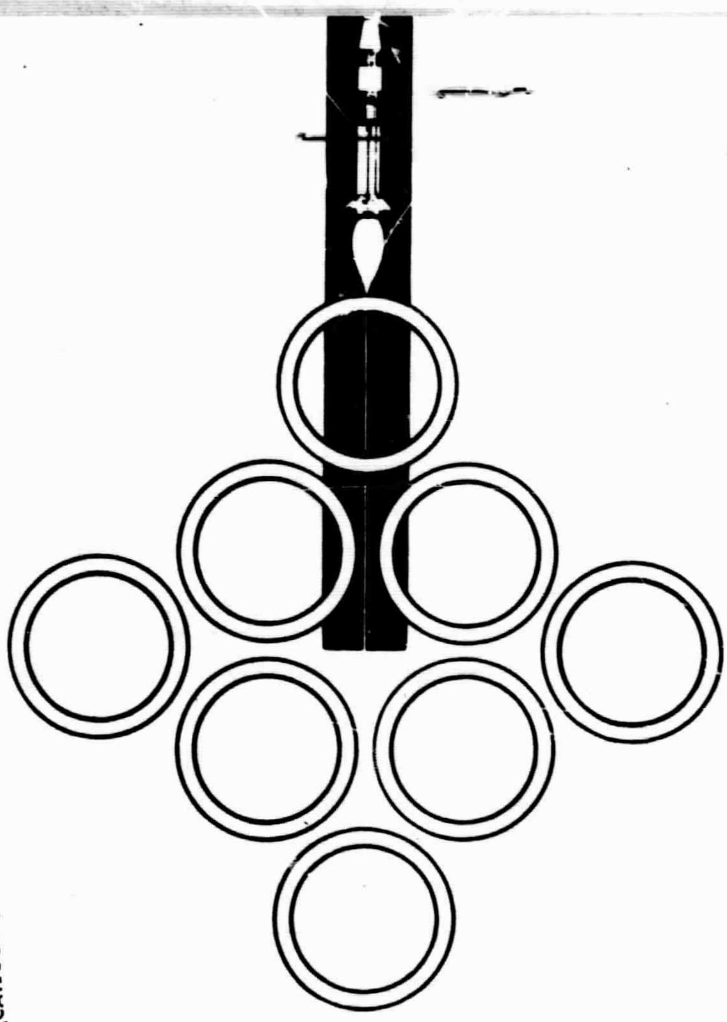
send this one to FAC

ENGINEERING DEPARTMENT

TECHNICAL NOTE

TN-AP-67-287



CHRYSLER IMPROVED NUMERICAL DIFFERENCING ANALYZER

FOR 3RD GENERATION COMPUTERS

SPACE DIVISION CHRYSLER CORPORATION

NASA CR 99595

SPACE DIVISION CHRYSLER CORPORATION

CINDA-3G

TECHNICAL NOTE

TN-AP-67-287

CHRYSLER IMPRØVED NUMERICAL DIFFERENCING ANALYZER for 3rd GENERATIØN CØMPUTERS

October 20, 1967

Prepared by: David R. Lewis

D. R. Lewis, Supervisor
Systems Programming Unit

J. D. Gaski

J. D. Gaski, Technical Staff
Aerospace Physics Branch

L. R. Thompson

L. R. Thompson, Sr. Programmer
Systems Programming Unit

Approved by: F. M. Van Sickle

F. M. Van Sickle, Chief Engineer
Service Engineering Branch

R. H. Ross

R. H. Ross, Chief Engineer
Aerospace Physics Branch

CHRYSLER CORPORATION SPACE DIVISION - NEW ORLEANS, LOUISIANA

TABLE OF CONTENTS

SECTION I

SUMMARY

The original CINDA computer program, coded in FØRTRAN-II and FAP for the IBM-7094 computers, is documented as Chrysler Corporation Space Division Technical Note TN-AP-66-15, dated April 30, 1966. It has gained wide acceptance and usage throughout the Thermodynamic community and in fact become a standardized program at several installations. However, the original program was unsuitable for standard operation on third generation computers. Therefore, the National Aeronautics and Space Administration's Manned Spacecraft Center awarded a contract to Chrysler to produce the version described herein. This new version entitled CINDA-3G is coded in FØRTRAN-V for the Univac-1108 computer. Minor portions are coded in the Sleuth II assembly language in order to achieve bit manipulation and shifting operations where required and also to allow certain user subroutines to have a variable number of arguments. Problem data decks prepared for the old version require only a few changes in order to run under the new version. Although numerous comparisons will be made to TN-AP-66-15, this document is intended to be complete and self-contained.

SECTION II

INTRODUCTION

The computer program described herein, Chrysler Improved Numerical Differencing Analyzer for 3rd Generation computers (CINDA-3G), was developed by the Thermodynamics Section of the Aerospace Physics Branch of Chrysler Corporation Space Division at the National Aeronautics and Space Administration's Michoud Assembly Facility. Programming and systems integration for the Univac-1108 computer was performed by the CCSD Computation Services Group at the NASA Central Computer Facility, Slidell, Louisiana. A major portion of this work was done under contract NAS9-7043 from the Manned Spacecraft Center, Houston, Texas.

This program appears virtually identical to its predecessor (CINDA, CCSD-TN-AP-66-15) but has been almost completely rewritten in order to take advantage of the improved systems software and machine speeds of the 3rd generation computers. The entire programming approach has changed. Whereas CINDA was virtually a self contained program having its own Update, Monitor and Compiler; the CINDA-3G foundation consists of a preprocessor (written in Fortran) which accepts the same user input data and converts it into advanced Fortran language subroutines and block data input which is then passed onto the system Fortran Compiler. While this requires a double pass on data where previously only one was required, the increased speed and improved software of the 3rd generation machines more than compensates. Transient thermal analysis solutions realize the increased machine speeds and in addition, perform fewer operations which further reduces solution times.

The CINDA-3G program 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 to quickly construct mathematical models of complex 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 1-D incompressible fluid flow including valving and transport delay effects, etc. The optional combination of these capabilities in conjunction with model size allowable ($>$ 4000 nodes for a linear 3-D system on 65K core) makes CINDA-3G an extremely potent analytical tool for thermal systems analysis, in the hands of a competent engineer analyist.

SECTION III

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 uniquely identify 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.

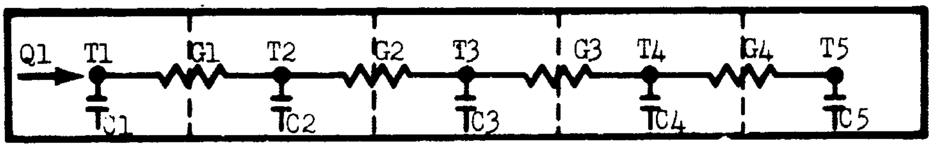
Perhaps the most critical aspect of the lumped parameter approach is determining the lump size. There are methods for optimizing the lump size but they usually involve more analytical effort and computer time than the original analysis. One must also keep in mind that for a transient problem, time is 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.

In order to reduce round-off error the explicit stability criteria of the lump (the capacitance value divided be 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 eight, sixteen or thirty-two 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 notstep over boundary excitation points or they will be missed. Don't 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, an interpolation call may be utilized 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.

The following diagram displays the lumped parameter representation and network superposition of a one dimensional heat transfer problem.



(F1)

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 Farenheit system allows easy automation of this nonlinear transfer function by the program and reduces the radiation conductor value to the product of the Stephan-Boltzman 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 straight forward. 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.,

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

That the diffusivity ($\alpha = k/\rho C_p$) may be temperature varying or nonlinear radiation transfer occuring is immaterial at this point. Of importance is how equation one 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;

$$\frac{\partial T}{\partial t} \approx \frac{T' - T}{\Delta t} \tag{2}$$

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

The right side of equation one 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.

$$\frac{\partial T}{\partial t} = \beta\left(\alpha \nabla^2 T + S\right) + \left(1-\beta\right)\left(\alpha' \nabla^2 T' + S'\right) \tag{3}$$

$$0 \leq \beta \leq 1$$

Any value of β less than one 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 one half yields an unconditionally stable set of equations or in other words, any time step desired may be used. Values of β greater than one half invoke stability criteria or limitations on the magnitude of the time step. A value of β equal to one half 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 equation one generally yield an implicit set of equations similar to equation three with β equal to one half. The following finite difference approach generally applies to transformed equations.

Let's consider the right side of equation three with β equal to one and rewrite it as follows;

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

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 underconsideration 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 equation four as follows;

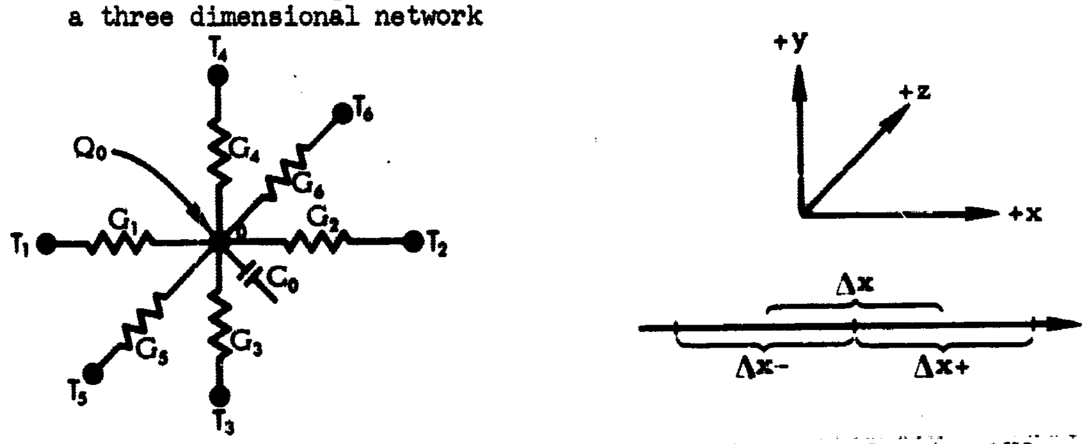
$$\frac{a}{\Delta x}\left(\frac{\partial T_2}{\partial x-} - \frac{\partial T_2}{\partial x+}\right) \approx \frac{a}{\Delta x}\left(\frac{T_1 - T_2}{\Delta x-} + \frac{T_3 - T_2}{\Delta x+}\right) \tag{5}$$

Applying the above step to the y and z portions of equation four yields the common denominator of volume ($V = \Delta x * \Delta y * \Delta z$). Using equation three with β equal to one, finite differencing with the steps used for equations three, four and five, substituting $\alpha = k/\rho Cp$ and multiplying both sides by ρVCp yields

$$\frac{\rho V Cp}{\Delta t}\left(T_0' - T_0\right) = \frac{k Ax}{\Delta x-}\left(T_1 - T_0\right) + \frac{k Ax}{\Delta x+}\left(T_2 - T_0\right)$$
$$+ \frac{k Ay}{\Delta y-}\left(T_3 - T_0\right) + \frac{k Ay}{\Delta y+}\left(T_4 - T_0\right)$$
$$+ \frac{k Az}{\Delta z-}\left(T_5 - T_0\right) + \frac{k Az}{\Delta z+}\left(T_6 - T_0\right) + Q \tag{6}$$

where $Ax = \Delta y \cdot \Delta z$, $Ay = \Delta x \cdot \Delta z$, $Az = \Delta x \cdot \Delta y$ and $Q = \rho VCpS$

The numbering system corresponds to the following portion of a three dimensional network



(F2)

It should be obvious that the network capacitance value is ρVCp, that the G1 value is $kAx/\Delta x-$, etc. Equation six may then be written as

$$C_0(T_0'-T_0)/\Delta t = G_1(T_1-T_0)+G_2(T_2-T_0)+G_3(T_3-T_0)+G_4(T_4-T_0) +G_5(T_5-T_0)+G_6(T_6-T_0)+Q_0 \tag{7}$$

or in engineering terminology the rate of change of temperature wi respect to time is proportional to the summation of heat flows int the node.

It should be noted that Figure F2 is essentially superimposed on a lumped parameter cube of a physical system and is the network representation of equation one. Since equation seven 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 β greater then zero, 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 β equal to zero.

$$\alpha \nabla^2 T + S = 0 \tag{8}$$

The solution of Poisson's equation (eight) 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 equation seven had all the right side values primed and the left side was subtracted from both sides, we could think of $Co/\Delta t$ as a Go term and To (old) would then become a boundary node. In a manner of speaking, the capacitor we look at in 3-D becomes a conductor in 4-D. 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 model, network model and mathematical model, let's investigate the commonality of the various methods of solution. By describing the network of Figure F1 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 equation six has only one unknown. It's solution is easily obtainable as long as any associated stability criteria are continously 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 a combination of explicit and implicit, the known explicit portion would have been calculated and added to the Q terms, then the β factor divided into the Q terms and multiplied times the Δt term.

If we divide the Δt term into the C terms and indicate this by priming C we can reformulate equation nine as follows:

$$
\begin{aligned}
(C_1'+G_1)\quad T_1' &= Q_1'+C_1'T_1+G_1T_2' \\
(C_2'+G_1+G_2)T_2' &= Q_2'+C_2'T_2+G_1T_1'+G_2T_3' \\
(C_3'+G_2+G_3)T_3' &= Q_3'+C_3'T_3+G_2T_2'+G_3T_4' \\
(C_4'+G_3+G_4)T_4' &= Q_4'+C_4'T_4+G_3T_3'+G_4T_5' \\
(C_5'+G_4)\quad T_5' &= Q_5'+C_5'T_5+G_4T_4'
\end{aligned}
\qquad (10)
$$

This equation can be generalized as

$$T_i' = \frac{C_i'T_i + \sum G_a T_a' + Q_i'}{C_i' + \sum G_a} \quad \text{, sub a for adjoining} \qquad (11)$$

where the sub 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 equation eleven, although utilized to obtain transient solutions, can be considered as a steady state equation in 4-D. By rewriting equations ten in the form of equation eleven we are in a position to discuss iterative techniques. By assuming all old values on the right hand side of ten 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, otherwise old. 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 continously seek the node with the maximum temperature difference between old and new and calculate it. Programming wise, 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 criteria 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

When working with a simultaneous set of equations such as equations ten, they are quite often treated by matrix methods and formulated as follows:

$$\left|A\right| \left|T'\right| = \left|B\right| \tag{12}$$

where

$$\left|A\right| = \begin{bmatrix} (C_1'+G_1) & -G_1 & 0 & 0 & 0 \\ -G_1 & (C_2'+G_1+G_2) & -G_2 & 0 & 0 \\ 0 & -G_2 & (C_3'+G_2+G_3) & -G_3 & 0 \\ 0 & 0 & -G_3 & (C_4'+G_3+G_4) & -G_4 \\ 0 & 0 & 0 & -G_4 & (C_5'+G_4) \end{bmatrix} \tag{13}$$

and

$$\left|T'\right| = \begin{Bmatrix} T_1' \\ T_2' \\ T_3' \\ T_4' \\ T_5' \end{Bmatrix} , \qquad \left|B\right| = \begin{Bmatrix} Q_1'+C_1'T_1 \\ Q_2'+C_2'T_2 \\ Q_3'+C_3'T_3 \\ Q_4'+C_4'T_4 \\ Q_5'+C_5'T_5 \end{Bmatrix}$$

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

$$\left|T'\right| = \left|A\right|^{-1} \left|B\right| \tag{14}$$

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 one thousand 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 Figure F1.

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

Note also the row and column position of conductor values off the main diagonal in the [A] coefficient matrix, equation 13; By retaining the adjoining node numbers for each conductor we are able to identify their element position in the coefficient matrix. As a consequence we need store only the finite values. The main diagonal term is a composite of the node capacitance and conductor values off of 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 one is found. When this occurs the conductor number and other adjoining node number are stored as a doublet value. The search is continued until all ones 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 is shown in Figure 16A. 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 is shown in Figure 16B.

LPCS		SPCS		
-1,2		-1,2		
1,1		-2,3		
-2,3		-3,4		
2,2		-4,5		
-3,4		-0,0	(B)	(16)
3,3				
-4,5				
-4,4	(A)			

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 one. 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 long pseudo-compute sequence 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.

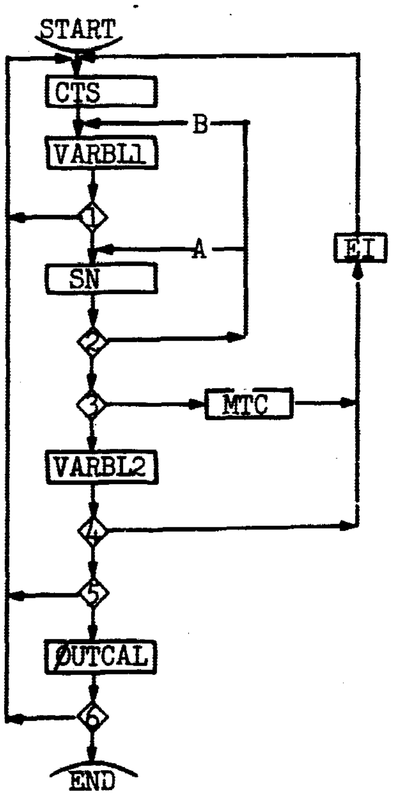
The short pseudo-compute sequence identifies each conductor only once and in this manner takes advantage of the symmetry of the coefficient matrix as well as the sparsity. It is well suited for explicit methods of solution. The node being operated on and the adjoining node number reveal their temperature 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 sequences 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 short pseudo-compute sequence takes advantage of any symmetry which may exist. Multiply connected conductors which will be covered in the next section also 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 one 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. [illegible]o may require from zero to three extra locations per node fo[illegible]rmediate 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 pseudo-compute sequence 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 per node can vary from nine to fifteen.

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.



OPERATION	DESCRIPTION
CTS	Calculate time step
VARBL1	Variables 1 operations
SN	Solve Network
VARBL2	Variables 2 operations
ØUTCAL	Output calls operations
MTC	Modify time control
EI	Erase iteration
Check	Reverse direction if
1	Backup nonzero
2	Relaxation criteria not net
3	Time or temp change to large
4	Backup nonzero
5	Not time to print
6	Problem stop time not reached

BASIC FLOW CHART FOR NETWORK SOLUTION SUBROUTINES

(F3)

Order of Computation

A problem data deck consists of data and operations "blocks" which are preprocessed by CINDA-3G and passed on to the system FØRTRAN compiler. The operations blocks are named EXECUTIØN, VARIABLES 1, VARIABLES 2 and ØUTPUT CALLS. The FØRTRAN compiler constructs these blocks as individual subroutines with the entry names EXECTN, VARBL1, VARBL2 and ØUTCAL 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 EXECUTIØN 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 in Section 5.1 internally call upon VARBL1, VARBL2 and ØUTCAL. Their internal order of computation is quite similar, the primary difference being the analytical method by which they solve the network. Figure F3 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-3G is actually an operating system rather than an applications program. The more one studies and uses the program the more apparent this becomes. In order for the program to accomplish the desired operations with regards to overlay features, data packing, dynamic storage allocation, subroutine library file and yet be written in Fortran, it was necessary to program CINDA-3G as a preprocessor. This preprocessor operates in an integral fashion with a large library of assorted subroutine which can be called in any sequence desired yet operate in an integrated manner. It reads all of the input data, assigns relative numbers, packs it, forms the pseudo-compute sequence and writes it on a peripheral unit as Fortran source language with all of the data values dimensioned exactly in name common. It then turns control over to the system Fortran compiler which compiles the constructed subroutines and enters execution. The Fortran allocator has access to the CINDA-3G subroutine library and loads only those subroutines referred to by the problem being processed.

Due to this type of operation, CINDA-3G is extremely dependent on the systems 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. The user need only consult the section in this document on "Control Cards and Deck Setup" to switch his problem from one machine to another.

SECTION IV

DATA INPUT REQUIREMENTS

A CINDA-3G problem data deck consists of both data and instruction cards. The card reading subroutines for CINDA-3G do not utilize a fixed format type of input; they use a free form format quite similar to the old SHARE decimal data read routine. The type of data is designated by a mnemonic code in columns eight, nine and ten. This is followed by the data field which consists of columns twelve through eighty or the instruction field which consists of columns twelve through seventy-two. 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 FØRTRAN common data and reforms instructions into FØRTRAN source language which are then passed on to the system FØRTRAN compiler. Instruction cards which contain an F in column one are passed on exactly as received. Any instruction card with or without an F in column one may contain a statement or sequence number in columns two through five which is passed on to and used by the FØRTRAN compiler.

The most frequently used mnemonic code was the old DEC designation which has been replaced by 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 alpha-numeric 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 one through nine in column twelve. The integer designates the number of six character words immediately following it. Blanks are retained and only the designated number of six 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 FØRTRAN comment card; it is not processed by the program but allows the user to insert non-data for clarification purposes. The code ØCT may be utilized and allows the input of a single octal word starting in column twelve. 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:

```
Col 8
    BCD 3GENERAL
 or BCD 3THERMAL SPCS
 or BCD 3THERMAL LPCS
```

The GENERAL indicates that a non-network 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 twenty 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 non-network problem.

Node Data Block

As discussed in section three, 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 for temperature and source only and a boundary node only receives 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 capaciatance 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) of 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:

```
Col 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 interpolation times factor products as the capacitance value.

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

```
Col 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 product 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:

```
Col 8   12
    BCD 3NØDE DATA
        1,80.,1.2,2,80.,1.3           $TWØ DIFFUSIØN NØDES
    CGS 3,80.,A1,4.63                 $SINGLE MATERIAL NØDE
    CGD 4,80.,A1,2.31,A2,4.76         $DØUBLE MATERIAL NØDE
    GEN 5,10,1,80.,4.63,1.,1.,1.      $GENERATE 10 NØDES, 5-14
        15,80.,-1,16,80.,-1           $TWØ ARITHMETIC NØDES
        -18,-460.,0                   $ØNE BØUNDARY NØDE
    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,

```
Col 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

$$G = k1(Tm)*F1, Tm = (Ta + Tb)/2.0$$

For CGS with F1 negative

$$G = k1(Ta)*|F1|$$

For CGD

$$G = 1.0/\left[1.0/(k1(Ta)*F1) + 1.0/(k2(Tb)*F2)\right]$$

The CGS mnemonic code may be utilized for either regular or radiation conductors. The data consists of the integer conductor number, one pair only of integer adjoining node numbers and is 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 versus 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-Boltzman 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:

```
Col 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 consists of the integer conductor number, integer number of how many 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:

```
Col 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

```
Col 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 uneffected 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 input prior to any radiation conductors. The following is illustrative of the various conductor input options.

```
Col 8
    BCD 3CØNDUCTØR DATA
        1,1,2,1.2,2,2,3,1.7                   $TWØ REGULAR CØNDUCTØRS
        3,3,4,4,5,5,6,1.5                     $TRIPLE PLACED CØNDUCTØR
        4,-7,8,-8,9,7.6                       $DØUBLE PLACED ØNE-WAY CØND.
    CGS 5,10,11,A3,4.6                        $VARIABLE CØNDUCTØR, SINGLE
    CGD 6,12,13,A3,4.1,A4,7.6                 $VARIABLE CØNDUCTØR, DØUBLE
    GEN 7,3,1,1,1,9,1,1.6,4.0,1.,1.           $GENERATE THREE CØNDUCTØRS
        -10,1,99,1.E-15                       $RADIATIØN CØNDUCTØR
    CGS -11,2,99,A5,1.E-14                    $VARIABLE EMISSIVITY RADIATIØN
    GEN -12,4,1,3,1,99,0,1.E-14,1.,1.,1.      $GENERATE FØUR RADIATIØN CØND.
    END
```

The first GEN card is equivalent to the following:

```
Col 12
     7,1,9,6.4,8,2,10,6.4,9,3,11,6.4
```

and the second GEN card is equivalent to

```
Col 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:

```
Col 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 two hundred 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 ($\sim$ 50) have alpha-numeric names while user constants receive a number. User constants are simply data storage locations which may contain integers, floating point numbers or up to six character alpha-numeric 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 non-zero, the just done time step 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/ΣG) 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.	
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 non-zero.	
ØUTPUT	Time interval for activating ØUTPUT 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

Dummy control constants with integer names.

RTEST,STEST,TTEST,UTEST,VTEST

Dummy control constants with non-integer names.

The following is representative of a constants data block.

```
Col 8
    BCD 3CØNSTANTS DATA
        TIMEND,10.0,ØUTPUT,1.0      $CØNTRØL CØNSTANTS
        1,10,2,3,3,7,4,8            $INTEGERS
        5,1.,6,1.E3,7,1.E-3         $FLØATING PØINT
        8,TEMP,9,ALPHA              $ALPHA-NUMERIC
    END
```

Array Data Block

Array data is exceedingly simple to input. 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-3G 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:

```
Col 12
    1,1.6,2.4,3.8,END
```

Array 1 above 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;

```
Cøl 12
    -2,1.6,2.4,3.8,END
```

Inputing 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.

```
Col 8
        BCD 3ARRAY DATA
            1,1.6,2.4,3.8,END          $FLØATING PØINT NUMBERS
            2,TEMP1,TEMP2,END          $ALPHA-NUMERIC
            3                          $ALPHA-NUMERIC
        BCD 3TEMPERATURE STUDY
            END
            -4,SPACE,100,END           $SPACE ØPTIØN
        END
```

Two types of alpha-numeric input 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 EXECUTIØN, VARIABLES 1, VARIABLES 2 and ØUTPUT CALLS. The operations or instructions called for in these blocks determine the program control. They are preprocessed by CINDA-3G and passed on to the system FØRTRAN compiler as four separate subroutines entitled EXECTN, VARBL1, VARBL2 and ØUTCAL respectively. When the FØRTRAN compilation is successfully completed, control is passed to the EXECTN subroutine which sequentially performs the operations in the same order as input by the user in the EXECUTIØN 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 EXECUTIØN 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 upon them. Network solution subroutines internally call upon VARBL1, VARBL2 and ØUTCAL (see Figure F3, page 3.12). They also use numerous control constants but their individual writeups in Section 5.1 must be consulted in order 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 alpha-numeric 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. The above plus option is only available for arrays. If perhaps a user desired to address the twenty 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 FØRTRAN cards at the start of the execution block, for example,

```
Col 1     7
    F     DIMENSIØN X(100)
    F     NDIM = 100
    F     NTH  = 0
```

The names used must be exactly as shown and in the above would cause a working array of 100 locations to be created. If more or less locations are needed the integer 100 may be changed as desired (both first and second cards). If no working locations are required the cards may be omitted. The program user must check the writeups of subroutines he is using in order to determine if, when and how much of a working array is required.

An F in column one indicates to the preprocessor that the card is FØRTRAN and should be passed on as received. This F option allows the user to program FØRTRAN operations directly into the operations blocks. However, the CINDA-3G arguments listed above are not FØRTRAN compatible with the exception of the control constant names. Therefore, it is recommended that the program user utilize CINDA-3G subroutine calls wherever possible. This is impossible however when logical operations are required. In this case it is recommended that the user place CINDA-3G data values as needed into the available dummy control constant names allowed for that purpose. Then, FØRTRAN logical operations can be utilized with the dummy control constant names as arguments. FORTRAN statement numbers for routing purposes may be placed in columns two through five on any operations cards.

The data field for node, conductor, constant and array data consists of columns twelve through eighty. However, the data field of operations cards ends with column seventy-two. In a manner of speaking, a CINDA-3G 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 paranthesis. The left paranthesis is accepted as a comma while the right paranthesis is accepted as a comma followed by a data END. This allows what would have been this

```
Col 12
    ADD,K1,K2,K3,END
```

to be more esthetically formatted as this

```
Col 12
    ADD(K1,K2,K3)
```

which is almost identical to a FØRTRAN subroutine call.

<u>Execution Operations Block</u>

An execution operation block might be as follows:

```
Col 1      8   12
           BCD 3EXECUTIØN
    F      DIMENSION  X(25)
    F      NDIM=25
    F      NTH=0
    F   10 TIMEND=TIMEND+1.0
               CNFRWD               $EXPLICIT FØRWARD DIFFERENCING
               STFSEP(T20,TTEST)    $PLACE T10 INTØ DUMMY CC
    F      IF(TTEST.LE.100.) GØ TØ 10
           END
```

The above indicates a transient thermal problem in which the user desires to terminate the analysis when the temperature at node 20 exceeds one hundred degrees. The problem must have been fairly small because only twenty five working locations were dimensioned and CNFRWD requires one per node. It does demonstrate the use of both CINDA-3G calls and FØRTRAN operations and that control constants are referred to by name in either. Another example might be

```
Col 1     8   12
          BCD 3EXECUTIØN
    F     DIMENSIØN  X(500)
    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 pseudo-compute sequence requirements and the program would be terminated with an appropriate error message. A further example might be:

```
Col 8   12
    BCD 3EXECUTIØN
        INVRSE(A1,A2)           $SEE MATRIX SUBRØUTINE
        MULT(A2,A3)             $WRITEUPS FØR ØPERATIØNS
        LIST(A2,K1,17)          $PERFØRMED
        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 has been dimensioned. Furthermore, no reference, direct or indirect, is made to VARBL1, VARBL2 or ØUTCAL 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 input.

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 automates overlay, tape 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 non changing 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 post-interpolation multiplication for units conversion only could be prescaled, thereby deleting the multiplication process. Complex functions of a single independent variable requiring several interpolation values which are then combined in a multiplicative fashion can be precalculated versus the 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 Figure F3). 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-3G preprocessor. The user must specify any addit onal 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:

```
Col 1     8   12
          BCD 3VARIABLES 1
              STFSEP(10.0,Q17)                  $CØNSTANT IMPRESSED SØURCE
              D1DEG1(TIMEM,A8,Q19)              $TIME VARYING SØURCE
              D2D1WM(T18,TIMEM,A19,7.63,G18)    $BIVARIATE FUNCTION
    F     TTEST=11.6
    F     IF(TIMEN.GT.10.)  TTEST=0.0
              STFSEP(TTEST,Q27)                 $VARIABLE SØURCE
          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 FØRTRAN 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 "stair-case" 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:

```
Col 1     8   12
          BCD 3VARIABLES 1
              BLDARY(A12+1,T1,T7,T3,T4)      $CØNSTRUCT VECTØR
              D1DEG1(T7,A19,A13+2)           $INTERPØLATIØN
              IRRADE(A7,A13,A10,A12)         $IR RADIØSITY EXPLICIT
              BRKARY(A12+1,Q1,Q7,Q3,Q4)      $DISTRIBUTE Q RATES
              D1D1WM(TIMEM,A9,0.35,TTEST)    $INTERPØLATE
              ADD(TTEST,Q1,Q1)               $ADD TWØ RATES
          END
```

The first call above 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 absorbtivity. 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 powered components, radiation depositation, 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

In regards to the network solution, the Variables 1 operations may be thought of as pre-solution operations and the Variables 2 operations as post-solution 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 just solved network. He may meter and integrate flow rates, make corrections in order to account for material phase changes or compare just calculated answers with test data in order to derive emperical relationships. A simple Variables 2 operations block might be as follows:

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

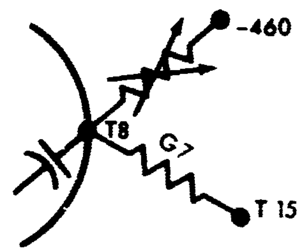
The first call measures the heat flow from node one to node two through regular conductor one and stores the result in constant location one. The second call performs a simple integration with respect to time and sums the result into constants location two. The third call measures heat flow through a 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:

```
Col 8   12
    BCD 3VARIABLES 2
        ABLATS(A1,1.76,K8,A7,T15,C15)       $ABLATIVE ØN NØDE 15
    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 1-D 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:

```
Col 1   5   8   12
            BCD 3VARIABLES 2
                D1DEG1(TIMEN,A10,K8)        $GET TEST TEMPERATURE
                SUB(T8,K8,TTEST)            $ØBTAIN TEMP DIFFERENCE
    F       IF(TTEST.LE.2.0) GØ TØ 10
                MLTPLY(G7,0.99,G7)          $REDUCE CØNDUCTANCE
        5       STFSEP(-1.0,BACKUP)         $SET BACKUP NØN-ZERØ
    F       GØ TØ 20
    F  10   IF(TTEST.GE.-2.0) GØ TØ 15
                MLTPLY(G7,1.01,G7)          $INCREASE CØNDUCTANCE
    F       GØ TØ 5
       15       QMETER(T8,T15,K9)
                QINTEG(K9,DTIMEU,K10)
    F  20   CØNTINUE
            END
```

This corresponds to a portion of a network as follows:



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 seven which will yield a calculated temperature at node eight that is within ±2.0 degrees of the test history. The above Variables 2 operations will attempt to modify conductor seven so that it will meet the constraints on temperature eight. It is quite "brute-force" and unsophisticated. However, the corrector operations are at the descretion 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 seven would satisfy the constraints. The end result of such a study would be to produce a plot of conductance seven versus time which could be used to derive an emperical relationship with other parameters. Too wide a tolerance would cause the plot to resemble a stair-case function. Please note that either condition being unsatisfied causes control constant BACKUP to be set non-zero and the iteration to be redone with the corrected conductor seven 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 Figure F3) so it is only logical that it would primarily contain instructions for outputing information. There is a variety of output subroutines offering the user several format options. A very simple Output Calls block would be as follows:

```
Col 8   12
    BCD 3ØUTPUT 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 versus actual node numbers is listed. By utilizing it a user can correlate the relative node temperatures with his actual numbers.

In addition to the various subroutines for printing output, there are several plotting subroutines available. However, the plotting subroutines require that the information to be plotted exist as arrays. In order to plot transient temperatures versus time it is necessary for the user to store the information until the transient is completed and then perform plotting. The operations to do this could be as follows:

```
Col 8   12
    BCD 3ØUTPUT CALLS
        PRNTMP
        INDEX(K10,1)
        STØARY(K10,A1,TIMEN)
        STØARY(K10,A2,T1)
    END
```

The Output Calls will be performed at problem start time and on the output interval until problem stop time is reached. A 100 minute transient with an output interval of 5 minutes would cause the Output Calls operations to be performed 21 times. With constant ten initially at zero, the INDEX call will add an integer one to it each time it is performed. The STØARY call causes the third arguments (TIMEN and T1) to be stored into the $K10^{th}$ location of array one and two respectively. Therefore, A1 and A2 must contain at least as many data locations as required to accommodate the STØARY operations. When the transient analysis is completed, A1 and A2 contain array data suitable for plotting or printing in a columnar format. Such operations are easily called for in the Execution Operations Block immediately following the network solution call.

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

```
Col 8   12
    BCD 3END ØF DATA
```

Parameter Runs

Parametric analysis which do not involve network or 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 preceeding discussion with the exception of conductors. When specifying new conductances the adjoining node information is deleted; only the conductor number and value are required.

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 input is referred to as the original problem. Any and all INITIAL parameter runs refer to it exactly as it was input. The FINAL parameter run refers to the just completed problem 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-3G control cards necessary to specify a parameter run as follows:

```
Col 8   12
    BCD 3INITIAL PARAMETERS
 or BCD 3FINAL PARAMETERS
    END
    BCD 3NØDE DATA
    END
    BCD 3CØNDUCTØR DATA
    END
    BCD 3CØNSTANTS DATA
    END
    BCD 3ARRAY DATA
    END
```

The parameter run decks are inserted in the problem data deck immediately preceeding the BCD 3END ØF DATA card. After the BCD parameter card, the user may insert additional BCD data to replace the original problem output page heading. When changing an array, the entire new array must be input and be exactly the length of its original. Parameter runs conserve machine time mainly due to not having to reform the pseudo-compute sequence. If a user desires, he may accomplish FINAL parameter runs by calling the network execution subroutine twice in the EXECUTIØN block and inserting the necessary calls to modify data values between them.

Store and Recall Problem Options

The capability to store complete problems on and recall them from magnetic tape is a useful feature of CINDA-3G. While the parameter run capability is useful for performing parametric analysis in the same run, the store and recall capability allows an indefinite time lapse between parametric analysis. 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:

```
Col 12
    STØREP(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 the user must insure that each activation references a unique name. It is up to the user to insure that the stored problem tapes have been mounted with the "write" ring in, are properly positioned and that the computer operator has been instructed to save the tapes. The user should check Section V, Control Cards and Deck Setup, to determine which tapes his problem is being stored on and the control cards if any for assigning it within the system.

The recall problem feature is a CINDA-3G preprocessor option which is activated by the following card:

```
Col 1          13
    RECALL     AAAAAA
```

where AAAAAA is the alphanumeric identification name of the stored problem. This single card replaces the blank card preceeding the problem data deck and must be followed by initial parameter and block data change cards exactly as shown for parameter runs, including the first BCD 3 parameter and END cards and also the BCD 3END ØF DATA card. 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 then control is passed to the first subroutine call in the EXECUTIØN block. The user must remember that the recalled problem contains the STØREP call. The user is again advised to consult Section V for the tape unit designations, control card requirements and operator instructions necessary for mounting the stored problem tapes.

SECTION V

CONTROL CARDS AND DECK SETUP

UNIVAC-1108 Deck Setup NASA Houston

The EXEC-II, CUR and FØRTRAN V systems software for the UNIVAC-1108 are well suited for operation of the CINDA-3G program. The two portions of the program, Preprocessor and Variables, are contained in binary on magnetic tape as files one and two respectively. The user must instruct the operator to mount the tape on drive G. The ∇ symbol indicates a seven and eight punch in the card column. The deck setup is as follows:

```
Col 1    6      12
    ∇    RUN
    ∇    ASG    G=RXXXXX
    ∇    ASG    J=J,K=K
    ∇    XQT    CUR
         TRW    G
         IN     G
    ∇    XQT    CØ0045
                          ← blank card unless RECALL
                          ← problem data deck through END ØF DATA
    ∇    XQT    CUR
         ERS
         IN     G
         TRI    G
    ∇N   FØR,K LINKO
    ∇N   FØR,K EXECTN
    ∇N   FØR,K VARBL1
    ∇N   FØR,K VARBL2
    ∇N   FØR,K ØUTCAL
                          ←"load and go" subroutines if any, with ∇FØR
    ∇    XQT   LINKO
    ∇    EØF
```

It is recommended that the CINDA-3G user acquaint himself with the CUR operating system and the basics of FØRTRAN V, in particular, logical IF statements.

The operator instruction ticket accompanying the job must have the RXXXXX designated as input on G and request K as a scratch tape. This job's compatable with all the various 1108 systems at MSC and is required to be run under the FØRTRAN V system.

NOTE: The latest CINDA-3G reel number may be obtained from R. L. Dotts, phone 483-2378.

UNIVAC-1108 Tape Usage NASA Houston

UNIT DESIGNATION	FORTRAN NUMBER	PROGRAM VARIABLE	FUNCTION
B	2	LUT3	Copy of original problem data.
C	3	LUT4	Parameter change data.*
D	4	LUT1	Data number definitions.
F	8	LUT2	NA-NB pairs; data number definitions from parameter changes.
G	9	----	CUR(IN) CINDA-3G Master tape.
H	10	----	CUR(ØUT), if any.*
J	12	LB3D	Data tape (original problem and all parameter changes).
K	13	LB4P	Program tape (contains generated Fortran routines; LINKO,EXECTN, VARBL1,VARBL2,ØUTCAL)
M	15	LUT7	Variables 1 calls generated from node and conductor data blocks.*
N	16	INTERN	Data conversion scratch tape
	17		System plot tape (Restricted use)
R	21	----	Problem recall data tape.*
S	22	----	Problem store data tape.*
Reread	30	KRR	Fortran reread unit.

*These tapes need not be assigned if the particular options are not used. The STØREP option requires assigning and saving tapes 4 and 22. The RECALL options requires assigning and mounting the above tapes on 4 and 21 respectively.

UNIVAC-1108 Deck Setup NASA Michoud

The EXEC-II, CUR and FØRTRAN V systems software for the UNIVAC-1108 are well suited for operation of the CINDA-3G program. The two portions of the program, Preprocessor and Variables, are contained in binary on magnetic tape as files one and two respectively. The user must instruct the operator to mount the tape on drive F. The ∇ symbol indicates a seven and eight punch in the card column. The deck setup is as follows:

```
Col 1   6     12
    ∇   RUN
    ∇   TPR
    ∇   ASG   D=D,F=F,I=I,K=K, L = L
    ∇   XQT   CUR
        TRW   F
        IN    F
    ∇   XQT   CØ0045
                   ←blank card unless RECALL
                   ←problem data deck through END ØF DATA
    ∇   XQT   CUR
        ERS
        IN    F
        TRI   F
    ∇N  FØR,K LINKO
    ∇N  FØR,K EXECTN
    ∇N  FØR,K VARBL1
    ∇N  FØR,K VARBL2
    ∇N  FØR,K ØUTCAL
                   ←"load and go" subroutines if any, with∇ FØR cards
    ∇N  XQT   LINKO
    ∇   FIN
```

It is recommended that the CINDA-3G user acquaint himself with the CUR operating system and the basics of FØRTRAN V, in particular, logical IF statements.

The operator instruction ticket accompanying the job must have the RXXXXX designated as input on F. The job is compatable with all the various 1108 systems at Michoud and is required to be run under the FØRTRAN V system.

NOTE: The latest CINDA-3G reel number may be obtained from R. L. Thompson, phone 255-6448.

UNIVAC-1108 Tape Usage NASA Michoud

UNIT DESIGNATION	FORTRAN NUMBER	PROGRAM VARIABLE	FUNCTION
DRUM	2	LUT3	Copy of original problem data.
DRUM	3	LUT4	Parameter change data.
D	4	LUT1	Data number definitions.
DRUM	8	LUT2	NA-NB pairs; data number definitions from parameter changes.
F	9	----	CUR(IN) CINDA-3G Master tape.
G	10	----	CUR(ØUT), if any. *
I	12	LB3D	Data tape (original problem and all parameter changes).
J	13	----	System plot tape (restricted use).
K	14	LB4P	Program tape (contains generated Fortran routines; LINKO,EXECTN, VARBL1,VARBL2,ØUTCAL)
L	15	LUT7	Variables 1 calls generated from node and conductor data blocks.
DRUM	27	INTERN	Data conversion scratch tape.
R	21	----	Problem recall data tape. *
S	22	----	Problem store data tape. *
Reread	0	ERR	Fortran reread unit.

* These tapes need not be assigned if the particular options are not used. The STØREP option requires assigning and saving tapes 4 and 22. The RECALL options requires assigning and mounting the above tapes on 4 and 21 respectively.

Alphabetic Listing of Available Subroutines

NAME	PAGE	NAME	PAGE	NAME	PAGE	NAME	PAGE
AABB	6.5.7	CINDSR	6.1.3	D1D2WM	6.2.3	ENTRØ	INTERN
ABLATS	6.6.4	CINDSS	6.1.1	D1MDG1	6.2.2	EØF	6.4.7
ABØRT		CINSIN	6.3.14	D1MDG2	6.2.3	EXPARY	6.3.16
ACSARY	6.3.15	CINTAN	6.3.14	D1M1DA	6.2.2	EXPNTL	6.3.16
ADARIN	6.3.10	CMPXDV	6.3.18	D1M1MD	6.2.2	FILE	6.5.13
ADD	6.3.3	CMPXMP	6.3.18	D1M1WM	6.2.2	FILTER	
ADDALP	6.5.7	CMPXSR	6.3.17	D1M2DA	6.2.3	FITIT	
ADDARY	6.3.3	CMPYI	6.3.18	D1M2MD	6.2.4	FIX	6.3.1
ADDFIX	6.3.3	CNBACK	6.1.8	D1M2WM	6.2.4	FLIGHT	
ADDINV	6.3.10	CNEXPN	6.1.6	D11CYL	6.2.5	FLIP	6.3.1
AERØ	INTERN	CNFAST	6.1.5	D11DAI	6.2.4	FLØAT	6.3.1
ALPHAA	6.5.7	CNFRWD	6.1.4	D11DIM	6.2.4	FØRM	INTERN
AMAT		CNFWBK	6.1.7	D11MCY	6.2.5	FØRPM	INTERN
ARCCØS	6.3.15	CØLMAX	6.5.2	D11MDA	6.2.2	FRAMEV	6.4.3
ARCSIN	6.3.15	CØLMIN	6.5.2	D11MDI	6.2.4	FRMFAC	INTERN
ARCTAN	6.3.15	CØLMLT	6.5.6	D12CYL	6.2.5	FUDGE	
ARINDV	6.3.10	CØNE		D12MCY	6.2.5	GARZAM	
ARYADD	6.3.3	CØNVEC		D12MDA	6.2.3	GENALP	6.5.1
ARYDIV	6.3.6	CØPY	INTERN	D2DEG1	6.2.8	GENARY	6.3.7
ARYEXP	6.3.16	CØSARY	6.3.14	D2DEG2	6.2.8	GENCØL	6.5.1
ARYINV	6.3.10	CØSD	INTERN	D2D1WM	6.2.8	GENM	INTERN
ARYMNS	6.3.2	CRØSS	INTERN	D2D2WM	6.2.8	GENST	INTERN
ARYMPY	6.3.5	CSGDMP	6.1.5	D2MXD1	6.2.8	GSLØPE	6.2.6
ARYPLS	6.3.2	CSQRI	6.3.17	D2MXD2	6.2.8	HANTIM	INTERN
ARYSTØ	6.3.11	CVQ1HT	6.2.6	D2MX1M	6.2.8	HEDCØL	INTERN
ARYSUB	6.3.4	CVQ1WM	6.2.6	D2MX2M	6.2.8	HØNEYK	
ASNARY	6.3.15	DATEUP	INTERN	D3DEG1	6.2.9	IEXPAN	INTERN
ASSMBL	6.5.6	DA11CY	6.2.5	D3D1WM	6.2.9	INDEX	
ATAND	INTERN	DA11MC	6.2.5	EFACS	6.5.4	INGRAT	
ATNARY	6.3.15	DA12CY	6.2.5	EFASN	6.5.4	INTRFC	6.3.1
BCØNE		DA12MC	6.2.5	EFATN	6.5.4	INVRSE	6.5.8
BENDIT		DECAY	INTERN	EFCØS	6.5.4	IRRADE	6.6.2
BIVLV	6.6.5	DIAG	6.5.6	EFEXP	6.5.5	IRRADI	6.6.2
BKARAD	6.3.7	DISAS	6.5.6	EFFEMS	6.3.21	ITERAT	INTERN
BLDARY	6.3.7	DIVARY	6.3.6	EFFG	6.3.21	ITRATE	INTERN
BRKARY	6.3.7	DIVFIX	6.3.6	EFLØG	6.5.5	JACØBI	6.5.10
BTAB	6.5.7	DIVIDE	6.3.6	EFPØW	6.5.5	JØIN	6.3.12
BVSPDA	6.2.7	DØT	INTERN	EFSIN	6.5.4	KERNEL	
BVSPSA	6.2.7	DØUINT	INTERN	EFSQR	6.5.5	KMAT	
BVTRN1	6.2.7	D1DEG1	6.2.1	EFTAN	6.5.4	LAGRAN	6.2.1
BVTRN2	6.2.7	D1DEG2	6.2.3	ELEADD	6.5.3	LGRNDA	6.2.1
CALL	6.5.13	D1DG1I	6.2.4	ELEDIV	6.5.3	LIMAT	
CAP	INTERN	D1D1DA	6.2.1	ELEINV	6.5.3	LINE	
CDIVI	6.3.18	D1D1IM	6.2.4	ELEMUL	6.5.3	LININT	
CHANGE		D1D1MI	6.2.4	ELESUB	6.5.3	LINRES	
CINCØS	6.3.14	D1D1WM	6.2.2	ENDFIL	6.4.3	LIST	6.5.13
CINDSL	6.1.2	D1D2DA	6.2.3	ENDMØP	6.5.13	LØGE	6.3.16

Alphabetic Listing of Available Subroutines (Continued)

Execution Subroutines

EXECUTION SUBROUTINE NAME: 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 NLØØP) and the relaxation criteria 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 NLØØP 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 over relaxation.

If a series of steady state solutions at various times are desired it can be accomplished by specifying control constants TIMEND and ØUTPUT. ØUTPUT 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 ØUTPUT 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 NLØØP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants TIMEND AND ØUTPUT. Other control constants which are activated or used are; LØØPCT, DRLXCC and/or ARLXCC, TIMEN, TIMEM, TIMEØ, DAMPD, DAMPA, DTIMEU, LINECT and PAGECT. Control constant ØPEITR is checked for output each iteration.

CALLING SEQUENCE: CINDSS

*This subroutine utilizes one dynamic storage core location for each diffusion node.

EXECUTION SUBROUTINE NAME: 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 NLØØP) and the relaxation criteria 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 NLØØP 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 over relaxation.

If a series of steady state solutions at various times are desired it can be accomplished by specifying control constants TIMEND and ØUTPUT. ØUTPUT 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 ØUTPUT 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 NLØØP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants TIMEND and ØUTPUT. Other control constants which are activated or used are; LØØPCT, DRLXCC, and/or ARLXCC, TIMEN, TIMEM, TIMEØ, DAMPD, DAMPA, DTIMEU, LINECT and PAGECT. Control constant ØPEITR 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 NAME: CINDSR

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 re-evaluates 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 criteria. This system energy balance is the difference between all energy into the system and all energy out and is specified as control constant BALENG. CINDSR 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 analysis may be performed and CINDSR 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 NLØØP and BALENG must be specified and greater than zero. Successive steady state solutions can be obtained by specifying control constants TIMEND and ØUTPUT. Other control constants which are activated or used are: LØØPCT, ENGBAL, DRLXCC and/or ARLXCC, TIMEN, TIMEM, TIMEØ, DTIMEU, LINECT and PAGECT. Control constant ØPEITR is checked for output each iteration. Caution: Each radiation conductor must have a unique conductor number.

CALLING SEQUENCE: CINDSR

*This subroutine utilizes 3 dynamic storage core locations for each diffusion and arithmetic node and one more for each radiation conductor.

EXECUTION SUBROUTINE NAME: CNFRWD

PURPOSE: This subroutine performs transient thermal analysis by the explicit forward differencing method. The stability criteria 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 DTMPCC 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 NLØØP and ARLXCA. If the arithmetic node iteration count exceeds NLØØP 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 one time step lag.

RESTRICTIONS: The SPCS option is required and control constants TIMEND and ØUTPUT must be specified. Problem start time if other than zero may be specified as TIMEØ. Other control constants used or activated are: TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, NLØØP, LØØPCT, DAMPA, ARLXCA, ARLXCC, ØPEITR, BACKUP, LINECT and PAGECT.

CALLING SEQUENCE: CNFRWD

*This subroutine utilizes one dynamic storage core location for each diffusion and arithmetic node.

EXECUTION SUBROUTINE NAME: 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 ØUTPUT 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, TIMEØ, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCC, ATMPCC, DAMPA, ARLXCA, ARLXCC, NLØØP, LØØPCT, LINECT and PAGECT.

CALLING SEQUENCE: CNFAST

*This subroutine utilizes one dynamic storage core location for each diffusion node.

EXECUTION SUBROUTINE NAME: CSGDMP

PURPOSE: This subroutine is designed to aid in the checkout of thermal problem data decks. It calls upon Variables 1 and Output Calls and then prints out each relative diffusion node number with the capacitance and CSGMIN value of the node. For each node it identifies the attached conductors by relative conductor number, lists the type and conductance value and the relative number and type of the adjoining node. Either the SPCS or LPCS option may be used. While the LPCS option allows every conductor attached to a node to be identified, the SPCS option only identifies 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.

RESTRICTIONS: This subroutine is generally called in the Execution block and possibly in Variables 2 but not in Variables 1 or Output Calls.

CALLING SEQUENCE: CSGDMP

EXECUTION SUBROUTINE NAME: CNEXPN

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

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

The reader is referred to page 5.1.3 of CCSD TN-AP-66-15 for the derivation. The above equation is unconditionally stable no matter what size time step is taken and 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 NLØØP, ARLXCA and DAMPA exactly the same as CNFRWD.

RESTRICTIONS: The SPCS option is required and control constants TIMEND and ØUTPUT must be specified. Problem start time if other than zero may be specified as TIMEØ. Other control constants used or activated are: TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, ARLXCA, ARLXCC, DAMPA, ØPEITR, BACKUP, LINECT and PAGECT.

CALLING SEQUENCE: CNEXPN

*This subroutine utilizes one dynamic storage core location for each diffusion and arithmetic node.

EXECUTION SUBROUTINE NAME: 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 ØPEITR 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, ØUTPUT, DTIMEI, NLØØP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used or activated are: TIMEN, TIMEØ, TIMEM, CSGMIN, DTIMEU, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LØØPCT, BACKUP, ØPEITR, 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 NAME: 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 NLØØP and DRLXCA and/or ARLXCA. If the number of iterations during a time step exceeds NLØØP a message is printed but the problem proceeds.

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 ØPEITR 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, ØUTPUT, DTIMEI, NLØØP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used on activated are: TIMEN, TIMEØ, TIMEM, CSGMIN, DTIMEV, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LØØPCT, BACUP, ØPEITR, 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

SUBROUTINE NAMES: 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

$$Y = Pn\ (X) = \sum_{k=0}^{n} Yk \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{X - Xi}{Xk - Xi} \quad , \quad n = 1,2,3,\ldots,50\text{max.}$$

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)

and singlet arrays are formed as follows:

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

SUBROUTINE NAMES: 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 NAMES: 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 NAMES: 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 NAMES: 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 NAMES: 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 NAME: 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 NAMES: 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 NAMES: 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 NAMES: 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 NAMES: 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 NAMES: 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 NAMES: 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 lets say R(X) and S(X). Additional arguments of Y, Z and T complete the data values. The post interpolation calculations are respectively:

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

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 NAMES: GSLØPE

PURPOSE:

This subroutine will generate a slope array so that point slope interplation subroutines can be used instead of standard linear interpolation subroutines. The user must address two singlet type 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:

```
GSLØPE(AX(IC),AY(IC),AS(IC))
```

SUBROUTINE NAMES: 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, full and input 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 NAMES: 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 the X array can be reached 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))
BVSPDA(Y,BA(IC),AXZ(IC))
```

SUBROUTINE NAMES: BVTRN1 or BVTRN2

PURPOSE: These subroutines construct a bivariate array of Y's versus X and Z from an input bivariate array of Z's versus X and Y. BVTRN1 should be used when the Z values increase with increasing Y values and BVTRN2 when the Z values decrease with increasing Y values.

RESTRICTIONS: The user must appropriately place the X and Z values and spaces for Y's in the array to be constructed. These subroutines will fill in the Y spaces. The new array can differ in size from the old. Subroutine D1DEG1 is called and its linear extrapolation feature applies.

CALLING SEQUENCE:

```
   BVTRN1(BAØ(IC),BAN(IC))
or BVTRN2(BAØ(IC),BAN(IC))
```

SUBROUTINE NAMES: D2DEG1 or D2DEG2

PURPOSE: These subroutines perform bivariate linear and parabolic interpolation respectively. The arrays must be formated as shown for Bivariate Array Format.

RESTRICTIONS:

For D2DEG1 , $N \geq 2, M \geq 2$
for D2DEG2 , $N \geq 3, M \geq 3$
See page 6.2.7 array format

CALLING SEQUENCE:

```
   D2DEG1(X,Y,BA(IC),Z)
or D2DEG2(X,Y,BA(IC),Z)
```

SUBROUTINE NAMES: 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 NAMES: 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 NAMES: 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 $\quad$ 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 input 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 NAMES: $\quad$ D3DEG1 or D3D1WM

PURPOSE:

These subroutines perform trivariate linear intepolation. The interpolation array must be constructed as shown for 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)
D3D1WM(X,Y,Z,TA(IC),F,T)

SUBROUTINE NAMES: 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 input as a constant. The following mnemonic codes in the nodal block

```
Col 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

```
Col 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 later 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 NAMES: VARGSM or VARGCM or VARG1 or VARG2

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 conductor data block. They are similar to the preceeding 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. The 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

SUBROUTINE NAMES: FLØAT or FIX or INTRFC

PURPOSE:

Subroutine FLØAT 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 as a floating point number. Their respective operations are:

X = N
or N = X
or N = X
Y = N
F = X-Y

RESTRICTIONS:

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

CALLING SEQUENCE:

FLØAT(N,X)
or FIX(X,N)
or INTRFC(X,N,F)

SUBROUTINE NAMES: 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:

A(i) = B(i) , i = 1,N
or A(N-i+1) = B(i) , i = 1,N
or A(i)new = A(N-i+2) old , i = 2,N+1

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))
FLIP(A(IC))

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

SUBROUTINE NAMES: 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 literial integer or the address of a location containing an integer and A(DV) addresses the first data value in the array.

SUBROUTINE NAMES: 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 literial integer or the address of a location containing an interger and A(DV) addresses the first data value in the array.

SUBROUTINE NAMES: ADD or ADDFIX

PURPOSE:

To sum a variable number of floating point or integer numbers respectively.

$$S = \Sigma X_i \quad , \quad i = 1,2,3,\ldots,N \quad , \quad N \geq 2$$

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 NAMES: 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:

$$A_i = B_i + C_i \quad , \quad i = 1,N$$
$$\text{or } A_i = B_i + C \quad , \quad i = 1,N$$

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))
```

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

SUBROUTINE NAMES: SUB or SUBFIX

PURPOSE:

To subtract a variable number of floating point or integer numbers respectively.

$$R = Y - \Sigma Xi \;,\; i = 1,2,3,\ldots,N \;,\; N \geq 1$$

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 NAMES: 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:

$$Ai = Bi - Ci \quad , \quad i = 1,N$$
$$\text{or } Ai = Bi - C \quad , \quad i = 1,N$$

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))
```

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

SUBROUTINE NAMES: MLTPLY or MPYFIX

PURPOSE:

To multiply a variable number of floating point or integer numbers respectively.

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

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 NAMES: 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:

$$A_i = B_i * C_i \quad , \quad i = 1,N$$
$$\text{or } A_i = B_i * C \quad , \quad i = 1,N$$

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))
```

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

SUBROUTINE NAMES: DIVIDE or DIVFIX

PURPOSE:

To perform a division of floating point or integer numbers respectively.

$$Q = Y/\Sigma Xi \ , \ i = 1,2,3,\ldots,N \ , \ N \geq 1$$

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 NAMES: 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:

$$Ai = Bi/Ci \ , \ i = 1,N$$
$$\text{or } Ai = Bi/C \ , \ i = 1,N$$

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))
```

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

SUBROUTINE NAME: 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 NAME: BLDARY

PURPOSE:

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

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

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 NAME: 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 whats in the location. Respective operations are:

$$X_i = A_i \quad , \quad i = 1,N$$
$$\text{or} \quad X_i = X_i + A_i \quad , \quad i = 1,N$$

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 NAMES: 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:

$$Xi = Y \quad , \quad i = 1,2,3,\ldots,N$$
$$\text{or } Xi = Y*Zi \quad , \quad i = 1,2,3,\ldots,N$$

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 NAMES: 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:

$$Ai = A1 \quad , \quad i = 2,N$$
$$\text{or } Ai = B \quad , \quad i = 1,N$$

RESTRICTIONS:

N must be an integer but the constant data value may be integer, floating point or alpha-numeric.

CALLING SEQUENCE:

```
   STFSEQ(A(DV),N)
or STFSQS(B,N,A(DV))
```

SUBROUTINE NAME: SUMARY

PURPOSE:

To sum an array of floating point values:

$$S = \Sigma Ai \quad , \quad i = 1,N$$

RESTRICTIONS:

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

CALLING SEQUENCE:

SUMARY(N,A(DV),S)

SUBROUTINE NAMES: 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

$$D = \left| Ai - Bi \right|_{max} \quad , \quad i = 1,N$$

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 NAMES: 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:

$$Ai = 1.0/Ai \quad , \quad i = 1,N$$
$$\text{or } Ai = B/Ci \quad , \quad i = 1,N$$

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 overlayed into the input array area.

SUBROUTINE NAMES: ADDINV or ADARIN

PURPOSE:

Subroutine ADDINV will calculate one over the sun 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:

$$Y = 1.0/(1/X1+1./X2+\ldots+1./XN) \quad , \quad N \geq 2$$
$$\text{or } Y = 1.0/\Sigma(1./Xi) \quad , \quad i = 1,N$$

RESTRICTIONS:

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

CALLING SEQUENCE:

```
   ADDINV(X1,X2,X3,...XN,Y)
or ADARIN(N,X(DV),Y)
```

SUBROUTINE NAMES: STØARY or ARYSTØ

PURPOSE:

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

$$A_i = X \quad , \quad i = N \quad , \quad N > 0$$
$$\text{or} \quad X = A_i \quad , \quad i = N \quad , \quad N > 0$$

RESTRICTIONS:

The values may be anything but N must be an integer.

CALLING SEQUENCE: STØARY(N,X,A(DV))
or ARYSTØ(N,X,A(DV))

SUBROUTINE NAMES: SCLDEP or SCLIND

PURPOSE:

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

$$A_i = X*A_i \quad , \quad i = 3,5,7,\ldots,N+1$$
$$\text{or} \quad A_i = X*A_i \quad , \quad i = 2,4,6,\ldots,N$$

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 NAMES: 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 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 phenomina. Their respective operations are:

$$A_i = A_{i+1} \quad , \quad i = 2,N$$
$$\text{and} \quad A_i = X \quad , \quad i = N+1$$
$$\text{or} \quad A_i = A_{i+2} \quad , \quad i = 2,N-1$$
$$\text{and} \quad A_i = X \text{ and } A_{i+1} = Y \quad , \quad i = N$$

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))
SLDARD(X,Y,A(IC))

SUBROUTINE NAMES: SPLIT or JØIN

PURPOSE:

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

$$B_i = A_{2i-1} \quad , \quad i = 1,N$$
$$C_i = A_{2i} \quad , \quad i = 1,N$$
or
$$A_{2i-1} = B_i \quad , \quad i = 1,N$$
$$A_{2i} = C_i \quad , \quad i = 1,N$$

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 JØIN(N,B(DV),C(DV),A(DV))

SUBROUTINE NAME: SPREAD

PURPOSE:

This subroutine applies interpolation subroutine D1D1DA to two singlet arrays to obtain an array of dependent variables versus 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 JØIN.

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 Ø for output.

CALLING SEQUENCE: SPREAD(N,X(IC),Y(IC),XI(DV),YØ(DV))

SUBROUTINE NAMES: QMETER or RDTNQS or QMTRI or QFØRCE

PURPOSE:

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

$$\begin{aligned} A &= B*(C-D) \\ \text{or } A &= B*((C+460.)^4-(D+460.)^4) \\ \text{or } Ai &= Bi*(Ci-Ci+1) \quad , \quad i = 1,N \\ \text{or } Ai &= Bi*(Ci-Di) \quad , \quad i = 1,N \end{aligned}$$

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 QFØRCE(N,C(DV),D(DV),B(DV),A(DV))
```

SUBROUTINE NAMES: QINTEG or QINTGI

PURPOSE:

These subroutines perform a simple integration. They are useful for obtai the integrals of flow rates calculated by QMETER, RDTNQS, QMTRI or QFØRCE Their respective operations are:

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

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 NAMES: CINSIN or SINARY

PURPOSE:

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

$$A = \text{sine } (B)$$
$$\text{or} \quad Ai = \text{sine } (Bi) \quad , \quad i = 1,N$$

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 NAMES: CINCØS or CØSARY

PURPOSE:

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

$$A = \text{cosine } (B)$$
$$\text{or} \quad Ai = \text{cosine } (Bi) \quad , \quad i = 1,N$$

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: CINCØS(B,A)
or CØSARY(N,B(DV),A(DV))

SUBROUTINE NAMES: CINTAN or TANARY

PURPOSE:

These subroutines obtain the tagent function of an angle or array of angles. Their respective operations are:

$$A = \text{tangent } (B)$$
$$\text{or} \quad Ai = \text{tangent } (Bi) \quad , \quad i = 1,N$$

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 NAMES: ARCSIN or ASNARY

PURPOSE:

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

$$A = \text{sine}^{-1}(B)$$
$$\text{or} \quad A_i = \text{sine}^{-1}(B_i) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following 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 NAMES: ARCCØS or ACSARY

PURPOSE:

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

$$A = \text{cosine}^{-1}(B)$$
$$\text{or} \quad A_i = \text{cosine}^{-1}(B_i) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following 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: ARCCØS(B,A)
or ACSARY(N,B(DV),A(DV))

SUBROUTINE NAMES: ARCTAN or ATNARY

PURPOSE:

Thses subroutines obtain the angle corresponding to a tangent function value of array of tangent values. Their respective operations are:

$$A = \text{tangent}^{-1}(B)$$
$$\text{or} \quad A_i = \text{tangent}^{-1}(B_i) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following limits, . 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 NAMES: EXPNTL or ARYEXP or EXPARY

PURPOSE:

These subroutines perform an exponential operation. Their respective operations are:

$$A = B^{C}$$
$$\text{or} \quad Ai = Bi^{C} \quad , \quad I = 1,N$$
$$\text{or} \quad Ai = Bi^{C_i} \quad , \quad I = 1,N$$

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 NAMES: LØGT or LØGTAR

PURPOSE:

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

$$A = \log_{10}(B)$$
$$\text{or} \quad Ai = \log_{10}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

All values must be positive floating point numbers except N which must be an integer.

CALLING SEQUENCE: LØGT(B,A)
or LØGTAR(N,B(DV),A(DV))

SUBROUTINE NAMES: LØGE or LØGEAR

PURPOSE:

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

$$A = \log_{e}(B)$$
$$\text{or} \quad Ai = \log_{e}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

All values must be positive floating point numbers except N which must be an integer.

CALLING SEQUENCE: LØGE(B,A)
or LØGEAR(N,B(DV),A(DV))

SUBROUTINE NAMES: SQRØØT or SQRØTI

PURPOSE:

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

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

$$\text{or} \quad Ai = +\sqrt{Bi} \quad , \quad i = 1,N$$

RESTRICTIONS:

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

CALLING SEQUENCE: SQRØØT(B,A)
or SQRØTI(N,B(DV),A(DV))

SUBROUTINE NAMES: 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:

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

$$\text{or} \quad Aj + iBj = \sqrt{Cj + iDj} \quad , \quad j = 1,N$$

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 NAMES: CMPXMP or CMPYI

PURPOSE:

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

$$A + iB = (C + iD)*(E + iF) \quad , \quad i = \sqrt{-1}$$
$$\text{or } Aj + iBj = (Cj + iDj)*(Ej + iFj) \quad , \quad j = 1,N$$

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 NAMES: CMPXDV or CDIVI

PURPOSE:

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

$$A + iB = (C + iD)/(E + iF) \quad , \quad i = \sqrt{-1}$$
$$\text{or } Aj + iBj = (Cj + iDj)/(Ej + iFj) \quad , \quad j = 1,N$$

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 NAMES: 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:

$$f(X) = A1+A2*X+A3*X^2+A4*X^3 = 0.0 \pm T$$

or

$$g(X) = A1+A2*X+A3*X^2+A4*X^3+A5*X^4 = 0.0 \pm T$$

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 NAMES: PLYNML or PLYARY

PURPOSE:

These subroutines calculate Y from the following polynomial equation:

$$Y = A1+A2*X+A3*X^2+A4*X^3+ \dots +AN*X^{N-1}$$

The number of terms is variable but all the A coefficients must be input 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 NAMES: 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:

A = DX*(Y1+4Y2+2Y3+4Y4+...+YN)/3
or A = DX*(Y1+2Y2+2Y3+2Y4+...+YN)/2

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 NAME: 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:

$$A = \frac{1}{2}\Sigma(Xi-Xi-1)*(Yi+Yi-1) \quad , \quad i = 2,N$$

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 NAMES: 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:

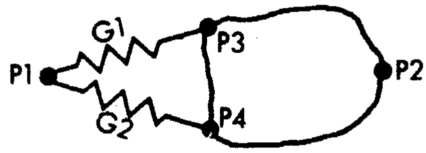
P2 = P1-W/G
or P1(i + 1) = P1(i)-W/G(i), i = 1,2,3,···N

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)
SPRESS(N,P1(DV),W,G(DV))

SUBROUTINE NAME: EFFG

PURPOSE: For a pressure network of the following type



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 NAME: 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

SUBROUTINE NAMES: 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 - 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 NAMES: 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 preceeded or labeled by a six 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 NAME: 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 six 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 would look as follows:

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

SUBROUTINE NAME: 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 two line heading of 12 alphanumeric characters each. The two 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),...)

SC-4020 PLØTTING SUBRØUTINES AND SYMBØLS

CINDA-3G contains an integrated package of SC-4020 plotting subroutines that may be used to produce a variety of plotted output. These plots are output by the computer onto magnetic tape which when processed by the SC-4020 yields the plots on 35 mm. film which may then be processed to produce Zerox or some other type of hard copy. The plotting symbols (IS) available are as follows:

Decimal Integer	Plot Char.	Decimal Integer	Plot Char.	Decimal Integer	Plot Char.	Decimal Integer	Plot Char.
0	0	16	+	32	-	48	
1	1	17	A	33	J	49	/
2	2	18	B	34	K	50	S
3	3	19	C	35	L	51	T
4	4	20	D	36	M	52	U
5	5	21	E	37	N	53	V
6	6	22	F	38	Ø	54	W
7	7	23	G	39	P	55	X
8	8	24	H	40	Q	56	Y
9	9	25	I	41	R	57	Z
10	∂	26	π	42	.	58	°
11	=	27	.	43	$	59	,
12	"	28	)	44	*	60	(
13	'	29	β	45	γ	61	$\int$
14	δ	30	I	46	~	62	Σ
15	α	31	?	47	d	63	□

SUBROUTINE NAMES: ØPNPLT or PLTND or ENDFIL or FRAMEV

PURPØSE: These subroutines perform the following operations:

ØPNPLT This call rewinds the plot output tape. It should be the first plot call within any job and appear only once. A "job" may consist of one or more stacked problem runs.

PLTND This call empties the plot buffers. It should appear in every problem run within a job and after all the quick plot calls.

ENDFIL This call writes an end of file on the plot output tape. It should be used only once in a job as the last call of the last problem run in the job.

FRAMEV The plot frames produced on 35 mm. film are quite close together. This call places a blank frame on the film thereby allowing the good frames to be cut large enough for mounting as projector slides.

RESTRICTIØNS: Check Section V, Control Cards and Deck Setup, for tape usage and control cards necessary.

CALLING SEQUENCE:

```
   ØPNPLT
or PLTND
or ENDFIL
or FRAMEV(3)
```

NOTE: These subroutines are not required on the UNIVAC-1108 system.

SUBROUTINE NAMES: PLØTX1 or PLØTX2

PURPOSE:

These are FØRTRAN coded quick plot subroutines for the SC-4020 which call upon a large package of undocumented subroutines specifically for the SC-4020. They will produce up to three XY graphs per frame and several variables may be plotted per graph. A suitable grid will be drawn with certain lines emphasized. The grid lines will have reasonable numerical indicia and a centered title will be printed for both axes and at the top of the graph.

PLØTX1 computes the minimum and maximum values of the stored X and Y arrays to be plotted and calls upon PLØTX2 which uses the values as grid limits for the graph. The user may set the grid limits by calling PLØTX2 directly. The X, Y and top titles (XT, YT and TT respectively) must consist of 12 alphanumeric words of six characters each.

RESTRICTIONS:

The user should consult Section 5, Control Cards and Deck Setup, to check tape designation requirements. The X and Y values must be floating point numbers.

CALLING SEQUENCE:

```
   PLØTX1(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV))
or PLØTX2(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV))
```

where N is the integer number of graphs per frame (1, 2 or 3). If negative, the frame is advanced and a new grid produced; if zero, the grid from the previous plot call is used and if positive, the second or third graph for the frame is produced.

- XL is the floating point X axis left limit
- XR is the floating point X axis right limit
- YB is the floating point Y axis bottom limit
- YT is the floating point Y axis top limit
- IS is an integer identifying the plotting symbol to be used
- TX is the address of the X title
- TY is the address of the Y title
- TT is the address of the top title
- NP is the integer number of XY values or points to be plotted, if negative the points will be connected by straight lines.
- AX is the address of the X array
- AY is the address of the Y array

SUBROUTINE NAMES: PLØTX3 or PLØTX4

PURPOSE:

These subroutines are similar to PLØTX1 and PLØTX2 but have 6 additional arguments which allow the user to modify the grid as desired.

RESTRICTIONS:

See PLØTX1 and PLØTX2.

CALLING SEQUENCE:

PLØTX3(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV)
DX,DY,L,M,I,J)
or PLØTX4(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),NP
AX(DV),AY(DV),DX,DY,L,M,I,J)

where the arguments are identical to PLØTX1 and PLØTX2 except for

DX,DY these floating point values are used for spacing the grid lines which are centered on the zero values. If zero, no grid lines will be drawn.

L,M these integers cause every L^{th} vertical and M^{th} horizontal grid line to be redrawn for emphasis. If zero, no grid lines will be emphasized. If negative, a square grid will be produced.

I,J these integers cause every I^{th} vertical and J^{th} horizontal grid line to be labeled with its value. If zero, no grid lines will be labeled. If negative, the labels will be placed outside the grid, otherwise they will appear on the zero axis.

SUBROUTINE NAMES: PLØTL1 or PLØTL2

PURPOSE:

These subroutines are similar to PLØTX1 and PLØTX2 but produce log-semi, log-log or semi-log plots. The arguments are identical to PLØTX1 and PLØTX2 except for one additional one which sets the plotting mode.

RESTRICTIONS:

See PLØTX1 and PLØTX2. No limit may be zero.

CALLING SEQUENCE:

PLØTL1(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),
AY(DV),LM)
or PLØTL2(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),
NP,AX(DV),AY(DV),LM)

where the arguments are identical to PLØTX1 and PLØTX2 except for LM which is an integer for identifying the plotting mode as follows:

LM < 0 produced plot will be log X versus linear Y
LM = 0 produced plot will be log X versus log Y
LM > 0 produced plot will be linear X versus log Y.

SUBROUTINE NAME: 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 FØRTRAN FØRMAT which has been input as an array, including the outer parenthesis but deleting the word FØRMAT. 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. The output is written onto logical tape 15, the user must provide the necessary control cards and processing information for the operator.

RESTRICTIONS: The user should check Section V for the appropriate control card requirements. Punched output is written on logical tape 15, operator processing instructions should be supplied.

CALLING SEQUENCE: PUNCHA(F(DV), A(DV),N)

SUBROUTINE NAME: PNCHMA

PURPOSE: This subroutine is similar to PUNCHA, but up to 10 equal length arrays of data values may be punched. Again the first argument must reference a FØRTRAN FØRMAT which has been input as an array, including the outer parenthesis, but deleting the word FØRMAT. The integer number of data values in an array must be supplied as the second argument N. The array starting locations then follow as arguments three up to twelve. The first values in each array is punched, then the second, etc.

RESTRICTIONS: The user should check Section V for the appropriate control card requirements. Punched output is written on logical tape 15, operator processing instructions should be supplied.

CALLING SEQUENCE: PNCHMA(F(DV),N,A1(DV),A2(DV),...)

SUBROUTINE NAMES: **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 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 V to determine which logical tape 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 NAME: **EØF** 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 V to determine available logical tapes.

CALLING SEQUENCE: EØF (L)
or REWIND (L)

MATRIX SUBROUTINES

Note: All of the above subroutines require that matrices be input 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 often referred to (within CCSD) as MØPAS, for Matrix Oriented Production Assembly System.

SUBROUTINE NAMES: ZERØ or ØNES

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: ZERØ(NR,NC,Z(IC))
or ØNES(NR,NC,Z(IC))

SUBROUTINE NAMES: 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.

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.

CALLING SEQUENCE: UNITY(N,Z(IC))
or SIGMA(N,Z(IC))

SUBROUTINE NAMES: GENALP or GENCØL

PURPOSE: These are special matrix generation subroutines. GENALP will generate a matrix such that every element is equal to a constant C. GENCØL 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 matrices 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 GENCØL(X1,X2,NR,Z(IC))

SUBROUTINE NAMES: SHIFT or REFLCT

PURPOSE: These subroutines may be used to move an entire matrix from one location to another. SHIFT moves the matrix exactly as is and REFLCT 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.

RESTRICTIONS: The matrices 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 overlayed into the [A] matrix.

CALLING SEQUENCE: SHIFT(A(IC),Z(IC))
or REFLCT(A(IC),Z(IC))

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

SUBROUTINE NAME: 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 matrices 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 time columns for [A] must equal the number of rows times columns of [Z].

CALLING SEQUENCE: SHUFL(A(IC),Z(IC))

SUBROUTINE NAMES: CØLMAX or CØLMIN

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: CØLMAX(A(IC),Z(IC))
or CØLMIN(A(IC),Z(IC))

SUBROUTINE NAMES: ELEADD or ELESUB

PURPOSE: These subroutines add or subtract the corresponding elements of two matrices respectively.

$$\overset{m*n}{[Z]} = \overset{m*n}{[A]} \pm \overset{m*n}{[B]} \quad , \quad z_{ij} = a_{ij} \pm b_{ij}$$

RESTRICTIONS: All matrices 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 NAMES: ELEMUL or ELEDIV

PURPOSE: These subroutines multiply or divide the corresponding elements of two matrices respectively.

$$\overset{m*n}{[Z]} = \overset{m*n}{[A]} \; */ \; \overset{m*n}{[B]} \quad , \quad z_{ij} = a_{ij} \; */ \; b_{ij}$$

RESTRICTIONS: All matrices 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 NAME: 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 matrices 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 NAMES: EFSIN or EFASN

PURPOSE: These subroutines perform elementry functions on all of the [A] matrix elements as follows:

$$z_{ij} = \text{sine}\ (a_{ij}) \quad \text{or} \quad z_{ij} = \text{arcsine}\ (a_{ij})$$

RESTRICTIONS: The matrices 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 overlayed into the [A] matrix.

CALLING SEQUENCE: EFSIN(A(IC),Z(IC))
or EFASN(A(IC),Z(IC))

SUBROUTINE NAMES: EFCØS or EFACS

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

$$z_{ij} = \text{cosine}\ (a_{ij}) \quad \text{or} \quad z_{ij} = \text{arccosine}\ (a_{ij})$$

RESTRICTIONS: The matrices 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 overlayed into the [A] matrix.

CALLING SEQUENCE: EFCOS(A(IC),Z(IC))
or EFACS(A(IC),Z(IC))

SUBROUTINE NAMES: EFTAN or EFATN

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

$$z_{ij} = \text{tangent}\ (a_{ij}) \quad \text{or} \quad z_{ij} = \text{arctangent}\ (a_{ij})$$

RESTRICTIONS: The matrices 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: EFTAN(A(IC),Z(IC))
or EFATN(A(IC),Z(IC))

SUBROUTINE NAMES: EFLOG or EFSQR

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

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

RESTRICTIONS: The matrices 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 NAMES: EFEXP or EFPØW

PURPOSE: These subroutine perform elementary functions on all of the [A] matrix elements as follows:

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

RESTRICTIONS: The matrices 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 overlayed 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 EFPØW(A(IC),α,Z(IC))

SUBROUTINE NAMES: 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.

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

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 NAMES: DISAS or ASSMBL

PURPOSE: These subroutines allow a user to operate on matrices in a partioned 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 matrices 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 NAMES: 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 matrices 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 NAMES: 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 matrices must have exactly enough space and contain the integer number of rows and columns as the first two data values. The matrices being multiplied must be conformable.

CALLING SEQUENCE: CØLMLT(A(IC),V(IC),Z(IC))
or RØWMLT(V(IC),A(IC),Z(IC))

SUBROUTINE NAMES: ADDALP or ALPHAA

PURPOSE: To 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 matrices 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 overlayed into the [A] matrix.

CALLING SEQUENCE: ADDALP(C,A(IC),Z(IC))
or ALPHAA(C,A(IC),Z(IC))

SUBROUTINE NAME: AABB

PURPOSE: To sum two scaled matrices.

$$\overset{m*n}{[Z]} = C1\overset{m*n}{[A]} + C2\overset{m*n}{[B]} \quad , \quad z_{ij} = C1*a_{ij} + C2*b_{ij}$$

RESTRICTIONS: All matrices 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 overlayed into either of the input matrices.

CALLING SEQUENCE: AABB(C1,A(IC),C2,B(IC),Z(IC))

SUBROUTINE NAME: BTAB

PURPOSE: To perform the following matrix operation:

$$\overset{n*n}{[Z]} = \overset{n*m}{[B]^t} \; \overset{m*m}{[A]} \; \overset{m*n}{[B]}$$

RESTRICTIONS: The matrices 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.

CALLING SEQUENCE: BTAB(A(IC),B(IC),Z(IC))

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

SUBROUTINE NAME: INVRSE

PURPOSE: To invert a square matrix.

$$\text{given } \overset{n*n}{[A]} \;,\quad \overset{n*n}{[Z]} = \overset{n*n}{[A]}^{-1}$$

RESTRICTIONS: The matrices 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 overlayed into the [A] matrix.

CALLING SEQUENCE: INVRSE(A(IC),Z(IC))

NOTE: This subroutine requires n dynamic storage allocations.

SUBROUTINE NAME: MULT

PURPOSE: To multiply two conformable matrices together.

$$\overset{m*n}{[Z]} = \overset{m*p}{[A]}\;\overset{p*n}{[B]} \;,\quad z_{ij} = a_{ik}*b_{kj}$$

RESTRICTIONS: The matrices 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 overlayed into either of them.

CALLING SEQUENCE: MULT(A(IC),B(IC),Z(IC))

NOTE: This subroutine required n*m dynamic storage locations.

SUBROUTINE NAME: TRANS

PURPOSE:

Given a matrix $\overset{m*n}{[A]}$ form its transpose as $\overset{n*m}{[Z]}$

RESTRICTIONS: Both matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. Theoutput matrix [Z] may be overlayed into the [A] matrix.

CALLING SEQUENCE: TRANS(A(IC),Z(IC))

NOTE: This subroutine requires n*m dynamic storage locations.

SUBROUTINE NAME: PØLMLT

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 matrices with the number of rows equal and each row receives the following operation.

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

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values.

CALLING SEQUENCE: PØLMLT(A(IC),B(IC),((IC))

SUBROUTINE NAME: PØLVAL

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*n but only the first row is evaluated.

CALLING SEQUENCE: PØLVAL(A(IC),X,Y,U,V)

SUBROUTINE NAME: 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 matrices 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 NAME: PØLSØV

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 RTPØLY and utilizes some double precision.

CALLING SEQUENCE: PØLSØV(A(IC),Z(IC))

SUBROUTINE NAME: JACØBI

PURPOSE: This subroutine will find the eigenvalues [E] and eigenvector matrix [Z] associated with an input matrix [A].

$$\underset{[A]}{n*n}\ \underset{[Z]}{n*n} = \underset{[Z]}{n*n}\ \underset{[E]}{n*1}$$

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values.

CALLING SEQUENCE: JACØBI(A(IC),E(IC),Z(IC))

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

SUBROUTINE NAME: MØDES

PURPOSE: This subroutine solves the following dynamic vibration equation

$$\underset{[A]}{n*n}\ \underset{[Z]}{n*n} = \underset{[B]}{n*n}\ \underset{[Z]}{n*n}\ \underset{\left[\frac{1}{W^2}\right]}{n*1}$$

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 radians/sec as [R] and in cycles/sec as [C], both [R] and [C] are n*1 matrices.

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. Subroutine JACØBI is called on.

CALLING SEQUENCE: MØDES(A(IC),B(IC),Z(IC),R(IC),C(IC))

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

SUBROUTINE NAME: 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:

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

RESTRICTIONS:

The mass and inertia data input 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))

where X is the matrix (N X 1) of collocation points referred to an arbitrary origin.

DMPL is the matrix (NDM X 4) of distributed mass per unit length 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 mass value at the rear of the slice.
Col 4 is the mass value at the front of the slice.

RIPL is the matrix (NRI X 4) of distributed rotary inertia per unit length slices. The columns here are similar to DMPL.

CM is the matrix (NCM X 4) 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 moment of inertia about the C. of G.

A is the output (2N X 2N) inertia matrix.

NØTE:

Having application to DMPL, RIPL and CM, it is noted that the location of the values may not go beyond the limits of the collocation points in either direction.

SUBROUTINE NAME: 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:

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

RESTRICTIONS:

The stiffness and shear data input 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 the matrix (N X 1) of collocation points referred to an arbitrary origin.

EI is the matrix (NEI X 4) 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 the matrix (NGA X 4) 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.

NØTE:

Having application to EI and GA, it is noted that the location of the values may not go beyond the limits of the collocation points in either direction.

SUBROUTINE NAME: 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 alpha-numeric 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 NAME: PLØT

PURPOSE: This subroutine produces SC-4020 plots of the columns of a matrix A , size (n*m) versus a column matrix X , size (n*1). It order the data internally and then calls on subroutine PLØTX1 (page 6.4.4). Each column in A requires as 12 word label for the Y axis title (YT) which must be entered sequentially as an array. The X axis title (XT) and top title (TT) must each consist of 12 work arrays.

RESTRICTIONS: The matrices must have exactly enough space and contain the integer number of rows and columns as the first two data values. The titles must have been input as positive arrays.

CALLING SEQUENCE: PLOT(A(IC),X(IC),TT(IC),YT(IC),XT(IC))

NOTE: This subroutine requires m+3 dynamic storage locations.

SUBROUTINE NAME: PUNCH

PURPOSE: This subroutine punchs out a matrix A , size n*m, one column at a time in any desired format. The argument FØR must reference a FØRTRAN format statement that has been input as a positive array. It must include the outer parenthesis but not the word FØRMAT. The argument HEAD must be a single BCD word used to identify the matrix. Each column is designated and restarts use of the FØRMAT statement.

RESTRICTIONS: The matrix A must have exactly enough space and contain the integer number of rows and columns as the first two data values.

CALLING SEQUENCE: PUNCH(A(IC),HEAD,FØR(IC))

NOTE: This subroutine requires n+3 dynamic storage locations.

MATRIX DATA STORAGE AND RETRIEVAL

The ability to store and retreive matrices from tape is easily achieved thru the use of the FILE and CALL subroutines. Matrices 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 13 and brings the desired matrix into core. The FILE subroutine writes a matrix onto the logical 12 tape. Subroutine ENDMØP causes all matrices from the logical 12 tape to be updated onto the logical 13 tape. In case of duplicate matrices the one from logical 12 replaces the one on logical 13. A matrix which has been filed cannot be called until an ENDMØP operation has been performed. To create a new tape the user merely sets control constant NØCØPY nonzero and has a scratch tape mounted on logical 13. The user should check the section on control cards and deck setup to determine control card requirements.

SUBROUTINE NAMES: CALL or FILE

PURPOSE: To allow the user to retrieve or store matrices on magnetic tape, see above. The H argument must be a six 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 NAMES: ENDMØP or LSTAPE

PURPOSE: Subroutine ENDMØP should be used in conjunction with subroutines CALL and FILE, see above. It causes matrices which have been filed by FILE on logical 12 to be updated onto logical 13. A call to subroutine LSTAPE will cause the output of the name, problem number and size of every matrix stored on tape on logical 13.

RESTRICTIONS: See above.

CALLING SEQUENCE: ENDMØP
or LSTAPE

Special Subroutines

SUBROUTINE NAME: 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),...,A(N,N),B1,...,BN,S1,...,SN

SUBROUTINE NAME: 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.

RESTRICTIONS:

All values must be floating point numbers except N and M which must be integers. N is the order of the polynominal 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))

*This subroutine requires 2*M dynamic storage core locations.

SUBROUTINE NAME: IRRADI or IRRADE

PURPOSE: These subroutines simulate a radiosity network* within a multiple gray surface enclosure containing a non-absorbing 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 Fahreheit system is required. The arbitrary number of temperature arguments may be constructed by a preceeding 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-Boltzman 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 SEQUENCE: IRRADI(AA(IC),Aϵ(IC),AFA(IC),ATQ(IC))
or IRRADE(AA(IC),Aϵ(IC),AFA(IC),ATQ(IC))

where 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
```

where FA(1,2) is defined as A(1)*F(1,2). After the subroutine is performed the ATQ array is ATQ(IC),Q1,Q2,Q3, . . . QN,END.
Since FA(1,2) ≡ FA(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-3G on a 65K core machine.

*"Radiation Analysis by the Network Method," A. K. Oppenheim, Transaction of the ASME, May 1956, pp. 725-735.

SUBROUTINE NAME: 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 a half symetric 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-3G 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 SEQUENCE: SLRADI(AA(IC),Aϵ(IC),AFA(IC),AT(IC),AI(IC),AQ(IC))
or SLRADE(AA(IC),Aϵ(IC),AFA(IC),AT(IC),AI(IC),AQ(IC))

SUBROUTINE NAME: SCRPFA

PURPOSE: To obtain the script FA value for radiant transfer within an enclosure. The input arrays are formated 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 identical to the AFA array and may overlay it.

RESTRICTIONS: All array arguments must reference the integer count set by the CINDA-3G preprocessor and all arrays must be exactly the required length.

CALLING SEQUENCE: SCRPFA(AA(IC),Aϵ(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 symetric coefficient matrix and call on SYMINV(AFA(IC)+3,AFA(IC)+1) to obtain the symetric inverse.

SUBROUTINE NAME: ABLATS

PURPOSE: To provide a simple ablation (sublimation) capability for the CINDA-3G user. The user constructs the 3-D network without considering the ablative. Then in Variables 2 he simulates 1-D ablative attachments by calling ABLATS. ABLATS constructs the 1-D 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-Boltzman constant and density units must agree in area and length units.

CALLING SEQUENCE: ABLATS(AA(IC),R,CP,G,T,C)

where
- C is the capacitance location of the 3-D node attached to.
- T is the temperature location of the 3-D node attached to.
- 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 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.
- 2 integer number of sublayers (NSL) desired, ABLATS subtracts from this the number of sublayers ablated.
- 3 the initial temperature of the material, ABLATS replaces this with the outer surface temperature, always in degrees F.
- 4 the impressed outer surface heating rate per unit area, radiation rates not included.
- 5 material thickness, this is replaced by the sublayer thickness.
- 6 surface area of the 3-D node attached to, need not be unity.
- 7 ablation temperature, degrees F.
- 8 heat of ablation
- 9 Stephan-Boltzman constant in consistent units.
- 10 surface emissivity
- 11 space "sink" temperature, degrees F.
- 12 SPACE,N,END where N equals NSI + 4.

NOTE: The outer surface radiation loss is integrated over the time step.
This subroutine requires 3(NSL+1) dynamic storage core locations.

SUBROUTINE NAME: 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 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 thru 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 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.
2 contains the liquid specific heat.
3 contains the liquid vaporization temperature.
4 contains the liquid heat of vaporization.
5 receives the liquid vaporization rate (weight/time)
6 receives the liquid vaporization total (total weight)
7 contains the liquid initial temperature.

SUBROUTINE NAME: 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))

SECTION VII

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 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 output by the preprocessor. These error messages are listed below and grouped according to various preprocessor functions. All error messages are preceeded by three asterisks which have been deleted below. Self-explanatory messages are not enlarged upon.

1. Processing Data Blocks

DATA BLØCKS IN IMPRØPER ØRDER ØR ILLEGAL BLØCK DESIGNATIØN ENCØUNTERED.

AN IMBEDDED BLANK HAS BEEN ENCØUNTERED IN THE LAST LINE.

INTEGER FIELD EXCEEDS 10.

REAL NUMBER FIELD EXCEEDS 20.

ALPHAMERIC FIELD EXCEEDS 6.

MULTIPLE DECIMAL PØINTS HAVE BEEN ENCØUNTERED.

NØDES MUST BE ØRDERED - DIFFUSIØN, ARITHMETIC, BØUNDARY.

CØNDUCTØRS MUST BE ØRDERED - REGULAR, RADIATIØN.

NØDE NUMBER, XXXXX, IS THE DUPLICATE ØF THE XXXXXTH NØDE.

CØNDUCTØR NUMBER, XXXXX, IS THE DUPLICATE ØF THE XXXXXTH CØNDUCTØR.

CØNSTANT NUMBER, XXXXX, IS THE DUPLICATE ØF THE XXXXXTH CØNSTANT.

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

FIXED CØNSTANT NAME IS NØT IN LIST.

2. Forming Pseudo Compute Sequence

NØDE, XXXXX, HAS NØ MATCH IN THE NA-NB PAIRS.

ADJØINING NØDE, XXXXX, ØF NA-NB PAIR HAS NØ MATCH IN THE NØDAL BLØCK.

3. Processing Program Blocks

EXECUTIØN BLØCKS IN IMPRØPER ØRDER ØR ILLEGAL BLØCK DESIGNATIØN ENCØUNTERED.

VARIABLE DESIGNATØR, AAAAA, NØT DEFINED FØR GENERAL PRØBLEM.

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 NØDE NUMBER, XXXXX.

MISSING CØNDUCTØR NUMBER, XXXXX.

MISSING CØNSTANT NUMBER, XXXXX.

MISSING ARRAY NUMBER, XXXXX.

FIXED CØNSTANT NAME, AAAAA, NØT IN LIST.

NUMBER ØF SUBRØUTINES REQUESTED EXCEEDS 75.

Explanation: More than 75 unique subroutines have been called.

4. Processing Parameter Changes

The first five parameter change error messages are prefaced with the words: PARAMETER CHANGE ERRØR.

NØDE NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

CØNDUCTØR NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

CØNSTANT NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

ARRAY NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

CØNSTANTS BLØCK WAS EMPTY IN THE ØRIGINAL PRØBLEM.

ARRAY BLØCK WAS EMPTY IN THE ØRIGINAL PRØBLEM.

5. Terminations Due to Errors (no preceeding asterisks)

THE ABØVE PARAMETER CHANGE WILL NØT BE EXECUTED.

ERRØR TERMINATIØN - LØADING IS SUPPRESSED.

SECTION VIII

CINDA-3G OPERATING SYSTEM DESCRIPTION

The increased rate of change in machine characteristics of digital computers requires a corresponding change in the design of large, flexible programs. The cost of conversion of machine dependent programs exceeds their worth in many cases. The Fortran V version of CINDA-3G is an attempt to minimize conversion efforts to succeeding machine generations by providing primarily Fortran coded routines capable of linking the engineering problem with the Fortran compiler. In general, the effectiveness of this method is limited only by the capability of the operating system to allow automatic selective composition of a program from a large file of subprograms. The functions performed by the UNIVAC Fortran V compiler, allocator, and CUR facility provides an efficient, flexible method of program maintenance and execution.

I. The Fortran V version of CINDA-3G exists logically as a pre-processor, processor, and library. The operational continuity of these portions is made possible by the UNIVAC 1108 software.

A. The function of the pre-processor is to operate on a user supplied problem and produce the following:

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 programs are operational equivalents of the users 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.

The pre-processor and appropriate use of the UNIVAC 1108 system control cards allows construction of the above from tape when the RECALL option is utilized.

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

C. The CINDA-3G library contains a large number of various types of subprograms to accomplish most user requirements. UNIVAC software 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.

II. Preprocessor

A. Operation of the pre-processing phase.
The main program (PREPRØ) accomplishes the initialization of data values and tape units and defines the order of processing which is carried out by seven overlay links:

1. If the problem being processed is a RECALL problem, SPLIT is called to read the recalled problem data and number definitions from the input tape and write these on the appropriate work tapes. SPLIT calls SKIP if the input tape is not positioned at the problem being recalled. (Section II. C.)

2. CØDERD reads the title block and the block title cards. DATARD reads the free-form data cards in the 4 (or 2, if General Problem) data blocks and any parameter change data. Each card is read, a format constructed for it, and then re-read. The data from each block is 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. PSEUDØ reads the node number definitions and NA-NB pairs from work tape. The pseudo compute sequence (Long or short) is constructed, packed by PACK43 (Slueth), and flagged by ØRMIN (Slueth), and written on the data tape. (Section II. B.)

4. GENLNK constructs the main program of the processor including CØMMØN and DIMENSIØN information. On the UNIVAC 1108, routines which are to be compiled from tape must have a particular format. BLKCRD, STFFB, and WRTBLK (Slueth) generate records in this format. (Section II. D.)

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 CØMMØN data arrays.

6. INITAL combines the original set of 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.

B. Construction of the Pseudo Compute Sequence
The pseudo compute sequence is a string of paired integers, the first consisting of a sign and 13 bits, the second of a sign and 12 bits. Four of these 27 bit composite words are packed into 3 machine (36 bits) words by PACK43.

The formation of the string takes place in PSUEDØ according to the following rules:

N_i are the diffusion and arithmetic mean node number definitions input in the BCD 3NØDE DATA block. LT_{nbj} is the relative location of node number NB_j. NA_j-NB_j are the node pairs joined by conductors input in the BCD 3CØNDUCTØR DATA block. m is the conductor number of the NA_j-NB_j under consideration. LKN_m is the relative location of conductor number m. The occurrence of $\underline{NA}_j$ and $\underline{NB}_j$ may be reversed throughout.

1. Short

   a. $N_i \in$ diffusion nodes, and
      $N_i = NA_j$
         (1) if $NB_j > 0$, and
             $NB_j \in$ diffusion nodes
             $\rightarrow LKN_m$, LT_{NB_j}, and
             $NB_j = -|NB_j|$

(2) if $NB_j < 0$, or
NB_j not $\in$ diffusion nodes
$\rightarrow LKN_m$, $-LT_{NB_j}$

b. $N_i \in$ arithmetic mean nodes, and
$N_i = NA_j$
(1) if $NB_j > 0$; and
$NB_j \in$ diffusion nodes
$\rightarrow LKN_m$, LT_{NB_j}, and
$NB_j = |NB_j|$

(2) if NB_j not $\in$ diffusion nodes
$\rightarrow LKN_m$, LT_{NB_j}

2. Long
Same as short, except the NB_j is not set negative in any case.

C. Store and Recall 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, and continue processing. The tapes saved from the above run can then be used in conjunction with a RECALL card and a BCD 3INITIAL PARAMETERS data deck to make necessary data changes and restart the saved problem at point of the interrupt.

Fortran logical tapes 22 and 13 are saved when running a store problem option; they are mounted on 21 and 13 when running the recall option.

The data tape 21 contains a six character identification, specified in the users call to STØREP, the problem type (GENERAL or THERMAL), the data number definitions from LUT1, and the data values from core.

The program tape, LB4P, contains the Fortran routines: LINKO, EXECTN, VARBL1, VARBL2, and ØUTCAL.

D. Format of Subroutines to be Compiled from Tape.
Routines must be written on tape (a non-Fortran write routine must be used to omit the Fortran control word on each record) in 507 word blocks with a maximum of 36 14-word card images per block. Each block contains three signal words, for example,

Word #	Block 1
1	Subroutine Name
2	Integer (Fortran) number of card images in block
3-16	1st card image
17-30	2nd card image
	etc.
507	+0 (denotes more blocks of same subroutine to follow)
	Block 2
1	0 (denotes continuation of above subroutine)
2	same as block 1
	card images
507	-0 (777777777777_8) (denotes last block of subroutine)

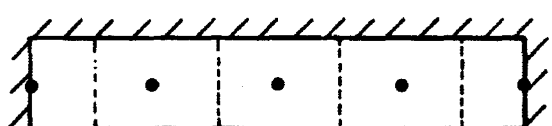
NOTE: Only the last block of a subroutine may be a short block (less than 36 card images) in which case no fill is needed.

SAMPLE PROBLEM 1A

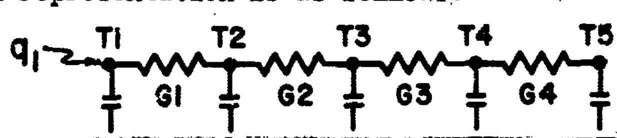
A perfectly insulated one dimensional bar has a constant heating rate applied to one end. Obtain the ten minute transient temperature response, at half minute intervals, of the bar ends and at points 1/4, 1/2 and 3/4 of the way along the bar. The bar is initially at 80°F and receives a constant heating rate of 3.0 BTU's/min. The length of the bar is four inches and it has a cross-sectional area of one square inch. It has the following material properties:

density = 172.8 lbs./ft^3
specific heat = 0.35 BTU/lb°F
thermal conductivity = 0.2 BTU/in min°F

Below is 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 as follows:



Capacitors receive the same number as the temperatures but with a C prefix. From the above information, we immediately calculate:

$C2=C3=C4=\rho \cdot V \cdot Cp=0.035$ BTU/°F
$C1=C5=C2/2.0=0.0175$ BTU/°F
$G1=G2=G3=G4=k \cdot Ac/\ell = 0.2$ BTU/min°F

where $V = \ell * Ac$, length times cross-sectional area.

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:

```
Col 8
        BCD 3THERMAL SPCS
        BCD 9SAMPLE PRØBLEM NØ.1A
        END
```

The nodal block is next and requires the node number, initial temperature, and capacitance of each node be listed.

```
Col 8
        BCD 3NØDE 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.

```
               Col 8
                    BCD 3CØNDUCTØR DATA
                    GEN 1,4,1,1,1,2,1,.2,1.,1.,1.
                    END
```

The only control constants required for CNFRWD are as follows:

```
               Col 8
                    BCD 3CONSTANTS DATA
                         TIMEND,10.,ØUTPUT,.',CSGFAC,2.
                    END
```

There is no array data and only one execution call, hence:

```
      Col 1    Col 8
                    BCD 3ARRAY DATA
                    END
                    BCD 3EXECUTIØN
          F        DIMENSION X(100)
          F        NDIM = 100
          F        NTH = 0
                         CNFRWD
                    END
```

There are no second variables operations but we must apply the heating rate in the first variables.

```
               Col 8
                    BCD 3VARIABLES 1
                         STFSEP (3.,Q1)
                    END
                    BCD 3VARIABLES 2
                    END
```

Our actual node numbers will have a one to one correspondence with the relative node numbers so the following completes the data input.

```
               Col 8
                    BCD 3ØUTPUT CALLS
                         PRNTMP
                    END
                    BCD 3END ØF DATA
```

The above problem data deck processed by the CINDA-3G program on the Univac 1108 as a standard run produces the following output:

NOTE: The only option to the BCD 3END ØF DATA card is a parameter change. A new job would require another set of control cards.

```
@ RUN, 9446      ,000005,005000,200    C0004590    1101000030GASKI            1327

@    ASG    D=U,F=F,I=1,K=N,L=L
@    XQT    CUR
  1.        TRW    F
  2.        IN     F
     END OF FILE -- UNIT F
     END CUR LLC 1102-0035 L6

@    XQT    C00045

REAL TIME CLOCK INTERROGATED AT 15:21:57
```

```
    BCD 3THERMAL SPCS
    BCD 9SAMPLE PROBLEM 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.
    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
    BCD 3EXECUTION
   DIMENSION X(100)                                            F
   NDIM = 100                                                  F
   NTH = 0                                                     F
        CNFRWD
    END
    BCD 3VARIABLES 1
        STFSEP(3.,01)              $ APPLY HEATING RATE
    END
    BCD 3VARIABLES 2
    END
    BCD 3OUTPUT CALLS
        PRNTMP
    END
            3     SUBROUTINES  NEEDED

            CNFRWD              STFSEP              PRNTMP


6   XWT   CUR

  1.      ERS

  2.      IN    F

   END OF FILE -- UNIT F

  3.      TRI   F

   END CUR LCC 1102-0035 L6
```

```
@N   FOR,K LINKO
UNIVAC 1108 FORTRAN V LEVEL   2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 13:22:12

        END OF UNIVAC 1108 FORTRAN V COMPILATION,     0 *DIAGNOSTIC* MESSAGE(S)

@N   FOR,K EXECIN
UNIVAC 1108 FORTRAN V LEVEL   2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 13:22:12

        END OF UNIVAC 1108 FORTRAN V COMPILATION,     0 *DIAGNOSTIC* MESSAGE(S)

@N   FOR,K VARBL1
UNIVAC 1108 FORTRAN V LEVEL   2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 13:22:13

        END OF UNIVAC 1108 FORTRAN V COMPILATION,     0 *DIAGNOSTIC* MESSAGE(S)

@N   FOR,K VARBL2
UNIVAC 1108 FORTRAN V LEVEL   2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 13:22:14

        END OF UNIVAC 1108 FORTRAN V COMPILATION,     0 *DIAGNOSTIC* MESSAGE(S)

@N   FOR,K OUTCAL
UNIVAC 1108 FORTRAN V LEVEL   2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 13:22:14

        END OF UNIVAC 1108 FORTRAN V COMPILATION,     0 *DIAGNOSTIC* MESSAGE(S)

@N     XQT     LINKO

       95 LOCATIONS AVAILABLE
```

SAMPLE PROBLEM 1A

```
* * * *
TIME  0.00000      DTIMEU  0.00000     CSGMIN(    1)   8.75000-02 DTMPCC(    1)  0.00000     ARLXCC(    0)   0.00000
          1  THRU    5        8.000000+01         8.000000+01         8.000000+01         8.000000+01        8.000000+01
* * * *
TIME  5.00000-01  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  6.23199-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.082763+02         8.252366+01         9.577512+01         8.790357+01        8.377844+01
* * * *
TIME  1.00000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.85894-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.207638+02         9.146831+01         1.077420+02         9.861606+01        9.324014+01
* * * *
TIME  1.50000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.63215-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.317719+02         1.018888+02         1.186640+02         1.093303+02        1.037467+02
* * * *
TIME  2.00000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.59454-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.425349+02         1.125543+02         1.294128+02         1.200446+02        1.144265+02
* * * *
TIME  2.50000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.58830-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.532573+02         1.232605+02         1.401327+02         1.307589+02        1.251350+02
* * * *
TIME  3.00000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.58726-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.639729+02         1.339734+02         1.508480+02         1.414732+02        1.358484+02
* * * *
TIME  3.50000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.58709-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.746874+02         1.446875+02         1.615624+02         1.521874+02        1.465625+02
* * * *
TIME  4.00000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.58706-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.654017+02         1.554017+02         1.722767+02         1.629017+02        1.572767+02
* * * *
TIME  4.50000+00  DTIMEU  2.14063-02  CSGMIN(    1)   8.75000-02 DTMPCC(    1)  4.58706-01 ARLXCC(    0)   0.00000
          1  THRU    5        1.961160+02         1.661160+02         1.629910+02         1.736160+02        1.679910+02
```

CHRYSLER IMPROVED NUMERICAL DIFFERENCING ANALYZER - C00045 (FORTRAN V VERSION)

SAMPLE PROBLEM 1A

```
* * * *
TIME  5.00000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   3)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.068303+02       1.768303+02       1.937053+02       1.843303+02       1.787053+02
* * * *
TIME  5.50000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   4)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.175445+02       1.875445+02       2.044195+02       1.950445+02       1.894195+02
* * * *
TIME  6.00000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   3)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.282588+02       1.982588+02       2.151338+02       2.057588+02       2.001338+02
* * * *
TIME  6.50000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   4)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.389731+02       2.089731+02       2.258481+02       2.164731+02       2.108481+02
* * * *
TIME  7.00000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   3)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.496873+02       2.196873+02       2.365623+02       2.271873+02       2.215623+02
* * * *
TIME  7.50000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   5)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.604016+02       2.304016+02       2.472766+02       2.379016+02       2.322766+02
* * * *
TIME  8.00000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   1)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.711159+02       2.411159+02       2.579909+02       2.486159+02       2.429909+02
* * * *
TIME  8.50000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   1)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.818301+02       2.518301+02       2.687051+02       2.593301+02       2.537051+02
* * * *
TIME  9.00000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   2)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        2.925444+02       2.625444+02       2.794194+02       2.700444+02       2.644194+02
* * * *
TIME  9.50000+00 DTIMEU  2.14063-02 CSGMIN(   1)   8.75000-02 DTMPCC(   2)  4.58706-01 ARLXCC(   0)   0.00000
      1  THRU    5        3.032587+02       2.732587+02       2.901337+02       2.807587+02       2.751337+02
```

```
SAMPLE PROBLEM 1A

* * * *
TIME  1.00000+01 DTIMEU   2.14065-02 CSGMIN(    1)    8.75000-02 DIMPCC(    2)    4.58706-01 ARLXCC(    0)    0.00000
       1  THRU     5        3.139729+02         2.839729+02         3.008479+02         2.914729+02         2.858479+02

       END OF DATA
```

SAMPLE PROBLEM 1B

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 Boltzman constant is $\sigma = 1.991E-13$ BTU/min in^2°R^4 and the emissivity varies linearly with temperature as follows:

$\epsilon = 0.4$ at -100°F
$\epsilon = 0.8$ at 300°F

2. The thermal conductivity of the bar varies with temperature as follows:

k = 0.15 at -100°F~(BTU/in min °F)
k = 0.25 at 100°F
k = 0.40 at 200°F
k = 0.60 at 300°F

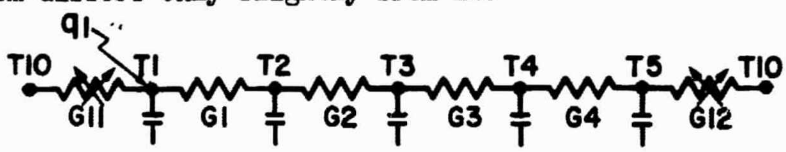
3. The density remains unchanged but the specific heat varies with temperature as follows:

Cp = 0.3 at -100°F~(BTU/lb °F)
= 0.39 at 100°F
= 0.49 at 200°F
= 0.65 at 300°F

4. The heating rate is a function of time as follows:

q = 3.0 at 0 min~(BTU/min)
q = 4.0 at 3. min
q = 4.0 at 7. min
q = 3.0 at 10. min

In addition, obtain the rate of heat loss and integral of the radiation transfer from the unheated end of the bar. The network representation of this problem differs only slightly from 1A.



Now however, the capacitances are a function of temperature. We therefore require multiplying factors such that:

C = ρ V Cp(T), MF= ρ V
MF = 0.1 for capacitors 2,3 and 4
MF = 0.05 for capacitors 1 and 5

The conductors are now:

$G = k(Tm)Ac/\ell$,MF = AC/ℓ , TM is the mean of the end T's
MF = 1.0 for conductors 1, 2, 3 and 4

A radiation conductor requires the input value $\sigma\epsilon$ FA, however FA = 1.0, hence

Grad = $\sigma\epsilon(T)$, MF = 1.991 BTU/min °F. Also, q = q(T)

The capacitors and conductors will be specified with CGS and CGD calls, the problem data deck may be constructed as follows:

```
Col 1      8
           BCD 3THERMAL SPCS
           BCD 9SAMPLE PRØBLEM 1B
           END
           BCD 3NØDE 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 3CØNDUCTØR 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 3CØNSTANTS DATA
               TIMEND,10.,OUTPUT,15,CSGFAC,2.,4,0,5,0
           END
           BCD 3ARRAY DATA
               1,-100.,.4,300.,0.8,END    $ EPSILØN 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,QTØTAL,END      $ A LABEL ARRAY
           END
           BCD 3 EXECUTIØN
    F      DIMENSION X(100)
    F      NDIM = 100
    F      NTH = 0
               CNFRWD
           END
           BCD 3VARIABLES 1
               D1DEG1 (TIMEM,A4,Q1)       $APPLY HEATING RATE
           END
           BCD 3VARIABLES 2
               RDTNQS(T10,T5,G12,K4)      $ØBTAIN HEAT FLØW RATE
               QINTEG(K4,DTIMEU,K5)       $INTEGRATE SAME
           END
           BCD 3ØUTPUT CALLS
               PRNTMP
               PRINTL (A5,K4,K5)
           END
           BCD 3END ØF DATA
```

The above problem data deck processed by the Univac 1108 version of CINDA-3G produces the following output.

```
@ RUN   6443      ,00003,00000,200    C0004596    110100030GASKI            0896

@     ASG    U=D,F=F,I=I,K=K,L=L
@     XQT    COR
  1.         TRW    F
  2.         IN     F
      END OF FILE -- UNIT F
      END COR LCC 1102-0035 LC

@     XQT    C00045


REAL TIME CLOCK INTERROGATED AT 09:11:41
```

```
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 NUMBERS                                  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
END
BCD 3ARRAY DATA
    1,-100.,.4,300.,.8,END     $ EPSILON VS T
    2,-100.,.15,100.,.25,200.,.4,300.,.8,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
BCD 3EXECUTION
DIMENSION X(100)                                               F
NDIM = 100                                                     F
NTH = 0                                                        F
    CNFRWD
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
    RUTNQS(T10,T5,G12,K4)     $ OBTAIN HEAT FLOW RATE
    QINTEG(K4,DTIMEU,K5)     $ INTEGRATE SAME
END
```

```
OUTPUT CALLS
     PRINTF
     PRINTL(K3,K4,K5)
   END



      5      SUBROUTINES   NEEDED

      CHFAND           VARGSM           VARGSM           GIDEG1           ROTNQS           QINTEG
      PRINTP           PRINTL


  XQT    COR

 1.      ERS

 2.      IN     F

   END OF FILE -- UNIT F

 3.      TRI    F

   END COR CCC 1102-J035 LG
```

```
ÐN    FOR,K LINKO
UNIVAC 1108 FORTRAN V LEVEL    2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 09:11:58

       END OF UNIVAC 1108 FORTRAN V COMPILATION.      0 *DIAGNOSTIC* MESSAGE(S)


ÐN    FOR,K EXECTN
UNIVAC 1108 FORTRAN V LEVEL    2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 09:11:58

       END OF UNIVAC 1108 FORTRAN V COMPILATION.      0 *DIAGNOSTIC* MESSAGE(S)


ÐN    FOR,K VARBL1
UNIVAC 1108 FORTRAN V LEVEL    2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 09:11:59

       END OF UNIVAC 1108 FORTRAN V COMPILATION.      0 *DIAGNOSTIC* MESSAGE(S)


ÐN    FOR,K VARBL2
UNIVAC 1108 FORTRAN V LEVEL    2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 09:12:00

       END OF UNIVAC 1108 FORTRAN V COMPILATION.      0 *DIAGNOSTIC* MESSAGE(S)


ÐN    FOR,K OUTCAL
UNIVAC 1108 FORTRAN V LEVEL    2206 0009 F600A
THIS COMPILATION WAS DONE ON 11 DEC 67 AT 09:12:01

       END OF UNIVAC 1108 FORTRAN V COMPILATION.      0 *DIAGNOSTIC* MESSAGE(S)
```

```
W    AWT   LINK0



          STARTING ADDRESS  014000

          CORE LIMITS 014000 023106     100000 102741     163772 163777



     LINK0 /CODE
     0   100000-100000
     1   014000-014013

     NSTOP$/CODE
     1   014014-014031

     NIER$ /CODE
     0   100001-100001
     1   014032-014133
     2   100002-100064

     NERR$ /CODE
     0   100065-100214
     1   014134-014530

     EXECTN/CODE
     0   100215-100216
     1   014531-014552

     CNFRWD/CODE
     0   100217-100331
     1   014553-015674

     NOUT$ /CODE
     0   100332-100333
     1   015675-016731
     2   100334-100374

     NTAB$ /CODE
     0   100375-100476

     NFTV$ /CODE
     1   016732-016754

     NIOIN$/CODE
     1   016755-017020
     2   100477-100527

     NOTIN$/CODE
```

```
0   100530-100530
1   017021-017322
2   100531-100571

NFMTS /CODE
1   017323-020324
2   100572-100712

OUTCAL/CODE
0   100713-100714
1   020325-020350

PRINT /CODE
0   100715-101011

WRTLOB/CODE
0   101012-101031
1   020351-020465

TOPLIN/CODE
0   101032-101066
1   020466-020527

SKPLIN/CODE
0   101067-101074
1   020530-020572

PRNTMP/CODE
0   101075-101120
1   020573-020711

STNDRD/CODE
0   101121-101157
1   020712-020774

VARBL2/CODE
0   101160-101161
1   020775-021024

QINTEG/CODE
0   101162-101165
1   021025-021054

RDTNQS/CODE
0   101166-101176
1   021055-021122

UNPAK /CODE
1   021123-021152

VARBL1/CODE
0   101177-101204
1   021153-021304

DIDEG1/CODE
0   101205-101224
1   021305-021474
```

```
VARGSM/CODE
0    101225-101234
1    021475-021551

DIDIWM/CODE
0    101235-101241
1    021552-021611

LAGRAN/CODE
0    101242-101274
1    021612-022032

VARCSM/CODE
0    101275-101276
1    022033-022065

XSPACE/******
0    101277-101444

INPUTT/CODE
0    101445-101502
1    022066-022525

NFINP$/CODE
0    101503-101506
1    022526-023051

NBUFF$/CODE
1    023052-023106
2    101507-102515

LOGIC /******
0    102516-102521

DIMENS/******
0    102522-102533

FIXCON/******
0    102534-102615

ARRAY /******
0    102616-102657

KONST /******
0    102660-102661

PCS   /******
0    102662-102666

COND  /******
0    102667-102674

SOURCE/******
0    102675-102701

CAP   /******
0    102702-102707
```

9.18

```
TEMP  /******
0    102710-102715

TITLE /******
0    102716-102741


END OF ALLOCATION 1103    0033

      95 LOCATIONS AVAILABLE
```

```
          CHRYSLER IMPROVED NUMERICAL DIFFERENCING ANALYZER - C00045          (FORTRAN V VERSION)

   SAMPLE PROBLEM 15

  *   *   *   *
 TIME    0.00000     DTIMEU   0.00000    CSGMIN(     1)    7.93690-02 DTMPCC(     1)    0.00000    ARLXCC(     0)    0.00000

        1  THRU      5         8.000000+01         8.000000+01         8.000000+01         8.000000+01         8.000000+01
        6  THRU      6        -4.600000+02

 QRATE     0.00000     QTOTAL     0.00000

  *   *   *   *
 TIME    5.00000-01 DTIMEU    2.52757-02 CSGMIN(     2)    7.82618-02 DTMPCC(     1)    6.81163-01 ARLXCC(     0)    0.00000

        1  THRU      5         1.050446+02         9.449641+01         8.761895+01         8.387773+01         8.268623+01
        6  THRU      6        -4.600000+02

 QRATE     1.00565-02 QTOTAL     4.94005-03

  *   *   *   *
 TIME    1.00000+00 DTIMEU    2.92845-02 CSGMIN(     2)    7.63027-02 DTMPCC(     1)    6.41844-01 ARLXCC(     0)    0.00000

        1  THRU      5         1.167556+02         1.059681+02         9.799292+01         9.320243+01         9.158634+01
        6  THRU      6        -4.600000+02

 QRATE     1.08922-02 QTOTAL     1.01776-02

  *   *   *   *
 TIME    1.50000+00 DTIMEU    1.77686-02 CSGMIN(     2)    7.37220-02 DTMPCC(     4)    3.86078-01 ARLXCC(     0)    0.00000

        1  THRU      5         1.275381+02         1.168016+02         1.087716+02         1.037800+02         1.020570+02
        6  THRU      6        -4.600000+02

 QRATE     1.19551-02 QTOTAL     1.58963-02

  *   *   *   *
 TIME    2.00000+00 DTIMEU    2.51335-02 CSGMIN(     2)    7.14967-02 DTMPCC(     5)    5.60459-01 ARLXCC(     0)    0.00000

        1  THRU      5         1.383390+02         1.276921+02         1.197355+02         1.147887+02         1.130795+02
        6  THRU      6        -4.600000+02

 QRATE     1.31536-02 QTOTAL     2.21878-02

  *   *   *   *
 TIME    2.50000+00 DTIMEU    3.17726-02 CSGMIN(     2)    6.94692-02 DTMPCC(     5)    7.21013-01 ARLXCC(     0)    0.00000

        1  THRU      5         1.493132+02         1.387631+02         1.309002+02         1.260208+02         1.243351+02
```

CHRYSLER IMPROVED NUMERICAL DIFFERENCING ANALYZER - C00045 (FORTRAN V VERSION)

9.20

```
    SAMPLE PROBLEM 1B

       5  THRU      6            -4.600000+02

QRATE      1.44755-02 QTOTAL     2.91109-02

*  *   *  *
TIME   3.00000+00 DTIMEU   2.17097-02 CSGMIN(     2)     6.75580-02 DTMPCC(     5)     5.00361-01 ARLXCC(     0)     0.00000

       1  THRU      5            1.604636+02         1.500197+02         1.422505+02         1.374380+02         1.357755+02
       6  THRU      6           -4.600000+02

QRATE      1.59354-02 QTOTAL     3.67297-02

*  *   *  *
TIME   3.50000+00 DTIMEU   2.76265-02 CSGMIN(     2)     6.59192-02 DTMPCC(     5)     6.36307-01 ARLXCC(     0)     0.00000

       1  THRU      5            1.711316+02         1.611111+02         1.536147+02         1.489519+02         1.473359+02
       6  THRU      6           -4.600000+02

QRATE      1.75182-02 QTOTAL     4.51110-02

*  *   *  *
TIME   4.00000+00 DTIMEU   1.74566-02 CSGMIN(     2)     6.44299-02 DTMPCC(     5)     3.91818-01 ARLXCC(     0)     0.00000

       1  THRU      5            1.614217+02         1.718384+02         1.646895+02         1.602503+02         1.587101+02
       6  THRU      6           -4.600000+02

QRATE      1.92066-02 QTOTAL     5.43117-02

*  *   *  *
TIME   4.50000+00 DTIMEU   2.25795-02 CSGMIN(     2)     6.31386-02 DTMPCC(     5)     4.89232-01 ARLXCC(     0)     0.00000

       1  THRU      5            1.915047+02         1.823174+02         1.754866+02         1.712539+02         1.697840+02
       6  THRU      6           -4.600000+02

QRATE      2.09631-02 QTOTAL     6.43759-02

*  *   *  *
TIME   5.00000+00 DTIMEU   2.67976-02 CSGMIN(     2)     6.19734-02 DTMPCC(     5)     5.71008-01 ARLXCC(     0)     0.00000

       1  THRU      5            2.013994+02         1.925709+02         1.860252+02         1.819765+02         1.805689+02
       6  THRU      6           -4.600000+02

QRATE      2.27951-02 QTOTAL     7.53356-02
```

SAMPLE PROBLEM 1B

```
* * * *
TIME   5.50000+00 DTIMEU   1.54995-02 CSGMIN(    2)    6.08160-02 DTMPCC(    5)    3.42910-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.110437+02      2.026091+02      1.963219+02      1.924378+02      1.910854+02
       6  THRU   6      -4.600000+02

QRATE     2.47138-02 QTOTAL    8.72342-02

* * * *
TIME   6.00000+00 DTIMEU   2.03045-02 CSGMIN(    2)    5.99038-02 DTMPCC(    5)    4.11912-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.204166+02      2.123606+02      2.063703+02      2.026540+02      2.013541+02
       6  THRU   6      -4.600000+02

QRATE     2.66987-02 QTOTAL    1.00108-01

* * * *
TIME   6.50000+00 DTIMEU   2.37155-02 CSGMIN(    2)    5.90644-02 DTMPCC(    5)    4.64838-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.295401+02      2.218219+02      2.161003+02      2.125576+02      2.113168+02
       6  THRU   6      -4.600000+02

QRATE     2.87449-02 QTOTAL    1.13993-01

* * * *
TIME   7.00000+00 DTIMEU   2.67642-02 CSGMIN(    2)    5.83134-02 DTMPCC(    5)    5.08066-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.384312+02      2.310157+02      2.255325+02      2.221425+02      2.209532+02
       6  THRU   6      -4.600000+02

QRATE     3.08419-02 QTOTAL    1.26916-01

* * * *
TIME   7.50000+00 DTIMEU   1.57623-02 CSGMIN(    2)    5.76442-02 DTMPCC(    5)    2.85070-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.465667+02      2.396901+02      2.345584+02      2.313713+02      2.302473+02
       6  THRU   6      -4.600000+02

QRATE     3.29906-02 QTOTAL    1.44900-01

* * * *
TIME   8.00000+00 DTIMEU   1.82364-02 CSGMIN(    2)    5.70813-02 DTMPCC(    5)    3.07639-01 ARLXCC(    0)    0.00000
       1  THRU   5       2.541671+02      2.477187+02      2.429721+02      2.400250+02      2.389816+02
```

```
SAMPLE PROBLEM 1B

      6  THRU     6         -4.600000+02

QRATE     3.51065-02 QTOTAL     1.61959-01

 *   *   *   *
TIME   8.50000+00 DTIMEU   2.03899-02 CSGMIN(    2)    5.65876-02 DTMPCC(    5)    3.20296-01 ARLXCC(    0)    0.00000

      1  THRU     5          2.611250+02          2.552035+02          2.508109+02          2.480850+02          2.471158+02
      6  THRU     6         -4.600000+02

QRATE     3.71736-02 QTOTAL     1.60047-01

 *   *   *   *
TIME   9.00000+00 DTIMEU   2.22825-02 CSGMIN(    2)    5.61523-02 DTMPCC(    5)    3.25861-01 ARLXCC(    0)    0.00000

      1  THRU     5          2.676713+02          2.621808+02          2.581129+02          2.555889+02          2.546872+02
      6  THRU     6         -4.600000+02

QRATE     3.91845-02 QTOTAL     1.99164-01

 *   *   *   *
TIME   9.50000+00 DTIMEU   2.39538-02 CSGMIN(    2)    5.57670-02 DTMPCC(    5)    3.25928-01 ARLXCC(    0)    0.00000

      1  THRU     5          2.737683+02          2.686783+02          2.649104+02          2.625720+02          2.617322+02
      6  THRU     6         -4.600000+02

QRATE     4.11331-02 QTOTAL     2.19270-01

 *   *   *   *
TIME   1.00000+01 DTIMEU   2.54350-02 CSGMIN(    2)    5.54248-02 DTMPCC(    5)    3.21664-01 ARLXCC(    0)    0.00000

      1  THRU     5          2.794356+02          2.747200+02          2.712311+02          2.690647+02          2.682821+02
      6  THRU     6         -4.600000+02

QRATE     4.30139-02 QTOTAL     2.40334-01

      END OF DATA
```

9.22