH = 0

```

In the DIMENSION card, columns 21-25 *must* be reserved for the integer which must be in an I5 format. The names used must be exactly as shown and in the above would cause a working array of 100 locations to be created. If fewer or more locations are needed, the integer 100 may be changed as desired (both for DIMENSION and NDIM). If no working locations are required, the cards should be omitted.

An F in column 1 indicates to the preprocessor that the card is FORTRAN and should be passed on as received. This F option allows the user to program FORTRAN operations directly into the operations blocks. However, the CINDA arguments listed above are not FORTRAN compatible with the exception of the control constant names. Therefore, it is recommended that the program user utilize CINDA subroutine calls wherever possible. This is impossible however when logical operations are required. In this case it is recommended that the user place CINDA data values as needed into the available dummy control constant names allowed for that purpose. Then, FORTRAN logical operations can be utilized with the dummy control constant names as arguments. FORTRAN statement numbers for routing purposes may be placed in columns 2-5 on any operations cards.

The data field for node, conductor, constant, and array data consists of columns 12-80. However, the data field of operations cards ends with column 12. In a manner of speaking, a CINDA subroutine call is a special array and should terminate with a data END. In order to simplify input for the user, the operations read subroutines recognize two special characters; the left and right parentheses. The left parenthesis is accepted as a comma, while the right parenthesis is accepted as a comma followed by a data END. This allows what would have been

```

12
↓
ADD,K1,K2,K3,END
    
```

to be more aesthetically formatted as

```

12
↓
ADD(K1,K2,K3)
    
```

which is almost identical to a FORTRAN subroutine call.

Execution Operations Block

An Execution operation block might be as follows:

1	7	12	21	25	
↓	↓	↓	↓	↓	
F	DIMENSION		X(25)	
	BCD 3EXECUTION				
F	NDIM = 25				
F	NTH = 0				
F	10	TIMEND = TIMEND + 1.0			
		CNFRWD			\$EXPLICIT FORWARD DIFFERENCING
		STFSEP(T20, TTEST)			\$PLACE T10 INTO DUMMY CC
F	IF(TTEST .LE. 100.) GO TO 10				
	END				

MARY E. GEALY

The above indicates a transient thermal problem in which the user desires to terminate the analysis when the temperature at node 20 exceeds 100°F. The problem must have been fairly small because only 25 working locations were dimensioned and CNFRWD requires one per node. It does demonstrate the use of both CINDA calls and FORTRAN operations and that control constants are referred to by name in either. Another example might be

```

1      8  12      21  25
↓      ↓  ↓      ↓  ↓
F      DIMENSION X( 500)
        BCD 3EXECUTION
F      NDIM =500
F      NTH =0
        CINDSL                      $ STEADY STATE (USES LPCS)
F      TIMEND = 10.0
        CNFRWD                      $ TRANSIENT ANALYSIS (USES SPCS)
        END

```

In this case the user desires to have a steady state analysis performed on the network and then a transient analysis performed utilizing the steady state answer as initial conditions. However, the two-network solution subroutines referred to are incompatible in their PCS requirements and the program would be terminated with an appropriate error message. A further example might be

```

8  12
↓  ↓
BCD 3EXECUTION
      INVERSE(A1,A2)                $ SEE MATRIX SUBROUTINE
      MULT(A2,A3)                   $ WRITEUPS FOR OPERATIONS
      LIST(A2,K1,17)                $ PERFORMED
      LIST(A3,K2,17)
END

```

The above problem consists entirely of matrix operations and therefore is run as a GENERAL. The subroutines do not require any working space so none have been dimensioned. Furthermore, no reference, direct or indirect, is made to VARBL1, VARBL2, or OUTCAL, and those operations blocks should be empty. Even though they may be empty or not referred to, their blockheader and mnemonic END cards must still be entered.

There is no end to the variety of examples that could be generated. In reality, the program user is actually programming, although it is somewhat disguised as data input. However, the program does simplify the task of data logistics, and it automates data input and output, construction of the PCS, loading the subroutine library, and other systems features, thereby greatly lessening the programming knowledge which might otherwise be required of a user.

A point well worth considering is proper initialization. All instructions contained in the other three operations blocks are performed each iteration or on the output interval. If an operation being performed in Variables 1 is utilizing and producing nonchanging constants, it should be placed in the Execution block (prior to the network solution call) so that it will be performed only once. Input arrays requiring postinterpolation multiplication

for units conversion only could be prescaled, thereby deleting and multiplication process. Complex functions of a single independent variable requiring several interpolation values which are then combined in a multiplicative fashion can be precalculated vs the independent variable. Such a precalculated complex function reduces the amount of work performed during the transient analysis. A great many operations of this type can be performed in the Execution block prior to call for a transient analysis. Also, output operations to be performed once the transient analysis is completed may be placed directly into the Execution block following the transient network solution call.

Variables 1 Operations Block

The statement that this program solves nonlinear partial differential equations of the diffusion type is not quite accurate. In reality the program only solves linear equations. However, nonlinearities are evaluated at each computation interval and in this manner generally yield acceptable answers to nonlinear problems. This method is more properly termed quasilinearization. The Variables 1 operation block allows a point in the computational sequence at which the user can specify the evaluation of nonlinear network elements, coefficients, and boundary values (see Fig. 3). The CGS and CGD mnemonic codes utilized for node and conductor data cause the construction of various subroutine calls which are placed in this block by the CINDA preprocessor. The user must specify any additional subroutine calls necessary to completely define the network prior to entering the network solution phase.

Prior to inclusion of the CGS and CGD mnemonic codes, the Variables 1 operations block primarily consisted of linear interpolation subroutine calls input by the user for the evaluation of temperature varying properties. While these linear interpolation calls are automated through use of the CGS and CGD codes, it is up to the program user to specify any required bivariate or trivariate interpolations or other functional evaluations necessary. Just prior to performing the Variables 1 operations, all network solution subroutines zero out all source locations. Therefore, the user is required to specify constant as well as variable or nonlinear impressed sources in this block. A Variables 1 operations block could be as follows:

```

1      8   12
↓      ↓   ↓
      BCD 3VARIABLES 1
          STFSEP(10.0,Q17)           $CONSTANT IMPRESSED SOURCE
          D1DEG1(TIMEN,A8,Q19)       $TIME VARYING SOURCE
          D2DIWM(T18,TIMEN,A19,7.63,G18) $BIVARIATE FUNCTION
F      TTEST=11.6
F      IF (TIMEN.GT. 10.)TTEST =0.0
          STFSEP(TTEST,Q27)         $VARIABLE SOURCE
      END

```

The first call above places a constant heating rate of 10.0 into the source location of node 17. The second call causes a linear interpolation to be performed on array 8 using mean time as the independent variable to obtain a time-varying heating rate for node 19. The third call uses mean time and the temperature at node 18 as independent variables to perform a bivariate interpolation. The interpolated answer is then multiplied by 7.63

and placed as the conductance value of conductor 18. The next two cards are FORTRAN and allow a value of 11.6 to be placed into control constant TTEST until TIMEN exceeds 10.0, after which a value of 0.0 is placed into TTEST. This amounts to a single step in a "staircase" function. The last card places the value from TTEST into the source location for node 27. Another sample Variables 1 block might look as follows:

```

8      12
↓      ↓
BCD 3VARIABLES 1
      BLDARY(A12+1,T1,T7,T3,T4)          $CONSTRUCT VECTOR
      D1DEGI(T7,A19,A13+2)              $INTERPOLATION
      IRRADE(A7,A13,A10,A12)            $IR RADIOSITY EXPLICIT
      BRKARY(A12+1,Q1,Q7,Q3,Q4)         $DISTRIBUTE Q RATES
      D1D1WM(TIMEM,A9,0.35,TTEST)
      ADD(TTEST,Q1,Q1)                  $ADD TWO RATES
END

```

The first call causes the construction of an array of four temperature values necessary as input to an infrared radiosity subroutine (third card). The second call causes the linear interpolation of a temperature-varying property from array 19 to be placed into array 13 + 2 which is the second array argument for the radiosity call. This second argument must be an array of surface emissivities for the surfaces under consideration; therefore array 19 must be an array of temperature-varying emissivity. The BRKARY call takes data values from array 12 + 1, 2, 3, and 4 and places them into the source locations for nodes 1, 7, 3, and 4, respectively. The fifth call performs linear interpolation on array 9 using TIMEM as the independent variable, multiplies the result by 0.35 and places it in control constant TTEST. This might be a time-varying solar heating rate where 0.35 is the solar absorptivity. The ADD call adds this rate to what is already contained in the source location for node 1. Each node has one and only one source location. If a user desires to impress more than one heating rate on a node, he must sum the rates and supply the value to the single source location available per node.

The Variables 1 operations block is the logical point in the network computational sequence for the calculation of impressed sources whether they are due to internal dissipation of power components, radiation deposition, aerodynamic heating, or orbital heating. If a desired subroutine is not available, the user may always add his own; data communication is obtained through subroutine arguments as in any other subroutine.

Variables 2 Operations Block

With regard to the network solution, the Variables 1 operations may be thought of as presolution operations and the Variables 2 operations as postsolution operations. In Variables 1 the network was completely defined with respect to nonlinear elements and boundary conditions. Variables 2 allows the user to look at the network just solved. He may meter and integrate flow rates, make corrections in order to account for material phase changes, or compare answers just calculated with test data in order to derive empirical relationships. A simple Variables 2 operations block might be as follows:

```

8   12
↓   ↓
BCD 3VARIABLES 2
    QMETER(T1,T2,G1,K1)           $METER HEAT FLOW
    QINTEG(K2,DTIMEU,K2)         $INTEGRATE HEAT FLOW
    RDTNQS(T5,T1,G8,K3)         $METER RADIATION FLOW
    QINTEG(K3,DTIMEU,K4)         $INTEGRATE RADIANT FLOW
    ADD(K2,K4,K5)
END

```

The first call measures the heat flow from node 1 to node 2 through regular conductor 1 and stores the result in constant location 1. The second call performs a simple integration with respect to time and sums the result into constants location 2. The third call measures heat flow through a radiation conductor which is then integrated by the fourth call. The sum of the two integrations is obtained by the fifth call. Another Variables 2 operations block might be as follows:

```

8   12
↓   ↓
BCD 3VARIABLES 2
    ABLATS(A1,1.76,K8,A7,T15,C15)  $ABLATIVE ON NODE15
END

```

Phase change subroutines such as the above are unique in that they perform automatic corrector operations. Node 15 has been solved by the network solution subroutine as though no ablative existed. The ABLATS subroutine then corrects the temperature at node 15 to account for the ablative material. It does this by calculating the average heating rate to node 15 over the time step just performed and utilizes it as an inner-surface boundary condition for the internally constructed one-dimensional network representation of the ablative material. The correctness of this analytical approach can be rigorously substantiated for use with explicit network solution subroutines. However, when used with large time step implicit methods it yields a controlled instability and the results may be questionable. It is up to the user to determine the solution accuracy by whatever means available. A more complicated Variables 2 operations block could be as follows:

```

1   5   8   12
↓   ↓   ↓   ↓
    BCD 3VARIABLES 2
        D1DEGI(TIMEN,A10,K8)           $GET TEST TEMPERATURE
        SUB(T8,K8,TTEST)               $OBTAIN TEMP DIFFERENCE
F    IF(TTEST.LE.2.0)GO TO 10
        MLTPY(G7,0.99,G7)              $REDUCE CONDUCTANCE
    5    STFSEP(-1.0,BACKUP)           $SET BACKUP NON-ZERO
F    GO TO 20
F  10  IF(TTEST.GE.-2.0) GO TO 15
        MLTPY(G7,1.01,G7)              $INCREASE CONDUCTANCE
F    GO TO 5
    15  QMETER(T8,T15,K9)
        QINTEG(K9,DTIMEU,K10)
F  20  CONTINUE
      END

```

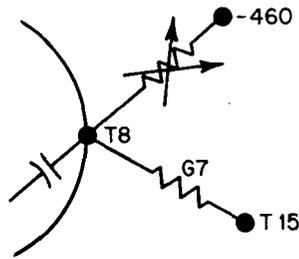


Fig. 4—Three-dimensional network

This corresponds to a portion of a network such as shown in Fig. 4.

Array 10 is a time-temperature test history of node 8, and node 15 is a known boundary reference temperature. The problem is to calculate the value of conductor 7 which will yield a calculated temperature at node 8 that is within ± 2.0 degrees of the test history. The above Variables 2 operations will attempt to modify conductor 7 so that it will meet the constraints on temperature 8. It is quite "brute force" and unsophisticated. However, the corrector operations are at the discretion of the user. If the tolerances were too severe or the correction operations too strong, the correction for one tolerance could lead to dissatisfaction of the other and an impasse result. If the reference temperature at node 15 were incorrect, possibly no value of conductor 7 would satisfy the constraints. The end result of such a study would be to produce a plot of conductance 7 vs time which could be used to derive an empirical relationship with other parameters. Too wide a tolerance would cause the plot to resemble a staircase function. Please note that either condition being unsatisfied causes control constant BACKUP to be nonzero and the iteration to be redone with the corrected conductor 7 value. Only when all criteria are met are the metering and integration operations performed.

Output Calls Operations Block

This operations block could have been entitled Variables 3 but Output Calls seemed more appropriate. In it a user may call upon any desired subroutine. However, its contents are performed on the output interval (see Fig. 3), so it is only logical that it would primarily contain instructions for outputting information. There is a variety of output subroutines offering the user several format options. A very simple Output Calls block would be as follows:

```

8   12
↓   ↓
BCD 3OUTPUT CALLS
    PRNTMP
END
    
```

The above call will output certain time control information and the temperature of every node in the network under consideration. The node temperatures will correspond to the relative node numbers set up by the preprocessor, not the actual node numbers set by

the user. The preprocessor lists out all of the input data. Immediately after the input node data a dictionary of relative node numbers vs actual node numbers is listed. By utilizing it a user can correlate the relative node temperatures with his actual numbers.

The Output Calls will be performed at problem start time and on the output interval until problem stop time is reached. For example, a 100-min transient analysis with an output interval of 5 min would cause the Output Calls operations to be performed 21 times.

The above data and operations blocks constitute a problem data deck which must be terminated by the following card:

```

      8   12
      ↓   ↓
    BCD 3END OF DATA

```

Parameter Runs

Parametric analyses which do not involve network of operations changes to the original problem may be performed on the same computer run. Only data values such as output page heading, temperatures, capacitances, conductances, constants, and arrays may be changed. The data change blocks must all be specified whether changes occur in the block or not, and the data input is identical to the preceding discussion with the exception of conductors. When specifying new conductances, the adjoining node information is deleted; only the conductor number and value are required. The only mnemonics allowed are the three blanks and BCD. When changing an array, the entire new array must be entered and be exactly the length of its original. No new arrays or numbered constants may be defined.

Two parametric run options are available, INITIAL and/or FINAL, and they may be used several times within the problem data deck. The problem data deck as initially entered is referred to as the original problem. Any and all INITIAL parameter runs refer to it exactly as it was put in. The FINAL parameter run refers to the problem just completed exactly as terminated. When two INITIAL parameter runs are attached to the end of a problem data deck, they both refer to the original problem at start time. However, when two FINAL parameter runs are attached to the end of a problem data deck, the first refers to the original as terminated, and the second refers to the first FINAL parameter run as completed. The CINDA control cards necessary to specify a parameter run are as follows:

```

      8   12
      ↓   ↓
      BCD 3INITIAL PARAMETERS
or   BCD 3FINAL PARAMETERS
      END
      BCD 3NODE DATA
      END
      BCD 3CONDUCTOR DATA
      END
      BCD 3CONSTANTS DATA
      END
      BCD 3ARRAY DATA
      END

```

The parameter run decks are inserted in the problem data deck immediately preceding the BCD 3END OF DATA card. After the BCD parameter card, the user may insert additional BCD data to replace the original problem output page heading. Parameter runs conserve machine time mainly because the PCS does not have to be reformed. If a user desires, he may accomplish FINAL parameter runs by calling the network execution subroutine twice in the Execution block and inserting the necessary calls to modify data values between them.

Store and Recall Problem Options

The purpose of the store and recall options is to provide the user with the means to interrupt his program at any point, store the current data values on tape, and continue processing. While the parameter run capability is useful for performing parametric analyses in the same run, the store and recall capability allows an indefinite time lapse between parametric analyses. In addition, long duration problems may be broken into several short duration runs. If a parametric analysis is such that the first portion of the runs are identical, then the problem can be run for the constant portion, stored and then recalled as many times as necessary.

The store problem feature is achieved by a user initiated subroutine call which is as follows:

```

12
 ↓
STOREP(KX)

```

where KX refers to a constant location containing an alphanumeric identification name for the stored problem. The call may be used as many times as desired, but each activation must reference a unique name. To allow the STOREP call to be placed in a loop, the programmer must use the control constant IDCNT. By incrementing this constant within the loop (not to exceed 999), a unique combination of KX and IDCNT will be available to identify the problem.

The recall problem feature is a CINDA preprocessor option, which is activated by the following card:

1	13	22
↓	↓	↓
RECALL	AAAAAA	NNN

where AAAAAA is the alphanumeric identification name of the stored problem, and NNN is the integer IDCNT. (If IDCNT was not used with STOREP, NNN = 0). This single card replaces the blank card preceding the problem data deck and must be followed by a BCD 3INITIAL PARAMETERS data deck. The stored problem identified will be searched for and brought into core from the two storage tapes. Any data changes specified will be performed and the control is passed to the first subroutine call in the EXECUTION block.

The problem is stored on logical unit 22 and recalled from unit 21; the processor (tape 40) must be saved in a store run and remounted on a recall run. The user must remember that the recalled problem contains the STOREP call. Because of this feature, the user has the option whether or not to store the problem again. Logical unit 22 is equipped depending on this option. Section VII should be consulted for details concerning deck setups, tape usage, and Job Request forms.

Data Printing Option

At times, the user may wish to see what is being stored on the data tapes during preprocessing (i.e., LUT1, LUT2, LUT3, LUT4, and LB3D). A printout will occur if an asterisk is put in column 80 of the first BCD card (the GENERAL, THERMAL, INITIAL PARAMETERS, or FINAL PARAMETERS card). After each block of the problem deck is printed, there will be a listing of the appropriate data (most of which will be binary).

IV. ERROR MESSAGES

Due to the variety of subroutines available and the variable number of arguments which some of them have, no check is made to determine if a subroutine call has the correct number of arguments. An incorrect number of arguments on a subroutine call will generally cause job termination immediately after successful compilation, usually without any error message. If the above occurs, the user should closely check the number of arguments for his subroutine calls.

Numerous error messages can be put out by the preprocessor. These error messages are listed below and are grouped according to various preprocessor functions. All error messages are preceded by three asterisks, which have been deleted below. Self-explanatory messages are not enlarged upon.

Processing Data Blocks—

DATA BΛØCKS IN IMPRØPER ØRDER ØR ILLEGAL BΛØCK
DESIGNATIØN ENCØUNTERED.

AN IMBEDDED BLANK HAS BEEN ENCØUNTERED IN THE LAST LINE.

BLANK CØUNT ØF 10 HAS BEEN EXCEEDED.

INTEGER FIELD EXCEEDS 10.

REAL NUMBER FIELD EXCEEDS 20.

ALPHAMERIC FIELD EXCEEDS 6.

MULTIPLE DECIMAL POINTS HAVE BEEN ENCOUNTERED.

NODES MUST BE ORDERED - DIFFUSION, ARITHMETIC, BOUNDARY.

CONDUCTORS MUST BE ORDERED - REGULAR, RADIATION.

NODE NUMBER, XXXXX, IS THE DUPLICATE OF THE XXXXXTH NODE.

CONDUCTOR NUMBER, XXXXX, IS THE DUPLICATE OF THE XXXXXTH
CONDUCTOR.

CONSTANT NUMBER, XXXXX, IS THE DUPLICATE OF THE XXXXXTH
CONSTANT.

ARRAY NUMBER, XXXXX, IS THE DUPLICATE OF THE XXXXXTH ARRAY.

FIXED CONSTANT NAME NOT IN LIST.

NUMBER OF GEN ARGUMENTS, XXX, EXCEEDS NUMBER REQUIRED.

STORAGE ALLOTTED FOR THIS DATA BLOCK HAS BEEN EXCEEDED.
PROCESSING WILL RESUME WITH THE NEXT DATA BLOCK.

Forming PCS—

NODE, XXXXX, HAS NO MATCH IN THE NA-NB PAIRS.

ADJOINING NODE, XXXXX, OF NA-NB PAIR HAS NO MATCH IN THE NODAL
BLOCK, CONDUCTOR IS NO., XXXXX

Processing Program Blocks—

EXECUTION BLOCKS IN IMPROPER ORDER OR ILLEGAL BLOCK
DESIGNATION ENCOUNTERED.

Explanation: Some alpha character other than K or A has been used to reference a
data block. In a thermal problem a designator other than G, K, or A
is assumed to be referencing the nodal block.

MISSING NODE NUMBER, XXXXX.

MISSING CONDUCTOR NUMBER, XXXXX.

MISSING CONSTANT NUMBER, XXXXX.

MISSING ARRAY NUMBER, XXXXX.

FIXED CONSTANT NAME, AAAAA, NOT IN LIST.

NUMBER OF SUBROUTINES REQUESTED EXCEEDS 75.

Explanation: More than 75 unique subroutines have been called.

Processing Parameter Changes—The first five parameter change error messages are prefaced with the words: PARAMETER CHANGE ERROR.

NODE NUMBER, XXXXX, WAS NOT DEFINED IN THE ORIGINAL PROBLEM.

CONDUCTOR NUMBER, XXXXX, WAS NOT DEFINED IN THE ORIGINAL PROBLEM.

CONSTANT NUMBER, XXXXX, WAS NOT DEFINED IN THE ORIGINAL PROBLEM.

ARRAY NUMBER, XXXXX, WAS NOT DEFINED IN THE ORIGINAL PROBLEM.

CONSTANTS BLOCK WAS EMPTY IN THE ORIGINAL PROBLEM.

ARRAY BLOCK WAS EMPTY IN THE ORIGINAL PROBLEM.

ARRAY NUMBER XXXXX - DIMENSIONS NOT EQUAL. ORIGINAL, XXXXX, CHANGE, XXXXX.

Terminations Due to Errors (no preceding asterisks)—

THE ABOVE PARAMETER CHANGE WILL NOT BE EXECUTED.

ERROR TERMINATION - LOADING IS SUPPRESSED.

V. OPERATING SYSTEM DESCRIPTION

General

The CDC-3800 (FORTRAN IV) version of CINDA exists logically as a preprocessor, processor, and library. The operational continuity of these portions is made possible by the CDC Drum SCOPE system (see Fig. 5).

The function of the preprocessor is to operate on a user-supplied problem and produce the following items.

1. Processor Main Program—This small routine acts primarily as a communications link in providing addressing relationships between the operational user program and user data.

2. User Program—These FORTRAN source subroutines are operational equivalents of the user' Execution, Variables 1 and 2, and Output Calls blocks.

3. User Data—Binary data generated consists of definitions of parameters referenced in the various user data blocks and their corresponding values.

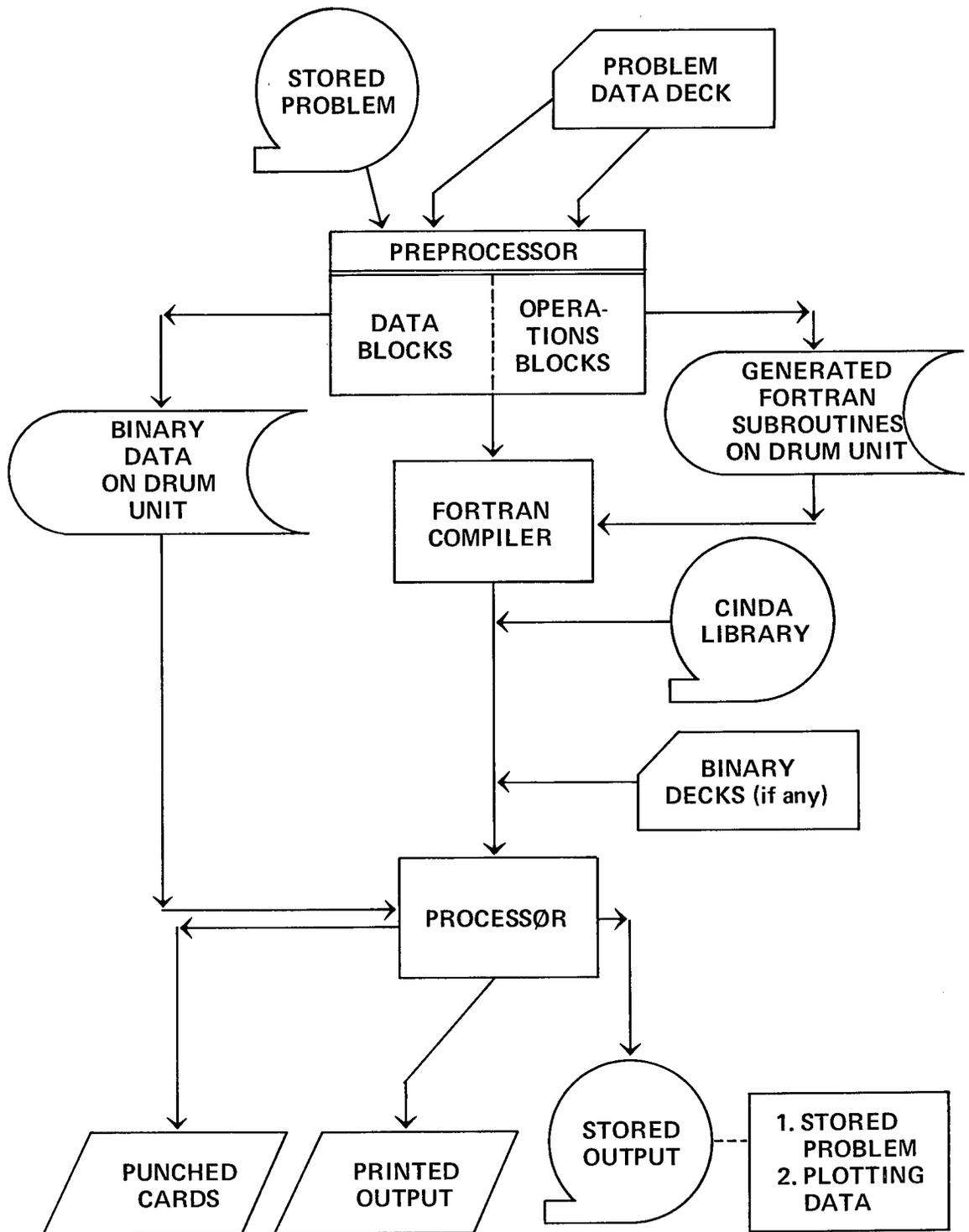


Fig. 5—Flow of CINDA operating system

The preprocessor and appropriate use of the CDC-3800 system control cards allows construction of the above from tape when the RECALL option is utilized.

The processor performs reading of the user data values prepared previously and calls the user program (i.e., Execution block).

The CINDA library contains a large number of various types of subprograms to accomplish most user requirements. Drum SCOPE's LIBEDIT provides simple, flexible methods for the maintenance of this library. In addition, it is not necessary that a subroutine be updated to the library prior to availability in the user problem.

Preprocessor

Operation of the Preprocessing Phase.—(See Fig. 6 for flowchart.) The main program PREPRO accomplishes the initialization of data values and tape units and defines the order of processing by calling seven subroutines.

1. If the problem being processed is a RECALL problem, subroutine SPLIT is called to read the recalled problem data and number definitions from the input tape and write these on the appropriate work tape. SPLIT calls SKIP if the input tape is not positioned at the problem being recalled (see *Store and Recall Problems Options*).
2. CODERD reads the title block and the block title cards. It then calls DATARD, which reads the free-form data cards in the four (or two, if General Problem) data blocks and any parameter change data. Each card is read, a format is constructed for it, and then it is reread. The data from each block are written on the data tape as one record. The number definitions of the data and the NA-NB pairs are written on work tapes.
3. PSEUDO reads the node number definitions and NA-NB pairs from work tape. The PCS (long or short) is constructed, packed by PACK43 and flagged by ORMIN, and written on the data tape. PACK43 and ORMIN call BIT, a COMPASS packing and unpacking routine.
4. GENLNK constructs the main program of the processor (LINK0), including COMMON and DIMENSION information. BLKCRD and STFFB are called upon to fill an array with FORTRAN source code, which is then written onto logical unit LB4P by WRTBLK. WRTSCOPE writes SCOPE at the end of LB4P after completion of the other four subroutines (EXECTN, VARBL1, VARBL2, and OUTCAL).
5. PRESUB reads the title cards of the four program blocks and initiates the construction of each new subroutine. CINDA4 converts the CINDA "calls" in the program blocks into FORTRAN subroutine calls. Data referenced by input number definition is changed to refer to its relative location in COMMON data arrays.
6. INITAL combines the original set of the data and the initial parameter changes and writes the updated set of data on the data tape.

7. FINAL converts final parameter change data (number definitions and values) to relative array locations and values and writes number-value records on the data tape.

VI. EXTERNAL PLOTTING PACKAGE

CINDA's plot package, for use on the CALCOMP plotter, is an external program that will plot a graph of time vs temperature for each problem node. The input to this program is an output file (unit 24) generated by TSAVE during a previous CINDA problem run. The program can be run separately from the CINDA problem using the tape from TSAVE as input, or it can be placed behind the CINDA problem in a single run. In the latter case, unit 24 may be either a drum unit or a tape equipped for later use.

The package, available as a binary deck, consists of three routines. The main program, PLOTTEMP, calls PLOTPREP, which rearranges the data from the input tape and writes it on unit 25. Unit 25 contains the actual node numbers, the time array, and the temperature profile for each node. PLOTTEMP then reads a set of data cards which give the plot heading, X- and Y-axis limits, and nodes to be plotted. The temperature array for each node is read, and if the node is to be plotted, PLOTT is called, which in turn activates CALCOMP routines. A separate set of axes is drawn for each node. When all temperatures have been read, the tape is rewound and a new set of data (if any) is read.

Data Set

A data set consists of at least three cards.

*CARD 1—TITLE

Columns 1-40 will be used as the plot heading.

*CARD 2—AXIS LIMITS and TEMPERATURE SCALE OPTION

Four floating-point values must be entered in an E9.2 format. The minimum and maximum times (X-axis) must be in fields 4-12 and 14-22, respectively; the minimum and maximum temperatures (Y-axis) must be in fields 24-32 and 34-42, respectively. Column 43 contains the temperature scale option. A blank indicates that the data will be plotted in Fahrenheit temperatures, and a 1 specifies Centigrade.

4	12	22	32	43
↓	↓	↓	↓	↓
NNNNNN.NN	NNNNNN.NN	NNNNNN.NN	NNNNNN.NN	NNNNNN.NN1
(TIMEMIN)	(TIMEMAX)	(TEMPMIN)	(TEMPMAX)	(Centigrade scale)

*CARD 3—NODES and OPTION

Options:

Plot all nodes—card is blank.

Plot certain nodes—column 1 contains a \$; nodes are listed.

MARY E. GEALY

Plot *all but* certain nodes—column 1 contains any character but * or \$; nodes are listed.

The nodes may be listed individually or in inclusive pairs. Each node must be followed by a comma except for the inclusive pair, which must be separated by a blank instead of a comma. (The second node of the pair must be followed by a comma, however.) An asterisk (*), instead of a comma, must follow the last node.

The node numbers must be in I4 format, right adjusted to columns 5, 10, 15, . . . , 80. The commas or separating blanks go in columns 6, 11, 16, . . . , 76. Data may be continued to a following card by putting a comma or blank in column 1 of that card. Data cards will be read until an asterisk is encountered (limit of nine cards).

Examples

```

1.  1    5    10    15    20
    ↓    ↓    ↓    ↓    ↓
    $ 10, 15, 25 30*
```

Plot nodes 10, 15, 25, 26, 27, 28, 29, 30. (Pairs must be in ascending order; otherwise, order of magnitude isn't important.)

```

2.  1    5    10    15    20    25
    ↓    ↓    ↓    ↓    ↓    ↓
    +  5,  8, 10 12, 15*
```

Plot all nodes but 5, 8, 10, 11, 12, 15.

```

3.  1    5    10    15    20    .....    75    80
    ↓    ↓    ↓    ↓    ↓
(Card 1) $1050,1060 1062,1070, .....    2000,2005
(Card 2)  2007,2012*
```

Plot 1050, 1060, 1061, 1062, 1070, ... , 2000, 2005, 2006, 2007, 2012. (A comma in column 1 of card 2 would exclude node 2006.)

```

4.  1    5    10    15    .....    80
    ↓    ↓    ↓    ↓
(Card 1) + 30, 25 28, .....    100
(Card 2) *
```

Plot all but nodes 30, 25, 26, 27, 28, ... , 100.

Another set of data may follow the last node card, thus enabling the programmer to redefine the title and limits and to plot different nodes. Any number of data sets may be used.

Diagnostics

Messages that may be encountered while plotting are listed below. The first two will not cause job termination.

1. *** SOME NODES TO BE PLOTTED WERE NOT FOUND ON INPUT TAPE, OR WERE DUPLICATED ON NODE CARD. ***

NODES ON INPUT TAPE—
(listing of nodes)

NODES TO BE PLOTTED—
(listing of nodes)

Cause: The number of nodes to be plotted is larger than the number of nodes actually plotted.

2. IN FOLLOWING NODE CARD, AN INCLUSIVE PAIR OF NODES IS FOLLOWED BY A BLANK—A COMMA IS ASSUMED AND PROCESSING IS CONTINUED.

3. ***** WRONG FORMAT USED ON FOLLOWING NODE CARD—
PROCESS NEXT SET OF DATA, IF ANY.

Cause: A character other than a comma, blank, or asterisk has been found in a column intended for those characters only.

This error terminates the processing of a current data set, and processing continues to the next set if there is one. (Most data format errors will cause job termination by the system.)

VII. TAPE USAGE AND DECK SETUPS

This section shows deck setups and tape usage for various types of runs on the 3800 Drum SCOPE system.

CINDA Operating System

Table 1 lists the program and data files used in the preprocessor, processor, and library.

Units 15, 17, 18, 19, and 27 are preprocessing units only and are available as scratch units during processing. Units 16, 21, 22, 24, 30, 33, and 40 can also be used for that purpose if the corresponding options are not activated.

Table 1
CINDA Tape Usage

Logical Unit	Program Variable	Function
*9	—	CINDA Master tape; (file 1 contains preprocessor, file 2 contains Library).
12	LB3D	Data tape (original problem and all parameter changes).
14	LB4P	Program tape (contains generated FORTRAN routines LINK0, EXECTN, VARBL1, VARBL2, OUTCAL).
15	LUT7	Variables 1 calls generated from node and conductor data blocks.
**16	—	Matrix retrieval unit in library.
17	—	NA-NB pairs; data number definitions. (From parameter changes.)
18	LUT3	Copy of original problem data.
19	LUT4	Parameter Change data.
20	LUT1	Data number definitions.
**21	—	Problem recall data tape.
**22	—	Problem store data tape.
**24	—	Output from TSAVE.
27	INTERN	Data conversion scratch tape.
**30	KRR	FORTRAN reread unit in preprocessor. Matrix storage unit in library.
33	—	Scratch unit in STOREP.
**40	—	Binary program tape (processor) used with store and recall options.

*Equipped units.

**Equipped units depending on options. Matrix storage and retrieval requires equipping tapes 16 and 30. The STOREP option requires equipping tapes 40 and 22; the RECALL option requires 40 and 21 (and 22 if desired).

General deck structures for different kinds of CINDA runs are shown below. The character Δ denotes a 7 and 9 punch in the column, and a \triangle is for an 11(-), 0, 7, 9 punch. The number enclosed in parentheses in the job card (e.g., 5) may have to be increased if several tapes are used. The current label for tape unit 9 is CINDA MASTER, 1,1,999. The other tapes may be unlabeled. Corresponding job request forms are found in Fig. 7.

MAGNETIC TAPES				
LOGICAL UNIT #	INPUT	OUTPUT	SAVE	TAPE SERIAL NO.
<u>9</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>CINDA MASTER</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

SEE REVERSE SIDE FOR ADDITIONAL INSTRUCTIONS
NDW-NRL-10462/6006 (5-66)
IBM B45077

(a) Not RECALL (ordinary run)

MAGNETIC TAPES				
LOGICAL UNIT #	INPUT	OUTPUT	SAVE	TAPE SERIAL NO.
<u>9</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>CINDA MASTER</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
<u>40</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<u>Proc-essor</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
<u>22</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<u>Stored Problem</u>

SEE REVERSE SIDE FOR ADDITIONAL INSTRUCTIONS
NDW-NRL-10462/6006 (5-66)
IBM B45077

(b) Not RECALL (STOREP run)

MAGNETIC TAPES				
LOGICAL UNIT #	INPUT	OUTPUT	SAVE	TAPE SERIAL NO.
<u>9</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>CINDA MASTER</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
<u>40</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>Proc-essor</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
<u>21</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>Stored Problem</u>

SEE REVERSE SIDE FOR ADDITIONAL INSTRUCTIONS
NDW-NRL-10462/6006 (5-66)
IBM B45077

(c) RECALL

MAGNETIC TAPES				
LOGICAL UNIT #	INPUT	OUTPUT	SAVE	TAPE SERIAL NO.
<u>9</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>CINDA MASTER</u>
<u>40</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>Proc-essor</u>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
<u>21</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>stored Problem</u>
<u>22</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<u>new stored Problem</u>

SEE REVERSE SIDE FOR ADDITIONAL INSTRUCTIONS
NDW-NRL-10462/6006 (5-66)
IBM B45077

(d) RECALL (Re-store new problem)

Fig. 7—Job Request forms (magnetic tape portions)

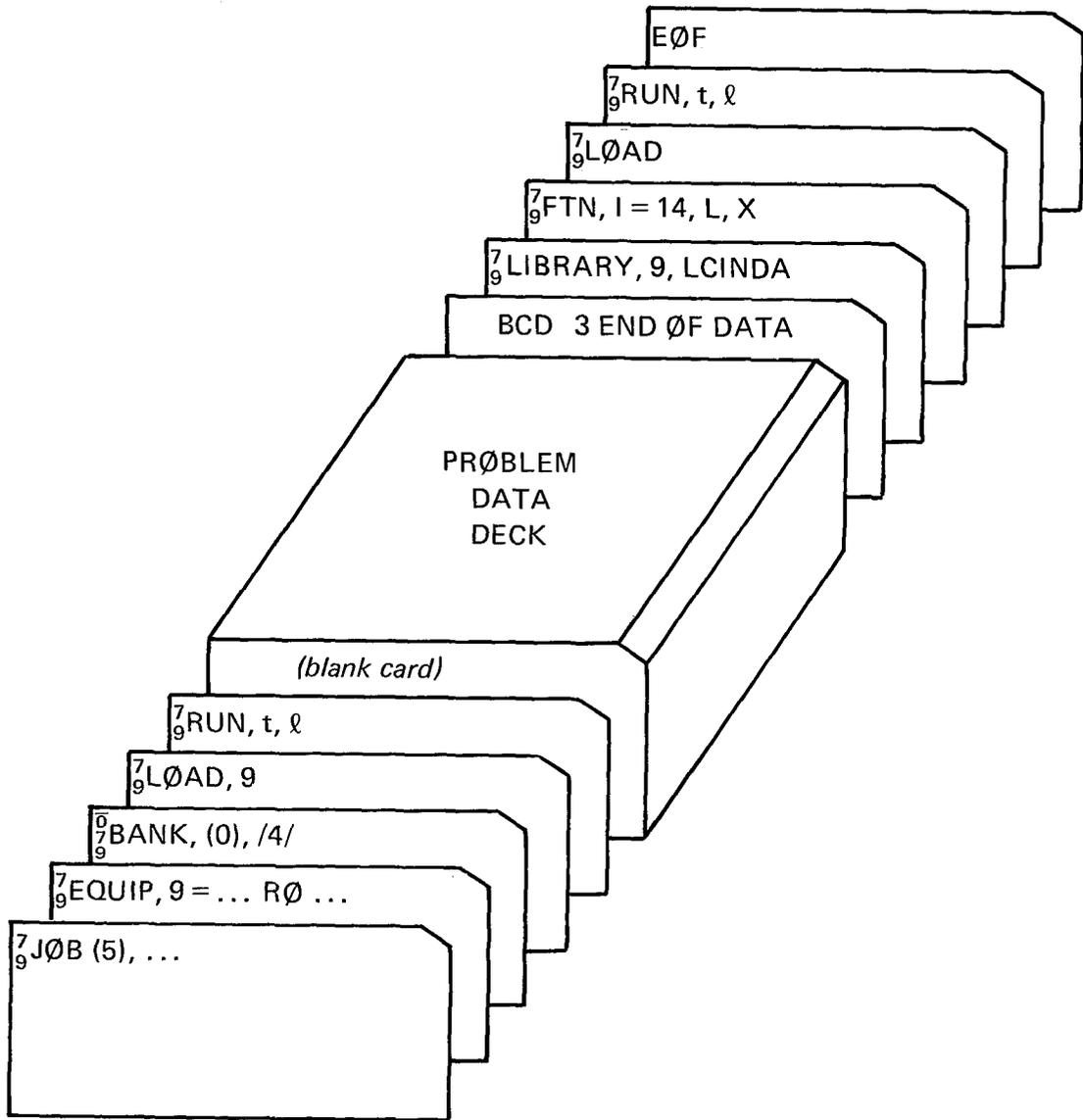


Fig. 8—Ordinary CINDA run

Not RECALL (Ordinary Run)—(See Fig. 8.)

$\Delta J\emptyset B(5), \dots$
 $\Delta EQUIP, 9 = (\text{label}), R\emptyset, HI, DA$
 $\Delta BANK, (0), /4/$
 $\Delta L\emptyset AD, 9$
 $\Delta RUN, t, \ell$
 blank card
 .
 .
 problem data deck through BCD 3END $\emptyset F$ DATA
 .
 .
 $\Delta LIBRARY, 9, LCINDA$
 $\Delta FTN, I=14, L, X$
 $\Delta L\emptyset AD$
 binary decks, if any
 .
 .
 $\Delta RUN, t, \ell$
 $E\emptyset F$

Not RECALL (STOREP run)

$\Delta J\emptyset B(5), \dots$
 $\Delta EQUIP, 9 = (\text{label}), R\emptyset, HI, DA$
 $\Delta EQUIP, 40 = (\text{label}), RW, HI, DA$
 $\Delta EQUIP, 22 = (\text{label}), W\emptyset, HI, DA$
 $\Delta BANK, (0), /4/$
 $\Delta L\emptyset AD, 9$
 $\Delta RUN, t, 1$
 blank card
 .
 .
 problem data deck through BCD 3END $\emptyset F$ DATA (includes at least one call to
 STOREP)
 .
 .
 $\Delta LIBRARY, 9, LCINDA$
 $\Delta FTN, I=14, L, X=40$
 $\Delta L\emptyset AD, 40$
 .
 .
 binary decks, if any
 .
 .
 $\Delta RUN, t, \ell$
 $E\emptyset F$

RECALL

```

ΔJOB(5),...
ΔEQUIP,9=(label),RØ,HI,DA
ΔEQUIP,40=(label),RØ,HI,DA
ΔEQUIP,21=(label),RØ,HI,DA
*ΔEQUIP,22=(label),WØ,HI,DA
ΔBANK,(0),/4/
ΔLOAD,9
ΔRUN,t,ℓ
RECALL Card
.
.
.
INITIAL PARAMETERS blocks and BCD 3END ØF DATA
.
.
.
ΔLIBRARY,9,LCINDA
ΔLOAD,40
.
.
.
binary decks if any
.
.
.
ΔRUN,t,ℓ
EØF

```

For any other options using tapes, the tapes should be equipped as those shown above, using the appropriate logical unit number and label. The user must also designate whether the tape is read only (RØ), only written on, (WØ), or both (RW). In the latter case, if the tape is written on first, the output block of the job request form is checked. If the tape is read, then written on, both the input and output blocks should be checked. (Check the Drum SCOPE manual for more details.)

Plot Package

Table 2 lists the files used in the plot packages.

*Optional—used only if problem is to be restored.

Table 2

Logical Unit	Function
10	Plotting unit
24	Output from TSAVE; Input to PLOTPREP
25	Output from PLOTPREP; Input to PLOTT

NOTE: Units 25 and 10 are drum units. Unit 24 is usually a tape, but can also be a drum unit if the plotting run follows the CINDA run in the same job.

Below are sample deck sets showing a CINDA run that generates the TSAVE tape, and a plotting run that uses the TSAVE tape as input. Figure 9 shows a job request form for the plotting run. Figure 10 displays the deck setup for a combined CINDA and plotting run.

Not RECALL (Generate TSAVE tape)

```

ΔJØB(5),...
ΔEQUIP,9=(label),RØ,HI,DA
ΔEQUIP,24=(label),WØ,HI,DA
ΔBANK,(0),/4/
ΔLØAD,9
ΔRUN,t,ℓ
blank card
.
.
.
problem data deck through BCD 3END ØF DATA (includes a call to TSAVE
in Output Calls)
.
.
.
ΔLIBRARY,9,LCINDA
ΔFTN,I=14,L,X
ΔLØAD
.
.
.
binary decks, if any
.
.
.
ΔRUN,t,ℓ
EØF

```

(IDENTIFICATION)

MAXIMUM TIME THIS JOB	5	MIN
TOTAL # OF PRINT LINES ON RUN CARDS	3500	LINES
PUNCHED CARD OUTPUT	<input checked="" type="checkbox"/>	
PAPER TAPE OUTPUT: APPROX. LENGTH	_____ FT.	
PLOTTER OUTPUT: CHART PAPER #	00	
NUMBER OF PLOTS	5	APPROX. LENGTH 4 1/2 ft

PAPER TAPE INPUT				
TAPE LABELS				
MAGNETIC TAPES				
LOGICAL UNIT #	INPUT	OUTPUT	SAVE	TAPE SERIAL NO.
24	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	7SAVE tape
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	_____

SEE REVERSE SIDE FOR ADDITIONAL INSTRUCTIONS
NDW-NRL-10462/6006 (5-66)

IBM B45077

Fig. 9—Sample Job Request form for plot run

Plotting Run

ΔJØB,
 ΔEQUIP, 24=(label), RØ, HI, DA
 .
 .
 .
 binary deck { PLØTTEMP
 PLØTPREP
 PLØTT }
 .
 .
 ΔRUN, t, 1
 .
 .
 .
 plotting data
 .
 .
 .
 EØF

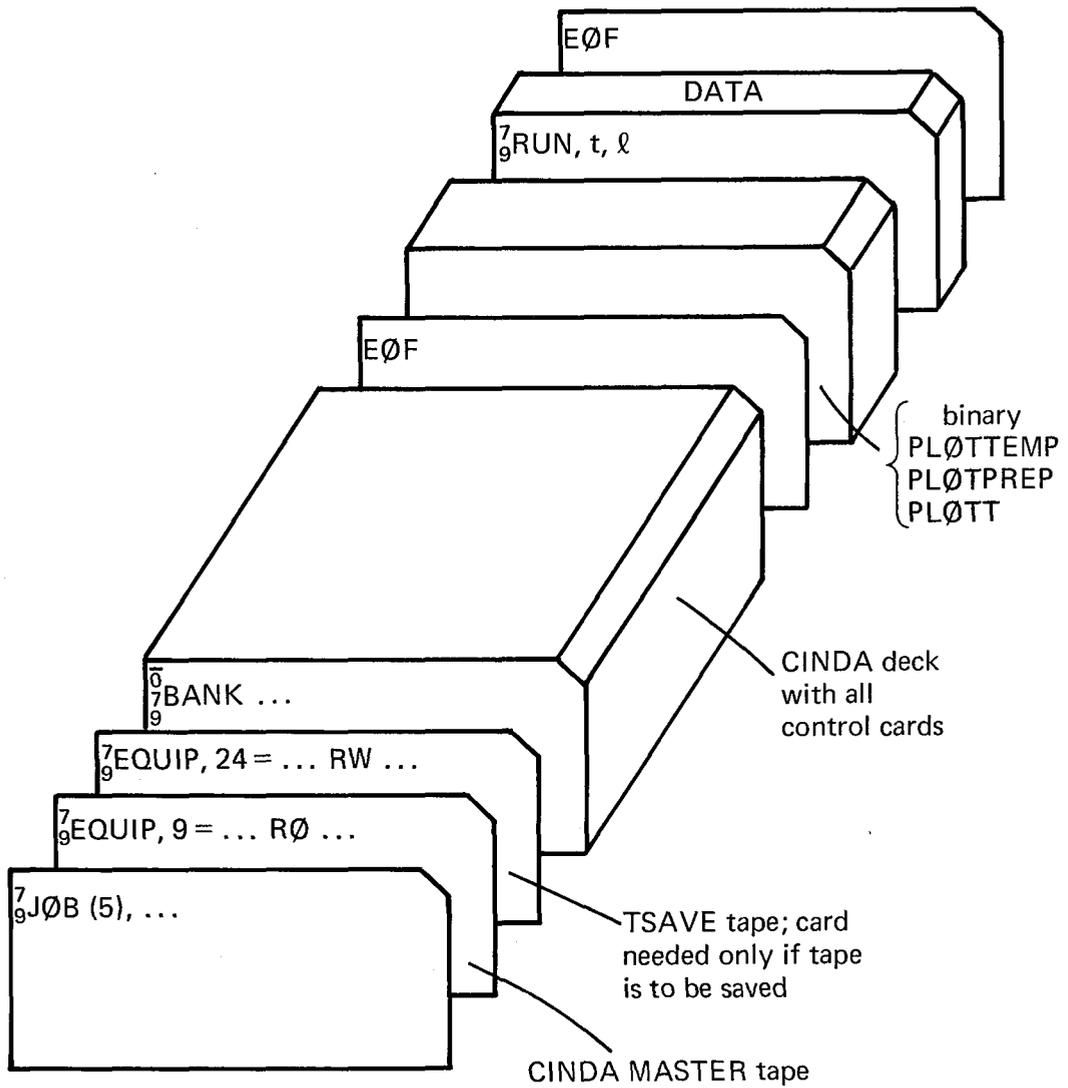


Fig. 10—Combined TSAVE and plot run

VIII. ALPHABETIC LISTING OF AVAILABLE SUBROUTINES

Name	Page	Name	Page	Name	Page
AABB		CMPYI		D11MDA	
ABLATS		CNBACK		D11MDI	
ACSARY		CNEXPN		D12CYL	
ADARIN		CNFAST		D12MCY	
ADD		CNFRWD		D12MDA	
ADDALP		CNFWBK		D2DEG1	
ADDARY		CØLMAX		D2DEG2	
ADDFIX		CØLMIN		D2D1WM	
ADDINV		CØLMLT		D2D2WM	
ALPHAA		CØPY		D2MXD1	
ARCCØS		CØSARY		D2MXD2	
ARCSIN		CSGDMP		D2MX1M	
ARCTAN		CSQRI		D2MX2M	
ARINDV		CVQ1HT		D3DEG1	
ARYADD		CVQ1WM		D3D1WM	
ARYDIV		DA11CY		EFACS	
ARYEXP		DA11MC		EFASN	
ARYINV		DA12CY		EFATN	
ARYMNS		DA12MC		EFCØS	
ARYMPY		DIAG		EFEXP	
ARYPLS		DISAS		EFFEMS	
ARYSTØ		DIVARY		EFFG	
ARYSUB		DIVFIX		EFLØG	
ASNARY		DIVIDE		EFPØW	
ASSMBL		D1DEG1		EFSIN	
ATNARY		D1DEG2		EFSQR	
BIT		D1DG1I		EFTAN	
BIVLV		D1D1DA		ELEADD	
BKARAD		D1D1IM		ELEDIV	
BLDARY		D1D1MI		ELEINV	
BRKARY		D1D1WM		ELEMUL	
BTAB		D1D2DA		ELESUB	
BVSPDA		D1D2WM		ENDMØP	
BVSPSA		D1MDG1		EØF	
CALL		D1MDG2		EXPARY	
CDIVI		D1M1DA		EXPNTL	
CINCØS		D1M1MD		FILE	
CINDSL		D1M1WM		FIX	
CINDSM		D1M2DA		FLIP	
CINDSS		D1M2MD		FLØAT	
CINSIN		D1M2WM		GENALP	
CINTAN		D11CYI		GENARY	
CMPXDV		D11DAI		GENCØL	
CMPXMP		D11DIM		GENM	
CMPXSR		D11MCY		GENST	

NRL REPORT 7656

Name	Page	Name	Page	Name	Page
GSLØPE		PRINT		SMPINT	
HEDCØL		PRINTA		SPLIT	
INPUTG		PRINTL		SPREAD	
INPUTT		PRNTMA		SPRESS	
INTRFC		PRNTMP		SQRØØT	
INVRSE		PSINTR		SQRØTI	
IRRADE		PSNTWM		STATE	
IRRADI		PSØFTS		STFSEP	
ITRATE		PUNCH		STFSEQ	
JACØBI		PUNCHA		STFSQS	
JØIN		PYMLT1		STIFF	
LAGRAN		QFØRCE		STNDRD	
LGRNDA		QINTEG		STØARY	
LINE		QINTGI		STØREP	
LIST		QMAP		SUB	
LØGE		QMETER		SUBARY	
LØGEAR		QMTRI		SUBFIX	
LØGT		RDTNQS		SUMARY	
LØGTAR		READ		SYMINV	
LQDVAP		REFLCT		SYMLST	
LSTAPE		REWIND		TANARY	
LSTPCS		RØWMLT		TØPLIN	
LSTSQU		RTPØLY		TPRINT	
MASS		SCALAR		TRANS	
MATRIX		SCALE		TRNPRT	
MAXDAR		SCLDEP		TRPZDA	
MLTPLY		SCLIND		TRPZD	
MØDES		SCRPFA		TSAVE	
MPYARY		SETMNS		TSØFP	
MPYFIX		SETPLS		UNPAK	
MULT		SETUP		UPDMØP	
MXDRAL		SHFTV		UNITY	
NEWRT4		SHFTVR		VARCCM	
NEWTRT		SHIFT		VARCSM	
ØNES		SHUFL		VARC1	
PLYARY		SIGMA		VARC2	
PLYEVL		SIMEQN		VARGCM	
PLYNML		SINARY		VARGSM	
PNTABL		SKPLIN		VARG1	
PØLMLT		SLDARD		VARG2	
PØLSØV		SLDARY		WRITE	
PØLVAL		SLRADE		WRTARY	
PØLYADD		SLRADI		WRTLØS	
PRESS		SMØPAS		ZERØ	

Execution Subroutines

Name	Page
CINDSS	(Steady state, block iteration)
CINDSL	(Steady state, accelerated)
CINDSM	(Steady state, radiation dominated)
CNFRWD	(Explicit forward differencing)
CNFAST	(Accelerated forward differencing)
CNEXPN	(Explicit exponential prediction)
CNFWBK	(Implicit forward-backward differencing)
CNBACK	(Implicit backward differencing)

Execution Subroutine CINDSS

Purpose—This subroutine ignores the capacitance values of diffusion nodes to calculate the network steady state solution. Due to the SPCS requirement, diffusion nodes are solved by a “block” iterative method. However, if all diffusion nodes were specified as arithmetic nodes they would be calculated by a successive point iterative method. The user is required to specify the maximum number of iterations to be performed in attempting to reach the steady state solution (control constant NLOOP) and the relaxation criterion which determines when it has been reached (DRLXCA for diffusion nodes and/or ARLXCA for arithmetic nodes). The subroutine will continue to iterate until one of the above criteria is met. If the iteration count exceeds NLOOP, an appropriate message is printed. Variables 1 and Output Calls are performed at the start and Variables 2 and Output Calls are performed upon completion. If not specified, control constants DAMPD and DAMPA are set at 1.0. They are used as multipliers times the new temperatures, whereas 1.0 minus their value is used as multipliers times the old temperatures in order to “weight” the returned answer. This weighting of so much new and so much old is useful for damping oscillations due to nonlinearities. They may also be used to achieve overrelaxation.

If a series of steady state solutions at various times is desired it can be accomplished by specifying control constants TIMEND and OUTPUT. OUTPUT will be used both as the output interval and the computation interval. In this case appropriate calls would have to be made in Variables 1 to modify boundary conditions with time.

If desired, the CINDSS call can be followed by a call to one of the transient solution subroutines which has the same SPCS requirement. In this manner the steady state solution becomes the initial conditions for the transient analysis. However, since CINDSS utilizes control constants TIMEND and OUTPUT the user must specify their values in the execution block after the steady state call and prior to the transient analysis call.

Restrictions—The SPCS option is required. Diffusion nodes receive a “block” iteration, while arithmetic nodes receive a successive point iteration; no acceleration features are utilized. Control constants NLOOP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants

TIMEND and OUTPUT. Other control constants which are activated or used are LOOPCT, DRLXCC and/or ARLXCC, TIMEN, TIMEM, TIMEO, DAMPD, DAMPA, DTIMEU, LINECT, and PAGECT. Control constant OPEITR is checked for output each iteration.

Calling Sequence—CINDSS—This subroutine utilizes one dynamic storage core location for each diffusion node.

Execution Subroutine CINDSL

Purpose—This subroutine ignores the capacitance values of diffusion nodes to calculate the network steady state solution. Since this subroutine has the LPCS requirement, both diffusion and arithmetic nodes receive a successive point iteration. In addition, each third iteration a linear extrapolation is performed on the error function plot of each node in an attempt to accelerate convergence. The user is required to specify the maximum number of iterations to be performed in attempting to reach the steady state solution (control constant NLOOP) and the relaxation criterion, which determines when it has been reached (DRLXCA for diffusion nodes and/or ARLXCA for arithmetic nodes). The subroutine will continue to iterate until one of the above criteria is met. If the iteration count exceeds NLOOP an appropriate message is printed. Variables 1 and Output Calls are performed at the start, and Variables 2 and Output Calls are performed upon completion. If not specified, control constants DAMPD and DAMPA are set at 1.0. They are used as multipliers times the new temperatures while 1.0 minus their value is used as multipliers times the old temperatures in order to weight the returned answer. This weighting of so much new and so much old is useful for damping oscillations due to nonlinearities. They may also be used to achieve overrelaxation.

If a series of steady state solutions at various times is desired it can be accomplished by specifying control constants TIMEND and OUTPUT. OUTPUT will be used both as the output interval and the computation interval. In this case appropriate calls would have to be made in Variables 1 to modify boundary conditions with time.

If desired, the CINDSL call can be followed by a call to one of the transient solution subroutines which has the same LPCS requirement. In this manner the steady state solution becomes the initial conditions for the transient analysis. However, since CINDSL utilizes control constants TIMEND and OUTPUT the user must specify their values in the execution block after the steady state call and prior to the transient analysis call.

Restrictions—The LPCS option is required. Diffusion and arithmetic nodes receive a successive point iteration and an extrapolation method of acceleration. Control constants NLOOP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants TIMEND and OUTPUT. Other control constants which are activated or used are: LOOPCT, DRLXCC, and/or ARLXCC, TIMEN, TIMEM, TIMEO, DAMPD, DAMPA, DTIMEU, LINECT, and PAGECT. Control constant OPEITR is checked for output each iteration.

Calling Sequence—CINDSL—This subroutine utilizes two dynamic storage core locations for each diffusion and arithmetic node.

Execution Subroutine CINDSM

Purpose—This subroutine is designed to calculate the network steady state solution of moderately radiation-dominated problems. It is similar to CINDSL in that the LPCS option is required and that all nodes receive a successive point iteration and the same extrapolation method of acceleration. Other execution subroutines evaluate the nonlinear radiation conductors each time they are encountered during an iteration. CINDSR differs in that it linearizes the problem by calculating effective radiation conductors and solves the linearized problem. It then reevaluates the effective radiation conductors, solves the linear problem and continuously repeats the process. The user must specify the maximum number of iterations to perform in attempting to reach the steady state solution and the energy balance of the system to be satisfied as a criterion. This system energy balance is the difference between all energy into the system and all energy out and is specified as control constant BALENG. CINDSM internally calculates the iterative relaxation criteria damping factors and loopings to be performed in solving the linearized problem. It continuously increases the severity of the relaxation criteria until the BALENG criteria is met for two successive linearized problems with virtually no temperature change between the two. Systems with small energy transfer rates to the boundaries are difficult to solve. A reasonable rule is to set BALENG at 1% of the rate in or out. Successive steady state analyses may be performed and CINDSM may be followed by a call to a transient analysis routine with the same LPCS option requirement.

Restrictions—The LPCS option is required. Control constants NLOOP, LAXFAC, and BALENG must be specified and be greater than zero. DAMPD may be used. If it is not specified, the routine will set DAMPD to 1.0. Successive steady state solutions can be obtained by specifying control constants TIMEND and OUTPUT. Other control constants which are activated or used are LOOPCT, ENGBAL, and/or ARLXCC, TIMEN, TIMEM, TIMEO, DTIMEU, LINECT, and PAGECT. Control constant OPEITR is checked for output each iteration. Caution: Each radiation conductor must have a unique conductor number.

Calling Sequence—CINDSM—This subroutine utilizes three dynamic storage core locations for each diffusion and arithmetic node and one more for each radiation conductor.

Execution Subroutine CNFRWD

Purpose—This subroutine performs transient thermal analysis by the explicit forward-differencing method. The stability criterion of each diffusion node is calculated and the

minimum value is placed in control constant CSGMIN. The time step used (control constant DTIMEU) is calculated as 95% of CSGMIN divided by CSGFAC. Control constant CSGFAC is set at 1.0 unless specified larger by the user. A "look-ahead" feature is used when calculating DTIMEU. If one time step will pass the output time point, the time step is set to come out exactly on the output time point; if two time steps will pass the output time point, the time step is set so that two time steps will come out exactly on the output time point. DTIMEU is also compared to DTIMEH and DTIMEL. If DTIMEU exceeds DTIMEH it is set equal to it, if DTIMEU is less than DTIMEL the problem is terminated. If no input values are specified, DTIMEL is set at zero and DTIMEH it is set at infinity. The maximum temperature change calculated over an iteration is placed in control constant DTMPC and/or ATMPCC. They are compared to DTMPCA and/or ATMPCA, respectively, and if larger cause DTIMEU to be modified so that they compare as equal to or less than DTMPCA and/or ATMPCA. If DTMPCA and/or ATMPCA are not specified they are set at infinity.

All diffusion nodes are calculated prior to solving the arithmetic nodes. The user may iterate the arithmetic node solution by specifying control constants NLOOP and ARLXCA. If the arithmetic node iteration count exceeds NLOOP, the answers are accepted as is and the subroutine continues without any user notification. In addition the user may specify control constant DAMPA in order to dampen possible oscillations due to nonlinearities. The arithmetic nodes may be used to specify an incompressible pressure or radiosity network. In this manner they would be solved implicitly each time step, but evaluation of temperature varying properties would suffer a lag of one time step.

Restrictions—The SPCS option is required and control constants TIMEND and OUTPUT must be specified. Problem start time if other than zero may be specified as TIMEO. Other control constants used or activated are: TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPC, ATMPCA, ATMPCC, NLOOP, LOOPCT, DAMPA, ARLXCA, ARLXCC, OPEITR, BACKUP, LINECT, and PAGECT.

Calling Sequence—CNFRWD—This subroutine utilizes one dynamic storage core location for each diffusion and arithmetic node.

Execution Subroutine CNFAST

Purpose—This subroutine is a modified version of CNFRWD which allows the user to specify the minimum time step to be taken. The time step calculations proceed exactly as in CNFRWD until the check with DTIMEL is made. If DTIMEU is less than DTIMEL it is set equal to it. As each node is calculated its CSGMIN is obtained and compared to DTIMEU. If equal to or greater, the nodal calculation is identical to CNFRWD. If the CSGMIN for a node is less than DTIMEU the node receives a steady state calculation. If only a small portion of the nodes in a system receive the steady state calculation the answers are generally reasonable. However, as the number of nodes receiving steady state calculations increases, so do the solution inaccuracies.

Restrictions—The SPCS option is required and control constants TIMEND and OUTPUT must be specified. The checks on control constants DTMPCA, ATMPCA and BACKUP are not performed. Other control constants which are used or activated are TIMEN, TIMEM, TIMEO, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPC, DAMPA, ARLXCA, ARLXCC, NLOOP, LOOPCT, LINECT, and PAGECT.

Calling Sequence—CNFAST—This subroutine utilizes one dynamic storage core location for each diffusion node.

Execution Subroutine CNEXPN

Purpose—This subroutine performs transient thermal analysis by the exponential prediction method, and the solution equation is of the following form:

$$T'_i = \left(\frac{\sum_j G_j T_j Q_i}{\sum_j G_j} \right) \left(1 - e^{-\frac{\sum_j G_j \Delta t}{C_i}} \right) + T_i e^{-\frac{\sum_j G_j \Delta t}{C_i}}$$

This equation is unconditionally stable, no matter what size time step is taken, and it reduces to the steady state equation for an infinite time step. However, stability is not to be confused with accuracy. Time steps larger than would be taken with CNFRWD remain stable but tend to lose or gain energy in the system. For this reason this subroutine is not recommended where accuracy is sought. However, it is suitable for parametric analysis where trends are sought and a more accurate method will be utilized for a final analysis.

The inner workings of the subroutine are virtually identical to CNFRWD with the exception of the solution equation and the use of CSGFAC. The time step used (DTIMEU) is calculated as CSGMIN times CSGFAC. The look-ahead feature for calculating the time step is identical, as are the checks with DTIMEH, DTIMEL, and DTMPCA. The diffusion nodes are calculated prior to the arithmetic nodes, and the arithmetic nodes utilize NLOOP, ARLXCA, and DAMPA, exactly the same as CNFRWD.

Restrictions—The SPCS option is required and control constants TIMEND and OUTPUT must be specified. Problem start time if other than zero may be specified as TIMEO. Other control constants used or activated are TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPC, ATMPCA, ATMPCC, ARLXCA, ARLXCC, DAMPA, OPEITR, BACKUP, LINECT, and PAGECT.

Calling Sequence—CNEXPN—This subroutine utilizes one dynamic storage core location for each diffusion and arithmetic node.

Execution Subroutine CNFWBK

Purpose—This subroutine performs transient thermal analysis by implicit forward-backward differencing. The LPCS option is required and allows the simultaneous set of equations to be solved by “successive point” iterations. During the first iteration for a time step, the capacitance values are doubled and divided by the time step and the energy transfer rates based on old temperatures are added to the source locations. Upon completing the time step the capacitance values are returned to their original state. The iteration looping, convergence criteria and other control constant checks are identical to CNBACK. The time step checks and calculations and look ahead feature are identical to that used for CNBACK.

The automatic radiation transfer damping and extrapolation method of acceleration mentioned under the CNBACK subroutine writeup are also employed in this subroutine. Diffusion and/or arithmetic temperature calculations may be damped through use of DAMPD and/or DAMPA respectively. Control constants BACKUP and OPEITR are continuously checked. CNFWBK internally performs forward-backward differencing of boundary conditions. For this reason the user should utilize TIMEN as the appropriate independent variable in Variables 1 operations.

It is interesting to note that CNFWBK generally converges in 25% fewer iterations than CNBACK. The probable reason for this is that the boundary of the mathematical system is better defined. While every future temperature node under CNBACK is connected to its present temperature, under CNFWBK every future temperature node is also receiving an impressed source based on the present temperature.

Restrictions—The LPCS option is required. Control constants TIMEND, OUTPUT, DTIMEI, NLOOP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used or activated are TIMEN, TIMEO, TIMEM, CSGMIN, DTIMEU, DTIMEH, DTMPCA, DTMPC, ATMPCA, ATMPC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LOOPCT, BACKUP, OPEITR, LINECT, and PAGECT.

Calling Sequence—CNFWBK — This subroutine utilizes three dynamic storage core locations for each diffusion node and one for each arithmetic and boundary node.

Execution Subroutine CNBACK

Purpose—This subroutine performs transient thermal analysis by implicit backward differencing. The LPCS option is required and allows the simultaneous set of equations to be solved by “successive point” iteration. Each third iteration, diffusion node temperatures which trace a continuous decreasing slope receive an extrapolation on their error function curve in an attempt to accelerate convergence. For convergence criteria the user is required to specify NLOOP and DRLXCA and/or ARLXCA. If the number of iterations during a time step exceeds NLOOP a message is printed but the problem proceeds.

MARY E. GEALY

Variables 1 is performed only once for each time step. Since this subroutine is implicit the user must specify the time step to be used as DTIMEI in addition to TIMEND and OUTPUT. The look ahead feature for the time step calculation in CNFRWD is used as are the checks for DTIMEH, DTMPCA and ATMPCA but not DTIMEL. Damping of the solutions can be achieved through use of control constants DAMPD and/or DAMPA. Control constants BACKUP and OPEITR are continuously checked.

Implicit methods of solution often oscillate at start up or for boundary step changes when radiation conductors are present. CNBACK contains an automatic damping feature which is applied to radiation conductors. The radiation transfer to a node is calculated for its present temperature and a temporary new temperature is calculated. Then the radiation transfer is recalculated and the final node temperature is calculated based on the arithmetic mean of the two radiation transfer calculations. This automatic radiation damping has proven to be quite successful and lessens the need for use of DAMPD and DAMPA.

Restrictions—The LPCS option is required. Control constants TIMEND, OUTPUT, DTIMEI, NLOOP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used on activated are: TIMEN, TIMEO, TIMEM, CSGMIN, DTIMEV, DTIMEH, DTMPCA, DTMPC, ATMPCA, ATPCC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LOOPCT, BACKUP, OPEITR, LINECT, and PAGECT.

Calling Sequence—CNBACK — This subroutine utilizes three dynamic storage core locations for each diffusion node and one for each arithmetic and boundary node.

Interpolation Subroutines

Name	Page
LAGRAN, LGRNDA, D1DEG1, D1D1DA, D1D1WM, D11MDA	58
D1MDG1, D1M1DA, D1M1WM, D1M1MD, D1DEG2, D1D2DA D1D2WM, D12MDA	59
D1MDG2, D1M2DA, D1M2WM, D1M2MD, D1DG1I, D1D1IM, D1D1MI, D11DAI, D11DIM, D11MDI	60
D11CYL, DA11CY, D12CYL, DA12CY, D11MCY, DA11MC, D12MCY, DA12MC	61
CVQ1HT, CVQ1WM, GSLØPE, PSINTR, PSNTWM	62
Bivariate Array Format, BVSPSA, BVSPDA, BVTRN1, BVTRN2 D2DEG1, D2DEG2, D2D1WM, D2D2WM	63
D2MXD1, D2MXD2, D2MX1M, D2MX2M, Trivariate Array Format	64
D3DEG1, D3D1WM, VARCSM, VARCCM, VARC1, VARC2	65
VARGSM, VARGCM, VARG1, VARG2	66

Subroutine LAGRAN or LGRNDA

Purpose—These subroutines perform Lagrangian interpolation of up to order 50. The first requires one doublet array of X, Y pairs while the second requires two singlet arrays, one of X's and the other of Y's. They contain an extrapolation feature such that if the X value falls outside the range of the independent variable the nearest dependent Y variable value is returned and no error is noted.

$$Y = P_n(X) = \sum_{k=0}^n Y_k \prod_{\substack{i=0 \\ i \neq k}}^n \frac{X - X_i}{X_k - X_i}, \quad r = 1, 2, 3, \dots, 50 \text{ max.}$$

Restrictions—All values must be floating point except N which is the order of interpolation plus one and must be an integer. The independent variable values must be in ascending order.

Calling Sequence—LAGRAN(X,Y,A(IC),N) or LGRNDA(X,Y,AX(IC),AY(IC),N)

NOTE: A doublet array is formed as follows:

IC, X1, Y1, X2, Y2, X3, Y3, ..., XN, YN
where IC = 2*N (set by program).

Singlet arrays are formed as follows:

IC, X1, X2, X3, ..., XN
IC, Y1, Y2, Y3, ..., YN
and IC = N (set by program).

Subroutine D1DEG1 or D1D1DA

Purpose—These subroutines perform single variable linear interpolation on doublet or singlet arrays respectively. They are self-contained subroutines that are called upon by virtually all other linear interpolation subroutines.

Restrictions—All values must be floating point numbers. The X independent variable values must be in ascending order.

Calling Sequence—D1DEG1(X,A(IC),Y) or D1D1DA(X,AX(IC),AY(IC),Y)

Subroutine D1D1WM or D11MDA

Purpose—These subroutines perform single-variable, linear interpolation by calling on D1DEG1 or D1D1DA, respectively. However, the interpolated answer is multiplied by the value addressed as Z prior to being returned as Y.

Restrictions—Same as D1DEG1 or D1D1DA, and Z must be a floating-point number.

Calling Sequence—D1D1WM(X,A(IC),Z,Y) or D11MDA(X,AX(IC),AY(IC),Z,Y)

Subroutine D1MDG1 or D1M1DA

Purpose—These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. They require a doublet or two singlet arrays, respectively.

Restrictions—See D1DEG1 or D1D1DA as they are called on, respectively.

Calling Sequence—D1MDG1(X1,X2,A(IC),Y) or
D1M1DA(X1,X2,AX(IC),AY(IC),Y)

Subroutine D1M1WM or D1M1MD

Purpose—These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

Restrictions—Same as D1MDG1 or D1M1DA, and Z must be a floating-point number.

Calling Sequence—D1M1WM(X1,X2,A(IC),Z,Y) or
D1M1MD(X1,X2,AX(IC),AY(IC),Z,Y)

Subroutine D1DEG2 or D1D2DA

Purpose—These subroutines perform single-variable parabolic interpolation. The first requires a doublet array of X, Y pairs while the second requires singlet arrays of X and Y values. They call on subroutines LAGRAN and LGRNDA, respectively.

Restrictions—See LAGRAN or LGRNDA.

Calling Sequence—D1DEG2(X,A(IC),Y) or D1D2DA(X,AX(IC),AY(IC),Y)

Subroutine D1D2WM or D12MDA

Purpose—These subroutines perform single-variable parabolic interpolation by calling on LAGRAN or LGRNDA, respectively. However, the interpolated answer is multiplied by the value addressed as Z prior to being returned as Y.

Restrictions—Same as LAGRAN or LGRNDA, and Z must be a floating-point number.

Calling Sequence—D1D2WM(X,A(IC),Z,Y) or
D12MDA(X,AX(IC),AY(IC),Z,Y)

Subroutine D1MDG2 or D1M2DA

Purpose—These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. They require a doublet or two singlet arrays, respectively.

Restrictions—See LAGRAN or LGRNDA as they are called.

Calling Sequence—D1MDG2(X1,X2,A(IC),Y) or
D1M2DA(X1,X2,AX(IC),AY(IC),Y)

Subroutine D1M2WM or D1M2MD

Purpose—These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

Restrictions—Same as D1MDG2 or D1M2DA, and Z must be a floating point number.

Calling Sequence—D1M2WM(X1,X2,A(IC),Z,Y) or
D1M2MD(X1,X2,AX(IC),AY(IC),Z,Y)

Subroutine D1DG1I or D1D1IM or D1D1MI

Purpose—These subroutines perform single-variable linear interpolation on an array of X's to obtain an array of Y's. D1D1IM multiplies all interpolated values by a constant Z value while D1D1MI allows a unique Z value for each X value. They all call on D1DEG1.

Restrictions—The number of input X's must be supplied as the integer N and agree with the number of Y and Z locations where applicable. Z values must be floating-point numbers.

Calling Sequence—D1DG1I(N,X(DV),A(IC),Y(DV)) or
D1D1IM(N,X(DV),A(IC),Z,Y(DV)) or
D1D1MI(N,X(DV),A(IC),Z(DV),Y(DV))

Subroutine D11DAI or D11DIM or D11MDI

Purpose—These subroutines are virtually identical to D1DG1I, D1D1IM, and D1D1MI, respectively. The difference is that they require singlet arrays for interpolation and call on D1D1DA.

Restrictions—Same as D1DG1I, D1D1IM, and D1D1MI.

Calling Sequence—D11DAI(N,X(DV),AX(IC),AY(IC),Y(DV)) or
D11DIM(N,X(DV),AX(IC),AY(IC),Z,Y(DV)) or
D11MDI(N,X(DV),AX(IC),AY(IC),Z(DV),Y(DV))

Subroutine D11CYL or DA11CY

Purpose—These subroutines reduce core storage requirements for cyclical interpolation arrays. The arrays need cover one period only, and the period (PR) must be specified as the first argument. Linear interpolation is performed, and the independent variable must be in ascending order.

Restrictions—All values must be floating point. Subroutine INTRFC is called on by both D11CYL and DA11CY, then D1DEG1 or D1D1DA, respectively.

Calling Sequence—D11CYL(PR,X,A(IC),Y) or
DA11CY(PR,X,AX(IC),AY(IC),Y)

Subroutine D12CYL or DA12CY

Purpose—These subroutines are virtually identical to D11CYL and DA11CY, except that parabolic interpolation is performed.

Restrictions—See D11CYL and DA11CY. Subroutines LAGRAN and LGRNDA, respectively, are called on.

Calling Sequence—D12CYL(PR,X,A(IC),Y) or DA12CY(PR,X,AX(IC),AY(IC),Y)

Subroutine D11MCY or DA11MC

Purpose—These subroutines are virtually identical to D11CYL or DA11CY, except that the interpolated answer is multiplied by the floating-point Z value prior to being returned as Y.

Restrictions—Call on subroutines D1DEG1 and D1D1DA, respectively.

Calling Sequence—D11MCY(PR,X,A(IC),Z,Y) or
DA11MC(PR,X,AX(IC),AY(IC),Z,Y)

Subroutine D12MCY or DA12MC

Purpose—These subroutines are virtually identical to D11MCY and DA11MC except that parabolic interpolation is performed.

Restrictions—Calls on subroutines LAGRAN and LGRNDA, respectively.

Calling Sequence—D12MCY(PR,X,A(IC),Z,Y) or
DA12MC(PR,X,AX(IC),AY(IC),Z,Y)

Subroutine CVQ1HT or CVQ1WM

Purpose—These subroutines perform two single-variable linear interpolations. The interpolation arrays must have the same independent variable X and dependent variables of, say, R(X) and S(X). Additional arguments of Y, Z, and T complete the data values. The postinterpolation calculations are, respectively:

$$Y = S(X)*(R(X)-T) \quad \text{or}$$

$$Y = Z*S(X)(R(X)-T).$$

Restrictions—Interpolation arrays must be of the doublet type and have a common independent variable. All values must be floating-point numbers.

Calling Sequence—CVQ1HT(X,AR(IC),AS(IC),T,Y) or
CVQ1WM(X,AR(IC),AS(IC),T,Z,Y)

Subroutine GSLOPE

Purpose—This subroutine will generate a slope array so that point slope interpolation subroutines can be used instead of standard linear interpolation subroutines. The user must address two singlet arrays, and a singlet slope array will be produced.

Restrictions—The X independent-variable array must be in ascending order. All arrays must be of equal length and contain floating-point numbers.

Calling Sequence—GSLOPE(AX(IC),AY(IC),AS(IC))

Subroutine PSINTR or PSNTWM

Purpose—These subroutines perform linear interpolation and require arrays of the Y points and slopes which correspond to the independent variable X array. All values must be floating-point numbers. PSNTWM multiplies the interpolated answer by Z prior to returning it as Y.

Restrictions—The independent X and dependent Y and slope arrays must be of equal length.

Calling Sequence—PSINTR(X,AX(IC),AY(IC),AS(IC),Y) or
PSNTWM(X,AX(IC),AY(IC),AS(IC),Z,Y)

Bivariate Array Format, $Z = f(X,Y)$

Bivariate arrays must be rectangular and full and must be entered in the following row order:

```

IC,N ,X 1,X 2,X 3, . . . , X N
  Y1,Z11,Z12,Z13, . . . , Z1N
  Y2,Z21,Z22,Z23, . . . , Z2N
  :
  :
  :
  YM,ZM1,ZM2,ZM3, . . . , ZMN

```

where N is the integer number of X variables. All other values must be floating-point numbers, and the X and Y values must be in ascending order.

Subroutine BVSPSA or BVSPDA

Purpose—These subroutines use an input Y argument to address a bivariate array and pull off a singlet array of Z's corresponding to the X's or pull off a doublet array of X, Z values, respectively. The integer count for the constructed arrays must be exactly N or 2*N, respectively. To use the singlet array for an interpolation call, reach the X array by addressing the N in the bivariate array.

Restrictions—As stated above, and all values must be floating point.

Calling Sequence—BVSPSA(Y,BA(IC),AZ(IC)) or BVSPDA(Y,BA(IC),AXZ(IC))

Subroutine D2DEG1 or D2DEG2

Purpose—These subroutines perform bivariate linear and parabolic interpolation, respectively. The arrays must be formatted as shown for Bivariate Array Format.

Restrictions—For D2DEG1, $N \geq 2$, $M \geq 2$ } See bivariate
 For D2DEG2, $N \geq 3$, $M \geq 3$ } array format

Calling Sequence—D2DEG1(X,Y,BA(IC),Z) or D2DEG2(X,Y,BA(IC),Z)

Subroutine D2D1WM or D2D2WM

Purpose—These subroutines perform bivariate linear or parabolic interpolation by calling on D2DEG1 or D2DEG2, respectively. The interpolated answer is multiplied by the W value prior to being returned as Z.

Restrictions—Same as D2DEG1 or D2DEG2, and W must be a floating-point value.

Calling Sequence—D2D1WM(X,Y,BA(IC),W,Z) or D2D2WM(X,Y,BA(IC),W,Z)

Subroutine D2MXD1 or D2MXD2

Purpose—These subroutines are virtually identical to D2DEG1 and D2DEG2 except that the arithmetic mean of two X values is used as the X-independent variable for interpolation.

Restrictions—Same as D2DEG1 or D2DEG2.

Calling Sequence—D2MXD1(X1,X2,Y,BA(IC),Z) or D2MXD2(X1,X2,Y,BA(IC),Z)

Subroutine D2MX1M or D2MX2M

Purpose—These subroutines are virtually identical to D2D1WM and D2D2WM except that the arithmetic mean of two X values is used as the X-independent variable for interpolation.

Restrictions—Same as D2D1WM and D2D2WM.

Calling Sequence—D2MX1M(X1,X2,Y,BA(IC),W,Z) or
D2MX2M(X1,X2,Y,BA(IC),W,Z)

Trivariate Array Format, $T = f(X, Y, Z)$

Trivariate arrays may be thought of as two or more bivariate arrays, each bivariate array a function of a third independent variable Z. Trivariate arrays must be entered in row order and be constructed as follows:

```

IC,NX1,NY1, Z1, X 1, X 2, X 3, . . . , X N
      Y1, T11, T12, T13, . . . , T1N
      Y2, T21, T22, T23, . . . , T2N
      . . . . .
      YM, TM1, TM2, TM3, . . . , TMN
NX2,NY2, Z2, X 1, X 2, X 3, . . . , X J
      Y1, T11, T12, T13, . . . , T1J
      Y2, T21, T22, T23, . . . , T2J
      . . . . .
      YK, TK1, TK2, TK3, . . . , TKJ
NX3,NY3, Z3, . . . . .
      . . . . .
      . . . . .
    
```

The trivariate array may consist of as many bivariate “sheets” as desired. The number of X and Y values in each sheet must be specified as integers (NX-NY). The “sheets” must be rectangular and full but need not be identical in size.

Subroutine D3DEG1 or D3D1WM

Purpose—These subroutines perform trivariate linear interpolation. The interpolation array must be constructed as shown for the Trivariate Array Format. Subroutine D2DEG1 is called on, which calls on D1DEG1. Hence, the linear extrapolation feature of these routines applies. Subroutine D3D1WM multiplies the interpolated answer by F prior to returning it as T.

Restrictions—See Trivariate Array Format. F must be a floating-point value.

Calling Sequence—D3DEG1(X,Y,Z,TA(IC),T) or D3D1WM(X,Y,Z,TA(IC),F,T)

Subroutine VARCSM or VARCCM or VARC1 or VARC2

Purpose—These are linear interpolation subroutines which are set up as Variables 1 calls by the preprocessor when processing the CGS and CGD mnemonic codes in the nodal data block. VARCSM is utilized for the CGS code. VARCCM is utilized for the CGD code when two array arguments appear. VARC1 and VARC2 are used for the CGD code when either the first or second respective array arguments are entered as a constant. The following mnemonic codes in the nodal block

```

8
↓
CGS 1,80.,A1,10.2
CGD 2,80.,A1,10.2,A2,1.6
CGD 3,80.,1.4,5.1,A2,1.6
CGD 4,80.,A1,5.1,6.3,8.7

```

would cause the construction in Variables 1 of

```

12
↓
VARCSM(T1,C1,A1,10.2)
VARCCM(T2,C2,A1,10.2,A2,1.6)
VARC1(T3,C3,1.4,5.1,A2,1.6)
VARC2(T4,C4,A1,5.1,6.3,8.7)

```

The second call causes the sum of two interpolations with multiplications to be used as the C2 value. The latter two calls only perform one interpolation but use the sum of the two products as the C value.

Restrictions—The array arguments must address the integer count.

Calling Sequence—VARCSM(T,C,A(IC),F) or VARCCM(T,C,A1(IC),F1,A2(IC),F2) or VARC1(T,C,X,F1,A2(IC),F2) or VARC2(T,C,A1(IC),F1,X,F2)

Subroutine VARGSM or VARGCM or VARG1 or VARG2

Purpose—These are linear interpolation subroutines set up as Variables 1 calls by the preprocessor when processing the CGS and CGD mnemonic codes in the conductor data block. They are similar to the preceding four calls for the nodal data block except that the conductor argument is first followed by two temperature arguments. VARGSM is used for the CGS code. If the F value is positive, the mean of the two addressed temperatures is used for interpolation. If it is negative, only T1 is used for interpolation and the absolute value of F is used as a multiplier. VARGCM, VARG1, and VARG2 perform the one or two interpolations required, multiply by the F values to obtain G1 and G2 components, and then calculate G as

$$G = 1.0/(1.0/G1 + 1.0/G2).$$

Restrictions—The array arguments must address the integer count.

Calling Sequence—VARGSM(G,T1,T2,A(IC),F) or
 VARGCM(G,T1,T2,A1(IC),F1,A2(IC),F2) or
 VARG1(G,T1,T2,X,F1,A2(IC),F2) or
 VARG2(G,T1,T2,A1(IC),F1,X,F2)

Arithmetic Subroutines

Name	Page
FLØAT, FIX, INTRFC, SHFTV, SHFTVR, FLIP, SETPLS, ARYPLS	68
SETMNS, ARYMNS, ADD, ADDFIX, ADDARY, ARYADD	69
SUB, SUBFIX, SUBARY, ARYSUB, MLTPLY, MPYFIX, MPYARY, ARYMPY	70
DIVIDE, DIVFIX, DIVARY, ARYDIV, GENARY	71
BLDARY, BRKARY, BKARAD, STFSEP, SCALE,	72
STFSEQ, STFSQS, SUMARY, MAXDAR, MXDRAL	73
ARYINV, ARINDV, ADDINV, ADARIN, STØARY, ARYSTØ	74
SCLDEP, SCLIND, SLDARY, SLDARD, SPLIT, JØIN	75
SPREAD, QMETER, RDTNQS, QMTRI, QFØRCE, QINTEG, QINTGI	76
CINSIN, SINARY, CINCØS, CØSARY, CINTAN, TANARY, ARCSIN, ASNARY	77
ARCØS, ACSARY, ARCTAN, ATNARY, EXPNTL, ARYEXP, EXPARY	78
LØGT, LØGTAR, LØGE, LØGEAR, SQRØØT, SQRØTI, CMPXSR, CSQRI	79
CMPXMP, CMPYI, CMPXDV, CDIVI, NEWTRT, NEWRT4,	80
PLYNML, PLYARY, SMPINT, TRPZD, TRPZDA	81
PRESS, SPRESS, EFFG, EFFEMS	82

Subroutine FLOAT or FIX or INTRFC

Purpose—Subroutine FLOAT will convert an integer to a floating-point number. Subroutine FIX will convert a floating-point number to an integer. Subroutine INTRFC will fracture a floating-point number to yield the largest integer value possible and the remainder or fractional portion is a floating-point number. Their respective operations are

$$\begin{aligned} X &= N \\ \text{or } N &= X \\ \text{or } N &= X \\ Y &= N \\ F &= X - Y \end{aligned}$$

Restrictions—X and F arguments must address floating-point values and the N argument must address an integer.

Calling Sequence—FLOAT(N,X) or FIX(X,N) or INTRFC(X,N,F)

Subroutine SHFTV or SHFTVR or FLIP

Purpose—Subroutine SHFTV will shift a sequence of data from one array to another. Subroutine SHFTVR will shift a sequence of data from one array and place it in another array in reverse order. Subroutine FLIP will reverse an array in its own array location. Their respective operations are

$$\begin{aligned} A(i) &= B(i), & i &= 1, N \\ \text{or } A(N-i+1) &= B(i), & i &= 1, N \\ \text{or } A(i)_{\text{new}} &= A(N-i+2)_{\text{old}}, & i &= 2, N+1. \end{aligned}$$

The answer array may not be overlaid into the input array.

Restrictions—The data values to be shifted or reversed in order may be anything. The N must be an integer.

Calling Sequence—SHFTV(N,B(DV),A(DV)) or SHFTVR(N,B(DV),A(DV)) or FLIP(A(IC))

Subroutine SETPLS or ARYPLS

Purpose—SETPLS will set the sign positive for a variable number of arguments while ARYPLS will set the sign positive for every data value in a specified length array.

Restrictions—The values addressed may be either integers or floating-point numbers. The number (N) of data values in the array must be specified as an integer.

Calling Sequence—SETPLS(A,B,C...) or ARYPLS(N,A(DV))

where N may be a literal integer or the address of a location containing an integer, and A(DV) addresses the first data value in the array.

Subroutine SETMNS or ARYMNS

Purpose—SETMNS will set the sign negative for a variable number of arguments, while ARYMNS will set the sign negative for every data value in a specified length array.

Restrictions—The values addressed may be either integers or floating-point numbers. The number (N) of data values in the array must be specified as an integer.

Calling Sequence—SETMNS(A,B,C,...) or ARYMNS(N,A(DV))

where N may be a literal integer or the address of a location containing an integer and A(DV) addresses the first data value in the array.

Subroutine ADD or ADDFIX

Purpose—To sum a variable number of floating-point or integer numbers, respectively.

$$S = \sum X_i, \quad i = 1, 2, 3, \dots, N, \quad N \geq 2$$

Restrictions—Subroutine ADD is for floating-point numbers, while subroutine ADDFIX is for integers.

Calling Sequence—ADD(X1,X2,X3,...,XN,S) or ADDFIX(X1,X2,X3,...,XN,S)

Subroutine ADDARY or ARYADD

Purpose—Subroutine ADDARY will add the corresponding elements of two specified length arrays to form a third array. Subroutine ARYADD will add a constant value to every element in an array to form a new array. Their respective operations are

$$\begin{aligned} A_i &= B_i + C_i, \quad i = 1, N \\ \text{or } A_i &= B_i + C, \quad i = 1, N. \end{aligned}$$

The answer array may be overlaid into one of the input array areas.

Restrictions—All data values to be operated on must be floating-point numbers. The array length N must be an integer.

Calling Sequence—ADDARY(N,B(DV),C(DV),A(DV)) or
ARYADD(N,B(DV),C,A(DV))

Subroutine SUB or SUBFIX

Purpose—To subtract a variable number of floating-point or integer numbers, respectively,

$$R = Y - \sum X_i, \quad i = 1, 2, 3, \dots, N, \quad N \geq 1$$

Restrictions—Subroutine SUB is for floating-point numbers while subroutine SUBFIX is for integers.

Calling Sequence—SUB(Y,X1,X2,X3,...,XN,R) or
SUBFIX(Y,X1,X2,X3,...,XN,R)

Subroutine SUBARY or ARYSUB

Purpose—Subroutine SUBARY will subtract the corresponding elements of one array from another to form a third array. Subroutine ARYSUB will subtract a constant value from every element in an array to form a new array. Their respective operations are

$$\begin{aligned} A_i &= B_i - C_i, \quad i = 1, N \\ \text{or } A_i &= B_i - C, \quad i = 1, N \end{aligned}$$

The answer array may be overlaid into one of the input array areas.

Restrictions—All data values to be operated on must be floating-point numbers. The array length N must be an integer.

Calling Sequence—SUBARY(N,B(DV),C(DV),A(DV)) or
ARYSUB(N,B(DV),C,A(DV))

Subroutine MLTPLY or MPYFIX

Purpose—To multiply a variable number of floating-point or integer numbers, respectively.

$$P = X1 * X2 * X3 * \dots * XN, \quad N \geq 2$$

Restrictions—Subroutine MLTPLY is for floating-point numbers, while subroutine MPYFIX is for integers.

Calling Sequence—MLTPLY(X1,X2,X3,...,XN,P) or MPYFIX(X1,X2,X3,...,XN,P)

Subroutine MPYARY or ARYMPY

Purpose—Subroutine MPYARY will multiply the corresponding elements of two arrays to form a third. Subroutine ARYMPY will multiply a constant value times each element of an array to form a new array. Their respective operations are

$$A_i = B_i * C_i, \quad i = 1, N$$

or $A_i = B_i * C, \quad i = 1, N$

The answer array may be overlaid into one of the input array areas.

Restrictions—All data values to be operated on must be floating-point numbers. The array length N must be an integer.

Calling Sequence—MPYARY(N,B(DV),C(DV),A(DV)) or
ARYMPY(N,B(DV),C,A(DV))

Subroutine DIVIDE or DIVFIX

Purpose—These subroutines are used to perform a division of floating-point or integer numbers, respectively;

$$Q = Y/\Sigma X_i, \quad i = 1, 2, 3, \dots, N, \quad N \geq 1.$$

Restrictions—Subroutine DIVIDE is for floating-point numbers, while DIVFIX is for integers.

Calling Sequence—DIVIDE(Y,X1,X2,X3, ..., XN,Q) or
DIVFIX(Y,X1,X2,X3, ..., XN,Q)

Subroutine DIVARY or ARYDIV

Purpose—Subroutine DIVARY will divide the elements of one array into the corresponding elements of another array to produce a third array. Subroutine ARYDIV will divide each element of an array by a constant value to produce a new array. Their respective operations are

$$A_i = B_i/C_i, \quad i = 1, N$$

or $A_i = B_i/C, \quad i = 1, N.$

The answer array may be overlaid into one of the input array areas.

Restrictions—All data values to be operated on must be floating-point numbers. The array length N must be an integer.

Calling Sequence—DIVARY(N,B(DV),C(DV),A(DV)) or
ARYDIV(N,B(DV),C,A(DV))

Subroutine GENARY

Purpose—This subroutine will generate an array of equally incremented ascending values. The user must supply the minimum value, maximum value, number of values in the array to be generated, and the space for the generated array.

Restrictions—All numbers must be floating point.

Calling Sequence—GENARY(B(DV),A(DV))

where

- B(1) = minimum value
- B(2) = maximum value
- B(3) = length of array to be generated (floating point).

Subroutine BLDARY

Purpose—This subroutine will build an array from a variable number of arguments in the order listed. The operation performed is

$$A_i = X_i, \quad i = 1, N.$$

Restrictions—Data may be of any form. The subroutine obtains the integer array length N by counting the arguments.

Calling Sequence—BLDARY(A(DV),X1,X2,X3,...,XN)

Subroutine BRKARY or BKARAD

Purpose—These subroutines will distribute values from within an array to a variable number of arguments in the order listed. The first places the value into the location while the second adds it to what is in the location. Respective operations are

$$\begin{aligned} X_i &= A_i, & i &= 1, N \\ \text{or } X_i &= X_i + A_i, & i &= 1, N. \end{aligned}$$

Restrictions—Floating-point numbers must be used for BKARAD. The integer array length N is obtained by the routines by counting the number of arguments.

Calling Sequence—BRKARY(A(DV),X1,X2,X3,...,XN) or
BKARAD(A(DV),X1,X2,X3,...,XN)

Subroutine STFSEP or SCALE

Purpose—Subroutine STFSEP will place a constant value into a variable number of locations. Subroutine SCALE will utilize a constant value to multiply a variable number of arguments, each having a location for the product. The respective operations are

$$\begin{aligned} X_i &= Y, & i &= 1, 2, 3, \dots, N \\ \text{or } X_i &= Y * Z_i, & i &= 1, 2, 3, \dots, N. \end{aligned}$$

Restrictions—STFSEP may be used to move any desired value, but SCALE can only be used for floating-point numbers.

Calling Sequence—STFSEP(Y,X1,X2,X3,...,XN) or
SCALE(Y,X1,Z1,X2,Z2,...,XN,ZN)

Subroutine STFSEQ or STFSQS

Purpose—Both subroutines will stuff a constant data value into a specified length array or group of sequential locations. STFSEQ expects the constant data value to be in the first array location, while STFSQS requires it to be supplied as an additional argument. The respective operations performed are

$$A_i = A_1, \quad i = 2, N$$

$$\text{or } A_i = B, \quad i = 1, N.$$

Restrictions—N must be an integer, but the constant data value may be integer, either floating point or alphanumeric.

Calling Sequence—STFSEQ(A(DV),N) or STFSQS(B,N,A(DV))

Subroutine SUMMARY

Purpose—SUMMARY is used to sum an array of floating-point values:

$$S = \sum A_i, \quad i = 1, N.$$

Restrictions—The values to be summed must be floating-point numbers and the array length N must be an integer.

Calling Sequence—SUMMARY(N,A(DV),S)

Subroutine MAXDAR or MXDRAL

Purpose—These subroutines will obtain the absolute maximum difference between corresponding elements of two arrays of equal length N. The array values must be floating-point numbers. The operation performed is

$$D = |A_i - B_i|_{\max}; \quad i = 1, N.$$

Subroutine MXDRAL also locates the position P between 1 and N where the maximum occurs.

Restrictions—The N argument must be an integer. The D and P arguments are returned as floating-point numbers.

Calling Sequence—MAXDAR(N,A(DV),B(DV),D) or MXDRAL(N,A(DV),B(DV),D,P)

Subroutine ARYINV or ARINDV

Purpose—Subroutine ARYINV will invert each element of an array in its own location. Subroutine ARINDV will divide each element of an array into a constant value to form a new array. Their respective operations are

$$\begin{aligned} A_i &= 1.0/A_i, & i &= 1, N \\ \text{or } A_i &= B/C_i, & i &= 1, N. \end{aligned}$$

Restrictions—All data values must be floating-point numbers. The array length N must be an integer.

Calling Sequence—ARYINV($N, A(DV)$) or ARINDV($N, C(DV), B, A(DV)$)

The ARINDV answer array may be overlaid into the input array area.

Subroutine ADDINV or ADARIN

Purpose—Subroutine ADDINV will calculate one over the sum of the inverses of a variable number of arguments. Subroutine ADARIN will calculate one over the sum of inverses of an array of values. These subroutines are useful for calculating the effective conductance of series conductors. Their respective operations are

$$\begin{aligned} Y &= 1.0/(1./X_1 + 1./X_2 + \dots + 1./X_N), & N &\geq 2 \\ \text{or } Y &= 1.0/\Sigma(1./X_i), & i &= 1, N. \end{aligned}$$

Restrictions—All data values must be floating-point numbers. The array length N must be an integer.

Calling Sequence—ADDINV($X_1, X_2, X_3, \dots, X_N, Y$) or ADARIN($N, X(DV), Y$)

Subroutine STOARY or ARYSTO

Purpose—These subroutines will place a value into or take a value out of a specific array location, respectively. Their respective operations are

$$\begin{aligned} A_i &= X, & i &= N, & N &> 0 \\ \text{or } X &= A_i, & i &= N, & N &> 0. \end{aligned}$$

Restrictions—The values may be anything, but N must be an integer.

Calling Sequence—STOARY($N, X, A(DV)$) or ARYSTO($N, X, A(DV)$)

Subroutine SCLDEP or SCLIND

Purpose—These subroutines will multiply the dependent or independent variables of a doublet interpolation array, respectively. Their respective operations are

$$\begin{aligned} A_i &= X * A_i, & i &= 3, 5, 7, \dots, N+1 \\ \text{or } A_i &= X * A_i, & i &= 2, 4, 6, \dots, N. \end{aligned}$$

Restrictions—All values must be floating point. The arrays must contain the length integer count as the first value, which must be even.

Calling Sequence—SCLDEP(A(IC),X) or SCLIND(A(IC),X)

Subroutine SLDARY or SLDARD

Purpose—These subroutines are useful for updating fixed-length interpolation arrays during a transient analysis. The array data values are moved back one or two positions, the first one or two values are discarded, and the last one or two values updated, respectively. The “sliding array” thus maintained can then be used with standard interpolation subroutines to simulate transport delay phenomena. Their respective operations are

$$\begin{aligned} A_i &= A_{i+1}, & i &= 2, N \\ \text{and } A_i &= X, & i &= N + 1 \\ \text{or } A_i &= A_{i+2}, & i &= 2, N-1 \\ \text{and } A_i &= X \text{ and } A_{i+1} = Y, & i &= N. \end{aligned}$$

Restrictions—The addressed arrays must have the array integer count N as the first value. For SLDARD, N must be even.

Calling Sequence—SLDARY(X,A(IC)) or SLDARD(X,Y,A(IC))

Subroutine SPLIT or JOIN

Purpose—These subroutines separate a doublet array into two singlet arrays or combine two singlet arrays into a doublet array respectively. Their respective operations are

$$\begin{aligned} B_i &= A_{2i-1}, & i &= 1, N \\ C_i &= A_{2i}, & i &= 1, N \\ \text{or } A_{2i-1} &= B_i, & i &= 1, N \\ A_{2i} &= C_i, & i &= 1, N. \end{aligned}$$

Restrictions—The arrays may contain any values, but N must be an integer. N is the length of the B and C arrays, and the A array must be of length 2N.

Calling Sequence—SPLIT(N,A(DV),B(DV),C(DV)) or
JOIN(N,B(DV),C(DV),A(DV))

Subroutine SPREAD

Purpose—This subroutine applies interpolation subroutine D1D1DA to two singlet arrays to obtain an array of dependent variables vs an array of independent variables. It is extremely useful for obtaining singlet arrays of various dependent variables with a corresponding relationship to one singlet independent variable array. The dependent variable arrays thus constructed can then be operated on by array manipulation subroutines in order to form composite or complex functions. Doublet arrays can first be separated with subroutine SPLIT and later reformed with subroutine JOIN.

Restrictions—All data values must be floating point except N, which must be the integer length of the array to be constructed. The arrays fed into D1D1DA for interpolation must start with the integer count. X is for independent and Y is for dependent. I is for input and O for output.

Calling Sequence—SPREAD(N,X(IC),Y(IC),XI(DV),YO(DV))

Subroutine QMETER or RDTNQS or QMTRI or QFORCE

Purpose—These subroutines are generally used for calculating flow rates. Their respective operations are

$$\begin{aligned} A &= B * (C-D) \\ \text{or } A &= B * ((C+460.)^4 - (D+460.)^4) \\ \text{or } A_i &= B_i * (C_i - C_{i+1}), \quad i = 1, N \\ \text{or } A_i &= B_i * (C_i - D_i), \quad i = 1, N. \end{aligned}$$

Restrictions—All values must be floating-point numbers except the array length N, which must be an integer.

Calling Sequence—QMETER(C,D,B,A) or RDTNQS(D,C,B,A) or
QMTRI(N,C(DV),B(DV),A(DV)) or
QFORCE(N,C(DV),D(DV),B(DV),A(DV))

Subroutine QINTEG or QINTGI

Purpose—These subroutines perform a simple integration. They are useful for obtaining the integrals of flow rates calculated by QMETER, RDTNQS, QMTRI, or QFORCE. Their respective operations are

$$\begin{aligned} S &= S + Q * DT \\ \text{or } S_i &= S_i + Q_i * DT, \quad i = 1, N. \end{aligned}$$

Restrictions—All values must be floating-point numbers except N which must be an integer. Control constant DTIMEU should be used for the step size when doing an integration with respect to time. These subroutines should be called in Variables 2.

Calling Sequence—QINTEG(Q,DT,S) or QINTGI(N,Q(DV),DT,S(DV))

Subroutine CINSIN or SINARY

Purpose—These subroutines obtain the sine function of an angle or an array of angles. Their respective operations are

$$\begin{aligned} A &= \sin(B) \\ \text{or } A_i &= \sin(B_i), \quad i = 1, N. \end{aligned}$$

Restrictions—All angles must be in radians. All values must be floating-point numbers except N, which must be an integer.

Calling Sequence—CINSIN(B,A) or SINARY(N,B(DV),A(DV))

Subroutine CINCOS or COSARY

Purpose—These subroutines obtain the cosine function of an angle or array of angles. Their respective operations are

$$\begin{aligned} A &= \cos(B) \\ \text{or } A_i &= \cos(B_i), \quad i = 1, N. \end{aligned}$$

Restrictions—All angles must be in radians. All values must be floating-point numbers except the array length N, which must be an integer.

Calling Sequence—CINCOS(B,A) or COSARY(N,B(DV),A(DV))

Subroutine CINTAN or TANARY

Purpose—These subroutines obtain the tangent function of an angle or array of angles. Their respective operations are

$$\begin{aligned} A &= \tan(B) \\ \text{or } A_i &= \tan(B_i), \quad i = 1, N. \end{aligned}$$

Restrictions—All angles must be in radians. All values must be floating point numbers except the array length N, which must be an integer.

Calling Sequence—CINTAN(B,A) or TANARY(N,B(DV),A(DV))

Subroutine ARCSIN or ASNARY

Purpose—These subroutines obtain the angle corresponding to a sine function value or array of sine values. Their respective operations are

$$\begin{aligned} A &= \sin^{-1}(B) \\ \text{or } A_i &= \sin^{-1}(B_i), \quad i = 1, N. \end{aligned}$$

Restrictions—The angles are returned in radians with the limits $-\pi/2 \leq A \leq \pi/2$. All values must be floating point except for the array length N, which must be an integer.

Calling Sequence—ARCSIN(B,A) or ASNARY(N,B(DV),A(DV))

Subroutine ARCCOS or ACSARY

Purpose—These subroutines obtain the angle corresponding to a cosine function value or array of cosine values. Their respective operations are

$$A = \cos^{-1}(B)$$

$$\text{or } A_i = \cos^{-1}(B_i), \quad i = 1, N.$$

Restrictions—The angles are returned in radians with the limits $0 \leq A \leq \pi$. All values must be floating-point numbers except for the array length N, which must be an integer.

Calling Sequence—ARCCOS(B,A) or ACSARY(N,B(DV),A(DV))

Subroutine ARCTAN or ATNARY

Purpose—These subroutines obtain the angle corresponding to a tangent function value or array of tangent values. Their respective operations are

$$A = \tan^{-1}(B)$$

$$\text{or } A_i = \tan^{-1}(B_i), \quad i = 1, N.$$

Restrictions—The angles are returned in radians with the limits $-\pi/2 \leq A \leq \pi/2$. All values must be floating-point numbers except the array length N, which must be an integer.

Calling Sequence—ARCTAN(B,A) or ATNARY(N,B(DV),A(DV))

Subroutine EXPNTL or ARYEXP or EXPARY

Purpose—These subroutines perform an exponential operation. Their respective operations are

$$A = B^C$$

$$\text{or } A_i = B_i^C, \quad I = 1, N$$

$$\text{or } A_i = B_i^{C_i}, \quad I = 1, N.$$

Restrictions—All values must be positive floating-point numbers except N, which must be an integer.

Calling Sequence—EXPNTL(C,B,A) or ARYEXP(N,C,B(DV),A(DV)) or EXPARY(N,C(DV),B(DV),A(DV))

Subroutine LOGT or LOGTAR

Purpose—These subroutines obtain the base 10 log function of a number or array of numbers. Their respective operations are

$$A = \log_{10}(B)$$

$$\text{or } A_i = \log_{10}(B_i), \quad i = 1, N.$$

Restrictions—All values must be positive floating-point numbers except N, which must be an integer.

Calling Sequence—LOGT(B,A) or LOGTAR(N,B(DV),A(DV))

Subroutine LOGE or LOGEAR

Purpose—These subroutines obtain the base e log function of a number or array of numbers. Their respective operations are

$$A = \log_e(B)$$

$$\text{or } A_i = \log_e(B_i), \quad i = 1, N.$$

Restrictions—All values must be positive floating-point numbers except N, which must be an integer.

Calling Sequence—LOGE(B,A) or LOGEAR(N,B(DV),A(DV))

Subroutine SQROOT or SQROTI

Purpose—These subroutines obtain the square root of a number or array of numbers, respectively. Their respective operations are

$$A = +\sqrt{B}$$

$$\text{or } A_i = +\sqrt{B_i}, \quad i = 1, N.$$

Restrictions—The A and B values must be floating-point numbers. The N must be an integer.

Calling Sequence—SQROOT(B,A) or SQROTI(N,B(DV),A(DV))

Subroutine CMPXSR or CSQRI

Purpose—These subroutines obtain the complex square root of a complex number or an array of complex numbers, respectively. Their respective operations are

$$A + iB = \sqrt{C} + iD, \quad i = \sqrt{-1}$$

$$\text{or } A_j + iB_j = \sqrt{C_j} + iD_j, \quad j = 1, N.$$

Restrictions—All numbers must be floating-point except N, which must be an integer.

Calling Sequence—CMPXSR(C,D,A,B) or CSQRI(N,C(DV),D(DV),A(DV),B(DV))

Subroutine CMPXMP or CMPYI

Purpose—These subroutines will multiply two complex numbers or the corresponding elements of arrays of complex numbers. Their respective operations are

$$\begin{aligned} A + iB &= (C + iD)*(E + iF), & i &= \sqrt{-1} \\ \text{or } A_j + iB_j &= (C_j + iD_j)*(E_j + iF_j), & j &= 1, N \end{aligned}$$

Restrictions—All numbers must be floating point except for N, which must be an integer.

Calling Sequence—CMPXMP(C,D,E,F,A,B) or
CMPYI(N,C(DV),D(DV),E(DV),F(DV),A(DV),B(DV))

Subroutine CMPXDV or CDIVI

Purpose—These subroutines will divide two complex numbers or the corresponding elements of arrays of complex numbers. Their respective operations are

$$\begin{aligned} A + iB &= (C + iD)/(E + iF), & i &= \sqrt{-1} \\ \text{or } A_j + iB_j &= (C_j + iD_j)/(E_j + iF_j), & j &= 1, N \end{aligned}$$

Restrictions—All numbers must be floating point except for N, which must be an integer.

Calling Sequence—CMPXDV(C,D,E,F,A,B) or
CDIVI(N,C(DV),D(DV),E(DV),F(DV),A(DV),B(DV))

Subroutine NEWTRT or NEWRT4

Purpose—These subroutines utilize Newton's method to obtain one root of a cubic or quartic equation, respectively. The root must be in the neighborhood of the supplied initial guess, and up to 100 iterations are performed in order to obtain an answer within the specified tolerance. If the tolerance is not met, an answer of 10^{38} is returned. The respective equations are

$$\begin{aligned} f(X) &= A1+A2*X+A3*X^2+A4*X^3 = 0.0\pm T \\ \text{or } g(X) &= A1+A2*X+A3*X^2+A4*X^3+A5*X^4 = 0.0\pm T \end{aligned}$$

where X starts as the initial guess RI and finishes as the final answer RF. T is the tolerance.

Restrictions—All data values must be floating-point numbers.

Calling Sequence—NEWTRT(A(DV),T,RI,RF) or NEWRT4(A(DV),T,RI,RF)

Subroutine PLYNML or PLYARY

Purpose—These subroutines calculate Y from the following polynomial equation:

$$Y = A_1 + A_2 * X + A_3 * X^2 + A_4 * X^3 + \dots + A_N * X^{N-1}.$$

The number of terms is variable, but all the A coefficients must be entered no matter what their value.

Restrictions—All values must be floating-point numbers except the number of coefficients N, which must be an integer.

Calling Sequence—PLYNML(X,A1,A2,A3,...,AN,Y) or PLYARY(N,X,A(DV),Y)

Subroutine SMPINT or TRPZD

Purpose—These subroutines perform area integrations by Simpson's rule and the trapezoidal rule, respectively. Simpson's rule requires that an odd number of points be supplied. If an even number of points is supplied, SMPINT will apply the trapezoidal rule to the last incremental area but Simpson's rule elsewhere. The respective operations are

$$\begin{aligned} A &= DX * (Y_1 + 4Y_2 + 2Y_3 + 4Y_4 + \dots + Y_N) / 3 \\ \text{or } A &= DX * (Y_1 + 2Y_2 + 2Y_3 + 2Y_4 + \dots + Y_N) / 2. \end{aligned}$$

Restrictions—The DX increment must be uniform between all the Y points. All values must be floating point except N, which must be an integer.

Calling Sequence—SMPINT(N,DX,Y(DV),A) or TRPZD(N,DX,Y(DV),A)

Subroutine TRPZDA

Purpose—This subroutine performs area integration by the trapezoidal rule. It should be used where the DX increment is not uniform between the Y values but the corresponding X value for each Y value is known. The operation performed is as follows:

$$A = \frac{1}{2} \sum (X_i - X_{i-1}) * (Y_i + Y_{i-1}), \quad i = 2, N.$$

Restrictions—All values must be floating-point numbers except the array length N, which must be an integer.

Calling Sequence—TRPZDA(N,X(DV),Y(DV),A)

Subroutine PRESS or SPRESS

Purpose—These routines are useful for impressing nodal pressures in one-dimensional flow paths once the entry pressure P1, path conductance G, and flow rate W are known. The respective equations are

$$P2 = P1 - W/G$$

$$\text{or } P_{i+1} = P_i - W/G_i, \quad i = 1, 2, 3, \dots, N.$$

Restrictions—For SPRESS, the pressures and conductors must be sequential and in ascending order; the number of pressure points to be calculated must be supplied as the integer N.

Calling Sequence—PRESS(P1,W,G,P2) or SPRESS(N,P1(DV),W,G(DV))

Subroutine EFFG

Purpose—Subroutine EFFG is a pressure network of the type in Fig. 11.

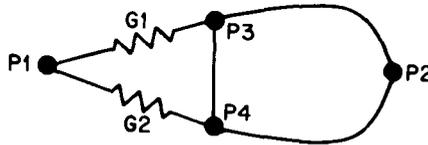


Fig. 11

Where the values of the identified elements are known, this subroutine will calculate the effective conductance GE from P1 to P2. Any interconnections may occur in the space, but only P2, P3 and P4 may be on the boundary and no elements may cross it. The equation utilized is

$$GE = (G1*(P1-P3) + G2*(P1-P4))/(P1-P2).$$

Restrictions—See above. May not be used where capacitors appear on the internal nodes.

Calling Sequence—EFFG(P1,P2,P3,P4,G1,G2,GE)

Subroutine EFFEMS

Purpose—This subroutine calculates the effective emissivity E between parallel flat plates by the following equation:

$$E = 1.0/(1.0/E1 + 1.0/E2 - 1.0),$$

where E1 and E2 are the emissivities of the two surfaces under consideration.

Restrictions—Arguments must be floating-point numbers.

Calling Sequence—EFFEMS(E1,E2,E)

Output Subroutines

Name	Page
STNDRD, PRNTMP, PRINT, PRINTL	83
PRINTA, PRNTMA, PUNCHA	84
TPRINT, READ, WRITE, EOF, REWIND	85

Subroutine STNDRD or PRNTMP

Purpose—Subroutine STNDRD causes a line of output to be printed giving the present time, the last time step used, the most recent CSGMIN value, the maximum diffusion temperature change calculated over the last time step, and the maximum relaxation change calculated over the last iteration. RNN refers to the relative node number on which something occurred. The line of output looks as follows:

* * * * *

TIME_____DTIMEU_____CSGMIN(RNN)_____DTMPCC(RNN)_____ARLXCC(RNN)_____

Subroutine PRNTMP internally calls on STNDRD and also lists the temperature of every node in the network according to relative node number. The relative node number vs actual node number dictionary printed out with the input data should be consulted to determine temperature locations on the thermal network model.

Restrictions—No arguments are required or allowed. These subroutines should be used with network problems only.

Calling Sequence—STNDRD or PRNTMP

Subroutine PRINT or PRINTL

Purpose—These subroutines allow individual floating-point numbers to be printed out. The arguments may reference temperature, capacitance, source locations, conductors, constants, or unique array locations. In addition, subroutine PRINTL allows each value to be preceded or labeled by a 6-character alphanumeric word. The number of arguments is variable, but the “label” array used for PRINTL should contain a label for each argument.

Restrictions—These subroutines do not call on STNDRD. The user may call on it if he desires time control information. Any control constant may be addressed in order to see what its value is; integers must first be floated.

Calling Sequence—PRINT(T,C,Q,G,K,...,A+) or PRINTL(LA(DV),T,C,Q,G,K,...,A+)

Subroutine PRINTA

Purpose—This subroutine allows the user to print out an array of values, five to the line. The integer array length N and the first data value location must be specified. Each value receives an indexed label. The user must supply a 6-character alphanumeric word L to be used as a common label and an integer value M to begin the index count.

Restrictions—The array values to be printed must be floating-point numbers.

Calling Sequence—PRINTA(L,A(DV),N,M)

If the label was the work TEMP, N was 3, and M was 6, the line of output will look as follows:

TEMP (6) valueTEMP (7)value TEMP (8)value

Subroutine PRNTMA

Purpose—This subroutine allows the user to print out up to 10 arrays in a column format. The individual elements are not labeled, but each column receives a 2-line heading of 12 alphanumeric characters each. The 2-line heading must be supplied as a single array of four words, six characters each. The user must supply the starting location of each label array and value array. The number of values in each value array must agree and be supplied as the integer N. The value arrays must contain floating-point numbers.

Restrictions—Labels must be alphanumeric, while values must be floating point. All floating-point-value arrays must contain the same number of values.

Calling Sequence—PRNTMA(N,LA1(DV),VA1(DV),LA2(DV),VA2(DV),...)

Subroutine PUNCHA

Purpose—This subroutine enables a user to punch out an array of data values in any desired format. The F argument must reference a FORTRAN format which has been input as an array, including the outer parentheses but deleting the word *format*. The second argument must address the first data value of the array of sequential values. The third argument N must be the integer number of data values in the array.

Restrictions—Punched cards must be asked for on the job request form.

Calling Sequence—PUNCHA(F(DV),A(DV),N)

Subroutine TPRINT

Purpose—Subroutine TPRINT makes a call to STNDRD, then lists the actual node number and corresponding temperature for every node in a network.

Restrictions—This subroutine may be called from any of the operations blocks.

Calling Sequence—TPRINT

Subroutine READ or WRITE

Purpose—These subroutines enable the user to read and write arrays of data as binary information on magnetic tape. The first argument L must be the integer number of the logical tape being addressed. The second argument X must address the first data value of the array to be written out or the starting location for data to be read into. The third argument N must be an integer. For WRITE it is the number of data values to be written on tape as a record. For READ it is the number of data values to be read in from tape from the next record, not necessarily the entire record.

Restrictions—The user should check section VII to determine which logical units are available and control card requirements. All processed information must be in binary.

Calling Sequence—READ(L,X(DV),N) or WRITE(L,X(DV),N)

Subroutine EOF or REWIND

Purpose—These subroutines enable the user to write end of file marks on magnetic tape and to rewind them. They are generally used in conjunction with subroutines READ and WRITE discussed above. The single argument L must be the integer logical tape number of the unit being activated.

Restrictions—The user should check section VII to determine available logical units.

Calling Sequence—EOF (L) or REWIND (L)

Matrix Subroutines

Name	Page
ZERO, ONES, UNITY, SIGMA, GENALP, GENCOL	87
SHIFT, REFLCT, SHUFL, COLMAX, COLMIN	88
ELEADD, ELESUB, ELEMUL, ELEDIV, ELEINV, EFSIN, EFASN	89
EFCOS, EFACS, EFTAN, EFATN, EFLOG, EFSQR	90
EFEXP, EFPOW, MATRIX, SCALAR, DISAS, ASSMBL	91
DIAG, COLMLT, ROWMLT, ADDALP, ALPHAA, AABB	92
BTAB, INVRSE, MULT	93
TRANS, POLMLT, POLVAL, PLYEVL	94
POLSOV, JACOBI, MODES	95
MASS	96
STIFF, LIST	97
PUNCH, Matrix Data Storage and Retrieval, CALL, FILE, ENDMOP, LSTAPE	98

NOTE: All of the above subroutines require that matrixes be entered as positive numbered arrays having the integer number of rows and columns as the first two data values followed by the floating-point element values in row order. The above package of subroutines is referred to as MOPAS, for Matrix Oriented Production Assembly System.

Subroutine ZERO or ONES

Purpose—These subroutines generate a matrix [Z] such that every element is zero or one, respectively.

Restrictions—The matrix to be generated must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values. The NR and NC arguments are the integer number of rows and columns, respectively.

Calling Sequence—ZERO(NR,NC,Z(IC)) or ONES(NR,NC,Z(IC))

Subroutine UNITY or SIGMA

Purpose—These are square matrix generation subroutines. UNITY generates a square matrix such that the main diagonal elements are one and all other elements are zero. SIGMA generates a square matrix such that all elements on and below the main diagonal are one and the remaining elements are zero.

Calling Sequence—UNITY(N,Z(IC)) or SIGMA(N,Z(IC))

Restrictions—The matrix [Z] to be generated must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values. The integer number of rows and columns are equal and must be input as the argument N.

Subroutine GENALP or GENCOL

Purpose—These are special matrix generation subroutines. GENALP will generate a matrix such that every element is equal to a constant C. GENCOL will generate a column matrix such that the first element is equal to X1 and the last element is equal to X2. The intermediate elements receive equally incremented values such that a linear relationship is established between row number and element value.

Restrictions—The NR and NC arguments refer to the integer number of rows and columns, respectively. X1, X2, and C must be floating-point values. The generated matrixes must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values.

Calling Sequence—GENALP(NR,NC,C,Z(IC)) or GENCOL(X1,X2,NR,Z(IC))

Subroutine SHIFT or REFLECT

Purpose—These subroutines may be used to move an entire matrix from one location to another. SHIFT moves the matrix exactly as is and REFLECT moves it and reverses the order of the elements within each column. The last element in each column becomes the first and the first becomes the last, etc.

REFLECT uses three dynamic storage locations plus an additional one for each row.

Restrictions—The matrixes must be of identical size, and the integer number of rows and columns must be the first two data values. The [Z] matrix may be overlaid into the [A] matrix.

Calling Sequence—SHIFT(A(IC),Z(IC)) or REFLECT(A(IC),Z(IC))

Subroutine SHUFL

Purpose—This subroutine allows the user to reorder the size of a matrix as long as the total number of elements remains unchanged. The row order input matrix [A] is transposed to achieve column order and then reformed as a vector by sequencing the columns in ascending order. This vector is then reformed into a column order matrix by taking a column at a time sequentially from the vector. The newly formed column matrix is then transposed and output as the row order matrix [Z].

Restrictions—The matrixes must be identical in size and have their respective integer number of rows and columns as the first two data values. The number of rows times columns for [A] must equal the number of rows times columns of [Z].

Calling Sequence—SHUFL(A(IC),Z(IC))

Subroutine COLMAX or COLMIN

Purpose—These subroutines search an input matrix to obtain the maximum or minimum values within each column, respectively. These values are output as a single row matrix [Z] having as many columns as the input matrix [A].

Restrictions—Each matrix must have its integer number of rows and columns as the first two data values.

Calling Sequence—COLMAX(A(IC),Z(IC)) or COLMIN(A(IC),Z(IC))

Subroutine ELEADD or ELESUB

Purpose—These subroutines add or subtract the corresponding elements of two matrixes, respectively;

$$\begin{matrix} m*n & m*n & m*n \\ [Z] = [A] \pm [B], & z_{ij} = a_{ij} \pm b_{ij}. \end{matrix}$$

Restrictions—All matrixes must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] or [B] matrix.

Calling Sequence—ELEADD(A(IC),B(IC),Z(IC)) or ELESUB(A(IC),B(IC),Z(IC))

Subroutine ELEMUL or ELEDIV

Purpose—These subroutines multiply or divide the corresponding elements of two matrixes, respectively;

$$\begin{matrix} m*n & m*n & m*n \\ [Z] = [A] */ [B], & z_{ij} = a_{ij} */ b_{ij}. \end{matrix}$$

Restrictions—All matrixes must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] or [B] matrix.

Calling Sequence—ELEMUL(A(IC),B(IC),Z(IC)) or ELEDIV(A(IC),B(IC),Z(IC))

Subroutine ELEINV

Purpose—This subroutine obtains the reciprocal of each element of the [A] matrix and places it in the corresponding element location of the [Z] matrix;

$$z_{ij} = 1.0/a_{ij}.$$

Restrictions—The matrixes must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

Calling Sequence—ELEINV(A(IC),Z(IC))

Subroutine EFSIN or EFASN

Purpose—These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \sin(a_{ij}) \text{ or } z_{ij} = \arcsin(a_{ij}).$$

Restrictions—The matrixes must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlaid into the [A] matrix.

Calling Sequence—EFSIN(A(IC),Z(IC)) or EFASN(A(IC),Z(IC))

Subroutine EFCOS or EFACS

Purpose—These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \cos (a_{ij}) \text{ or } z_{ij} = \arccos (a_{ij}).$$

Restrictions—The matrixes must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlaid into the [A] matrix.

Calling Sequence—EFCOS(A(IC),Z(IC)) or EFACS(A(IC),Z(IC))

Subroutine EFTAN or EFATN

Purpose—These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \tan (a_{ij}) \text{ or } z_{ij} = \arctan (a_{ij}).$$

Restrictions—The matrixes must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlaid into the [A] matrix.

Calling Sequence—EFTAN(A(IC),Z(IC)) or EFATN(A(IC),Z(IC))

Subroutine EFLOG or EFSQR

Purpose—These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \log_e (a_{ij}) \text{ or } z_{ij} = \sqrt{a_{ij}}.$$

Restrictions—The matrixes must be identical in size and have the integer number of rows and columns as the first two data values. All elements in the [A] matrix must be positive.

Calling Sequence—EFLOG(A(IC),Z(IC)) or EFSQR(A(IC),Z(IC))

Subroutine EFEXP or EFPOW

Purpose—These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = e^{a_{ij}} \text{ or } z_{ij} = a_{ij}^{\alpha}.$$

Restrictions—The matrixes must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlaid into the [A] matrix. The exponent α may be an integer or floating-point number. However, if any elements in [A] are negative then α must be an integer.

Calling Sequence—EFEXP(A(IC),Z(IC)) or EFPOW(A(IC), α ,Z(IC))

Subroutine MATRIX or SCALAR

Purpose—Subroutine MATRIX allows a constant to replace a specific matrix element, and subroutine SCALAR allows a specific matrix element to be placed into a constant location. The integers I and J designate the row and column position of the specific element;

$$z_{ij} = C \text{ or } C = z_{ij}.$$

Restrictions—The matrix must have the integer number of rows and columns as the first two data values. Checks are made to insure that the identified element is within the matrix boundaries.

Calling Sequence—MATRIX(C,I,J,Z(IC)) or SCALAR(Z(IC),I,J,C)

Subroutine DISAS or ASSMBL

Purpose—These subroutines allow a user to operate on matrixes in a partitioned manner by disassembling a submatrix [Z] from a parent matrix [A] or assembling a submatrix [Z] into a parent matrix [A].

Restrictions—The I and J arguments are integers which identify (by row and column number, respectively) the upper left-hand corner position of the submatrix within the parent matrix. All matrixes must have exactly enough space and contain the integer number of rows and columns as the first two data values. The NR and NC arguments are the integer number of rows and columns, respectively, of the disassembled submatrix. If the submatrix exceeds the bounds of the parent matrix an appropriate error message is written and the program terminated.

Calling Sequence—DISAS(A(IC),I,J,NR,NC,Z(IC)) or ASSMBL(Z(IC),I,J,A(IC))

Subroutine DIAG

Purpose—Given a 1*N or N*1 matrix [V], this subroutine forms a full square N*N matrix [Z]. The [V] values are placed sequentially on the main diagonal of [Z] and all off-diagonal elements are set to zero.

Restrictions—Both matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values.

Calling Sequence—DIAG(V(IC),Z(IC))

Subroutine COLMLT or ROWMLT

Purpose—To multiply each element in a column or row of matrix [A] by its corresponding element from the matrix [V] which is conceptually a diagonal matrix but stored as a vector; i.e., 1*N or N*1 matrix. The matrix [Z] is the product.

Restrictions—The matrixes must have exactly enough space and contain the integer number of rows and columns as the first two data values. The matrixes being multiplied must be conformable.

Calling Sequence—COLMLT(A(IC),V(IC),Z(IC)) or ROWMLT(V(IC),A(IC),Z(IC))

Subroutine ADDALP or ALPHAA

Purpose—These subroutines add a constant to or multiply a constant times every element in a matrix;

$$z_{ij} = C + a_{ij} \quad \text{or} \quad z_{ij} = C * a_{ij}.$$

Restrictions—The matrixes must have exactly enough space and contain the integer number of rows and columns as the first two data values. C and all elements must be floating-point numbers. The [Z] matrix may be overlaid into the [A] matrix.

Calling Sequence—ADDALP(C,A(IC),Z(IC)) or ALPHAA(C,A(IC),Z(IC))

Subroutine AABB

Purpose—To sum two scaled matrixes;

$$\begin{matrix} m*n & m*n & m*n \\ [Z] = C1 [A] + C2 [B] & \text{and} & z_{ij} = C1 * a_{ij} + C2 * b_{ij}. \end{matrix}$$

Restrictions—All matrixes must be of identical size, contain exactly enough space, and contain the integer number of rows and columns as the first two data values. The output matrix [Z] may be overlaid into either of the input matrixes.

Calling Sequence—AABB(C1,A(IC),C2,B(IC),Z(IC))

Subroutine BTAB

Purpose—To perform the following matrix operation:

$$\begin{matrix} n*n & n*m & m*m & m*n \\ [Z] = [B]^t & [A] & [B] \end{matrix} \cdot$$

Restrictions—The matrixes must be conformable, contain exactly enough space, and contain the integer number of rows and columns as the first two data values. Subroutines *MULT* and *TRANS* are called on.

This subroutine (due to *MULT* and *TRANS*) uses $2*m*n+6$ dynamic storage locations.

Calling Sequence—BTAB(A(IC),B(IC),Z(IC))

Subroutine INVRSE

Purpose—To invert a square matrix;

$$\begin{matrix} n*n & n*n & n*n \\ \text{given } [A], & [Z] = [A]^{-1}. \end{matrix}$$

This subroutine requires n dynamic storage locations.

Restrictions—The matrixes must be square, identical in size, and contain the integer number of rows and columns as the first two data values. The output matrix $[Z]$ may be overlaid into the $[A]$ matrix.

Calling Sequence—INVRSE(A(IC),Z(IC))

Subroutine MULT

Purpose—To multiply two conformable matrixes together;

$$\begin{matrix} m*n & m*p & p*n \\ [Z] = [A] [B], & z_{ij} = a_{ik} * b_{kj}. \end{matrix}$$

This subroutine requires $n*m$ dynamic storage locations.

Restrictions—The matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values. If $[A]$ and $[B]$ are square, $[Z]$ may be overlaid into either of them.

Calling Sequence—MULT(A(IC),B(IC),Z(IC))

Subroutine TRANS

Purpose—Given a matrix $[A]$, $m \times n$, form its transpose as $[Z]$, $n \times m$.

This subroutine requires $n \times m$ dynamic storage locations.

Restrictions—Both matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values. The output matrix $[Z]$ may be overlaid into the $[A]$ matrix.

Calling Sequence—TRANS(A(IC),Z(IC))

Subroutine POLMLT

Purpose—This subroutine performs the multiplication of a given number of n th order polynomial coefficients by a similar number of m th order polynomial coefficients. The polynomials must be input as matrixes with the number of rows equal, and each row receives the following operation:

$$(c_1, c_2, c_3, \dots, c_k) = (a_1, a_2, \dots, a_n) * (b_1, b_2, \dots, b_m), k = m+n-1.$$

Restrictions—The matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values.

Calling Sequence—POLMLT(A(IC),B(IC),(IC))

Subroutine POLVAL

Purpose—Given a set of polynomial coefficients as the first row of matrix $[A]$, this subroutine evaluates the polynomial for the input complex number $X+iY$. The answer is returned as $U+iV$.

Restrictions— $[A]$ may be $m \times n$, but only the first row is evaluated.

Calling Sequence—POLVAL(A(IC),X,Y,U,V)

Subroutine PLYEVL

Purpose—Given a matrix $[A]$ containing an arbitrary number NRA of n th order polynomial coefficients and a column matrix $[X]$ containing an arbitrary number NRX of x values, this subroutine evaluates each polynomial for each x value. The answers are output as a matrix $[Z]$ of size NRX*NRA. Each set of polynomial coefficients in $[A]$ is a row in ascending order. An x value evaluated for the polynomials creates a row in $[Z]$ where the column number agrees with the polynomial row number.

Restrictions—The matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values.

Calling Sequence—PLYEVL(A(IC),Z(IC),Z(IC))

Subroutine POLSOV

Purpose—Given a set of polynomial coefficients as the first row in matrix [A], size (m,n+1), this subroutine calculates the complex roots which are returned as matrix [Z], size (n,2). Column 1 contains the real part and column 2 the imaginary part of the roots.

Restrictions—This subroutine presently is limited to n = 20. It internally calls on RTPOLY and utilizes some double precision.

Calling Sequence—POLSOV(A(IC),Z(IC))

Subroutine JACOBI

Purpose—This subroutine will find the eigenvalues [E] and eigenvector matrix [Z] associated with an input matrix [A];

$$\begin{matrix} n*n & n*n & n*n & n*1 \\ [A] & [Z] & = & [Z] [E]. \end{matrix}$$

This subroutine requires 2*n*n+6 dynamic storage locations.

Restrictions—The matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values.

Calling Sequence—JACOBI(A(IC),E(IC),Z(IC))

Subroutine MODES

Purpose—This subroutine solves the dynamic vibration equation

$$\begin{matrix} n*n & n*n & n*n & n*n & n*1 \\ [A] & [Z] & = & [B] [Z] & \begin{bmatrix} 1 \\ W^2 \end{bmatrix}, \end{matrix}$$

where [A] is the input inertia matrix associated with the kinetic energy and [B] is the input stiffness matrix associated with the strain energy. [Z] is the output eigenvector matrix associated with the frequencies of vibration W_i which are output in rad/sec as [R] and in hertzes as [C]; both [R] and [C] are n*1 matrixes.

This subroutine requires 3*n*n+9 dynamic storage locations.

Restrictions—The matrixes must have exactly enough space and contain their integer number of rows and columns as the first two data values. Subroutine JACOBI is called on.

Calling Sequence—MODES(A(IC),B(IC),Z(IC),R(IC),C(IC))

Subroutine MASS

Purpose—If a dynamic vibration problem is referred to a set of coordinates consisting of the deflections ζ_i and the rotations θ_i at N collocation points along the beam under consideration, then this subroutine generates the 2N by 2N inertia matrix [A] which appears in the following expression for kinetic energy.

$$T = \frac{1}{2} \left\{ \dot{\zeta}_1 \dots \dot{\zeta}_N \dot{\theta}_1 \dots \dot{\theta}_N \right\} [A] \begin{bmatrix} \dot{\zeta}_1 \\ \vdots \\ \dot{\zeta}_N \\ \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_N \end{bmatrix}$$

Restrictions—The mass and inertia data inputs to this subroutine are to be supplied as piecewise continuous slices; however, these arrays may be of arbitrary size and different in length from each other. The number of collocation points N which determines the ultimate size, 2N by 2N, of the output inertia matrix, is also chosen arbitrarily.

Calling Sequence—MASS(X(IC),DMPL(IC),RIPL(IC),CM(IC),A(IC))

Here

X is an N*1 matrix of collocation points referred to an arbitrary origin.

DMPL is an NDM*4 matrix of distributed mass per unit length slices, in which

- Col 1 is the location of the rear of a slice.
- Col 2 is the location of the front of a slice.
- Col 3 is the mass value at the rear of the slice.
- Col 4 is the mass value at the front of the slice.

RIPL is an NRI*4 matrix of distributed rotary inertia per unit length slices. The columns here are similar to DMPL.

CM is an NCM*4 matrix of concentrated mass items, where

- Col 1 is the attach point location for each item.
- Col 2 is the mass at this location.
- Col 3 is the location of its center of gravity.
- Col 4 is the amount of inertia about the center of gravity.

A is a 2N*2N output inertia matrix.

NOTE: Since this applies to DMPL, RIPL, and CM, the location of the values may not go beyond the limits of the collocation points in either direction.

Subroutine STIFF

Purpose—If a dynamic vibration problem is referred to a set of coordinates consisting of the deflections ζ_i and the rotations θ_i at N collocation points along the beam under consideration, then this subroutine generates the $2N$ by $2N$ stiffness matrix $[K]$ which appears in the following expression for the strain energy

$$U = \frac{1}{2} \left\{ \zeta_1 \dots \zeta_N \theta_1 \dots \theta_N \right\} [K] \begin{bmatrix} \zeta_1 \\ \vdots \\ \zeta_N \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix}$$

Restrictions—The stiffness and shear data inputs to this subroutine are to be supplied as piecewise continuous slices; however, these arrays may be of arbitrary size and different in length from each other. The number of collocation points N , which determine the ultimate size ($2N$ by $2N$) of the output stiffness matrix, is also chosen arbitrarily.

Calling Sequence—STIFF(X(IC),EI(IC),GA(IC),K(IC))

where

X is an N by 1 matrix of collocation points referred to an arbitrary origin.

EI is an NEI by 4 matrix of bending stiffness slices, where

Col 1 is the location of the rear of a slice.

Col 2 is the location of the front of a slice.

Col 3 is the stiffness value at the rear of a slice.

Col 4 is the stiffness value at the front of a slice.

GA is an NGA by 4 matrix of shear stiffness slices, where the columns here are similar to those for the EI distribution.

K is the output stiffness matrix size $2N$ by $2N$.

NOTE: Since this applies to EI and GA, the location of the values may not go beyond the limits of the collocation points in either direction.

Subroutine LIST

Purpose—This subroutine prints out the elements of a matrix $[A]$ and identifies each by its row and column number. The user must supply an alphanumeric name ALP and integer number NUM to identify the matrix. This is to maintain consistency with subroutines FILE and CALL.

Restrictions—The matrix must have its integer number of rows and columns as the first two data values.

Calling Sequence—LIST(A(IC),ALP,NUM)

Subroutine PUNCH

Purpose—This subroutine punches out a matrix [A], size $n*m$, one column at a time in any desired format. The argument FOR must reference a FORTRAN format statement that has been entered as a positive array. It must include the outer parenthesis but not the word FORMAT. The argument HEAD must be a single BCD word used to identify the matrix. Each column is designated and restarts use of the FORMAT statement.

This subroutine requires $n+3$ dynamic storage locations.

Restrictions—The matrix [A] must have exactly enough space and contain the integer number of rows and columns as the first two data values. Punched cards must be asked for on the job request form.

Calling Sequence—PUNCH(A(IC),HEAD,FOR(IC))

Matrix Data Storage and Retrieval

The ability to store and retrieve matrixes from tape is easily achieved through the use of the FILE and CALL subroutines. Matrixes are identified by an alphanumeric name, integer problem number, and the core address of or for the matrix. The CALL subroutine searches the matrix storage tape on logical 16 and brings the desired matrix into core. The FILE subroutine writes a matrix onto the logical 30 tape. Subroutine ENDMOP causes all matrixes from the logical 30 tape to be updated onto the logical 16 tape. In case of duplicate matrixes the one from logical 30 replaces the one on logical 16. A matrix which has been filed cannot be called until an ENDMOP operation has been performed. To create a new tape the user merely sets control constant NOCOPY nonzero and has a scratch tape mounted on logical 16. The user should check the section on control cards and deck setup to determine control card requirements.

Subroutine CALL or FILE

Purpose—To allow the user to retrieve or store matrixes on magnetic tape, see above. The H argument must be a 6-character alphanumeric word and N must be an integer number, both of which are used to identify the matrix.

Restrictions—See above. The matrix must have exactly enough space and contain the integer number of rows and columns as the first two data values.

Calling Sequence—CALL (H,N,A(IC)) or FILE (A(IC),H,N)

Subroutine ENDMOP or LSTAPE

Purpose—Subroutine ENDMOP should be used in conjunction with subroutines CALL and FILE, see above. It causes matrixes which have been filed by FILE on logical 30 to be updated onto logical 16. A call to subroutine LSTAPE will cause the output of the name, problem number, and size of every matrix stored on tape on logical 16.

Restrictions—See above.

Calling Sequence—ENDMOP or LSTAPE

Special Subroutines

Name	Page
SIMEQN, LSTSQU	99
IRRADE, IRRADI	100
SLRADE, SLRADI, SCRPFSA	101
ABLATS	102
LQDVAP, BIVLV	103
LINE	104
STATE	105
PSOFTS, TSOFP, TRNPRT	106
GSGDMP, LSTPCS, QMAP, TSAVE	107

Subroutine SIMEQN

Purpose—This subroutine solves a set of up to 10 linear simultaneous equations by the factorized inverse method. The problem size and all input and output values are communicated as a single, specially formatted, positive input array. The array argument must address the matrix order (N) which is input by the user. The first data value must be the integer order of the set (or size of the square matrix) followed by the coefficient matrix [A] in column order, the boundary vector {B}, and space for the solution vector {S}:

$$[A] \{S\} = \{B\}.$$

Restrictions—The integer count and matrix size must be integers; all other values must be floating point. The coefficient matrix is not modified by SIMEQN. Hence, changes to {B} only allow additional solutions to be easily obtained.

Calling Sequence—SIMEQN(A(N))

where the array is formatted exactly as follows:

$$IC, N, A(1,1), A(1,2), \dots, A(N,N), B1, \dots, BN, S1, \dots, SN$$

Subroutine LSTSQU

Purpose—This subroutine performs a least squares curve fit to an arbitrary number of X, Y pairs to yield a polynomial equation of up to order 10. Rather than using a double precision matrix inverse, this subroutine calls on subroutine SIMEQN to obtain a simultaneous solution.

This subroutine requires 2*M dynamic storage core locations.

Restrictions—All values must be floating-point numbers except N and M, which must be integers. N is the order of the polynomial desired and is one less than the number of coefficients desired. M is the array length of the independent X or dependent Y values.

Calling Sequence—LSTSQU(N,M,X(DV),Y(DV),A(DV))

Subroutine IRRADI or IRRADE

Purpose—These subroutines simulate a radiosity network* within a multiple gray surface enclosure containing a nonabsorbing media. The input is identical for both subroutines. However, IRRADE utilizes explicit equations to obtain the solution by relaxation, and IRRADI initially performs a symmetric matrix algebra inverse and thereafter obtains the exact solution implicitly by matrix multiplication. The relaxation criteria of IRRADE is internally calculated and severe enough so that both routines generally yield identical results. However, IRRADE should be used when temperature-varying emissivities are to be considered, and IRRADI should be used when the surface emissivities are constant. Both subroutines solve for the J-node radiosity, obtain the net radiant heat flow rates to each surface, and return them sequentially in the last array that was initially used to input the surface temperatures. The user need not specify any radiation conductors within the enclosure.

Restrictions—The Fahrenheit system is required. The arbitrary number of temperature arguments may be constructed by a preceding BLDARY call. The emissivity, area, temperature-Q and upper half-FA arrays must be in corresponding order and of exact length. The first data value of the FA array must be the integer number of surfaces and the second the Stephan-Boltzmann constant in the proper units and then the FA floating-point values in row order. The diagonal elements (even if zero) must be included. As many radiosity subroutine calls as desired may be used. However, each call must have unique array arguments. The user should follow the radiosity routine by SCALE, BRKARY, or BKARAD to distribute the Q's to the proper source locations.

Calling Sequences—IRRADI(AA(IC),Aε(IC),AFA(IC),ATQ(IC)) or
IRRADE(AA(IC),Aε(IC),AFA(IC),ATQ(IC))

The arrays are formatted as follows:

```
AA(IC),A1,A2,A3,A4,...,AN,END
Aε(IC),ε1,ε2,ε3,ε4,...,εN,END
AFA(IC),N,σ,FA(1,1),FA(1,2),FA(1,3),FA(1,4),FA(1,5),...FA(1,N)
      FA(2,2),FA(2,3),FA(2,4),FA(2,5),...FA(2,N)
      FA(N-2,N-2),FA(N-2,N-1),FA(N-2,N)
      FA(N-1,N-1),FA(N-1,N)
      FA(N,N),END
ATQ(IC),T1,T2,T3,...TN,END
```

*A. K. Oppenheim, "Radiation Analysis by the Network Method," Trans. ASME 78, 725-735 (1956).

where $FA(1,2)$ is defined as $A(1) * F(1,2)$. After the subroutine has been performed the ATQ array is ATQ(IC),Q1,Q2,Q3,...QN,END.

Since $FA_1(1,2) \equiv FA_2(2,1)$ only the upper half triangle of the full FA matrix is required. IRRADI inverts this half-matrix in its own area; hence, approximately 300 surfaces may be considered using CINDA on a 65k-core machine.

Subroutine SLRADI or SLRADE

Purpose—These subroutines are very similar to IRRADI and IRRADE but are designed to solve for the solar heating rates within an enclosure. SLRADI inverts half a symmetric matrix in order to obtain implicit solutions while SLRADE obtains solutions explicitly by relaxation. SLRADE should be used when temperature varying solar emissivities are to be considered. The second data value of the AFA array must be the solar constant in the proper units. The AT array allows the user to input the angle (degrees) between the surface normal and the surface-sun line. The AI array allows the user to input an illumination factor for each surface which is the ratio from zero to one of the unshaded portion of the surface. The solar constant S, AT, and AI values may vary during the transient for both routines. No input surface temperatures are required. The absorbed heating rates are returned sequentially in the AQ array; the user may utilize SCALE, BRKARY, or BKARAD to distribute the heating rates to the proper source locations.

Restrictions—These routines are independent of the temperature system being used. All of the array arguments must reference the integer count set by the CINDA preprocessor and be of the exact required length. As many calls as desired may be made, but each call must have unique array arguments.

Calling Sequences—SLRADI(AA(IC),A ϵ (IC),AFA(IC),AT(IC),AI(IC),AQ(IC)) or
SLRADE(AA(IC),A ϵ (IC),AFA(IC),AT(IC),AI(IC),AQ(IC))

Subroutine SCRPF A

Purpose—To obtain the script FA value for radiant transfer within an enclosure. The input arrays are formatted as shown for subroutines IRRADI and IRRADE. The second data value in the AFA array is used as a final multiplier. If 1.0 the script FA values are returned; if σ then script σ FA values are returned. The script FA values are returned in the ASFA array which is formatted identically to the AFA array and may overlay it.

Restrictions—All array arguments must reference the integer count set by the CINDA preprocessor, and all arrays must be exactly the required length.

Calling Sequence—SCRPF A(AA(IC),A ϵ (IC),AFA(IC),ASFA(IC))

NOTE: Subroutine SYMLST(ASFA(IC)+3,ASFA(IC)+1) may be called to list the matrix values and identify them by row and column number. This routine and the implicit radiosity routines finalize the half-symmetric-coefficient matrix and call on SYMINV(AFA(IC)+3,AFA(IC)+1) to obtain the symmetric inverse.

Subroutine ABLATS

Purpose—ABLATS provides a simple ablation (sublimation) capability for the CINDA user. The user constructs the three-dimensional network without considering the ablative. Then in Variables 2 he simulates one-dimensional ablative attachments by calling ABLATS. ABLATS constructs the one-dimensional network and solves it by implicit forward-backward differencing (Crank-Nicholson method) using the time step set by the execution subroutine. Separate ablation arrays (AA) must be used for each ABLATS call. Required working space is obtained from unused program COMMON. Several ABLATS calls thereby share unused COMMON. The user must call subroutine PNTABL(AA) in the OUTPUT CALLS to obtain the ablation totals and temperature distribution.

Restrictions—ABLATS must be called in Variables 2 and may be used with any execution subroutine. Subroutines D1DEG1, NEWTR4, and INTRFC are called. All units must be consistent. The Fahrenheit system is required. Temperature-varying material property arrays must not exceed 60 doublets. Bivariate material properties may be simulated by calling BVSPSA prior to ABLATS. Cross-sectional area is always considered unity. Thermal conductivity, Stephan-Boltzmann constant, and density units must agree in area and length units.

This subroutine requires $3*(NSL+1)$ dynamic storage core locations.

Calling Sequence—ABLATS(AA(IC),R,CP,G,T,C)

where

C is the capacitance location of the three-dimensional node.

T is the temperature location of the three-dimensional node.

G is the location of the material thermal conductivity or the starting location (integer count) of a doublet G vs T array.

CP is the location of the material specific heat or the starting location (integer count) of a doublet C_p vs T array.

R is the location of the material density or the starting location (integer count) of a doublet ρ vs T array.

AA(IC) is the starting location of the ablation array which must be formatted as follows:

AA(IC)+1—the ablative link number, a user-specified identification integer.

AA(IC)+2—integer number of sublayers (NSL) desired; ABLATS subtracts from this the number of sublayers ablated.

AA(IC)+3—the initial temperature of the material; ABLATS replaces this with the outer surface temperature, always in degrees F.

AA(IC)+4—the impressed outer surface heating rate per unit area, radiation rates not included.

AA(IC)+5—material thickness; this is replaced by the sublayer thickness.

AA(IC)+6—surface area of the three-dimensional node; need not be unity.

AA(IC)+7—ablation temperature, degrees F.

AA(IC)+8—heat of ablation.

AA(IC)+9—Stephan-Boltzmann constant in consistent units.

AA(IC)+10—surface emissivity.

AA(IC)+11—space “sink” temperature, degrees F.
 AA(IC)+12—SPACE,N,END where N equals NSI + 4.

NOTE: The outer surface radiation loss is integrated over the time step.

Subroutine LQDVAP

Purpose—This subroutine allows the user to simulate the addition of liquid to a node. The network data is prepared as though no liquid exists at the node and is solved that way by the network execution subroutine. Then LQDVAP, which must be called from Variables 2, corrects the nodal solution in order to account for the liquid. If the nodal temperature exceeds the boiling point of the liquid, it is set to the boiling point.

The excess energy above that required to reach the boiling point is calculated and considered as absorbed through vaporization. If the liquid is completely vaporized the subroutine deletes its operations. The method of solution holds very well for explicit solutions, but may introduce some error when large time steps are used with implicit solutions.

Restrictions—This subroutine must be called from Variables 2.

Calling Sequence—LQDVAP(T,C,A(IC))

where

T is the temperature location of the node.
 C is the capacitance location of the node.
 A + 1 contains the initial liquid weight.
 A + 2 contains the liquid specific heat.
 A + 3 contains the liquid vaporization temperature.
 A + 4 contains the liquid heat of vaporization.
 A + 5 receives the liquid vaporization rate (weight/time).
 A + 6 receives the liquid vaporization total (total weight).
 A + 7 contains the liquid initial temperature.

Subroutine BIVLV

Purpose—This subroutine allows the user to specify the percentage flow rates through two parallel tubes with common end points. One tube must consist of a single flow conductor (G1) while the other tube may consist of one or more sequential flow conductors (G2(I), I = 1,N). The ratio of flow through G1 divided by the total flow may be calculated in any desired manner and must be supplied as the argument W. The conductor values of either one tube or the other are reduced in order to achieve the desired percentage flow rates irregardless of the pressure drop.

Restrictions—N must be an integer. G2 must address the first of the sequential conductors in that tube.

Calling Sequence—BIVLV(N,W,G1,G2(DV))

Subroutine LINE

Purpose—This subroutine computes the steady state changes in the thermodynamic and flow properties through a line of length L. The upstream properties must be defined and supplied. The following equations are simultaneously solved in an iterative fashion:

$$\rho_u v_u = \rho_d v_d \text{ (one-dimensional conservation of mass),}$$

where u denotes upstream and d denotes downstream;

$$I_u + \frac{v_u^2}{2} + \frac{Q}{W} = I_d + \frac{v_d^2}{2} \text{ (one-dimensional energy equation);}$$

$$P_u = P_d \text{ (momentum equation, simplified because of a very small pressure drop and low velocities);}$$

$$\rho = f(X,P,h)$$

$$T = g(X,P,h) \text{ (equations of state);}$$

$$X_L = h(X,P,h)$$

$$Q = h(W_p)L(T_w - T) \text{ (energy loss)}$$

$$R_e = W(4A/W_p)/A\mu \text{ (} R_e \text{ is the Reynolds number)}$$

$$P_r = C_p \mu / K \text{ (} P_r \text{ is the Prandtl number)}$$

$$h = 0.332 (R_e^{0.5})(P_r^{0.333}) \text{ for } R_e < 2100 \text{ (laminar flow)}$$

$$h = 0.023 (R_e^{0.8})(P_r^{0.4}) \text{ for } R_e > 2100 \text{ (turbulent flow)}$$

C_p , μ , and K are obtained from subroutine TRNPRT.

Restrictions—Subroutine LINE assumes that the flowing fluid is composed of a perfect noncondensable gas and a perfect condensable gas. This assumption involves the STATE subroutine which is called on. However, LINE does not need the variables X_L and T to evaluate the transport properties and the heat transfer coefficient h for the calculation of Q . The thermodynamic property arguments are upstream properties when calling LINE, but the downstream thermodynamic properties are in the same locations.

Calling Sequence—

LINE(A,W_p,L,T_w,W,A1(IC),A2(IC),A3(IC),...,A10(IC),P,T,X_L,X,I,V, ρ ,Q)

where

- A = flow area
- W_p = wetted perimeter
- L = tube length (floating point)
- T_w = wall temperature

W = mass flow rate
 A1 = the doublet interpolation array of condensable gas, μ vs T
 A2 = the doublet interpolation array of noncondensable gas, μ vs T
 A3 = the doublet interpolation array of condensable gas, k vs T
 A4 = the doublet interpolation array of noncondensable gas, k vs T
 A5 = the doublet interpolation array of condensable gas, C_p vs T
 A6 = the doublet interpolation array of noncondensable gas, C_p vs T
 A10 = the doublet interpolation array of condensable gas, heat of vaporization vs T
 P = pressure
 T = temperature
 X_L = mass fraction of liquid
 X = mass fraction of noncondensable gas
 I = enthalpy (floating point)
 V = v = velocity
 ρ = density
 Q = energy lost to wall

Subroutine STATE

Purpose—This subroutine computes the thermodynamic state for a mixture of an assumed noncondensable gas (hydrogen) and a condensable gas (water vapor). The subroutine establishes whether the mixture is superheated or saturated and gives its density (ρ_m), temperature (T), and liquid mass fraction (X_L). The hydrogen mass fraction (X_h), mixture pressure (P_m), and mixture enthalpy (I_m) are input. Vapor components are assumed to be perfect gases; that is,

$$\rho_v = \frac{P_v}{R_v T} \quad \rho_h = \frac{P_h}{R_h T}$$

$$I_v = C_{pv} T \quad I_h = C_{ph} T$$

where subscripts h and v refer to hydrogen and water vapor, respectively. The liquid constituent is assumed to have the following properties:

$$\rho_L = 62.4 \quad \text{and} \quad I_L = I_v - HV,$$

where

HV = heat of vaporization
 C_p = specific heat at constant pressure
 R = gas constant.

If the mixture is saturated, P_v is related to T by the saturation equation of subroutines PSOFTS and/or TSOFP. Mixture properties are obtained from the following equations:

$$\rho_m = \frac{1.0}{\frac{X_h}{\rho_h} + \frac{(1 - X_h - X_L)}{\rho_v} + \frac{X_L}{\rho_L}}$$

and

$$I_m = X_h I_h + (1 - X_L) I_v - X_L (HV).$$

Restrictions—The restrictions are those that are imposed for perfect gases and incompressible liquids. The pressures must be well below the critical point. A is a doublet interpolation array of HV as a function of temperature T. I_m must be a floating-point number.

Calling Sequence—STATE(X,P,I, ρ , X_L ,T,A(IC))

Subroutine PSOFTS

Purpose—This subroutine computes the saturation pressure of water vapor as a function of gas temperature. The relationship used is

$$\log_{10} \frac{P_c}{P} = \frac{x}{T_s} \left(\frac{A + Bx + Cx^2}{1 + Dx} \right),$$

where $x = T_c - T_s$, A, B, C, and D are constants, and P_c and T_c are critical points.

Restrictions—The gas temperature should be between 10° and 150° C.

Calling Sequence—PSOFTS(TS,P)

Subroutine TSOFP

Purpose—This subroutine computes gas temperature as a function of the saturation pressure of water vapor, using the same relation as in PSOFTS.

Restrictions—The same restrictions apply to TSOFP as to PSOFTS.

Calling Sequence—TSOFP(P,TS)

Subroutine TRNPRT

Purpose—This subroutine calculates the transport properties of a two-component gas mixture.

Restrictions—Only a two-component gas mixture is allowed, and the component properties must have already been evaluated at the desired temperature.

Calling Sequence—TRNPRT(V1,V2,G1,G2,C1,C2,V,G,C,P1)

where

VN is the viscosity of component N.

GN is the thermal conductivity of component N.

CN is the specific heat of component N.

P1 is the percent (by weight) of component 1.

V, G, and C are the viscosity, thermal conductivity, and specific heat of the mixture.

Subroutine CSGDMP or LSTPCS or QMAP

Purpose—These routines are designed to aid in the checkout of thermal problem data decks by listing the pseudo-compute sequence. CSGDMP calls upon Variables 1 and then prints out each relative diffusion node number with the capacitance and CSGMIN value of the node. For each node, all three routines identify the attached conductors by relative conductor number and type, and by the relative number of the adjoining node. CSGDMP also lists the conductance of the attached conductor and the type of the adjoining node. Either the SPCS or the LPCS option may be used. While the LPCS option allows every conductor attached to a node to be identified, the SPCS option identifies only conductors for the first relative node number on which they occur. After the diffusion nodes are processed, the connection information for the arithmetic nodes is listed. After listing the above information, control passes to the next sequentially listed subroutine.

QMAP has all the properties of CSGDMP. In addition, it prints the temperatures of each node and adjoining node, and the flux between them.

Restrictions—These routines are generally called from EXECNTN. CSGDMP and QMAP should never be called from Variables 1.

Calling Sequence—CSGDMP or LSTPCS or QMAP

Subroutine TSAVE

Purpose—This subroutine generates an external plotting data output file (unit 24) that can be used with the external plotting option to plot nodal temperature vs time. TSAVE records each nodal temperature at TIMEN and also saves the actual node numbers.

Restrictions—This routine should not be called more than 2000 times.

Calling Sequence—TSAVE

Internal Subroutines

Name	Name	Name
BIT	ITRATE	SMOPAS
COPY	POLYADD	TOPLIN
GENM	PYMLT1	UNPAK
GENST	RTPOLY	UPDMOP
HEDCOL	SETUP	WRTARY
INPUTG	SKPLIN	WRTL08
INPUTT		

These subroutines are called from other routines and not normally by the user.

ACKNOWLEDGMENTS

The author would like to thank Mr. Richard Perlut of the Mechanical Engineering Branch, Engineering Services Division, for his suggestions and technical assistance.

Appendix A

SAMPLE PROBLEM 1A

ORIGINAL RUN

A perfectly insulated one-dimensional bar has a constant heating rate applied to one end. Obtain the 10-min transient temperature response, at half-minute intervals, of the bar ends and at points 1/4, 1/2, and 3/4 of the way along the bar. The bar is initially at 80°F and receives a constant heating rate of 3.0 Btu/min. The length of the bar is 4 in., and it has a cross-sectional area of 1 sq. in. It has the following material properties:

$$\text{density} = 172.8 \text{ lb/ft}^3$$

$$\text{specific heat} = 0.35 \text{ Btu/lb}^\circ\text{F}$$

$$\text{thermal conductivity} = 0.2 \text{ Btu/in.}\cdot\text{min.}\cdot^\circ\text{F}$$

Figure A1a shows a schematic of the physical problem with the nodes appropriately placed and the dashed lines indicating the lumping of the system for capacitance purposes. The network representation is illustrated in Fig. A1b.

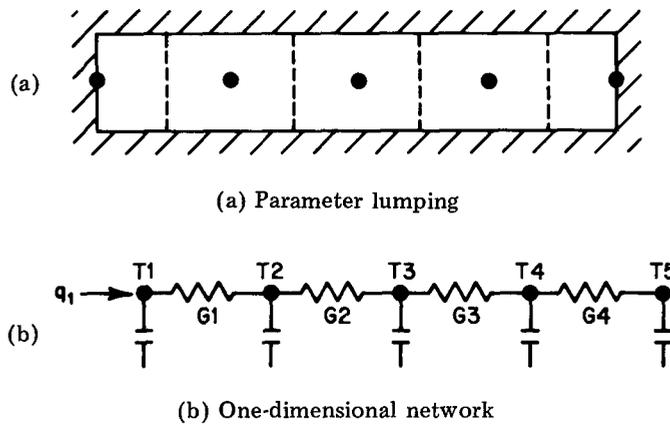


Fig. A1—Representation of a perfectly insulated bar

Capacitors receive the same number as the temperatures but with a C prefix. From the above information, we immediately calculate

$$C2 = C3 = C4 = \rho \cdot V \cdot C_p = 0.035 \text{ Btu/}^\circ\text{F}$$

$$C1 = C5 = C2/2.0 = 0.0175 \text{ Btu/}^\circ\text{F}$$

$$G1 = G2 = G3 = G4 = k \cdot A_c / \ell = 0.2 \text{ Btu/min}^\circ\text{F},$$

where $V = \ell * A_c$; length times cross-sectional area.

Since this is not a RECALL run, the first data card should be blank.

To apply explicit forward differencing to this problem, we must utilize the CNFRWD execution subroutine which requires the short pseudo-compute sequence. Hence, the title block is as follows:

```

8
↓
BCD 3THERMAL SPCS
BCD 9SAMPLE PROBLEM NO.1A
END
    
```

The nodal block is next and requires the node number, initial temperature, and capacitance of each node be listed.

```

8
↓
BCD 3NODE DATA
      1,80.,.0175,5,80.,.0175
GEN 2,3,1,80.,.035,1.,1.,1.
END
    
```

The conductor block requires that each conductor number be listed with the node numbers at either end, and the conductor value.

```

8
↓
BCD 3CONDUCTOR DATA
GEN 1,4,1,1,1,2,1,.2,1.,1.,1.
END
    
```

The only control constants required for CNFRWD are as follows:

```

8
↓
BCD 3CONSTANTS DATA
      TIMEND,10.,OUTPUT,.5,CSGFAC,2.
END
    
```

There are no array data and only one execution call; hence,

NRL REPORT 7656

```

1      8          21  25
↓      ↓          ↓   ↓
      BCD 3ARRAY DATA
      END
F      DIMENSION X( 100)
      BCD 3EXECUTION
F      NDIM=100
F      NTH=0
      CSGDMP
      CNFRWD
      END

```

There are no second variables operations, but we must apply the heating rate in the first variables;

```

8
↓
BCD 3VARIABLES 1
      STFSEP (3.,Q1)
END
BCD 3VARIABLES 2
END

```

The following completes the data input.

```

8
↓
BCD 3OUTPUT CALLS
      PRNTMP
END
BCD 3END OF DATA

```

Since PRNTMP lists the relative node numbers, and not the actual ones, the node dictionary will have to be consulted for conversion of relative to actual.

The above problem data deck processed by the CINDA program on the CDC-3800 as a standard run produces the output as given in the following printouts.

NOTE: The only alternative to the BCD 3END OF DATA card is a parameter change. A new job would require another set of control cards.

SEQUENCE 01713 STARTED PRINTING 07/13/73 AT 185705 ON LP01
DRUM SCOPE 2.1 COMPUTER TWO. MAX. DEMAND IS 570008 VERSION 004 10/31/72
SEQUENCE NUMBER 001713 STARTED AT TIME 182721 DATED 07/13/73
JOB(5)*40800800*899RCC.5
EQUIP.09=(CINDA MASTER*1.1.999)*RO*HI*DA
BINARY DECK
BANK*(0)*4/
LOAD*09
RUN*1*1000

PROGRAM NAMES

1 77672	SEARCH	00105	1 77603	STFFB	00067	1 76026	GENLNK	00540	1 74614	SPLIT	01026
1 74272	SKIP	00322	1 72717	CODERD	01353	1 71712	PSUEDO	00661	1 71627	ORMIN	00063
1 71417	PACK43	00210	1 71345	BIT	00052	1 71051	PRESUB	00274	1 70731	WRTBLK	00120
1 70673	WRTSCOPE	00036	1 65267	CINDA4	03404	1 62747	BLKCRD	01170	0 71134	DATARD	06643
1 62141	PREPRO	00605	0 60445	INITAL	10467	1 57571	FINAL	02350	1 57335	ALLOC.	00234
0 56177	IOH.	02246	1 57303	Q7QLODLC	00032	0 54622	IOP.	01355	1 57072	QBQERROR	00211
1 57005	RFI.	00065	1 56740	ENC.	00045	1 56705	DEC.	00033	0 54570	QBQIFUNI	00032
0 54544	EFT.	00024	0 54472	SLI.	00052	0 54430	QBINOUT4	00042	0 53654	IOB.	00554
0 53562	BSP.	00072	0 53527	QBQIFIOC	00033	0 53471	STH.	00036	0 53424	TSH.	00045
0 53407	REW.	00015	0 52764	IOS.	00423	0 52604	Q1GREINT	00160	0 52420	Q1QSTORE	00164
0 52342	Q1QENTRY	00056									

PROGRAM EXTENS.

NONE

LABELED COMMON

1 76566	CROBLK	01015	1 76014	TAPE	00012	1 75775	DATA	00017	1 75657	SUBLST	00116
1 75654	QLOGIC	00003	1 75642	PLOGIC	00012	1 72573	LOGIC	00124	1 64613	NARRAY	00454
1 64137	DIMARY	00454	1 62746	WJS	00001						

NUMBERED COMMON

0 23042	4	17500	1 00001	1	17500	1 17501	3	17500	1 37201	2	17500
---------	---	-------	---------	---	-------	---------	---	-------	---------	---	-------

ENTRY POINTS

0 77777	SENTRY		1 77675	SEARCH		0 52342	QBQDICT.		1 77607	STFFB	
1 76214	GENLNK		0 52420	Q3Q10040		0 52655	Q3Q00040		0 53303	THEND.	
1 63176	BLKCRD		0 53412	REW.		0 53430	TSH.		0 53475	STH.	
0 53275	QNSINGL.		1 74656	SPLIT		1 74275	SKIP		0 53536	QBQIFEOF	
0 53565	BSP.		0 53657	TSB.		0 53671	STB.		0 54446	QBQINP4	
1 73077	CODERD		0 72636	DATARD		0 54500	SLO.		0 54472	SLI.	
1 71760	PSUEDO		1 71422	PACK43		1 71632	ORMIN		0 54547	EFT.	
1 71345	BIT		1 71110	PRESUB		1 66371	CINDA4		1 70701	WRTSCOPE	
0 54573	QBQIFUNI		1 70764	WRTBLK		1 56710	DEC.		1 56743	ENC.	
1 57020	BFO.		1 62214	PREPRO		0 52346	QBQENTRY		0 67407	INITAL	
1 60643	FINAL		1 57074	QBQERROR		0 52343	EXIT		0 54430	QBQOUT4	
0 54625	IOP.		0 53037	QBQHIST.		1 57303	Q7QLODLC		0 53004	IOS.	
0 53307	IOR.		0 56223	IOH.		0 60320	.TSERR.		0 56177	BCDBUF.	
0 52764	IOE.		0 53156	QBQCHAIN		0 53300	QNDQUBL.		0 55644	ETAB.	
1 57473	ALLOC.		1 57343	RETURN.		1 57447	BUSY.		1 57456	IRETURN.	
1 57537	ALLOCIN.		0 60013	ELD.		0 60277	.REPCNT.		1 57072	QBERRORN	
1 57073	QBNOTRAC		1 57206	QBQERSET		1 57010	BFI.		0 54012	ELB.	
0 53532	QBQIFIOC		0 52604	Q1Q00100		0 52604	Q1Q01100		0 52623	Q1Q02100	
0 52627	Q1Q03100		0 52633	Q1Q04100		0 52637	Q1Q05100		0 52607	Q3Q00140	
0 52664	Q3Q01040		0 52615	Q3Q01140		0 52723	Q3Q02040		0 52673	Q3Q02140	
0 52732	Q3Q03040		0 52701	Q3Q03140		0 52741	Q3Q04040		0 52707	Q3Q04140	
0 52750	Q3Q05040		0 52715	Q3Q05140		0 52506	Q1Q10010		0 52506	Q1Q10020	
0 52454	Q1Q10030		0 52527	Q1Q10100		0 52501	Q1Q10120		0 52460	Q1Q10130	
0 52535	Q1Q10200		0 52463	Q1Q10210		0 52463	Q1Q10230		0 52527	Q1Q10300	
0 52512	Q1Q10310		0 52501	Q1Q10320		0 52522	Q1Q10400		0 52515	Q1Q10410	
0 52474	Q1Q10420		0 52474	Q1Q10430		0 52420	Q3Q10140		0 52420	Q3Q10240	
0 52420	Q3Q10340		0 52420	Q3Q10440							

EXECUTION STARTED AT 1828 -02

```

BCD 3THERMAL SPCS
BCD 9SAMPLE PROBLEM NO.1A
END
BCD 3NODE DATA
    1,80.,.0175,5,80.,.0175
GEN 2,3,1,80.,.035,1.,1.,1.
END

```

```

RELATIVE NODE NUMBERS                ACTUAL NODE NUMBERS

```

```

    1 THRU      5                1    5    2    3    4
BCD 3CONDUCTOR DATA
GEN 1*4*1*1*1*2*1*2*1*1*1*1.
END

```

```

RELATIVE CONDUCTOR NUMBERS          ACTUAL CONDUCTOR NUMBERS

```

```

    1 THRU      4                1    2    3    4
BCD 3CONSTANTS DATA
    TIMEND,10.,.OUTPUT,.,5,CSGFAC,2.

```

```

END
BCD 3ARRAY DATA
END

```

```

DIMENSION X( 100)                F

```

```

BCD 3EXECUTION                    F

```

```

NDIM = 100                        F

```

```

NTH = 0                            F

```

```

    CSGDMP
    CNFRWD

```

```

END

```

```

BCD 3VARIABLES 1
    STFSEP (3.,Q1)

```

```

END

```

```

BCD 3VARIABLES 2

```

```

END

```

```

BCD 3OUTPUT CALLS

```

```

    PRNTMP

```

```

END

```

4 SUBROUTINES NEEDED
CSGOMP CNFRND STFSEP PRNTP

MARY E. GEALY

LIBRARY,9,LCINDA
FTN,1=14,L,X

```

PROGRAM      LINKO
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN ,      DTIMEU,      TIMEND,      CSGFAC,
INLOOP ,      OTMPCA,      OPEITR,      DTIMEH,      DAMPA ,
IDAMPD ,      ATMPCA,      BACKUP,      TIMEO ,      TIMEM ,
IDTMPCC,      ATMPCC,      CSGMIN,      OUTPUT,      ARLXCA,
ILOOPECT,      DTIMEL,      DTIMEI,      CSGMAX,      CSGRAL,
ICSGRCL,      DRLXCA,      DRLXCC,      LINECT,      PAGECT,
IARLXCC,      LSPCS ,      ENGBAL,      BALENG,      NOCOPY,
INCSGM ,      NDTMPC,      NARLXC,      NATMPC,      ITEST ,
IJTEST ,      KTEST ,      LTEST ,      MTEST ,      RTEST ,
ISTEST ,      TTEST ,      UTEST ,      VTEST ,      LAXFAC
1,      IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
COMMON /LOGIC/ LNODE, LCOND, LCONST, LARRAY
LOGICAL LNODE, LCOND, LCONST, LARRAY
DIMENSION T( 5),C( 5),Q( 5),G( 4),K( 1),A( 1),
ICSEQ( 6),X( 100), NT( 5)
LNODE = .TRUE.
LCOND = .TRUE.
LCONST = .FALSE.
LARRAY = .FALSE.
1 CALL INPUTT
CALL EXECTN
GO TO 1
END

```

5.4DS LINK0

07/13/73

ED 0

PAGE NO.

2

		IDENT	LINK0
PROGRAM LENGTH		00033	
ENTRY POINTS	LINK0	00003	
BLOCK NAMES			
	TITLE	00024	
	TEMP	00005	
	CAP	00005	
	SOURCE	00005	
	COND	00004	
	PCS	00006	
	KONST	00001	
	ARRAY	00001	
	FIXCON	00062	
	DIMENS	00012	
	PRNT	00005	
	XSPACE	00146	
	LOGIC	00004	
EXTERNAL SYMBOLS			
	QBENTRY		
	Q3Q10040		
	QBQDICT.		
	INPUTT		
	EXECTN		
00126 SYMBOLS			

```

SUBROUTINE EXECTN
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
NDIM = 100
NTH = 0
CALL CSGDMP
CALL CNFRWD
RETURN
END

```

119

F
F

NRL REPORT 7656


```

SUBROUTINE VARBL1
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
INLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
IDAMPD , ATMPCA, BACKUP, TIMEO, TIMEM ,
IDTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
LLOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
LCSGRCL, ORLXCA, DRLXCC, LINECT, PAGECT,
LARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
LNCSGM , NOTMPC, NARLXC, NATMPC, ITEST ,
LJTEST , KTEST , LTEST , MTEST , RTEST ,
LSTEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL STFSEP(3.,Q(1))
RETURN
END

```

5.4DS VARBL1

07/13/73

ED 0

PAGE NO. 2

	IDENT	VARBL1
PROGRAM LENGTH		00016
ENTRY POINTS	VARBL1	00003
BLOCK NAMES		
TITLE		00024
TEMP		00001
CAP		00001
SOJRCE		00001
COND		00001
PCS		00001
KONST		00001
ARRAY		00001
FIXCON		00062
DIMENS		00012
PRNT		00001
XSPACE		00003
EXTERNAL SYMBOLS		
08QDICT.		
STFSEP		
00121 SYMBOLS		

```

SUBROUTINE VARBL2
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC
1NLOOP , DTMPCA, DPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, RACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTST , KTEST , LTEST , MTEST , RTEST ,
1STST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
RETURN
END

```

5.4DS

VARBL2

07/13/73

ED 0

PAGE NO.

2

	IDENT	VARBL2
PROGRAM LENGTH	00012	
ENTRY POINTS	00003	VARBL2
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
Q8QDICT.		
00120 SYMBOLS		

```

SUBROUTINE OUTCAL
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM?
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL PRNTMP
RETURN
END

```

5.4DS OUTCAL

07/13/73

ED 0

PAGE NO. 2

	IDENT	OUTCAL
PROGRAM LENGTH	00014	
ENTRY POINTS	00003	OUTCAL
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOJRCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLES		
QBQDICT.		
PRNTMP		
00121 SYMBOLES		

LOAD
RUN*5*2500

PROGRAM NAMES

1 77744	LINK0	00033	1 77371	EXECTN	00021	1 77353	VARBL1	00016	1 77341	VARBL2	00012
1 77325	OUTCAL	00014	1 77253	BIT	00052	1 75005	IOH.	02246	1 74753	Q7QLODLC	00032
1 74542	Q8QERROR	00211	1 74306	ALLOC.	00234	1 74165	STNDRD	00121	1 73657	UNPAK	00306
1 73571	TOPLIN	00066	1 73517	SLI.	00052	1 73455	Q8INOUT4	00042	1 73417	STH.	00036
1 72643	IOB.	00554	1 72463	Q1QREINT	00160	1 72040	IOS.	00423	1 70463	IOP.	01355
1 70335	PRNTMP	00126	1 70247	STFSEP	00066	1 67107	CNFRWD	01140	1 65445	CSGDMP	01442
1 64674	INPUTT	00551	1 64510	Q1QSTORE	00164	1 64432	Q8QENTRY	00056			

PROGRAM EXTENS.

NONE

LABELED COMMON

1 77720	TITLE	00024	1 77713	TEMP	00005	1 77706	CAP	00005	1 77701	SOURCE	00005
1 77675	COND	00004	1 77667	PCS	00006	1 77666	KONST	00001	1 77665	ARRAY	00001
1 77603	FIXCON	00062	1 77571	DIMENS	00012	1 77564	PRNT	00005	1 77416	XSPACE	00146
1 77412	LOGIC	00004									

NUMBERED COMMON

NONE

ENTRY POINTS

0 77777	SENTRY		1 77747	LINK0		1 64436	Q8QENTRY		1 64510	Q3Q10040	
1 64432	Q8QDICT.		1 64711	INPUTT		1 77374	EXECTN		1 66110	CSGDMP	
1 67143	CNFRWD		1 77356	VARBL1		1 70252	STFSEP		1 77344	VARBL2	
1 77330	OUTCAL		1 70356	PRNTMP		1 70466	IOP.		1 72113	Q8QHIST.	
1 72357	THEND.		1 72534	Q3Q00040		1 64433	EXIT		1 72646	TSB.	
1 73423	STH.		1 73473	Q8QINP4		1 73517	SLI.		1 72351	QNSINGL.	
1 73630	TOPLIN		1 73664	UNPAK		1 74246	STNDRD		1 74444	ALLOC.	
1 74314	RETURN.		1 74420	BUSY.		1 74427	IRETURN.		1 74544	Q8QERROR	
1 74753	Q7QLODLC		1 72354	QNDOUBL.		1 72060	IOS.		1 72363	IOR.	
1 72040	IOE.		1 75031	IOH.		1 75005	BCDBUF.		1 73525	SLO.	
1 77253	BIT		1 76621	ELD.		1 77105	.REPCNT.		1 77126	.TSERR.	
1 74542	Q8ERRORN		1 74543	Q8NOTRAC		1 74656	Q8QERSET		1 74510	ALLOCIN.	
1 73455	Q8QOUT4		1 73001	ELB.		1 72660	STB.		1 72463	Q1Q00100	
1 72463	Q1Q01100		1 72502	Q1Q02100		1 72506	Q1Q03100		1 72512	Q1Q04100	
1 72516	Q1Q05100		1 72466	Q3Q00140		1 72543	Q3Q01040		1 72474	Q3Q01140	
1 72602	Q3Q02040		1 72552	Q3Q02140		1 72611	Q3Q03040		1 72560	Q3Q03140	
1 72620	Q3Q04040		1 72566	Q3Q04140		1 72627	Q3Q05040		1 72574	Q3Q05140	
1 72232	Q8QCHAIN		1 71505	ETAB.		1 64576	Q1Q10010		1 64576	Q1Q10020	
1 64544	Q1Q10030		1 64617	Q1Q10100		1 64571	Q1Q10120		1 64550	Q1Q10130	
1 64625	Q1Q10200		1 64553	Q1Q10210		1 64553	Q1Q10230		1 64617	Q1Q10300	
1 64602	Q1Q10310		1 64571	Q1Q10320		1 64612	Q1Q10400		1 64605	Q1Q10410	
1 64564	Q1Q10420		1 64564	Q1Q10430		1 64510	Q3Q10140		1 64510	Q3Q10240	
1 64510	Q3Q10340		1 64510	Q3Q10440							

EXECUTION STARTED AT 1831 -41

SAMPLE PROBLEM NO.1A

A 5 NODE PROBLEM USING SPCS

NODE 1 HAS THE CSGMIN OF 87.5000-003, NODE 1 HAS THE CSGMAX OF 87.5000-003

NODE	C-VALUE	CSG-VALUE	COND TYPE	G-VALUE TO NODE	TYPE
1	17.500-003	87.500-003	1 LIN	20.000-002	3 DIFF
2	17.500-003	87.500-003	4 LIN	20.000-002	5 DIFF
3	35.000-003	87.500-003	2 LIN	20.000-002	4 DIFF
4	35.000-003	87.500-003	3 LIN	20.000-002	5 DIFF
5	35.000-003	87.500-003			

THIS NODE HAS BEEN PROCESSED

SAMPLE PROBLEM NO.1A

```

* * * *
TIME 00.00000+000 DTIMEU 00.00000+000 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 00.00000+000 ARLXCC( 0) 00.00000+000
      1 THRU      5      80.000000+000      80.000000+000      80.000000+000      80.000000+000

* * * *
TIME 50.00000-002 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 62.31991-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      10.827633+001      82.523666+000      95.775126+000      87.903573+000      83.778445+000

* * * *
TIME 10.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 48.58940-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      12.076382+001      91.468321+000      10.774199+001      98.616072+000      93.240150+000

* * * *
TIME 15.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 46.32151-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      13.177193+001      10.188878+001      11.866404+001      10.933036+001      10.374667+001

* * * *
TIME 20.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.94534-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      14.253495+001      11.255433+001      12.941279+001      12.004464+001      11.442650+001

* * * *
TIME 25.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.88294-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      15.325732+001      12.326054+001      14.013279+001      13.075893+001      12.513507+001

* * * *
TIME 30.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87259-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      16.397295+001      13.397348+001      15.084803+001      14.147321+001      13.584840+001

* * * *
TIME 35.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87088-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      17.468746+001      14.468754+001      16.156247+001      15.218750+001      14.656253+001

* * * *
TIME 40.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87059-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      18.540178+001      15.540179+001      17.227678+001      16.290179+001      15.727679+001

* * * *
TIME 45.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87055-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      19.611607+001      16.611607+001      18.299107+001      17.361607+001      16.799107+001
    
```

130

MARY E. GEALY

SAMPLE PROBLEM NO.1A

```

* * * *
TIME 50.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      20.683036+001      17.683036+001      19.370536+001      18.433036+001      17.870536+001
* * * *
TIME 55.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      21.754464+001      18.754464+001      20.441964+001      19.504464+001      18.941964+001
* * * *
TIME 60.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 3) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      22.825893+001      19.825893+001      21.513393+001      20.575893+001      20.013393+001
* * * *
TIME 65.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 3) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      23.897321+001      20.897321+001      22.584821+001      21.647321+001      21.084821+001
* * * *
TIME 70.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      24.968750+001      21.968750+001      23.656250+001      22.718750+001      22.156250+001
* * * *
TIME 75.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      26.040179+001      23.040179+001      24.727679+001      23.790179+001      23.227679+001
* * * *
TIME 80.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      27.111607+001      24.111607+001      25.799107+001      24.861607+001      24.299107+001
* * * *
TIME 85.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      28.183036+001      25.183036+001      26.870536+001      25.933036+001      25.370536+001
* * * *
TIME 90.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      29.254464+001      26.254464+001      27.941964+001      27.004464+001      26.441964+001
* * * *
TIME 95.00000-001 DTIMEU 21.40625-003 CSGMIN( 1) 87.50000-003 DTMPCC( 1) 45.87054-002 ARLXCC( 0) 00.00000+000
      1 THRU      5      30.325893+001      27.325893+001      29.013393+001      28.075893+001      27.513393+001
    
```

131

NRL REPORT 7656

SAMPLE PROBLEM NO.1A

* * * *

TIME 10.00000+000 DTIMEU 21.40625-003 CSGMIN(1) 87.50000-003 DTMPCC(1) 45.87054-002 ARLXCC(0) 00.00000+000
1 THRU 5 31.397321+001 28.397321+001 30.084821+001 29.147321+001 28.584821+001

END OF DATA

JOB MESSAGES

JOB:001713, 300,899RCC
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=

ST,LP00

LP 000 ASSIGNED
RELEASED 9=MT16=(CINDA MASTER),ED=01,RL=01,DATE=052373,RC=999

SAMPLE PROBLEM NO. 1A (Continued)

TSAVE AND PLOT RUN

This is an example of a TSAVE run that was made after the problem had been satisfactorily debugged and run. Since *that* run had produced printed output, all calls to output subroutines were removed from the deck to save processing time. Hence, the Execution block is now

```

1      7      21  25
↓      ↓      ↓   ↓
F      DIMENSION X( 100)
        BCD 3EXECUTION
F      NDIM = 100
F      NTH = 0
        END
    
```

TSAVE is the only subroutine in Output Calls.

```

8  12
↓  ↓
BCD 3OUTPUT CALLS
    TSAVE
END
    
```

Since the time and temperature limits were determined from the output in the previous run, it is possible to supply the plotting data so that the plot program can be run immediately after the CINDA problem (in the same job). This eliminates the need for equipping the TSAVE output tape (tape 24). The plotting data are as follows:

	1	12	22	32	42
	↓	↓	↓	↓	↓
card 1	CINDA	SAMPLE PROBLEM 1A			
card 2		0.00	10.00	0.00	320.00
card 3	(blank)				
	EOF				

The above plot data and revised problem data deck for problem 1A produce the following output and plots when processed on the CDC-3800 and plotted on the CalComp 565 plotter (the actual dimensions of the X and Y axes are 7 and 9 in., respectively). Plots of the data (Fig. A2a-e) follow the printouts.

SEQUENCE 01712 STARTED PRINTING 07/13/73 AT 185627 ON LP01
DRUM SCOPE 2.1 COMPUTER TWO. MAX. DEMAND IS 570008 VERSION 004 10/31/72
SEQUENCE NUMBER 001712 STARTED AT TIME 183152 DATED 07/13/73
JOB(5),40800800,899RCC,5
EQUIP,09=(CINDA MASTER,1.1,999),R0,HI,DA
BINARY DECK
BANK,(0),/4/
LOAD,09
RUN,1,1000

PROGRAM NAMES										
1 77672 SEARCH	00105	1 77603 STFFB	00067	1 76026 GENLNK	00540	1 74614 SPLIT	01026			
1 74272 SKIP	00322	1 72717 CODERD	01353	1 71712 PSUEDO	00661	1 71627 ORMIN	00063			
1 71417 PACK43	00210	1 71345 BIT	00052	1 71051 PRESUB	00274	1 70731 WRTBLK	00120			
1 70673 WRTSCOPE	00036	1 65267 CINDA4	03404	1 62747 BLKCRD	01170	0 71134 DATARD	06643			
1 62141 PREPRO	00605	0 60445 INITAL	10467	1 57571 FINAL	02350	1 57335 ALLOC.	00234			
0 56177 IOH.	02246	1 57303 Q7QLODLC	00032	0 54622 IOP.	01355	1 57072 QBERROR	00211			
1 57005 BFI.	00065	1 56740 ENC.	00045	1 56705 DEC.	00033	0 54570 QBQIFUNI	00032			
0 54544 EFT.	00024	0 54472 SLI.	00052	0 54430 QBINOUT4	00042	0 53654 IOB.	00554			
0 53562 BSP.	00072	0 53527 QBQIFIOC	00033	0 53471 STH.	00036	0 53424 TSH.	00045			
0 53407 REW.	00015	0 52764 IOS.	00423	0 52604 Q1QREINT	00160	0 52420 Q1QSTORE	00164			
0 52342 QBQENTRY	00056									

PROGRAM EXTENS.
NONE

LABELED COMMON									
1 76566 CRDBLK	01015	1 76014 TAPE	00012	1 75775 DATA	00017	1 75657 SUBLST	00116		
1 75654 QLOGIC	00003	1 75642 PLOGIC	00012	1 72573 LOGIC	00124	1 64613 NARRAY	00454		
1 64137 DIMARY	00454	1 62746 WJS	00001						

NUMBERED COMMON									
0 23042 4	17500	1 00001 1	17500	1 17501 3	17500	1 37201 2	17500		

ENTRY POINTS

0 77777 SENTRY		1 77675 SEARCH		0 52342 QBQDICT.		1 77607 STFFB	
1 76214 GENLNK		0 52420 Q3Q10040		0 52655 Q3Q00040		0 53303 THEND.	
1 63176 BLKCRD		0 53412 REW.		0 53430 TSH.		0 53475 STH.	
0 53275 QNSINGL.		1 74656 SPLIT		1 74275 SKIP		0 53536 QBQIFEOF	
0 53565 BSP.		0 53657 TSB.		0 53671 STB.		0 54446 QBQINP4	
1 73077 CODERD		0 72636 DATARD		0 54500 SLO.		0 54472 SLI.	
1 71760 PSUEDO		1 71422 PACK43		1 71632 ORMIN		0 54547 EFT.	
1 71345 BIT		1 71110 PRESUB		1 66371 CINDA4		1 70701 WRTSCOPE	
0 54573 QBQIFUNI		1 70764 WRTBLK		1 56710 DEC.		1 56743 ENC.	
1 57020 BFO.		1 62214 PREPRO		0 52346 QBQENTRY		0 67407 INITAL	
1 60643 FINAL		1 57074 QBQERROR		0 52343 EXIT		0 54430 QBQOUT4	
0 54625 IOP.		0 53037 QBQHIST.		1 57303 Q7QLODLC		0 53004 IOS.	
0 53307 IOR.		0 56223 IOH.		0 60320 .TSERR.		0 56177 BCDBUF.	
0 52764 IOE.		0 53156 QBQCHAIN		0 53300 QNDQOUBL.		0 55644 ETAB.	
1 57473 ALLOC.		1 57343 RETURN.		1 57447 BUSY.		1 57456 IRETURN.	
1 57537 ALLOCIN.		0 60013 ELD.		0 60277 .REPCNT.		1 57072 QBERRORN	
1 57073 QBNOIRAC		1 57206 QBQERSET		1 57010 BFI.		0 54012 ELB.	
0 53532 QBQIFIOC		0 52604 Q1Q00100		0 52604 Q1Q01100		0 52623 Q1Q02100	
0 52627 Q1Q03100		0 52633 Q1Q04100		0 52637 Q1Q05100		0 52607 Q3Q00140	
0 52664 Q3Q01040		0 52615 Q3Q01140		0 52723 Q3Q02040		0 52673 Q3Q02140	
0 52732 Q3Q03040		0 52701 Q3Q03140		0 52741 Q3Q04040		0 52707 Q3Q04140	
0 52750 Q3Q05040		0 52715 Q3Q05140		0 52506 Q1Q10010		0 52506 Q1Q10020	
0 52454 Q1Q10030		0 52527 Q1Q10100		0 52501 Q1Q10120		0 52460 Q1Q10130	
0 52535 Q1Q10200		0 52463 Q1Q10210		0 52463 Q1Q10230		0 52527 Q1Q10300	
0 52512 Q1Q10310		0 52501 Q1Q10320		0 52522 Q1Q10400		0 52515 Q1Q10410	
0 52474 Q1Q10420		0 52474 Q1Q10430		0 52420 Q3Q10140		0 52420 Q3Q10240	
0 52420 Q3Q10340		0 52420 Q3Q10440					

EXECUTION STARTED AT 1832 -20

```

BCD 3THERMAL SPCS
BCD 9SAMPLE PROBLEM NO.1A
END
BCD 3NODE DATA
1.80.0.0175,5.80.0.0175
GEN 2,3,1.80.0.035,1.0,1.0,1.0
END

```

```

RELATIVE NODE NUMBERS                                ACTUAL NODE NUMBERS

```

```

      1 THRU      5                1      5      2      3      4
BCD 3CONDUCTOR DATA
GEN 1,4,1,1,1,2,1,0,2,1,0,1,0,1
END

```

```

RELATIVE CONDUCTOR NUMBERS                            ACTUAL CONDUCTOR NUMBERS

```

```

      1 THRU      4                1      2      3      4
BCD 3CONSTANTS DATA
TIMEND,10.0,OUTPUT,.5,CSGFAC,2.

```

```

END
BCD 3ARRAY DATA
END

```

```

DIMENSION X( 100)

```

F

```

BCD 3EXECUTION

```

F

```

NDIM = 100

```

F

```

NTH = 0

```

```

CNFRWD

```

```

END
BCD 3VARIABLES 1

```

```

STFSEP (3.0,01)

```

```

END
BCD 3VARIABLES 2

```

```

END
BCD 3OUTPUT CALLS

```

```

TSAVE

```

```

END

```

```

END

```

```

END

```

3	SUBROUTINES NEEDED	CNFRWD	STFSEP	TSAVE
---	-----------------------	--------	--------	-------

LIBRARY,9,LCINDA
FTN,IE=14,L,X

```

PROGRAM      LINK0
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN ,      OTIMEU,      TIMEND,      CSGFAC,
INLOOP ,      DTMPCA,      OPEITR,      DTIMEH,      DAMPA ,
IDAMPD ,      ATMPCA,      BACKUP,      TIMEO ,      TIMEM ,
1DTMPCC,      ATMPCC,      CSGMIN,      OUTPUT,      ARLXCA ,
1LOOPCT,      DTIMEL,      DTIMEI,      CSGMAX,      CSGRAL,
1CSGRCL,      DRLXCA,      DRLXCC,      LINECT,      PAGECT,
1ARLXCC,      LSPCS ,      ENGBAL,      BALENG,      NOCOPY,
1NCSGM ,      NDTMPC,      NARLXC,      NATMPC,      ITEST ,
1JTEST ,      KTEST ,      LTEST ,      MTEST ,      RTEST ,
1STEST ,      TTEST ,      UTEST ,      VTEST ,      LAXFAC
1,      IDCNT
COMMON /DIMENS/ NNT, NND, NNC, NRG, NNG, NCON, NARY, LSEQ, DUM1, NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
COMMON /LOGIC/ LNODE, LCOND, LCONST, LARRAY
LOGICAL LNODE, LCOND, LCONST, LARRAY
DIMENSION T( 5),C( 5),Q( 5),G( 4),K( 1),A( 1),
1CSEQ( 6),X( 100), NT( 5)
LNODE = .TRUE.
LCOND = .TRUE.
LCONST = .FALSE.
LARRAY = .FALSE.
1 CALL INPUTT
CALL EXECTN
GO TO 1
END

```

5.4DS LINK0

07/13/73

ED 0

PAGE NO.

2

	IDENT	LINK0
PROGRAM LENGTH	00033	
ENTRY POINTS	LINK0	00003
BLOCK NAMES		
TITLE	00024	
TEMP	00005	
CAP	00005	
SOURCE	00005	
COND	00004	
PCS	00006	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00005	
XSPACE	00146	
LOGIC	00004	
EXTERNAL SYMBOLS		
QBQENTRY		
Q3Q10040		
QBQDICT.		
INPUTT		
EXECTN		
00126 SYMBOLS		

142

MARY E. GEALY

```

SUBROUTINE EXECTN
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
INLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
IDAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
IDTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
ILOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
ICSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
IARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
INCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
IJTEST , KTEST , LTEST , MTEST , RTEST ,
ISTEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
NDIM = 100
NTH = 0
CALL CNFRWD
RETURN
END

```

5.4Ds EXECTN

07/13/73

ED 0

PAGE NO. 2

	IDENT	EXECTN
PROGRAM LENGTH	00017	
ENTRY POINTS	EXECTN	00003
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
QBQDICT.		
CNFRWD		
00121 SYMBOLS		

144

MARY E. GEALY

```

SUBROUTINE VARBL1
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS, ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL STFSEP(3.,Q(1))
RETURN
END

```

5.4DS VARBL1 07/13/73 ED 0 PAGE NO. 2

IDENT VARBL1

PROGRAM LENGTH	00016
ENTRY POINTS	00003
BLOCK NAMES	00024
TITLE	00001
TEMP	00001
CAP	00001
SOJRCE	00001
COND	00001
PCS	00001
KONST	00001
ARRAY	00001
FIXCON	00062
DIMENS	00012
PRNT	00001
XSPACE	00003

EXTERNAL SYMBOLS
0800ICT,
STFSEP

00121 SYMBOLS

```

SUBROUTINE VARBL2
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
RETURN
END

```

5.4DS VARBL2 07/13/73 ED 0 PAGE NO. 2

IDENT VARBL2

PROGRAM LENGTH VARBL2
ENTRY POINTS
BLOCK NAMES
TITLE
TEMP
CAP
SOURCE
COND
PCS
KONST
ARRAY
FIXCON
DIMENS
PRNT
XSPACE
00012
00003
00024
00001
00001
00001
00001
00001
00001
00001
00001
00062
00012
00001
00003

EXTERNAL SYMBOLS
00120 SYMBOLS 08DDICT.

```

SUBROUTINE OUTCAL
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
INLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
IDAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
IDTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
LLOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
LCSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
LARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
LNCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
LJTEST , KTEST , LTEST , MTEST , RTEST ,
LSTEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DU1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), G(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL TSAVE
RETURN
END

```

5.4DS OUTCAL

07/13/73

ED 0

PAGE NO. 2

	IDENT	OUTCAL
PROGRAM LENGTH	00014	
ENTRY POINTS	OUTCAL	00003
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
QBDDICT.		
TSAVE		
00121 SYMROLS		

LOAD
RUN,5,2500

150

MARY E. GEALY

PROGRAM NAMES

1 77744 LINK0	00033	1 77373 EXECTN	00017	1 77355 VARBL1	00016	1 77343 VARBL2	00012
1 77327 OUTCAL	00014	1 77255 BIT	00052	1 75007 IOH.	02246	1 74755 Q7QLODLC	00032
1 74544 QBQERROR	00211	1 74310 ALLOC.	00234	1 74273 REW.	00015	1 73765 UNPAK	00306
1 73713 SLI.	00052	1 73651 QBINOUT4	00042	1 73613 STH.	00036	1 73037 IOB.	00554
1 72657 Q1QREINT	00160	1 72234 IOS.	00423	1 70657 IOP.	01355	1 70524 TSAVE	00133
1 70436 STFSEP	00066	1 67276 CNFRWD	01140	1 66525 INPUIT	00551	1 66341 Q1QSTORE	00164
1 66263 QBQENTRY	00056						

PROGRAM EXTENS.

NONE

LABELED COMMON

1 77720 TITLE	00024	1 77713 TEMP	00005	1 77706 CAP	00005	1 77701 SOURCE	00005
1 77675 COND	00004	1 77667 PCS	00006	1 77666 KONST	00001	1 77665 ARRAY	00001
1 77603 FIXCON	00062	1 77571 DIMENS	00012	1 77564 PRNT	00005	1 77416 XSPACE	00146
1 77412 LOGIC	00004						

NUMBERED COMMON

NONE

ENTRY POINTS

0 77777 SENTRY	1 77747 LINK0	1 66267 QBQENTRY	1 66341 Q3Q10040
1 66263 QBQDICT.	1 66542 INPUIT	1 77376 EXECTN	1 67332 CNFRWD
1 77360 VARBL1	1 70441 STFSEP	1 77346 VARBL2	1 77332 OUTCAL
1 70540 TSAVE	1 70662 IOP.	1 72307 QBQHIST.	1 72553 THEND.
1 72730 Q3Q00040	1 66264 EXIT	1 73042 TSB.	1 73617 STH.
1 73667 QBQINP4	1 73713 SLI.	1 72545 QNSINGL.	1 73772 UNPAK
1 74276 REW.	1 73054 STB.	1 74446 ALLOC.	1 74316 RETURN.
1 74422 BUSY.	1 74431 IRETURN.	1 74546 QBQERROR	1 74755 Q7QLODLC
1 72550 QNDQUBL.	1 72254 IOS.	1 72557 IOR.	1 72234 IOE.
1 75033 IOH.	1 75007 BCQBUF.	1 77255 BIT	1 76623 ELD.
1 77107 .REPCNT.	1 77130 .TSERR.	1 74544 QBERRORN	1 74545 QBNOTRAC
1 74660 QBQERSET	1 74512 ALLOCIN.	1 73721 SLO.	1 73651 QBQOUT4
1 73175 ELB.	1 72657 Q1Q00100	1 72657 Q1Q01100	1 72676 Q1Q02100
1 72702 Q1Q03100	1 72706 Q1Q04100	1 72712 Q1Q05100	1 72662 Q3Q00140
1 72737 Q3Q01040	1 72670 Q3Q01140	1 72776 Q3Q02040	1 72746 Q3Q02140
1 73005 Q3Q03040	1 72754 Q3Q03140	1 73014 Q3Q04040	1 72762 Q3Q04140
1 73023 Q3Q05040	1 72770 Q3Q05140	1 72426 QBQCHAIN	1 71701 ETAB.
1 66427 Q1Q10010	1 66427 Q1Q10020	1 66375 Q1Q10030	1 66450 Q1Q10100
1 66422 Q1Q10120	1 66401 Q1Q10130	1 66456 Q1Q10200	1 66404 Q1Q10210
1 66404 Q1Q10230	1 66450 Q1Q10300	1 66433 Q1Q10310	1 66422 Q1Q10320
1 66443 Q1Q10400	1 66436 Q1Q10410	1 66415 Q1Q10420	1 66415 Q1Q10430
1 66341 Q3Q10140	1 66341 Q3Q10240	1 66341 Q3Q10340	1 66341 Q3Q10440

EXECUTION STARTED AT 1835 -54

95 LOCATIONS AVAILABLE

END OF DATA
BINARY DECK
RUN,1,1000

PROGRAM NAMES

1 70655	PLOTTEMP	07122	1 41101	PLOTPREP	04102	1 40663	PLOTT	00216	1 40641	QBQLOADA	00022
1 40405	ALLOC.	00234	1 40340	ENC.	00045	1 40305	DEC.	00033	1 40073	SINF	00212
1 35625	IOH.	02246	1 35536	QBQRESID	00067	1 35504	Q7QLODLC	00032	1 35273	QBQERROR	00211
1 33716	IOP.	01355	1 33572	NUMBER	00124	1 33032	SYMBOL	00540	1 32305	AXIS	00525
1 32233	SLI.	00052	1 32175	STH.	00036	1 31421	IOB.	00554	1 31354	TSH.	00045
1 31337	REW.	00015	1 31245	BSP.	00072	1 31212	QBQIFIOC	00033	1 27734	PLOT	01256
1 27553	QIQREINT	00160	1 27367	QIQSTORE	00164	1 26744	IOS.	00423	1 26666	QBQENTRY	00056

PROGRAM EXTENS.

NONE

LABELED COMMON

1 64735	TM	03720	1 55075	TP	07640	1 45235	ND	07640	1 45231	MINMAX	00004
1 45207	AXES	00022	1 45203	MISCELL	00004	1 27733	SIZ29	00001			

NUMBERED COMMON

NONE

ENTRY POINTS

0 77777	SENTRY		1 76723	PLOTTEMP		1 26672	QBQENTRY		1 27263	THEND.	
1 27476	QIQ10100		1 27606	QIQ05100		1 27367	Q3Q10040		1 27624	Q3Q00040	
1 26666	QBQDICT.		1 45024	PLOTPREP		1 27737	PLOTS		1 40675	PLOTT	
1 30657	PLOT		1 30135	STOPPLOT		1 31221	QBQIFEOF		1 31250	BSP.	
1 31342	REW.		1 31360	TSH.		1 31424	TSB.		1 32201	STH.	
1 32241	SLO.		1 32233	SLI.		1 27255	QNSINGL.		1 31436	STB.	
1 32316	AXIS		1 33035	SYMBOL		1 33575	NUMBER		1 33721	IOP.	
1 27017	QBQHIST.		1 35275	QBQERROR		1 35504	Q7QLODLC		1 35542	QBQRESID	
1 26744	IOE.		1 27136	QBQCHAIN		1 26764	IOS.		1 27267	IOR.	
1 35651	IOH.		1 37746	.TSERR.		1 35625	BCDBUF.		1 27260	QNDOUBL.	
1 27602	QIQ04100		1 40125	SINF		1 40102	COSF		1 40310	DEC.	
1 40343	ENC.		1 37441	ELD.		1 40543	ALLOC.		1 40413	RETURN.	
1 40517	BUSY.		1 40526	IRETURN.		1 26667	EXIT		1 40641	QBQLOADA	
1 40662	QBQLDCON		1 40657	QBQLODA		1 40607	ALLOCIN.		1 40076	QBQCOSF	
1 40121	QBQSINF		1 37725	.REPCNT.		1 35273	QBERRORN		1 35274	QBNOTRAC	
1 35407	QBQERSET		1 34740	ETAB.		1 31557	ELB.		1 31215	QBQIFIOC	
1 30436	SPACE00		1 27553	QIQ00100		1 27553	QIQ01100		1 27572	QIQ02100	
1 27576	QIQ03100		1 27556	Q3Q00140		1 27633	Q3Q01040		1 27564	Q3Q01140	
1 27672	Q3Q02040		1 27642	Q3Q02140		1 27701	Q3Q03040		1 27650	Q3Q03140	
1 27710	Q3Q04040		1 27656	Q3Q04140		1 27717	Q3Q05040		1 27664	Q3Q05140	
1 27455	QIQ10010		1 27455	QIQ10020		1 27423	QIQ10030		1 27450	QIQ10120	
1 27427	QIQ10130		1 27504	QIQ10200		1 27432	QIQ10210		1 27432	QIQ10230	
1 27476	QIQ10300		1 27461	QIQ10310		1 27450	QIQ10320		1 27471	QIQ10400	
1 27464	QIQ10410		1 27443	QIQ10420		1 27443	QIQ10430		1 27367	Q3Q10140	
1 27367	Q3Q10240		1 27367	Q3Q10340		1 27367	Q3Q10440				

EXECUTION STARTED AT 1836 -06

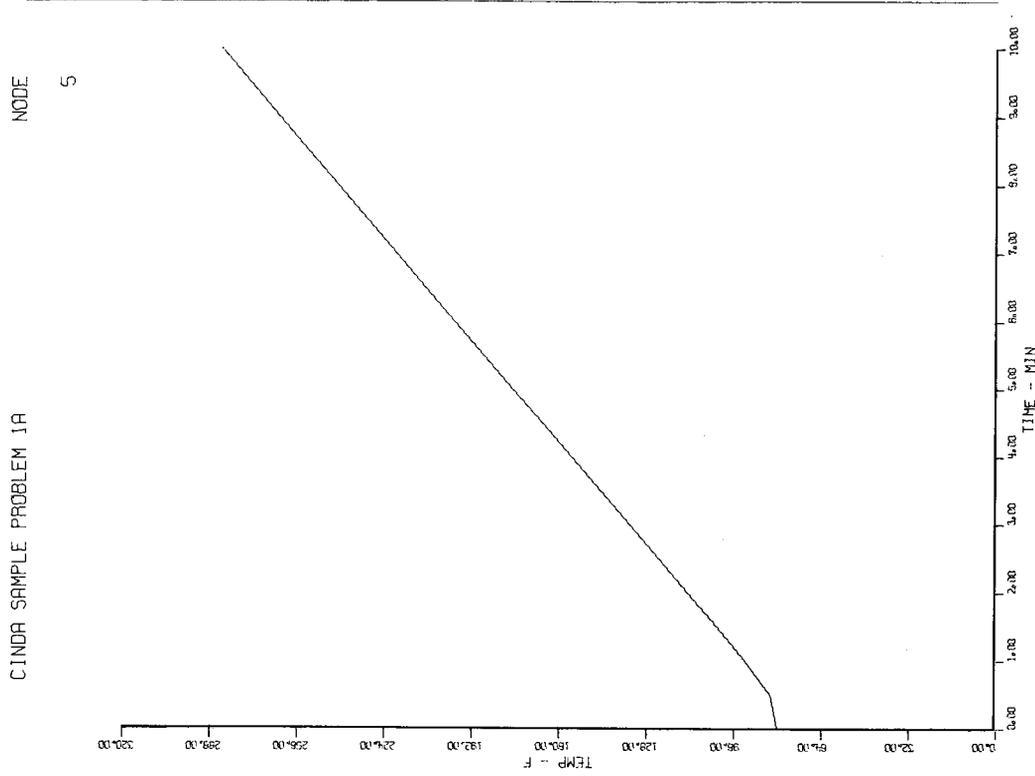
CINDA SAMPLE PROBLEM 1A

X-AXIS LIMITS -- 0.00+000, 1.00+001,

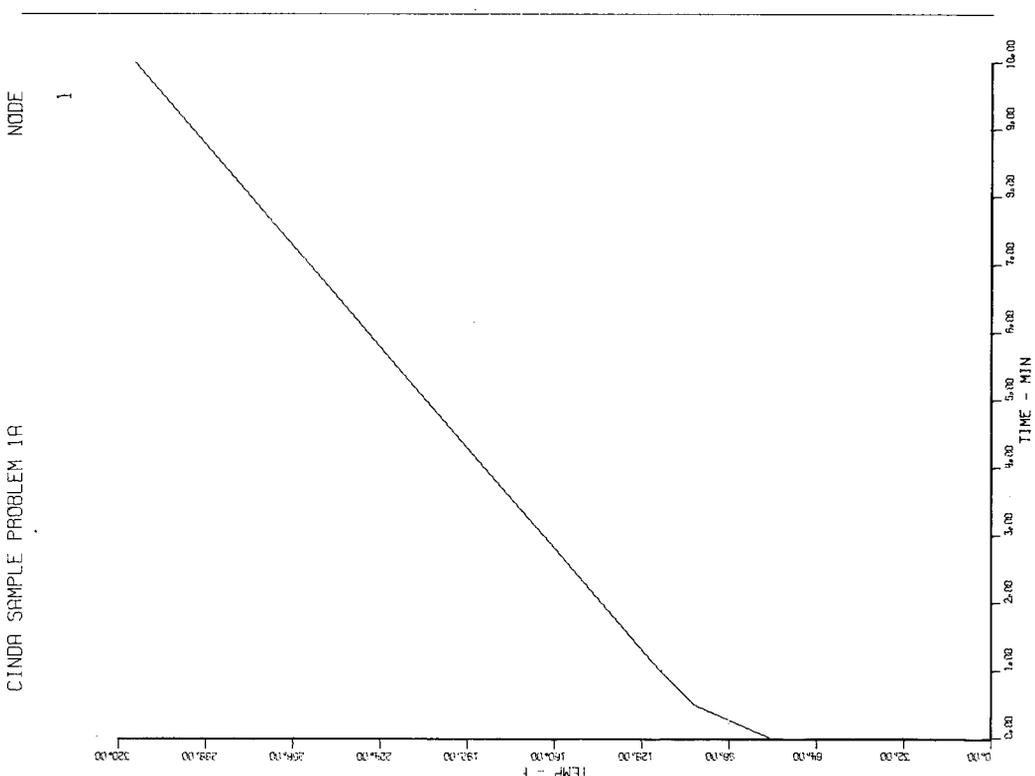
Y-AXIS LIMITS -- 0.00+000, 3.20+002

ALL NODES WILL BE PLOTTED.

JOB MESSAGES
JOB,001712, 300,899RCC
JOB SEQ 01712 WILL PLOT 7 MIN.
LENGTH 4 FT. 06 IN.
RELEASED 9=MT16=(CINDA MASTER),ED=01,RL=01,DATE=052373,RC=999

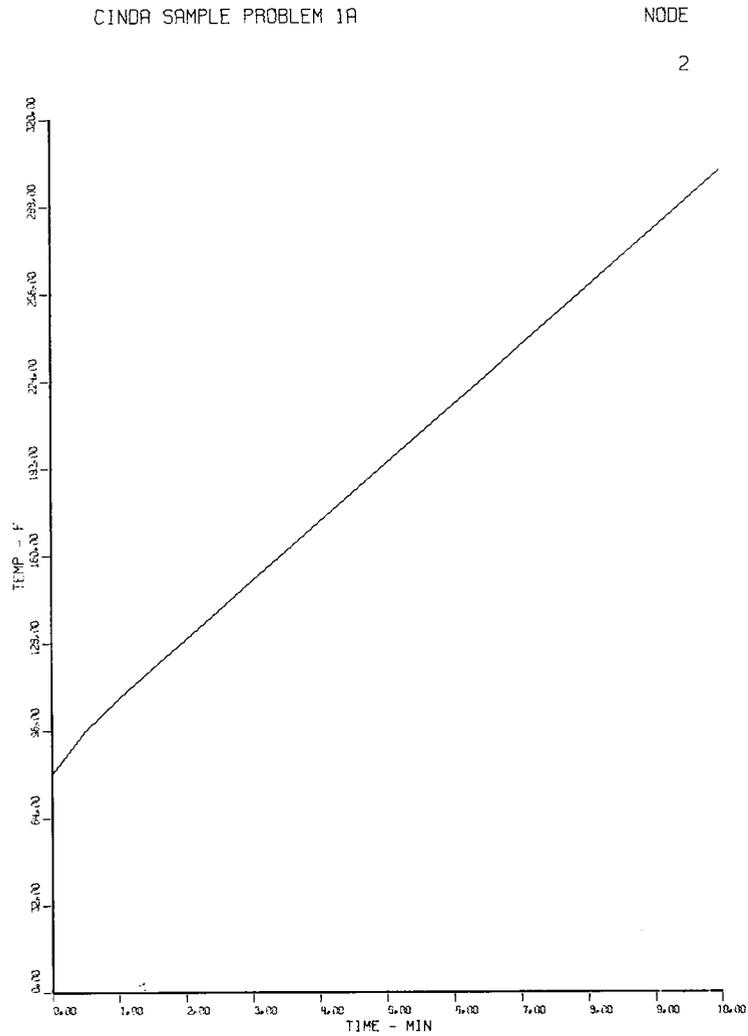


(a)

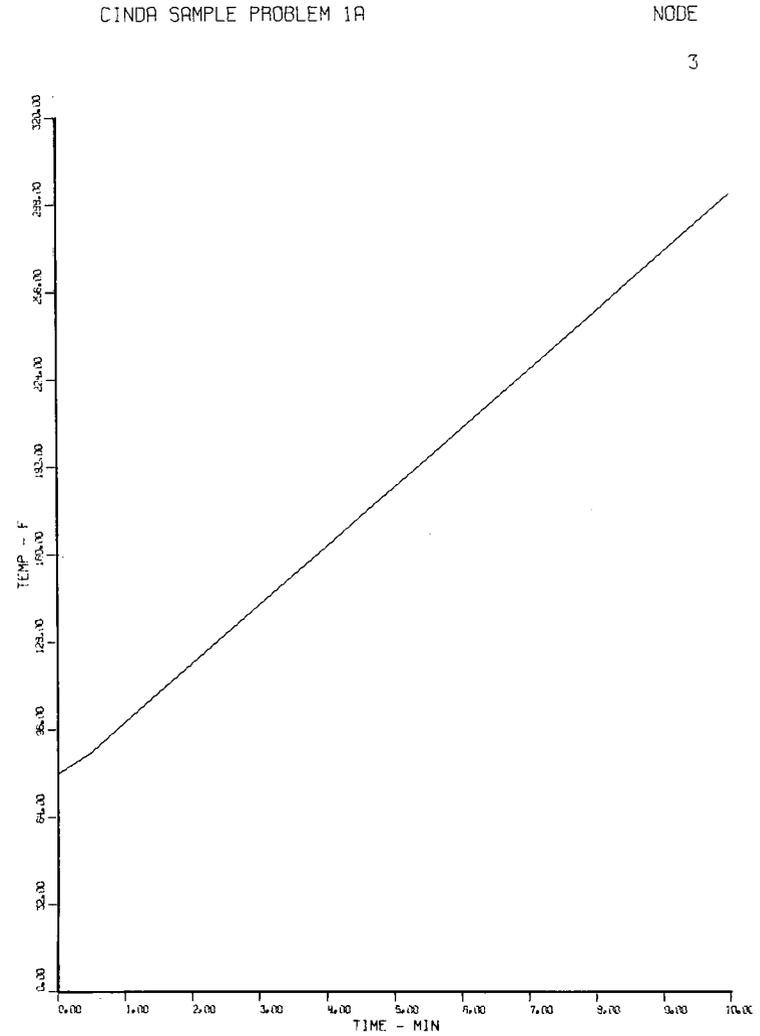


(b)

Fig. A2—Time vs temperature profiles for CINDA Sample Problem 1A. Plots are shown in the order in which they are plotted. Time and temperature numbers will be in the E format, rather than in the F format as shown.

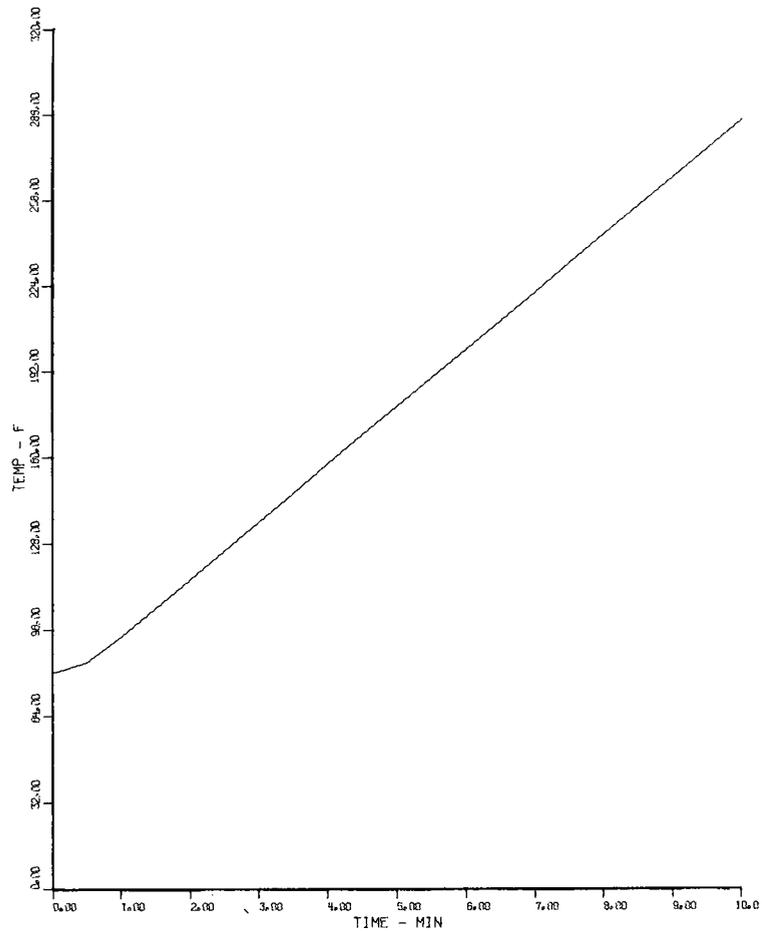


(c)



(d)

Fig. A2 (Cont'd)—Time vs temperature profiles for CINDA Sample Problem 1A. Plots are shown in the order in which they are plotted. Time and temperature numbers will be in the E format, rather than in the F format as shown.



(e)

Fig. A2 (Cont'd)—Time vs temperature profiles for CINDA Sample Problem 1A. Plots are shown in the order in which they are plotted. Time and temperature numbers will be in the E format, rather than in the F format as shown.

Appendix B

SAMPLE PROBLEM 1B

ORIGINAL RUN

Sample problem 1A was linear and can be rigorously solved by means of the Laplace transform. However, the introduction of nonlinearities makes rigorous solutions virtually impossible and makes the use of finite difference techniques mandatory. To demonstrate, apply the following nonlinearities to sample problem 1A and obtain the solution.

1. Both ends of the bar are uninsulated and allowed to radiate to absolute zero. The Stephan-Boltzmann constant is $\sigma = 1.991\text{E-}13 \text{ Btu/min-in.}^2\text{R}^4$, and the emissivity varies linearly with temperature as follows:

$$\begin{aligned}\epsilon &= 0.4 \text{ at } -100^\circ\text{F} \\ \epsilon &= 0.8 \text{ at } 300^\circ\text{F}.\end{aligned}$$

2. The thermal conductivity of the bar varies with temperature as follows:

$$\begin{aligned}k &= 0.15 \text{ at } -100^\circ\text{F (Btu/in.-min-}^\circ\text{F)} \\ k &= 0.25 \text{ at } 100^\circ\text{F} \\ k &= 0.40 \text{ at } 200^\circ\text{F} \\ k &= 0.60 \text{ at } 300^\circ\text{F}.\end{aligned}$$

3. The density remains unchanged but the specific heat varies with temperature as follows:

$$\begin{aligned}C_p &= 0.3 \text{ at } -100^\circ\text{F (Btu/lb-}^\circ\text{F)} \\ &= 0.39 \text{ at } 100^\circ\text{F} \\ &= 0.49 \text{ at } 200^\circ\text{F} \\ &= 0.65 \text{ at } 300^\circ\text{F}.\end{aligned}$$

4. The heating rate is a function of time as follows:

$$\begin{aligned}q &= 3.0 \text{ at } 0 \text{ min (Btu/min)} \\ q &= 4.0 \text{ at } 3 \text{ min} \\ q &= 4.0 \text{ at } 7 \text{ min} \\ q &= 3.0 \text{ at } 10 \text{ min}.\end{aligned}$$

In addition, obtain the rate of heat loss and the integral of the radiation transfer from the unheated end of the bar. The network representation of this problem (shown in Fig. B1) differs only slightly from problem 1A. Now however, the capacitances are a function of temperature. We therefore require multiplying factors such that

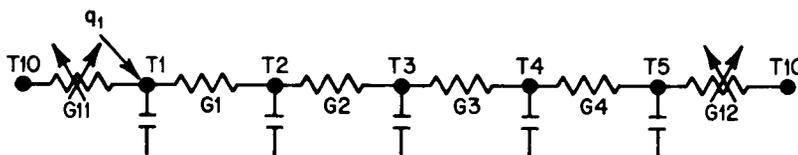


Fig. B1—Network of a nonlinear bar

$$C = \rho V C_p(T), \text{ MF} = \rho V$$

$$\text{MF} = 0.1 \text{ for capacitors 2, 3, and 4}$$

$$\text{MF} = 0.05 \text{ for capacitors 1 and 5.}$$

The conductors are now

$$G = k(T_m)Ac/\ell, \text{ MF} = Ac/\ell, \text{ where } T_m \text{ is the mean of the end T's.}$$

$$\text{MF} = 1.0 \text{ for conductors 1, 2, 3, and 4.}$$

A radiation conductor requires the input value $\sigma\epsilon FA$; however, $FA = 1.0$, hence

$$\text{Grad} = \sigma\epsilon(T)$$

$$\text{MF} = 1.991 \text{ Btu/min}^\circ\text{F.}$$

Also,

$$q = q(T).$$

The capacitors and conductors will be specified with CGS and CGD calls.

A. Original Run

Since this is not a RECALL problem, the first card of the problem data deck will be blank. The rest of the deck may be constructed as follows:

```

1      8  12
↓      ↓  ↓
BCD 3THERMAL SPCS
BCD 9SAMPLE PROBLEM 1B
END
BCD 3NODE DATA
CGS 1,80.,A3,.05,2,80.,A3,.1,3,80.,A3,.1
CGS 4,80.,A3,.1,5,80.,A3,.05
    -10,-460.,0
END
BCD 3CONDUCTOR DATA
CGS 1,1,2,A2,1.,2,2,3,A2,1.,3,3,4,A2,1.,4,4,5,A2,1.
CGS -11,1,10,A1,-1.991E-13,-12,5,10,A1,-1.991E-13
END
BCD 3CONSTANTS DATA
    TIMEND,10.,OUTPUT,.5,CSGFAC,2.,4,0,5,0,6,STOR1,7,STOR2
BCD 3ARRAY DATA
    1,-100.,.4,300.,0.8,END $ EPSILON VS T
    2,-100.,.15,100.,.25,200.,.4,300.,.6,END $ K VS T
    3,-100.,.3,100.,.39,200.,.49,300.,.65,END $ CP VS T
    4,0.,3.,3.,4.,7.,4.,10.,3.,END $ Q VS TIME
    -5,QRATE,QTOTAL,END $A LABEL ARRAY
END
F DIMENSION X( 100)
BCD 3EXECUTION
F NDIM = 100
F NTH = 0
    STOREP (K6)
    CNFRWD
F IDCNT =IDCNT + 1
    STOREP(K7)
END
BCD 3VARIABLES 1
    DIDEG1(TIMEM,A4,Q1) $APPLY HEATING RATE
END
BCD 3VARIABLES 2
    RDTNQS(T10,T5,G12,K4) $OBTAIN HEAT FLOW RATE
    QINTEG(K4,DTIMEU,K5) $INTEGRATE SAME
END
BCD 3OUTPUT CALLS
    TPRINT
    PRINTL (A5,K4,K5)
END
BCD 3END OF DATA

```

MARY E. GEALY

This problem will be stored twice on tape 22. The original data will be stored under the I.D. name, STØR1, and the number, 0 (IDCNT). The final values will be identified as STØR2, 1 because IDCNT was incremented. (It actually would not have been necessary to use IDCNT in this case, since neither call to STØREP was in a loop. The second call *could* have been uniquely identified as STØR2,0.) The binary constructed subroutines (processor) will be stored on tape 40. See Section VII for the proper deck setup and job request form.

The above problem data deck processed by the CDC-3800 version of CINDA produces output given in the following pages (original run).

SEQUENCE 04645 STARTED PRINTING 07/30/73 AT 191547 ON LP01
DRUM SCOPE 2.1 COMPUTER TWO. MAX. DEMAND IS 570008 VERSION 004 10/31/72
SEQUENCE NUMBER 004645 STARTED AT TIME 191209 DATED 07/30/73
JOB(5),40800800,899RCC,10
EQUIP,09=(CINDA MASTER,1,1,999),RO,HI,DA
EQUIP,22=**,WO,HI,DA
EQUIP,40=**,RW,HI,DA
BINARY DECK
BANK,(0),/4/
LOAD,09
RUN,1,1000

PROGRAM NAMES

1 77672 SEARCH	00105	1 77603 STFFB	00067	1 76026 GENLNK	00540	1 74614 SPLIT	01026
1 74272 SKIP	00322	1 72717 CODERD	01353	1 71712 PSUEDO	00661	1 71627 ORMIN	00063
1 71417 PACK43	00210	1 71345 BIT	00052	1 71051 PRESUB	00274	1 70731 WRTBLK	00120
1 70673 WRTSCOPE	00036	1 65267 CINDA4	03404	1 62747 BLKCRD	01170	0 71134 DATARD	06643
1 62141 PREPRO	00605	0 60445 INITIAL	10467	1 57571 FINAL	02350	1 57335 ALLOC.	00234
0 56177 IOH.	02246	1 57303 Q7QLODLC	00032	0 54622 IOP.	01355	1 57072 QBQERROR	00211
1 57005 BF1.	00065	1 56740 ENC.	00045	1 56705 DEC.	00033	0 54570 QBQIFUNI	00032
0 54544 EFT.	00024	0 54472 SLI.	00052	0 54430 Q8INOUT4	00042	0 53654 IOB.	00554
0 53562 BSP.	00072	0 53527 QBQIFIOC	00033	0 53471 STH.	00036	0 53424 TSH.	00045
0 53407 REW.	00015	0 52764 IOS.	00423	0 52604 Q1QREINT	00160	0 52420 Q1QSTORE	00164
0 52342 QBQENTRY	00056						

PROGRAM EXTENS.
NONE

LABELED COMMON

1 76566 CRDBLK	01015	1 76014 TAPE	00012	1 75775 DATA	00017	1 75657 SUBLST	00116
1 75654 QLOGIC	00003	1 75642 PLOGIC	00012	1 72573 LOGIC	00124	1 64613 NARRAY	00454
1 64137 DIMARY	00454	1 62746 WJS	00001				

NUMBERED COMMON

0 25555 4	17500	1 00001 1	17500	1 17501 3	17500	1 37201 2	17500
-----------	-------	-----------	-------	-----------	-------	-----------	-------

ENTRY POINTS

0 77777 SENTRY	1 77675 SEARCH	0 52342 QBQDICT.	1 77607 STFFB
1 76214 GENLNK	0 52420 Q3Q10040	0 52655 Q3Q00040	0 53303 THEND.
1 63176 BLKCRD	0 53412 REW.	0 53430 TSH.	0 53475 STH.
0 53275 QNSINGL.	1 74656 SPLIT	1 74275 SKIP	0 53536 QBQIFEOF
0 53565 BSP.	0 53657 TSB.	0 53671 STB.	0 54446 QBQINP4
1 73077 CODERD	0 72636 DATARD	0 54500 SLO.	0 54472 SLI.
1 71760 PSUEDO	1 71422 PACK43	1 71632 ORMIN	0 54547 EFT.
1 71345 BIT	1 71110 PRESUB	1 66371 CINDA4	1 70701 WRTSCOPE
0 54573 QBQIFUNI	1 70764 WRTBLK	1 56710 DEC.	1 56743 ENC.
1 57020 BFO.	1 62214 PREPRO	0 52346 QBQENTRY	0 67407 INITIAL
1 60643 FINAL	1 57074 QBQERROR	0 52343 EXIT	0 54430 QBQOUT4
0 54625 IOP.	0 53037 QBQHIST.	1 57303 Q7QLODLC	0 53004 IOS.
0 53307 IOR.	0 56223 IOH.	0 60320 .TSERR.	0 56177 BCDBUF.
0 52764 IOE.	0 53156 QBQCHAIN	0 53300 QNDOUBL.	0 55644 ETAB.
1 57473 ALLOC.	1 57343 RETURN.	1 57447 BUSY.	1 57456 IRETURN.
1 57537 ALLOCIN.	0 60013 ELD.	0 60277 .REPCNT.	1 57072 QBERRORN
1 57073 QBNOTRAC	1 57206 QBQERSET	1 57010 BF1.	0 54012 ELB.
0 53532 QBQIFIOC	0 52604 Q1Q00100	0 52604 Q1Q01100	0 52623 Q1Q02100
0 52627 Q1Q03100	0 52633 Q1Q04100	0 52637 Q1Q05100	0 52607 Q3Q00140
0 52664 Q3Q01040	0 52615 Q3Q01140	0 52723 Q3Q02040	0 52673 Q3Q02140
0 52732 Q3Q03040	0 52701 Q3Q03140	0 52741 Q3Q04040	0 52707 Q3Q04140
0 52750 Q3Q05040	0 52715 Q3Q05140	0 52506 Q1Q10010	0 52506 Q1Q10020
0 52454 Q1Q10030	0 52527 Q1Q10100	0 52501 Q1Q10120	0 52460 Q1Q10130
0 52535 Q1Q10200	0 52463 Q1Q10210	0 52463 Q1Q10230	0 52527 Q1Q10300
0 52512 Q1Q10310	0 52501 Q1Q10320	0 52522 Q1Q10400	0 52515 Q1Q10410
0 52474 Q1Q10420	0 52474 Q1Q10430	0 52420 Q3Q10140	0 52420 Q3Q10240
0 52420 Q3Q10340	0 52420 Q3Q10440		

EXECUTION STARTED AT 1915 -43

```

BCD 3THERMAL SPCS
BCD 9SAMPLE PROBLEM 1B
END
BCD 3NODE DATA
CGS 1,80.,A3,.05,2,80.,A3,.1,3,80.,A3,.1
CGS 4,80.,A3,.1,5,80.,A3,.05
    -10,-460.,0
END

```

```

RELATIVE NODE NUMBE S          ACTUAL NODE NUMBERS
    1 THRU      6          1      2      3      4      5      10
BCD 3CONDUCTOR DATA
CGS 1,1,2,A2,1.,2,2 3,A2,1.,3,3,4,A2,1.,4,4,5,A2,1.
CGS -11,1,10,A1,-1.991E-13,-12,5,10,A1,-1.991E-13
END

```

```

RELATIVE CONDUCTOR NUMBERS          ACTUAL CONDUCTOR NUMBERS
    1 THRU      6          1      2      3      4      11      12
BCD 3CONSTANTS DATA
TIMEND,10.,OUTPUT,,5,CSGFAC,2.,4,0,5,0,6,STOR1,7,STOR2
END
BCD 3ARRAY DATA
1,-100.,4,300.,0.8,END          $EPSILON VS T
2,-100.,15,100.,25,200.,4,300.,.6,END          $K VS T
3,-100.,3,100.,39,200.,49,300.,.65,END          $CP VS T
4,0.,3.,3.,4.,7.,4.,10.,3.,END          $Q VS TIME
-5,QRATE,QTOTAL,END          $A LABEL ARRAY

```

```

END
DIMENSION X( 100)          F
BCD 3EXECUTION
NDIM = 100          F
NTH = 0          F
STOREP(K6)
CNFRWD
IDCNT = IDCNT + 1          F
STOREP(K7)
END
BCD 3VARIABLES 1
VARCSM(T 1,C 1,A3,.05)
VARCSM(T 2,C 2,A3,.1)
VARCSM(T 3,C 3,A3,.1)
VARCSM(T 4,C 4,A3,.1)
VARCSM(T 5,C 5,A3,.05)
VARGSM(G 1,T 1,T 2,A2,1.)
VARGSM(G 2,T 2,T 3,A2,1.)
VARGSM(G 3,T 3,T 4,A2,1.)
VARGSM(G 4,T 4,T 5,A2,1.)
VARGSM(G 11,T 1,T 10,A1,-1.991E-13)
VARGSM(G 12,T 5,T 10,A1,-1.991E-13)
ENDV1
D1DEG1 (TIMEM,A4,Q1)          $APPLY HEATING RATE
END
BCD 3VARIABLES 2
RDTNQS(T10,T5,G12,K4)          $OBTAIN HEAT FLOW RATE
QINTEG(K4,DTIMEU,K5)          $INTEGRATE SAME
END
BCD 3OUTPUT CALLS

```


LIBRARY,09,LCINDA
FTN,I=14,L,X=40

```

PROGRAM      LINKO
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN ,      DTIMEU,      TIMEND,      CSGFAC,
1DAMPD ,      DTMPCA,      OPEITR,      DTIMEH,      DAMPA ,
1DTMPCC,      ATMPCA,      BACKUP,      TIMEO ,      TIMEM ,
1LOOPCT,      DTIMEL,      DTIMEI,      CSGMAX,      CSGRAL,
1CSGRCL,      DRLXCA,      DRLXCC,      LINECT,      PAGECT,
1ARLXCC,      LSPCS ,      ENGBAL,      BALENG,      NOCOPY,
1NCSGM ,      NDTMPC,      NARLXC,      NATMPC,      ITEST ,
1JTEST ,      KTEST ,      LTEST ,      MTEST ,      RTEST ,
1TEST ,      TTEST ,      UTEST ,      VTEST ,      LAXFAC
1,      IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
COMMON /LOGIC/ LNODE, LCOND, LCONST, LARRAY
LOGICAL LNODE, LCOND, LCONST, LARRAY
DIMENSION T( 6),C( 6),Q( 5),G( 6),K( 4),A( 34),
1CSEQ( 6),X( 100), NT( 6)
LNODE = .TRUE.
LCOND = .TRUE.
LCONST = .TRUE.
LARRAY = .TRUE.
1 CALL INPUTT
CALL EXECTN
GO TO 1
END

```

	IDENT	LINK0
PROGRAM LENGTH	00033	
ENTRY POINTS	LINK0	00003
BLOCK NAMES		
TITLE	00024	
TEMP	00006	
CAP	00006	
SOURCE	00005	
COND	00006	
PCS	00006	
KONST	00004	
ARRAY	00042	
FIXCON	00062	
DIMENS	00012	
PRNT	00006	
XSPACE	00146	
LOGIC	00004	
EXTERNAL SYMBOLS		
QBQENTRY		
Q3Q100*0		
Q8QDICT.		
INPUTT		
EXECTN		
00126 SYMBOLS		

```

SUBROUTINE EXECTN
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NGG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
NDIM = 100
NTH = 0
CALL STOREP(K(3))
CALL CNFRWD
IDCNT = IDCNT + 1
CALL STOREP(K(4))
RETURN
END

```

F
F
F

	IDENT	EXECTN
PROGRAM LENGTH	00027	
ENTRY POINTS	00003	
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
QB0DICT.		
STOREP		
CNFRWD		
00122 SYMBOLS		

```

SUBROUTINE VARBL1
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEH, DAMPA , CSGFAC,
INLOOP , OPEITR, TIMEO, TIMEM ,
IDAMPD , DTMPCA, BACKUP, TIMEO, TIMEM ,
IDTMPCC , ATMPCA, CSGMIN, OUTPUT, ARLXCA,
ILOOPC , DTIMEL, CSGMX, CSGRAL,
ICSGRCL, DRLXCC, DRLXCC, LINECT, PAGECT,
IARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
INCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
IJTEST , KIEST , LTEST , UTEST , RTEST ,
I , IDCNT , VTEST , LAXFAC
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRINT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1), XK(1),X(1)
1, CSFQ(1) NT(1)
CALL VARCSM(T(1),C(1),A(15),.05)
CALL VARCSM(T(2),C(2),A(15),.1)
CALL VARCSM(T(3),C(3),A(15),.1)
CALL VARCSM(T(4),C(4),A(15),.1)
CALL VARCSM(T(5),C(5),A(15),.05)
CALL VARGSM(G(1),T(1),T(2),A(6),1.)
CALL VARGSM(G(2),T(2),T(3),A(6),1.)
CALL VARGSM(G(3),T(3),T(4),A(6),1.)
CALL VARGSM(G(4),T(4),T(5),A(6),1.)
CALL VARGSM(G(5),T(1),T(6),A(1),-1.991E-13)
CALL VARGSM(G(6),T(5),T(6),A(1),-1.991E-13)
CALL DIDEGL(TIMEM ,A(24),Q(1))
RETURN
END
    
```

	IDENT	VARBL1
PROGRAM LENGTH	00110	
ENTRY POINTS	00003	VARBL1
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
Q8QDICT.		
VARCSM		
VARGSM		
D1DEG1		
00124 SYMBOLS		

```

SUBROUTINE VARBL2
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC,
1NLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEQ , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL ROTNQS(T(6),T(5),G(6),K(1))
CALL QINTEG(K(1),DTIMEU,K(2))
RETURN
END

```

	IDENT	VARBL2
PROGRAM LENGTH	00022	
ENTRY POINTS	00003	VARBL2
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
QBQDICT.		
RDTNQS		
QINTEG		
00122 SYMBOLS		

```

SUBROUTINE OUTCAL
COMMON /TITLE/ H
COMMON /TEMP/ T
COMMON /CAP/ C
COMMON /SOURCE/ Q
COMMON /COND/ G
COMMON /PCS/ CSEQ
COMMON /KONST/ K
COMMON /ARRAY/ A
COMMON /FIXCON/ TIMEN , DTIMEU, TIMEND, CSGFAC
INLOOP , DTMPCA, OPEITR, DTIMEH, DAMPA ,
1DAMPD , ATMPCA, BACKUP, TIMEO , TIMEM ,
1DTMPCC, ATMPCC, CSGMIN, OUTPUT, ARLXCA,
1LOOPCT, DTIMEL, DTIMEI, CSGMAX, CSGRAL,
1CSGRCL, DRLXCA, DRLXCC, LINECT, PAGECT,
1ARLXCC, LSPCS , ENGBAL, BALENG, NOCOPY,
1NCSGM , NDTMPC, NARLXC, NATMPC, ITEST ,
1JTEST , KTEST , LTEST , MTEST , RTEST ,
1STEST , TTEST , UTEST , VTEST , LAXFAC
1, IDCNT
COMMON /DIMENS/ NNT,NND,NNC,NRG,NNG,NCON,NARY,LSEQ,DUM1,NUM2
DIMENSION H(20)
COMMON /PRNT/ NT
COMMON /XSPACE/ NDIM, NTH, X
EQUIVALENCE (K(1),XK(1))
DIMENSION T(1), C(1), Q(1), G(1), K(1), A(1) ,XK(1),X(1)
1, CSEQ(1) , NT(1)
CALL TPRINT
CALL PRINTL(A(33),K(1),K(2))
RETURN
END

```

	IDENT	OUTCAL
PROGRAM LENGTH	00020	
ENTRY POINTS	00003	
BLOCK NAMES		
TITLE	00024	
TEMP	00001	
CAP	00001	
SOURCE	00001	
COND	00001	
PCS	00001	
KONST	00001	
ARRAY	00001	
FIXCON	00062	
DIMENS	00012	
PRNT	00001	
XSPACE	00003	
EXTERNAL SYMBOLS		
QBDDICT.		
TPRINT		
PRINTL		

00122 SYMBOLS

LOAD,40
RUN,5,2500

PROGRAM NAMES															
1	77744	LINK0	00033	1	77312	EXECTN	00027	1	77202	VARBL1	00110	1	77160	VARBL2	00022
1	77140	OUTCAL	00020	1	77103	SKPLIN	00035	1	77061	QBQLOADA	00022	1	76601	LAGRAN	00260
1	76527	BIT	00052	1	74261	IOH.	02246	1	74227	QTQLODLC	00032	1	74016	QBQERROR	00211
1	73562	ALLOC.	00234	1	73365	WRTL08	00175	1	73244	STNDRD	00121	1	73156	TOPLIN	00066
1	73131	QBQXMODF	00025	1	73043	D1D1WM	00066	1	72535	UNPAK	00306	1	72520	REW.	00015
1	72446	SLI.	00052	1	72404	QBINOOUT4	00042	1	72346	STH.	00036	1	71572	IOB.	00554
1	71412	QIQREINT	00160	1	70767	IOS.	00423	1	67412	IOP.	01355	1	67176	PRINT	00214
1	67021	TPRINT	00155	1	66741	QINTEG	00060	1	66643	RD TNQS	00076	1	66415	D1DEG1	00226
1	66277	VARGSM	00116	1	66221	VARCSM	00056	1	65061	CNFRWD	01140	1	63373	STOREP	01466
1	62622	INPUTT	00551	1	62436	QIQSTORE	00164	1	62360	QBQENTRY	00056				

PROGRAM EXTENS.
NONE

LABELED COMMON															
1	77720	TITLE	00024	1	77712	TEMP	00006	1	77704	CAP	00006	1	77677	SOURCE	00005
1	77671	CONO	00006	1	77663	PCS	00006	1	77657	KONST	00004	1	77615	ARRAY	00042
1	77533	FIXCON	00062	1	77521	DIMENS	00012	1	77513	PRNT	00006	1	77345	XSPACE	00146
1	77341	LOGIC	00004												

NUMBERED COMMON
NONE

ENTRY POINTS															
0	77777	SENTRY		1	77747	LINK0		1	62364	QBQENTRY		1	62436	Q3Q10040	
1	62360	QBQDICT.		1	62637	INPUTT		1	77315	EXECTN		1	64216	STOREP	
1	65115	CNFRWD		1	77205	VARBL1		1	66224	VARCSM		1	66302	VARGSM	
1	66431	D1DEG1		1	77163	VARBL2		1	66646	RD TNQS		1	66744	QINTEG	
1	77143	OUTCAL		1	67042	TPRINT		1	67315	PRINTL		1	67415	IOP.	
1	71042	QBQHIST.		1	71306	THFND.		1	71463	Q3Q00040		1	62361	EXIT	
1	71575	TSB.		1	72352	STH.		1	72422	QBQINP4		1	72446	SLI.	
1	71300	QNSINGL.		1	72523	REW.		1	71607	STB.		1	72404	QBQOUT4	
1	72454	SLO.		1	72542	UNPAK		1	73046	D1D1WM		1	73140	XMODF	
1	73215	TOPLIN		1	73325	STNDRD		1	73411	WRTL08		1	73720	ALLOC.	
1	73570	RETURN.		1	73674	BUSY.		1	73703	IRETURN.		1	74020	QBQERROR	
1	74227	QTQLODLC		1	71303	QND0UBL.		1	71007	IOS.		1	71312	IOR.	
1	70767	IOE.		1	74305	IOH.		1	74261	BCDBUF.		1	76527	BIT	
1	76614	LAGRAN		1	77061	QBQLOADA		1	77112	SKPLIN		1	77102	QBQLDCON	
1	77077	QBQLODA		1	76075	ELD.		1	76361	.REPCNT.		1	76402	.TSERR.	
1	74016	QBERRORN		1	74017	QBNO TRAC		1	74132	QBQERSET		1	73764	ALLOCIN.	
1	73134	QBQXMODF		1	71730	ELB.		1	71412	Q1Q00100		1	71412	Q1Q01100	
1	71431	Q1Q02100		1	71435	Q1Q03100		1	71441	Q1Q04100		1	71445	Q1Q05100	
1	71415	Q3Q00140		1	71472	Q3Q01040		1	71423	Q3Q01140		1	71531	Q3Q02040	
1	71501	Q3Q02140		1	71540	Q3Q03040		1	71507	Q3Q03140		1	71547	Q3Q04040	
1	71515	Q3Q04140		1	71556	Q3Q05040		1	71523	Q3Q05140		1	71161	QBQCHAIN	
1	70434	ETAB.		1	67306	PRINT		1	62524	Q1Q10010		1	62524	Q1Q10020	
1	62472	Q1Q10030		1	62545	Q1Q10100		1	62517	Q1Q10120		1	62476	Q1Q10130	
1	62553	Q1Q10200		1	62501	Q1Q10210		1	62501	Q1Q10230		1	62545	Q1Q10300	
1	62530	Q1Q10310		1	62517	Q1Q10320		1	62540	Q1Q10400		1	62533	Q1Q10410	
1	62512	Q1Q10420		1	62512	Q1Q10430		1	62436	Q3Q10140		1	62436	Q3Q10240	
1	62436	Q3Q10340		1	62436	Q3Q10440									

EXECUTION STARTED AT 1919 -55

SAMPLE PROBLEM 1B

```

* * * *
TIME 00.00000+000 DTIMEU 00.00000+000 CSGMIN( 1) 79.36899-003 DTMPCC( 1) 00.00000+000 ARLXCC( 0) 00.00000+000
T 1= 80.00000+000 T 2= 80.00000+000 T 3= 80.00000+000 T 4= 80.00000+000 T 5= 80.00000+000 T 10=-46.00000+001

GRATE 00.00000+000 QTOTAL 00.00000+000

* * * *
TIME 50.00000-002 DTIMEU 25.27368-003 CSGMIN( 2) 78.26181-003 DTMPCC( 1) 68.11626-002 ARLXCC( 0) 00.00000+000
T 1= 10.50446+001 T 2= 94.49641+000 T 3= 87.61895+000 T 4= 83.87774+000 T 5= 82.68624+000 T 10=-46.00000+001

GRATE 10.05630-003 QTOTAL 49.40046-004

* * * *
TIME 10.00000-001 DTIMEU 29.28453-003 CSGMIN( 2) 76.30274-003 DTMPCC( 1) 64.18434-002 ARLXCC( 0) 00.00000+000
T 1= 11.67556+001 T 2= 10.59682+001 T 3= 97.99293+000 T 4= 93.20244+000 T 5= 91.58636+000 T 10=-46.00000+001

GRATE 10.89220-003 QTOTAL 10.17759-003

* * * *
TIME 15.00000-001 DTIMEU 17.76855-003 CSGMIN( 2) 73.72196-003 DTMPCC( 4) 38.60775-002 ARLXCC( 0) 00.00000+000
T 1= 12.75381+001 T 2= 11.68016+001 T 3= 10.87716+001 T 4= 10.37800+001 T 5= 10.20570+001 T 10=-46.00000+001

GRATE 11.95506-003 QTOTAL 15.89831-003

* * * *
TIME 20.00000-001 DTIMEU 25.13348-003 CSGMIN( 2) 71.49668-003 DTMPCC( 5) 56.04585-002 ARLXCC( 0) 00.00000+000
T 1= 13.83390+001 T 2= 12.76921+001 T 3= 11.97355+001 T 4= 11.47887+001 T 5= 11.30795+001 T 10=-46.00000+001

GRATE 13.15377-003 QTOTAL 22.18786-003

* * * *
TIME 25.00000-001 DTIMEU 31.77283-003 CSGMIN( 2) 69.46924-003 DTMPCC( 5) 72.10139-002 ARLXCC( 0) 00.00000+000
T 1= 14.93133+001 T 2= 13.87631+001 T 3= 13.09002+001 T 4= 12.60208+001 T 5= 12.43351+001 T 10=-46.00000+001

GRATE 14.47555-003 QTOTAL 29.11089-003

```

SAMPLE PROBLEM 1B

* * * *
TIME 30.00000-001 DTIMEU 21.70966-003 CSGMIN(2) 67.55799-003 DTMPCC(5) 50.03609-002 ARLXCC(0) 00.00000+000
T 1= 16.04687+001 T 2= 15.00197+001 T 3= 14.22506+001 T 4= 13.74381+001 T 5= 13.57756+001 T 10=-46.00000+001
QRATE 15.93544-003 QTOTAL 36.72968-003

* * * *
TIME 35.00000-001 DTIMEU 27.62055-003 CSGMIN(2) 65.91918-003 DTMPCC(5) 63.63066-002 ARLXCC(0) 00.00000+000
T 1= 17.11316+001 T 2= 16.11112+001 T 3= 15.36148+001 T 4= 14.89520+001 T 5= 14.73360+001 T 10=-46.00000+001
QRATE 17.51821-003 QTOTAL 45.11098-003

* * * *
TIME 40.00000-001 DTIMEU 17.45065-003 CSGMIN(2) 64.42989-003 DTMPCC(5) 39.18180-002 ARLXCC(0) 00.00000+000
T 1= 18.14217+001 T 2= 17.18384+001 T 3= 16.46896+001 T 4= 16.02504+001 T 5= 15.87101+001 T 10=-46.00000+001
QRATE 19.20685-003 QTOTAL 54.31173-003

* * * *
TIME 45.00000-001 DTIMEU 22.37952-003 CSGMIN(2) 63.13855-003 DTMPCC(5) 48.92322-002 ARLXCC(0) 00.00000+000
T 1= 19.15048+001 T 2= 18.23175+001 T 3= 17.54867+001 T 4= 17.12540+001 T 5= 16.97841+001 T 10=-46.00000+001
QRATE 20.96310-003 QTOTAL 64.37390-003

* * * *
TIME 50.00000-001 DTIMEU 26.79780-003 CSGMIN(2) 61.97342-003 DTMPCC(5) 57.10075-002 ARLXCC(0) 00.00000+000
T 1= 20.13995+001 T 2= 19.25710+001 T 3= 18.60253+001 T 4= 18.19766+001 T 5= 18.05691+001 T 10=-46.00000+001
QRATE 22.79516-003 QTOTAL 75.33565-003

* * * *
TIME 55.00000-001 DTIMEU 16.49958-003 CSGMIN(2) 60.81596-003 DTMPCC(5) 34.29108-002 ARLXCC(0) 00.00000+000
T 1= 21.10438+001 T 2= 20.26093+001 T 3= 19.63220+001 T 4= 19.24379+001 T 5= 19.10855+001 T 10=-46.00000+001
QRATE 24.71381-003 QTOTAL 87.23433-003

183

NRL REPORT 7656

SAMPLE PROBLEM 1B

* * * *
 TIME 60.00000-001 DTIMEU 20.30452-003 CSGMIN(2) 59.90380-003 DTMPCC(5) 41.19123-002 ARLXCC(0) 00.00000+000
 T 1= 22.04167+001 T 2= 21.23607+001 T 3= 20.63704+001 T 4= 20.26541+001 T 5= 20.13542+001 T 10=-46.00000+001
 QRATE 26.69869-003 QTOTAL 10.01085-002

* * * *
 TIME 65.00000-001 DTIMEU 23.71529-003 CSGMIN(2) 59.06441-003 DTMPCC(5) 46.48373-002 ARLXCC(0) 00.00000+000
 T 1= 22.95402+001 T 2= 22.18220+001 T 3= 21.61004+001 T 4= 21.25578+001 T 5= 21.13170+001 T 10=-46.00000+001
 QRATE 28.74491-003 QTOTAL 11.39935-002

* * * *
 TIME 70.00000-001 DTIMEU 26.76422-003 CSGMIN(2) 58.31336-003 DTMPCC(5) 50.80665-002 ARLXCC(0) 00.00000+000
 T 1= 23.84313+001 T 2= 23.10159+001 T 3= 22.55327+001 T 4= 22.21426+001 T 5= 22.09534+001 T 10=-46.00000+001
 QRATE 30.84195-003 QTOTAL 12.89163-002

* * * *
 TIME 75.00000-001 DTIMEU 15.76236-003 CSGMIN(2) 57.64421-003 DTMPCC(5) 28.50700-002 ARLXCC(0) 00.00000+000
 T 1= 24.65868+001 T 2= 23.96903+001 T 3= 23.45586+001 T 4= 23.13714+001 T 5= 23.02475+001 T 10=-46.00000+001
 QRATE 32.99067-003 QTOTAL 14.49003-002

* * * *
 TIME 80.00000-001 DTIMEU 18.23648-003 CSGMIN(2) 57.08125-003 DTMPCC(5) 30.76391-002 ARLXCC(0) 00.00000+000
 T 1= 25.41073+001 T 2= 24.77189+001 T 3= 24.29722+001 T 4= 24.00252+001 T 5= 23.89818+001 T 10=-46.00000+001
 QRATE 35.10650-003 QTOTAL 16.19507-002

* * * *
 TIME 85.00000-001 DTIMEU 20.38997-003 CSGMIN(2) 56.58755-003 DTMPCC(5) 32.02967-002 ARLXCC(0) 00.00000+000
 T 1= 26.11252+001 T 2= 25.52037+001 T 3= 25.08111+001 T 4= 24.80852+001 T 5= 24.71160+001 T 10=-46.00000+001
 QRATE 37.17370-003 QTOTAL 18.00475-002

184

MARY E. GEALY

SAMPLE PROBLEM 18

* * * *

TIME 90.00000-001 DTIMEU 22.28256-003 CSGMIN(2) 56.15228-003 DTMPCC(5) 32.58615-002 ARLXCC(0) 00.00000+000

T 1= 26.76715+001 T 2= 26.21810+001 T 3= 25.81131+001 T 4= 25.55891+001 T 5= 25.46874+001 T 10=-46.00000+001

QRATE 39.18459-003 QTOTAL 19.91642-002

* * * *

TIME 95.00000-001 DTIMEU 23.95381-003 CSGMIN(2) 55.76696-003 DTMPCC(5) 32.59281-002 ARLXCC(0) 00.00000+000

T 1= 27.37685+001 T 2= 26.86785+001 T 3= 26.49106+001 T 4= 26.25722+001 T 5= 26.17324+001 T 10=-46.00000+001

QRATE 41.13322-003 QTOTAL 21.92709-002

* * * *

TIME 10.00000+000 DTIMEU 25.43501-003 CSGMIN(2) 55.42483-003 DTMPCC(5) 32.16639-002 ARLXCC(0) 00.00000+000

T 1= 27.94358+001 T 2= 27.47203+001 T 3= 27.12314+001 T 4= 26.90650+001 T 5= 26.82824+001 T 10=-46.00000+001

QRATE 43.01397-003 QTOTAL 24.03348-002

END OF DATA

185

MARY E. GEALY

JOB MESSAGES

JOB,004645, 600,899RCC
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=
MT12
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=
WHICH MT HAS L.U. 40

BL,MT14
MT14

WHICH MT HAS L.U. 22

BL,MT12
MT12

RELEASED 9=MT13=(CINDA MASTER),ED=01,RL=01,DATE=052373,RC=999
RELEASED 22=MT12=(UNLABELED)
RELEASED 40=MT14=(UNLABELED)

The stored original problem of sample 1B is used to illustrate the RECALL option. The initial data will be taken and given a new TIMEND of 5.0 min, as well as a negative heating rate.

The first card in the problem data deck is

1	13	22
↓	↓	↓
RECALL	STOR1	0

The title block is changed to

```

8
↓
BCD 3INITIAL PARAMETERS
BCD 3SAMPLE PROBLEM 1B---RECALL
END
    
```

Since there are no temperature or conductor data changes, the nodal and conductor blocks are blank.

```

8
↓
BCD 3NODE DATA
END
BCD 3CONDUCTOR DATA
END
    
```

The change in TIMEND produces the following constants block:

```

8
↓
BCD 3CONSTANTS DATA
    TIMEND,5.0
END
    
```

Array 4 is changed to induce a negative heating rate.

```

8
↓
BCD 3ARRAY DATA
    4,0.,-3.,3.,-4.,7.,-4.,10.,-3.,END $-Q VS TIME
END
    
```

Since no operations block changes are allowed in parameter runs, the data deck is terminated by

```

8
↓
BCD 3END OF DATA
    
```

NRL REPORT 7656

The binary program tape (processor) was stored during the original run, so recompilation of the constructed subroutines is not necessary. Since the STOREP calls are still in the processor, the original and final data of this RECALL problem are stored on drum unit 22. The unit was not equipped in this example, however. See Section VII for the correct deck setup.

The processed problem printouts of the second run follow.

SEQUENCE 04060 STARTED PRINTING 08/08/73 AT 192112 ON LP01
DRUM SCOPE 2.1 COMPUTER TWO. MAX. DEMAND IS 570008 VERSION 004 10/31/72
SEQUENCE NUMBER 004060 STARTED AT TIME 191527 DATED 08/08/73
JOB(5),40800800,898RCC,10
EQUIP,0=(CINDA MASTER,1,1.999),RO,HI,DA
EQUIP,21=**,RO,HI,DA
EQUIP,40=**,RO,HI,DA
BINARY DECK
BANK,(0),/4/
LOAD,09
RUN,1,1000

PROGRAM NAMES

1 77672	SEARCH	00105	1 77603	STFFB	00067	1 76026	GENLNK	00540	1 74614	SPLIT	01026
1 74272	SKIP	00322	1 72717	CODERD	01353	1 71712	PSUEDO	00661	1 71627	ORMIN	00063
1 71417	PACK43	00210	1 71345	BIT	00052	1 71051	PRESUB	00274	1 70731	WRTBLK	00120
1 70673	WRTSCOPE	00036	1 65267	CINDA4	03404	1 62747	BLKCRD	01170	0 71134	DATARD	06643
1 62141	PREPRO	00605	0 60645	INITAL	10467	1 57571	FINAL	02350	1 57335	ALLOC.	00234
0 56177	IOH.	02246	1 87803	Q7QLDLCL	00032	0 54622	IOF.	01355	1 57072	QBQERROR	00211
1 57005	BFI.	00065	1 56740	ENC.	00045	1 56705	DEC.	00033	0 54570	QBQIFUNI	00032
0 54544	EFT.	00024	0 54472	SLI.	00052	0 54430	QBINOOUT4	00042	0 53654	IOB.	00554
0 53562	BSP.	00072	0 53527	QBQIFIOC	00033	0 53471	STM.	00036	0 53424	TSH.	00045
0 53407	REW.	00015	0 52764	IOS.	00423	0 52604	Q1BREINT	00160	0 52420	Q1QSTORE	00164
0 52342	QBQENTRY	00056									

PROGRAM EXTENS.
NONE

LABELED COMMON

1 76566	CRDBLK	01015	1 76914	TAPE	00012	1 75775	DATA	00017	1 75657	SUBLST	00116
1 75654	QLOGIC	00003	1 75642	PLOGIC	00012	1 72573	LOGIC	00124	1 64613	NARRAY	00454
1 64137	DIMARY	00454	1 62746	WJS	00001						

NUMBERED COMMON

0 24550	4	17500	1 00001	1	17500	1 17501	3	17500	1 37201	2	17500
---------	---	-------	---------	---	-------	---------	---	-------	---------	---	-------

ENTRY POINTS

0 77777	SENTRY	1 77675	SEARCH	0 52342	QBQDICT.	1 77607	STFFB
1 76214	GENLNK	0 52420	Q3Q10040	0 52655	Q3Q00040	0 53303	THEND.
1 63176	BLKCRD	0 53612	REW.	0 53430	TSH.	0 53475	STM.
0 53275	QNSINGL.	1 74656	SPLIT	1 74275	SKRP	0 53536	QBQIFEOF
0 53565	BSP.	0 53657	TSB.	0 53671	STB.	0 54446	QBQINP4
1 73077	CODERD	0 72636	DATARD	0 54500	SLB.	0 54472	SLI.
1 71760	PSUEDO	1 71622	PACK43	1 71632	ORMIN	0 54547	EFT.
1 71345	BIT	1 71310	PRESUB	1 66371	CINDA4	1 70701	WRTSCOPE
0 54573	QBQIFUNI	1 70764	WRTBLK	1 56710	DEC.	1 56743	ENC.
1 57020	BFO.	1 62214	PREPRO	0 52346	QBQENTRY	0 67407	INITAL
1 60643	FINAL	1 57974	QBQERROR	0 52343	EXIT	0 54430	QBQOUT4
0 54625	IOF.	0 53937	QBQHIST.	1 57303	Q7QLDLCL	0 53004	IOS.
0 53307	IOR.	0 56223	IOH.	0 60320	TERR.	0 56177	BCDBUF.
0 52764	IOE.	0 53156	QBQCHAIN	0 53300	QNDQUBL.	0 55644	ETAB.
1 57473	ALLOC.	1 57843	RETURN.	1 57447	BUSY.	1 57456	IRETURN.
1 57537	ALLOCIN.	0 60213	ELD.	0 60277	.RRPCNT.	1 57072	QBERRORN
1 57073	QBNOTRAC	1 57806	QBQERSET	1 57010	BFI.	0 54012	ELB.
0 53532	QBQIFIOC	0 52604	Q1Q00100	0 52604	Q1Q01100	0 52623	Q1Q02100
0 52627	Q1Q03100	0 52633	Q1Q04100	0 52637	Q1Q05100	0 52607	Q3Q00140
0 52664	Q3Q01040	0 52615	Q3Q01140	0 52723	Q3Q02040	0 52673	Q3Q02140
0 52732	Q3Q03040	0 52701	Q3Q03140	0 52741	Q3Q04040	0 52707	Q3Q04140
0 52750	Q3Q05040	0 52715	Q3Q05140	0 52506	Q1Q10010	0 52506	Q1Q10020
0 52454	Q1Q10030	0 52527	Q1Q10100	0 52501	Q1Q10120	0 52460	Q1Q10130
0 52535	Q1Q10200	0 52463	Q1Q10210	0 52463	Q1Q10230	0 52527	Q1Q10300
0 52512	Q1Q10310	0 52501	Q1Q10320	0 52522	Q1Q10400	0 52515	Q1Q10410
0 52474	Q1Q10420	0 52474	Q1Q10430	0 52420	Q3Q10140	0 52420	Q3Q10240
0 52420	Q3Q10340	0 52420	Q3Q10440				

EXECUTION STARTED AT 1916 -34

RECALL PROBLEM --- STORI , 0

```
BCD 3INITIAL PARAMETERS
BCD 3SAMPLE 18---RECALL
END
BCD 3NODE DATA
END
BCD 3CONDUCTOR DATA
END
BCD 3CONSTANTS DATA
TIMEND,5.
END
BCD 3ARRAY DATA
4,0,,-3,3,,-4,7,,-4,-,10,,-3,END S =Q VS TIME
END
LIBRARY,09,LCINDA
LOAD,40
RUN,5,2500
```

PROGRAM NAMES															
1	77744	LINKO	00033	1	77812	EXEC TN	00027	1	77202	VARBL1	00110	1	77160	VARBL2	00022
1	77140	OUTCAL	00020	1	77103	SKPLIN	00035	1	77061	QBQLODA	00022	1	76601	LAGRAN	00260
1	76527	BIT	00052	1	74261	IOH.	02246	1	74227	Q7QLODLC	00032	1	74016	QBQERROR	00211
1	73562	ALLOC.	00234	1	73865	WRTL08	00175	1	73244	STNDRD	00121	1	73158	TOPLIN	00066
1	73131	QBQXMODF	00025	1	73043	D1D1WM	00066	1	72535	UNPAK	00306	1	72520	REW.	00015
1	72446	SLI.	00052	1	72404	QBINOUT4	00042	1	72346	STH.	00036	1	71572	IOB.	00554
1	71412	Q1QREINT	00160	1	70967	IOS.	00423	1	67412	IOP.	01355	1	67176	PRINT	00214
1	67021	TPRINT	00155	1	66741	QINTEG	00060	1	66643	RDYNQS	00076	1	66415	D1DEG1	00226
1	66277	VARGSM	00116	1	66221	VARCSM	00056	1	65061	CNFRWD	01140	1	63373	STOREP	01466
1	62622	INPUTT	00551	1	62436	Q1QSTORE	00164	1	62360	QBQENTRY	00056				

PROGRAM EXTENS.
NONE

LABELED COMMON															
1	77720	TITLE	00024	1	77712	TEMP	00006	1	77704	CAM	00006	1	77677	SOURCE	00005
1	77671	COND	00006	1	77663	PCS	00006	1	77657	KONST	00004	1	77615	ARRAY	00042
1	77533	FIXCON	00062	1	77521	DIMENS	00012	1	77513	PRNT	00006	1	77348	XSPACE	00146
1	77341	LOGIC	00004												

NUMBERED COMMON
NONE

ENTRY POINTS															
0	77777	SENTRY		1	77747	LINKO		1	62364	QBQENTRY		1	62436	Q3Q10040	
1	62360	QBQOICT.		1	62637	INPUTT		1	77315	EXEC TN		1	64216	STOREP	
1	65115	CNFRWD		1	77205	VARBL1		1	66324	VARGSM		1	66302	VARGSM	
1	66431	D1DEG1		1	77163	VARBL2		1	66646	RDYNQS		1	66744	QINTEG	
1	77143	OUTCAL		1	67842	TPRINT		1	67315	PRNTNL		1	67415	IOP.	
1	71042	QBQHIST.		1	71906	THEND.		1	71463	Q3Q00040		1	62361	EXIT	
1	71575	TSB.		1	72852	STH.		1	72422	QBQINP4		1	72448	SLI.	
1	71300	QNSINGL.		1	72923	REW.		1	71607	STB.		1	72404	QBQOUT4	
1	72454	SLO.		1	72542	UNPAK		1	73046	D1D1WM		1	73140	XMODF	
1	73215	TOPLIN		1	73325	STNDRD		1	73411	WRTL08		1	73720	ALLOC.	
1	73570	RETURN.		1	73674	BUSY.		1	73703	IRETURN.		1	74020	QBQERROR	
1	74227	Q7QLODLC		1	71803	QNDQUBL.		1	71007	IOB.		1	71312	IOR.	
1	70767	IOE.		1	74905	IOH.		1	74261	BCQBUF.		1	76527	BIT	
1	76614	LAGRAN		1	77061	QBQLODA		1	77112	SKPLIN		1	77102	QBQLDCON	
1	77077	QBQLODA		1	76975	ELD.		1	76361	.RQPCNT.		1	76402	.TSERR.	
1	74016	QBERRORN		1	74017	QBNOTRAC		1	74132	QBQERSET		1	73764	ALLOCIN.	
1	73134	QBQXMODF		1	71730	ELB.		1	71412	Q1Q00100		1	71412	Q1Q01100	
1	71431	Q1Q02100		1	71435	Q1Q03100		1	71441	Q1Q04100		1	71445	Q1Q05100	
1	71415	Q3Q00140		1	71472	Q3Q01040		1	71423	Q3Q01140		1	71531	Q3Q02040	
1	71501	Q3Q02140		1	71540	Q3Q03040		1	71507	Q3Q03140		1	71547	Q3Q04040	
1	71515	Q3Q04140		1	71556	Q3Q05040		1	71523	Q3Q05140		1	71161	QBQCHAIN	
1	70434	ETAB.		1	67306	PRINT		1	62524	Q1Q10010		1	62524	Q1Q10020	
1	62472	Q1Q10030		1	62545	Q1Q10100		1	62517	Q1Q10120		1	62476	Q1Q10130	
1	62553	Q1Q10200		1	62501	Q1Q10210		1	62501	Q1Q10230		1	62545	Q1Q10300	
1	62530	Q1Q10310		1	62517	Q1Q10320		1	62540	Q1Q10400		1	62533	Q1Q10410	
1	62512	Q1Q10420		1	62512	Q1Q10430		1	62436	Q3Q10140		1	62436	Q3Q10240	
1	62436	Q3Q10340		1	62436	Q3Q10440									

EXECUTION STARTED AT 1921 -20

95 LOCATIONS AVAILABLE

SAMPLE 1B---RECALL

* * * *
 TIME 00.00000+000 DTIMEU 00.00000+000 CSGMIN(1) 79.36899-003 DTMPCC(1) 00.00000+000 ARLXCC(0) 00.00000+000
 T 1= 80.00000+000 T 2= 80.00000+000 T 3= 80.00000+000 T 4= 80.00000+000 T 5= 80.00000+000 T 10=-46.00000+001

GRATE 00.00000+000 QTOTAL 00.00000+000

* * * *
 TIME 50.00000-002 DTIMEU 23.69129-003 CSGMIN(5) 79.63843-003 DTMPCC(1) 70.43746-002 ARLXCC(0) 00.00000+000
 T 1= 53.68598+000 T 2= 65.16701+000 T 3= 72.30302+000 T 4= 76.05114+000 T 5= 77.17596+000 T 10=-46.00000+001

GRATE 95.74068-004 QTOTAL 48.74938-004

* * * *
 TIME 10.00000-001 DTIMEU 22.30141-003 CSGMIN(5) 80.33910-003 DTMPCC(1) 58.36632-002 ARLXCC(0) 00.00000+000
 T 1= 40.07690+000 T 2= 52.92449+000 T 3= 61.59724+000 T 4= 66.58331+000 T 5= 68.18559+000 T 10=-46.00000+001

GRATE 88.11568-004 QTOTAL 94.67770-004

* * * *
 TIME 15.00000-001 DTIMEU 20.12225-003 CSGMIN(5) 81.18391-003 DTMPCC(1) 54.53454-002 ARLXCC(0) 00.00000+000
 T 1= 26.75812+000 T 2= 40.70173+000 T 3= 50.19081+000 T 4= 55.69574+000 T 5= 57.47918+000 T 10=-46.00000+001

GRATE 79.65650-004 QTOTAL 13.64045-003

* * * *
 TIME 20.00000-001 DTIMEU 37.05690-003 CSGMIN(5) 82.09882-003 DTMPCC(1) 10.68712-001 ARLXCC(0) 00.00000+000
 T 1= 12.72987+000 T 2= 27.81440+000 T 3= 38.06279+000 T 4= 44.00820+000 T 5= 45.93819+000 T 10=-46.00000+001

GRATE 71.33532-004 QTOTAL 17.40886-003

* * * *
 TIME 25.00000-001 DTIMEU 34.47240-003 CSGMIN(5) 83.16484-003 DTMPCC(1) 10.58945-001 ARLXCC(0) 00.00000+000
 T 1=-21.91916-001 T 2= 14.12754+000 T 3= 25.18076+000 T 4= 31.58364+000 T 5= 33.66399+000 T 10=-46.00000+001

GRATE 63.20805-004 QTOTAL 20.75598-003

199

MARY E. GEALY

SAMPLE 1B---RECALL

* * * *
 TIME 30.00000-001 DTIMEU 31.55878-003 CSGMIN(5) 84.36528-003 DTMPCC(1) 10.33833-001 ARLXCC(0) 00.00000+000
 T 1=-18.10093+000 T 2=-41.82387-002 T 3= 11.51046+000 T 4= 18.40611+000 T 5= 20.64749+000 T 10=-46.00000+001

GRATE 55.41477-004 QTOTAL 23.70579-003

* * * *
 TIME 35.00000-001 DTIMEU 28.27367-003 CSGMIN(5) 85.71831-003 DTMPCC(1) 89.41939-002 ARLXCC(0) 00.00000+000
 T 1=-33.98462+000 T 2=-15.48183+000 T 3=-28.20476-001 T 4= 45.04702-001 T 5= 08.93648-001 T 10=-46.00000+001

GRATE 48.03297-004 QTOTAL 26.27684-003

* * * *
 TIME 40.00000-001 DTIMEU 24.59518-003 CSGMIN(5) 87.21536-003 DTMPCC(1) 79.26287-002 ARLXCC(0) 00.00000+000
 T 1=-49.93476+000 T 2=-30.57536+000 T 3=-17.44109+000 T 4=-98.14308-001 T 5=-73.24506-001 T 10=-46.00000+001

GRATE 41.24798-004 QTOTAL 28.49637-003

* * * *
 TIME 45.00000-001 DTIMEU 41.56360-003 CSGMIN(5) 88.78902-003 DTMPCC(1) 13.73111-001 ARLXCC(0) 00.00000+000
 T 1=-66.25569+000 T 2=-46.01367+000 T 3=-32.33825+000 T 4=-24.41424+000 T 5=-21.82787+000 T 10=-46.00000+001

GRATE 35.18335-004 QTOTAL 30.39069-003

* * * *
 TIME 50.00000-001 DTIMEU 37.54882-003 CSGMIN(5) 90.59414-003 DTMPCC(1) 12.76585-001 ARLXCC(0) 00.00000+000
 T 1=-83.02761+000 T 2=-61.79634+000 T 3=-47.52776+000 T 4=-39.28323+000 T 5=-36.59381+000 T 10=-46.00000+001

GRATE 29.72409-004 QTOTAL 31.99857-003

END OF DATA

197

NRL REPORT 7656

JOB MESSAGES

JOB,004060, 600,89#RCC
NEED 09 = MT =(CINDA MASTER),ED=01,RL=01,DATE=
WHICH MT HAS L.U. 21

BL,MT16
MT14 GO
MT14
BL,MT16
BL,MT16
BL,MT16

WHICH MT HAS L.U. 40

MT15

RELEASED 9=MT12=(CINDA MASTER),ED=01,RL=01,DATE=052373,RC=999
RELEASED 21=MT14=(UNLABELED)
RELEASED 40=MT15=(UNLABELED)

SAMPLE NO. 1B (Continued)

TSAVE AND PLOT RUN

This run and the plots are similar to the TSAVE run of sample 1A. The following data were used.

1	5	12	22	32	42
↓	↓	↓	↓	↓	↓
CINDA SAMPLE PROBLEM 1B					
		0.00	10.00	80.00	280.00
* 10*					
CINDA SAME PROBLEM 1B					
		0.00	10.00	-470.00	-450.00
+ 10*					
EOF					

The printed output from the plot run are as follows.

CINDA SAMPLE PROBLEM 1B

X-AXIS LIMITS -- 0.00+000, 1.00+001, Y-AXIS LIMITS -- 8.00+001, 280+002

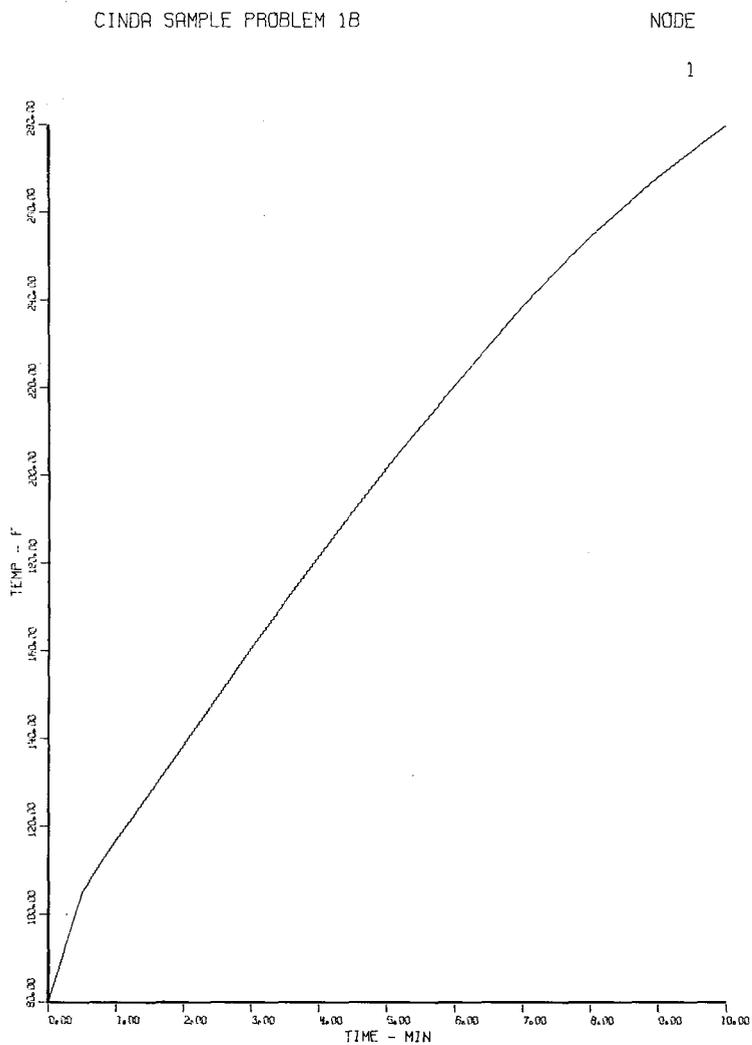
ALL NODES BUT THE FOLLOWING WILL BE PLOTTED ---
10*

CINDA SAMPLE PROBLEM 1B

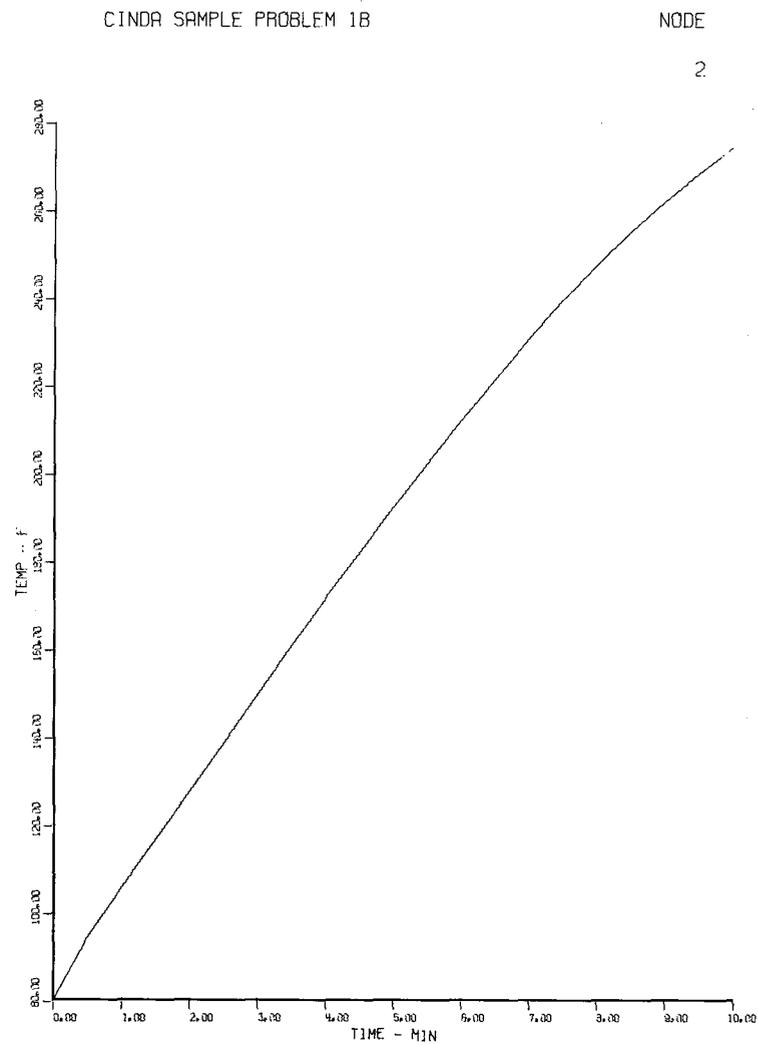
X-AXIS LIMITS -- 0.00+000, 1.00+001, Y-AXIS LIMITS -- -4.70+002, -450+002

THE FOLLOWING NODES WILL BE PLOTTED ---
10*

The plots are given in Fig. B2.

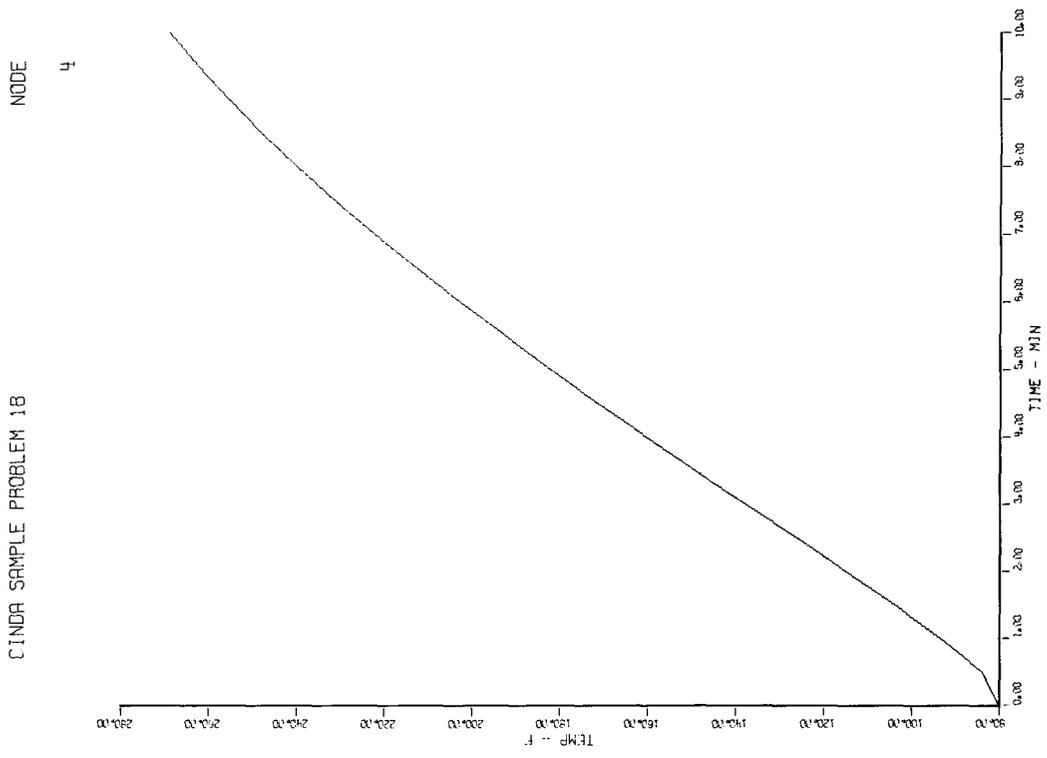


(a)

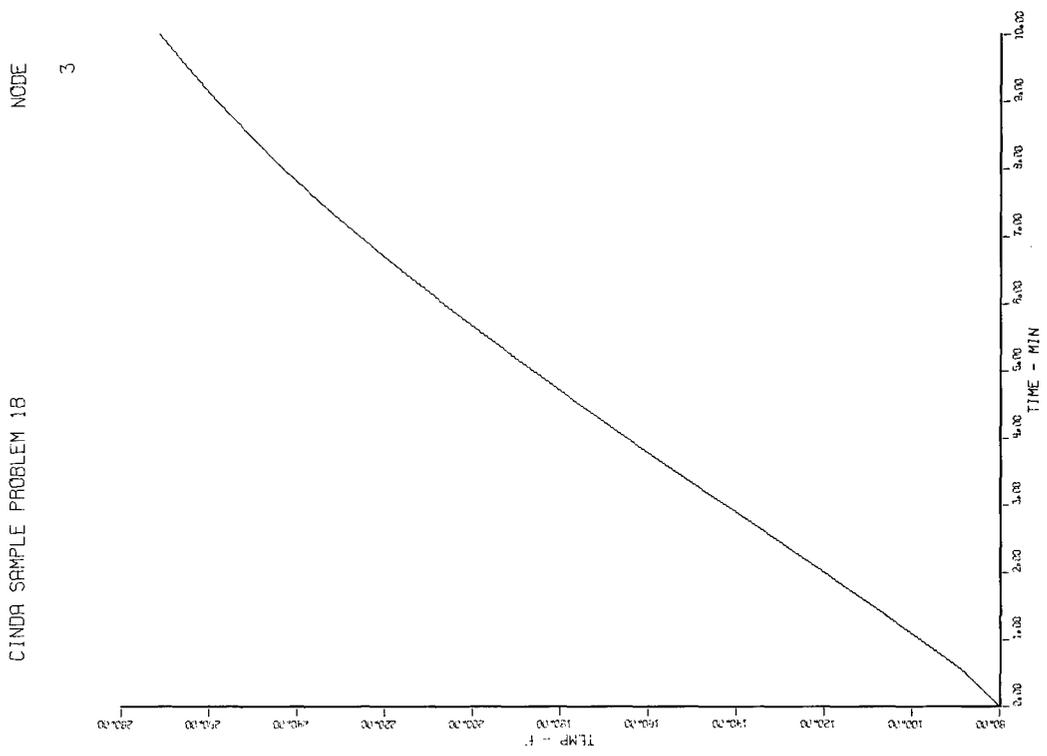


(b)

Fig. B2—Time vs temperature plots for CINDA Sample Problem 1B. Plots are shown in the order in which they are plotted. Time and temperature values will be in the E format, rather than in the F format as shown.

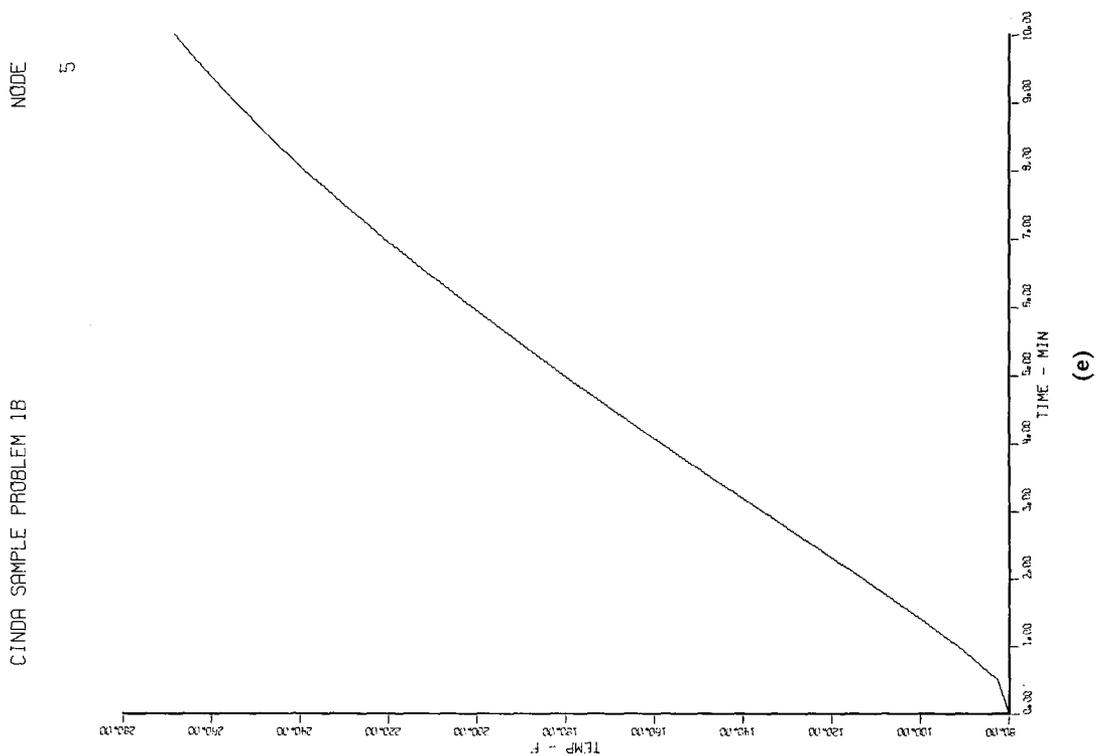


(d)



(c)

Fig. B2 (Cont'd)—Time vs temperature plots for CINDA Sample Problem 1B. Plots are shown in the order in which they are plotted. Time and temperature values will be in the E format, rather than in the F format as shown.



(e)

Fig. B2 (Cont'd)—Time vs temperature plots for CINDA Sample Problem 1B. Plots are shown in the order in which they are plotted. Time and temperature values will be in the E format, rather than in the F format as shown.

