NBSIR 88-3814

#3422

PROGRESS TOWARD A GENERAL ANALYTICAL METHOD FOR PREDICTING INDOOR AIR POLLUTION IN BUILDINGS

INDOOR AIR QUALITY MODELING PHASE III REPORT

James Axley

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Building Environment Division
Gaithersburg, MD 20899

July 1988

Prepared for:
U.S. Environmental Protection Agency
U.S. Department of Energy
U.S. Consumer Products Safety Commission

U.S. DEPARTMENT OF COMMERCE, C. William Verity, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

ABSTRACT

This interim report presents the results of Phase III of the NBS General Indoor Air Pollution Concentration Model Project. It describes;

a) a general *element-assembly* formulation of multi-zone contaminant dispersal analysis theory that provides a general framework for the development of detailed (*element*) models of mass transport phenomena that may affect contaminant dispersal in buildings,

b) an approach to modeling the dispersal of *interactive* contaminants involving contaminant mass transport phenomena governed by basic principals of kinetics and introduces a *linear first order kinetics element* to achieve this end,

c) an approach to modeling the details of contaminant dispersal driven by convection-diffusion processes in one-dimensional flow situations (e.g., HVAC ductwork) and introduces a *convection-diffusion flow element* to achieve this end,

and

d) the features and use of CONTAM87, a program that provides a computational implementation of the theory and methods discussed.

The theory and methods presented are based upon a generalization of the building idealization employed earlier [Axley, 1987]. Here, building air flow systems are idealized as assemblages of *mass transport elements*, rather than simply flow elements as used previously, connected to discrete *system nodes* corresponding to well-mixed air zones within the building and its HVAC system. Equations governing contaminant dispersal in the whole building air flow system due to air flow and reaction or sorption mass transport phenomena are formulated by assembling element equations, that characterize a specific instance of mass transport in the building air flow system, in such a manner that the fundamental requirement of conservation of mass is satisfied in each zone.

KEY WORDS: building simulation, indoor air quality, contaminant dispersal analysis, element assembly, discrete modeling techniques

ACKNOWLEDGEMENT

This work was supported by the following Interagency Agreements:

IAG: DE AI01-86-CD 21013 Amendment Number 4 with the Department of Energy,

IAG: DW 1391103-01-2 with the Environmental Protection Agency, and

IAG: 74-25 Task Number 87-4 with the Consumer Products Safety Commission.

CONTENTS

PREFACE

The work reported here is a product of the General Indoor Air Pollution Concentration Model Project initiated in 1985 at the National Bureau of Standards and supported by the U. S. Environmental Protection Agency, the U.S. Department of Energy, and the Consumer Products Safety Commission. The fundamental objective of this project is to develop a comprehensive validated computer model to simulate dynamic pollutant movement and concentration variation in buildings. The scope of the project is ambitious; a full-scale, multi-zone building contaminant dispersal model that simulates flow processes (e.g., infiltration, dilution, & exfiltration) and contaminant generation, reaction, and removal processes is being developed.

During the planning stage of this project it was decided to organize efforts into three distinct phases:

Phase I: formulation of a general framework for the development of general indoor air quality analysis models (see [McNall et.al., 1986] for report of Phase I work),

Phase II: development of a residential-scale model, based on the simplifying assumption that air is well-mixed within each building zone, providing simple simulation of HVAC system interaction, and

Phase III: extension of modeling capabilities to allow more complete simulation of HVAC system interaction and consideration of rooms that are not well-mixed.

This report presents analytical methods that, together with those methods developed during Phase II of the project, satisfies the scope and objectives set for Phase III of the "General Indoor Air Pollution Concentration Model" Project and, as such, completes Phase III efforts. The report is organized in two parts. In the first part of the report the underlying theory is presented;

Section 1: outlines the general aspects of indoor air quality analysis – making the distinction between contaminant dispersal analysis, inverse contaminant dispersal analysis, and air flow analysis – that the project has attempted to address, and defines the approach taken to modeling,

Section 2: presents a general formulation of multi-zone contaminant dispersal theory, using an element assembly approach,

Section 3: applies the theory from Section 2 to develop an interactive, multiple-contaminant dispersal analysis method based upon the formulation of a kinetics element designed to model mass transport phenomena governed by the principals of kinetics,

Section 4: applies the theory from Section 2 to model the details of one-dimensional contaminant dispersal driven by combined convection-diffusion mass transport processes,

The second part of the report presents the practical implementation of the contaminant dispersal analysis theory in the program CONTAM87;

Sections 5 -8: provide a users manual for the program CONTAM87, and

Section 9: gives examples of application of CONTAM87 to representative problems of contaminant dispersal analysis.

The last section, Section 10, provides a summary of the work reported here and outlines possible directions of future study.

The complete source code for CONTAM87 is listed in the appendix.

NOTATION

Scalars

Scalar variables will be designated by lower and upper case, plain, english and greek characters. An equals sign enclosed in square brackets, [=], is used to designate typical units of a variable. The more commonly used scalars are listed below:

C	concentration in terms of mass fraction (mass-species/mass-air) [=] lb-species/lb-air or kg-species/kg-air
G	species generation rate (mass/unit time) [=] lb-species/hr or kg-species/s
M	mass of the volume of air within a given zone (mass-air) [=] lb-air or kg-air
P	pressure (force/unit area) [=] lb/ft^2 or Pascals (Pa)
t	time
T	temperature [=] °F or °C
v	velocity (e.g., of a fluid particle) [=] ft/hr or m/s
w	mass transport rate (e.g., due to flow, chemical reaction, etc.) [=] lb/hr or kg/s
x,y,z	spacial coordinates
κ	reaction rate coefficient [=] s^{-1}
ρ	density (mass/unit volume) [=] lb/ft^3 or kg/m^3

Indices

The indicial notation used in this report is modeled after the conventions that are commonly used in structural analysis and Finite Element analysis literature. A variety of indices may be associated with any single variable including pre-subscripts, pre-superscripts, post-subscripts, and post-superscripts. Although the meaning of any index should be clear from the context of the discussion, the conventions diagrammed below will be followed to help maintain clarity.

$$\text{species index} \rightarrow \; {}^{\alpha}_{con}X^{a}_{j} \;\begin{array}{l}\leftarrow \text{element index} \\ \leftarrow \text{variable} \\ \leftarrow \text{node index}\end{array}$$

$$\text{descriptive index} \rightarrow con$$

(In some contexts the post-subscript is used to indicate an element of an array as well.)

In addition to these indices which serve to specifically identify a variable, it will be necessary to use additional indices to indicate the value (or approximation to the value) of a variable at a discrete point in time or for a discrete step in an iterative scheme. In both cases, additional superscripts and subscripts may have to be introduced. To distinguish these time-step or iterative-step indices they shall be enclosed in parenthesis as indicated below.

$${}^{\alpha}_{con}X^{a(k)}_{j(n+1)} \quad \begin{array}{l}\leftarrow \text{the "kth" iterate} \\ \leftarrow \text{the "n+1th" time step}\end{array}$$

Generally, the following conventions will be used for superscripts and subscripts:

a,b,c,...	specific element indices
e	general element index
$\alpha,\beta,\gamma,\ldots$	specific species indices
α	general species index
i,j,k,l,m,n	node indices (or array element indices)

Vectors and Matrices

Vectors and matrices will be designated by **bold** characters. In general, vector quantities will also be enclosed in braces (e.g., {**V**}) while matrix quantities will be enclosed in square brackets (e.g., [**M**]) to emphasize and clarify the form of expressions.

PART I - THEORY

The first section of this part of the report gives definition to the meaning of indoor air quality analysis and describes the modeling approach taken to develop practical indoor analysis tools. It will be seen that indoor air quality analysis involves *(forward) contaminant dispersal analysis*, *inverse contaminant dispersal analysis*, *air flow analysis*, and *thermal analysis*. The following three sections extend the contaminant dispersal analysis theory developed during Phase II of the present project [Axley, 1987] by first presenting a more general formulation of multi-zone contaminant dispersal analysis theory and then applying this more general theory to a) dispersal problems involving interactions between contaminant species and/or the building fabric and b) dispersal problems where the details of convection-diffusion flow processes are important (e.g., HVAC ductwork). Current research efforts focused on the inverse contaminant dispersal analysis problem have led to promising new multi-zone tracer gas techniques, the *Pulse Tracer Techniques* and have provided a better understanding of existing tracer gas techniques. Formulations of building air flow and thermal analysis theories that are compatible with the formulations of the forward and inverse contaminant dispersal analysis theories have been presented earlier [Axley 1987; Axley 1985] and will become the focus of future work.

1. Introduction

During the past decade, indoor air pollution emerged as an international health issue and, as a result, a new field of simulation, *indoor air quality analysis*, is emerging to provide the means to predict concentration variation of indoor air contaminants in existing and proposed buildings and, thereby, to assess the nature and severity of potential indoor air quality problems. It may be expected that this new field will come to play a key role in the development of strategies to mitigate indoor air quality problems and, eventually, become central to the design of high quality indoor air environments.

1.1 Indoor Air Quality Analysis

The central concern of indoor air quality analysis is the prediction of airborne contaminant dispersal in buildings. Airborne contaminants disperse throughout buildings in a complex manner that depends on the nature of air movement in-to, out-of, and within the building system; the influence of the heating, ventilating, and air conditioning (HVAC) systems; the possibility of removal, by filtration, or contribution, by generation, of contaminants; and the possibility of chemical reaction, radio-chemical decay, settling, or sorption of contaminants. In indoor air quality analysis we seek to comprehensively model all of these phenomena.

More precisely, in indoor air quality analysis we consider building air flow systems to be three dimensional fields within which we seek to completely describe the *state* of infinitesimal air parcels. The state of such an air parcel is defined by its temperature, pressure, velocity, and contaminant concentration(s) – the *state variables* of indoor air quality analysis.

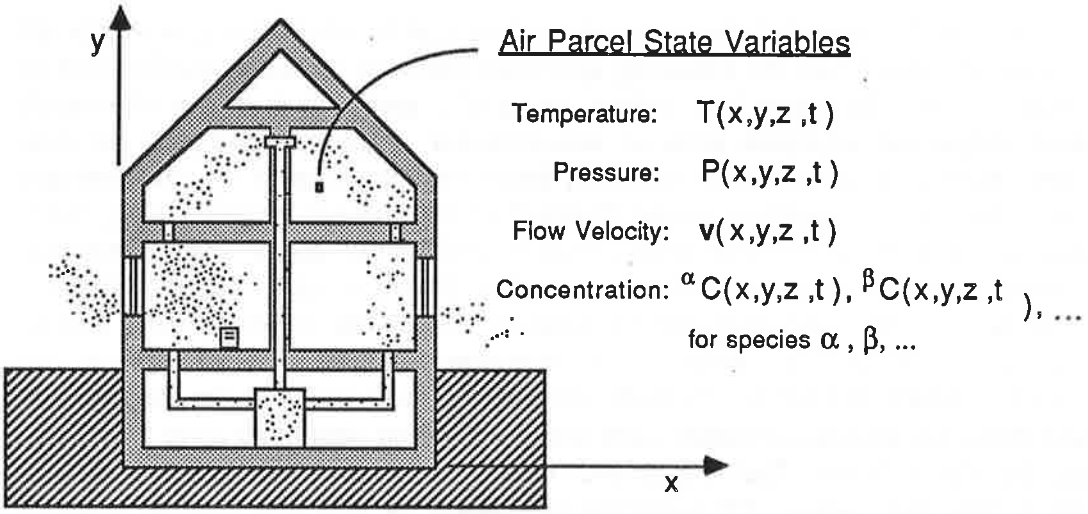


Figure 1-1 Indoor Air Quality State Variables

The central problem of indoor air quality analysis is, then, the determination of the spacial (x,y,z) and temporal (t) variation of contaminant species concentrations or *contaminant dispersal analysis*.

For a single *noninteractive*[1] species, α, contaminant dispersal is driven by the air velocity field and its variation with time and thus the contaminant dispersal analysis problem, for this case, may be represented as:

Noninteractive Contaminant Dispersal Analysis

$$^{\alpha}C(x,y,z,t) = {}^{\alpha}C(\,\mathbf{v}(x,y,z,t), \ldots)$$

where the ellipses, ... , are used to indicate initial and boundary conditions required to complete the definition of the analytical problem. To solve the contaminant dispersal problem, then, the flow field must be either specified or determined.

Two approaches to flow determination exist. In the first approach a nonlinear *flow analysis* problem and, in general, a coupled thermal analysis problem is formulated and solved, given the environmental excitation (e.g., wind, solar, and thermal excitation) acting on the building system. Alternatively, for existing buildings it may be possible to "measure" building air flows using tracer gas techniques. These techniques are based on the formulation and solution of the *inverse contaminant dispersal analysis* problem. Functionally, these related problems take the following forms:

---

[1] *Noninteractive Contaminant*: a contaminant whose dispersal is not affected by kinetics of reaction, sorption, settling, or other similar or related mass transport phenomena.

<u>Coupled Flow/Thermal Analysis</u>

$$\mathbf{v}(x,y,z,t) = \mathbf{v}(\ P(x,y,z,t), \ldots) \quad \textit{Flow Analysis}$$
$$P(x,y,z,t) = P(\ T(x,y,z,t), \ldots) \quad \textit{Buoyancy Effects}$$
$$T(x,y,z,t) = T(\ \mathbf{v}(x,y,z,t), \ldots) \quad \textit{Thermal Analysis}$$

<u>Inverse Contaminant Dispersal Analysis</u>

$$\mathbf{v}(x,y,z,t) \overset{?}{=} \mathbf{v}(\ {}^{\alpha}C(x,y,z,t), \ldots)$$

(the basis of tracer gas techniques)

When contaminant reaction, settling, sorption, etc. kinetics is important, the contaminant dispersal analysis problem becomes a coupled (and, generally, nonlinear) analysis problem as (the rate of change of) each species' concentration will depend upon both species' concentrations and the air flow velocity field:

<u>Interactive Contaminant Dispersal Analysis</u>

$${}^{\alpha}C(x,y,z,t) = {}^{\alpha}C(\ \mathbf{v}(x,y,z,t), {}^{\alpha}C(x,y,z,t), {}^{\beta}C(x,y,z,t), \ldots\ )$$

For such cases we say the contaminant is an *interactive* contaminant and describe the analytical problem as a problem of *interactive contaminant dispersal analysis.*

A complete indoor air quality analysis package should provide the analyst with tools to consider this relatively complex set of analytical problems related to the central task of contaminant dispersal analysis. As indicated in Figure 1-2 one may anticipate three basic indoor air quality analysis scenarios;

1) in some instances the analyst may choose to simply specify the flow field (e.g., in design situations or in those cases where the HVAC system substantially determines air flow in the building system) and directly consider the contaminant dispersal analysis problem for specific indoor air pollutant sources or sinks,

2) for existing buildings, tracer gas techniques, based upon inverse contaminant dispersal analysis methods, may be used to determine airflows that may then be used to complete the required contaminant dispersal analysis for any number of specific indoor air pollutant sources or sinks, or

3) in some instances the analyst may choose to complete an airflow analysis of the building system, given building and wind characteristics, to determine the airflows needed to complete the contaminant dispersal analysis task.

Many specific pollutant source or sink models will involve chemical or mass transport governed by the kinetics of the mass transport phenomena; thus analytical tools are needed to properly account for this. Finally, when airflow analysis is elected the analyst will either have to specify the temperature field or determine it by solving the coupled flow-thermal analysis problem to properly account for buoyancy effects; a complete indoor air quality analysis package should provide this capability.

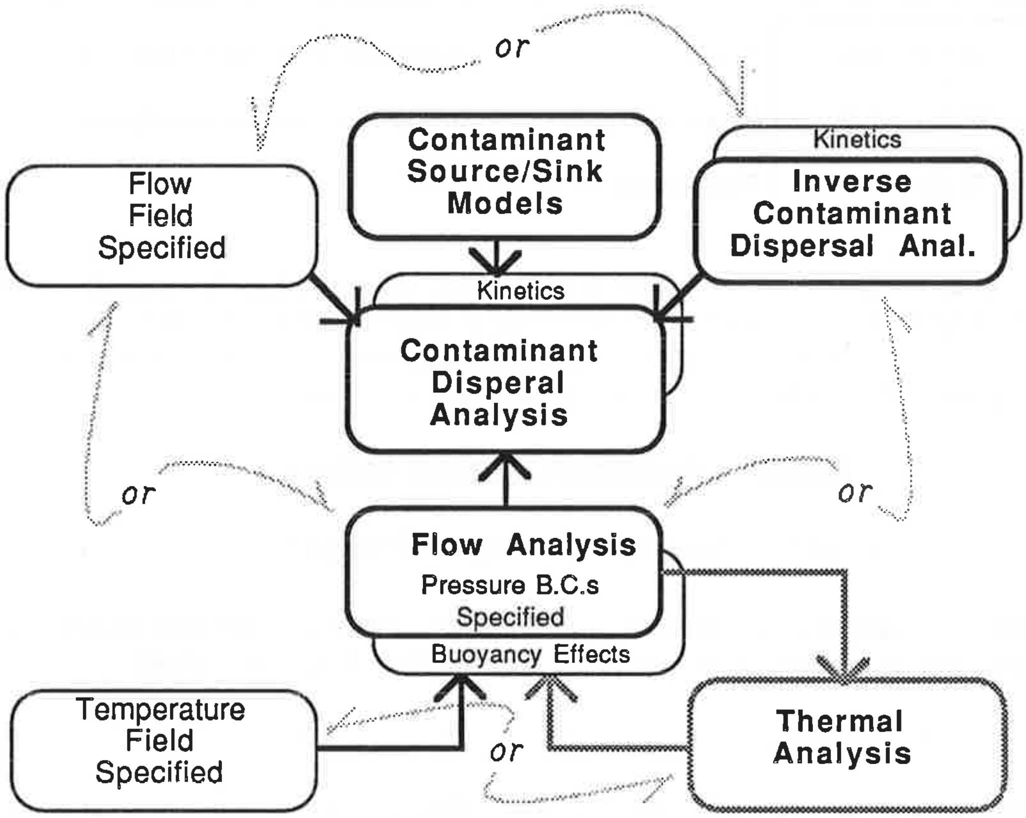


Figure 1-2 The Central and Related Problems of Indoor Air Quality Analysis

1.2 The Well-Mixed Macroscopic Model

To develop this needed indoor air quality analysis capability we follow the tradition established by others in the field of indoor air quality analysis [Sinden, 1979, Sandberg, 1984, Walton, 1985] and model building air flow systems using a well-mixed zone simplification of the macroscopic equations of motion (i.e., mass, momentum, and energy balances for flow systems) that, in essence, transforms the indicated field problems discussed above into spatially discrete, but temporally continuous, ordinary differential equations. The present approach breaks from this tradition, however, in that an *element assembly* approach is taken to formulate the respective analytical problems. That is to say:

> building air flow systems (fields) will be idealized as *assemblages* of discrete *flow elements,* that are used to model specific flow transport processes between well-mixed zones, and *kinetics elements,* that are used to model specific transport processes that occur within the well-mixed zones that may be described using the principals of kinetics.

Such idealizations of building air flow systems may be represented graphically in a direct and intuitive way as illustrated in Figure 1-3 for a hypothetical building system.

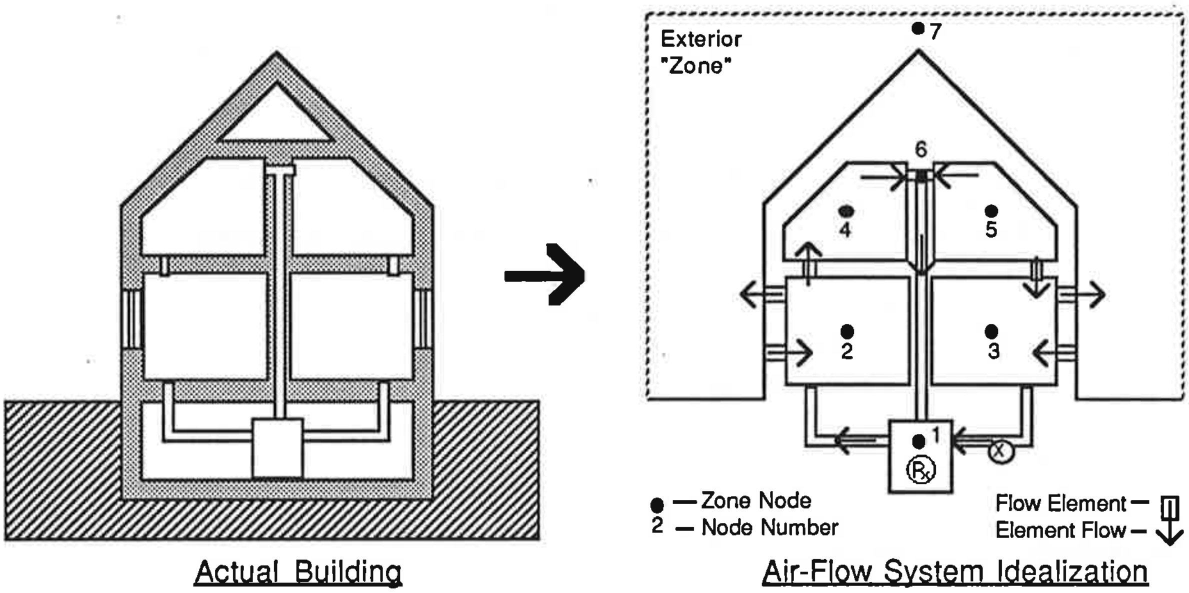


Figure 1-3 Idealization of A Hypothetical Building Air Flow System

With a knowledge of the air flow paths in the building system the analyst selects from the *library* of available air flow elements to assemble graphically, and, hence mathematically, the building air-flow idealization. *Kinetics elements* may then be added to this assemblage to account for the nonflow transport processes that may occur within a given zone. Thus, for example, the idealization presented above would, conceivably, be appropriate for the analysis of carbon monoxide dispersal. The indicated flow elements would model HVAC, infiltration, and exfiltration flow paths and the single kinetics element (labeled R_x) would model the kinetics of carbon monoxide generation within the furnace system. Note that in this case a well-mixed zone is associated with the furnace, a junction of the HVAC ducts, the exterior environment and each of the rooms of the building system.

Presently, the library of flow elements contains those indicated in Figure 1-4. The kinetics element and the convection-diffusion element are presented in the next section of this report; the other elements were presented earlier [Axley 1987].

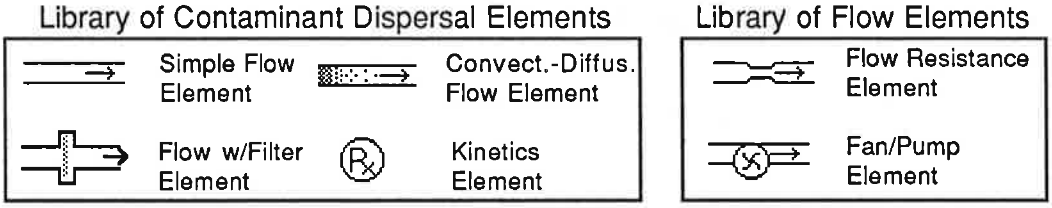


Figure 1-4 Current Library of Indoor Air Quality Analysis Elements

With each well-mixed zone we associate a set of discrete state variables with a distinct, but arbitrary point in the zone, the zone *node*. These discrete variables are meant to approximate the corresponding field variables in the zone at that point. For a system idealized as n well-mixed zones, then, the key discrete state variables would include:

$\{P\} \equiv \{P_1, P_2, \ldots P_n\}^T$: the vector of system pressure variables (1.1)

$\{T\} \equiv \{T_1, T_2, \ldots T_n\}^T$: the vector of system temperature variables (1.2)

and the vector of system concentration variables defined as:

- for the dispersal of a single species, α:

$$\{C\} \equiv \{{}^{\alpha}C_1, {}^{\alpha}C_2, \ldots {}^{\alpha}C_n\}^T \tag{1.3a}$$

- for the dispersal of two species, α and β:

$$\{C\} \equiv \{{}^{\alpha}C_1, {}^{\beta}C_1, {}^{\alpha}C_2, {}^{\beta}C_2, \ldots {}^{\alpha}C_n, {}^{\beta}C_n\}^T \tag{1.3b}$$

- etc.

where the subscripts are zone/node indices. These variables will be referred to as the *system (state) variables.*

With each element "e" in the system assembly we associate one or more *element nodes*. With each node we associate variables that define the state of the element – the *element (state) variables,* which will normally be subsets of the system variables[2], and note their association with the system variables. Thus, for example, a contaminant dispersal element having three nodes, i, j, and k, would have the element state variables;

- for the dispersal of a single species, α:

$$\{C^e\} \equiv \{{}^{\alpha}C_i^e, {}^{\alpha}C_j^e, {}^{\alpha}C_k^e\}^T \tag{1.4a}$$

- for the dispersal of two species, α and β:

$$\{C^e\} \equiv \{{}^{\alpha}C_i^e, {}^{\beta}C_i^e, {}^{\alpha}C_j^e, {}^{\beta}C_j^e, {}^{\alpha}C_k^e, {}^{\beta}C_k^e\}^T \tag{1.4b}$$

- etc.

where we shall use the symbol $\{C^e\}$ to represent the vector of element variables, in

2 As subsets of the system variables, one must distinguish , mathematically, these element variables from the system variables even though, most often, there will be no physical distinction.

general.

With these element variables in hand, *element equations* are formulated that describe the specific mass and/or energy transport phenomena that the element is meant to represent and, by demanding conservation of mass or energy transport at each of the system nodes, these element equations are then *assembled* to form *system equations* governing the behavior of the building air flow system as a whole.

From a practical point of view, the element assembly approach is intuitively satisfying and allows consideration of systems of arbitrary complexity. From a research and development point of view this approach separates the general problem of indoor air quality analysis into two primary subproblems; element development and development of solution method. Research efforts can, thus, focus on the modeling of specific transport phenomena to develop improved or new elements or, alternatively, focus on developing improved methods of solving the resulting equations while accounting for the complex coupling that exists between the thermal, dispersal, and flow analysis problems.

The approach has been formulated to be completely analogous and compatible with approaches based upon Generalized Finite Element Method [Zienkiewicz, 1983] solutions of the microscopic equation of motion for fluids and makes use of the numerical methods and computational strategies that have been developed to support the Finite Element and associated methods. It is expected that this compatibility will, eventually, allow the analyst to employ mixed idealizations of building air flow systems wherein a portion of the building air flow system is modeled in detail using microscopic elements (e.g., elements based upon Finite Element approximations of the governing microscopic continuum equations) while the rest of the air flow system is modeled using discrete elements. In this way the analyst may study the details of dispersal in one area of the system, accounting for whole-system interaction, without the overhead of modeling the entire system microscopically. The one-dimensional convection-diffusion element presented in the next section represents the first step in this direction.

2. Contaminant Dispersal Analysis

Multi-zone building contaminant dispersal analysis theory has placed a singular emphasis on contaminant dispersal driven by flow mass transport processes [Sinden, 1979, Sandberg 1984, Walton 1985] even though it has long been recognized that the dispersal of many important indoor air contaminants are affected by other mass transport processes as well, most notably, processes associated with reaction, sorption, and settling phenomena. The flow-element-assembly formulation of multi-zone building contaminant dispersal analysis theory developed during Phase II of the current project provides a conceptual framework to extend existing dispersal analysis theory to account for these other mass transport processes. Extending the flow-element-assembly approach, this section presents a general formulation of a multi-zone contaminant dispersal analysis theory that provides a basis for the development of more complete models of contaminant dispersal in buildings.

The general formulation of multi-zone contaminant dispersal theory is straightforward. We first establish a restricted, but very general, form for equations that will be used to describe mass transport at the element level[1]. Then, by establishing the correspondence between the element concentration variables, $\{\mathbf{C}^e\}$, and the system concentration variables, $\{\mathbf{C}\}$, and demanding species mass conservation at each of the system nodes we show that these element equations may be assembled to form equations governing the system as a whole. Consideration of boundary conditions, the qualitative character of these equations, and the solution of these equations was presented earlier [Axley 1987] and, therefore, will not be emphasized here.

2.1 Element Equations

As indicated above, it will be useful to distinguish those elements that model the transport of mass from zone to zone by flow processes from those elements that model the transport mass within a zone from species to species (e.g., by chemical or radio-chemical reaction) or, possibly, from species to the environment of the zone itself (e.g., by chemical or radio-chemical decay to a "noncontaminant" product, absorption, adsorption, or settling processes). In either case, we shall attempt to describe the *behavior* of an element by equations of the form:

$$\{\mathbf{w}^e\} = L^e(\{\mathbf{C}^e\}) - \{\mathbf{g}^e\} \tag{2.1}$$

where;

$\{\mathbf{w}^e\}$ is a vector of species mass transport rates <u>into</u> the element

$L^e(\{\mathbf{C}^e\})$ is a transformation of $\{\mathbf{C}^e\}$ that has the <u>form</u> of a linear transformation that is specific to a given class of elements

[1] That is to say, to model specific instances of mass transport phenomena in a building's air flow system.

$\{g^e\}$ is a vector of element derived species generation rates.

Element Mass Transport Rates The vector of species mass transport rates, $\{w^e\}$, for the dispersal of a single species α, may be represented diagrammatically as shown below for a hypothetical three-node flow element and a single-node kinetics element, where the arrows indicate positive mass transport rates. In the case of the flow element, mass is transported physically

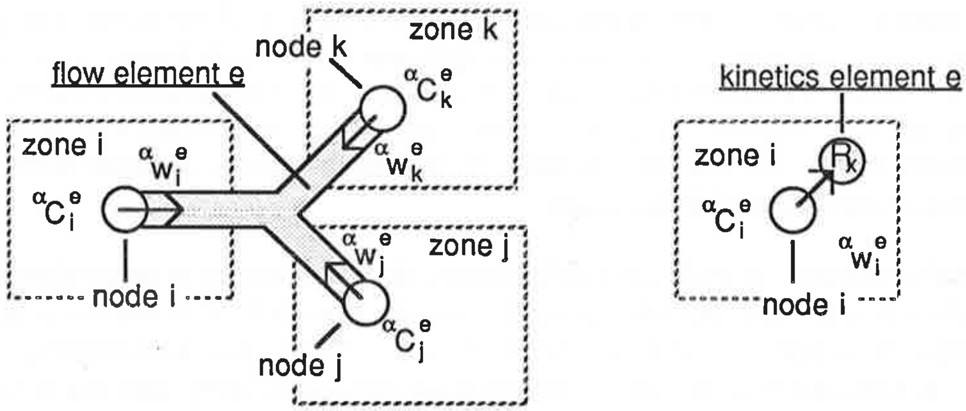


Figure 2-1 Element Mass Transport Rate Variables

by the air flow moving from the zone into the element and, thus, the arrows used indicate the sense or direction of the averaged or bulk fluid velocity – a common convention. For the kinetics element mass transport is somewhat more subtle; mass may not literally be transported out of the zone, rather mass of species α is, typically, converted from a form that is considered to be a contaminant to another form (e.g., another compound or phase) that is not of any special interest. Thus, for the kinetics element the arrow indicating mass transport is directed into the "element" from the zone node to indicate only that mass of species α is being removed from the zone by the element. It should be noted that:

> for each of the element concentration variables, C_i^e, there exists a corresponding element mass transport rate variables, w_i^e.

This will also be true for the dispersal of more than one species.

For contaminant dispersal involving multiple species, then, a single physical flow element might be thought to transport each individual species from zone-to-zone while a kinetics element might be thought to transport mass, by conversion, from each of the species to any or all of the other species and/or from any of the species to a form that has no special interest, within the single zone associated with the kinetics element. Extending this notion of an element, a combined flow and kinetics element, that not only transports mass of one species from one zone to another, but transports, by conversion, that species to another species or form during the flow passage, is also not only conceivable but reasonable for many reactive contaminants such as NO_2 and the radon chain of decay products. (Inasmuch as it is difficult to represent these possible multi-species mass transport/conversion phenomena diagrammatically we shall not attempt to do so, here.)

Element Transformation Operator The element transformation operator $L^e()$ is restricted to the form of a linear transformation:

$$L^e(\{C^e\}) \equiv [x^e]\{C^e\} + [y^e]\frac{d\{C^e\}}{dt} + [z^e]\frac{d^2\{C^e\}}{dt^2} + \ldots \tag{2.2}$$

where;

$[x^e]$, $[y^e]$, $[z^e]$	are transformation coefficient matrices
$[x^e]$	is the *element transport matrix*
$[y^e]$	is the *element mass matrix*

but we admit transformation coefficient-matrices that may, in fact, vary with time and/or depend, nonlinearly, on $\{C^e\}$:

$$[x^e] = [x^e(t, \{C^e\})] \quad \text{, in general} \tag{2.3a}$$

$$[y^e] = [y^e(t, \{C^e\})] \quad \text{, in general} \tag{2.3b}$$

$$[z^e] = [z^e(t, \{C^e\})] \quad \text{, in general} \tag{2.3c}$$

etc.

thus a practically endless variety of element equations may be formulated that have this form and, as such, the restriction to this form should not lead to any serious limitation.

Simple Flow Elements By assuming flow through a two-node flow element is practically instantaneous and well-mixed, the mass transport of a single species, say α, from element node i to j, due to an air mass flow rate $w^e(t)$ from i to j, may be described by the following element equations [Axley 1987]:

$$\begin{Bmatrix} {}^{\alpha}w_i^e \\ {}^{\alpha}w_j^e \end{Bmatrix} = w^e(t)\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}\begin{Bmatrix} {}^{\alpha}C_i^e \\ {}^{\alpha}C_j^e \end{Bmatrix} \quad ; \quad w^e(t) \geq 0 \tag{2.4a}$$

or, in this case we have:

$$[x^e] \equiv [f^e] = w^e(t)\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \quad ; \quad [y^e] = [0] \quad ; \quad [z^e] = [0] \quad ; \quad \{g^e\} = \{0\} \tag{2.4b}$$

$$\{w^e\} = \{\,{}^{\alpha}w_i^e\,,\ {}^{\alpha}w_j^e\,\}^T \tag{2.4c}$$

where we identify the element transport matrix for this case as the *element mass flow rate matrix*, $[f^e]$. It should be noted that the transformation matrix $[x^e]$ is seen to vary with time to account for the time variation of flow through the element. (Figure 2-2, below, should help to clarify the meaning of the element variables in this case.)

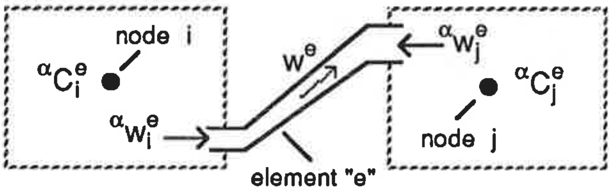


Figure 2-2 Simple Two-Node Contaminant Dispersal Flow Element Variables

Simple Flow Element with Filter The simple flow element equations, above, may be modified to account for the action of a filter that removes a fraction, η, of the contaminant as it passes through the element [Axley 1987], to yield the following element equations;

$$\left\{\begin{matrix} {}^{\alpha}w_i^e \\ {}^{\alpha}w_j^e \end{matrix}\right\} = w^e(t)\begin{bmatrix} 1 & 0 \\ (\eta - 1) & 0 \end{bmatrix}\left\{\begin{matrix} {}^{\alpha}C_i^e \\ {}^{\alpha}C_j^e \end{matrix}\right\} \tag{2.5a}$$

or, in this case we have:

$$[x^e] \equiv [f^e] = w^e(t)\begin{bmatrix} 1 & 0 \\ (\eta - 1) & 0 \end{bmatrix} \ ;\ [y^e] = [0] \ ;\ [z^e] = [0] \ ;\ \{g^e\} = \{0\} \tag{2.5b}$$

where, again, we identify the element transport matrix as the *element mass flow rate matrix*, $[f^e]$. In this case the time variation of the first transformation matrix, $[x^e]$, could be due to both the time variation of flow through the element and the time variation of the filter efficiency, $\eta = \eta(t)$. Furthermore, it should be recognized that the filter efficiency will, in general, vary with each contaminant so that this first transformation matrix may be expected to be species dependent. Following the notational convention established to distinguish species types (i.e., the use of a leading superscript) we shall indicate this species dependency as $[{}^{\alpha}x^e]$, $[{}^{\beta}x^e]$, ... for species α, β, ... where:

$$[{}^{\alpha}\mathbf{x}^{e}] \equiv [{}^{\alpha}\mathbf{f}^{e}] = w^{e}(t)\begin{bmatrix} 1 & 0 \\ ({}^{\alpha}\eta - 1) & 0 \end{bmatrix} \tag{2.5c}$$

2.2 System Equations

Equations governing the dispersal of contaminants in the system as a whole may be assembled from element equations of the form of equation (2.1) by first transforming the element equations so that they are expressed in terms of system variables. To this end we recognize that there exists a one-to-one correspondence between an element's concentration variables, $\{C^e\}$, and the system's concentration variables, $\{C\}$, that may be described by a simple Boolean transformation as:

$$\{C^e\} = [B^e]\{C\} \tag{2.6}$$

where;

$[B^e]$ is an m x n Boolean Transformation matrix (i.e., consisting of only ones and zeros) for an m-node element within an n-node system idealization

The Boolean transformation is simply a means to express the <u>equality</u> of each of the element concentrations variables with its associated system concentration variable within the framework of concise vector notation; it defines the relation between the (larger) vector of system concentration variables and the (smaller) vector of a specific elements concentration variables, a subset of the system variables.

This same Boolean transformation matrix may be used to transform the vector element mass transport rates, $\{w^e\}$, into a "system-sized" vector of mass transport rates for element "e", $\{W^e\}$, as:

$$\{W^e\} = [B^e]^T\{w^e\} \tag{2.7}$$

This vector $\{W^e\}$ will have the same number of elements as the system concentration vector $\{C\}$, providing a correspondence between each system concentration variable and a "system-sized" mass transport rate for the element "e". It represents the net species mass transport rate from <u>each</u> of the system nodes into a specific element "e" and, therefore, the sum of these mass transport vectors for all elements in the system assemblage will equal a vector describing the total mass transport from the system nodes into all elements combined.

$$\sum_{e=a,b,\ldots} \{W^e\} = \left\{ \begin{array}{c} \text{total species} \\ \text{mass transport} \\ \text{from each node} \\ \text{into connected elements} \\ \text{at each node} \end{array} \right\} \qquad (2.8)$$

Demanding the conservation of species mass at each of the system nodes, the sum of the quantity above plus the rate of change of species mass within each zone must be equal to any species mass generated within the zone:

$$\sum_{e=a,b,\ldots} \{W^e\} + [M]\frac{d\{C\}}{dt} = \{G\} \qquad (2.9)$$

where;

[M] the (diagonal) *zone air volume mass matrix* defined as (for n zones):

• for a single species (i.e., with {C} defined by equation (1.3a)):

$$[M] = \begin{bmatrix} M_1 & 0 & \ldots & 0 \\ 0 & M_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & M_n \end{bmatrix} \qquad (2.10a)$$

• for two species[2] (i.e., with {C} defined by equation (1.3b)):

$$[M] = \begin{bmatrix} M_1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & M_1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & M_2 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & M_2 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & M_n & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & M_n \end{bmatrix} \qquad (2.10b)$$

• etc.

M_i = the mass of the air in the volume of zone i

[G] is the zone species generation rate vector, defined as

• for a single species, α:

$$\{G\} \equiv \{ {}^{\alpha}G_1, {}^{\alpha}G_2, \ldots {}^{\alpha}G_n\}^T \qquad (2.11a)$$

2 One could conceivably associate a different "active" zone air volume with each species and have, in this case, a diagonal mass matrix of the form: diag (${}^aM_1$, ${}^bM_1$, ${}^aM_2$, ${}^bM_2$, ... ${}^aM_n$, ${}^bM_n$,).

- for two species, α and β:

$$\{G\} \equiv \{ {}^{\alpha}G_1, {}^{\beta}G_1, {}^{\alpha}G_2, {}^{\beta}G_2, \ldots {}^{\alpha}G_n, {}^{\beta}G_n\}^T \tag{2.11b}$$

- etc.

${}^{\alpha}G_i$ = the mass generation rate of species α in zone i

General Expression for Multi-Zone Dispersal Analysis Substituting the transformation relations, equations (2.6) and (2.7), along with the general form of the element equations, equation (2.1), into the species mass conservation relation, equation (2.9), we obtain the final result — a general expression for multi-zone contaminant dispersal analysis:

$$[W]\{C\} + [M]\frac{d\{C\}}{dt} + [Z]\frac{d^2\{C\}}{dt^2} + \ldots = \{G\} \tag{2.12a}$$

where;

$$[W] = \sum_{e=a,b,\ldots} [B^e]^T [x^e] [B^e]$$ the *system (mass) transport matrix* (2.12b)

$$[M] = [M] + \sum_{e=a,b,\ldots} [B^e]^T [y^e] [B^e]$$ the *system mass matrix* (2.12c)

$$[Z] = \sum_{e=a,b,\ldots} [B^e]^T [z^e] [B^e] \tag{2.12d}$$

etc.

$$\{G\} = \{G\} + \sum_{e=a,b,\ldots} [B^e]^T \{g^e\}$$ the *system generation vector* (2.12e)

It should be noted that in this general formulation the system mass matrix and system generation vector have element contributions that add to the more familar zonal values. The kinetics element, that will be presented in Section 2.2, will be seen to provide element contributions to the system generation vector. The convection-diffusion element, that will be presented in Section 2.3, will be seen to provide element contributions to both the system mass matrix and the system generation vector.

The Assembly Operator The summation and Boolean transformation of element matrices, contained in the expressions above, is an operation that recurs frequently in the Finite Element and related discrete modeling literature and, therefore, has come to be defined as a standard operation – the *assembly operation* – designated by the symbol **A**, the so-called *assembly operator* where;

$$\underset{e=a,b,\ldots}{\mathbf{A}} \{v^e\} \equiv \sum_{e=a,b,\ldots} [B^e]^T \{v^e\} \quad \text{for element vector assembly} \qquad (2.13a)$$

and

$$\underset{e=a,b,\ldots}{\mathbf{A}} [m^e] \equiv \sum_{e=a,b,\ldots} [B^e]^T [m^e] [B^e] \quad \text{for element matrix assembly} \qquad (2.13b)$$

The assembly operator is therefore simply a generalization of the conventional summation operator, Σ, and equal to the summation operator when the Boolean transformation matrices equal the identity matrix.

The assembly operation is important, theoretically, in that it provides the necessary formal definition of the assembly process required for subsequent mathematical analysis. It does not, however, define an efficient numerical procedure for assembling the element arrays needed to form the system equations for practical contaminant dispersal analysis — the indicated Boolean transformations involve multiplications by either zero or one and, therefore, need not be actually implemented. Practically, then, the assembly operation is carried out using relatively simple algorithms that accumulate element array terms within system array memory locations according to the physical connectivity of each element. The "LM Algorithm" presented by Bathe [Bathe 1982] provides an example of one possible algorithm.

Relation to the Phase II Formulation Previously [Axley 1987], we considered contaminant dispersal due only to flow mass transport via simple elements, with and without filtration, and at that time the element transport matrices, $[x^e]$, defined above by equations (2.4b) and (2.5b) were identified as *element mass flow rate matrices* and given the symbol $[f^e]$. These element mass flow matrices were then assembled to form the system transport matrix identified, then, as the *system flow matrix* and given the symbol [F]. We shall see that, in the general case, the system transport matrix [W], defined above, will be equal to the sum of the system flow matrix and what will be called the system kinetics matrix, [K], assembled from element kinetics transport matrices, $[k^e]$, as:

$$[W] = [F] + [K] \qquad (2.14a)$$

$$[F] = \underset{\text{flow elements}}{\mathbf{A}} [f^e] \qquad (2.14b)$$

$$[K] = \underset{\text{kinetics elements}}{\mathbf{A}} [k^e] \tag{2.14c}$$

2.3 Solution of System Equations

The contaminant dispersal elements developed to date are all described in terms of first order linear transformations (i.e., involve only $[x^e]$ and $[y^e]$ transformation matrices) having, in some cases, time varying element transport matrices (i.e., having $[x^e]$ = $[x^e(t)]$, consequently system equations assembled from these element equations will be limited to the first order form:

$$[W]\{C\} + [M]\frac{d\{C\}}{dt} = \{G\} \tag{2.15}$$

where the system transport matrix will, in general, vary with time: $[W] = [W(t)]$.

System equations of this form are identical, in form, to those system equations that result from idealizations restricted to assemblages of simple flow elements. Therefore, the procedures used to account for boundary conditions and possibilities of massless nodes, the solution options that may be considered (i.e., steady state analysis, eigenanalysis, and dynamic analysis), and the numerical methods that may be employed to affect these solutions are identical to those discussed earlier [Axley 1987] and will not be considered here. The qualitative character of equations (2.15) depends critically upon the qualitative character of the element equations from which they are assembled, therefore, the qualitative analysis presented earlier [Axley 1987] will have to be reconsidered with the introduction of each new element.

3. Interactive Contaminant Dispersal Analysis

Often, indoor air quality analysis will involve consideration of several contaminants and their dispersal in a building. Some of these contaminants may:

a) be absorbed or adsorbed by building materials or other contaminant particles,

b) settle from suspension or precipitate from (gaseous) solution, or

c) decay radiochemically, decompose chemically, or react with other contaminants to produce product contaminants (or other substances that are of no particular interest).

That is to say, contaminant dispersal processes may be complicated by the *kinetics* of:

a) *sorption processes,*

b) *settling* or *precipitation processes*, or

c) *chemical* or *radio-chemical reaction processes*

that must be accounted for.

In this section we shall introduce a general approach to extend noninteractive contaminant dispersal theory to account for mass transport phenomena governed by the kinetics of these processes, based upon the principals of *reaction kinetics*, to develop an *interactive contaminant dispersal theory*. We shall set the stage by first considering possible forms of system equations for multiple, noninteractive contaminant dispersal analysis then, after a review of some basic concepts of reaction kinetics, go on to develop the so-called *kinetics element equations* that will become the basis of the interactive contaminant dispersal theory.

3.1 Multiple, Noninteractive, Contaminant Dispersal

The dispersal of each contaminant of a given set of noninteractive contaminants will be governed by the single-species contaminant dispersal equation, thus the dispersal of the noninteractive contaminant set may be represented by a set of equations of the form:

$$
\begin{aligned}
&[{}^{\alpha}F]\{{}^{\alpha}C\} + [M]\frac{d\{{}^{\alpha}C\}}{dt} = \{{}^{\alpha}G\} \\
&[{}^{\beta}F]\{{}^{\beta}C\} + [M]\frac{d\{{}^{\beta}C\}}{dt} = \{{}^{\beta}G\} \\
&\quad \cdots
\end{aligned}
\tag{3.1}
$$

where;

$\alpha, \beta, \ldots$ are species indices

$[{}^{\alpha}\mathbf{F}]$ is defined by equations (2.5c) and (2.13b)

$[\mathbf{M}]$ is by equations (2.10)

(Note that, in general, the flow matrices for each species may not be identical because individual flow elements may act to filter contaminant species differently.)

Contaminant dispersal analysis for the set could, then, be computed by simply completing a separate analysis for each species of interest. If, however, the system characteristics change with time (e.g., airflow within the building is nonsteady and thus the flow matrix, [F], changes with time) <u>and</u> the flow matrices for each species are identical then it would be computationally more efficient to simultaneously compute the response of each species while stepping through time as suggested by rewriting equation (3.1) in the form:

$$[{}^{*}\mathbf{F}]\,[\{{}^{\alpha}\mathbf{C}\},\{{}^{\beta}\mathbf{C}\},\ldots] + [\mathbf{M}]\left[\frac{d\{{}^{\alpha}\mathbf{C}\}}{dt},\frac{d\{{}^{\alpha}\mathbf{C}\}}{dt},\ldots\right] = [\{{}^{\alpha}\mathbf{G}\},\{{}^{\alpha}\mathbf{G}\},\ldots] \tag{3.2}$$

for; $\quad [{}^{\alpha}\mathbf{F}] = [{}^{\beta}\mathbf{F}] = \ldots \equiv [{}^{*}\mathbf{F}]$

As an alternative approach, that will help set the stage for multiple interactive contaminants, we may write the <u>uncoupled</u> set of equations given by equation (3.1) as an expanded system of equations of the form:

$$[\mathbf{F}]\{\mathbf{C}\} + [\mathbf{M}]\frac{d\{\mathbf{C}\}}{dt} = \{\mathbf{G}\} \tag{3.3a}$$

where system variables are organized by species for each node of the system idealization as:

$$\{\mathbf{C}\} \equiv \{{}^{\alpha}C_1, {}^{\beta}C_1, \ldots\ {}^{\alpha}C_2, {}^{\beta}C_2, \ldots\ {}^{\alpha}C_n, {}^{\beta}C_n, \ldots\}^T \tag{3.3b}$$

$$\{\mathbf{G}\} \equiv \{{}^{\alpha}G_1, {}^{\beta}G_1, \ldots\ {}^{\alpha}G_2, {}^{\beta}G_2, \ldots\ {}^{\alpha}G_n, {}^{\beta}G_n, \ldots\}^T \tag{3.3c}$$

The system flow transport matrix, [F], in this case, may be assembled by species and, then, by element as:

$$[\mathbf{F}] = \underset{e=a,b,\ldots}{\mathbf{A}}\left[\underset{\sigma=\alpha,\beta,\ldots}{\mathbf{A}}\ [{}^{\sigma}\mathbf{f}^{e}]\right] \tag{3.3d}$$

where;

σ is a general species index ($\alpha, \beta, \ldots$ are specific species indices) e is a general element index (a, b, ... are specific element indices)

The inner assembly sum, by species, may be thought to generate element equations for a *noninteractive, multi-species flow element.* This multi-species flow element could, then, be assembled in the usual manner to form the system equations. For three contaminant species, α, β, γ, the noninteractive, multi-species, flow element transport matrix (with filtration of each species) would have the form:

$$\left[\underset{\sigma=\alpha,\beta,\ldots}{\mathbf{A}} [{}^{\sigma}f^{e}] \right] = \text{the } \textit{noninteractive, multi-species, flow element transport matrix}$$

$$= (w^{e}) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ ({}^{\alpha}\eta - 1) & 0 & 0 & 0 & 0 & 0 \\ 0 & ({}^{\beta}\eta - 1) & 0 & 0 & 0 & 0 \\ 0 & 0 & ({}^{\gamma}\eta - 1) & 0 & 0 & 0 \end{bmatrix} \quad (3.4a)$$

for element flow from node i to node j and element concentration variables organized as:

$$\{C^{e}\} \equiv \{ {}^{\alpha}C_i^{e}, {}^{\beta}C_i^{e}, {}^{\gamma}C_i^{e}, {}^{\alpha}C_j^{e}, {}^{\beta}C_j^{e}, {}^{\gamma}C_j^{e} \}^{T} \quad (3.4b)$$

This noninteractive, multi-species, element flow matrix is seen to be very sparse. Consequently, assemblies of such elements would result in extremely sparse system equations. It would, therefore, be computationally impractical to employ this approach for noninteractive, multi-species, contaminant dispersal analysis — one should use the strategies indicated by equations (3.1) or (3.2) instead. It will be seen, however, that kinetic interactions among contaminant species will act to couple the species variables at the system nodes (zones) with the result that system matrices will tend not only to be much less sparse, but reasonably well-banded. The use of noninteractive, multi-species, flow elements will, therefore, become attractive when combined with the kinetics elements presented subsequently for interactive contaminant dispersal analysis.

The program CONTAM87, documented in the second part of this report, organizes system and element variables following equations (3.3b), (3.3c), and (3.4b) and makes use of the double assembly process for simple flow elements defined by equations (3.3d) and (3.4a).

3.2 Basic Concepts of Reaction Kinetics

Reaction kinetics involves the study of the rate of change of chemical components in a single or related series of chemical reactions. Some basic concepts of (isothermal, constant volume) gas reaction kinetics will be reviewed here; greater detail may be found in one of several texts on the subject [Moore 1981, Nicholas 1976, Walas 1959]. Much of this material may also be applied to sorption, settling or precipitation, radio-chemical, and other chemical phenomena that may be important for some interactive contaminants in indoor air quality analysis.

A general form of a chemical reaction involving reactants, α, β, ..., that react to form products, ρ, σ, ..., may be represented as;

$$\alpha + \beta + \ldots \xrightarrow{\text{catalyst}} \rho + \sigma + \ldots \tag{3.5}$$

where $\xrightarrow{\text{catalyst}}$ indicates the possible affect of catalysts on the reaction.

Given the rate of change of a selected component's concentration, say α, is defined as;

$${}^{\alpha}R \equiv \frac{d\,{}^{\alpha}C}{dt} \quad ;\textit{rate of reaction} \text{ (in terms of reactant } \alpha) \tag{3.6}$$

where:

${}^{\alpha}C$ is the concentration of species α measured in terms of mass of α per unit mass of air (i.e., strictly speaking the *mass fraction* of α)

and the stoichiometry of the reaction, expressed in terms of relative masses, ${}^{\alpha}m$, ${}^{\beta}m$, ..., of reactants and products as;

$${}^{\alpha}m\,\alpha + {}^{\beta}m\,\beta + \ldots \xrightarrow{\text{catalyst}} {}^{\rho}m\rho + {}^{\sigma}m\,\sigma + \ldots \tag{3.7}$$

the rate of change of the other components' concentrations may be related to that of the selected component's as;

$${}^{\alpha}R = \left(\frac{{}^{\beta}m}{{}^{\alpha}m}\right) {}^{\beta}R = \ldots = -\left(\frac{{}^{\rho}m}{{}^{\alpha}m}\right) {}^{\rho}R = -\left(\frac{{}^{\sigma}m}{{}^{\alpha}m}\right) {}^{\sigma}R \tag{3.8}$$

thus, the rate of a given chemical reaction may be described in terms of the rate of change of concentration of any one of the reactants or products.

In general, the rate of a given chemical reaction may depend upon a variety of factors including reactant and catalyst concentrations, temperature, T, pressure, P, and the detailed mechanisms of the chemical reaction, therefore, rate expressions take the general functional form of;

$$^{\alpha}R = {^{\alpha}R}(\,^{\alpha}C,\ ^{\beta}C, \ldots\ ^{\rho}C,\ ^{\sigma}C, \ldots T, P, \ldots) \tag{3.9}$$

Constant Rate Expressions In some instances the rate of reaction may remain more or less constant:

$$^{\alpha}R = {^{\alpha}R_o} \tag{3.10a}$$

or depend solely on temperature and pressure:

$$^{\alpha}R = {^{\alpha}R_o}(T, P) \tag{3.10b}$$

Examples include the catalytic decomposition of some gases, such as ammonia, the radioactive decay of isotopes with very long half lives, such as Ra^{226}, the controlled burning of fossil fuels, and other relatively slow reactions driven by reactant and product concentrations that remain, more or less, constant over the time of interest.

Power Law Rate Expressions In many cases the explicit form of a rate expression will prove to be rather complex. In some cases, however, (empirical or semi-empirical) rate expressions may be employed that take the form of so-called power expressions:

$$^{\alpha}R = {^{\alpha}\kappa}(T,P)(\,^{\alpha}C)^{a}(\,^{\beta}C)^{b} \ldots (\,^{\rho}C)^{r}(\,^{\sigma}C)^{s} \ldots \tag{3.11}$$

where:

$^{\alpha}\kappa$ = the *rate constant* [=] 1/time
a, b, ...r, s, ... = constant exponents

Reactions governed by such power expression are classified by their *overall order* - the sum of the constant exponents - or by their *order with respect to each kinetically active component* - the constant exponent of that component. For the reaction described by equation (3.11), then, the overall order will be (a + b + ... + r + s + ...) and the order with respect to component α will be simply (a).

First Order Rate Expressions Rate expressions for certain general classes of reactions, including single-reactant, consecutive, opposing, and concurrent first order reactions, often take the form of linear combinations of contaminant concentrations:

$$\{\mathbf{R}\} = -[\kappa]\{\mathbf{C}\} + \{\mathbf{R}_o\} \tag{3.12a}$$

or

$$\begin{Bmatrix} {}^{\alpha}R \\ {}^{\beta}R \\ \cdots \\ {}^{\sigma}R \end{Bmatrix} = -\begin{bmatrix} {}^{\alpha\alpha}\kappa & -{}^{\alpha\beta}\kappa & \cdots & -{}^{\alpha\sigma}\kappa \\ -{}^{\beta\alpha}\kappa & {}^{\beta\beta}\kappa & \cdots & -{}^{\beta\sigma}\kappa \\ \cdots & \cdots & \cdots & \cdots \\ -{}^{\sigma\alpha}\kappa & -{}^{\sigma\beta}\kappa & \cdots & {}^{\sigma\sigma}\kappa \end{bmatrix} \begin{Bmatrix} {}^{\alpha}C \\ {}^{\beta}C \\ \cdots \\ {}^{\sigma}C \end{Bmatrix} + \begin{Bmatrix} {}^{\alpha}R_o \\ {}^{\beta}R_o \\ \cdots \\ {}^{\sigma}R_o \end{Bmatrix} \tag{3.12b}$$

where we have included the constant component, $\{\mathbf{R}_o\}$, for completeness and recognize that, again, the rate coefficient matrix, $[\kappa]$, and the constant component vector, $\{\mathbf{R}_o\}$, will, in general, vary with temperature and pressure. It should be noted that equation (3.12b) has been written so that all rate coefficients, ${}^{\psi\omega}\kappa$; $\psi, \omega = \alpha, \beta, \ldots \sigma$, will be positive for realistic reactions.

In fact, it is possible, in principal, to *linearize* any given rate expression about some (current) state of concentration, say $({}^{\alpha}C_o, {}^{\beta}C_o, \ldots)$, by employing a Taylor's expansion about that state, to obtain an approximate rate expression expressed as the sum of a series of first order rate expressions, as:

$$\begin{aligned} {}^{\alpha}R({}^{\alpha}C, {}^{\beta}C, \ldots) \approx {}^{\alpha}R({}^{\alpha}C_o, {}^{\beta}C_o, \ldots) &+ \frac{\partial\, {}^{\alpha}R({}^{\alpha}C_o, {}^{\beta}C_o, \ldots)}{\partial\, {}^{\alpha}C}({}^{\alpha}C - {}^{\alpha}C_o) \\ &+ \frac{\partial\, {}^{\alpha}R({}^{\alpha}C_o, {}^{\beta}C_o, \ldots)}{\partial\, {}^{\beta}C}({}^{\beta}C - {}^{\beta}C_o) + \ldots \end{aligned} \tag{3.13}$$

that, together with equation (3.8), may be used to form a linearized system of first order rate expressions of the form of equations (3.12). One could, conceivably, employ this linearization strategy, within an appropriate nonlinear solution method, to account for arbitrarily complex kinetics. The *first order kinetics element*, that will be presented subsequently, provides a first step in this direction.

Linear systems of first order reaction expressions are defined by the characteristics of their *reaction rate coefficient matrices*, $[\kappa]$. To gain a better understanding of the types and characteristics of such reactions several specific classes of reactions are described below.

Single-Reactant First Order Reactions For reactions involving single contaminant reactants that decompose or decay to form products (that are of little particular interest):

$$\begin{aligned} &\alpha \rightarrow \text{products} \\ &\beta \rightarrow \text{products} \\ &\quad \cdots \\ &\sigma \rightarrow \text{products} \end{aligned} \qquad (3.14)$$

the rate coefficient matrix takes the following form:

$$[\kappa] = \begin{bmatrix} {}^{\alpha\alpha}\kappa & 0 & \cdots & 0 \\ 0 & {}^{\beta\beta}\kappa & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & {}^{\sigma\sigma}\kappa \end{bmatrix} \qquad (3.15)$$

Consecutive First Order Reactions The radioactive decay chain of Radon gas is an especially important example of a consecutive first order reaction series. The reaction rate expression for a simple two-step consecutive reaction will be discussed first then the general case will be considered.

For a two-step consecutive reaction series involving a single reactant at each step:

$$\begin{aligned} &\alpha \rightarrow \beta \\ &\beta \rightarrow \text{products} \end{aligned} \qquad (3.16)$$

with reactions governed by rate expressions of the following form:

$$\begin{aligned} {}^{\alpha}R &= -{}^{\alpha\alpha}\kappa\, {}^{\alpha}C \\ {}^{\beta}R &= {}^{\beta\alpha}\kappa\, {}^{\alpha}C - {}^{\beta\beta}\kappa\, {}^{\beta}C \end{aligned} \qquad (3.17)$$

the matrix of rate coefficients becomes:

$$[\kappa] = \begin{bmatrix} {}^{\alpha\alpha}\kappa & 0 \\ -{}^{\beta\alpha}\kappa & {}^{\beta\beta}\kappa \end{bmatrix} \qquad (3.18)$$

Here, the generalization to a multi-step reaction involving single reactants at each step is straightforward. For a general multi-step consecutive reaction series:

$$
\begin{aligned}
&\alpha \rightarrow \beta \\
&\beta \rightarrow \ldots \\
&\ldots \\
&\rho \rightarrow \sigma \\
&\sigma \rightarrow \text{products}
\end{aligned}
\tag{3.19}
$$

governed by rate expressions of the form:

$$
\begin{aligned}
{}^{\alpha}R &= -{}^{\alpha\alpha}\kappa\,{}^{\alpha}C \\
{}^{\beta}R &= {}^{\beta\alpha}\kappa\,{}^{\alpha}C - {}^{\beta\beta}\kappa\,{}^{\beta}C \\
&\ldots \\
{}^{\sigma}R &= {}^{\sigma\rho}\kappa\,{}^{\rho}C - {}^{\sigma\sigma}\kappa\,{}^{\sigma}C
\end{aligned}
\tag{3.20}
$$

the matrix of rate coefficients becomes:

$$
[\kappa] = \begin{bmatrix}
{}^{\alpha\alpha}\kappa & 0 & \ldots & 0 & 0 \\
-{}^{\beta\alpha}\kappa & {}^{\beta\beta}\kappa & \ldots & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & {}^{\rho\rho}\kappa & 0 \\
0 & 0 & \ldots & -{}^{\rho\sigma}\kappa & {}^{\sigma\sigma}\kappa
\end{bmatrix}
\tag{3.21}
$$

Opposing First Order Reactions For simple reversible reactions involving a single reactant and single product:

$$
\alpha \leftrightarrow \beta
\tag{3.22}
$$

governed by rate expressions of the form:

$$
\begin{aligned}
{}^{\alpha}R &= -{}^{\alpha\alpha}\kappa\,{}^{\alpha}C + {}^{\alpha\beta}\kappa\,{}^{\beta}C \\
{}^{\beta}R &= {}^{\beta\alpha}\kappa\,{}^{\alpha}C - {}^{\beta\beta}\kappa\,{}^{\beta}C
\end{aligned}
\tag{3.23}
$$

the matrix of rate coefficients becomes:

$$[\kappa] = \begin{bmatrix} {}^{\alpha\alpha}\kappa & -{}^{\alpha\beta}\kappa \\ -{}^{\beta\alpha}\kappa & {}^{\beta\beta}\kappa \end{bmatrix} \tag{3.24}$$

For the more general case of a series of reversible reactions involving single reactants and products:

$$\alpha \leftrightarrow \beta \leftrightarrow \gamma \leftrightarrow \delta \leftrightarrow \ldots \tag{3.25}$$

governed by rate expressions of the form:

$$\begin{aligned} {}^{\alpha}R &= -{}^{\alpha\alpha}\kappa\,{}^{\alpha}C + {}^{\alpha\beta}\kappa\,{}^{\beta}C \\ {}^{\beta}R &= {}^{\beta\alpha}\kappa\,{}^{\alpha}C - {}^{\beta\beta}\kappa\,{}^{\beta}C + {}^{\beta\gamma}\kappa\,{}^{\gamma}C \\ {}^{\gamma}R &= {}^{\gamma\beta}\kappa\,{}^{\beta}C - {}^{\gamma\gamma}\kappa\,{}^{\gamma}C + {}^{\gamma\delta}\kappa\,{}^{\delta}C \\ &\ldots \end{aligned} \tag{3.26}$$

the matrix of rate coefficients takes the tridiagonal form:

$$[\kappa] = \begin{bmatrix} {}^{\alpha\alpha}\kappa & -{}^{\alpha\beta}\kappa & 0 & 0 & \ldots \\ -{}^{\beta\alpha}\kappa & {}^{\beta\beta}\kappa & -{}^{\beta\gamma}\kappa & 0 & \ldots \\ 0 & -{}^{\gamma\beta}\kappa & {}^{\gamma\gamma}\kappa & -{}^{\gamma\delta}\kappa & \ldots \\ 0 & 0 & -{}^{\delta\gamma}\kappa & {}^{\delta\delta}\kappa & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix} \tag{3.27}$$

Concurrent Linear Reactions Due to their linearity, rate coefficient matrices for concurrent linear reactions may simply be added to obtain an effective rate coefficient matrix for all reactions combined. For example, consider a set of concurrent reactions involving a single contaminant reactant, α, that decays or decomposes to produce products β, γ, σ :

$$\begin{aligned} &\alpha \rightarrow \beta \quad \text{; reaction "a"} \\ &\alpha \rightarrow \gamma \quad \text{; reaction "b"} \\ &\alpha \rightarrow \delta \quad \text{; reaction "c"} \end{aligned} \tag{3.28}$$

governed by rate expressions of the form:

$$
\begin{aligned}
{}^{\alpha}R &= -\left({}^{\alpha\alpha}_{a}\kappa + {}^{\alpha\alpha}_{b}\kappa + {}^{\alpha\alpha}_{c}\kappa\right){}^{\alpha}C \\
{}^{\beta}R &= {}^{\beta\alpha}_{a}\kappa\,{}^{\alpha}C \\
{}^{\gamma}R &= {}^{\gamma\alpha}_{b}\kappa\,{}^{\alpha}C \\
{}^{\delta}R &= {}^{\delta\alpha}_{c}\kappa\,{}^{\alpha}C
\end{aligned}
\tag{3.29}
$$

The matrix of rate coefficients is:

$$
[\kappa] = \begin{bmatrix}
\left({}^{\alpha\alpha}_{a}\kappa + {}^{\alpha\alpha}_{b}\kappa + {}^{\alpha\alpha}_{c}\kappa\right) & 0 & 0 & 0 \\
-{}^{\beta\alpha}_{a}\kappa & 0 & 0 & 0 \\
-{}^{\gamma\alpha}_{b}\kappa & 0 & 0 & 0 \\
-{}^{\delta\alpha}_{c}\kappa & 0 & 0 & 0
\end{bmatrix}
\tag{3.30a}
$$

which is seen to be the sum of the individual reaction rate coefficient matrices:

$$
= \begin{bmatrix}
{}^{\alpha\alpha}_{a}\kappa & 0 & 0 & 0 \\
-{}^{\beta\alpha}_{a}\kappa & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
+ \begin{bmatrix}
{}^{\alpha\alpha}_{b}\kappa & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-{}^{\gamma\alpha}_{b}\kappa & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
+ \begin{bmatrix}
{}^{\alpha\alpha}_{c}\kappa & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-{}^{\delta\alpha}_{c}\kappa & 0 & 0 & 0
\end{bmatrix}
\tag{3.30b}
$$

3.3 Kinetics Element Equations

The development of a *general kinetics element* is straightforward. Limiting our considerations to mass transport phenomena occurring within a specific zone "i", containing a set of contaminant species, α, β, γ, ..., we first identify the relevant element variables as:

$$\{C^e\} = \{{}^{\alpha}C_i^e, {}^{\beta}C_i^e, {}^{\gamma}C_i^e, \ldots\}^T \quad \text{; the element state variables} \tag{3.31}$$

and

$$\{w^e\} = \{{}^{\alpha}w_i^e, {}^{\beta}w_i^e, {}^{\gamma}w_i^e, \ldots\}^T \quad \text{; the element mass flow rate vector} \tag{3.32}$$

Then, assuming that the mass transport phenomena to be modeled is governed by the kinetics discussed above, a general kinetics element equation follows directly from the definition of rate of reaction, equation (3.6) and the general form of rate expressions, equation (3.9), as:

$$\{\mathbf{w}^e\} = -[\mathbf{M}_i^e]\{\mathbf{R}_i^e(\{\mathbf{C}^e\},T,P)\} \quad \text{: the } \textit{general kinetics element equation} \qquad (3.33a)$$

where;

$$[\mathbf{M}_i^e] \equiv \begin{bmatrix} M_i & 0 & 0 & \ldots \\ 0 & M_i & 0 & \ldots \\ 0 & 0 & M_i & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix} \qquad (3.33b)$$

$$\{\mathbf{R}_i^e(\{\mathbf{C}^e\},T,P)\} \equiv \begin{Bmatrix} {}^{\alpha}R({}^{\alpha}C_i^e, {}^{\beta}C_i^e, {}^{\gamma}C_i^e, \ldots T, P) \\ {}^{\beta}R({}^{\alpha}C_i^e, {}^{\beta}C_i^e, {}^{\gamma}C_i^e, \ldots T, P) \\ {}^{\gamma}R({}^{\alpha}C_i^e, {}^{\beta}C_i^e, {}^{\gamma}C_i^e, \ldots T, P) \\ \ldots \end{Bmatrix} \qquad (3.33c)$$

(The superscript e has been added to identify the specific kinetics element and the subscript i has been added to identify the specific node/zone being considered. The negative sign is needed as species mass transport "into" the element (i.e., removed from the zone) is defined to be positive.)

Using the notation introduced earlier (equations (2.1) and (2.2)) it is seen that the general kinetics element is one defined by the following element arrays:

$$[\mathbf{x}^e] = 0 \; ; \; [\mathbf{y}^e] = 0 \; ; \; [\mathbf{z}^e] = 0 \; ; \; \ldots \; ; \; \{\mathbf{g}^e\} = [\mathbf{M}_i^e]\{\mathbf{R}_i^e(\{\mathbf{C}^e\},T,P)\} \qquad (3.33d)$$

an element that is defined in terms of only element derived species generation rates.

If the rate expressions are constant rate expressions, then the element derived generation rate terms will simply add to any direct species generation rates specified within a zone, after equation (2.12e) as:

$$\{\mathrm{G}_i\} = \{\mathbf{G}_i\} + \underset{\text{kinetics elements}}{\mathrm{A}} [\mathbf{M}_i^e]\{\mathbf{R}_o^e\} \qquad (3.34)$$

where $\{\mathrm{G}_i\}$ and $\{\mathbf{G}_i\}$ are the subsets of the system vectors $\{\mathrm{G}\}$ and $\{\mathbf{G}\}$ corresponding to node/zone i. In these cases there will be no practical difference between the physical generation of species mass (e.g., by physical release of a contaminant) and the kinetics generation of species mass (e.g., by chemical or physical-chemical processes) and, therefore, the analyst may model either using simple noninteractive contaminant dispersal analysis techniques.

The form of equations (3.33) is deceptively simple. The rate expressions defining these element derived species generation rates depend on species concentration, in general, so that the general kinetics element introduces a nonlinear species generation contribution (i.e., a species generation rate that depends nonlinearly on the solution vector {**C**}), which is distinctly different from the (constant or time dependent) nodal direct generation contribution. The solution of the contaminant dispersal problem involving general kinetics elements will, therefore, require the application of a nonlinear solution strategy in the solution process. While this adds complexity to the analysis process it should not be difficult to develop an appropriate nonlinear solution strategy (e.g., using one or more of the strategies considered earlier to solve the nonlinear air flow analysis problem [Axley 1987]) for certain classes of kinetics.

Few interactive indoor contaminants have been studied in sufficient detail to completely define their kinetics, therefore, the development of nonlinear solution techniques for arbitrary nonlinear kinetics would be premature at this time. Instead we have limited our attention to kinetics described by linear systems of first order rate expressions (that lead directly to linear systems of equations for interactive contaminant dispersal analysis) to develop a practically useful interactive contaminant dispersal analysis method. It seems likely, however, that more complex kinetics may be modeled in the future by employing the combination of the linear method described below and the Taylor's expansion presented earlier (equation (3.13)) to linearize rate expressions, within an appropriate iterative solution scheme.

First Order Kinetics Element Equations For reaction kinetics described by systems of first order equations, equations (3.12):

$$\{\mathbf{R}_i^e(\{\mathbf{C}^e\}, T, P)\} = -[\kappa_i^e]\{\mathbf{C}^e\} + \{\mathbf{R}_{o_i}^e\} \tag{3.35}$$

the kinetics element equations (3.33) become:

$$\{\mathbf{w}^e\} = [\mathbf{M}_i^e][\kappa_i^e]\{\mathbf{C}^e\} - [\mathbf{M}_i^e]\{\mathbf{R}_{o_i}^e\} \tag{3.36a}$$

or:

$$[\mathbf{x}^e] = [\mathbf{M}_i^e][\kappa_i^e] \ ; \ [\mathbf{y}^e] = 0 \ ; \ [\mathbf{z}^e] = 0 \ ; \ \ldots \ ; \ \{\mathbf{g}^e\} = [\mathbf{M}_i^e]\{\mathbf{R}_{o_i}^e\} \tag{3.36b}$$

and, again, one must keep in mind that the rate coefficient matrix and constant rate component will, in general, be temperature and pressure dependent:

$$[\kappa_i^e] = [\kappa_i^e(T,P)] \tag{3.36c}$$

$$\{R_{o_i}^e\} = \{R_{o_i}^e(T,P)\} \tag{3.36d}$$

It will be convenient to introduce a new variable for the linear first order *element kinetics transport matrix*, as:

$$[k^e] \equiv [M_i^e][\kappa_i^e] \quad \text{: the \textit{element kinetics transport matrix}} \tag{3.37}$$

and a corresponding variable for the *system kinetics transport matrix*, that is assembled from the element kinetics transport matrices in the usual manner, as:

$$[K] \equiv \underset{\text{kinetics elements}}{\mathrm{A}} [k^e] \quad \text{: the \textit{system kinetics transport matrix}} \tag{3.38}$$

so that the system transport matrix, [W], (equation (2.12b)) may be thought to equal the sum of the familiar system flow matrix, [F], and the system kinetics transport matrix as noted in equations (2.13) and repeated here:

$$[W] = [F] + [K] = \underset{\text{flow elements}}{\mathrm{A}} [f^e] + \underset{\text{kinetics elements}}{\mathrm{A}} [k^e] \tag{3.39}$$

The program CONTAM87, presented in the second part of this report implements the interactive contaminant dispersal theory, presented above, providing the linear first order kinetics element in its library of elements and ordering system concentration variables as discussed above (equation (3.3b)) to enable the proper assembly of flow elements (i.e., by equation (3.3d)) for multi-contaminant dispersal analysis. Examples of the application of these techniques are presented in Section 9.

4. One Dimensional Convection-Diffusion Flow

The flow element presented earlier [Axley 1987] provides the simplest modeling of species mass flow from one zone to another. This simple element is based on the implicit assumption that the volume and the length of the flow passage is negligible and, therefore does not account for any dynamic dispersal phenomena occurring within the flow passage.

For problems where zonal dynamics is of primary interest and it is suspected that flow passage dynamics need not be considered (i.e., for systems with zonal volumetric masses much greater than flow passage volumetric masses and for which flow through these passages is practically instantaneous) the simple flow element should suffice. For those problems where some interest is focused on the detail of flow passage dynamics, or where it is believed that dynamic phenomena in the flow passages can not be ignored, an alternative flow element is required. In this section a convection-diffusion flow element will be developed that will answer this need.

4.1 Convection-Diffusion Equation

Consider the flow through a flow passage (e.g., a section of duct work) that connects zone i to zone j ;

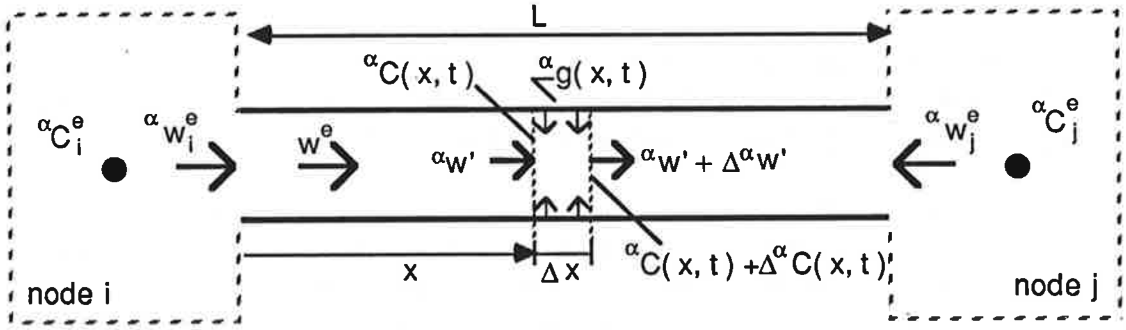


Figure 4-1 One Dimensional Flow Passage

Isolating a segment Δx of the flow passage and demanding the conservation of species mass flowing through this segment we may write;

$$w^e({}^{\alpha}C - ({}^{\alpha}C + \Delta\,{}^{\alpha}C)) + ({}^{\alpha}w' - ({}^{\alpha}w' + \Delta\,{}^{\alpha}w')) + {}^{\alpha}g\Delta x = \rho A \Delta x \frac{\partial\,{}^{\alpha}C}{\partial t} \qquad (4.1)$$

or, in the limit as $\Delta x \rightarrow 0$;

$$-w^{e}\frac{\partial\,{}^{\alpha}C}{\partial x} - \frac{\partial\,{}^{\alpha}w'}{\partial x} + {}^{\alpha}g = \rho A\frac{\partial\,{}^{\alpha}C}{\partial t} \qquad (4.2)$$

where;

w^{e}	is the total fluid (air) mass flow rate through the flow passage
${}^{\alpha}C$	is the α species mass concentration
${}^{\alpha}w'$	is the α species mass flow rate relative to total fluid mass flow rate
ρ	is the density of the fluid (air)
A	is the cross-sectional area of the flow passage
${}^{\alpha}g$	is the α species mass generation rate per unit length of passage
x, t	are distance and time, respectively

The first term above accounts for species mass flow due to *convection* and the second term accounts for species mass flow due to species *diffusion* that is superimposed on the bulk flow.

The generation term, ${}^{\alpha}g$, may be replaced by an appropriate generation (kinetics) rate expression. For example, if the generation involves the single species α one could replace the generation term with a *linear* generation rate expression of the form;

$${}^{\alpha}g = {}^{\alpha}g_{o} + \rho A\, {}^{\alpha\alpha}\kappa\, {}^{\alpha}C \qquad (4.3)$$

where ${}^{\alpha}g_{o}$ is a constant rate component and ${}^{\alpha\alpha}\kappa$ is a generation rate constant. This form of generation rate expression would be appropriate, for example, for formaldehyde emission from the flow passage walls [Mathews et. al. 1984, Grot et. al. 1985].

The diffusion of species mass relative to the (bulk) total mass fluid flow, ${}^{\alpha}w'$, will, in general, depend upon the details of the fluid velocity profile, and its turbulence characteristics (e.g., the *eddy diffusivity*), and molecular diffusion along and perpendicular to the flow passage length. In many practical situations, however, this diffusion component may be modeled using an expression based upon Fick's law of diffusion which may be written as;

$${}^{\alpha}w' = -\rho A\, {}^{\alpha}D\frac{\partial\,{}^{\alpha}C}{\partial x} \qquad \text{: By Analogy to Fick's Law} \qquad (4.4)$$

where ${}^{\alpha}D$ is the *axial dispersion coefficient* of α in the flow fluid (air).

Substituting equation (4.4), equation (4.2) may be rewritten as;

$$\rho A\, {}^{\alpha}D \frac{\partial^2\, {}^{\alpha}C}{\partial x^2} + {}^{\alpha}g = \rho A \frac{\partial\, {}^{\alpha}C}{\partial t} + w^e \frac{\partial\, {}^{\alpha}C}{\partial x} \tag{4.5a}$$

an equation that is commonly called the one dimensional *convection-diffusion equation* that also appears in thermal convection-diffusion problems. The convection-diffusion equation is often expresses in dimensionless form as;

$$\frac{1}{Pe} \frac{\partial^2\, {}^{\alpha}C}{\partial \chi^2} + {}^{\alpha}\gamma = \frac{\partial\, {}^{\alpha}C}{\partial \tau} + \frac{\partial\, {}^{\alpha}C}{\partial \chi} \tag{4.5b}$$

where;

$$Pe \equiv \frac{w^e L}{\rho A\, {}^{\alpha}D} = \frac{\bar{u} L}{{}^{\alpha}D}$$ the dimensionless *Peclet Number* (4.5c)

L is the (characteristic) length of the flow passage
χ is the dimensionless length $\equiv x/L$
τ is the dimensionless time $\equiv t/\bar{t}$
${}^{\alpha}\gamma$ is the dimensionless generation rate $\equiv {}^{\alpha}gL/w^e$
$\bar{t}$ is the nominal transit time $\equiv L/\bar{u}$
$\bar{u}$ is the bulk fluid velocity $= w^e/\rho A$

The Peclet number alone, then, characterizes the convection-diffusion process in a flow passage not involving a kinetic rate expression. It provides a measure of the importance of convection mass transport relative to diffusion mass transport.

The convection-diffusion equation presented above, equation (4.5), is referred to as the *axial dispersion model* or *axial-dispersed plug-flow model* in the chemical engineering literature where it has played an important role in the simulation of flow systems found in the chemical process industries since 1908 when Langmuir introduced it. As one might suspect, the utility of this equation depends critically upon the determination of the dispersion coefficient to be used for a given set of flow circumstances. A complete discussion this problem is well beyond the scope of this report and the reader is, therefore, directed to the excellent general discussion of this approach by Nauman and Buffham [1983] and the more practically useful, reference work by Wen and Fan [1975]. Suffice it to say that for turbulent, isothermal flow conditions in relatively long flow passages, the dispersal coefficient is reasonably well correlated to the characteristic Reynolds number, Re, of the flow and is practically independent of species molecular diffusivity as indicated by the Taylor expression (reported by Wen and Fan [1975 p. 47]):

$$^{\alpha}D \approx {}^{\beta}D \approx \ldots \equiv D \approx 2\bar{u}R\left(\frac{3.0 \times 10^{7}}{Re^{2.1}} + \frac{1.35}{Re^{0.125}}\right) \quad ; \; Re > 2000 \tag{4.6}$$

where;

R is the flow passage radius

Re $\equiv 2\rho\bar{u}R/\mu$

μ is the flow fluid's viscosity

Under turbulent flow conditions, the fluid velocity profile is relatively flat and diffusion is dominated by mass transport by flow eddies rather than by molecular diffusion, thus the dispersion coefficient becomes primarily dependent upon the turbulence characteristics, as measured by the Reynolds number, and is, therefore, often identified as the *eddy diffusivity.*

Under laminar flow conditions, on the other hand, the velocity profile tends to be parabolic, turbulence subsides and as a result both radial and axial molecular diffusion come to play an important role and the diffusion becomes two dimensional in nature. Nevertheless, if the fluid can be assumed to be homogenous, so that the radial and axial molecular diffusivities may be assumed to be identical, then a solution of the complete two dimensional convection diffusion problem reveals that the asymptotic behavior is equivalent to that described by the one dimensional convection diffusion equation using the Taylor-Aris dispersal coefficient [Nauman & Buffham 1983 pp.112-113]:

$$^{\alpha}D \approx {}^{\alpha}\mathcal{D} + \frac{\bar{u}^{2}R^{2}}{48\,{}^{\alpha}\mathcal{D}} \quad ; \; Re < 2000 \quad ; \; \frac{L}{R} < 0.16\frac{\bar{u}R}{{}^{\alpha}\mathcal{D}}$$

(4.7)

where;

$^{\alpha}\mathcal{D}$ is the molecular diffusivity of species α in the flow fluid (air)

To put this discussion of the dispersal coefficient into perspective consider the section of HVAC ductwork illustrated below.

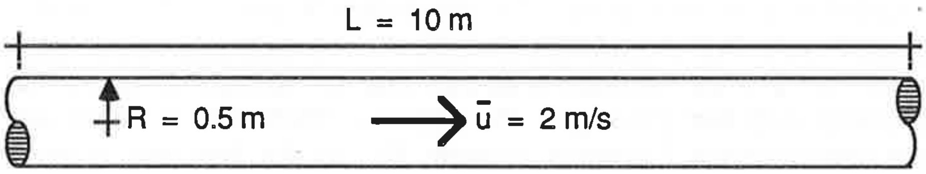


Figure 4-2 Representative HVAC Duct

In practice, building air ducts are normally designed for bulk air flow velocities greater than or equal to 2 m/s [ASHRAE 1985]. For this minimum operational flow rate, then, the Reynolds number for the flow in this duct would be ($\rho_{air} \approx 1.2$ kg/m^3 & $\mu_{air} \approx 1.8 \times 10^{-5}$ kg/m-s at 20°C):

$$Re = \frac{2\rho\bar{u}R}{\mu} = \frac{2\,(1.2\ \mathrm{kg/s})(2.0\ \mathrm{m/s})(0.5\ \mathrm{m})}{(1.8 \times 10^{-5}\ \mathrm{kg/m\text{-}s})} = 1.33 \times 10^{5}$$

and, by the Taylor correlation, equation (4.6), the dispersal coefficient would be:

$$D \approx 2\bar{u}R\left(\frac{3.0 \times 10^{7}}{Re^{2.1}} + \frac{1.35}{Re^{0.125}}\right)$$

$$\approx 2(2.0\ \mathrm{m/s})(0.5\ \mathrm{m})\left(\frac{3.0 \times 10^{7}}{(1.33 \times 10^{5})^{2.1}} + \frac{1.35}{(1.33 \times 10^{5})^{0.125}}\right) = 6.2 \times 10^{-1}\ \mathrm{m^2/s}$$

Typical diffusivities of gas pairs are on the order of 1 to 2 x 10^{-5} m^2/s. Thus, even under these relatively low flow conditions the dispersal coefficient (eddy diffusivity, in this case) is seen to be over four orders of magnitude greater than molecular diffusion. Continuing, the Peclet number, for this case would be:

$$Pe = \frac{\bar{u}L}{D} = \frac{(2.0\ \mathrm{m/s})(10\ \mathrm{m})}{6.2 \times 10^{-1}\ \mathrm{m^2/s}} = 32.3$$

An examination of the turbulent correlation expression reveals that the dispersal coefficient for turbulent conditions is dependent upon the average flow velocity and the flow passage radius. This dependency is plotted below for a range of flow velocities and radii:

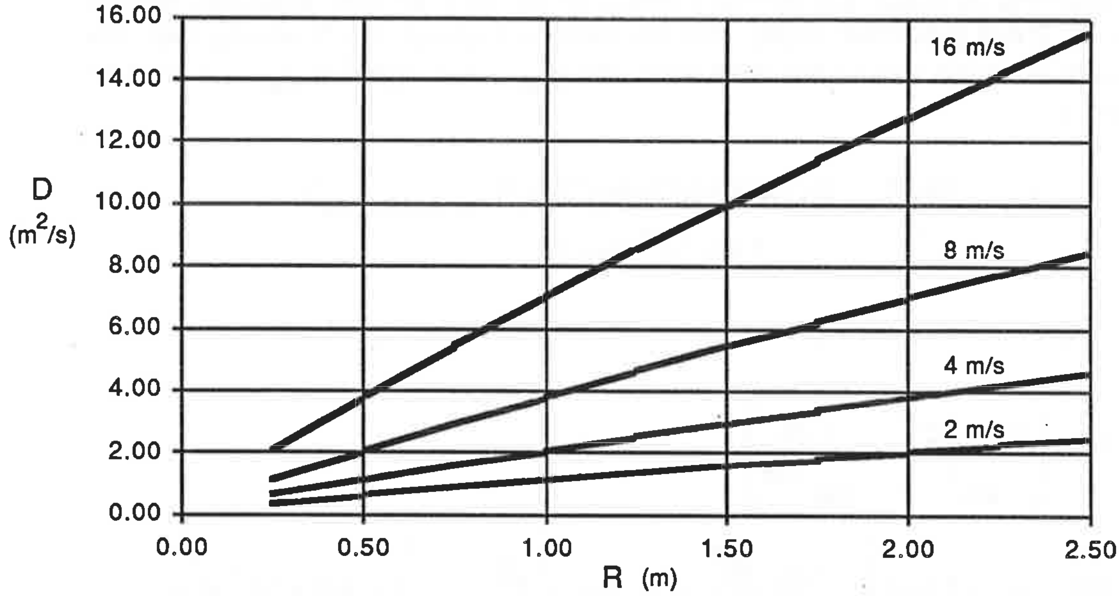


Figure 4-3 Dispersal Coefficient for Turbulent Flow in Ducts (20°C, 1 atm)

4.2 Convection-Diffusion Element Equations

Finite element solutions of convective-diffusion equations of the form of equation (4.5) have received considerable attention in recent years. Following the one-dimensional example discussed by Huebner and Thornton [1982] element equations for a two-node flow element may be developed from equation (4.5) using linear shape functions (i.e., assuming species concentrations vary in a piece-wise linear manner along the flow passage) and applying either the (conventional) Galerkin method or the (upwind) Petrov-Galerkin method in the formulation of these element equations. The resulting element equations are:

$$\{w^e\} = [[\,{}^{\alpha}_{c}f^e] + [\,{}^{\alpha}_{d}f^e]]\{C^e\} + [m^e]\frac{d\{C^e\}}{dt} - \{g^e\} \qquad (4.8a)$$

where;

$$\{w^e\} = \{\,{}^{\alpha}w_i^e,\ {}^{\alpha}w_j^e\}^T$$

$$\{C^e\} = \{\,{}^{\alpha}C_i^e,\ {}^{\alpha}C_j^e\}^T$$

$$[\,{}^{\alpha}_{c}f^e] = \frac{w^e}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} + \frac{\phi\, w^e}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad (4.8b)$$

the *convection component of the element flow transport matrix*

ϕ the so-called *upwind parameter*, $0 \leq \phi \leq 1$

$$[{}^{\alpha}_{d}f^{e}] = \frac{\rho A\, {}^{\alpha}D}{L^{e}} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{4.8c}$$

the *diffusion component of the element flow transport matrix*

L^e the length of the element (i.e., a portion of the length of the flow path)

$$[m^{e}] = \frac{\rho A L^{e}}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} + \frac{\phi\, \rho A L^{e}}{4} \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \tag{4.8d}$$

the *element volume mass matrix*

$$[g^{e}] = \frac{{}^{\alpha}g\, L^{e}}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} + \frac{\phi\, {}^{\alpha}g\, L^{e}}{2} \begin{Bmatrix} -1 \\ 1 \end{Bmatrix} \tag{4.8e}$$

the *internal generation rate vector*

for total <u>fluid mass flow rate</u>, w^e, through the flow passage <u>from node i to node j</u> as indicated in Figure 4-1.

The Upwind Parameter If ϕ is set equal to 0, the element equations become identical to those that would be obtained using the (conventional) Galerkin approach of element formulation. Unfortunately, these conventional element equations lead to solution approximations that exhibit spurious spacial variations when convective transport is large relative to transport by diffusion. The upwind parameter, ϕ, has been introduced, using the Petrov-Galerkin approach, to control this form of numerical instability, but at the cost of introducing artificial diffusion (vis a vis the second term on the right of equation (4.8b)) that introduces inaccuracies.

Generation Kinetics If the generation term of equation (4.5) is replaced by the generation (kinetics) rate expression of equation (4.3) the element equations will have the slightly modified form of:

$$\{w^{e}\} = [[{}^{\alpha}_{c}f^{e}] + [{}^{\alpha}_{d}f^{e}] + [{}^{\alpha}_{k}f^{e}]]\{C^{e}\} + [m^{e}]\frac{d\{C^{e}\}}{dt} - \{g^{e}_{o}\} \tag{4.9a}$$

where;

$$[{}^{\alpha}_{k}f^{e}] = \frac{\rho A L^{e}\, {}^{\alpha\alpha}\kappa}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} + \frac{\phi\, \rho A L^{e}\, {}^{\alpha\alpha}\kappa}{4} \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \tag{4.9b}$$

the *kinetics component of the element flow transport matrix*

$$[g_o^e] = \frac{{}^{\alpha}g_o L}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix} + \frac{\phi\,{}^{\alpha}g_o L}{2}\begin{Bmatrix} -1 \\ 1 \end{Bmatrix} \tag{4.9c}$$

the *constant component species generation rate vector*

Lumped Element Mass Matrix The convection diffusion element equations defined by either equations (4.8) or (4.9) may be assembled, in the usual manner, along with the simple flow element equations (equations (2.4) or (2.5)) and kinetics element equations (equations (3.36)) to form the system equations. The convection-diffusion element introduces, however, nondiagonal contributions to the system mass matrix, [M], that adds some complexity to the assembly and solution algorithms used in the computational implementation of the contaminant dispersal theory. To avoid this complexity one may replace the so-called *consistent* element volume mass matrix, equation (4.6d), with a diagonal *lumped mass* approximation to it, given by:

$$[m^e] \approx \frac{\rho A L^e}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{the } \textit{element lumped mass matrix} \tag{4.10}$$

This approximation may be expected, however, to introduce some additional error. The program CONTAM87 provides convection-diffusion elements having this lumped mass approximation.

4.3 Use of The Convection-Diffusion Flow Element

In this application, we are using the Finite Element method to approximate the spatial variation of contaminant concentration along a flow passage in a piece-wise linear manner where each linear segment of the approximation will correspond to an individual convection-diffusion flow element. Clearly, if the form of the actual concentration variation along the flow path is linear we could model the flow passage with a single element. In general, however, we will not have a priori knowledge about the form of the concentration variation and, therefore, should employ a series of flow elements to obtain a piece-wise approximation to the actual concentration variation.

Without some experience, the analyst may not know how many convection-diffusion elements to use in a given situation. In such a situation, however, a first analysis may be completed with a trial subdivision of the flow path then the analysis may be repeated with a finer subdivision. The finer subdivision may be expected to provide a better approximation to the solution (providing numerically stability has been achieved) and, therefore, may be used to access the accuracy of the solution. This process of subdivision could, then, be repeated until the solution converges to within an acceptable accuracy.

Steady State Analysis When considering steady-state flow without internal generation, Huebner and Thornton show that instability may be avoided if an upwind parameter is selected satisfying the conditions;

$$\phi \geq 1 - \frac{2}{P_e^e} \; ; \; P_e^e > 2$$

$$\phi = 0 \; ; \; P_e^e \leq 2 \tag{4.11}$$

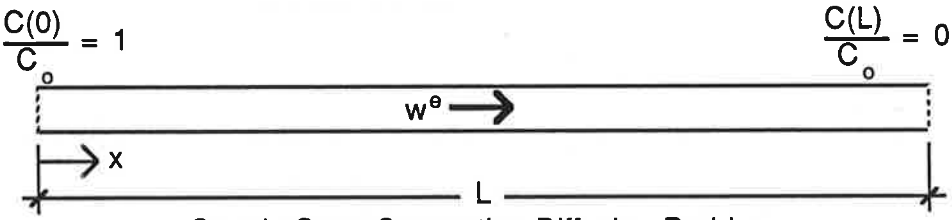
where;

$$P_e^e \equiv \frac{w^e L^e}{\rho A \, {}^{\alpha}D} = \frac{\bar{u} L^e}{{}^{\alpha}D} \tag{4.12}$$

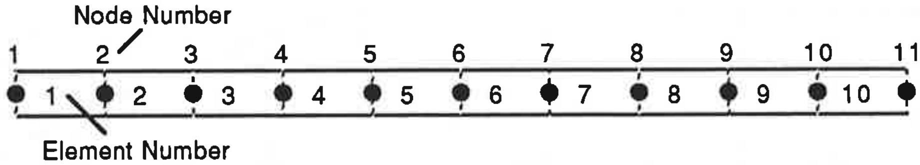
is the *element Peclet number*

(Note: $P_e^e = (Pe/n)$ for a flow passage idealized by an assembly of n equal-length convection-diffusion elements.)

To fix these ideas, consider the problem presented by Huebner and Thornton [1982]; the dispersal of a contaminant along a straight flow passage, under steady flow conditions, without generation, and with inlet contaminant concentration maintained at C_o and outlet concentration maintained at zero, as diagrammed in Figure 4-4.



Steady State Convection-Diffusion Problem



Ten Convection-Diffusion Element Idealization

Figure 4-4 A Steady State Convection-Diffusion Problem and Corresponding Finite Element Idealization

For this problem the convection-diffusion equation simplifies to:

$$\frac{1}{Pe}\frac{d^2C}{dx^2} - \frac{dC}{dx} = 0 \tag{4.13a}$$

which may be solved for the boundary conditions:

$$C(x=0) = C_o$$
$$C(x=L) = 0 \tag{4.13b}$$

to obtain an exact solution:

$$\frac{C(x/L)}{C_o} = \frac{e^{\frac{Pe}{L}(x/L)} - e^{\frac{Pe}{L}}}{1 - e^{\frac{Pe}{L}}} \; ; \; 0 \leq x/L \leq 1 \tag{4.14}$$

that will be compared to approximate solutions obtained using convection-diffusion elements.

For the approximate solution we shall consider an idealization consisting of a series of ten convection-diffusion flow elements, as illustrated in Figure 4-4, and solutions generated for two Peclet numbers, Pe = 0.2 and Pe = 20, and two upwind factors $\phi = 0.0$ and $\phi = 1.0$. The exact with the approximate solutions are compared below in Figure 4-5.

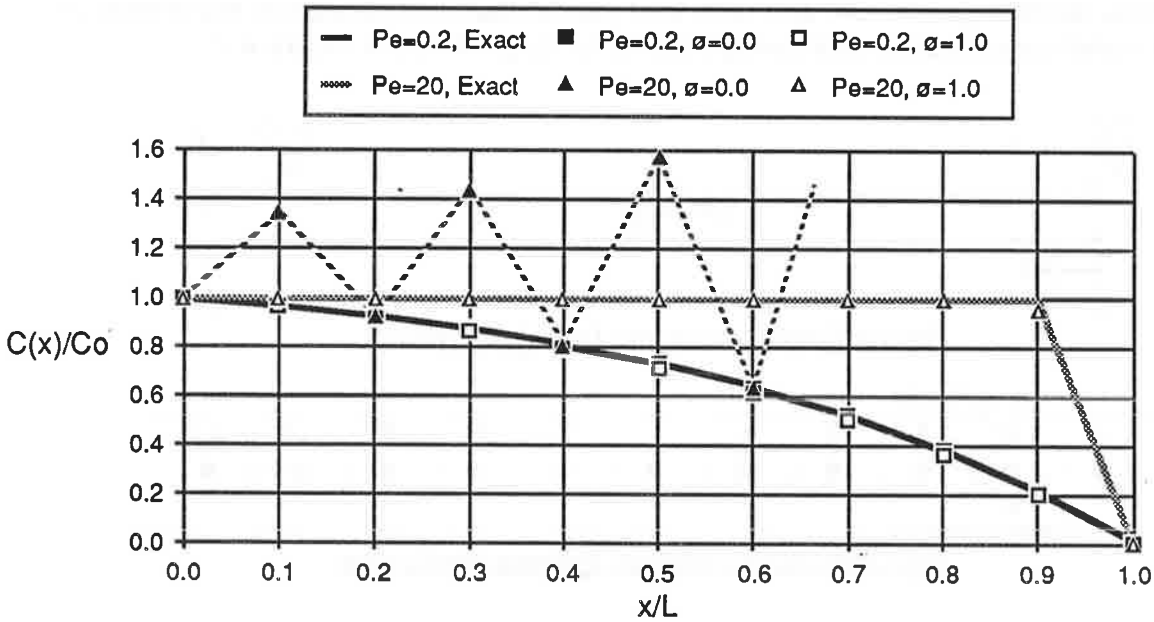


Figure 4-5 Comparison of Exact and Finite Element Solutions for a Steady Convection-Diffusion Problem

The results clearly demonstrate the numerical instability that may result when upwinding is not used for high element Peclet numbers. For convection-dominated flow, which should be expected to be typical in building HVAC ductwork under operating conditions, the analyst may, then, choose to either use a fine subdivision of a given duct or employ upwinding to maintain numerical stability, keeping in mind that the upwinding will introduce artificial diffusion that may add error. (A close examination of the results above reveals that full upwinding underestimated the concentration variation slightly.)

Dynamic Analysis The convection-diffusion flow element may also be employed for dynamic analysis, but the analyst must take special care to assure an accurate solution has been obtained. In dynamic analysis, accuracy is affected not only by element size (i.e., the subdivision of the flow path), and the degree of upwinding chosen, but also by the integration time step selected to complete the dynamic solution. Furthermore, the use of the lumped mass approximation, while avoiding the complexity demanded by nondiagonal mass contributions, tends to introduce spurious anomolies in the computed solution in some cases [Huebner & Thornton 1982].

Partly because of the challenge of these difficulties and partly because of the importance of the convection-diffusion equation in the area of fluid mechanics, finite element solutions of the convection-diffusion equation have become the focus of considerable research in recent years. Strategies have been put forward to improve the accuracy of the finite element approximation presented above that are, regrettably, beyond the scope of this presentation and the reader is, therefore, advised to review the current and emerging literature. The papers by Hughes and Brooks [1982], Tezduyar and Ganjoo [1986], and Yu and Heinrich [1986] are particularly useful in this regard.

In spite of the numerical pitfalls that await the use of the convection-diffusion flow element presented above we shall proceed and employ these elements (with the lumped mass approximation) to compute the transport of a contaminant pulse through a length of ductwork. The conditions of this problem are diagrammed in Figure 4-6: fluid flows through a duct of length L and radius R at a mass flow rate w^e; a contaminant is injected into the inlet stream at a rate G(t) for a short time interval introducing a pulse of contaminant of mass I into the inlet stream; the pulse is convected and dispersed as it moves along the duct. We seek to determine the concentration time history of the contaminant as it emerges from the outlet of the duct.

The exact solution to this problem is available for an impulse (i.e., a pulse defined by the dirac delta function), for "closed" inlet and outlet conditions, but it is expressed as an infinite sum that is practically difficult to use [Wen & Fan 1975 pp. 133-137]. For Pe=0 the duct becomes a well-mixed system, the initial concentration throughout the duct becomes, simply, (I/ρAL), and the outlet concentration decays exponentially:

$$\frac{C(L,t)}{(I/\rho AL)} = e^{-t/\bar{t}} \quad ; Pe = 0 \tag{4.15}$$

For relatively large Peclet numbers the outlet concentration is well approximated by the the following expression reported by Nauman and Buffham [1983 pp. 101-103]:

$$\frac{C(L,t)}{(I/\rho AL)} = \sqrt{\frac{Pe}{4\pi(t/\bar{t})^3}}\; e^{\left(\frac{-Pe(1-t/\bar{t})^2}{4t/\bar{t}}\right)} \quad ; Pe > 16 \qquad (4.16)$$

and for very large Peclet numbers the outlet concentration approaches a Gaussian distribution [Wen & Fan 1975 p. 133]:

$$\frac{C(L,t)}{(I/\rho AL)} = \sqrt{\frac{Pe}{4\pi}}\; e^{\left(\frac{-Pe(1-t/\bar{t})^2}{4}\right)} \quad ; Pe >> 16 \qquad (4.17)$$

Approximate solutions to this problem were computed using a 10-element subdivision, as shown in Figure 4-6, and a twenty-element subdivision. The "closed" boundary condition was modeled using the simple flow element as this element models (instantaneous) plug flow conditions as required. The impulse was approximated by a pulse of finite but small duration. In all studies the upwind parameter, ϕ, was chosen to satisfy the lower bound (i.e., equality) of the stability requirement of equation (4.11). The results are compared below, Figure 4-7, to the solutions discussed above, equations (4.15) to (4.17).

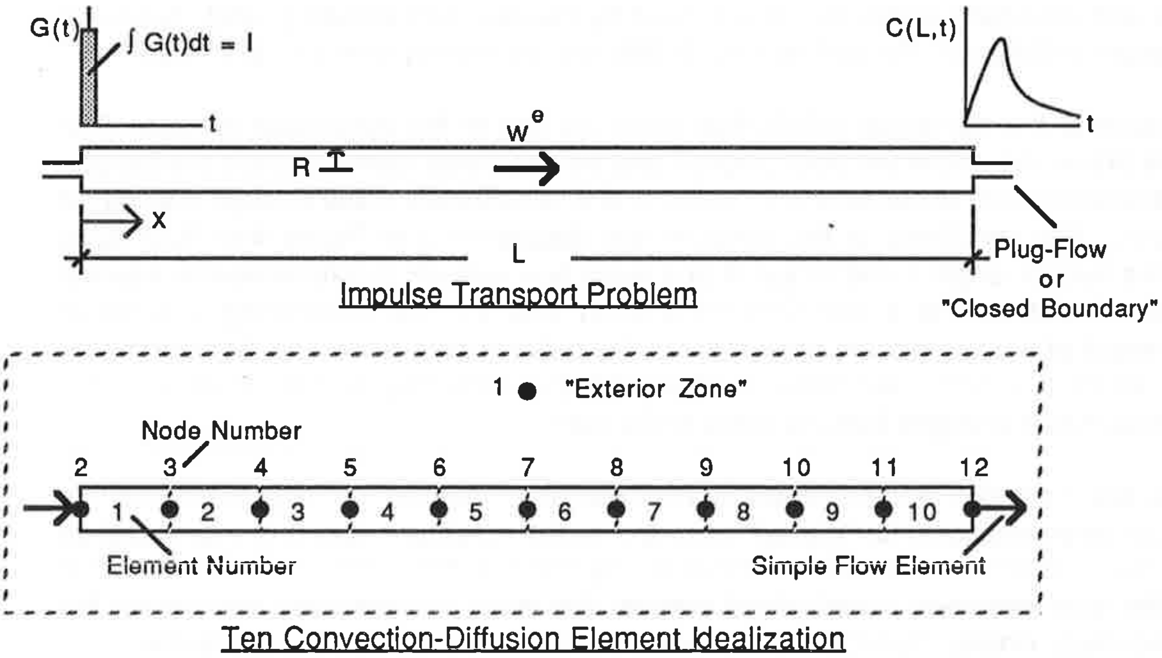


Figure 4-6 The Transport of a Pulse in a Duct and the Corresponding Finite Element Idealization

It is seen that in this case the approximate, finite element solution for the low Peclet number, Pe=1, approaches the exact well-mixed solution, as expected. The approximate solution for the higher Peclet numbers has some difficulty in capturing the amplitude of the exit pulse, although, the timing and the form of the pulse appear to be well-approximated. Some part of this error may be attributed to approximating the impulse of the analytic solutions by a pulse of finite duration in the computed solutions. In these studies the pulse duration was set at 0.001 units of dimensionless time, increasing the pulse duration by a factor of 4 resulted in an additional underestimation of the pulse amplitude at Pe=20 of approximately 5%.

Some part of the error may be attributed to the coarseness of the finite element subdivision. A comparison of the results of the 10-element and 20-element approximations for Pe=10 indicate that a convergent solution was obtained (i.e., further subdivision would not alter the solution), yet when these results are compared to the exact results reported by Wen and Fan [Wen & Fan 1975 Fig. 5-8 p. 136] the amplitude appears to be underestimated by approximately 10%. This same comparison for Pe=20 indicates that a convergent solution was almost but not quite achieved. An additional subdivision would presumably reveal convergence, and the error in amplitude estimation was approximately 20%. It is interesting to note that the element Peclet numbers for these two (nearly) convergent solutions – the 10-element solution at Pe=10 and the 20-element solution at Pe=20 – are both equal to 1.0, a condition that demands no upwinding (i.e., for which ϕ may be set to 0) to maintain numerical stability. Results were also computed for cases violating the stability requirement of equation (4.11) and, as expected, spurious variations in concentration responses – "wiggles" – were observed.

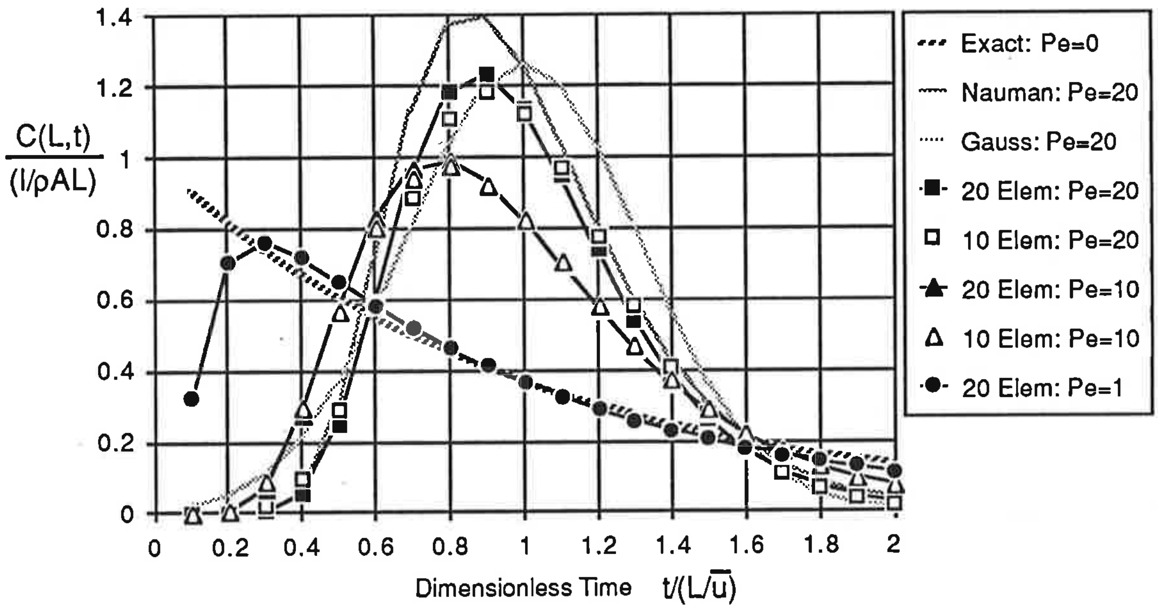


Figure 4-7 Comparison of Analytic Solutions with Finite Element Solutions for the Pulse Transport Problem

It may be useful to relate these nondimensional studies to more conventional units.

The study for Pe=20 corresponds to studying the transport of a pulse through a circular duct of 1 m radius having a length of 10 m with a bulk flow velocity of 2 m/s (the practical minimum operational flow rate in HVAC ducts). For these conditions, by Figure 4-3, the dispersal coefficient may be expected to be about 1.0 m^2/s. The results reported in Figure 4-7 were computed using a pulse duration of 0.005 sec (i.e., 0.001 times the nominal transit time, $\bar{t} = L/\bar{u} = 10 \text{ m} / 2 \text{ m/s} = 5 \text{ s}$). The dynamic solution was computed using a time step of 0.001 second, in part to capture the short-time pulse accurately and partly to achieve a practically convergent solution.

In practical situations the inaccuracies revealed in these studies are likely to be considered very small and, thus, the convection-diffusion flow element should provide a practically useful analytical tool. Nevertheless, to minimize error the analyst is well advised to seek a convergent solution through both *mesh refinement* (i.e., repeated subdivision of the flow path), starting, perhaps, with a subdivision that results in an element Peclet number of 1.0, and *time step refinement*, starting with a time step sufficiently small to capture the dynamic variation of any excitation with reasonable accuracy, being careful to select an upwind factor so that the stability requirement of equation (4.11) is always satisfied. When employing convection-diffusion elements in an idealization of a building airflow system it is very likely that extremely small time steps will be required to obtain a convergent solution.

4.4 Analytical Properties of the Convection-Diffusion Element Equations

The numerical properties of the convection-diffusion flow element have been seen to be dependent on the element Peclet number. To investigate this dependency in greater detail we may rewrite combined convection and diffusion components of the element flow transport matrices, equations (4.8b) and (4.8c), in terms of the element Peclet number, as:

$$[{}^{\alpha}f^{e}] = \left[[{}^{\alpha}_{c}f^{e}] + [{}^{\alpha}_{d}f^{e}] \right] = \frac{w^{e}}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} + \frac{\phi\, w^{e}}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \frac{\rho\, {}^{\alpha}DA}{L^{e}}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$= \frac{w^{e}}{2}\left[\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} + \left(\phi + \frac{2}{P_{e}^{e}} \right) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \right] \tag{4.18}$$

The stability requirement of equation (4.11), which may be rewritten as,:

$$\phi + \frac{2}{P_e^e} \geq 1 \ ; \quad P_e^e > 2$$

$$\phi = 0 \ ; \quad P_e^e \leq 2 \tag{4.19}$$

assures that the flow transport matrix will be an M-matrix, which is to say it is a real square matrix with positive diagonal elements and nonpositive off-diagonal elements such that $[\,[{}^{\alpha}f^{e}] + \xi[\mathbf{I}]\,]$ is strictly diagonally dominant for all scalars $\xi > 0$.

It was shown earlier [Axley 1987] that element flow transport matrices satisfying this condition (coupled with mass matrices that are positive diagonal matrices) lead to system transport matrices that are not only nonsingular, but may be decomposed to [**L**][**U**] form by a variant of Gauss elimination without the need for pivoting in an efficient and numerically stable manner and will have stable homogeneous forms.

4.5 Comparison to Tanks-in-Series Idealizations

In the chemical engineering literature the so-called *tanks-in-series* idealization is frequently employed to model the behavior of one dimensional convection-diffusion transport processes or other processes whose inlet-outlet transformation characteristics appear to match those described by one dimensional convection-diffusion processes. Below, in Figure 4-8, we compare a 5-element/6-node finite element idealization to a corresponding 6-node tanks-in-series idealization where the the fluid mass of volume of the flow path has been subdivided into four "unit" tanks containing one fifth of the total fluid mass each and two "half-unit" tanks containing one tenth of the total fluid mass each.

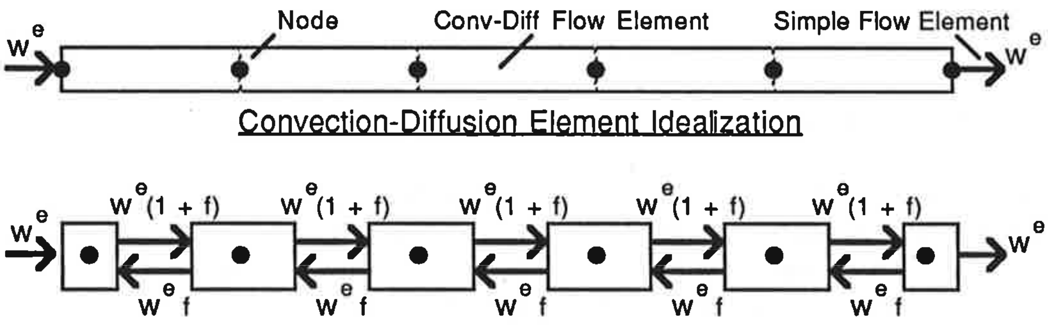
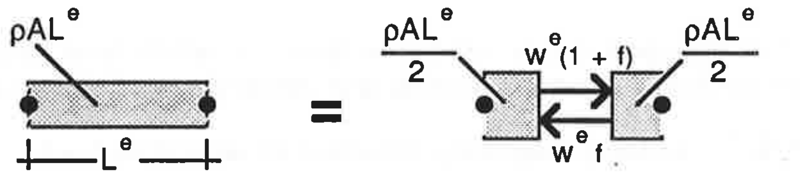


Figure 4-8 Comparison of Tanks-in-Series Idealization with Finite Element Idealization

In the tanks-in-series idealization a portion of the flow, f, assumed to recirculate between adjacent tanks is used to model the nature of turbulent and molecular diffusion.

The subassemblage of this tanks-in-series idealization consisting of half of two adjacent "unit" tanks and the connecting simple flow elements, which we shall refer to as a *tanks-in-series* element, may be compared directly to the convection-diffusion flow element, as indicated in Figure 4-9, below:



Conv-Diff Flow Element "Tanks-In-Series" Element

Figure 4-9 The Equivalence of the Convection-Diffusion Flow Element and a *Tanks-in-Series* Element

Element equations for the tanks-in-series flow element may be assembled directly from the simple flow element equations, equation (2.4a), producing:

$$\begin{Bmatrix} {}^{\alpha}w_i^e \\ {}^{\alpha}w_j^e \end{Bmatrix} = \left[w^e \begin{bmatrix}1&0\\-1&0\end{bmatrix} + f w^e \begin{bmatrix}1&-1\\-1&1\end{bmatrix}\right] \begin{Bmatrix} {}^{\alpha}C_i^e \\ {}^{\alpha}C_j^e\end{Bmatrix} + \frac{\rho A L^e}{2}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{Bmatrix}\dfrac{d\,{}^{\alpha}C_i^e}{dt}\\[2ex] \dfrac{d\,{}^{\alpha}C_j^e}{dt}\end{Bmatrix} \tag{4.20}$$

Comparing these equations with the convection-diffusion element equations, equations (4.8), we see that they become equivalent when:

$$f = \frac{\rho A\,{}^{\alpha}D}{w^e L^e} = \frac{1}{Pe^e} \tag{4.21}$$

and full upwinding, $\phi = 1.0$, is used.

It is interesting to note that the extreme of pure plug flow in the convection-diffusion case corresponds to conditions having a dispersal coefficient equal to zero. A fine subdivision of the flow path into a large number of finite elements would be required to provide a good approximation of the plug flow behavior. In comparison, by equation (4.21), plug flow would correspond to a tanks-in-series element with f = 0.0 – that is to say an element without the recirculating backflow. Nauman and Buffham [1983 pp. 58-59] show that as the number of tanks-in-series without backflow becomes large, the behavior of the assemblage of tanks approaches plug flow. The other extreme of well-mixed conditions may be modeled with an infinite dispersal coefficient in the convection-diffusion case. This corresponds to infinte recirculation in the tank-in-series element and we obtain the behavior of a simple well-mixed zone with either a single convection-diffusion element or a single tanks-in-series element.

The Imperfectly Mixed "Zone Element" The comparison of the convection-diffusion element and the tanks-in-series idealization, supports the conclusion drawn above that, in general, modeling high Peclet number flows will demand a fine subdivision of elements and modeling low Peclet number flows will not. It also points out the fact that the convection-diffusion element may be used to model a zone that is not perfectly mixed. In fact, although we developed the convection-diffusion element to model flow transport situations, it should now become apparent that this element provides one means to model imperfectly mixed zones. If the exit flow response of a flow zone to a supply flow pulse takes the form of the solutions presented above, equations (4.15) to (4.17), then one may employ an assemblage of convection-diffusion flow elements to model the global characteristics of that zone, even though the internal mechanisms governing the imperfect mixing are not apparent.

It may be shown that the variance of the nondimensional response, σ^2, is related directly to the Peclet number of the flow, for a "closed" system, as:

$$\sigma^2 = \frac{2}{Pe} - \frac{2}{Pe^2}(1 - e^{-Pe}) \tag{4.22}$$

For large Peclet flows the form of the (nondimensional) exit response is well approximated by the form of a Gaussian distribution (e.g., see the results of Figure 4-7) which has a variance of:

$$\sigma_G^2 = \frac{2}{Pe} \quad \text{; for the Gaussian approximation equation (4.17)} \tag{4.23}$$

Either of these two expression provides a means to determine an effective Peclet number for a zone, from a rather straightforward statistical reduction of actual pulse response measurements, that may, then, be used for modeling purposes.

Chapter 2 presented the general formulation of a multi-zone contaminant dispersal analysis theory, based on element assembly techniques, and briefly presented the element equations developed earlier: the simple flow element with and without filtration. Chapter 3 outlined a means to organize the multi-zone contaminant dispersal analysis equations for the consideration of multiple contaminants and introduced a kinetics element to account for chemical and physical interactions between contaminants and the materials of the building construction and furnishings. The present chapter introduced a fourth contaminant dispersal element, the one-dimensional convection diffusion element that may be employed to either study the details of dispersal in one-dimensional flow regimes (e.g., as found in HVAC ducts) or to simulate the general behavior of certain imperfectly mixed zones.

The program CONTAM87 implements this theory. Chapters 5 through 8 provide a users manual to this program and Chapter 9 provides some examples of its use.

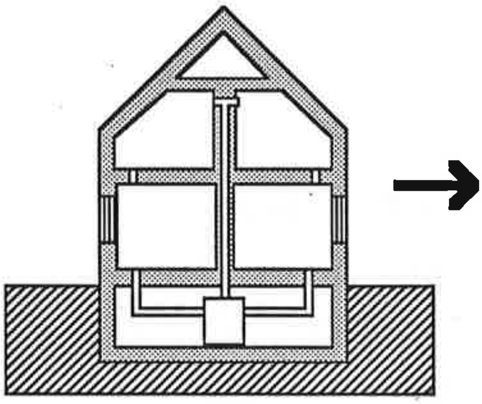
PART II - CONTAM87 USERS MANUAL

5. General Instructions

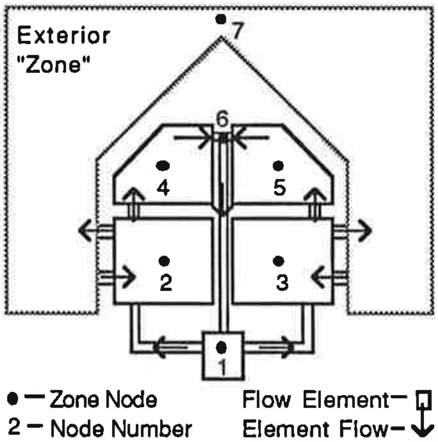
The program CONTAM87 is a command processor; it responds to commands in the order that they are presented and processes data associated with each command. Commands may be presented to the program interactively, using keyboard and monitor, or through the use of command/data input files; that is to say, it offers two modes of operation - interactive and batch modes.

For most practical problems of contaminant dispersal analysis the batch mode of operation will be preferred. For these problems, analysis involves three basic steps;

<u>Step 1</u>:
Idealization of the Building System and Excitation



Actual Building

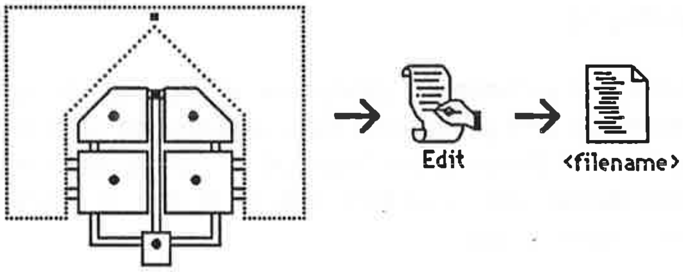


Air-Flow System Idealization

Idealization of the building flow system involves a) discretization of the system as an assemblage of appropriate flow elements connected at system nodes, b) identification of boundary conditions, and c) numbering of system nodes optimally (i.e., to minimize the bandwidth or, equivalently, node number difference of the system equations).

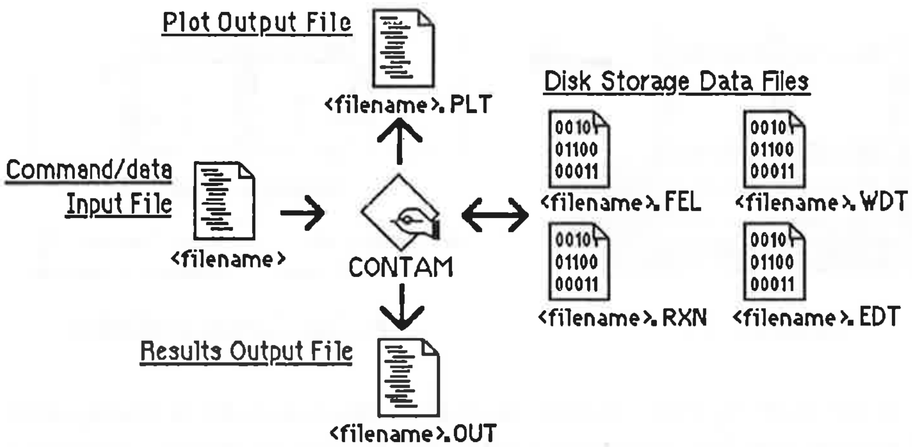
The excitation (i.e., specified contaminant concentrations and generation rates) may be modeled to be steady or defined in terms of arbitrary time histories. For the latter case initial conditions of nodal contaminant concentration will have to also be specified.

Step 2:
Preparation of Command/Data Input File



In the batch mode, the program reads ASCII text files of commands and associated data, collected together in distinct data groups, that define the building flow idealization and excitation. The command/data input file may be prepared with any available ASCII text editing program and given a file name, <filename>, specified by the user. The <filename> must, however, consist of 8 or less alphanumeric characters and can not include an extension (i.e., characters separated from the filename by a period, ".").

Step 3:
Execution of CONTAM87



CONTAM87 is then executed. Initially CONTAM87 will be in the interactive mode. To enter the batch mode the command "SUBMIT F=<filename>" may be used to submit the command/data input file to the program. The program will then proceed to form element and system arrays and compute the solution to the posed problem. CONTAM87 reads the ASCII command/data input file and creates an ASCII (i.e., printable) output file <filename>.OUT. The results of an analysis, <filename>.OUT, may be conveniently reviewed using an ASCII editor and, from the editor, portions or all of the results may be printed out. Key response results are also written to the ASCII file <filename>.PLT in a format that may easily be transferred to spreadsheet and plotting programs (data values within each line are separated by the tab character and lines of data are separated by a carriage return) for plotting or subsequent processing.

Depending upon the commands processed, CONTAM87 will also create a variety of binary files for disk storage needed for subsequent processing. A summary of files read and created includes;

Files Read

<filename> an ASCII input file specified by the user that contains commands and associated data

Files Created

<filename>.OUT a printable ASCII output file that contains analysis results

<filename>.PLT an ASCII output file that contains key analysis results in a form that may be transferred to spreadsheet and/or plotting programs

<filename>.FEL a binary file used for disk storage of flow element data

<filename>.KIN a binary file used for disk storage of kinetics element data

<filename>.WDT a binary file used for disk storage of element flow time history data

<filename>.EDT a binary file used for disk storage of excitation time history data

In the interactive mode <filename> is set to the default value of "CONTAM87" and commands are read from the keyboard. A help command, "HELP" or "H", will produce a screen listing of *intrinsic* commands.

6. Command Conventions

Commands and their associated data (if any) may be single-line or multiple-line command/data groups.

Single-Line Commands Single line command/data groups begin with the command keyword and may have any number of associated data items identified by data identifies of the typical form;

COMMAND A=n1,n2,n3 **B=**n4 **C=**n5,n6 **D=**c1c2c3

where n1,n2,n3,... is numeric data and c1c2c3 is character data. In this example the keyword **COMMAND** is the command keyword and the data identifiers are **A=**, **B=**, **C=**, and **D=**.

Multiple-Line Commands Multiple-line command/data groups are delimited by the command keyword and the keyword **END** and may have any number of data subgroups terminated by the less-than character "<" within. They have the typical form of;

```
COMMAND   A=n1,n2
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
<
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
<
c1c2c3c4c5c6
END
```

Classes of Commands Two general groups of commands are available, the *Intrinsic Commands* and the *CONTAM87 Commands*. The intrinsic commands are used to control the operation of the command processor CONTAM87 and to examine arrays generated by the CONTAM87 commands. The CONTAM87 commands provide contaminant dispersal analysis operations.

Command/data Lines Normally the line length (i.e., the number of character and spaces on a line) is limited to 80. A backslash "\" at the end of information on any line will, however, allow the next line to be interpreted as a continuation of the first line providing an effective line length of 160.

A less-than character "<" indicates the end of information on any line. Information entered to the right of the less-than character is <u>ignored</u> by the program and may, therefore, be used to annotate a command/data input file.

An asterisk "*" at the beginning of any line will cause the line to be echoed as a comment on the console and to the results output file. Lines marked in this way may, then, be used to annotate the results output file. Comment lines may also help indicate the progress of computation when using the batch mode of operation.

Data Identifiers Data identifiers and their associated data may be placed in any order within each line of the command/data group (with the exception that the first line of a command/data group must begin with the command keyword). In some instances data may not be associated with a data identifier, such data must be placed first in a line.

Data Decimal points are not required for real numeric data. Scientific notation of the form nnE+nn or nn.nnE+nn (e.g., 5.79E-13) may be used. Simple arithmetic expressions employing the conventional operators +, –, *, and / may be used. The order of evaluation is sequential from left to right - unlike FORTRAN or other programing languages where other "precedence" rules are used.

If fewer data values are supplied than required, the missing data will assumed to be zero, blank, or set to default values as appropriate.

7. Introductory Example

For purposes of contaminant dispersal analysis the specific command/data groups that need to be included in a command/data input file will depend upon the details of the flow system idealization, the nature of the excitation, and the type of analysis to be computed. A specific introductory example, should however, provide some useful insight into the more general aspects of contaminant dispersal analysis using CONTAM87

Consider the two-story residence with basement shown, in section, below. In this residence interior air is circulated by a forced-air furnace and exterior air infiltrates the house through leaks around the two first-floor windows. The flow system may be idealized using flow elements to model the ductwork, room-to-room, and infiltration flow paths as shown below.

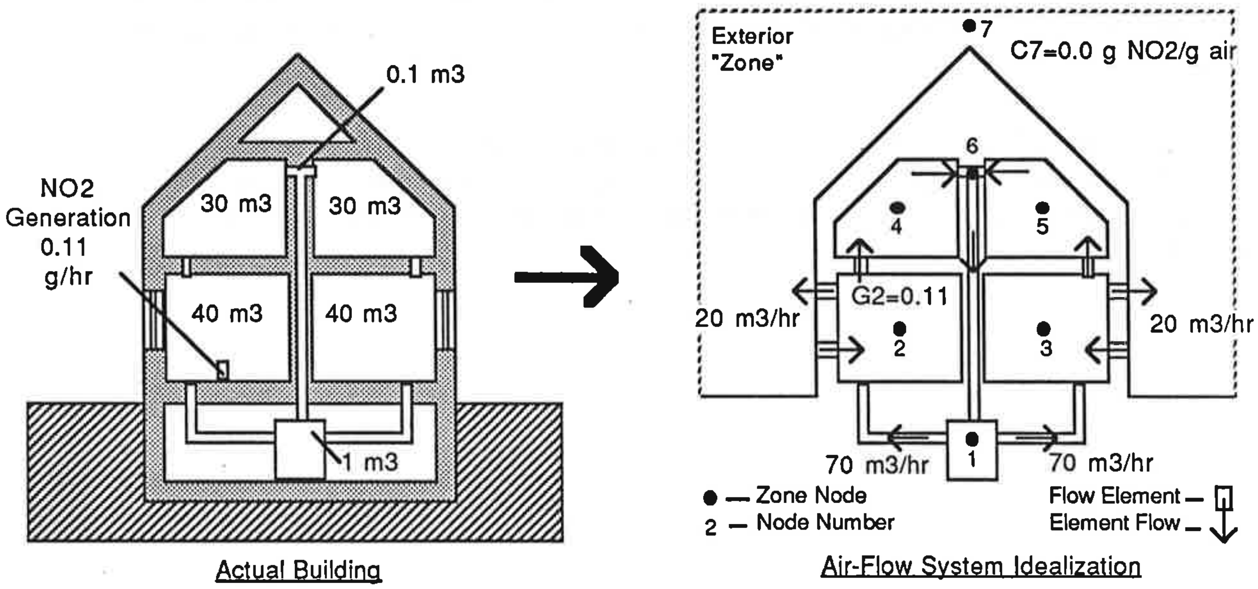


Figure 7-1 Hypothetical Residential Example

For this building idealization we shall consider the hypothetical problem of determining the steady state distribution of NO_2 generated by a kerosene heater placed in room "2", distributed by the furnace flow system operated at constant conditions, and diluted by infiltration at a constant rate. The NO_2 generation rate is assumed to be 0.11 g/hr, exterior NO_2 concentration is assumed to be negligible, and the assumed air volumetric flow rates are indicated on the drawings above. Inasmuch as NO_2 is a reactive gas it will also be assumed that NO_2 is constantly transformed into other products that, here, are of no particular interest, as;

$$NO_2 \rightarrow \text{products}$$

This reaction is often assumed to be governed by rate expressions of the form:

$$\frac{d[NO_2]}{dt} = - {}^{NO_2}\kappa\,[NO_2]$$

thus the matrix of rate coefficients for this case is a 1 x 1 matrix:

$$[\kappa] = [\,{}^{NO_2}\kappa]$$

where ${}^{NO_2}\kappa$, the reaction rate constant for this reaction, will be assumed to have a value of 0.40 hr^{-1}. (These values of NO_2 generation and reaction rate are based on values reported by Traynor [Traynor et. al. 1983] and Nitschke [Nitschke et. al. 1985]. The generation rate is representative of that produced by portable kerosene heaters. The reaction rate constant is thought to be representative of that to be expected indoors, but the kinetics of NO_2 chemical or physical-chemical behavior indoors is not yet well understood.)

The CONTAM87 command/data file to complete this steady state analysis is listed below. Command/data groups needed to complete a time constant analysis and dynamic analysis for this building idealization are presented as examples in the reference section of this manual.

Command/data File for Residential Example

Note: CONTAM87 keywords and identifiers are displayed in boldface below.

Description	Command/data File
Column	1
Comments:	*
Comments	* Six-Zone (7-Node) Example
Comments	* Units: g, m, hr
Comments	* Concentration [=] g-NO2/g-air
Comments	* Generation rate [=] g-NO2/hr
Comments	*
System Definition:	**FLOWSYS**
No.Nodes &.Species, Species IDs	**N**=7 **S**=1 **ID**=NO2
Boundary Conditions	7 **BC**=C < Ext."Zone" Conc.Spec.
	< (Air Dens. 1.2E+3 g/m3)
Nodal Volumetric Mass	1 **V**=1.2E+3*1.0 < Node 1 Vol. Mass
	2,3 **V**=1.2E+3*40.0 < Nodes 2 & 3 Vol. Mass
	4,5 **V**=1.2E+3*30.0 < Nodes 4 & 5 Vol. Mass
	6 **V**=1.2E+3*0.1 < Node 6 Vol. Mass
	7 **V**=1.2E+3*1.E+6 < Node 7 Ext. Vol. Mass
	END
Flow Element Data:	**FLOWELEM**
Element Number & Connectivity	1 **I**=1,2 < Flow Element 1
	2 **I**=1,3 < Flow Element 2
	3 **I**=7,2 < Flow Element 3
	4 **I**=2,7 < Flow Element 4
	5 **I**=7,3 < Flow Element 5
	6 **I**=3,7 < Flow Element 6
	7 **I**=2,4 < Flow Element 7
	8 **I**=3,5 < Flow Element 8
	9 **I**=4,6 < Flow Element 9
	10 **I**=5,6 < Flow Element 10
	11 **I**=6,1 < Flow Element 11
	END
Kinetics Element Data:	**KINELEM**
Rate Coef. (Matrix): Rxn 1	**K**=1 < Rxn 1:NO2 -> products
	0.4
	<
Kinetics Elem. Location & Type	1 **I**=1 **K**=1 < Node 1: Rxn 1
	6 **I**=6 **GEN**=1 **K**=1 < Nodes 2 to 6: Rxn 1
	END
Steady State Solution:	**STEADY** <(Air Dens.1.2E+3 g/m3)
Flow Element Mass Flow Rates	1,2 **W**=70*1.2E+3 < Supply Ducts
	3,6 **W**=20*1.2E+3 < Infiltration
	7,10 **W**=70*1.2E+3 < Return Loop
	11 **W**=140*1.2E+3 < Main Return Duct
	<
Contaminant Excitation	2 **CG**=0.11 < Node 2: NO2 Gen. Rate
	7 **CG**=0.0 < Node 7:Ext. NO2 Conc.
	END
Return to Interactive Mode	**RETURN**

Details are given on the following pages for each CONTAM87 command.

8. Command Reference

CONTAM87 provides two general classes of commands; *Intrinsic Commands* and *CONTAM Commands.*

8.1 Intrinsic Commands

The intrinsic commands are used to control the operation of the command processor CONTAM87 and to examine arrays generated by the CONTAM87 commands.

These intrinsic commands have been developed to provide general command processor operations that, together with the general command conventions outlined earlier, define a standard *user-machine interface* that may be used in the development of other simulation software.

8.1.1 HELP

The command **HELP**, or simply **H**, will produce a list of available intrinsic commands, in abbreviated form.

8.1.2 ECHO

The command **ECHO-ON** acts to cause computed results normally directed to the results output file to be echoed to the screen. The command **ECHO-OFF** turns this feature off. At start-up CONTAM87 is set to **ECHO-ON**. Selective use of **ECHO-ON** and **ECHO-OFF** can act to speed computation as writing results to the screen consumes a significant amount of time.

8.1.3 LIST

The command **LIST**, or simply **L**, will produce a list of all arrays currently in the array database.

8.1.4 PRINT A=<arrayname>

The command **PRINT A=**<arrayname> or simply **P A=**<arrayname> will print array named <arrayname>, a one-to four character name, to the screen. Arrays currently in memory are listed by name using the **LIST** command.

8.1.5 DIAGRAM A=<arrayname>

The command **DIAGRAM A=<arrayname>** or simply **D A=<arrayname>** will print a diagram of array named <arrayname>, a one-to four character name, to the screen indicating position of zero and nonzero terms. (Character arrays can not be diagramed.)

8.1.6 SUBMIT F=<filename>

The command **SUBMIT F=<filename>** or simply **S F=<filename>** will cause the program to switch to batch mode and read all subsequent commands from the batch file <filename>.

8.1.7 PAUSE

The command **PAUSE** will cause the execution of CONTAM87 to pause until a carriage return is entered from the keyboard. Selective use of **PAUSE** in a batch command/data input file will allow the user time to view results of interim calculations. (Note: **PAUSE** is a single line command and, therefore, cannot be placed <u>within</u> other multiline command/data groups.)

8.1.8 RETURN

The command **RETURN** returns the operation of the program from batch mode to interactive mode. **RETURN** or **QUIT** will normally be the last command of batch command/data input files.

8.1.9 QUIT

The command **QUIT** or simply **Q** terminates execution of the program and returns the user to the control of the operating system.

8.2 CONTAM87 Commands

The CONTAM87 Commands implement basic contaminant dispersal analysis operations. These operations are based upon the dimensionally homogeneous theory presented in the first part of this report, thus, the analyst may use any dimensional units that are convenient so long as a consistent set of units are employed. Following the underlying theory one may elect to express all quantities in terms of units of mass and time;

species concentration	[=] mass-species/mass-air
species generation rate	[=] mass-species/time
zone ("volumetric") mass	[=] mass-air
air flow rates	[=] mass-air/time
kinetics rate constants	[=] 1/time

or, if consideration is limited to isothermal cases, one may elect to use volumetric quantities;

species concentration	[=] volume-species/volume-air
species generation rate	[=] volume-species/time
zone volume	[=] volume-air
air flow rates	[=] volume-air/time
kinetics rate constants	[=] 1/time

Using the simple in-line arithmetic expressions allowed for numeric data, one may easily convert from one quantity to another while maintaining a record of the conversion in the input command/data file for future reference. For example, the zone "volumetric" mass (i.e., the mass of the air within the volume of each zone) required by the FLOWSYS command could be expressed in terms of zone dimensions and air density as `V=5.0*10.0*2.5*1.2` for a room that is 5 m × 10 m × 2.5 m containing air with a density of 1.2 kg/m^3.

The following conventions will be used for the command definitions presented in this section;

- ellipses, '. . . ', indicate unlimited repetition of similar data items or data lines within a data subgroup
- square brackets, [...], indicate optional data,
- numeric data is indicated by lower case n, as n1,n2, ... , and
- character data by lower case c, as c1.

8.2.1 FLOWSYS

The number of the flow system nodes and species, boundary conditions, and volumetric masses of system nodes are defined with the following command/data group;

FLOWSYS
N=n1 **S**=n2 [**ID**=c1,c2, ...]
n3,n4,n5 **BC**=c3,c4, ...
. . .
<
n3,n4,n5 **V**=n6
. . .
END

where;	n1	= the number of flow nodes
	n2	= the number of contaminant species
	c1,c2, ...	= species ID's; a four character (or less) identification for each species used for labeling results; species are identified by species-number and, optionally, by species ID given in species-number order; omitted species IDs will be set to species-number
	n3,n4,n5	= first node, last node, node increment of a series of nodes with identical boundary conditions
	c3,c4, ...	= boundary condition codes for each species by species number order; a single character code of **C** for concentration prescribed nodes or **G** , for generation prescribed nodes; (default = **G**),
	n6	= nodal volumetric mass; (default = 0.0)

The direct species mass generation rate or the species concentration, but not both, may be specified for each species at each node to establish the discrete boundary conditions of the analysis problem being posed.

Omitted boundary condition data will be assumed to be generation-prescribed. Typically, nodes associated with the outdoor environment will be assigned specific contaminant concentrations and nodes associated with indoor air zones will be assigned specific species generation rates (a zero generation rate will often be appropriate for the interior species/node combinations).

Volumetric mass data omitted will be assumed to be zero. The present version of CONTAM does not eliminate system variables associated with zero mass nodes. For time constant analysis, and in some instances dynamic analysis, a zero nodal mass value will result in numerical difficulties. From a practical point of view, all nodes of a flow system idealization will have some volume of air associated with them, although some may seem insignificantly small, and, therefore, to avoid numerical difficulties all of these volumes should be modeled with nonzero volumetric mass values.

At the other extreme, some nodes, such as those corresponding to the outdoor environment , may have practically infinite volumes associated with them. The analyst should realize practically accurate analysis results for these "infinite" nodes if their volumes are modeled with volumetric masses several orders of magnitude larger than that of the largest "non-infinite" node.

8.2.2 FLOWELEM

Presently two types of flow elements are available for contaminant dispersal analysis;

- a *simple flow element* that models fluid flow from one node to another ignoring the dynamic effects of diffusion and convection that result in species flow delay along the flow path (i.e., flow of a fluid parcel in simple flow elements is instantaneous) and,

- a *convection-diffusion flow element* that models fluid flow from one node to another accounting for these dynamic effects (presently limited to constant cross-section flow passage idealizations and lumped mass idealizations).

To use these elements effectively and reliably the analyst should be familiar with their underlying theoretical basis and numerical characteristics. This is especially important when using the convection-diffusion element. An inexperienced analyst is well-advised to avoid the use of convection-diffusion elements altogether.

Both simple flow elements and convection-diffusion flow elements may be added to the flow system assemblage with a command/data group having unique formats of data lines for each flow element type of the form;

FLOWELEM
n1 **I**=n2,n3 [**GEN**=n4] [**T**=**SIMP**] [**E**=n5,n6,...]

or

n1 **I**=n2,n3 [**GEN**=n4] **T**=**CNDF** **M**=n7 **L**=n8 [**D**=n9,n10,...] [**F**=n11]

. . .
END

where; n1 = the element number
n2, n3 = the system node numbers to which the element is connected
n4 = generation increment (default = 1)

For Simple Flow Elements: [**T**=**SIMP**]
n5,n6,... = the element filter efficiency for each species being considered, in species-number order, (must be ≥ 0.0; default = 0.0),

For Convection-Diffusion Elements: **T**=**CNDF**
n7 = the fluid mass per unit length of the (equivalent) constant cross-section element (must be ≥ 0.0; default = 0.0),
n8 = the flow passage length (must be > 0.0; default = 0.0),
n9,n10,... = species dispersal coefficient for each species considered, in

species-number order, (must be ≥ 0.0; default = 0.0)

n11 = upwind factor, ϕ; where $0 \leq \phi \leq 1$; (default = the lower bound of the stability criteria , equation (4.11))

For assemblages consisting of only Simple flow elements the command/data group would consist of data lines of the form with the type identifier **T=SIMP**. For assemblages consisting of only Convection-Diffusion flow elements the command/data group would consist of data lines of the form with the type identifier **T=CNDF**. For mixed assemblages the appropriate mix of the two forms of data lines would be used; there are no special restrictions on the use of mixed assemblages.

If the element type identifier is omitted the element will be assumed to be of type **SIMP.**

Normally, the analyst should accept the default upwind factor for the Convection - Diffusion element. This default will ensure that numerical solutions to the posed problem may be determined in an efficient and stable manner. (The option to specify the upwind parameter is provided to allow one to study the numerical characteristics of the upwinding strategy rather than the practical behavior of flow systems.)

Element data must be supplied in numerical order. Omitted data is automatically generated by incrementing the preceding node numbers by the current generation increment. Generated elements will have the properties of the current element. If, for example, an HVAC duct, included as part of a air flow system, was to modeled by a series of, say, ten convection diffusion elements, as illustrated below in Figure 8-1, then one could conveniently use the generation option to "generate" the intermediate elements by specifying only the first and last Convection-Diffusion flow element in the series. The portion of the input command/data file needed to implement this example is listed below.

```
FLOWELEM
...
21 I=12,15   T=CNDF  M=1.2E+03  L=1.0
30 I=39,42   T=CNDF  M=1.2E+03  L=1.0  GEN=3
...
END
```

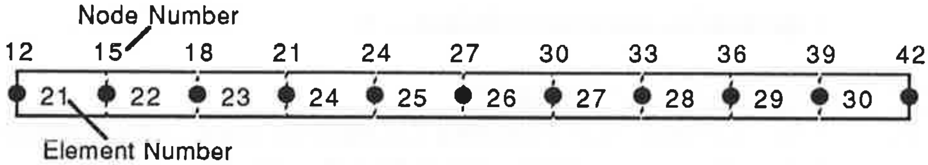


Figure 8-1 Hypothetical Conduction-Diffusion Element Subassembly

The command **FLOWELEM** may be invoked more than once to incrementally add flow elements to the assemblage. Using this feature an analyst may consider a series of successively more complex flow system assemblages and their response to specified excitations.

8.2.3 KINELEM

Interactive species behavior governed by first order kinetics may be accounted for in the model through the assembly of *kinetics elements*. A kinetics element will, typically, model chemical, radio-chemical, or sorption kinetics between a contaminant species and the immediate environment or other species within a well-mixed zone. As such they may be associated only with those system nodes that correspond to well-mixed zones. These elements may be added to the assembly with the following command/data group that first defines pertinent rate coefficient matrices and then assigns specific rate coefficient matrices to specific nodes of the system;

```
KINELEM
K=n1
n2, n3, ...
n4, n5, ...
...
K=n1
n2, n3, ...
n4, n5, ...
...
<
n6  I=n7  K= n8  [GEN=n9]
. . .
END
```

where;		
	n1	= the kinetics ID of the following rate coefficient matrix,
	n2, n3, ...	
	n4, n5, ..	= the rate coefficient matrix for kinetics ID n1
	...	
	n6	= the kinetics element number
	n7	= the <u>well-mixed zone</u> system node number at which the kinetics is to be applied,
	n8	= kinetics ID
	n9	= generation increment (default = 1),

The rate coefficient matrices are entered by rows, in species-number order, and <u>must</u> be defined in terms of all species considered whether or not all of the species are involved in a given rate expression. Therefore, for a system involving N species <u>all</u> rate coefficient matrices will be square matrices with N x N terms. If a particular species is not involved in a given rate expression the terms in the columns and rows corresponding to this species will simply be zero values.

Element data must be supplied in numerical order. Omitted data is automatically generated by incrementing the preceding node number by the current generation increment. Generated elements will have the properties of the current element.

Example

Given a system involving three species, say, A, B, and C involved in the following chemical reactions;

1) a simple reversible reaction

$A \leftrightarrow B$

governed by the rate expression;

$$\frac{d[A]}{dt} = -0.45[A] + 0.45[B] + 0.0[C]$$

$$\frac{d[B]}{dt} = 0.45[B] - 0.45[A] + 0.0[C]$$

$$\frac{d[C]}{dt} = 0.0[A] + 0.0[B] + 0.0[C]$$

or, more concisely by the first order rate coefficient matrix;

$$[\,{}^{1}\kappa] = \begin{bmatrix} 0.45 & -0.45 & 0 \\ -0.45 & 0.45 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and,

2) a single-step reaction

$B \rightarrow$ products (that are of no particular interest)

governed by the rate expression;

$$\frac{d[A]}{dt} = 0.0[A] + 0.0[B] + 0.0[C]$$

$$\frac{d[B]}{dt} = 0.0[A] - 0.35[B] + 0.0[C]$$

$$\frac{d[C]}{dt} = 0.0[A] + 0.0[B] + 0.0[C]$$

or, more concisely by the first order rate coefficient matrix;

$$[{}^{2}\kappa] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.35 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where the first reaction occurs at system nodes 3,5,7,9 while the second reaction only occurs at nodes 5 and 7 (e.g., due to the action of a specific catalyst at these nodes) kinetics elements could be added to the contaminant dispersal system using the following command/data group;

```
KINELEM
K=1                          < Kinetics ID 1:    A <=> B
 0.45   -0.45  0.0
-0.45    0.45  0.0
 0.0     0.0   0.0
K=2                          < Kinetics ID 2:   B => products
 0.0     0.0    0.0
 0.0     0.35   0.0
 0.0     0.0    0.0
<
1    I=3    K=1              < Kin Elem 1:   Node 3:   Kinetics ID 1:
4    I=9    K=1    GEN=2     < Kin Elems 2,3, & 4:  Nodes 5,7,& 9:  Kinetics ID 1:
5    I=5    K=2              < Kin Elem 5:   Node 5:   Kinetics ID 2:
6    I=7    K=2              < Kin Elem 6:   Node 7:   Kinetics ID 2:
END
```

The rate expressions, above, have been written in terms of all contaminant species, including the nonreactive species C, to emphasize the manner in which rate expressions are defined through the use of rate coefficient matrices.

8.2.4 FORM-[W]

In some instances an analyst may wish to examine the details of the mass transport matrix, [W]. The command **FORM-[W]** answers this special (and unusual) need. This command is not required as an interim step to complete any of the analyses options offered by subsequently defined commands.

The system mass transport matrix, [W], assembled from flow element and reaction element matrices may be formed with the following command/data group;

```
FORM-[W]  [F=c1c2c3c4]
n1,n2,n3  W=n4
...
END
```

where; c1c2c3c4 = FULL or BAND; (default = FULL)
n1,n2,n3 = first element number, last element number, element number increment of a series of elements with identical mass flow rates
n4 = element total mass flow rate

The matrix may be formed in its full form or compacted form (i.e., only the nonzero band of the [W] matrix). The system mass transport matrix may be printed or diagrammed using the intrinsic commands **PRINT** and **DIAGRAM**.

8.2.5 STEADY

The response of the system to steady contaminant generation with steady element mass flow may be computed with the following command/data group;

```
STEADY
n1,n2,n3  W=n4
...
<
n5,n6,n7  CG=n8,n9,...
...
END
```

where; n1,n2,n3 = first element number, last element number, element number increment of a series of elements with identical mass flow rates
n4 = element total mass flow rate
n5,n6,n7 = first node, last node, node increment of a series of nodes with identical excitation
n8,n9,... = contaminant concentration or contaminant generation rate for each species considered, as appropriate to the boundary condition of the node/species combination specified with the FLOWSYS command; (default = 0.0)

Net total mass flow rate at each system node will be reported, but computation will not be aborted if net mass flow is nonzero. The analyst must assume the responsibility to check continuity of mass flow from these reported values.

8.2.6 TIMECONS

System time constants, nominal and actual, may be computed with the following command/data group;

```
TIMECONS  [E=n1]
n2,n3,n4  W=n5
...
END
```

where; n1 = optional convergence parameter, epsilon ; (default = machine precision)
n2,n3,n4 = first element number, last element number, element number

increment of a series of elements with identical mass flow rates

n5 = element total mass flow rate

The *nominal* time constants are computed for each node as the quotient of the nodal volumetric mass divided by the total air flow out of a zone. The *actual* time constants are computed using an eigenanalysis routine that is a variant of Jacobi iteration adapted for nonsymmetric matrices [Eberlein et. al. 1971]. It should be noted that the actual time constants are likely to be very different from the nominal time constants for systems having well-coupled zones and the eigenanalysis of the flow system matrices is a time consuming task.

Example

To determine the time constants associated with the building idealization presented earlier, in the introductory example, the following command/data group would have to be added to the command/data file.

```
TIMECONS                 < (Air Density 1.2E+3 g/m3)
1,2    W=70*1.2E+3       <  Supply Ducts
3,6    W=20*1.2E+3       <  Infiltration
7,10   W=70*1.2E+3       < Return Loop
11     W=140*1.2E+3      < Main Return Duct
END
```

8.2.7 Dynamic Analysis

The response of the system, including transients, to general dynamic excitation, may be computed using the command **DYNAMIC**. The dynamic solution procedure used is driven by discrete time histories of excitation and element mass flow rate data that must first be generated with the commands **FLOWDAT** and **EXCITDAT**.

8.2.7.1 FLOWDAT

Discrete time histories of element mass flow rate may be defined, in step-wise manner, from given element mass flow data, as illustrated below;

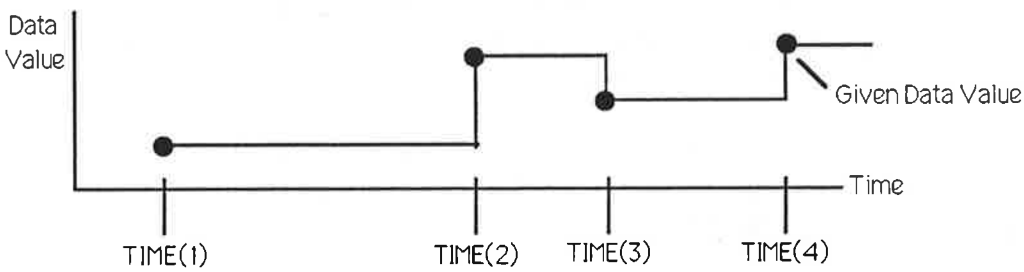


Figure 8-1 Arbitrarily Defined Time History Data

or, alternatively, discrete time histories of element mass flow data, defined in a step-wise manner at equal time-step intervals along piece-wise linear segments, may be generated from given element mass flow data over a time range defined by an initial time, T_i, a final time, T_f, and a generation time increment, ΔT, as illustrated below;

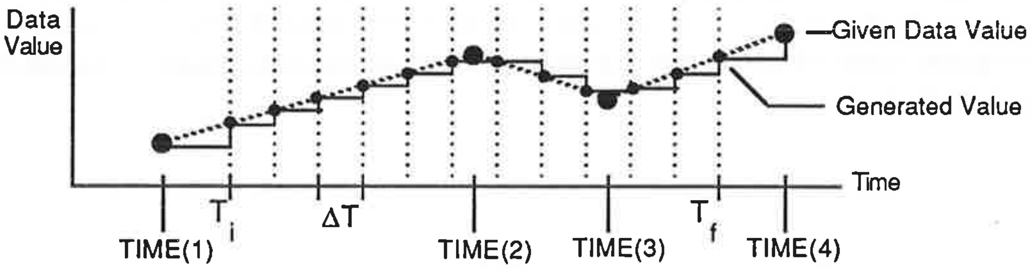


Figure 8-2 Equal-Time-Step-Generated Time History Data

using the following command/data group;

FLOWDAT [T=n1,n2,n3]
TIME=n4
n5,n6,n7 W=n8
. . .
<
TIME=n4 [additional TIME data to define the complete excitation time history]
n5,n6,n7 W=n8
. . .
<
END

where; n1,n2,n3 = initial time, final time, time step increment used for the generation option
n4 = time value for subsequent data subgroups
n5,n6,n7 = first element number, last element number, element number increment of a series of elements with identical mass flow data
n8 = prescribed element mass flow: (default = 0.0)

If data values n1,n2,n3 are specified, step-wise time histories will be generated from the given data, along piece-wise linear segments as illustrated in Fig. 7.2 above, otherwise the given data will be used directly, as illustrated in Fig. 7.1 above.

At least two "TIME" data subgroups must be provided. **FLOWDAT** writes the generated time history to the file <filename>.WDT so that this data may subsequently be accessed by the command **DYNAMIC**.

8.2.7.2 EXCITDAT

Discrete time histories of excitation data may be defined in the two ways discussed above for the **FLOWDAT** command using the following command/data group;

```
EXCITDAT  [T=n1,n2,n3]
TIME=n4
n5,n6,n7  CG=n8,n9,...
...
<
TIME=n4            [additional TIME data to define the complete excitation time history]
n5,n6,n7  CG=n8,n9,...
...
<
END
```

where; n1,n2,n3 = initial time, final time, time step increment used for generation option

- n4 = time value for subsequent data subgroups
- n5,n6,n7 = first node, last node, node number increment of a series of nodes with identical excitation data
- n8,n9,... = prescribed contaminant concentration or prescribed contaminant generation rate (as appropriate to node boundary condition) for each species considered: (default = 0.0)

If data values n1,n2,n3 are specified, step-wise time histories will be generated, from the given data, along piece-wise linear segments as illustrated in Fig. 7.2 above, otherwise the given data will be used directly, as illustrated in Fig. 7.1 above.

At least two "TIME" data subgroups must be provided. **EXCITDAT** writes the generated time history to the file <filename>.EDT so that it may subsequently be accessed by the command **DYNAMIC**.

8.2.7.3 DYNAMIC

The response of the system to excitation defined by the **EXCITDAT** command, using the prescribed element flow data defined by the **FLOWDAT** command, may be computed using the following command/data group;

```
DYNAMIC
T=n1,n2,n3  [A=n4]  [RI=n5]  [PS=n6]
n7,n8,n9  IC=n10,n11,...
...
END
```

where; n1,n2,n3 = initial time, final time, time step increment
n4 = integration parameter, α, where $0 \leq \alpha \leq 1$; (default = 0.75) instability may result for $\alpha < 0.5$,
n5 = response results report interval; (default = 1)
n6 = plot file results scale factor; if not equal to 0.0 an ASCII file, <filename>.PLT, of concentration response results will be created with values scaled by the factor n6
n7,n8,n9 = first node, last node, node increment of a series of nodes with identical data
n10,n11,...= initial nodal concentration for each species in species order; (default = 0.0)

The response is computed using the predictor-corrector method presented earlier [Axley 1987]. With this method the system flow matrix is updated at the discrete times used to define element flow rate time histories and the system excitation is updated at the discrete times used to define excitation time histories, as illustrated below;

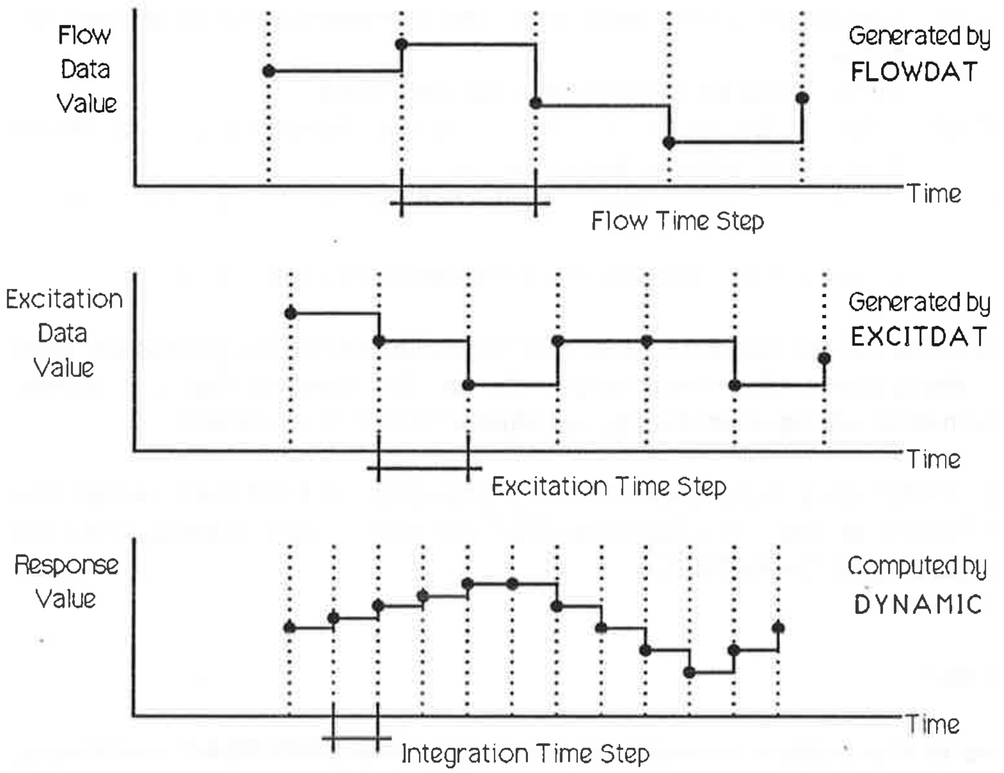


Figure 8-3 Flow and Excitation Driven Dynamic Solution Procedure

The accuracy of the computed response is, therefore, dependent upon the choice of the flow data time step, the excitation data time step, and the integration time step chosen by the analyst. Furthermore, the flow data and excitation data time steps may be nonconstant. The analyst should, therefore, consider investigating the effects of the choice of these time constants to gain a sense of the error they induce.

8.2.7.4 Dynamic Analysis Example

To provide an example of a command/data sequence needed for dynamic analysis we may consider an extension to the introductory example presented earlier; the analysis of the dynamic response of the given building system, under conditions of constant air flows, to a step change in NO_2 generation. Specifically, to consider the case where the kerosene heater is turned on and then turned off 133 minutes later, the following command/data group would have to be added to the command/data file used in the introductory example;

```
FLOWDAT                 <Element flow rates modeled as constant.
TIME=0.0                <(Air Density 1.2E+3 g/m3)
1,2    W=70*1.2E+3      < Supply Ducts
3,6    W=20*1.2E+3      < Infiltration
7,10   W=70*1.2E+3      < Return Loop
11     W=140*1.2E+3     < Main Return Duct
<
TIME=5
1,2    W=70*1.2E+3      < Supply Ducts
3,6    W=20*1.2E+3      < Infiltration
7,10   W=70*1.2E+3      < Return Loop
11     W=140*1.2E+3     < Main Return Duct
END
EXCITDAT                < Nodal Excitation
TIME=0.0                < Kerosene heater turned on at time = 0 mins.
2    CG=0.11            < Node 2: NO2 Generation Rate
7    CG=0.0             < Node 7: Ext. NO2 Conc.
<
TIME=133/60             < Kerosene heater turned off at time = 133 mins.
2    CG=0.0             < Node 2: NO2 Generation Rate
7    CG=0.0             < Node 7: Ext. NO2 Conc.
<
TIME=5                  < Kerosene heater still off at time = 5 hours.
2    CG=0.0             < Node 2: NO2 Generation Rate
7    CG=0.0             < Node 7: Ext. NO2 Conc.
END
DYNAMIC
T=0,4,0.1    PS=1.0E+6  < Time-step; Plot Scale
1,7    IC=0.0           < Initial Concentrations
END
```

8.2.8 RESET

The command **RESET** resets the system in preparation for a new analysis problem (i.e., key internal variables are re-initialized, contaminant dispersal analysis system arrays are deleted from memory, and existing binary files are deleted from disk storage).

9. Example Applications

Examples of the application of CONTAM87 to practical problems of building contaminant dispersal are presented in this section. The reader will also find a discussion of the use of the convection-diffusion flow element for both steady state and dynamic analysis of contaminant dispersal in one-dimensional flow paths presented in section 4.3.

9.1 IBR Test House Study

While working at the National Swedish Institute for Building Research (IBR) Kai Sirén developed a program for multi-zone contaminant dispersal analysis, MULTIC, and applied it to the analysis of the dynamic behavior of the five-room test house maintained by the IBR [Sirén 1986], Figure 9-1. Using data reported by Sirén the building idealization shown below, Figure 9-2, was formulated and the (dynamic) decay response of the idealization to an initial concentration in the bed room, node 2, was computed (for steady flow conditions) and compared to the results reported by Sirén. Air flow rates, zonal volumes, and initial conditions are reported below, Figure 9-2.

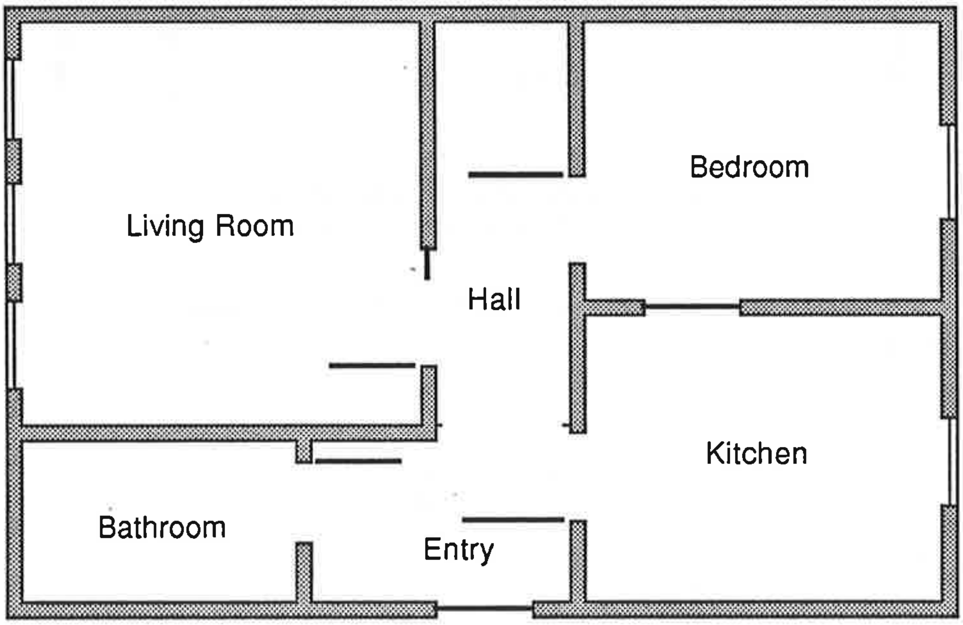


Figure 9-1 The IBR Five-Room Test House

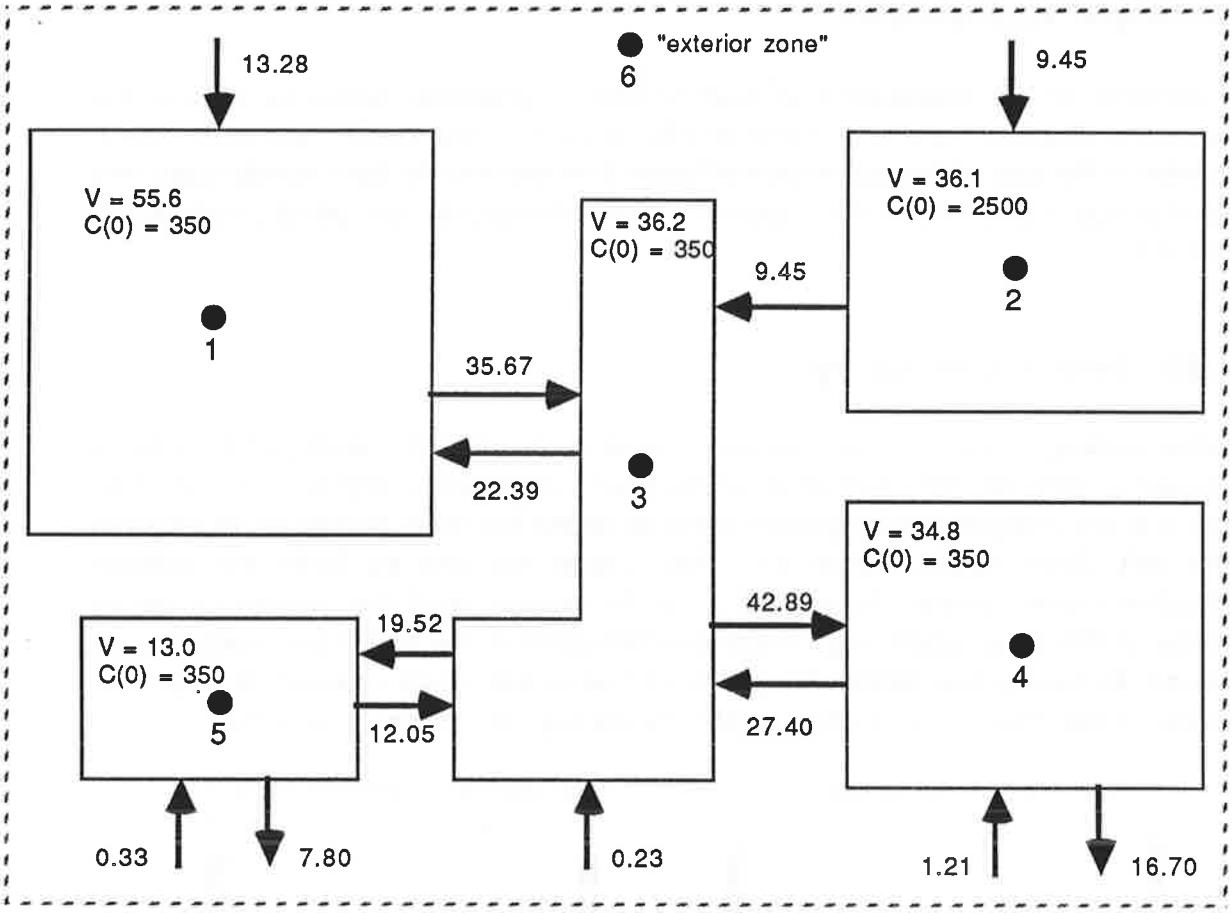


Figure 9-2 Idealization of IBR Test House
(all flows [=] l/s; volumes [=] m^3; initial concentrations [=] ppm)

CONTAM87 Command/Data File The CONTAM87 command/data file used to complete the analysis is listed below.

```
* IBR5ZONE:  Swedish IBR Test House: 5 Zone Model
*            Units m, hr
*            Analysis by volume rather than mass (i.e., air and
*            contaminant density set to unity).
FLOWSYS
N=6  S=1  ID=CO2
6   BC=C        <"Ext. Zone" Conc. Specified
<               < Air density set to unity.
1   V=55.6      < Node 1: Vol. Mass
2   V=36.1      < Node 2: Vol. Mass
3   V=36.2      < Node 3: Vol. Mass
4   V=34.8      < Node 4: Vol. Mass
5   V=13.0      < Node 5: Vol. Mass
6   V=1.0E+06   < Node 6: Exterior Vol. Mass
END
FLOWELEM
1   I=6,1
2   I=6,2
3   I=1,3
```

```
4   I=3,1
5   I=2,3
6   I=3,5
7   I=5,3
8   I=3,4
9   I=4,3
10  I=6,5
11  I=5,6
12  I=6,3
13  I=6,4
14  I=4,6
END
*
*  STEADY STATE ANALYSIS: CO2 generation assumed constant at 80 l/sec.
*
STEADY           < Flow rates = m3/sec x 3600 sec/hr
1   W=13.28E-03*3600
2   W=9.45E-03*3600
3   W=35.67E-03*3600
4   W=22.39E-03*3600
5   W=9.45E-03*3600
6   W=19.52E-03*3600
7   W=12.05E-03*3600
8   W=42.89E-03*3600
9   W=27.40E-03*3600
10  W=0.33E-03*3600
11  W=7.80E-03*3600
12  W=0.23E-03*3600
13  W=1.21E-03*3600
14  W=16.70E-03*3600
<                < Generation Rates & Specified Concentration
4   CG=80E-03    < Steady Generation [=] m3-CO2/hr
6   CG=350E-06   < Exterior Concentration [=] m3-CO2/m3-air
END
*
*  DYNAMIC ANALYSIS: Flow steady, CO2 generation 80 l/sec for 1.0 hr.
*
FLOWDAT
TIME=0.0
1   W=13.28E-03*3600
2   W=9.45E-03*3600
3   W=35.67E-03*3600
4   W=22.39E-03*3600
5   W=9.45E-03*3600
6   W=19.52E-03*3600
7   W=12.05E-03*3600
8   W=42.89E-03*3600
9   W=27.40E-03*3600
10  W=0.33E-03*3600
11  W=7.80E-03*3600
12  W=0.23E-03*3600
13  W=1.21E-03*3600
14  W=16.70E-03*3600
<
TIME=10.1
1   W=13.28E-03*3600
2   W=9.45E-03*3600
3   W=35.67E-03*3600
4   W=22.39E-03*3600
5   W=9.45E-03*3600
6   W=19.52E-03*3600
7   W=12.05E-03*3600
8   W=42.89E-03*3600
9   W=27.40E-03*3600
```

```
10  W=0.33E-03*3600
11  W=7.80E-03*3600
12  W=0.23E-03*3600
13  W=1.21E-03*3600
14  W=16.70E-03*3600
END
EXCITDAT
TIME=0.0
4    CG=80E-03        < Steady Generation [=] m3-CO2/hr
6    CG=350E-06       < Exterior Concentration [=] m3-CO2/m3-air
<
TIME=1.0
4    CG=0.0           < Steady Generation [=] m3-CO2/hr
6    CG=350E-06       < Exterior Concentration [=] m3-CO2/m3-air
<
TIME=10.1
4    CG=0.0           < Steady Generation [=] m3-CO2/hr
6    CG=350E-06       < Exterior Concentration [=] m3-CO2/m3-air
END
DYNAMIC
T=0,10,0.1 PS=1.0E+6
1      IC=350E-06
2      IC=2500E-06
3,6,1 IC=350E-06
END
RETURN
```

Results The results obtained using CONTAM87 are compared to those using Sirén's program MULTIC below, Figure 9-3. These results are practically identical, as they should be, as this study provides, in effect, a comparison of two numerical solutions of the identical system of equations. Nevertheless, the results do indicate that both programs have numerical procedures that have been correctly coded.

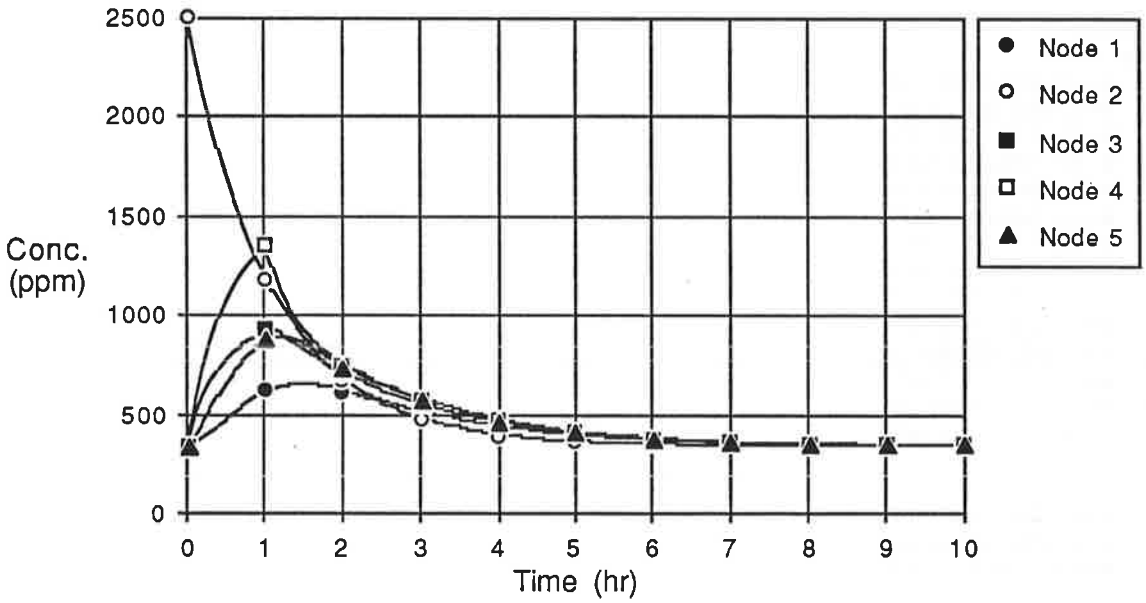
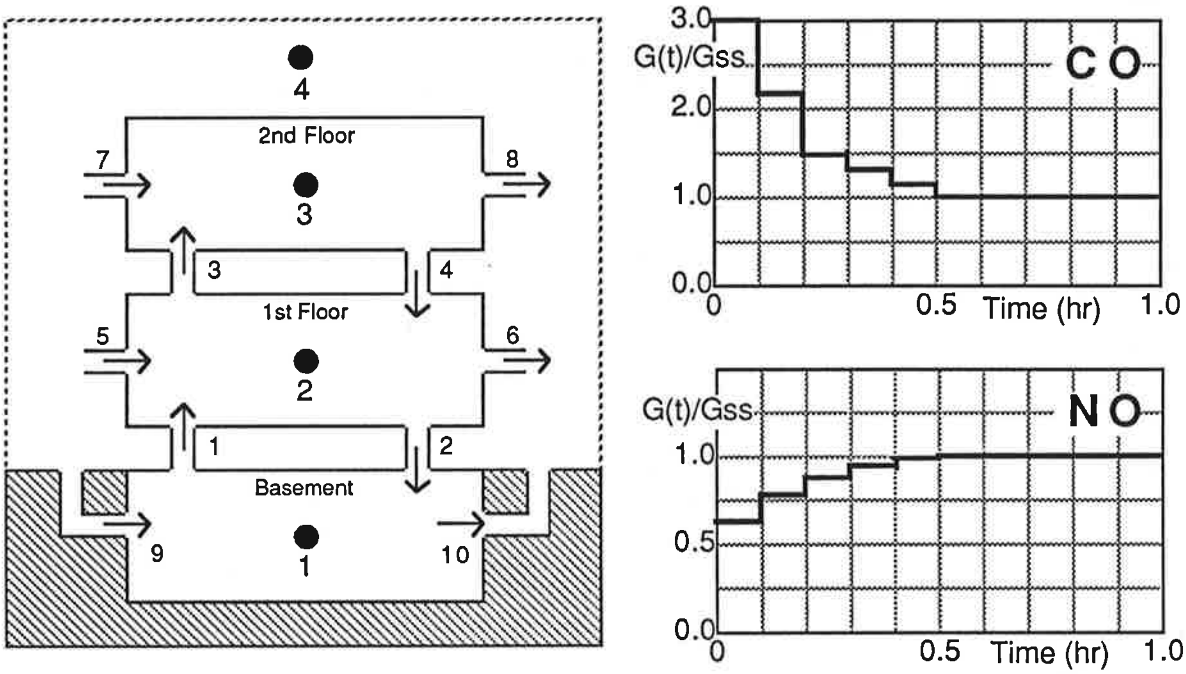


Figure 9-3 Comparison of MULTIC Results (markers) to CONTAM87 Results (lines)

9.2 Carnegie-Mellon Townhouse Study

Borrazzo and his colleagues at Carnegie-Mellon University, Pittsburg, Pennsylvania, have conducted detailed field investigations of a two-story townhouse measuring CO, NO, and NO_2 emissions characteristics of the gas appliances within the townhouse and the dispersal of these contaminants throughout the townhouse under a variety of different weather conditions [Borazzo et. al. 1987]. Illustrated in Figure 9-4 is an idealization of the townhouse and the measured dynamic emission characteristics of the principal pollutant source, the gas range. The instantaneous emission rate, G(t), is plotted relative to the steady state value, G_{ss}. The NO_2 emission characteristics were more or less constant and are, therefore, not illustrated. NO_2 is a reactive contaminant and was modeled as so using the <u>measured</u> reactivity of $K=2.4\ hr^{-1}$.



Building Idealization

Emission Characterics

Figure 9-4 Townhouse Building Idealization and Range Emission Characteristics

CONTAM87 Command/Data File The CONTAM87 command/data file used to complete the analysis for the NO_2 response is listed below.

```
*
* Borrazzo et al's. Townhouse 4-Node, 3-Zone Example
*  Units: g, hr, m
*
FLOWSYS
N=4 S=1 ID=NO2
4 BC=C                    < Node 4 is exterior node
<
```

```
1,3 V=126.5*1.2E+03              < Air Density = 1.2E+03 g/m3
4   V=126.5E+09*1.2E+03          < Exterior Zone Set At Large Value
END
FLOWELEM
1 I=1,2
2 I=2,1
3 I=2,3
4 I=3,2
5 I=4,2
6 I=2,4
7 I=4,3
8 I=3,4
9 I=4,1
10 I=1,4
END
FLOWDAT
TIME=10.0
1,2,1  W=0.4*126.5*1.2E+03       < ACH bsmnt-to-first
3,4    W=7.5*126.5*1.2E+3        < ACH first-to-second
5,8    W=0.21*126.5*1.2E+03      < 0.21  ACH ext-to-first & second
9,10   W=0.21*126.5*1.2E+03      < ACH ext-to-bsmnt
<
TIME=20.0
1,2,1  W=0.4*126.5*1.2E+03       < ACH bsmnt-to-first
3,4    W=7.5*126.5*1.2E+3        < ACH first-to-second
5,8    W=0.21*126.5*1.2E+03      < 0.21  ACH ext-to-first & second
9,10   W=0.21*126.5*1.2E+03      < ACH ext-to-bsmnt
END
KINELEM
K=1                              < Rxn 1: NO2 => products
2.4
<
1 I=1 K=1                        < Kinelem 1: Node 1: Rxn 1
2 I=2 K=1                        < Kinelem 2: Node 2: Rxn 1
3 I=3 K=1                        < Kinelem 2: Node 2: Rxn 1
END
*  Transient NO2 Emission Model (basis:Gss = 12 µg/kJ ; Igas = 150 kJ/min)
EXCITDAT
TIME=10.0
2   CG=0.0068                    <Pilot On - Burner Off
4   CG=0.0206E-06                <0.013 ppm * (46/28.98)
<
TIME=10.7
2   CG=0.108 +0.0068             <Pilot On - Burner On
4   CG=0.0206E-06                <0.013 ppm * (46/28.98)
<
TIME=11.7
2   CG=0.0068                    <Pilot On - Burner Off
4   CG=0.0206E-06                <0.013 ppm * (46/28.98)
<
TIME=20.0
2   CG=0.0068                    <Pilot On - Burner Off
4   CG=0.0206E-06                <0.013 ppm * (46/28.98)
END
DYNAMIC
T=10,16.0,0.1 RI=1  PS=28.98E+6/46
1     IC=0.0
2,3   IC=0.0
```

```
4    IC=0.0206E-06               <0.013 ppm * (46/28.98)
END
RETURN
```

The CONTAM87 command /data files for the CO and NO analysis would be identical to the file listed above with the species IDs changed from NO_2 to CO and NO, respectively, the KINELEM command removed, and the EXCITDAT and DYNAM commands replaced with those listed below. Note that in both cases a constant pilot light generation contribution plus a dynamically varying burner generation contribution is accounted for.

For CO:

```
*  Transient CO Emission Model (basis:Gss = 98 µg/kJ ; Igas = 150 kJ/min)
EXCITDAT
TIME=10.0
2  CG=0.0415                     <Pilot On - Burner Off
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=10.7
2  CG=3*0.882+0.0415             <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=10.8
2  CG=2.2*0.882+0.0415           <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=10.9
2  CG=1.5*0.882+0.0415           <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=11.0
2  CG=1.3*0.882+0.0415           <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=11.1
2  CG=1.2*0.882+0.0415           <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=11.2
2  CG=0.882+0.0415               <Pilot On - Burner On
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=11.7
2  CG=0.0415                     <Pilot On - Burner Off
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
<
TIME=20.0
2  CG=0.0415                     <Pilot On - Burner Off
4  CG=0.389E-06                  <0.403 ppm * (28/28.98)
END
DYNAMIC
T=10,16.0,0.1 RI=1  PS=28.98E+6/28
1     IC=0.389E-06
2,3   IC=0.389E-06
4     IC=0.389E-06
END
RETURN
```

For NO:

```
*  Transient NO Emission Model (basis:Gss = 17 µg/kJ; Igas = 150 kJ/min)
EXCITDAT
TIME=10.0
2   CG=0.0038                    <Pilot On - Burner Off
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=10.7
2   CG=0.7*0.153+0.0038          <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=10.8
2   CG=0.75*0.153+0.0038         <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=10.9
2   CG=0.8*0.153+0.0038          <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=11.0
2   CG=0.85*0.153+0.0038         <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=11.1
2   CG=0.95*0.153+0.0038         <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=11.2
2   CG=1.0*0.153+0.0038          <Pilot On - Burner On
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=11.7
2   CG=0.0038                    <Pilot On - Burner Off
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
<
TIME=20.0
2   CG=0.0038                    <Pilot On - Burner Off
4   CG=0.051E-06                 <0.049 ppm * (30/28.98)
END
DYNAMIC
T=10,16.0,0.1 RI=1  PS=28.98E+6/30
1     IC=0.124E-06
2,3   IC=0.144E-06
4     IC=0.051E-06
END
RETURN
```

Results In Figure 9-5 and 9-6 we compare computed response with measured data. The details of air flow in this building were unknown in some instances and uncertain in others so several assumptions about flow had to be made to effect the analysis. In particular, it was assumed that the measured whole-building fresh air infiltration rate of 0.21 air changes per hour (ACH) was distributed equally in all three zones, the first-to-second air exchange rate was assumed to be 7.5 ACH[1], the first-to-basement air

[1] Borrazzo et al. attempted to determine this interzonal air change rate from measured concentration

exchange rate was assumed to be 0.4 ACH, and all flows were assumed to be constant.

As may be seen, the CO response was under-predicted and the NO response was over-predicted, but both are within the reported uncertainty of the emission characteristics (CO: 18% & NO: 6.5%).

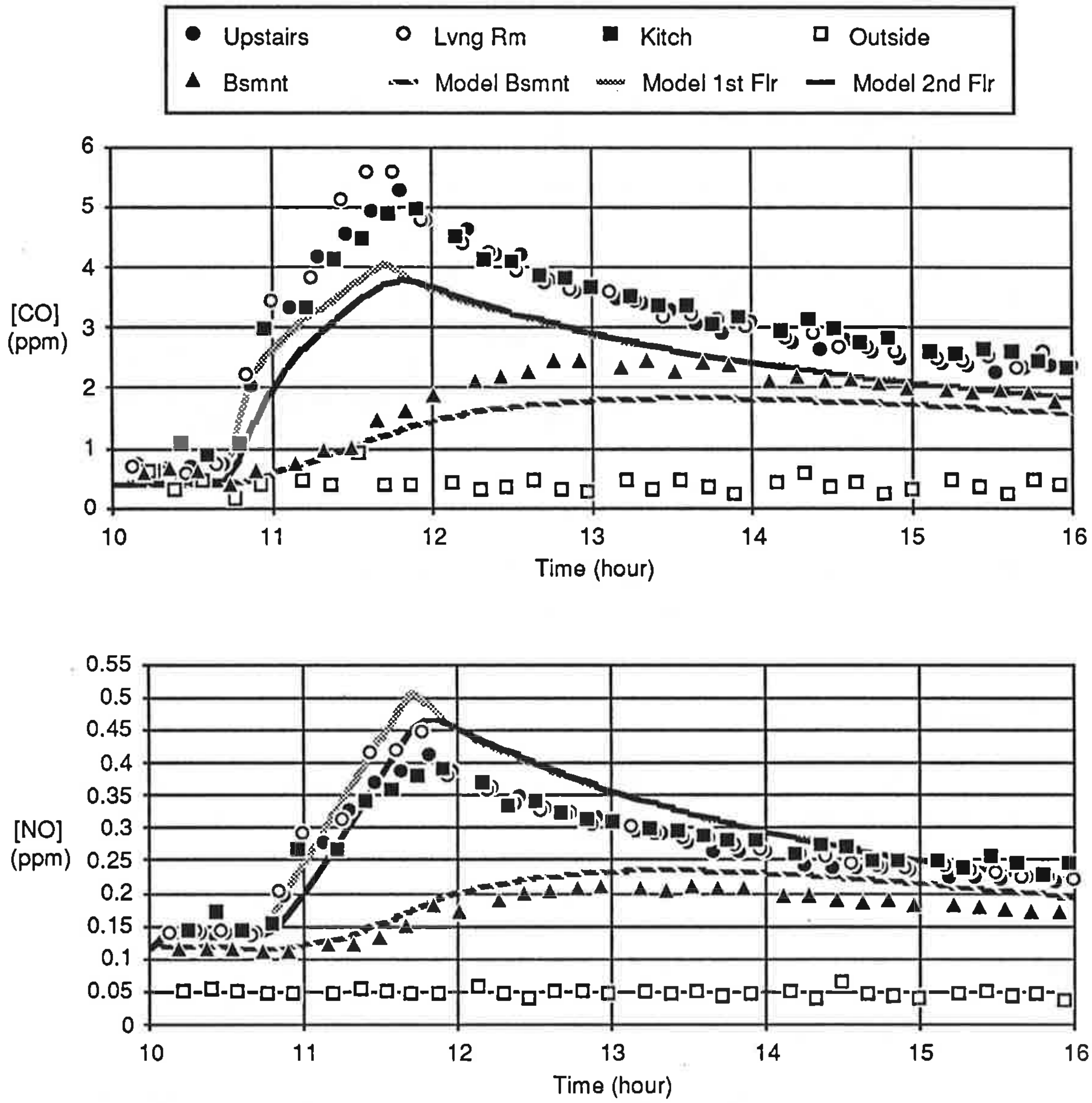


Figure 9-5 Comparison of Computed and Measured CO & NO Response

Although, the measured NO_2 data is quite suspect, because of its scatter and negative values, there appears to be some agreement between this data and the computed response. Importantly, it is noted that the NO_2 concentration can fall below ambient levels out-of-doors due to the reactivity of this contaminant.

data, reporting an interzonal exchange rate of 1.35 ACH. Their method was considered to be very poorly conditioned, thus, their results unreliable, and, therefore, the interzonal air change rate was assumed based upon the computed behavior of the townhouse and past experience.

Inasmuch as the measured data was used to determine the reactivity constant the agreement here may be an artifice. The basis of determination of the reactivity and the basis of the computed response are more or less the same as the system behaves, practically, as a single zone system, therefore the agreement may reflect no more than this.

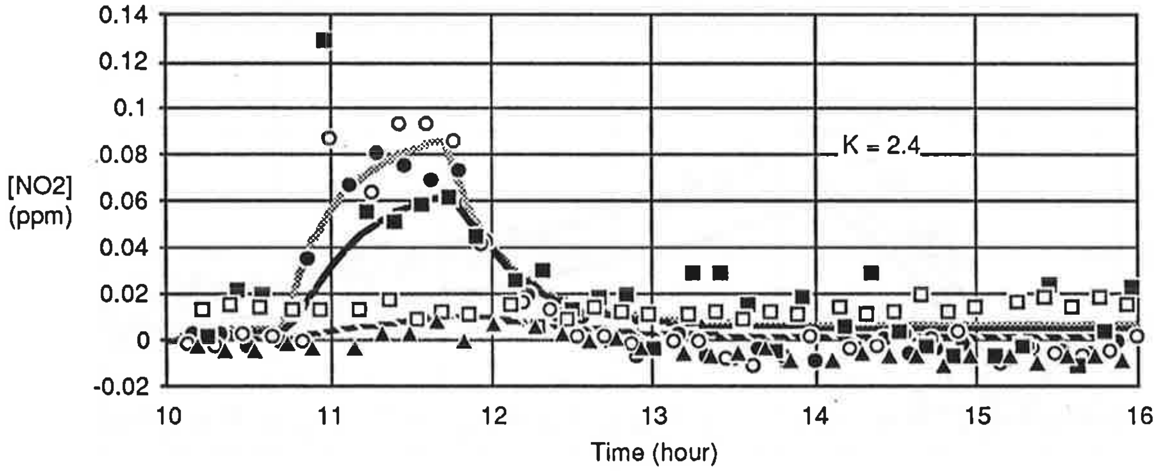


Figure 9-6 Comparison of Computed and Measured NO_2 Response
(NO_2 Reactivity = 2.4 hr^{-1})

9.3 NBS Office Building Study

Infiltration and ventilation studies of a fifteen story office building are presently being conducted by members of the Indoor Air Quality and Ventilation Group at NBS. Some of these studies involve periodic injections of a commonly used tracer gas, SF_6, into the fresh air supply ports of the building HVAC system. Flows in the supply ducts were measured (with significant uncertainty) by hot-wire anemometer traverse, SF_6 concentration time histories were recorded, and outdoor air change rates were estimated by tracer decay. Using the air flow measurements the upper two floors of this building were idealized as shown in Figure 9-7.

As indicated by this idealization, fresh air was supplied to each floor through a ceiling plenum space and exhausted via an exhaust duct to the outside. In Figure 9-8 we compare measured SF_6 concentration time histories (measured centrally within the "space" and at the "exhaust" ports) to computed values of the 15th floor for two supply flow rates: 100% and 75% of the measured flow. In this case, the agreement between measured and computed time histories is within the uncertainty of the measured flows and validation is therefore indicated.

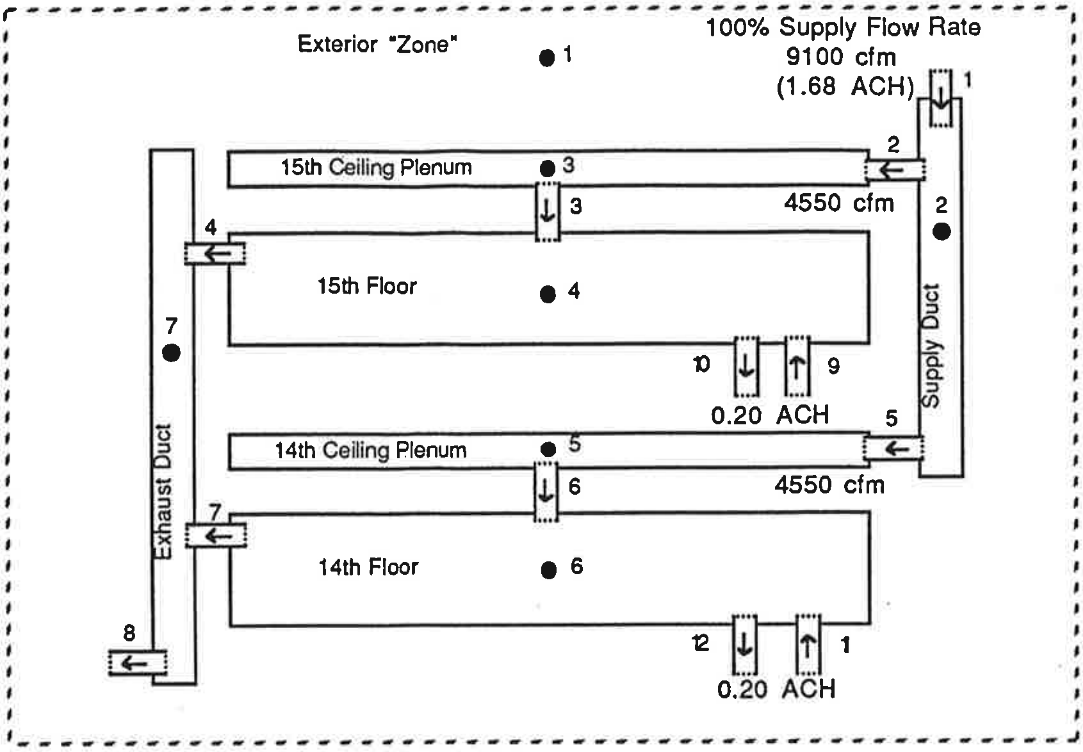


Figure 9-7 Idealization of the 14th and 15th Floors of an Office Building

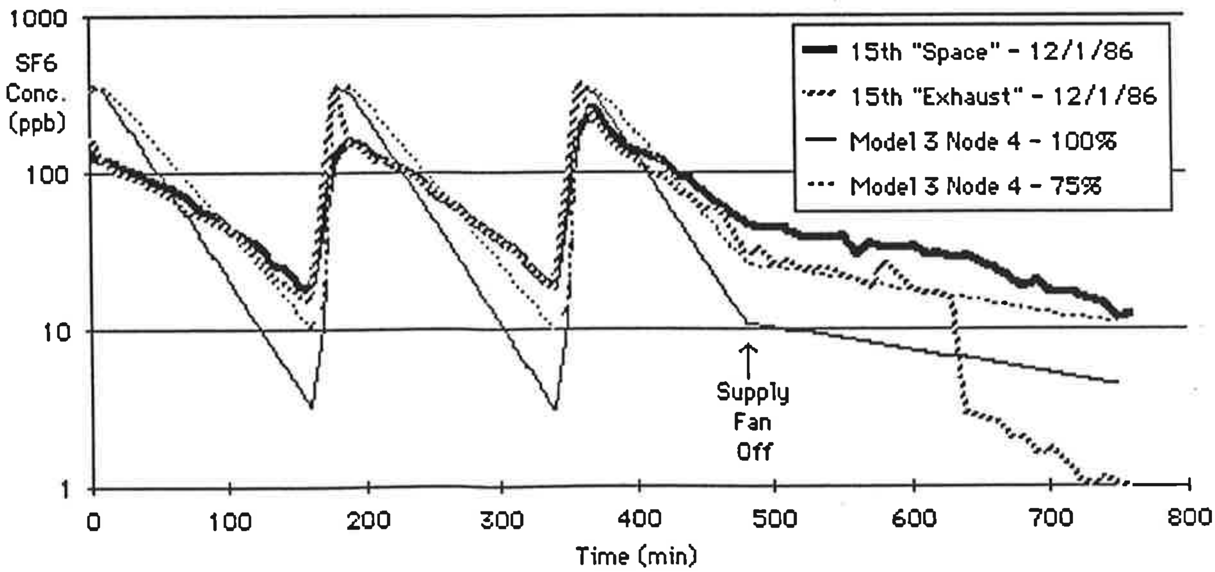


Figure 9-8 Comparison of Computed and Measured Response for an Office Building

10. Summary and Directions of Future Work

In the first section of this report we have attempted to give clearer definition to the emerging field of indoor air quality analysis. It has been argued that the central problem of indoor air quality analysis is *contaminant dispersal analysis* and that the related problems of *inverse contaminant dispersal analysis*, *flow analysis*, and *thermal analysis* may be thought to serve the needs of contaminant dispersal analysis. Furthermore, we have suggested that the central problem and these related problems may be addressed with an integrated set of computational tools based on an element assembly formulation of the familiar well-mixed zone simplification of the macroscopic equations of motion for multi-zone building systems of arbitrary complexity. The CONTAM family of programs is presently under development to provide one demonstration of this integrated approach; the first two members of the family CONTAM86 and CONTAM87 are presently available and provide contaminant dispersal analysis tools.

The noninteractive contaminant dispersal theory presented in the Phase II report of this project [Axley 1987] has been extended in this report through:

a) a discussion of strategies of forming contaminant dispersal analysis equations for multi-zone systems involving multiple contaminant species,

b) the introduction of element equations that may be used to model mass transport phenomena governed by first order kinetics, and

c) through the introduction of element equations that may be used to model the details of mass transport driven by conduction and diffusion processes in one dimensional flow paths.

CONTAM87 provides a complete computational implementation of the contaminant dispersal theory presented earlier and that introduced here. As such, CONTAM87 provides a set of indoor air quality analysis *commands*[1] that are a superset of those made available in CONTAM86. Future members of the CONTAM family of programs will provide additional indoor air quality analysis commands superseding or complimenting those made available by earlier members of the family.

Although it is well recognized that kinetics plays an important role in chemical, radio-chemical, sorption, and settling processes that affect contaminant dispersal processes in buildings, the detailed knowledge needed to apply the kinetics analysis techniques presented here is often not available and actual field or experimental measured data needed to validate any modeling effort is scarce. The application of the

[1] A *command*, here, is a set of computational procedures that completes a basic indoor air quality analysis task.

kinetics techniques presented here, nominally known as *source* or *sink modeling*, has become an area of emphasis in the direction of our future work.

Although "good" source and sink models are essential to interactive contaminant dispersal analysis, they introduce a source of uncertainty, as they are inevitably based upon empirical correlations, that is not a problem in noninteractive contaminant dispersal analysis. Therefore, while validation of the contaminant dispersal analysis techniques developed for noninteractive contaminant dispersal involved, primarily, the verification program logic (i.e., the primary assumption involved was the assumption of conservation of mass), the validation of techniques developed for interactive contaminant dispersal analysis will, necessarily, focus on the validity of the specific source or sink models being employed.

At this time the kinetics of radon decay is well understood and simple models of the kinetics of formaldehyde emission and NO_2 reaction are available, yet multi-zone field measurements needed to validate the use of these models in the multi-zone context are wanting.

In the development and application of the one dimensional convection-diffusion element presented in this report, it was recognized that this element provided one means to model certain classes of imperfectly mixed zones, those zones that behave *as-if* they were one dimensional flow systems. Thus this mass transport element could be considered, also, to be a *imperfectly-mixed zone element*. Following this train of thought, a well-mixed zone, whose mass transport behavior is defined by its volumetric mass, may be thought to be modeled by a *well-mixed zone element*, rather than being considered a basic assumption of the underlying theory, and, therefore, the contaminant dispersal theory presented here may be generalized to remove the restricting assumption of perfect mixing. Presently, an attempt is being made to recast the element assembly approach to contaminant dispersal analysis in such a way as to avoid the limiting assumption of perfect mixing. In this new formulation of the theory the well-mixed model becomes one special case and a framework is provided for the development of other imperfectly mixed zone elements.

Two parallel research efforts in the areas of inverse contaminant dispersal analysis and flow analysis, respectively, are also presently underway, complimenting the contaminant dispersal analysis work reported here. In the former area integral formulations of the multi-zone inverse analysis problem have been formulated and used to develop a new multi-zone tracer gas technique. Field applications of this technique have proven the technique to be promising. In the flow analysis area, the flow elements developed during Phase II of this project [Axley 1987] have undergone further refinement and some additional elements have been formulated. The results of first applications of these new flow elements have been encouraging.

REFERENCES

Axley, J.W., DTAM1: A Discrete Thermal Analysis Method for Building Energy Simulation: Part I Linear Thermal Systems with DTAM1 Users' Manual, 1985 (presently under review for publication by the National Bureau of Standards, Building Environment Division, Center for Building Technology)

Axley, James, Indoor Air Quality Modeling: Phase II Report, NBSIR 87-3661, CBT, National Bureau of Standards, Gaithersburg, MD, Oct., 87

ASHRAE, ASHRAE Handbook 1985 Fundamentals, SI Edition, ASHRAE, Atlanta, GA. , 1985, pp. 33.11-12

Bathe, K.J., Finite Element Procedures in Engineering Analysis, Prentice-Hall, Inc., 1982, pp. 702-706

Borrazzo, J.E., Osborn, J.F., Fortmann, R.C., & Davidson, C.L., " Modeling and Monitoring of CO, NO and NO_2 in a modern Townhouse", Atmospheric Environment, Vol. 21, No. 2, Pergamon Press, 1987, pp. 299-311

Eberlein, P.J. & Boothroyd, J., "Contribution II/12: Solution to the Eigenproblem by a Norm Reducing Jacobi Type Method," Handbook for Automatic Computation: Volume II: Linear Algebra, Wilkinson, J.H. & Reinsch,& Reinsch, C. - editors, Springer-Verlag, 1971

Grot, R.A., Silberstein, S., & Ishiguro, K., Validation of Models for Predicting Formaldehyde Concentrations in Residences Due to Pressed Wood Products: Phase I, NBSIR 85-3255, CBT, NBS, Gaithersburg, MD, Sept., 1985

Huebner, K.H. & Thornton, E.A., The Finite Element Method for Engineers, 2nd Edition, John Wiley & Sons, New York, 1982 , pp. 444-451

Hughes, T.J.R. & Brooks, A., "A Theoretical Framework for Petrov-Galerkin Methods with Discontinuous Weighting Functions: Application to the Streamline-Upwind Procedure," Finite Elements in Fluids, Volume 4, Edited by Gallagher et.al., John Wiley & Sons, 1982, pp. 47-65

Matthews, T.G., Reed, T.J., Daffron, C.R., & Hawthorne, A.R., "Environmental Dependence of Formaldehyde Emissions from Pressed-Wood Products: Experimental Studies and Modeling," Proceedings, 18th International Washington State University Particleboard/Composite Materials Symposium, pp. 10-23,1984

McNall, P., Walton, G., Silberstein, S., Axley, J., Ishiguro, K., Grot, R., & Kusuda, T., Indoor Air Quality Modeling Phase I Report: Framework for Development of General Models, NBSIR 85-3265, U. S. Dept. of Commerce, National Bureau of Standards, October 1986

Moore, J.W. & Pearson, R.G., Kinetics and Mechanism Third Edition, John Wiley & Sons, New York, 1981

Nauman, E.B. & Buffham, B.A., Mixing in Continuous Flow Systems, John Wiley & Sons, NY, 1983

Nicholas, J.E., Chemical Kinetics: A Modern Survey of Gas Reactions, John Wiley & Sons, New York, 1976

Nitschke, I. A., Traynor, G.W., Wadach, J.B., Clarkin, M.E., & Clarke, W.A., Indoor Air Quality, Infiltration and Ventilation in Residential Buildings, NYSERDA Report 85-10, N.Y. State ERDA, Albany, N.Y., Mar. 1985

Sandberg, Mats, "The Multi-Chamber Theory Reconsidered from the Viewpoint of Air Quality Studies," Building and Environment, Vol. 19, No. 4, Pergamon Press, Great Britain, 1984 pp. 221-233

Sinden, F.W., "Multi-Chamber Theory of Air Infiltration," Building and Environment, Vol. 13, Pergamon Press, Great Britain, 1978 pp. 21-28

Sirén, Kai, A Computer Program to Calculate the Concentration Histories and Some Air Quality Related Quantities in a Multi-Chamber System, Helsinki University of Technology, Institute of Energy Engineering, Otaniemi, 1986

Tezduyar, T.E. & Ganjoo, D.K., "Petrov-Galerkin Formulations with Weighting Functions Dependent Upon Spatial and Temporal Discretizations: Applications to Transient Convection-Diffusion Problems," Computer Methods in Applied Mechanics and Engineering, Vol. 59, Elsevier Science Publishers, North Holland, 1986, pp. 49-71

Traynor, G.W., Allen, J.R., Apte, M.G., Girman, J.R., & Holowell, C.D., "Pollution Emissions from Portable Kerosene-Fired Space Heaters", Environmental Science & Technology, Vol. 17, June 1983, pp. 369-371

Walas, S.M., Reaction Kinetics for Chemical Engineers, McGraw-Hill, New York, 1959

Walton, G.N., Estimating Interroom Contaminant Movements, NBSIR 85-3229,U.S. DOC, NBS, Gaithersburg, MD, August, 1985

Wen, C.Y., & Fan, L.T., Models for Flow Systems and Chemical Reactors, Marcel Dekker, Inc., NY,NY, 1975

Yu, C.C. & Heinrich, J.C., "Petrov-Galerkin Methods for the Time-Dependent Convective Transport Equation,", International Journal for Numerical Methods in Engineering, Vol. 23, John Wiley & Sons, 1986, pp. 883-901

Zienkiewicz, O.C. & Morgan, K., Finite Elements and Approximation, John Wiley & Sons, NY, 1983

APPENDIX
CONTAM87 FORTRAN SOURCE CODE

The FORTRAN77 source code for CONTAM87 is listed below. In this listing you will note the use of the compiler directives "INCLUDE". These directives are commonly available in most compilers but are not part of the FORTRAN language. They are used to "include" code stored in separate "include files" that, here, contain common block data specifications shared by many subroutines. The contents of these include files are listed on the last page of this appendix.

```
C================================================================= CONTAM
      PROGRAM CONTAM
C-----------------------------------------------------------------------
C--PRO:CONTAM - BUILDING CONTAMINANT DISPERSAL ANALYSIS PROGRAM
C              VERSION FY87
C
C----- Developed by James Axley
C                   Building Environment Division, NBS
C                   Spring 1987
C        Using;
C        A) CAL-SAP Library of subroutines developed by Ed Wilson,
C           U.C. Berkeley
C        B) MicroSoft FORTRAN V2.2 Compiler for Apple Macintosh
C           For Mac
C             1. Set logical unit numbers, in SUBROUTINE INITIO, as;
C                NTR = 9  ;  NTW = 9   ;    NCMD = 9
C             2. INCLUDE statements use <filename>.INC (i.e., without ')
C             3. In SUBROUTINE PROMPT use: WRITE(NTW,'(A,\)') STRING
C          4. In SUBROUTINE EIGEN2 use: WRITE(... A,\) at Section 2.0
C        C) IBM PC Professional FORTRAN (Ryan-McFarland)
C             1. Set logical unit numbers, in SUBROUTINE INITIO, as;
C                NTR = 5  ;  NTW = 6   ;    NCMD = 5
C             2. INCLUDE statements use '<filename>.INC' (i.e., with ')
C             3. In SUBROUTINE PROMPT use: WRITE(NTW,'(A)') STRING
C          4. In SUBROUTINE EIGEN2 don't use: WRITE(... A) at Section 2.0
C
C        Memory for dynamically allocated/defined arrays is located in
C        vector IA(MTOT) in blank common.  To increase or decrease this
C        area alter the dimension of IA, in the section 0.0 below, set
C        MTOT, in section 1.0 below, equal to this new dimension, and
C        recompile the code.  As integers are 4 bytes wide, memory
C        dedicated to IA(MTOT) is equal to MTOT*4 bytes.
C
C        The number of species is presently limited to MAXSPE=25.
C        This may be changed by altering MAXSPE in the CNTCOM.INC file.
C-----------------------------------------------------------------------
C
      IMPLICIT REAL*8 (A-H,O-Z)

C-----------------------------------------------------------------------
C--0.0 DATA SPECIFICATIONS & COMMON STORAGE
C-----------------------------------------------------------------------

      COMMON MTOT,NP,IA(50000)

      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'
      INCLUDE 'CMDCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR

C-----------------------------------------------------------------------
C--1.0 INITIALIZE INTERNAL VARIABLES
C-----------------------------------------------------------------------

      MTOT = 50000
      CALL INITAR(MTOT)
      CALL INITIO
      CALL INITCN
      ERR=.FALSE.

C-----------------------------------------------------------------------
C--2.0 WRITE BANNER
C-----------------------------------------------------------------------
      CALL BANNER(NTW)
      CALL BANNER(NOT)
      WRITE(NOT,2200)  (FNAME(1:LFNAME)//'.OUT')
 2200 FORMAT(/' === RESULTS OUTPUT FILE: ',(A))

C-----------------------------------------------------------------------
C--3.0 COMMAND PROCESSOR LOOP
C-----------------------------------------------------------------------
C
C---3.1 CHECK BLANK COMMON STORAGE
C
   30 NSTOR = (IDIR-NEXT-20)*IP(1)/IP(2)
      IF(NSTOR.LE.100) THEN
        WRITE(NTW,2300) NSTOR
        WRITE(NOT,*)
        WRITE(NOT,2300) NSTOR
      ENDIF
 2300 FORMAT(
     +' **** WARNING: Array storage available =',I9,' real numbers.')
C
C---3.2 GET COMMAND LINE
C
      IF(MODE.EQ.'INTER') CALL PROMPT(' CMND>'//CHAR(7))
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
C
C---3.3 INTERPRET COMMAND LINE
C
C------ GET COMMAND & ARRAY NAMES, IF ANY
C
      CALL FREEC(' ',NCMND,8,1)
      CALL FREEC('A',M1(1),4,7)
C
C------ INTRINSIC COMMANDS
C
      IF((NNCMND.EQ.'H').OR.(NNCMND.EQ.'HELP')) THEN
        IF(MODE.EQ.'BATCH') THEN
          CALL ALERT('ERROR: Command not defined in BATCH mode.',
     +    '$','$')
          CALL RETRN
        ELSE
          CALL HELP
        ENDIF

      ELSEIF((NNCMND.EQ.'ECHO-ON').OR.(NNCMND.EQ.'ON')) THEN
        ECHO = .TRUE.

      ELSEIF((NNCMND.EQ.'ECHO-OFF').OR.(NNCMND.EQ.'OFF')) THEN
        ECHO = .FALSE.

      ELSEIF((NNCMND.EQ.'L').OR.(NNCMND.EQ.'LIST')) THEN
        IF(MODE.EQ.'BATCH') THEN
          CALL ALERT('ERROR: Command not defined in BATCH mode.',
     +    '$','$')
          CALL RETRN
        ELSE
          CALL LIST
        ENDIF

      ELSEIF((NNCMND.EQ.'P').OR.(NNCMND.EQ.'PRINT')) THEN
        CALL PRINT

      ELSEIF((NNCMND.EQ.'D').OR.(NNCMND.EQ.'DIAGRAM')) THEN
        CALL DIAGRM

      ELSEIF(NNCMND.EQ.'PAUSE') THEN
        PAUSE '   ** PAUSE: Enter <CR> to continue.'

      ELSEIF((NNCMND.EQ.'S').OR.(NNCMND.EQ.'SUBMIT')) THEN
        IF(MODE.EQ.'BATCH') THEN
          CALL ALERT('ERROR: Command not defined in BATCH mode.',
     +    '$','$')
          CALL RETRN
        ELSE
          CALL SUBMIT
        ENDIF

      ELSEIF((NNCMND.EQ.'R').OR.(NNCMND.EQ.'RETURN')) THEN
        IF(MODE.EQ.'INTER') THEN
          WRITE(NTW,2320)
        ELSE
          CALL RETRN
        ENDIF
 2320   FORMAT(' **** ERROR: Command not defined in INTERACTIVE mode.')

      ELSEIF((NNCMND.EQ.'Q').OR.(NNCMND.EQ.'QUIT')) THEN
        CLOSE(NOT)
        STOP
C
C------ CONTAM COMMANDS
C
      ELSEIF(NNCMND.EQ.'FLOWSYS') THEN
        CALL FLOSYS

      ELSEIF(NNCMND.EQ.'FLOWELEM') THEN
        CALL FLOELM

      ELSEIF(NNCMND.EQ.'KINELEM') THEN
        CALL KINELM

      ELSEIF(NNCMND.EQ.'FORM-[W]') THEN
        CALL FORMF0

      ELSEIF(NNCMND.EQ.'STEADY') THEN
        CALL STEADY

      ELSEIF(NNCMND.EQ.'TIMECONS') THEN
        CALL TIMCON

      ELSEIF(NNCMND.EQ.'FLOWDAT') THEN
        CALL FLODAT

      ELSEIF(NNCMND.EQ.'EXCITDAT') THEN
        CALL EXCDAT

      ELSEIF(NNCMND.EQ.'DYNAMIC') THEN
        CALL DYNAM

      ELSEIF(NNCMND.EQ.'RESET') THEN
        CALL RESET

      ELSE
        CALL ALERT('ERROR: Command not defined.','$','$')
        IF(MODE.EQ.'BATCH') CALL RETRN

      ENDIF
      GO TO 30

      END
```

```
C---------------------------------------------------------------- INITAR
      SUBROUTINE INITAR(MTOT)
C--SUB:INITAR - INITIALIZES DYNAMIC ARRAY MANAGER VARIABLES
C               IN BLANK COMMON AND LABELED COMMON /ARYCOM/

      INCLUDE 'ARYCOM.INC'

      NUMA = 0
      NEXT = 1
      IDIR = MTOT
      IP(1) = 4
      IP(2) = 8
      IP(3) = 1
      RETURN
      END

C---------------------------------------------------------------- INITIO
      SUBROUTINE INITIO
C--SUB:INITIO - INITIALIZES LABELED COMMON /IOCOM/
C               OPENS DEFAULT RESULTS OUTPUT FILE

      INCLUDE 'IOCOM.INC'

      NTR  =  5
      NTW  =  6
      NCMD =  5
      NIN  = 10
      NOT  = 11
      NPLT = 12
      ND1  = 13
      ND2  = 14
      ND3  = 15
      ND4  = 16
      FNAME = 'CONTAM'
      LFNAME = 6
      EXT = '    '
      CALL NOPEN(NOT,(FNAME(1:LFNAME)//'.OUT'),'FORMATTED')
      MODE = 'INTER'
      ECHO = .TRUE.
      RETURN
      END

C---------------------------------------------------------------- INITCN
      SUBROUTINE INITCN
C--SUB:INITCN - INITIALIZES CONTAM LABELED COMMON /CNTCOM/

      INCLUDE 'CNTCOM.INC'

C--- INITIALIZE CONTAM CONTROL VARIABLES

      NSNOD  = 0
      NSSPE  = 1
      NSEQ   = 0
      MSBAN  = 0
      NFELM  = 0
      NKINEL = 0

C--- INITIALIZE POINTERS

      MPV    = 0
      MPVM   = 0
      MPF    = 0
      MPC    = 0
      MPE    = 0
      MPKSEQ = 0
      MPWE   = 0
      MPEFF  = 0
      MPDIFF= 0
      MPGENR= 0
      DO 10 N=1,9
   10 MPKIK(N) = 0
      MPTEMP = 0

C--- INITIALIZE OTHERS

      EP    = 1.0D-16

      RETURN
      END

C---------------------------------------------------------------- BANNER
      SUBROUTINE BANNER(LUN)
C--SUB: BANNER - WRITES PROGRAM BANNER TO LOGICAL UNIT LUN

      COMMON MTOT,NP,IA(1)

      WRITE(LUN,2000) MTOT
 2000 FORMAT(//,1X,78(1H-),/,
     .' |                       C O N T A M 8 7',T79,'|',/,
     .' |         Contaminant Dispersal Analysis for Building Systems'
     .,T79,'|',/,
     .' |                  Version 4/87 - Jim Axley - NBS',
     .T79,'|',/,1X,78(1H-),/,65X,'MTOT:',I9)


      RETURN
      END

C-----------------------------------------------------------------C
C                                                                 C
C                 I N T R I N S I C   C O M M A N D S             C
C                                                                 C
C-----------------------------------------------------------------C

C================================================================= HELP
      SUBROUTINE HELP
C--SUB: HELP - PROVIDES ON-SCREEN HELP
C
C--HELP LIST------------------------------------------------------
C
C     .' (H)ELP                List available intrinsic commands.',/,
C------------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      WRITE(NTW,2000)

      RETURN
C
C -------------------- HELP LIST ----------------------------------
C
 2000 FORMAT(/,' ==== INTRINSIC COMMANDS',//,
     .'' (H)ELP                   List available intrinsic commands.',/,
     .' ECHO-(ON)                 Echo results to screen.',/,
     .' ECHO-(OFF)                Do not echo results to screen.',/,
     .' (L)IST                    List the directory of all arrays.',/,
     .' (P)RINT   A=<array>       Print array named <array>.',/,
     .' (D)IAGRAM A=<array>       Diagram array named <array>.',/,
     .' (S)UBMIT F=<filename> Read commands from batch <filename>.',/,
     .' (R)ETURN                  Return to interactive mode.',/,
     .' (Q)UIT                    Quit program.'/)
C
      END

C================================================================= LIST
      SUBROUTINE LIST
C--SUB:LIST - LIST DIRECTORY OF ALL ARRAYS IN BLANK COMMON
C
C--HELP LIST------------------------------------------------------
C
C     .' (L)IST                List the directory of all arrays.',/,
C------------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'

      CHARACTER*1 NAM(4),LOC(4,2),TYPE(9,3),STOR(13,2)
      CHARACTER*1 CHK
C
      DATA TYPE/'I','N','T','E','G','E','R',' ',' ',
     1          'R','E','A','L',' ',' ',' ',' ',' ',
     2          'C','H','A','R','A','C','T','E','R'/
C
      DATA LOC/'C','O','R','E','D','I','S','K'/
C
      DATA STOR/'S','E','Q','U','E','N','T','I','A','L',' ',' ',' ',
     1          'D','I','R','E','C','T',' ','A','C','C','E','S','S'/
C
C-----LIST DIRECTORY OF ALL ARRAYS IN DATA BASE
      IF(NUMA.EQ.0) GO TO 900
C
C-----WRITE HEADER FOR SCREEN LISTING OF FILE DATA
      WRITE(NTW,1000)
C
C-----START COUNT OF LINES TO SCREEN
      IL = 5
C
      IC = IDIR
      DO 100 I=1,NUMA
      IL = IL + 1
      ILOC = 1
      IST = 0
      IA6 = IA(IC+6)
      IA7 = IA(IC+7)
      IA9 = IA(IC+9)
C-----CHECK FOR LOCATION AND STORAGE TYPE
      IF(IA9.GT.0) ILOC=2
      IF(IA7.LT.0) ILOC=2
      IF(IA7.EQ.-1) IST=1
      IF(IA7.EQ.-2) IST=2
      IF(IA9.GT.0) IST=3
      IPN = IC - 1
      DO 10 J=1,4
      IPN = IPN + 1
   10 NAM(J) = CHAR(IA(IPN))
C-----WRITE DATA TO TERMINAL
      IF(IST.EQ.0) WRITE(NTW,1100) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),(TYPE(K,IA6),K=1,9),
     * (LOC(L,ILOC),L=1,4)
C
      IF(IST.EQ.1) WRITE(NTW,1100) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),(TYPE(K,IA6),K=1,9),
     * (LOC(L,ILOC),L=1,4),(STOR(M,1),M=1,13)
C
      IF(IST.EQ.2) WRITE(NTW,1300) (NAM(J),J=1,4),
     * IA(IC+4),(LOC(L,ILOC),L=1,4),(STOR(M,2),M=1,13)
C
      IF(IST.EQ.3) WRITE(NTW,1200) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),IA(IC+6),(LOC(L,ILOC),L=1,4),
     * (STOR(M,2),M=1,13)
C
      IC = IC + 10
C-----CHECK FOR NUMBER OF LINES PRINTED
      IF(IL.LT.20) GO TO 100
      IF(I.EQ.NUMA) GO TO 100
      CALL PROMPT('   ** Do you want more ? (Y/N) ')
      READ(NTR,2200)
      IF((CHK.EQ.'n').OR.(CHK.EQ.'N')) GO TO 900
      IL = 0
      WRITE(NTW,2000)
  100 CONTINUE
C
  900 RETURN
C
 1000 FORMAT(' ==== LIST: ARRAY LIST',//,
     * ' Name',2X,'Number',2X,'Number',5X,'Data',5X,
     * 'Location',5X,'Storage',/,8X,'Rows',2X,
     * 'Columns',5X,'Type',19X,'Type',/)
 1100 FORMAT(1X,4A1,2X,I4,4X,I4,5X,9A1,4X,4A1,4X,13A1)
```

```
 1200 FORMAT(1X,4A1,'  NI=',I4,'  NR=',I4,'  NC=',I4,5X,4A1,4X,13A1)
 1300 FORMAT(1X,4A1,3X,'RECORD LENGTH = ',I6,7X,4A1,4X,13A1)
 2000 FORMAT()
 2200 FORMAT(1A1)
      END

C================================================================== PRINT
      SUBROUTINE PRINT
C--SUB:PRINT - COMMAND TO "PRINT" ARRAY TO RESULTS OUTPUT FILE
C
C--HELP LIST-----------------------------------------------------------
C
C     .' (P)RINT   A=<array>   Print array named <array>.',/,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'
      INCLUDE 'CMDCOM.INC'

      CHARACTER MA*4
      EQUIVALENCE (MA,M1(1))

C-----PRINT OF REAL OR INTEGER ARRAY
      CALL PROMH(1)
C-----LOCATE MATRIX TO BE PRINTED
      IF(ECHO) WRITE(NTW,2000) M1
      WRITE(NOT,2000) M1
      CALL LOCATE(M1,NA,NR,NC)
      IF(NA.EQ.0) THEN
        CALL ALERT('ERROR: Array '//MA//' does not exist.','$','$')
        CALL ABORT
        RETURN
      ELSEIF(NA.LT.0) THEN
        CALL ALERT('ERROR: Array '//MA//' is out of core.','$','$')
        CALL ABORT
        RETURN
      ELSE
        IF(NP.EQ.1) CALL IPRT(IA(NA),NR,NC)
        IF(NP.EQ.2) CALL RPRT(IA(NA),NR,NC)
        IF(NP.EQ.3) CALL CPRT(IA(NA),NR,NC)
      ENDIF

      RETURN
C
 2000 FORMAT(/' ==== PRINT   OF   ARRAY   "',4A1,'"')
      END

C------------------------------------------------------------------- IPRT
      SUBROUTINE IPRT(N,NR,NC)
C--SUB: IPRT - PRINTS INTEGER ARRAY TO RESULTS OUTPUT FILE

      DIMENSION N(NR,NC)

      INCLUDE 'IOCOM.INC'

      NUMC = 14
      DO 100 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000)  (K,K=I,IN)
      IF(ECHO) WRITE(NTW,2000)  (K,K=I,IN)
      DO 100 J=1,NR
      WRITE(NOT,2001) J, (N(J,K),K=I,IN)
      IF(ECHO) WRITE(NTW,2001) J, (N(J,K),K=I,IN)
  100 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',14I5)
 2001 FORMAT(' ROW',I4,14I5)
      END

C------------------------------------------------------------------- RPRT
      SUBROUTINE RPRT(A,NR,NC)
C--SUB: RPRT - PRINTS REAL ARRAY TO RESULTS OUTPUT FILE

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NR,NC)

      INCLUDE 'IOCOM.INC'

      XMAX = 0.00
      DO 50 I=1,NR
      DO 50 J=1,NC
      XX = DABS(A(I,J))
      IF(XX.GT.XMAX) XMAX = XX
   50 CONTINUE
      M = 1
      IF(XMAX.LT.99999.) M = 2
      IF(XMAX.LT.0.1000) M = 1
      IF(XMAX.EQ.0.0)    M = 2
C
      NUMC = 6
      DO 100 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000)  (K,K=I,IN)
      IF(ECHO) WRITE(NTW,2000)  (K,K=I,IN)
      DO 100 J=1,NR
      IF(M.EQ.1) THEN
        WRITE(NOT,2001) J, (A(J,K),K=I,IN)
        IF(ECHO) WRITE(NTW,2001) J, (A(J,K),K=I,IN)
      ELSEIF(M.EQ.2) THEN
        WRITE(NOT,2002) J, (A(J,K),K=I,IN)
        IF(ECHO) WRITE(NTW,2002) J, (A(J,K),K=I,IN)
      ENDIF
  100 CONTINUE
C     RETURN

 2000 FORMAT(/' COL# =',6I12)
 2001 FORMAT(' ROW',I4,6E12.5)
 2002 FORMAT(' ROW',I4,6F12.5)
      END

C------------------------------------------------------------------- CPRT
      SUBROUTINE CPRT(C,NR,NC)
C--SUB: CPRT - PRINTS CHARACTER*1 ARRAY TO RESULTS OUTPUT FILE

      CHARACTER C(NR,NC)*1

      INCLUDE 'IOCOM.INC'

      NUMC = 14
      DO 100 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000)  (K,K=I,IN)
      IF(ECHO) WRITE(NTW,2000)  (K,K=I,IN)
      DO 100 J=1,NR
      WRITE(NOT,2001) J, (C(J,K),K=I,IN)
      IF(ECHO) WRITE(NTW,2001) J, (C(J,K),K=I,IN)
  100 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',14I5)
 2001 FORMAT(' ROW',I4,14(4X,A1))
      END

C================================================================= DIAGRM
      SUBROUTINE DIAGRM
C--SUB:DIAGRM - COMMAND TO "DIAGRAM" ARRAY TO RESULTS OUTPUT FILE
C
C--HELP LIST-----------------------------------------------------------
C
C     .' (D)IAGRAM A=<array>   Diagram array named <array>.',/,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CMDCOM.INC'

      CHARACTER MA*4
      EQUIVALENCE (MA,M1(1))

C-----PRINT OF REAL OR INTEGER ARRAY
      CALL PROMH(1)
C-----LOCATE MATRIX TO BE PRINTED
      IF(ECHO) WRITE(NTW,2000) M1
      WRITE(NOT,2000) M1
      CALL LOCATE(M1,NA,NR,NC)
      IF(NA.EQ.0) THEN
        CALL ALERT('ERROR: Array '//MA//' does not exist.','$','$')
        CALL ABORT
        RETURN
      ELSEIF(NA.LT.0) THEN
        CALL ALERT('ERROR: Array '//MA//' is out of core.','$','$')
        CALL ABORT
        RETURN
      ELSEIF(NP.EQ.3) THEN
        CALL ALERT('ERROR: Array '//MA//' is a character array.',
     +  '$','$')
        CALL ABORT
        RETURN
      ELSE
        IF(NP.EQ.1) CALL IDIAGR(IA(NA),NR,NC)
        IF(NP.EQ.2) CALL RDIAGR(IA(NA),NR,NC)
      ENDIF

      RETURN
C
 2000 FORMAT(/' ==== DIAGRAM   OF   ARRAY   "',4A1,'"')
      END

C----------------------------------------------------------------- IDIAGR
      SUBROUTINE IDIAGR(N,NR,NC)
C--SUB: IDIAGR - "DIAGRAMS" INTEGER ARRAY TO RESULTS OUTPUT FILE

      INTEGER N(NR,NC)
      CHARACTER*1 ICON(36)

      INCLUDE 'IOCOM.INC'

C-----DIAGRAM INTEGER ARRAY
      NUMC = 36
      DO 200 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000)  (INT(K/10),K=I,IN)
      WRITE(NOT,2010)  ((K-INT(K/10)*10),K=I,IN)
      IF(ECHO) WRITE(NTW,2000)  (INT(K/10),K=I,IN)
      IF(ECHO) WRITE(NTW,2010)  ((K-INT(K/10)*10),K=I,IN)
      DO 200 J=1,NR
      DO 100 K=1,IN
       ICON(K) = '*'
       IF(N(J,K).EQ.0) ICON(K) = ' '
  100 CONTINUE
       WRITE(NOT,2020) J, (ICON(K),K=I,IN)
       IF(ECHO) WRITE(NTW,2020) J, (ICON(K),K=I,IN)
  200 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',36(1X,I1))
 2010 FORMAT(7X,36(1X,I1))
 2020 FORMAT(' ROW',I3,36(1X,A1))
      END
```

```
C------------------------------------------------------------------- RDIAGR
      SUBROUTINE RDIAGR(A,NR,NC)
C--SUB: RDIAGR - "DIAGRAMS" REAL ARRAY TO RESULTS OUTPUT FILE

      REAL*8 A(NR,NC)
      CHARACTER*1 ICON(36)

      INCLUDE 'IOCOM.INC'

C-----DIAGRAM INTEGER ARRAY
      NUMC = 36
      DO 200 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000) (INT(K/10),K=I,IN)
      WRITE(NOT,2010) ((K-INT(K/10)*10),K=I,IN)
      IF(ECHO) WRITE(NTW,2000) (INT(K/10),K=I,IN)
      IF(ECHO) WRITE(NTW,2010) ((K-INT(K/10)*10),K=I,IN)
      DO 200 J=1,NR
      DO 100 K=I,IN
       ICON(K) = '*'
       IF(A(J,K).EQ.0.0D0) ICON(K) = ' '
  100 CONTINUE
       WRITE(NOT,2020) J,(ICON(K),K=I,IN)
       IF(ECHO) WRITE(NTW,2020) J,(ICON(K),K=I,IN)
  200 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',36(1X,I1))
 2010 FORMAT(7X,36(1X,I1))
 2020 FORMAT(' ROW',I3,36(1X,A1))
      END

C=================================================================== SUBMIT
      SUBROUTINE SUBMIT
C--SUB: SUBMIT - SWITCHES TO BATCH MODE AND OPENS BATCH COMMAND FILE
C
C--HELP LIST-----------------------------------------------------------
C
C     .' (S)UBMIT F=<filename> Read commands from batch <filename>.',/,
C----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'
      LOGICAL FOUND

      CALL FREEC('F',FNAME,12,1)
      INQUIRE(FILE=FNAME(1:LENTRM(FNAME)),EXIST=FOUND)
      IF(FOUND) THEN
        MODE = 'BATCH'
        NCMD = NIN
        LFNAME = LENTRM(FNAME)
        WRITE(NTW,2010) FNAME
        WRITE(NOT,*)
        WRITE(NOT,2010) FNAME
 2010   FORMAT(' **** CONTAM set to BATCH mode using file: ',A)
        OPEN(NCMD,FILE=FNAME(1:LFNAME),STATUS='OLD')
        REWIND NCMD
        CLOSE(NOT)
        CALL NOPEN(NOT,(FNAME(1:LFNAME)//'.OUT'),'FORMATTED')
        CALL BANNER(NOT)
        WRITE(NOT,2020) (FNAME(1:LFNAME)//'.OUT')
 2020   FORMAT(/' ==== RESULTS OUTPUT FILE: ',(A))

      ELSE
        WRITE(NTW,2030)
 2030   FORMAT('     -- NOTE: Submit file not found.')
        CALL ABORT
      ENDIF

      RETURN
      END

C=================================================================== RETRN
      SUBROUTINE RETRN
C--SUB:RETRN - RETURNS TO INTERACTIVE MODE
C
C--HELP LIST-----------------------------------------------------------
C
C     .' (R)ETURN               Return to interactive mode.',/,
C----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      WRITE(NOT,*)
      WRITE(NOT,2010)

      CLOSE(NCMD)
      CLOSE(NOT)
      FNAME = 'CONTAM'
      LFNAME = 6
      OPEN(NOT,FILE=(FNAME(1:LFNAME)//'.OUT'),STATUS='OLD',
     +FORM='FORMATTED')
      CALL APPEND(NOT)
      NCMD = NTR
      MODE = 'INTER'

      WRITE(NTW,2010)
      WRITE(NOT,*)
      WRITE(NOT,2010)
 2010 FORMAT(' **** CONTAM returned to INTERACTIVE mode.')

      RETURN
      END

C--------------------------------------------------------------------C
C                                                                    C
C                    C O N T A M   C O M M A N D S                   C
C                                                                    C
C--------------------------------------------------------------------C

C=================================================================== FLOSYS
      SUBROUTINE FLOSYS
C--SUB:FLOSYS - COMMAND TO READ & PROCESS FLOW SYSTEM CONTROL VARIABLES
C               ESTABLISHES FLOW SYSTEM EQUATION NUMBERS & B.C.
C
C--HELP LIST-----------------------------------------------------------
C
C     .' FLOWSYS N=n1          Flowsystem control variables.',/,
C     .' N=n1 S=n2 ID=c1,c2,... n1 = no.flow nodes; n2= no. species
C     .' n3,n4,n5 BC=c3         c1,c2,... species IDs (4 chars)',/,
C     .' ...                    n3,n4,n5 = node: first,'last, incr.',/,
C     .' :                      c3 = boundary condition; G or C',/,
C     .' n3,n4,n5 V=n6          n6 = nodal volumetric mass',/,
C     .' ...',/,
C     .' END',//,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      INTEGER IJK(3)
      EXTERNAL BCDAT0,VDAT0

      ERR = .FALSE.

C
C--1.0 GET NUMBER OF FLOW SYSTEM NODES, NUMBER OF SPECIES, & SPECIES IDS
C
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2100)
      WRITE(NOT,2100)
 2100 FORMAT(/,' ==== FLOWSYS: FLOW SYSTEM CHARACTERISTICS')

      IF(NSNOD.NE.0) CALL RESET

      IF(MODE.EQ.'INTER') CALL PROMPT('DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)

C---1.1 NUMBER OF FLOW NODES

      CALL FREEI('N',NSNOD,1)
      CALL CKIZER('the number of flow system nodes',NSNOD,1,'GT',ERR)
      IF(ERR) GO TO 999

      IF(ECHO) WRITE(NTW,2120) NSNOD
      WRITE(NOT,2120) NSNOD
 2120 FORMAT(/,'      Number of flow system nodes ......',I5)

C---1.2 NUMBER OF SPECIES

      CALL FREEI('S',NSSPE,1)
      CALL CKIRNG('the number of contaminant species',NSSPE,1,
     +0,'LTLE',MAXSPE,ERR)
      IF(ERR) GO TO 999

      IF(ECHO) WRITE(NTW,2140) NSSPE
      WRITE(NOT,2140) NSSPE
 2140 FORMAT('      Number of contaminant species ....',I5)

C---1.3 SPECIES IDS

      CALL ZEROC(SID,4,NSSPE)
      CALL GETIDS(SID,NSSPE,'Contaminant species IDs:')

C
C--2.0 DEFINE KSEQ ARRAY AND NUMBER EQUATIONS IN (NODE,SPECIES) ORDER
C
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2150)
      WRITE(NOT,2150)
 2150 FORMAT(/,'  == Boundary Conditions and Equation Numbers')

      NSEQ = NSNOD*NSSPE

      CALL DELETE('KSEQ')
      CALL DEFINI('KSEQ',MPKSEQ,NSNOD,NSSPE)

      CALL EQNUM(IA(MPKSEQ),NSNOD,NSSPE)

C
C--3.0 PROCESS BOUNDARY CONDITION DATA & REPORT EQUATION NUMBERS
C
      CALL DATGEN(BCDAT0,NSNOD,ERR)
      IF(ERR) GO TO 999

      CALL RPRTK(IA(MPKSEQ),SID,NSNOD,NSSPE)

C
C--4.0 GET NODAL VOLUMETRIC MASSES
C
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2400)
      WRITE(NOT,2400)
 2400 FORMAT(/,'  == Nodal Volumetric Mass')

      CALL DELETE('V   ')
      CALL DEFINR('V   ',MPV,NSNOD,1)
      CALL ZEROR(IA(MPV),NSNOD,1)

      CALL DATGEN(VDAT0,NSNOD,ERR)
      IF(ERR) GO TO 999

      CALL RPRTNO(IA(MPV),NSNOD,'Node')

C
C--5.0 ORDERLY COMPLETION: SKIP TO "END"
C
  500 IF(EOC) RETURN
      IF(MODE.EQ.'INTER') CALL PROMPT('END?>')
      CALL FREE
      GO TO 500
```

```
C
C--9.0 ABORT IF ERR
C

  999 IF(ERR) THEN
        CALL DELETE('KSEQ')
        CALL DELETE('V   ')
        NSNOD = 0
        NSSPE = 0
        MPKSEQ = 0
        MPV = 0
        ERR = .FALSE.
        CALL ABORT
        RETURN
      ENDIF

      END
C------------------------------------------------------------------- EQNUM
      SUBROUTINE EQNUM(KSEQ,NSNOD,NSSPE)
C--SUB:EQNUM - ESTABLISHES EQUATION NUMBERS

      INTEGER KSEQ(NSNOD,NSSPE)

      NN = 0
      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
      NN = NN+1
   10 KSEQ(N,M) = NN
      RETURN
      END

C------------------------------------------------------------------ BCDAT0
      SUBROUTINE BCDAT0(N,ERR)
C--SUB:BCDAT0 - CALLS BCDAT1 PASSING TEMPORARY ARRAY
C
C-------POINTER--ARRAY----------------------------------------------------
C   MPBC BC(NSSPE)*1   :TEMPORARY STORAGE OF BC CODES
C-------------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR

      CALL BCDAT1(IA(MPKSEQ),CDATA,N,NSNOD,NSSPE,ERR)

      RETURN
      END

C------------------------------------------------------------------ BCDAT1
      SUBROUTINE BCDAT1(KSEQ,BC,N,NSNOD,NSSPE,ERR)
C--SUB: BCDAT1 - PROCESSES BC DATA

      INCLUDE 'IOCOM.INC'

      INTEGER KSEQ(NSNOD,NSSPE)
      CHARACTER BC(NSSPE)*1, BCM*1
      LOGICAL ERR

      CALL FREEC('C',BC(1),1,NSSPE)

      DO 10 M=1,NSSPE
      BCM = BC(M)
      IF((BCM.NE.'C').AND.(BCM.NE.'G').AND.(BCM.NE.' ')) THEN
        WRITE(NTW,2000) BCM,N,M
        WRITE(NOT,*)
        WRITE(NOT,2000) BCM,N,M
        ERR = .TRUE.
        RETURN
 2000 FORMAT(' **** ERROR: Boundary condition "',A1,'" not available.',
     +/,     '             Node:',I4,'    Species:',I4)
      ELSEIF(BCM.EQ.'C') THEN
        KSEQ(N,M) = -KSEQ(N,M)
   10 ENDIF

      RETURN
      END

C------------------------------------------------------------------- RPRTK
      SUBROUTINE RPRTK(KSEQ,SID,NSNOD,NSSPE)
C--SUB:RPRTK - REPORTS SYSTEM EQUATION NUMBERS STORED IN ARRAY
C              KSEQ(NSNOD,NSSPE)

      INCLUDE 'IOCOM.INC'

      INTEGER KSEQ(NSNOD,NSSPE)
      CHARACTER SID(NSSPE)*4

      IF(ECHO) WRITE(NTW,2000)  (SID(M),M=1,NSSPE)
      WRITE(NOT,2000)  (SID(M),M=1,NSSPE)

      DO 10 N=1,NSNOD
      IF(ECHO) WRITE(NTW,2010) N, (KSEQ(N,M),M=1,NSSPE)
      WRITE(NOT,2010) N, (KSEQ(N,M),M=1,NSSPE)
   10 CONTINUE

      RETURN

 2000 FORMAT(/,
     .6X,'Neg. Eqtn-# = concentration-prescribed (independent DOF).',/,
     .6X,'Pos. Eqtn-# = generation-prescribed (dependent DOF).',//,
     .13X,'Species ID:',/,
     .6X,'Node',10(3X,A4))
 2010 FORMAT(6X,I4,10(3X,I4))
      END

C------------------------------------------------------------------- VDAT0
      SUBROUTINE VDAT0(N,ERR)
C--SUB:VDAT0 - CALLS VDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR

      CALL VDAT1(IA(MPV),NSNOD,N,ERR)

      RETURN
      END
C------------------------------------------------------------------- VDAT1

      SUBROUTINE VDAT1(V,NSNOD,N,ERR)
C--SUB:VDAT1 - READS NODE VOLUMETRIC MASS DATA
C
      INCLUDE 'IOCOM.INC'

      REAL*8 V(NSNOD),VDAT
      LOGICAL ERR

      CALL FREER('V',VDAT,1)
      CALL CKRZER('nodal volumetric mass',VDAT,1,'GE',ERR)
      IF(ERR) RETURN
      V(N) = VDAT

      RETURN
      END

C================================================================== FLOELM
      SUBROUTINE FLOELM
C--SUB:FLOELM - COMMAND TO READ & WRITE FLOW ELEMENT DATA TO FILE *.FEL
C
C--HELP LIST--------------------------------------------------------------
C
C     .' FLOWELEM           Flow elements: Simple or Conv-diff.',/,
C     .' n1 T=SIMP I=n2,n3 E=n4',/,
C     .' or',/,
C     .' n1 T=CNDF I=n2,n3 M=n5  L=n6 D=n7,n8,... G=n9,n10,... F=n11',/,
C     .' ...                        n1 = elem. number; n2,n3 = end nodes ',/,
C     .' END                        n4 = filter eff.; n5 = mass/length;',/,
C     .'                            n6 = length; n7,n8,...= disp. coef.',/,
C     .'                            n9,n10 = generation; n11 = upwind fact.')
C-------------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR,FOUND
      EXTERNAL FLOEL0

C-----VARIABLE--------DESCRIPTION-----------------------------------------
C   ERR       ERROR FLAG
C   FOUND          FILE FOUND FLAG
C   FLOEL0         SUB. TO READ & WRITE FLOW ELEM DATA
C   NENOD          NUMBER OF ELEMENT NODES
C   NESTRT         FIRST ELEMENT NUMBER
C-------------------------------------------------------------------------
C
C--0.0 INITIALIZATION
C
      ERR = .FALSE.
      NENOD = 2

      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,' ==== FLOWELEM: FLOW ELEMENTS')
C
C--1.0 CHECK TO SEE IF SYSTEM NODES & EQUATION NUMBERS ARE DEFINED
C
      CALL CKSYS(1,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 OPEN <filename>.FEL
C
      INQUIRE(FILE=FNAME(1:LFNAME)//'.FEL',EXIST=FOUND)

      IF((.NOT.FOUND).OR.(NFELM.EQ.0)) THEN
        CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.FEL'),'UNFORMATTED')
      ELSEIF(FOUND) THEN
        WRITE(NTW,2200)
        WRITE(NOT,*)
        WRITE(NOT,2200)
 2200   FORMAT(
     +  '   ** NOTE: Additional flow elements being added to system.')
        OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +  FORM='UNFORMATTED')
        CALL APPEND(ND1)
      ENDIF
C
C--3.0 DEFINE TEMPORARY ARRAYS; GENERATE ELEMENT DATA; REPORT BANDWIDTH
C
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DEFINR('GENR',MPGENR,NSSPE,1)
      CALL DEFINR('DIFF',MPDIFF,NSSPE,1)
      CALL DEFINR('EFF ',MPEFF,NSSPE,1)

      NESTRT = NFELM+1
      CALL ELGEN(FLOEL0,NENOD,NESTRT,NSNOD,ERR)
```

```
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')

      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      IF(ECHO) WRITE(NTW,2300) MSBAN
      WRITE(NOT,*)
      WRITE(NOT,2300) MSBAN
 2300 FORMAT('    ** NOTE: Current system bandwidth is:', I5)
C
C--4.0 ORDERLY COMPLETION: CLOSE FILE ND1; SKIP TO "END"
C
      CLOSE(ND1)

      IF(MODE.EQ.'INTER') RETURN
  500 IF(EOC) RETURN
        CALL FREE
      GO TO 500

      END

C--------------------------------------------------------------- FLOEL0
      SUBROUTINE FLOEL0(NEL,LN,ERR)
C--SUB:FLOEL0 - CALLS FLOEL1 PASSING ARRAYS

      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      INTEGER LN(2)

      CALL FLOEL1(IA(MPKSEQ),IA(MPEFF),IA(MPDIFF),IA(MPGENR),NEL,LN,ERR)

      RETURN
      END

C--------------------------------------------------------------- FLOEL1
      SUBROUTINE FLOEL1(KSEQ,EFF,DIFF,GENR,NEL,LN,ERR)
C--SUB:FLOEL1 - READS FLOW ELEMENT PROPERTY DATA,
C               UPDATES SYSTEM BANDWIDTH MSBAN,
C               WRITES FLOW ELEMENT DATA TO LOGICAL UNIT ND1, AND
C               REPORTS ELEMENT DATA TO RESULTS OUTPUT FILE

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 EFF(NSSPE), DIFF(NSSPE), GENR(NSSPE),MASSL,LENGTH,FACTOR
      INTEGER KSEQ(NSNOD,NSSPE),LN(2),LS(1),NEL
      CHARACTER TYPE*4,TYPEON*4
      LOGICAL ERR
      SAVE TYPEON

C----VARIABLE--------DESCRIPTION---------------------------------------
C  TYPE*4        : ELEMENT TYPE: 'SIMP' OR 'CNDF'
C  TYPEON*4 : CURRENT ELEMENT TYPE "ON" OR ACTIVE
C  MASSL         : MASS PER UNIT LENGTH - 'CNDF' ELEMENTS.
C  LENGTH        : FLOW PASSAGE LENGTH - 'CNDF' ELEMENTS.
C  FACTOR        : UPWIND FACTOR - 'CNDF' ELEMENTS.
C  ERR      : ERROR FLAG
C---------------------------------------------------------------------

C
C--0.0 INITIALIZATION
C
      NESPE = 1
      NENOD = 2
C
C--1.0 GET ELEMENT TYPE
C
      TYPE = 'SIMP'
      CALL FREEC('T',TYPE,4,1)

C-------------------------------------------UNDEFINED ELEMENTS

      IF((TYPE.NE.'SIMP').AND.(TYPE.NE.'CNDF')) THEN
        ERR = .TRUE.
        CALL ALERT(
     + 'ERROR: Flow element type '//TYPE//' is not available',
     + '$','$')
        GO TO 999
      ENDIF
C
C--2.0 REPORT TABLE HEADER IF NECESSARY
C
      IF((NEL.EQ.NESTRT).OR.(TYPE.NE.TYPEON)) THEN
        TYPEON = TYPE
C------------------------------------------------- SIMPLE ELEMENTS
        IF(TYPEON.EQ.'SIMP') THEN
          IF(ECHO) WRITE(NTW,2020)
          WRITE(NOT,2020)
C---------------------------------------------- CONV-DIFF ELEMENTS
        ELSEIF(TYPEON.EQ.'CNDF') THEN
          IF(ECHO) WRITE(NTW,2022)
          WRITE(NOT,2022)
        ENDIF
      ENDIF

 2020 FORMAT(/,3X,'Num Type    I    J  Spec   Filt.Eff',/)
 2022 FORMAT(/,3x,'Num Type    I    J  M/Length    Length      Upw.
     +Fact     Spec  Disp.Coef.',/)

C
C--3.0 SIMPLE ELEMENTS
C
      IF(TYPE.EQ.'SIMP') THEN

C--------------------------------------------- READ ELEMENT DATA
        CALL ZEROR(EFF,NSSPE,1)
        CALL FREER('E',EFF(1),NSSPE)
        CALL CKRZER('filter efficiency',EFF,NSSPE,'GE',ERR)
        IF(ERR) GO TO 999

C--------------------------------------- UPDATE SYSTEM BANDWIDTH
        DO 30 N=1,NSSPE
          LS(1) = N
   30   CALL ELBAN(KSEQ,NSNOD,NSSPE,LN,NENOD,LS,NESPE,MSBAN)

C--------------------------------------- WRITE ELEM. DATA TO ND1
        WRITE(ND1) TYPE
        WRITE(ND1) LN(1), LN(2), (EFF(N),N=1,NSSPE)

C------------------------------------------ UPDATE ELEMENT COUNT
        NFELM = NEL

C--------------------------------------------- REPORT ELEMENT DATA
        WRITE(NOT,2030) NEL,TYPE, LN(1), LN(2), SID(1),EFF(1)
        IF(NSSPE.GE.2) WRITE(NOT,2032) (SID(N),EFF(N),N=2,NSSPE)
        IF(ECHO) THEN
          WRITE(NTW,2030) NEL,TYPE, LN(1), LN(2), SID(1), EFF(1)
          IF(NSSPE.GE.2) WRITE(NTW,2032) (SID(N),EFF(N),N=2,NSSPE)
        ENDIF
 2030 FORMAT(2X,I4,1X,A4,2I4,2X,A4,1X,G11.4)
 2032 FORMAT((21X,A4,1X,G11.4))

C
C--4.0 CONVECTION-DIFFUSION ELEMENTS
C
      ELSEIF(TYPE.EQ.'CNDF') THEN

C--------------------------------------------- READ ELEMENT DATA

        MASSL = 0.0D0
        CALL FREER('M',MASSL,1)
        CALL CKRZER('mass/length',MASSL,1,'GE',ERR)
        IF(ERR) GO TO 999

        LENGTH = 0.0D0
        CALL FREER('L',LENGTH,1)
        CALL CKRZER('flow passage length',LENGTH,1,'GT',ERR)
        IF(ERR) GO TO 999

        CALL ZEROR(DIFF,NSSPE,1)
        CALL FREER('D',DIFF(1),NSSPE)
        CALL CKRZER('dispersal coef.',DIFF,NSSPE,'GE',ERR)
        IF(ERR) GO TO 999

C       CALL ZEROR(GENR,NSSPE,1)
C       CALL FREER('G',GENR(1),NSSPE)

        FACTOR = 1.0D0
        CALL FREER('F',FACTOR,1)
        CALL CKRRNG('upwind factor',FACTOR,1,0.0D0,'LELE',1.0D0,ERR)
        IF(ERR) GO TO 999
        IF(NODATA) FACTOR = -1.0D0

C--------------------------------------- UPDATE SYSTEM BANDWIDTH
        DO 40 N=1,NSSPE
          LS(1) = N
   40   CALL ELBAN(KSEQ,NSNOD,NSSPE,LN,NENOD,LS,NESPE,MSBAN)

C--------------------------------------- WRITE ELEM. DATA TO ND1
        WRITE(ND1) TYPE
        WRITE(ND1) LN(1),LN(2),MASSL,LENGTH,FACTOR,
     +  (DIFF(N),N=1,NSSPE)

C------------------------------------------ UPDATE ELEMENT COUNT
        NFELM = NEL

C--------------------------------------------- REPORT ELEMENT DATA
        IF(FACTOR.NE.-1.0D0) THEN
          WRITE(NOT,2040) NEL,TYPE,LN(1),LN(2),MASSL,LENGTH,
     +    FACTOR,SID(1),DIFF(1)
        ELSE
          WRITE(NOT,2041) NEL,TYPE,LN(1),LN(2),MASSL,LENGTH,
     +    '  default   ',SID(1),DIFF(1)
        ENDIF
        IF(NSSPE.GE.2)
     +    WRITE(NOT,2042) (SID(N),DIFF(N),N=2,NSSPE)
        IF(ECHO) THEN
          IF(FACTOR.NE.-1.0D0) THEN
            WRITE(NTW,2040) NEL,TYPE,LN(1),LN(2),MASSL,LENGTH,
     +      FACTOR,SID(1),DIFF(1)
          ELSE
            WRITE(NTW,2041) NEL,TYPE,LN(1),LN(2),MASSL,LENGTH,
     +      '  default   ',SID(1),DIFF(1)
        ENDIF
          IF(NSSPE.GE.2)
     +      WRITE(NTW,2042) (SID(N),DIFF(N),N=2,NSSPE)
        ENDIF
 2040 FORMAT(2X,I4,1X,A4,2I4,3(G11.4),1X,A4,1(G11.4))
 2041 FORMAT(2X,I4,1X,A4,2I4,2(G11.4),A11,1X,A4,1(G11.4))
 2042 FORMAT((53X,A4,1(G11.4)))

      ENDIF

      RETURN

  999 CALL ALERT(
     +'WARNING: All flow element data has been deleted.','$','$')
      NFELM = 0
      CLOSE(ND1, STATUS='DELETE')
      RETURN
      END

C================================================================= KINELM
```

```
      SUBROUTINE KINELM
C--SUB:KINELM - COMMAND TO READ & WRITE KIN ELEMENT DATA TO FILE *.KIN
C
C--HELP LIST-------------------------------------------------------------
C
C     .' KINELEM             Kinetics elements:',/,
C     .' K=n1                 n1 = rate coefficient matrix ID number',/,
C     .' n2,n3,...            n2,n3,...= 1st row rate coef. matrix',/,
C     .' n4,n5,...            n4,n5,...= 2nd row rate coef. matrix',/,
C     .' ...                  additional rows as necessary',/,
C     .' K=n1',/,
C     .' ...',/,
C     .' <                    end of rate coef. matrices subgroup',/,
C     .' n6 I=n7 K=n8         n6 = elem. number; n7 = node number',/,
C     .' ...                  n8 = rate coefficient matrix ID number',/,
C     .' END')
C-----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      INTEGER NK
      LOGICAL ERR,FOUND
      EXTERNAL KINEL0

C-----VARIABLE--------DESCRIPTION----------------------------------------
C  ERR        ERROR FLAG
C  FOUND           FILE FOUND FLAG
C  KINEL0          SUB. TO READ & WRITE KIN ELEM DATA
C  NK         RATE COEF. MATRIX ID NUMBER
C  NENOD           NUMBER OF ELEMENT NODES
C  NESPE           NUMBER OF SPECIES PER ELEMENT  (CURRENTLY=NSSPE)
C  NESTRT          FIRST ELEMENT NUMBER
C-----------------------------------------------------------------------
C
C--0.0 INITIALIZATION
C
      ERR = .FALSE.
      NENOD = 1

      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,' ==== KINELEM: KINETICS ELEMENTS')
C
C--1.0 CHECK TO SEE IF SYSTEM NODES & EQUATION NUMBERS ARE DEFINED
C
      CALL CKSYS(1,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 OPEN <filename>.KIN
C
      INQUIRE(FILE=FNAME(1:LFNAME)//'.KIN',EXIST=FOUND)

      IF((.NOT.FOUND).OR.(NKINEL.EQ.0)) THEN
        CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.KIN'),'UNFORMATTED')
      ELSEIF(FOUND) THEN
        WRITE(NTW,2200)
        WRITE(NOT,*)
        WRITE(NOT,2200)
 2200   FORMAT(
     +  '    ** NOTE: Additional kin. elements being added to system.')
        OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.KIN'),STATUS='OLD',
     +  FORM='UNFORMATTED')
        CALL APPEND(ND1)
      ENDIF
C
C--3.0 GET RATE COEFFICIENT ARRAYS
C
      CALL DELETE('TEMP')
      CALL DEFINR('TEMP',MPTEMP,NSSPE,1)

   30 CALL FREE
      IF(EOD) GO TO 40
      NK = 0
      CALL FREEI('K',NK,1)
      CALL CKIRNG('rate coef. matrix ID',NK,1,0,'LTLE',9,ERR)
      IF(ERR) GO TO 999
      CALL DELETE('KIK'//CHAR(NK+48))
      CALL DEFINR('KIK'//CHAR(NK+48),MPKIK(NK),NSSPE,NSSPE)
      CALL GETKIK(IA(MPKIK(NK)),IA(MPTEMP),NK,ERR)
      IF(ERR) GO TO 999
      GO TO 30
C
C--4.0 GENERATE ELEMENT DATA; REPORT BANDWIDTH
C
   40 WRITE(NTW,2400)
      WRITE(NOT,2400)
 2400 FORMAT(/,'  == Kinetics Elements',//,6X,'Elem    Node    KinID')

C-----'TEMP' STORES SPECIES CONNECTIVITY ARRAY, LS(NSSPE), USED BY ELBAN

      CALL DELETE('TEMP')
      CALL DEFINI('TEMP',MPTEMP,NSSPE,1)
      DO 42 N=1,NSSPE
   42 IA(MPTEMP+N-1) = N

      NESTRT = NKINEL+1
      CALL ELGEN(KINEL0,NENOD,NESTRT,NSNOD,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      CALL DELETE('TEMP')
      IF(ECHO) WRITE(NTW,2410) MSBAN
      WRITE(NOT,*)
      WRITE(NOT,2410) MSBAN
 2410 FORMAT('    ** NOTE: Current system bandwidth is:', I5)
C
C--5.0 ORDERLY COMPLETION: CLOSE FILE ND1; SKIP TO "END"
C
      CLOSE(ND1)

      IF(MODE.EQ.'INTER') RETURN
  500 IF(EOC) RETURN
        CALL FREE
      GO TO 500
C
C-- ABORT COMMAND
C
  999 CALL ALERT(
     +'WARNING: All kinetics element data has been deleted.','$','$')
      NKINEL=0
      CLOSE(ND1, STATUS='DELETE')
      DO 900 NK=1,9
  900 CALL DELETE('KIK'//CHAR(NK+48))
      CALL DELETE('TEMP')
      CALL ABORT
      RETURN
      END

C----------------------------------------------------------------- GETKIK
      SUBROUTINE GETKIK(KIK,TEMP,NK,ERR)
C--SUB:GETKIK - READS AND REPORTS KINETICS RATE COEF. ARRAYS

      INCLUDE 'CNTCOM.INC'
      INCLUDE 'IOCOM.INC'

      REAL*8 KIK(NSSPE,NSSPE), TEMP(NSSPE)
      INTEGER NK
      LOGICAL ERR

      IF(ECHO) WRITE(NTW,2000) NK
      WRITE(NOT,2000) NK
 2000 FORMAT(/,'  == Kinetics Rate Coef. Matrix: KinID =',I3)
C
C--1.0 READ [K]
C
      DO 110 I=1,NSSPE
        IF(MODE.EQ.'INTER') WRITE(NTW,2100) I
 2100   FORMAT('    ** Enter terms in row number: ',I4)
        CALL FREE
        IF(EOD) THEN
          CALL ALERT(
     +    'ERROR: Data expected. Data subgroup terminator found.',
     +    '$','$')
          ERR = .TRUE.
          RETURN
        ENDIF
        CALL FREER(' ',TEMP,NSSPE)
        DO 100 J=1,NSSPE
  100   KIK(I,J) = TEMP(J)
  110 CONTINUE
C
C--2.0 REPORT FIVE COLUMNS AT A TIME
C
      DO 200 J1=1,NSSPE,5
        J2 = MIN(NSSPE,J1+4)
        IF(ECHO) WRITE(NTW,2200)  (SID(J),J=J1,J2)
        WRITE(NOT,2200)  (SID(J),J=J1,J2)
 2200   FORMAT(/,12X,5(:3X,A4,6X))

        DO 200 I=1,NSSPE
          IF(ECHO) WRITE(NTW,2210) SID(I),(KIK(I,J),J=J1,J2)
          WRITE(NOT,2210) SID(I),(KIK(I,J),J=J1,J2)
  200 CONTINUE
 2210 FORMAT(6X,A4,2X,5(:G11.3,2X))
      RETURN
      END

C----------------------------------------------------------------- KINEL0
      SUBROUTINE KINEL0(NEL,LN,ERR)
C--SUB:KINEL0 - READS ADDITIONAL KINETICS ELEMENT DATA,
C              WRITE KINETICS ELEMENT DATA TO FILE ND1,
C              UPDATES SYSTEM BANDWIDTH, AND REPORTS ELEMENT DATA

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      INTEGER LN(1), NK

      NENOD = 1
      NESPE = NSSPE
C
C--1.0 GET RATE COEFFICIENT MATRIX ID
C
      CALL FREEI('K',NK,1)
      IF(MPKIK(NK).EQ.0) THEN
        CALL ALERT('ERROR: Rate coefficient matrix not defined',
     +  '$','$')
        ERR = .TRUE.
        RETURN
      ENDIF
C
C--2.0 WRITE DATA TO ND1
C
      WRITE(ND1) LN(1), NK
C
C--3.0 UPDATE SYSTEM BANDWIDTH (NOTE: IA(MPTEMP)=LS(NSSPE) )
C
```

```
      CALL ELBAN(IA(MPKSEQ),NSNOD,NSSPE,LN,NENOD,IA(MPTEMP),NESPE,MSBAN)
C
C--4.0 REPORT DATA
C
      IF(ECHO) WRITE(NTW,2000) NEL, LN(1), NK
      WRITE(NOT,2000) NEL, LN(1), NK
 2000 FORMAT(6X,I4,4X,I4,4X,I4)

      NKINEL = NEL

      RETURN
      END

C=================================================================== FORMF0
      SUBROUTINE FORMF0
C--SUB:FORMF0 - COMMAND TO FORM CONTAM. DISPERSAL MASS TRANS ARRAY [W]
C--HELP LIST-----------------------------------------------------------
C
C     .' FORM-[W] F=cccc         Form [F], cccc = FULL or BAND',/,
C     .' n2,n3,n4 W=n5          n2,n3,n4 = elem: first, last, incr.',/,
C     .' ...                    n5 = element flow rate',/,
C     .' END')
C----------------------------------------------------------------------
C

      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      CHARACTER FORM*4

      ERR = .FALSE.
C
C--0.0 WRITE HEADER AND READ ARRAY FORM
C
      WRITE(NOT,2000)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
 2000 FORMAT(/,
     +' ==== FORM-[W]: FORM MASS TRANSPORT MATRIX [W] ')

      FORM = 'FULL'
      CALL FREEC('F',FORM,4,1)
      IF(FORM.EQ.'FULL') THEN
        IF(ECHO) WRITE(NTW,2002)
        WRITE(NOT,*)
        WRITE(NOT,2002)
 2002   FORMAT('    ** NOTE: [W] being formed in FULL form.')
      ELSEIF(FORM.EQ.'BAND') THEN
        IF(ECHO) WRITE(NTW,2004)
        WRITE(NOT,*)
        WRITE(NOT,2004)
 2004   FORMAT('    ** NOTE: [W] being formed in BAND form.')
      ELSE
        CALL ALERT('ERROR: '//FORM//' not defined.','$','$')
        CALL ABORT
        RETURN
      ENDIF
C
C--1.0 CHECK IF FLOW SYSTEM AND ELEMENT DATA ARE DEFINED
C
      CALL CKSYS(2,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 DEFINE AND INITIALIZE ARRAYS
C
      CALL DELETE('TEMP')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('WE  ')
      CALL DELETE('W   ')
      IF(FORM.EQ.'FULL') CALL DEFINR('W   ',MPF,NSEQ,NSEQ)
      IF(FORM.EQ.'BAND') CALL DEFINR('W   ',MPF,NSEQ,2*MSBAN-1)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL DEFINR('GENR',MPGENR,NSSPE,1)
      CALL DEFINR('DIFF',MPDIFF,NSSPE,1)
      CALL DEFINR('EFF ',MPEFF,NSSPE,1)
      CALL DEFINR('VCD ',MPVCD,NSNOD,1)
      CALL DEFINR('CONT',MPCONT,NSNOD,1)
C
C--3.0 GET ELEMENT FLOW RATES {WE}
C
      CALL ZEROR(IA(MPWE),NFELM,1)
      CALL READWE(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--4.0 FORM [W]
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')

      IF(NKINEL.GT.0) THEN
      OPEN(ND2,FILE=(FNAME(1:LFNAME)//'.KIN'),STATUS='OLD',
     +FORM='UNFORMATTED')
      ENDIF

      CALL FORMF(IA(MPKSEQ),IA(MPF),IA(MPWE),FORM,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      CLOSE(ND1)
      IF(NKINEL.GT.0) CLOSE(ND2)
C
C--5.0 DELETE ARRAYS
C
      CALL DELETE('TEMP')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('WE  ')

      RETURN
      END

C=================================================================== STEADY
      SUBROUTINE STEADY
C--SUB:STEADY - COMMAND TO FORM STEADY PROBLEM [F]{C} = {E} & SOLVE
C               SOLUTION {C} IS WRITTEN OVER {E}
C
C--HELP LIST-----------------------------------------------------------
C
C     .' STEADY                  Steady state solution.',/,
C     .' n1,n2,n3 W=n4            n1,n2,n3 = elem: first, last, incr.',/,
C     .' ...                      n4 = element flow rate',/,
C     .' n5,n6,n7 CG=n8,n9,...   n5,n6,n7 = node: first, last, incr.',/,
C     .' ...                      n8,n9,... = spec. conc. or gen. rate',/,
C     .' END',//,
C----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)
      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CMDCOM.INC'
      INCLUDE 'CNTCOM.INC'

      COMMON /STDCOM/ MPEDAT

      LOGICAL ERR

      ERR = .FALSE.

      WRITE(NOT,2000)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
 2000 FORMAT(/,' ==== STEADY: STEADY STATE SOLUTION')

C
C--1.0 CHECK IF FLOW SYSTEM AND ELEMENT DATA ARE DEFINED & AVAIL
C
      CALL CKSYS(2,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 DEFINE AND INITIALIZEARRAYS
C
      CALL DELETE('EDAT')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('E   ')
      CALL DELETE('WE  ')
      CALL DELETE('F   ')
      CALL DEFINR('F   ',MPF,NSEQ,2*MSBAN-1)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL DEFINR('E   ',MPE,NSEQ,1)
      CALL DEFINR('GENR',MPGENR,NSSPE,1)
      CALL DEFINR('DIFF',MPDIFF,NSSPE,1)
      CALL DEFINR('EFF ',MPEFF,NSSPE,1)
      CALL DEFINR('VCD ',MPVCD,NSNOD,1)
      CALL DEFINR('CONT',MPCONT,NSNOD,1)
      CALL DEFINR('EDAT',MPEDAT,NSSPE,1)
C
C--3.0 GET ELEMENT FLOW RATES {WE}
C
      CALL ZEROR(IA(MPWE),NFELM,1)
      CALL READWE(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--4.0 FORM [F]
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')

      IF(NKINEL.GT.0) THEN
      OPEN(ND2,FILE=(FNAME(1:LFNAME)//'.KIN'),STATUS='OLD',
     +FORM='UNFORMATTED')
      ENDIF

      CALL FORMF(IA(MPKSEQ),IA(MPF),IA(MPWE),'BAND',ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      CLOSE(ND1)
      IF(NKINEL.GT.0) CLOSE(ND2)
```

```
C
C--5.0 FORM {E}
C
      CALL ZEROR(IA(MPE),NSEQ,1)
      CALL FORMEX(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--6.0 MODIFY {E} AND [F] FOR PRESCRIBED CONCENTRATIONS
C
      CALL MODIF(IA(MPKSEQ),IA(MPF),IA(MPE))
C
C--7.0 SOLVE
C
      CALL FACTCA(IA(MPF),NSEQ,MSBAN,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
      CALL SOLVCA(IA(MPF),IA(MPE),NSEQ,MSBAN,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--8.0 REPORT SOLUTION
C
      IF(ECHO) WRITE(NTW,2800)
      WRITE(NOT,2800)
 2800 FORMAT(/,'   == Response: Node Concentrations')

      CALL RPRTEN(IA(MPE),IA(MPKSEQ))
C
C--9.0 DELETE ARRAYS
C
      CALL DELETE('EDAT')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('E   ')
      CALL DELETE('WE  ')
      CALL DELETE('F   ')

      RETURN
      END

C----------------------------------------------------------------- FORMEX
      SUBROUTINE FORMEX(ERR)
C--SUB:FORMEX - READS & REPORTS NODAL CONTAMINANT EXCITATION DATA

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      EXTERNAL EXDAT0

      WRITE(NOT,2100)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2100)
 2100 FORMAT(/,
     +'   == Excitation: Contaminant Concentration or Generation')

      CALL DATGEN(EXDAT0,NSNOD,ERR)

      CALL RPRTEN(IA(MPE),IA(MPKSEQ))

      RETURN
      END

C----------------------------------------------------------------- EXDAT0
      SUBROUTINE EXDAT0(N,ERR)
C--SUB:EXDAT0 - CALLS EXDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      COMMON /STDCOM/ MPEDAT
      LOGICAL ERR

      CALL EXDAT1(IA(MPE),IA(MPKSEQ),IA(MPEDAT),N,ERR)

      RETURN
      END

C----------------------------------------------------------------- EXDAT1
      SUBROUTINE EXDAT1(E,KSEQ,EDAT,N,ERR)
C--SUB:GDAT1 - READS CONTAMINANT EXCITATION DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 E(NSEQ), EDAT(NSSPE)
      INTEGER KSEQ(NSNOD,NSSPE)
      LOGICAL ERR

      CALL ZEROR(EDAT,NSSPE,1)
      CALL FREER('G',EDAT,NSSPE)

      DO 10 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) THEN
          E(NEQ) = EDAT(M)
        ELSE
          WRITE(NTW,2000) N
          WRITE(NOT,*)
          WRITE(NOT,2000) N
 2000     FORMAT(' **** ERROR: Node ',I5,' is not a defined flow node.')
          ERR = .TRUE.
          RETURN
        ENDIF
   10 CONTINUE

      RETURN
      END

C------------------------------------------------------------------ MODIF
      SUBROUTINE MODIF(KSEQ,F,E)
C--SUB:MODIF - MODIFIES [F] AND {E} FOR C-PRESCRIBED DOFS

      INCLUDE 'CNTCOM.INC'

      REAL*8 F(NSEQ,2*MSBAN-1),E(NSEQ)
      INTEGER KSEQ(NSNOD,NSSPE)

      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        NEQ = KSEQ(N,M)
        NNEQ = ABS(NEQ)
        IF(NEQ.LT.0) THEN
          F(NNEQ,MSBAN) = F(NNEQ,MSBAN)*1.0D15
          E(NNEQ) = E(NNEQ)*F(NNEQ,MSBAN)
        ENDIF
   10 CONTINUE
      RETURN
      END

C================================================================= TIMCON
      SUBROUTINE TIMCON
C--SUB:TIMCON - COMMAND TO FORM CONTAM. DISPERSAL EIGENVALUE PROBLEM
C
C              [[V]-1[F] - (1/T)[I]{E} = {0}
C
C
C              WHERE: [V]  = FLOW VOLUMETRIC MASS MATRIX (DIAGONAL)
C                     [F]  = FLOW SYSTEM FLOW MATRIX
C                     {E}  = (RIGHT) EIGENVECTORS
C                      T   = CONTAM. DISPERSAL TIME CONSTANTS
C
C
C                TO EVALUATE TIME CONSTANTS. EIGENVECTORS ARE NOT FOUND.
C--HELP LIST-------------------------------------------------------------
C
C     .' TIMECONS E=n1          Time constant solution, n1 = epsilon',/,
C     .' n2,n3,n4 W=n5            n2,n3,n4 = elem: first, last, incr.',/,
C     .' ...                      n5 = element flow rate',/,
C     .' END')
C-----------------------------------------------------------------------
C
C   MPTC TC(NSEQ) TEMPORARY ARRAY FOR STORAGE OF TIME CONS
C

      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR

      ERR = .FALSE.
C
C--0.0 WRITE HEADER AND READ PRECISION
C
      WRITE(NOT,2000)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
 2000 FORMAT(/,
     +' ==== TIMECONS: TIME CONSTANTS - CONTAMINANT DISPERSAL SYSTEM ')

      EP1 = EP
      CALL FREER('E',EP1,1)
      WRITE(NOT,2010) EP1
      IF(ECHO) WRITE(NTW,2010) EP1
 2010 FORMAT(/'   == Convergence parameter, epsilon: ', G10.3)
C
C--1.0 CHECK IF FLOW SYSTEM AND ELEMENT DATA ARE DEFINED
C
      CALL CKSYS(2,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 DEFINE ARRAYS
C
      CALL DELETE('TEMP')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('TC  ')
      CALL DELETE('VM  ')
      CALL DELETE('WE  ')
      CALL DELETE('F   ')
      CALL DEFINR('F   ',MPF,NSEQ,NSEQ)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL DEFINR('VM  ',MPVM,NSEQ,1)
      CALL DEFINR('TC  ',MPTC,NSEQ,1)
      CALL DEFINR('GENR',MPGENR,NSSPE,1)
      CALL DEFINR('DIFF',MPDIFF,NSSPE,1)
      CALL DEFINR('EFF ',MPEFF,NSSPE,1)
      CALL DEFINR('VCD ',MPVCD,NSNOD,1)
```

```
      CALL DEFINR('CONT',MPCONT,NSNOD,1)
C
C--3.0 GET ELEMENT FLOW RATES {WE}
C
      CALL ZEROR(IA(MPWE),NFELM,1)
      CALL READWE(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--4.0 FORM [F]
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')

      IF(NKINEL.GT.0) THEN
      OPEN(ND2,FILE=(FNAME(1:LFNAME)//'.KIN'),STATUS='OLD',
     +FORM='UNFORMATTED')
      ENDIF

      CALL FORMF(IA(MPKSEQ),IA(MPF),IA(MPWE),'FULL',ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      CLOSE(ND1)
      IF(NKINEL.GT.0) CLOSE(ND2)
C
C--5.0 FORM VOLUMETRIC MASS MATRIX
C
      CALL ZEROR(IA(MPVM),NSEQ,1)
      CALL FORMVM(IA(MPKSEQ),IA(MPV),IA(MPVCD),IA(MPVM))
C
C--6.0 COMPUTE & REPORT NOMINAL TIME CONSTANTS
C
      IF(ECHO) WRITE(NTW,2600)
      WRITE(NOT,2600)
 2600 FORMAT(/,'   == Nominal Time Constants')
      CALL ZEROR(IA(MPTC),NSEQ,1)
      CALL NOMTC(IA(MPKSEQ),IA(MPVM),IA(MPF),IA(MPTC))
      CALL RPRTEN(IA(MPTC),IA(MPKSEQ))
C
C--7.0 PREMULTIPLY [F] BY [V] INVERSE
C
      CALL VINVF(IA(MPF),IA(MPVM),ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--8.0 COMPUTE AND REPORT ACTUAL TIME CONSTANTS & ITERATION INFORMATION
C
      IF(ECHO) WRITE(NTW,2800)
      WRITE(NOT,2800)
 2800 FORMAT(/,'   == Actual Time Constants')
      WRITE(NTW,2810)
 2810 FORMAT(/,'   -- NOTE: Computation of actual time constants',
     +' may take considerable time.')

      NIT = 0
      CALL EIGEN2(IA(MPF),IA(MPF),NSEQ,NIT,EP1)
      CALL ACTTC(IA(MPF),IA(MPTC))
      CALL RPRTNO(IA(MPTC),NSEQ,'Num.')

      IF(ECHO) WRITE(NTW,2820) ABS(NIT)
      WRITE(NOT,2820) ABS(NIT)
 2820 FORMAT(/'      Number of iterations used ...',I5)
      IF((NIT.LT.0).OR.(NIT.EQ.50)) THEN
        CALL ALERT('WARNING: Procedure did not converge.','$','$')
      ENDIF
C
C--9.0 DELETE ARRAYS
C
      CALL DELETE('TEMP')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('TC  ')
      CALL DELETE('VM  ')
      CALL DELETE('WE  ')
      CALL DELETE('F   ')

      RETURN
      END
C------------------------------------------------------------------ NOMTC
      SUBROUTINE NOMTC(KSEQ,VM,F,TC)
C--SUB:NOMTC - FORMS NOMINAL TIME CONSTANTS = VM(I,I)/F(I,I)
C
      INCLUDE 'CNTCOM.INC'

      REAL*8 VM(NSEQ),F(NSEQ,1),TC(NSEQ)
      INTEGER KSEQ(NSNOD,NSSPE)

      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) TC(NEQ) = VM(NEQ)/F(NEQ,NEQ)
   10 CONTINUE

      RETURN
      END
C------------------------------------------------------------------ VINVF
      SUBROUTINE VINVF(F,VM,ERR)
C--SUB: VINVF: EVALUATES [V]-1[F] : CALLED BY TIMCON
```

```
      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 F(NSEQ,1), VM(NSEQ), EPZERO
      LOGICAL ERR
C
C--1.0 FIND MAX VOLUMETRIC MASS TO ESTABLISH RELATIVE MACHINE ZERO
C
      VMAX = 0.0D0
      DO 10 I=1,NSEQ
       IF(VM(I).GT.VMAX) VMAX=VM(I)
   10 CONTINUE
      EPZERO = EP*VMAX
C
C--2.0 EVALUATE PRODUCT [V]-1[F]: ERR IF DIV BY MACHINE-ZERO
C
      DO 20 I=1,NSEQ
       VII = VM(I)
       IF(VII.LE.EPZERO) THEN
         WRITE(NTW,2000) I
         WRITE(NOT,*)
         WRITE(NOT,2000) I
         ERR = .TRUE.
         RETURN
       ENDIF
 2000    FORMAT(
     +' **** ERROR: Volumetric mass less than relative machine zero.',/,
     +'            Equation number: ',I5)
      DO 20 J=1,NSEQ
        F(I,J) = F(I,J)/VII
   20 CONTINUE
      RETURN
      END
C------------------------------------------------------------------ ACTTC
      SUBROUTINE ACTTC(F,TC)
C--SUB:ACTTC -COMPUTES ACTUAL TIME CONSTANTS FROM EIGEN VALUE RESULTS
C
      INCLUDE 'CNTCOM.INC'

      REAL*8 F(NSEQ,1),TC(NSEQ)

      DO 10 N=1,NSEQ
        TC(N) = 1.0D0/F(N,N)
   10 CONTINUE
      RETURN
      END
C==================================================================FLODAT
      SUBROUTINE FLODAT
C--SUB:FLODAT - COMMAND TO READ ELEMENT FLOW DATA & GENERATE STEPWISE
C               TIME HISTORIES OF FLOW DATA AND WRITES TIME HISTORIES
C               IN FORMAT;
C
C          TIME
C                   (WE(I),I=1,NFELM)
C          TIME
C                   (WE(I),I=1,NFELM)
C           ...
C
C                TO FILE <filename>.WDT
C
C               OPTIONALLY EQUAL STEP TIME HISTORIES MAY BE GENERATED
C--HELP LIST-------------------------------------------------------------
C
C     .' FLOWDAT [T=n1,n2,n3]  Generate element flow time histories.',/,
C     .' TIME=n1                  n1 = time',/,
C     .' n1,n2,n3 W=n4            n1,n2,n3 = node: first, last, incr.',/,
C     .' ...                      n4 = element mass flow rate.',/,
C     .' :',/,
C     .' END',//,
C
C-----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)
C
C-- CAL-SAP: DATA & COMMON STORAGE
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

C
C-- FLODAT: DATA & COMMON STORAGE
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S ------------------
C
C  POINTER   VARIABLE            DESCRIPTION
C
C              TIME(3)        : START TIME, ENDTIME, TIMESTEP
C  MPWE      WE(NFELM)        : CURRENT ELEMENT MASS FLOW VALUES
C
C     T I M E   H I S T O R Y   D A T A
C
C     DAT(1) |          *- - -   Time histories of excitation data are
C            |          |        defined as step-wise functions of time
C            |        *-         using arbitrary values or, optionally,
C            |        |          generated intermediate values of
C            |      *-           equal step size.
C            |      |
C     DAT(2) |- - *-
C            |----|-----|-------
C                 TM(2) TM(1)
C
C   MPTDAT   TDAT(2)        : CURRENT ARBITRARY TIME VALUES
C   MPWDT1   WDT1(NFELM)    : ELEM. FLOW DATA AT TDAT(1)
C   MPWDT2   WDT2(NFELM)    : ELEM. FLOW DATA AT TDAT(1)
C-----------------------------------------------------------------------
```

```
      COMMON /FLODT/ MPTDAT,MPWDT1,MPWDT2
      REAL*8 TIME(3)
      LOGICAL ERR

      ERR = .FALSE.
      WRITE(NOT,2000)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
 2000 FORMAT(/,' ==== FLOWDAT: ELEMENT FLOW TIME HISTORY DATA')
C
C--1.0 CHECK TO SEE IF PERTINENT DATA HAS BEEN DEFINED
C
      CALL CKSYS(2,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 GET DATA GENERATION CONTROL DATA
C
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      CALL CKIZER('time step',TIME(3),1,'GE',ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ELSEIF(TIME(3).GT.0.0D0) THEN
        IF(TIME(2).LT.TIME(1)) THEN
          CALL ALERT(
     +    'ERROR: Final time must be greater than initial time.',
     +    '$','$')
          CALL ABORT
          RETURN
        ENDIF

        IF(ECHO) WRITE(NTW,2220)
        WRITE(NOT,2220)
 2220   FORMAT(/,'   == Generation Control Variables')
        IF(ECHO) WRITE(NTW,2230) (TIME(I),I=1,3)
        WRITE(NOT,2230) (TIME(I),I=1,3)
 2230   FORMAT(/,
     .'      Initial time ................... ',G10.3,/,
     .'      Final time ..................... ',G10.3,/,
     .'      Time step increment ............ ',G10.3)
      ENDIF
C
C--3.0 OPEN <filename>.WDT
C
      CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.WDT'),'UNFORMATTED')
C
C--4.0 READ & GENERATE FLOW DATA
C
      WRITE(NOT,2400)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2400)
 2400 FORMAT(/,'   == Element Mass Flow Time History Data')
C
C---4.1 DEFINE & INITIALIZE ARRAYS
C
      CALL DELETE('TDAT')
      CALL DELETE('WDT1')
      CALL DEFINR('WDT1',MPWDT1,NFELM,1)
      CALL DEFINR('TDAT',MPTDAT,1,2)
      CALL ZEROR(IA(MPWDT1),NFELM,1)
      CALL ZEROR(IA(MPTDAT),1,2)
      IF(TIME(3).GT.0.0D0) THEN
        CALL DELETE('WDT2')
        CALL DELETE('WE  ')
        CALL DEFINR('WE  ',MPWE,NFELM,1)
        CALL DEFINR('WDT2',MPWDT2,NFELM,1)
        CALL ZEROR(IA(MPWDT2),NFELM,1)
        CALL ZEROR(IA(MPWE),NFELM,1)
      ENDIF
C
C---4.2 GENERATE VALUES & WRITE TO <filename>.WDT
C
      IF(TIME(3).GT.0.0D0) THEN
        CALL GENWD1(IA(MPWE),IA(MPTDAT),IA(MPWDT1),IA(MPWDT2),TIME,ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ELSE
        CALL GENWD2(IA(MPTDAT),IA(MPWDT1),ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ENDIF

C
C--5.0 DELETE ARRAYS, CLOSE ELEMENT FLOW DATA FILE, SKIP TO "END"
C
      CALL DELETE('WE  ')
      CALL DELETE('WDT2')
      CALL DELETE('WDT1')
      CALL DELETE('TDAT')

      CLOSE(ND1)

      IF(MODE.EQ.'BATCH') THEN
  500   IF(EOC) RETURN
        CALL FREE
        GO TO 500
      ENDIF

      RETURN
      END
C-----------------------------------------------------------------GENWD1
      SUBROUTINE GENWD1(WE,TDAT,WDT1,WDT2,TIME,ERR)
C--SUB: GENWD1 - GENERATES ELEMENT MASS FLOW DATA, AT EQUAL TIME STEP
C                INTERVALS, FROM GIVEN ARBITRARY DISCRETE TIME DATA
C-----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

C
C---- FLOWDAT: DATA & COMMON STORAGE
C
      COMMON /FLODT/ MPTDAT,MPWDT1,MPWDT2
      LOGICAL ERR
C
C---- GENWD1: DATA & COMMON STORAGE
C
      REAL*8 WE(NFELM),TDAT(2),WDT1(NFELM),WDT2(NFELM),TIME(3)
C
C--1.0 GET FIRST TWO TIME HISTORY RECORDS
C
       CALL GETTDT(TDAT)
       IF(EOC) THEN
         CALL ALERT('ERROR: Insufficient data.','$','$')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETWDT(WDT1,WDT2,ERR)
       IF(ERR) RETURN

       CALL GETTDT(TDAT)
       IF(EOC) THEN
         CALL ALERT('ERROR: Insufficient data.','$','$')
         ERR = .TRUE.
         RETURN
       ELSEIF(TDAT(1).LT.TDAT(2)) THEN
         CALL ALERT('ERROR: Time data out of sequence.','$','$')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETWDT(WDT1,WDT2,ERR)
       IF(ERR) RETURN
C
C--2.0 GENERATION TIME LOOP
C
      DO 200 T=TIME(1),TIME(2),TIME(3)
C
C---2.1 UPDATE FLOW TIME HISTORY DATA IF NEEDED
C
   20 IF(T.GT.TDAT(1)) THEN
        CALL GETTDT(TDAT)
        IF(EOC) THEN
          CALL ALERT('ERROR: Insufficient data.','$','$')
          ERR = .TRUE.
          RETURN
        ELSEIF(TDAT(1).LT.TDAT(2)) THEN
          CALL ALERT('ERROR: Time data out of sequence.','$','$')
          ERR = .TRUE.
          RETURN
        ENDIF
        CALL GETWDT(WDT1,WDT2,ERR)
        IF(ERR) RETURN
        GO TO 20
      ENDIF
C
C---2.2 COMPUTE INTERPOLATION FRACTION
C
      XT = (T-TDAT(2))/(TDAT(1)-TDAT(2))
C
C---2.3 COMPUTE {WE(T)}
C
      DO 23 N=1,NFELM
      WE(N) = WDT2(N) + XT*(WDT1(N)-WDT2(N))
   23 CONTINUE
C
C---2.4 WRITE TIME,{WE(T)} TO ND1
C
      WRITE(ND1) T
      WRITE(ND1) (WE(I),I=1,NFELM)

  200 CONTINUE
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
      WRITE(ND1) T

      RETURN
      END

C-----------------------------------------------------------------GETWDT
      SUBROUTINE GETWDT(WDT1,WDT2,ERR)
C--SUB: GETWDT - UPDATES ELEMENT FLOW DATA VALUES
C
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      REAL*8 WDT1(NFELM), WDT2(NFELM)
      EXTERNAL WDAT0
C
C--1.0 UPDATE 'OLD' DATA VALUES; INITIALIZE 'NEW' DATA VALUES
C
      DO 10 N=1,NFELM
        WDT2(N) = WDT1(N)
        WDT1(N) = 0.0D0
   10 CONTINUE
C
C--2.0 READ NEW VALUES
C
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN
```

```
      CALL RPRTNO(WDT1(1),NFELM,'Elem')

      RETURN
      END

C-----------------------------------------------------------------WDAT0
      SUBROUTINE WDAT0(N,ERR)
C--SUB:WDAT0 - CALLS WDAT11 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      COMMON /FLODT/ MPTDAT,MPWDT1,MPWDT2
      LOGICAL ERR

      CALL WDAT1(IA(MPWDT1),NFELM,N)

      RETURN
      END

C-----------------------------------------------------------------WDAT1

      SUBROUTINE WDAT1(WDT1,NFELM,N)
C--SUB:WDAT1 - READS ELEMENT MASS FLOW RATE TIME HISTORY DATA
C
      REAL*8 WDT1(NFELM)

      CALL FREER('W',WDT1(N),1)

      RETURN
      END

C----------------------------------------------------------------GENWD2
      SUBROUTINE GENWD2(TDAT,WDT1,ERR)
C--SUB: GENWD2 - GENERATES ELEMENT MASS FLOW DATA, AT GIVEN TIME STEP
C                INTERVALS, FROM GIVEN DISCRETE TIME DATA
C----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'
C
C---- FLOWDAT: DATA & COMMON STORAGE
C
      COMMON /FLODT/ MPTDAT,MPWDT1,MPWDT2
      LOGICAL ERR
      EXTERNAL WDAT0
C
C---- GENWD2: DATA & COMMON STORAGE
C
      REAL*8 TDAT(2),WDT1(NFELM)

C
C--1.0 GET FIRST TIME HISTORY RECORD ( TDAT(1), WDT1(NFELM) )
C
      CALL GETTDT(TDAT)
      IF(EOC) RETURN
      TDAT(2) = TDAT(1)
      CALL ZEROR(WDT1,NFELM,1)
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN
      CALL RPRTNO(WDT1(1),NFELM,'Elem')
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (WDT1(I),I=1,NFELM)
C
C--2.0 GET ADDITIONAL TIME HISTORY RECORDS
C
   20 CALL GETTDT(TDAT)
      IF(EOC) GO TO 300
      IF(TDAT(1).LT.TDAT(2)) THEN
        CALL ALERT('ERROR: Time data out of sequence.','$','$')
        ERR = .TRUE.
        RETURN
      ENDIF
      TDAT(2) = TDAT(1)
      CALL ZEROR(WDT1,NFELM,1)
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN
      CALL RPRTNO(WDT1(1),NFELM,'Elem')
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (WDT1(I),I=1,NFELM)
      GO TO 20
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
  300 WRITE(ND1) TDAT(1)

      RETURN
      END

C================================================================EXCDAT
      SUBROUTINE EXCDAT
C--SUB:EXCDAT - COMMAND TO READ EXCITATION DATA & GENERATE STEPWISE
C               TIME HISTORIES OF EXCITATION VALUES, E(NSEQ),AND
C               WRITES TIME HISTORIES IN FORMAT;
C
C          TIME
C                 (E(I),I=1,NSEQ)
C          TIME
C                 (E(I),I=1,NSEQ)
C           ...
C
C                TO FILE <filename>.EDT
C--HELP LIST-----------------------------------------------------------
C
C    .' EXCITDAT [T=n1,n2,n3] Generate excitation time histories.',/,
C    .' TIME=n1               n1 = time',/,
C    .' n1,n2,n3 CG=n4,n5,... n1,n2,n3 = node: first, last, incr.',/,
C    .' ...                   n4,n5,... = conc. or gen. rate.',/,
C    .' :',/,
C    .' END',//,
C
C----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

C-- EXCDAT: DATA & COMMON STORAGE -------------------------------------
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S ------------------
C
C  POINTER   VARIABLE             DESCRIPTION
C
C            TIME(3)        : START TIME, ENDTIME, TIMESTEP
C  MPE       E(NSEQ,NSSPE)     : CURRENT EXCITATION VALUES
C
C    T I M E   H I S T O R Y   D A T A
C
C    DAT(1) |           * - - -  Time histories of excitation data are
C           |           |        defined as step-wise functions of time
C           |         *-         using arbitrary values or, optionally,
C           |         |          generated intermediate values of
C           |       *-           equal step size.
C           |       |
C    DAT(2) |- - *-
C           |----|-----|-------
C               TM(2) TM(1)
C
C   MPTDAT   TDAT(2)        : CURRENT ARBITRARY TIME VALUES
C   MPEDT1   EDT1(NSSPE,NSNOD) : EXCITATION DATA AT TDAT(1)
C   MPEDT2   EDT2(NSSPE,NSNOD) : EXCITATION DATA AT TDAT(2)
C----------------------------------------------------------------------
      COMMON /EXCDT/ MPTDAT,MPEDT1,MPEDT2
      REAL*8 TIME(3)
      LOGICAL ERR

      ERR = .FALSE.
      WRITE(NOT,2000)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
 2000 FORMAT(/,' ==== EXCITDAT: EXCITATION TIME HISTORY DATA')

C
C--1.0 CHECK TO SEE IF FLOW SYSTEM HAS BEEN DEFINED
C
      CALL CKSYS(1,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 GET DATA GENERATION CONTROL DATA
C
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      IF(TIME(3).LT.0.0D0) THEN
        CALL ALERT('ERROR: Time step may not be negative.','$','$')
        CALL ABORT
        RETURN
      ELSEIF(TIME(3).GT.0.0D0) THEN
        IF(TIME(2).LT.TIME(1)) THEN
         CALL ALERT(
     +   'ERROR: Final time must be greater than initial time.',
     +   '$','$')
          CALL ABORT
          RETURN
        ENDIF

        IF(ECHO) WRITE(NTW,2220)
        WRITE(NOT,2220)
 2220   FORMAT(/,'   == Generation Control Variables')
        IF(ECHO) WRITE(NTW,2230) (TIME(I),I=1,3)
        WRITE(NOT,2230) (TIME(I),I=1,3)
 2230   FORMAT(/,
     .'      Initial time .................. ',G10.3,/,
     .'      Final time .................... ',G10.3,/,
     .'      Time step increment ........... ',G10.3)
      ENDIF
C
C--3.0 OPEN <filename>.EDT
C
      CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.EDT'),'UNFORMATTED')
C
C--4.0 READ & GENERATE EXCITATION DATA
C
      WRITE(NOT,2400)
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2400)
 2400 FORMAT(/,'   == Nodal Excitation Time History Data')
C
C---4.1 DEFINE & INITIALIZE ARRAYS
C
      CALL DELETE('TDAT')
      CALL DELETE('EDT1')
      CALL DELETE('E   ')
      CALL DEFINR('E   ',MPE,NSEQ,1)
      CALL DEFINR('EDT1',MPEDT1,NSSPE,NSNOD)
      CALL DEFINR('TDAT',MPTDAT,1,2)
      CALL ZEROR(IA(MPE),NSEQ,1)
      CALL ZEROR(IA(MPEDT1),NSSPE,NSNOD)
      CALL ZEROR(IA(MPTDAT),1,2)
      IF(TIME(3).GT.0.0D0) THEN
        CALL DELETE('EDT2')
        CALL DEFINR('EDT2',MPEDT2,NSSPE,NSNOD)
```

```
      CALL ZEROR(IA(MPEDT2),NSSPE,NSNOD)
    ENDIF
C
C---4.2 GENERATE VALUES & WRITE TO <filename>.EDT
C
      IF(TIME(3).GT.0.0D0) THEN
        CALL GENED1(IA(MPKSEQ),IA(MPE),IA(MPTDAT),IA(MPEDT1),IA(MPEDT2),
     +  TIME,ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ELSE
        CALL GENED2(IA(MPKSEQ),IA(MPE),IA(MPTDAT),IA(MPEDT1),ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ENDIF
C
C--5.0 DELETE ARRAYS, CLOSE ELEMENT FLOW DATA FILE, SKIP TO "END"
C
      CALL DELETE('EDT2')
      CALL DELETE('TDAT')
      CALL DELETE('EDT1')
      CALL DELETE('E   ')

      CLOSE(ND1)

      IF(MODE.EQ.'BATCH') THEN
  500   IF(EOC) RETURN
        CALL FREE
        GO TO 500
      ENDIF

      RETURN
      END
C-------------------------------------------------------------------GENED1
      SUBROUTINE GENED1(KSEQ,E,TDAT,EDT1,EDT2,TIME,ERR)
C--SUB: GENED1 - GENERATES EXCITATION DATA, AT EQUAL TIME STEP
C                INTERVALS, FROM GIVEN ARBITRARY TIME DATA
C-------------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
C
C---- GENED1: DATA & COMMON STORAGE
C
      REAL*8 E(NSEQ),TDAT(2),EDT1(NSSPE,NSNOD),EDT2(NSSPE,NSNOD),TIME(3)
      INTEGER KSEQ(NSNOD,NSSPE)
C
C--1.0 GET FIRST TWO TIME HISTORY RECORDS
C
       CALL GETTDT(TDAT)
       IF(EOC) THEN
         CALL ALERT('ERROR: Insufficient data.','$','$')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(KSEQ,E,EDT1,EDT2,ERR)
       IF(ERR) RETURN

       CALL GETTDT(TDAT)
       IF(EOC) THEN
         CALL ALERT('ERROR: Insufficient data.','$','$')
         ERR = .TRUE.
         RETURN
       ELSEIF(TDAT(1).LT.TDAT(2)) THEN
         CALL ALERT('ERROR: Time data out of sequence.','$','$')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(KSEQ,E,EDT1,EDT2,ERR)
       IF(ERR) RETURN
C
C--2.0 GENERATION TIME LOOP
C
      DO 200 T=TIME(1),TIME(2),TIME(3)
C
C---2.1 UPDATE EXCITATION TIME HISTORY DATA IF NEEDED
C
   20 IF(T.GT.TDAT(1)) THEN
       CALL GETTDT(TDAT)
       IF(EOC) THEN
         CALL ALERT('ERROR: Insufficient data.','$','$')
         ERR = .TRUE.
         RETURN
       ELSEIF(TDAT(1).LT.TDAT(2)) THEN
         CALL ALERT('ERROR: Time data out of sequence.','$','$')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(KSEQ,E,EDT1,EDT2,ERR)
       IF(ERR) RETURN
       GO TO 20
      ENDIF
C
C---2.2 COMPUTE INTERPOLATION FRACTION
C
      XT = (T-TDAT(2))/(TDAT(1)-TDAT(2))
C
C---2.3 COMPUTE {E(T)}
C
      DO 23 N=1,NSNOD
      DO 23 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) E(NEQ) = EDT2(M,N) + XT*(EDT1(M,N)-EDT2(M,N))
   23 CONTINUE
C
C---2.4 WRITE TIME,{E(T)} TO ND1
C
      WRITE(ND1) T
      WRITE(ND1) (E(I),I=1,NSEQ)

  200 CONTINUE
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
      WRITE(ND1) T

      RETURN
      END
C-------------------------------------------------------------------GETEDT
      SUBROUTINE GETEDT(KSEQ,E,EDT1,EDT2,ERR)
C--SUB: GETEDT - UPDATES EXCITATION DATA VALUES
C-------------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'
C
C-- GETEDT: DATA & COMMON STORAGE
C
      LOGICAL ERR
      REAL*8 E(NSEQ),EDT1(NSSPE,NSNOD), EDT2(NSSPE,NSNOD)
      INTEGER KSEQ(NSNOD,NSSPE)
      EXTERNAL EDAT0
C
C--1.0 UPDATE 'OLD' DATA VALUES; INITIALIZE 'NEW' DATA VALUES
C
      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        EDT2(M,N) = EDT1(M,N)
        EDT1(M,N) = 0.0D0
   10 CONTINUE
C
C--2.0 READ NEW VALUES
C
      CALL DATGEN(EDAT0,NSNOD,ERR)
      IF(ERR) RETURN
C
C--3.0 REPORT VALUES
C
      DO 30 N=1,NSNOD
      DO 30 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) E(NEQ) = EDT1(M,N)
   30 CONTINUE

      CALL RPRTEN(E(1),IA(MPKSEQ))

      RETURN
      END
C--------------------------------------------------------------------EDAT0
      SUBROUTINE EDAT0(N,ERR)
C--SUB:EDAT0 - CALLS EDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      COMMON /EXCDT/ MPTDAT,MPEDT1,MPEDT2
      LOGICAL ERR

      CALL EDAT1(IA(MPEDT1),N)

      RETURN
      END
C--------------------------------------------------------------------EDAT1

      SUBROUTINE EDAT1(EDT1,N)
C--SUB:EDAT0 - READS EXCITATION TIME HISTORY DATA
C

      INCLUDE 'CNTCOM.INC'

      REAL*8 EDT1(NSSPE,NSNOD)

      CALL FREER('G',EDT1(1,N),NSSPE)

      RETURN
      END
C-------------------------------------------------------------------GENED2
      SUBROUTINE GENED2(KSEQ,E,TDAT,EDT1,ERR)
C--SUB: GENED2 - GENERATES EXCITATION DATA FROM GIVEN TIME DATA
C-------------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      EXTERNAL EDAT0
C
C---- GENED2: DATA & COMMON STORAGE
C
      REAL*8 TDAT(2),EDT1(NSSPE,NSNOD), E(NSEQ)
      INTEGER KSEQ(NSNOD,NSSPE)
```

```
C
C--1.0 GET FIRST TIME HISTORY RECORD ( TDAT(1), EDT1(NSSPE,NSNOD) )
C
      CALL GETTDT(TDAT)
      IF(EOC) RETURN
      TDAT(2) = TDAT(1)
      CALL ZEROR(EDT1,NSSPE,NSNOD)
      CALL DATGEN(EDAT0,NSNOD,ERR)
      IF(ERR) RETURN

      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) E(NEQ) = EDT1(M,N)
   10 CONTINUE

      CALL RPRTEN(E(1),IA(MPKSEQ))

      WRITE(ND1) TDAT(1)
      WRITE(ND1) (E(I),I=1,NSEQ)
C
C--2.0 GET ADDITIONAL TIME HISTORY RECORDS
C
   20 CALL GETTDT(TDAT)
      IF(EOC) GO TO 300
      IF(TDAT(1).LT.TDAT(2)) THEN
        CALL ALERT('ERROR: Time data out of sequence.','$','$')
        ERR = .TRUE.
        RETURN
      ENDIF
      TDAT(2) = TDAT(1)
      CALL ZEROR(EDT1,NSSPE,NSNOD)
      CALL DATGEN(EDAT0,NSEQ,ERR)
      IF(ERR) RETURN

      DO 22 N=1,NSNOD
      DO 22 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) E(NEQ) = EDT1(M,N)
   22 CONTINUE

      CALL RPRTEN(E(1),IA(MPKSEQ))

      WRITE(ND1) TDAT(1)
      WRITE(ND1) (E(I),I=1,NSEQ)
      GO TO 20
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
  300 WRITE(ND1) TDAT(1)

      RETURN
      END

C=================================================================DYNAM
      SUBROUTINE DYNAM
C--SUB:DYNAM - COMMAND TO FORM & SOLVE DYNAMIC PROBLEM
C
C             [F(t)]{C} + [V]d{C}/dt = {E(t)}
C
C           * EXCITATION {E}, {G} AND PRESCRIBED {C}, UPDATED AT
C             DISCRETE TIMES USED TO DEFINE EXCITATION (READ FROM ND1)
C           * FLOW MATRIX, [F], & VOLUMETRIC MASS MATRIX, [V], UPDATED
C             AT DISCRETE TIMES USED TO DEFINED ELEM. FLOW RATES (READ
C             FROM ND2)
C--HELP LIST-----------------------------------------------------------
C     .' DYNAMIC                 Dynamic solution.',/,
C     .' T=n1,n2,n3 A=n4          n1,n2,n3 = init,final,incr; n4 =alpha',/,
C     .' n5,n6,n7 IC=n8           n5,n6,n7 = node: first, last, incr.',/,
C     .' ...                      n8 = nodal initial concentrations',/,
C     .' :',/,
C     .' END',//,
C----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      COMMON /DYNM/ TWDAT,TEDAT,MPCDAT
      LOGICAL ERR, FOUND
      REAL*8 TIME(3), PSCALE
      INTEGER PINT
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S -------------------
C
C     VARIABLE      DESCRIPTION----------------------------------------
C     TIME(3)       START TIME, END TIME, TIME INCREMENT
C     TWDAT         TIME OF NEXT ELEMENT FLOW RATE RECORD
C     TEDAT         TIME OF NEXT EXCITATION RECORD
C     PINT          RESPONSE RESULTS PRINT INTERVAL
C     PSCALE        RESULTS PLOT FILE SCALE FACTOR
C
C     P O I N T E R S   T O   B L A N K   C O M M O N   L O C A T I O N S
C
C     MPFS    FS(NSEQ,2*MSBAN-1): [F*] DYNAM ALG. MATRIX (ASYM-COMPACT)
C     MPC               C(NSEQ)    : CURRENT {C}
C     MPCD             CD(NSEQ)    : CURRENT d{C}/dt
C     MPCDD           CDD(NSEQ)    : CURRENT d/dt(d{C}/dt)
C     MPCDAT        CDAT(NSSPE)    : TEMP. STORAGE OF INITIAL CONDS. DATA
C     MPE               E(NSEQ)    : CURRENT {E}
C     MPID             ID(NID)     : LIST OF INDEPENDENT DOF EQUATION NOS.
C----------------------------------------------------------------------

      ERR = .FALSE.

      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ==== DYNAMIC: DYNAMIC SOLUTION')

C
C--1.0 CHECK IF SYSTEM, ELEMENT, AND EXCITATION DATA ARE DEFINED & AVAIL
C
      CALL CKSYS(3,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

C
C--2.0 GET DYNAMIC SOLUTION CONTROL VARIABLES
C
      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2200)
      WRITE(NOT,2200)
 2200 FORMAT(/'    == Solution Control Variables')

      IF(MODE.EQ.'INTER') CALL PROMPT('DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      IF(TIME(3).LE.0.0D0) THEN
        CALL ALERT('ERROR: Time step must be greater than 0.','$','$')
        CALL ABORT
        RETURN
      ELSEIF(TIME(2).LT.TIME(1)) THEN
        CALL ALERT(
     +  'ERROR: Final time must be greater than initial time.',
     +  '$','$')
        CALL ABORT
        RETURN
      ENDIF

      ALPHA = 0.75D0
      CALL FREER('A',ALPHA,1)
      CALL CKRRNG('alpha',ALPHA,1,0.0D0,'LELE',1.0D0,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      PINT = 1
      CALL FREEI('I',PINT,1)
      CALL CKIZER('results print interval',PINT,1,'GT',ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      PSCALE = 0.0D0
      CALL FREER('S',PSCALE,1)

      IF(ECHO) WRITE(NTW,2250) (TIME(I),I=1,3),ALPHA,PINT
      WRITE(NOT,2250) (TIME(I),I=1,3),ALPHA,PINT
 2250 FORMAT(/,
     .'       Initial time ................... ',G10.3,/,
     .'       Final time ..................... ',G10.3,/,
     .'       Time step increment ............ ',G10.3,/,
     .'       Integration parameter: alpha .... ',G10.3,/,
     .'       Results print interval .......... ',I6)
      IF(PSCALE.NE.0.0D0) THEN
        IF(ECHO) WRITE(NTW,2260) PSCALE
        WRITE(NOT,2260) PSCALE
      ENDIF
 2260 FORMAT('       Results plot-file scale factor .. ',G10.3)

C
C--3.0 DEFINE AND INITIALIZE SYSTEM ARRAYS
C
      CALL DELETE('TEMP')
      CALL COUNTI(IA(MPKSEQ),NID)
      CALL DELETE('ID  ')
      CALL DELETE('FS  ')
      CALL DELETE('CD  ')
      CALL DELETE('CDD ')
      CALL DELETE('CDAT')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('WE  ')
      CALL DELETE('C   ')
      CALL DELETE('E   ')
      CALL DELETE('F   ')
      CALL DELETE('VM  ')
      CALL DEFINR('VM  ',MPVM,NSEQ,1)
      CALL DEFINR('F   ',MPF,NSEQ,2*MSBAN-1)
      CALL DEFINR('E   ',MPE,NSEQ,1)
      CALL DEFINR('C   ',MPC,NSEQ,1)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL DEFINR('GENR',MPGENR,NSSPE,1)
      CALL DEFINR('DIFF',MPDIFF,NSSPE,1)
      CALL DEFINR('EFF ',MPEFF,NSSPE,1)
      CALL DEFINR('VCD ',MPVCD,NSNOD,1)
      CALL DEFINR('CONT',MPCONT,NSNOD,1)
      CALL DEFINR('CDAT',MPCDAT,NSSPE,1)
      CALL DEFINR('CDD ',MPCDD,NSEQ,1)
      CALL DEFINR('CD  ',MPCD,NSEQ,1)
      CALL DEFINR('FS  ',MPFS,NSEQ,2*MSBAN-1)
      CALL DEFINI('ID  ',MPID,NID,1)
C
C--4.0 GET NODAL INITIAL CONCENTRATIONS
C
      CALL ZEROR(IA(MPC),NSEQ,1)
      CALL GETIC(ERR)
```

```
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--5.0 OPEN ELEMENT, FLOW AND EXCITATION DATA FILES, & PLOT FILE
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')

      IF(NKINEL.GT.0) THEN
        OPEN(ND2,FILE=(FNAME(1:LFNAME)//'.KIN'),STATUS='OLD',
     +  FORM='UNFORMATTED')
      ENDIF

      OPEN(ND3,FILE=(FNAME(1:LFNAME)//'.WDT'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND(ND3)
      READ(ND3) TWDAT

      OPEN(ND4,FILE=(FNAME(1:LFNAME)//'.EDT'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND(ND4)
      READ(ND4) TEDAT

      IF(PSCALE.NE.0.0D0) THEN
        CALL NOPEN(NPLT,(FNAME(1:LFNAME)//'.PLT'),'FORMATTED')
      ENDIF
C
C--6.0 FORM ID ARRAY
C
      CALL FORMID(IA(MPKSEQ),IA(MPID),NID)
C
C--7.0 CALL PREDIC TO DO THE WORK
C
      CALL ZEROR(IA(MPCD),NSEQ,1)
      CALL ZEROR(IA(MPCDD),NSEQ,1)
      CALL PREDIC(IA(MPID),IA(MPF),IA(MPFS),IA(MPVM),IA(MPE),IA(MPC),
     +IA(MPCD),IA(MPCDD),TIME,ALPHA,NID,NSEQ,MSBAN,PINT,PSCALE,ERR)

      IF(ERR) CALL ABORT
C
C--8.0 DELETE UNNEEDED ARRAYS & CLOSE FILES
C
      CALL DELETE('TEMP')
      CALL DELETE('ID  ')
      CALL DELETE('FS  ')
      CALL DELETE('CD  ')
      CALL DELETE('CDD ')
      CALL DELETE('CDAT')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
      CALL DELETE('EFF ')
      CALL DELETE('DIFF')
      CALL DELETE('GENR')
      CALL DELETE('WE  ')
      CALL DELETE('C   ')
      CALL DELETE('E   ')
      CALL DELETE('F   ')
      CALL DELETE('VM  ')

      CLOSE(NPLT)
      CLOSE(ND1)
      IF(NKINEL.GT.0) CLOSE(ND2)
      CLOSE(ND3)
      CLOSE(ND4)
C
C--9.0 SKIP TO END-OF-COMMAND DELIMITER 'END'
C
      IF(MODE.EQ.'INTER') RETURN
      IF(MODE.EQ.'BATCH') THEN
 900    IF(EOC) RETURN
        CALL FREE
        GO TO 900
      ENDIF
      END

C------------------------------------------------------------------ GETIC
      SUBROUTINE GETIC(ERR)
C--SUB:GETIC - READS & REPORTS INITIAL CONCENTRATION CONDITIONS DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      EXTERNAL ICDAT0

      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,'   == Initial Conditions: Nodal Concentrations')
      CALL DATGEN(ICDAT0,NSNOD,ERR)
      IF(ERR) RETURN

      CALL RPRTEN(IA(MPC),IA(MPKSEQ))

      RETURN
      END

C-----------------------------------------------------------------ICDAT0
      SUBROUTINE ICDAT0(N,ERR)
C--SUB:ICDAT0 - CALLS ICDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      COMMON /DYNM/ TWDAT,TEDAT,MPCDAT
      REAL*8 TWDAT,TEDAT
      LOGICAL ERR

      CALL ICDAT1(IA(MPC),IA(MPCDAT),IA(MPKSEQ),N,ERR)

      RETURN
      END

C-----------------------------------------------------------------ICDAT1

      SUBROUTINE ICDAT1(C,CDAT,KSEQ,N,ERR)
C--SUB:ICDAT1 - READS INITIAL CONCENTRATION CONDITIONS DATA
C
      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'


      INTEGER KSEQ(NSNOD,NSSPE)
      REAL*8 C(NSEQ),CDAT(NSSPE)
      LOGICAL ERR

      CALL ZEROR(CDAT,NSSPE,1)
      CALL FREER('C',CDAT,NSSPE)
      CALL CKRZER('nodal concentrations',CDAT,NSSPE,'GE',ERR)
      IF(ERR) RETURN

      DO 10 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) THEN
          C(NEQ) = CDAT(M)
        ELSE
          WRITE(NTW,2000) N
          WRITE(NOT,*)
          WRITE(NOT,2000) N
 2000     FORMAT(' **** ERROR: Node ',I5,' is not a defined flow node.')
          ERR = .TRUE.
          RETURN
        ENDIF
   10 CONTINUE

      RETURN
      END

C------------------------------------------------------------------ COUNTI

      SUBROUTINE COUNTI(KSEQ,NID)
C--SUB:COUNTI - COUNTS THE NUMBER OF INDEPENDENT DOF

      INCLUDE 'CNTCOM.INC'

      INTEGER KSEQ(NSNOD,NSSPE)

      NID = 0

      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        IF(KSEQ(N,M).LT.0) NID = NID + 1
   10 CONTINUE
      RETURN
      END

C------------------------------------------------------------------ FORMID

      SUBROUTINE FORMID(KSEQ,ID,NID)
C--SUB:FORMID - FORMS ID; THE LIST OF INDEPENDENT DOF EQUATION NUMBERS

      INCLUDE 'CNTCOM.INC'

      INTEGER KSEQ(NSNOD,NSSPE), ID(NID)

      NN = 0

      DO 10 N=1,NSNOD
      DO 10 M=1,NSSPE
        IF(KSEQ(N,M).LT.0) THEN
          NN = NN + 1
          ID(NN) = ABS(KSEQ(N,M))
        ENDIF
   10 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------PREDIC
      SUBROUTINE PREDIC(ID,K,KS,C,E,T,TD,TDD,TIME,ALPHA,NID,NEQN,MBAN,
     +PINT,PSCALE,ERR)
C-SUB: PREDIC - PREDICTOR-CORRECTOR 1ST O.D.E. EQUATION SOLVER
C              TIME STEP ESTIMATE BASED ON METHOD IN *HEAT*
C              BY R.L.TAYLOR - U.C. BERKELEY
C     SOLVES EQUATION;
C
C         [K(t)]{T} + [C]{dT/dt} = {E(t)}
C
C      WHERE;    [K(t)]    = STORED IN COMPACT ASYMMETRIC BANDED FORM
C        [C]  = DIAGONAL; STORED AS VECTOR
C        {E(t)}    = EXCITATION; DEFINED PIECE-WISE LINEAR
C
C     BASED ON DIFFERENCE APPROXIMATION;
C
C          {T}n+1 = {T}n + (1-a)DT{dT/dt}n + (a)DT{dT/dt}n+1
C
C     WHERE;    a = "alpha", an integration parameter
C                 = 0 corresponds to Forward Difference method
C                 = 1 corresponds to Backward Difference method
C                 = 1/2 corresponds to Crank-Nicholson method (unstable)
C              DT = time step increment
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S -------------------
C
C     VARIABLE      DESCRIPTION----------------------------------------
C-DUMMY
C        ID(NID)       : LIST OF INDEPENDENT DOF EQUATION NUMBERS
```

```
C            (I.E., TDOF EQUATION NUMBERS)
C     K(NEQN,2*MBAN-1) : [K] MATRIX: ASYM-BANDED COMPACT-STORED
C     KS(NEQN,2*MBAN-1) : [K*] = [C| + aDT[K] MATRIX (SCALED FOR NEG ID)
C     C(NEQN)        : CURRENT [C] (ORDERED BY EQTN #)
C     E(NEQN)        : CURRENT {E} (ORDERED BY EQTN #)
C     T(NEQN)        : CURRENT {T} (ORDERED BY EQTN #)
C     TD(NEQN) : CURRENT {dT/dt} (ORDERED BY EQTN #)
C     TDD(NEQN)      : INITIAL (d/dt{dT/dt}) TO EST TIME STEP
C     TIME(3)        : START TIME, END TIME, TIME INCREMENT
C     ALPHA          : INTEGRATION PARAMETER
C          NID              : NUMBER OF INDEPENEDENT DOF (TDOF)
C     NEQN      : NUMBER OF EQUATIONS
C     MBAN      : HALF BANDWIDTH OF SYSTEM
C          PINT             : OUTPUT RESULTS PRINT INTERVAL
C          PSCALE           : RESULTS PLOT-FILE SCALE FACTOR
C     ERR       : ERROR FLAG
C-----------------------------------------------------------------------

      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE 'IOCOM.INC'

C
C---- PREDIC: DATA & COMMON STORAGE
C
      REAL*8 K(NEQN,2*MBAN-1),KS(NEQN,2*MBAN-1),C(NEQN),E(NEQN),T(NEQN),
     +TD(NEQN),TDD(NEQN),TIME(3),ALPHA,PSCALE
      INTEGER PINT, ID(NID)
      LOGICAL ERR, TDOF, KUPDAT, EUPDAT
C
C--1.0 FORM INITIAL [K] & [C]
C
      CALL UPDAT1(K,C,TIME(1),KUPDAT,ERR)
      IF(ERR) RETURN
C
C--2.0 COMPUTE INITIAL TEMPERATURE RATES: {dT(0)/dt} FROM
C
C         [C]{dT(0)/dt} = {E(0)} - [K]{T(0)}
C
C---2.1 GET INITIAL EXCITATION
C
      CALL UPDAT2(E,TIME(1),EUPDAT,ERR)
      IF(ERR) RETURN
C
C---2.2 FORM RHS: {dT/dt}=0 FOR 'T'-DOF, {E}-[K]{T} FOR 'E'-DOF : SOLVE
C
      DO 22 I=1,NEQN
C-----'T'-DOF: SET {dT/dt}=0
      IF(TDOF(I,ID,NID)) THEN
C----- 'T'-DOF: CHECK FOR dT/dt INFINITE
        IF(T(I).NE.E(I)) THEN
          WRITE(NTW,2220) I
          WRITE(NOT,*)
          WRITE(NOT,2220) I
 2220     FORMAT(' **** ERROR: Can not compute for step change in',
     +     ' dependent variable number:',I5)
          ERR = .TRUE.
          RETURN
        ELSE
          TD(I) = 0.0D0
        ENDIF
C---- 'E'-DOF: FORM [E|-[K]{T} WHERE [K] IS IN COMPACT STORAGE
      ELSE
        TEMP = E(I)
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 20 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*T(J)
   20   CONTINUE
C----- SOLVE
        TD(I) = TEMP/C(I)
   22 ENDIF
C
C--3.0 COMPUTE TAYLOR'S TIMESTEP CHECK
C
      IF(ECHO) WRITE(NTW,2300)
      WRITE(NOT,2300)
 2300 FORMAT(/,'   == Time Step Estimate for Initial Conditions')
C
C---3.1 COMPUTE INITIAL RATE OF TEMP RATES
C         FORM AND SOLVE: [C]d{dT/dt}/dt = -[K]{dT/dt}
C
      DO 32 I=1,NEQN
      IF(TDOF(I,ID,NID)) THEN
        TDD(I) = 0.0D0
      ELSE
        TEMP = 0.0D0
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 30 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*TD(J)
   30   CONTINUE
        TDD(I) = TEMP/C(I)
   32 ENDIF
C
C---3.2 COMPUTE NORMS: ||{T(0)}||, ||{dT(0)/dt}||, ||d/dt{dT(0)/dt}||
C
      TN = 0.0D0
      TDN = 0.0D0
      TDDN = 0.0D0
      DO 34 N=1,NEQN
      TN = TN + T(N)**2
      TDN = TDN + TD(N)**2
   34 TDDN = TDDN + TDD(N)**2
      TN = SQRT(TN)
      TDN = SQRT(TDN)
      TDDN = SQRT(TDDN)
C
C---3.3 EVALUATE TAYLORS EXPRESSION FOR TIME STEP ESTIMATE
C
      B = 0.05D0
      IF(TDDN.NE.0.0D0) THEN
        DTEST = (B*TDN + SQRT(B*B*TDN*TDN + 2.0D0*B*TN*TDDN))/TDDN
        IF(ECHO) WRITE(NTW,2320) B*100.0D0, DTEST,TIME(3)
        WRITE(NOT,2320) B*100.0D0, DTEST,TIME(3)
 2320 FORMAT(/'   -- NOTE: Estimated time step to limit error to',
     .' approx.',F5.2,'% is:',G10.3,/
     .'              Specified time step is:',G10.3)
      ELSE
        IF(ECHO) WRITE(NTW,2340)
        WRITE(NOT,2340)
 2340 FORMAT(/'   -- NOTE: Unable to estimate time step to limit',
     .' error for the given system.')
      ENDIF
C
C--4.0 FORM AND FACTOR [K*]
C
      CALL FORMKS(ID,K,KS,C,ALPHA,TIME(3),NID,NEQN,MBAN)
      CALL FACTCA(KS,NEQN,MBAN,ERR)
      IF(ERR) RETURN
C
C--5.0 TIME STEP THRU SOLUTION
C
      ADT = ALPHA*TIME(3)
      DTA = (1.0D0 - ALPHA)*TIME(3)
      ISTEP = 0

      IF(PSCALE.NE.0.0D0) THEN
        WRITE(NPLT,2500)'DYNAMIC SOLUTION RESULTS'
 2500   FORMAT(1X,A)
        WRITE(NPLT,2502) 'TIME', (CHAR(9),'EQN-',I,I=1,NEQN)
 2502   FORMAT(1X,A4,1000(A1,A4,I4))
      ENDIF

      DO 500 TM=TIME(1)+TIME(3),TIME(2),TIME(3)
      ISTEP = ISTEP + 1
C
C---5.1 UPDATE [K], FORM AND FACTOR [K*]
C
      CALL UPDAT1(K,C,TM,KUPDAT,ERR)
      IF(ERR) RETURN
      IF(KUPDAT) THEN
        CALL FORMKS(ID,K,KS,C,ALPHA,TIME(3),NID,NEQN,MBAN)
        CALL FACTCA(KS,NEQN,MBAN,ERR)
        IF(ERR) RETURN
      ENDIF
C
C---5.2 FORM {E}
C
      CALL UPDAT2(E,TM,EUPDAT,ERR)
      IF(ERR) RETURN
C
C---5.3 PREDICT T: {T} = {T} + (1-a)DT{dT/dt}
C
      DO 51 N=1,NEQN
      IF(TDOF(N,ID,NID)) THEN
        T(N) = E(N)
      ELSE
        T(N) = T(N) + DTA*TD(N)
      ENDIF
   51 CONTINUE
C
C---5.4 FORM RHS:{E}-[K]{T} FOR FLUX DOF, {dT/dt}*DIAG[K*] FOR TEMP DOF
C
      CALL RHS(ID,T,TD,E,K,KS,NID,NEQN,MBAN)
C
C---5.5 SOLVE FOR {dT/dt}
C
      CALL SOLVCA(KS,TD,NEQN,MBAN,ERR)
      IF(ERR) RETURN
C
C---5.6 CORRECT T: {T} = {T} + aDT{dT/dt}
C
      DO 55 N=1,NEQN
      IF(TDOF(N,ID,NID)) THEN
        T(N) = E(N)
      ELSE
        T(N) = T(N) + ADT*TD(N)
      ENDIF
   55 CONTINUE
C
C---5.7 REPORT RESULTS
C
      IF(MOD(ISTEP,PINT).EQ.0) THEN
        IF(ECHO) WRITE(NTW,2570) TM
        WRITE(NOT,2570) TM
 2570   FORMAT(/,'   == Response ',48(1H=),' Time: ',G10.3)
        CALL RESULT(T)

C-------WRITE TO FILE <filename>.PLT for plotting
        IF(PSCALE.NE.0.0D0) THEN
          WRITE(NPLT,2572) TM, (CHAR(9),T(I)*PSCALE,I=1,NEQN)
 2572     FORMAT(F10.3,(1000(A1,E10.4)))
        ENDIF

      ENDIF

  500 CONTINUE
      RETURN
      END

C----------------------------------------------------------------UPDAT1
      SUBROUTINE UPDAT1(K,C,TM,KUPDAT,ERR)
C--SUB:UPDAT1 - UPDATES [K]=[F] & [C]=[V]
C                IF ELEMENT MASS FLOW RATES CHANGE

      COMMON MTOT,NP,IA(1)
```

```
      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      COMMON /DYNM/ TWDAT,TEDAT,MPCDAT
      REAL*8 K(NSEQ,2*MSBAN-1), C(NSEQ),TM, TWDAT, TEDAT
      LOGICAL ERR, KUPDAT
C
C--1.0 UPDATE ELEMENT FLOW RATES IF(TM.GE.TWDAT)
C
      CALL UPDAT(ND3,TM,TWDAT,IA(MPWE),NFELM,KUPDAT,ERR)
      IF(KUPDAT) THEN
        IF(ECHO) WRITE(NTW,2000) TM
        WRITE(NOT,2000) TM
 2000   FORMAT(/,'    == Element Flow Rate Update ',32(1H-),
     + ' Time: ',G10.3)

        CALL RPRTNO(IA(MPWE),NFELM,'Elem')

        CALL FORMF(IA(MPKSEQ),K,IA(MPWE),'BAND',ERR)

        CALL FORMVM(IA(MPKSEQ),IA(MPV),IA(MPVCD),C)
      ENDIF
      RETURN
      END
C-----------------------------------------------------------------UPDAT2
      SUBROUTINE UPDAT2(E,TM,EUPDAT,ERR)
C--SUB:UPDAT2 - UPDATES {E}={G} IF EXCITATION CHANGES

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      COMMON /DYNM/ TWDAT,TEDAT,MPCDAT
      REAL*8 E(NSEQ), TM, TWDAT, TEDAT
      LOGICAL ERR, EUPDAT

      CALL UPDAT(ND4,TM,TEDAT,E,NSEQ,EUPDAT,ERR)
      IF(EUPDAT) THEN
        IF(ECHO) WRITE(NTW,2000) TM
        WRITE(NOT,2000) TM
 2000   FORMAT(/,'    == Excitation Update ',39(1H-),' Time: ',G10.3)
        CALL RPRTEN(E,IA(MPKSEQ))
      ENDIF
      RETURN
      END

C------------------------------------------------------------------UPDAT
      SUBROUTINE UPDAT(LUN,T,TD,D,ND,UPDATE,ERR)
C--SUB: UPDAT
C       SEARCHES A SEQUENTIAL DATA RECORD, ON UNIT LUN, OF THE FORM;
C           TD
C           (D(I),I=1,ND)
C           TD
C           (D(I),I=1,ND)
C           ...
C       TO UPDATE DATA VALUES TO CURRENT TIME, "T". IF DATA VALUES ARE
C       UPDATED LOGICAL "UPDATE" IS SET TO TRUE.
C
C         TD          : DISCRETE TIME VALUE
C                     : UPDATED TO NEXT VALUE
C         D(I)        : CORRESPONDING DISCRETE DATA VALUES
C
C       UPDAT MUST BE "PRIMED" BY READING FIRST TD VALUE TO MEMORY
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      REAL*8 D(ND),T,TD
      LOGICAL ERR, UPDATE

      UPDATE = .FALSE.
   10 IF(T.GE.TD) THEN
C----UPDATE DISCRETE DATA VALUES
        READ(LUN, ERR=800, END=900) (D(I),I=1,ND)
        IF(ERR) RETURN
        UPDATE = .TRUE.
C----GET NEXT DISCRETE TIME
        READ(LUN, ERR=800, END=900) TD
        IF(ERR) RETURN
        GO TO 10
      ELSE
        RETURN
      ENDIF

  800 ERR = .TRUE.
      CALL ALERT('ERROR: Time history data file read error.','$','$')
      RETURN

  900 ERR = .TRUE.
      CALL ALERT(
     +'ERROR: EOF encountered on time history data file.',
     +'Insufficient time history data.','$')

      RETURN
      END
C-----------------------------------------------------------------FORMKS
      SUBROUTINE FORMKS(ID,K,KS,C,ALPHA,DT,NID,NEQN,MBAN)
C--SUB:FORMKS - FORMS;
C                     [K*] = [C] + aDT[K]
C
C               SCALES [K*] = [K*]*1.0D15 FOR 'T'-DOF
C-----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      REAL*8 K(NEQN,2*MBAN-1),KS(NEQN,2*MBAN-1),C(NEQN)
      INTEGER ID(NID)
      LOGICAL TDOF

      ADT = ALPHA*DT
      DO 10 N=1,NEQN
      DO 10 M=1,2*MBAN-1
   10 KS(N,M) =ADT*K(N,M)

      DO 20 N=1,NEQN
   20 KS(N,MBAN) = KS(N,MBAN) + C(N)

      DO 30 N=1,NEQN
   30 IF(TDOF(N,ID,NID)) KS(N,MBAN) = KS(N,MBAN)*1.0D15

      RETURN
      END
C--------------------------------------------------------------------RHS
      SUBROUTINE RHS(ID,T,TD,E,K,KS,NID,NEQN,MBAN)
C-SUB:RHS - FORMS RHS OF [K*]{dT/dt} = {E*}
C
C             {E*(t)} = {E(t)}-[K]{T(t)}              ; FOR 'E'-DOF
C             {E*(t)} = {dT(t)/dt}*DIAG OF [K*]       ; FOR 'T'-DOF
C
C
C            {E*} IS WRITTEN OVER {TD}
C            [K] & [K*] ARE AYSM-BANDED COMPACT STORED
C-----------------------------------------------------------------------

      IMPLICIT REAL*8 (A-H,O-Z)

      REAL*8 T(NEQN),TD(NEQN),E(NEQN),K(NEQN,2*MBAN-1),
     +KS(NEQN,2*MBAN-1)
      INTEGER ID(NID)
      LOGICAL TDOF

      DO 20 I=1,NEQN
C---- SCALE BY DIAGONAL FOR TEMP PRESCRIBED NODES
      IF(TDOF(I,ID,NID)) THEN
        TD(I) = TD(I)*KS(I,MBAN)
C---- FORM {E}-[K]{T} WHERE [K] IS IN COMPACT STORAGE
      ELSE
        TEMP = E(I)
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 10 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*T(J)
   10   CONTINUE
        TD(I) = TEMP
      ENDIF
   20 CONTINUE
      RETURN
      END

C------------------------------------------------------------------- TDOF
      FUNCTION TDOF(N,ID,NID)
C-FUN:TDOF - DETERMINES IF EQUATION NUMBER N IS A TEMPERATURE DOF
      LOGICAL TDOF
      INTEGER ID(NID)
      TDOF = .FALSE.
      DO 10 NN=1,NID
       IF((ID(NN).EQ.N)) THEN
         TDOF = .TRUE.
         RETURN
       ENDIF
   10 CONTINUE
      RETURN
      END

C----------------------------------------------------------------- RESULT
      SUBROUTINE RESULT(T)
C--SUB:RESULT - REPORTS RESPONSE RESULT VECTOR {T}

      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'

      REAL*8 T(NSEQ)

      CALL RPRTEN(T,IA(MPKSEQ))

      RETURN
      END

C================================================================== RESET
      SUBROUTINE RESET
C--SUB:RESET - COMMAND TO RESET CONTAM BY RE-INITIALIZING POINTERS AND
C              COUNTERS AND DELETES ARRAYS LEFT BY CONTAM IN BLANK COMMON
C--HELP LIST------------------------------------------------------------
C      .' RESET                    Reset CONTAM for new problem.'
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      LOGICAL FOUND
      CHARACTER BINEXT(4)*3
      INTEGER NBIN

      DATA BINEXT/'FEL','KIN','WDT','EDT'/, NBIN/4/
C
C--1.0 RE-INITIALIZE CONTAM CONTROL VARIABLES & DELETE CONTAM ARRAYS
C
      CALL INITCN
      CALL DELETE('G   ')
      CALL DELETE('VM  ')
      CALL DELETE('C   ')
      CALL DELETE('F   ')
      CALL DELETE('WE  ')
      CALL DELETE('V   ')
      CALL DELETE('KSEQ')
```

```
      DO 100 NK=1,9
  100 CALL DELETE('KIK'//CHAR(NK+48))
      CALL DELETE('TEMP')
      CALL DELETE('CONT')
      CALL DELETE('VCD ')
C
C--2.0 DELETE CONTAM BINARY FILES
C
      DO 20 N=1,NBIN
      INQUIRE(FILE=FNAME(1:LFNAME)//BINEXT(N),EXIST=FOUND)
      IF(FOUND) THEN
        OPEN(ND1,FILE=FNAME,STATUS='OLD',FORM='UNFORMATTED')
        WRITE(ND1) ND1
      ENDIF
   20 CLOSE(ND1,STATUS='DELETE')

      WRITE(NTW,2000)
      WRITE(NOT,*)
      WRITE(NOT,2000)
 2000 FORMAT(' **** CONTAM reset for new problem.')

      RETURN
      END

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C                                                                     C
C                 C O N T A M   U T I L I T I E S                     C
C                                                                     C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C

C----------------------------------------------------------------- CKSYS
      SUBROUTINE CKSYS(NOPT,ERR)
C--SUB: CKSYS - CHECKS IF PERTINENT DATA IS DEFINED, UPDATING POINTERS
C
C     NOPT = 1  CHECK FOR FLOSYS DATA
C     NOPT = 2  CHECK FOR FLOSYS, FLOELM, AND KINELEM DATA
C     NOPT = 3  CHECK FOR FLOSYS, FLOELM, KINELEM, FLODAT, AND EXCDAT DATA
C
C----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR, FOUND
      INTEGER NR,NC,NOPT
C
C-- 1.0 FLOWSYS DATA VERIFICATION
C
      IF(NSNOD.EQ.0) THEN
        CALL ALERT('ERROR: Number of flow system nodes = 0.',
     +  'FLOWSYS command must be executed.','$')
        ERR = .TRUE.
        RETURN
      ENDIF

      CALL LOCATE('KSEQ',MPKSEQ,NR,NC)
      IF(MPKSEQ.EQ.0) THEN
        CALL ALERT(
     +  'ERROR: System equation number array "KSEQ" not found.',
     +  'FLOWSYS command must be executed.','$')
        ERR = .TRUE.
        RETURN
      ENDIF

      CALL LOCATE('V   ',MPV,NR,NC)
      IF(MPV.EQ.0) THEN
        CALL ALERT('ERROR: Nodal volumetric mass array "V" not found.',
     +  'FLOWSYS command must be executed.','$')
        ERR = .TRUE.
        RETURN
      ENDIF

      IF(NOPT.EQ.1) RETURN
C
C-- 2.0 FLOWELEM & KINELEM DATA VERIFICATION
C
      IF((NFELM.EQ.0).AND.(NKINEL.EQ.0)) THEN
        CALL ALERT(
     +  'ERROR: Number of flow & kinetics elements both = 0.',
     +  'FLOWELEM &/or KINELEM must be executed.','$')
        ERR = .TRUE.
        RETURN

      ELSEIF(NFELM.EQ.0) THEN
        CALL ALERT('ERROR: Number of flow elements = 0.',
     +  'FLOWELEM must be executed.','$')
        ERR = .TRUE.
        RETURN

      ELSEIF(NKINEL.EQ.0) THEN
        IF(ECHO) WRITE(NTW,2210)
        WRITE(NOT,*)
        WRITE(NOT,2210)
 2210   FORMAT(
     +  '    ** NOTE: Number of kinetics elements = 0.')
      ENDIF

      INQUIRE(FILE=(FNAME(1:LFNAME)//'.FEL'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        CALL ALERT(
     +  'ERROR: Data file '//FNAME(1:LFNAME)//'.FEL not found.',
     +  'FLOWELEM command must be executed.','$')
        ERR = .TRUE.
        RETURN
      ENDIF

      IF(NKINEL.GT.0) THEN
        INQUIRE(FILE=(FNAME(1:LFNAME)//'.KIN'),EXIST=FOUND)
        IF(.NOT.FOUND) THEN
          CALL ALERT(
     +    'ERROR: Data file '//FNAME(1:LFNAME)//'.KIN not found.',
     +    'KINELEM command must be executed.','$')
          ERR = .TRUE.
          RETURN
        ENDIF
        ERR = .TRUE.
        DO 200 NK=1,9
          CALL LOCATE('KIK'//CHAR(48+NK), MPKIK(NK), NR,NC)
  200   IF(MPKIK(NK).NE.0) ERR = .FALSE.
        IF(ERR) CALL ALERT(
     +  'ERROR: Kinetics rate coefficent arrays not found.',
     +  'KINELEM command must be executed.','$')
        RETURN
      ENDIF

      IF(NOPT.EQ.2) RETURN

C
C-- 3.0 FLOWDAT DATA VERIFICATION
C
      INQUIRE(FILE=(FNAME(1:LFNAME)//'.WDT'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        CALL ALERT(
     +  'ERROR: Data file '//FNAME(1:LFNAME)//'.WDT not found.',
     +  'FLOWDAT command must be executed.','$')
       ERR = .TRUE.
       RETURN
      ENDIF
C
C-- 4.0 EXCITDAT DATA VERIFICATION
C
      INQUIRE(FILE=(FNAME(1:LFNAME)//'.EDT'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        CALL ALERT(
     +  'ERROR: Data file '//FNAME(1:LFNAME)//'.EDT not found.',
     +  'EXCITDAT command must be executed.','$')
       ERR = .TRUE.
       RETURN
      ENDIF

      IF(NOPT.EQ.3) RETURN

      RETURN
      END

C----------------------------------------------------------------- GETIDS
      SUBROUTINE GETIDS(IDS,NIDS,LABEL)
C--SUB: GETIDS - GETS CHAR*4 IDS FROM COMMAND/DATA LINE
C                SETS ID= NUMBER FOR BLANK IDS

      INCLUDE 'IOCOM.INC'

      CHARACTER IDS(NIDS)*4, LABEL*(*), HEADER*(*)
      PARAMETER (HEADER='Num.  ID')

      CALL FREEC('D',IDS(1),4,NIDS)

      DO 10 N=1,NIDS
      IF(IDS(N).EQ.'    ') THEN
        NCENT = N/100
        NTENS = (N - NCENT*100)/10
        NONES = N - NCENT*100 - NTENS*10
        IDS(N) = ' '//CHAR(NCENT+48)//CHAR(NTENS+48)//CHAR(NONES+48)
   10 ENDIF

      NCOLS = MIN(NIDS,5)
      IF(ECHO) THEN
        WRITE(NTW,2000) LABEL, (HEADER,N=1,NCOLS)
        WRITE(NTW,2010) (I,IDS(I),I=1,NIDS)
      ENDIF
      WRITE(NOT,2000) LABEL, (HEADER,N=1,NCOLS)
      WRITE(NOT,2010) (I,IDS(I),I=1,NIDS)
 2000 FORMAT(/,'  == ',(A),//,8X,5(:(A),3X))
 2010 FORMAT((8X,5(:I3,2X,A4,2X)))

      RETURN
      END

C----------------------------------------------------------------- READWE
      SUBROUTINE READWE(ERR)
C--SUB:READWE - READS & REPORTS ELEMENT TOTAL MASS FLOW RATE DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      LOGICAL ERR
      EXTERNAL WEDAT0

      IF(ECHO.OR.(MODE.EQ.'INTER')) WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,'  == Element Mass Flow Rates')
      CALL DATGEN(WEDAT0,NFELM,ERR)
      IF(ERR) RETURN

      CALL RPRTNO(IA(MPWE),NFELM,'Elem')

      RETURN
      END

C----------------------------------------------------------------- WEDAT0
      SUBROUTINE WEDAT0(N,ERR)
C--SUB:WEDAT0 - CALLS WEDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE 'CNTCOM.INC'
```

```
      LOGICAL ERR

      CALL WEDAT1(IA(MPWE),NFELM,N)

      RETURN
      END

C----------------------------------------------------------------- WEDAT1

      SUBROUTINE WEDAT1(WE,NFELM,N)
C--SUB:WEDAT0 - READS ELEMENT MASS FLOW RATE DATA
C
      REAL*8 WE(NFELM)

      CALL FREER('W',WE(N),1)

      RETURN
      END

C----------------------------------------------------------------- DATGEN
      SUBROUTINE DATGEN(DATA0,MAXNO,ERR)
C--SUB:DATGEN - READS AND GENERATES DATA BY INCREMENTING RULE;
C                  n1,n2,n3 = FIRST #, LAST #, INCREMENT
C               GIVEN DATA LINE OF FORM
C                  n1,n2,n3  D1=n4,n5,... D2=n6,n7,... etc.
C               CALLS SUBROUTINE "DATA0" TO READ DATA (D1, D2, etc.)
C               RETURNS WHEN DATA LINE IS BLANK, IS ":", OR IS "END"
C               CHECKS ALL GENERATED NUMBERS .LE. MAXNO FOR MAXNO.GT.0
C
      INCLUDE 'IOCOM.INC'

      LOGICAL ERR, FIRSTL
      INTEGER IJK(3)
      EXTERNAL DATA0

      ERR = .FALSE.
      FIRSTL = .TRUE.
C
C--1.0 GET LINE OF DATA
C
  100 IF(MODE.EQ.'INTER') CALL PROMPT('DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
C
C--2.0 CHECK FOR "END"
C
      IF(EOC) THEN
        IF(FIRSTL) THEN
          CALL ALERT('ERROR: Data expected; "END" found.','$','$')
          ERR = .TRUE.
          RETURN
        ELSE
          RETURN
        ENDIF
      ENDIF
C
C--3.0 GET INCREMENTING RULE; RETURN IF IJK(1).EQ.0
C
      IJK(1) = 0
      CALL FREEI(' ',IJK(1),3)
      IF(IJK(1).EQ.0) RETURN
      IF(IJK(2).EQ.0) IJK(2)=IJK(1)
      IF(IJK(3).EQ.0) IJK(3)=1
      DO 300 N=IJK(1), IJK(2), IJK(3)
        IF(MAXNO.GT.0)
     +  CALL CKIRNG('this index',N,1,0,'LTLE',MAXNO,ERR)
        IF(ERR) RETURN
        CALL DATA0(N,ERR)
        IF(ERR) RETURN
  300 CONTINUE

      FIRSTL = .FALSE.
      GO TO 100

      END

C------------------------------------------------------------------ELGEN
      SUBROUTINE ELGEN(ELEM0,NENOD,NESTRT,NSNOD,ERR)
C--SUB:ELEMIN - READS ELEMENT NUMBER, CONNECTIVITY, & GENERATION DATA
C               GENERATES MISSING ELEMENT DATA
C               CALLS "ELEM0" TO READ & WRITE ELEMENT PROPERTY DATA
C               AND TO UPDATE SYSTEM BANDWIDTH MSBAN
C               RETURNS WHEN DATA LINE IS BLANK, IS ":", OR IS "END"
C               CHECKS ALL GENERATED NODE NUMBERS .LE. NSNOD
C
C               ** CURRENTLY LIMITED TO FOUR-NODE ELEMENTS OR LESS **
C
C-----VARIABLE--------DESCRIPTION---------------------------------------
C--INPUT
C  ELEM0         PROCEDURE NAME TO READ & WRITE ELEM. PROP. DATA
C  NENOD         NUMBER OF ELEMENT NODES
C  NESTRT        STARTING ELEMENT NUMBER
C  NSNOD         NUMBER OF SYSTEM NODES
C--OUTPUT
C  ERR       ERROR FLAG
C--LOCAL
C  LNNEW,LNOLD  ELEMENT LOCATION/NODAL-CONNECTIVITY DATA
C  NEOLD,NENEW  ELEMENT NUMBERS: OLD AND NEW
C  INCR      GENERATION INCREMENT
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      LOGICAL ERR
      INTEGER LNNEW(4),LNOLD(4)
      EXTERNAL ELEM0
C
C--1.0 GET FIRST LINE OF ELEMENT DATA
C
      INCR = 0
      IF(MODE.EQ.'INTER') CALL PROMPT('DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)

C---- CHECK FOR "END"
      IF(EOC) RETURN

C---- CHECK FIRST ELEMENT NUMBER
      NEOLD = 0
      CALL FREEI(' ',NEOLD,1)
      IF(NEOLD.EQ.0) RETURN
      IF(NEOLD.NE.NESTRT) THEN
        ERR = .TRUE.
        WRITE(NTW,2100)
        WRITE(NOT,*)
        WRITE(NOT,2100)
        RETURN
      ENDIF
 2100 FORMAT(
     +' **** ERROR: Initial element number should be...',I4,/,
     +'             Given initial element number is....',I4)

C---- PROCESS FIRST ELEMENT
      CALL FREEI('I',LNOLD(1),NENOD)
      CALL CKIRNG('a node number',LNOLD,NENOD,0,'LTLE',NSNOD,ERR)
      IF(ERR) RETURN
      CALL ELEM0(NEOLD,LNOLD,ERR)
      IF(ERR) RETURN
C
C--2.0 GET NEXT LINE OF ELEMENT DATA
C
   20 IF(MODE.EQ.'INTER') CALL PROMPT('DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)

C---- CHECK FOR "END"
      IF(EOC) RETURN

C---- GET NEW ELEMENT CONNECTIVITY & GENERATION INFORMATION
      NENEW = 0
      CALL FREEI(' ',NENEW,1)
      IF(NENEW.EQ.0) RETURN
      CALL FREEI('I',LNNEW(1),NENOD)
      CALL FREEI('N',INCR,1)
      IF(INCR.EQ.0) INCR=1

C---- CHECK NUMERICAL ORDER
      IF(NENEW.LE.NEOLD) THEN
        WRITE(NTW,2200) NENEW
        WRITE(NOT,*)
        WRITE(NOT,2200) NENEW
        ERR=.TRUE.
        RETURN
      ENDIF
 2200 FORMAT(' **** ERROR: Element number ',I5,' is out of order.')

C---- GENERATE MISSING ELEMENTS
      IF(NENEW.GT.NEOLD+1) THEN
        DO 24 N=NEOLD+1,NENEW-1,1
         DO 22 I=1,NENOD
   22    LNOLD(I) = LNOLD(I) + INCR
        CALL CKIRNG('a node number',LNOLD,NENOD,0,'LTLE',NSNOD,ERR)
        IF(ERR) RETURN
        CALL ELEM0(N,LNOLD,ERR)
        IF(ERR) RETURN
   24   CONTINUE
      ENDIF

C---- PROCESS NEW ELEMENT
      NEOLD = NENEW
      DO 26 I=1,NENOD
   26 LNOLD(I) = LNNEW(I)
      CALL CKIRNG('a node number',LNOLD,NENOD,0,'LTLE',NSNOD,ERR)
      IF(ERR) RETURN
      CALL ELEM0(NEOLD,LNOLD,ERR)
      IF(ERR) RETURN

      GO TO 20

      END

C------------------------------------------------------------------ ELBAN

      SUBROUTINE ELBAN(KSEQ,NSNOD,NSSPE,LN,NENOD,LS,NESPE,MSBAN)
C--SUB:ELBAN - COMPUTES ELEMENT BANWIDTH & UPDATES SYSTEM BANDWIDTH

      DIMENSION KSEQ(NSNOD,NSSPE),LN(NENOD),LS(NESPE)

C-----VARIABLE--------DESCRIPTION---------------------------------------
C--INPUT
C  LN(NENOD)     ELEMENT NODAL CONNECTIVITY
C            (SYS. NODE-# CORRESPONDING TO ELEM. NODE-#)
C  NENOD         NUMBER OF ELEM. NODES
C  LS(NESPE)     ELEMENT SPECIES    CONNECTIVITY
C            (SYS. SPECIE-# CORRESPONDING TO ELEM. SPECIE-#)
C  NESPE         NUMBER OF ELEMENT SPECIES
C KSEQ(NSNOD,NSSPE)  SYSTEM EQUATION NUMBERS
C  NSNOD         NUMBER OF SYSTEM NODES
C  NSSPE         NUMBER OF SYSTEM SPECIES
C  MSBAN         (CURRENT) BANDWIDTH OF SYSTEM EQUATIONS
C--OUTPUT
C  MSBAN         (UPDATED) BANDWIDTH OF SYSTEM EQUATIONS
C--INTERNAL
C  MEBAN         BANDWIDTH OF ELEMENT EQUATIONS
C-----------------------------------------------------------------------
C
C--1.0 FIND MAX AND MIN ELEMENT EQUATION NUMBERS
C
      MAX = ABS(KSEQ(LN(1),LS(1)))
```

```
      MIN = MAX
      DO 10 I=1,NENOD
      DO 10 J=1,NESPE
        NN = ABS(KSEQ(LN(I),LS(J)))
        IF(NN.GT.MAX) MAX=NN
        IF(NN.LT.MIN) MIN=NN
   10 CONTINUE
C
C--2.0 COMPUTE ELEM. BADWIDTH AND COMPARE TO CURRENT MAX SYST. BANDWIDTH
C
      MEBAN = MAX-MIN+1
      IF(MEBAN.GT.MSBAN) MSBAN=MEBAN
      RETURN
      END

C---------------------------------------------------------------- RPRTNO
      SUBROUTINE RPRTNO(X,NX,LABEL)
C--SUB:RPRTNO - REPORTS REAL VECTOR(X) BY INDEX NUMBER

C-----VARIABLE---------DESCRIPTION--------------------------------------
C   X(NX)     VECTOR OF REAL VALUES ORDERED BY INDEX NUMBER
C   LABEL     TABLE LABEL CHARACTER*4
C-----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE 'IOCOM.INC'

      REAL*8 X(NX)
      CHARACTER LABEL*4

        WRITE(NOT,2000)  (LABEL,N=1,4)
        IF(ECHO) WRITE(NTW,2000)  (LABEL,N=1,4)
        WRITE(NOT,2010)  (N, X(N), N=1,NX)
        IF(ECHO) WRITE(NTW,2010)  (N, X(N), N=1,NX)

 2000 FORMAT(/,6X,4(2X,A4,'    Value',3X))
 2010 FORMAT((6X,4(I6,1X,G11.3)))

      RETURN
      END

C---------------------------------------------------------------- RPRTEN
      SUBROUTINE RPRTEN(X,KSEQ)
C--SUB:RPRTEN - REPORTS(X)IN NODE ORDER SEQUENCE FOR EACH SPECIES

C-----VARIABLE---------DESCRIPTION--------------------------------------
C   X(NSEQ)        VECTOR OF VALUES ORDERED BY EQUATION NUMBER
C-----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 X(NSEQ),XX(4)
      INTEGER KSEQ(NSNOD,NSSPE)
      CHARACTER FLG(4)*1

      WRITE(NOT,2000)
      IF(ECHO) WRITE(NTW,2000)
 2000 FORMAT(/,
     .13X,'"*" = independent DOFs          "U" = undefined DOFs.')

      DO 100 M=1,NSSPE
        WRITE(NOT,2010) SID(M)
        IF(ECHO) WRITE(NTW,2010) SID(M)
 2010   FORMAT(/,8X,'Species: ',A4,/,
     +  6X,4(2X,'Node     Value',3X))
      DO 100 N=1,NSNOD,4
        NN = MIN(N+3,NSNOD)
        DO 10 I=N,NN,1
          NEQ = KSEQ(I,M)
          NNEQ = ABS(NEQ)
          IF(NEQ.LT.0) THEN
            XX(I-N+1)  = X(NNEQ)
            FLG(I-N+1) = '*'
          ELSEIF(NEQ.EQ.0) THEN
            XX(I-N+1)  = 0.0D0
            FLG(I-N+1) = 'U'
          ELSE
            XX(I-N+1)  = X(NNEQ)
            FLG(I-N+1) = ' '
          ENDIF
   10   CONTINUE
        IF(ECHO) WRITE(NTW,2020)  (I,FLG(I-N+1),XX(I-N+1),I=N,NN)
        WRITE(NOT,2020)  (I,FLG(I-N+1),XX(I-N+1),I=N,NN)
 2020   FORMAT((6X,4(I6,1A1,G11.3)))
  100 CONTINUE

      RETURN
      END

C---------------------------------------------------------------- FORMF
      SUBROUTINE FORMF(KSEQ,F,WE,FORM,ERR)
C--SUB:FORMF - FORMS SYSTEM FLOW MATRIX AND CONDF CONTRIBUTION TO [V]
C              ARRAY CONT(NSEQ) USED TO CHECK NODAL MASS FLOW CONTINUITY

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 F(NSEQ,1), WE(NFELM)
      INTEGER KSEQ(NSNOD,NSSPE), MPCONT
      LOGICAL ERR, LN
      CHARACTER FORM*4, TYPE*4

C-----VARIABLE---------DESCRIPTION--------------------------------------
C   NEL       : (CURRENT) ELEMENT NUMBER
C   NSP       : (CURRENT) SPECIES NUMBER
C   FORM      : FORM OF SYSTEM ARRAY 'FULL' OR 'BAND'
C   CONT(NSNOD)   : NODAL MASS CONTINUITY ACCUMULATOR
C   LN        : (LOCATION) NODE OF KINETICS ELEMENT
C-----------------------------------------------------------------------

C
C--0.0 INITIALIZE SYSTEM ARRAYS
C
      IF(FORM.EQ.'BAND') CALL ZEROR(F,NSEQ,2*MSBAN-1)
      IF(FORM.EQ.'FULL') CALL ZEROR(F,NSEQ,NSEQ)

C
C--1.0 PROCESS FLOW ELEMENTS
C
C----1.1 FORM AND ASSEMBLE ELEMENT ARRAYS
C
      REWIND(ND1)
      CALL ZEROR(IA(MPVCD),NSNOD,1)
      CALL ZEROR(IA(MPCONT),NSNOD,1)
      DO 10 NEL=1,NFELM
      READ(ND1, ERR=900, END=900) TYPE

      IF(TYPE.EQ.'SIMP') THEN
        CALL SIMP(NEL,WE,IA(MPEFF),KSEQ,F,IA(MPCONT),FORM,ERR)
        IF(ERR) RETURN
      ELSEIF(TYPE.EQ.'CNDF') THEN
        CALL CNDF(NEL,WE,IA(MPDIFF),IA(MPGENR),KSEQ,F,IA(MPVCD),
     +  IA(MPCONT),FORM,ERR)
        IF(ERR) RETURN
      ELSE
        GO TO 900
   10 ENDIF
C
C----1.2 REPORT NET TOTAL MASS FLOW
C
      WRITE(NOT,2200)
      IF(ECHO) WRITE(NTW,2200)
 2200 FORMAT(/,'   == Net Total Mass Flow')
      CALL RPRTNO(IA(MPCONT),NSNOD,'Node')
C
C--2.0 PROCESS KINETICS ELEMENTS
C

C----'TEMP' STORES SPECIES CONNECTIVITY ARRAY, LM(NSSPE)

      CALL DELETE('TEMP')
      CALL DEFINR('TEMP',MPTEMP,NSSPE,1)

      REWIND(ND2)

      DO 200 NEL=1,NKINEL
        READ(ND2, ERR=950, END=950) LN, NK
  200 CALL KINELK(LN,KSEQ,IA(MPKIK(NK)),F,IA(MPTEMP),IA(MPV),FORM)

      CALL DELETE('TEMP')

      RETURN
C
C--3.0 READ ERROR TERMINATION
C
  900 ERR = .TRUE.
      CALL ALERT(
     +'ERROR: Read error or EOF in file '//FNAME//'.FEL','$','$')
      RETURN

  950 ERR = .TRUE.
      CALL ALERT(
     +'ERROR: Read error or EOF in file '//FNAME//'.KIN','$','$')
      RETURN
      END

C---------------------------------------------------------------- SIMP
      SUBROUTINE SIMP(NEL,WE,EFF,KSEQ,F,CONT,FORM,ERR)
C--SUB:SIMP - FORMS AND ASSEMBLES SIMPLE FLOW ELEMENT EQUATIONS
C             FOR ALL SPECIES CONSIDERED

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      REAL*8 WE(NFELM),EFF(NSSPE),F(NSEQ,1),CONT(NSNOD),ELF(2,2),W
      INTEGER KSEQ(NSNOD,NSSPE),LN(2), LM(2)
      CHARACTER FORM*4
      LOGICAL ERR

C-----VARIABLE---------DESCRIPTION--------------------------------------
C   NEL       : (CURRENT) ELEMENT NUMBER
C   FORM      : FORM OF SYSTEM ARRAY 'FULL' OR 'BAND'
C   CONT(NSNOD)   : NODAL MASS CONTINUITY ACCUMULATOR
C   EF(NENOD,NENOD)  : ELEMENT [F] ARRAY
C   W           : ELEMENT TOTAL MASS FLOW RATE
C   LN(2)           : ELEMENT NODE LOCATION/CONNECTIVITY
C   NSP       : (CURRENT) SPECIES NUMBER
C   LM(2)           : SYSTEM DOF CORRESPONDING TO EACH ELEMENT DOF
C-----------------------------------------------------------------------

C
C--1.0 GET ELEMENT DATA
C
      READ(ND1,END=900,ERR=900) LN(1),LN(2),(EFF(I),I=1,NSSPE)
      W = WE(NEL)
C
C--2.0 FORM ELEMENT ARRAYS
C
      IF(W.GT.0.0D0) THEN
        ELF(1,1) = W
        ELF(1,2) = 0.0D0
        ELF(2,2) = 0.0D0
        CONT(LN(1)) = CONT(LN(1)) + W
        CONT(LN(2)) = CONT(LN(2)) - W
      ELSEIF(W.LT.0.0D0) THEN
```

```
        ELF(1,1) = 0.0D0
        ELF(2,1) = 0.0D0
        ELF(2,2) = -W
        CONT(LN(1)) = CONT(LN(1)) + W
        CONT(LN(2)) = CONT(LN(2)) - W
      ELSE
        ELF(1,1) = 0.0D0
        ELF(1,2) = 0.0D0
        ELF(2,1) = 0.0D0
        ELF(2,2) = 0.0D0
      ENDIF
C
C----2.1 LOOP OVER SPECIES FOR NONZERO OFF-DIAGONAL TERM
C
      DO 10 NSP=1,NSSPE
        LM(1) = ABS(KSEQ(LN(1),NSP))
        LM(2) = ABS(KSEQ(LN(2),NSP))
        IF(W.GT.0.0D0) THEN
          ELF(2,1) = -W*(1.0D0-EFF(NSP))
        ELSEIF(W.LT.0.0D0) THEN
          ELF(1,2) = W*(1.0D0-EFF(NSP))
        ENDIF
C
C----2.2 ASSEMBLE ELEMENT ARRAYS
C
      CALL ADDA(ELF,2,F,NSEQ,MSBAN,LM,1.0D0,FORM)

   10 CONTINUE

      RETURN

  900 ERR = .TRUE.
      CALL ALERT(
     +'ERROR: Read error or EOF in file '//FNAME//'.FEL','$','$')
      RETURN
      END

C------------------------------------------------------------------- CNDF
      SUBROUTINE CNDF(NEL,WE,DIFF,GENR,KSEQ,F,VCD,CONT,FORM,ERR)
C--SUB:CNDF - FORMS AND ASSEMBLES CONV-DIFF FLOW ELEMENT EQUATIONS
C               FOR ALL SPECIES CONSIDERED

      INCLUDE 'IOCOM.INC'
      INCLUDE 'CNTCOM.INC'

      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 WE(NFELM),DIFF(NSSPE),GENR(NSSPE),F(NSEQ,1),VCD(NSNOD),
     +CONT(NSNOD),ELF(2,2),W,MASSL,LENGTH,FACTOR
      INTEGER KSEQ(NSNOD,NSSPE),LN(2),LM(2)
      CHARACTER FORM*4
      LOGICAL ERR

C-----VARIABLE---------DESCRIPTION-------------------------------------
C   NEL        : (CURRENT) ELEMENT NUMBER
C   FORM       : FORM OF SYSTEM ARRAY 'FULL' OR 'BAND'
C    CONT(NSNOD) : NODAL MASS CONTINUITY ACCUMULATOR
C    ELF(NENOD,NENOD) : ELEMENT [F] ARRAY
C   W          : ELEMENT TOTAL MASS FLOW RATE
C   LN(2)            : ELEMENT NODE LOCATION/CONNECTIVITY
C   NSP        : (CURRENT) SPECIES NUMBER
C   LM(2)            : SYSTEM DOF CORRESPONDING TO EACH ELEMENT DOF
C   MASSL            : AIR MASS PER UNIT LENGTH - CNDF ELEMENTS
C   LENGTH           : FLOW PASSAGE LENGTH - CNDF ELEMENTS
C   FACTOR           : UPWIND FACTOR - CNDF ELEMENTS
C---------------------------------------------------------------------

C
C--1.0 GET ELEMENT DATA
C
      READ(ND1,END=900,ERR=900) LN(1),LN(2),MASSL,LENGTH,FACTOR,
     +  (DIFF(N),N=1,NSSPE)

      W = WE(NEL)
C
C--2.0 FORM ELEMENT ARRAYS
C
      IF(W.LT.0.0D0) THEN
        LNTEMP = LN(2)
        LN(2) = LN(1)
        LN(1) = LNTEMP
      ENDIF
C
C----2.1 FORM ELEMENT LUMPED VOLUMETRIC MASS TERMS
C
      VCD(LN(1)) = VCD(LN(1)) + MASSL*LENGTH*0.50D0
      VCD(LN(2)) = VCD(LN(2)) + MASSL*LENGTH*0.50D0
C
C----2.2 ACCUMULATE MASS FLOW RATES FOR CONTINUITY CHECK
C
      CONT(LN(1)) = CONT(LN(1)) + W
      CONT(LN(2)) = CONT(LN(2)) - W
C
C----2.3 LOOP OVER SPECIES TO FORM ELEM MASS TRANSPORT MATRIX & ASSEMBLE
C
      DO 10 NSP=1,NSSPE
        LM(1) = ABS(KSEQ(LN(1),NSP))
        LM(2) = ABS(KSEQ(LN(2),NSP))

        COEF = MASSL*DIFF(NSP)/LENGTH
        IF(FACTOR.EQ.-1.0D0) FACTOR=MAX(0.0D0,1.0D0-COEF/W)
        ELF(1,1) = (W/2)*( 1 + FACTOR) + COEF
        ELF(1,2) = (W/2)*( 1 - FACTOR) - COEF
        ELF(2,1) = (W/2)*(-1 - FACTOR) - COEF
        ELF(2,2) = (W/2)*(-1 + FACTOR) + COEF

C
C----2.2 ASSEMBLE ELEMENT FLOW ARRAYS
C
        CALL ADDA(ELF,2,F,NSEQ,MSBAN,LM,1.0D0,FORM)

   10 CONTINUE

      RETURN

  900 ERR = .TRUE.
      CALL ALERT(
     +'ERROR: Read error or EOF in file '//FNAME//'.FEL','$','$')
      RETURN
      END

C---------------------------------------------------------------- KINELK
      SUBROUTINE KINELK(LN,KSEQ,KIK,F,LM,V,FORM)
C--SUB:KINELK - FORMS KINETICS ELEMENT ARRAY FROM KIN RATE COEF. MATRIX
C
C   LN          : (LOCATION) NODE OF KINETICS
C   KIK(NSSPE,NSSPE): KINETICS RATE COEF. MATRIX
C   F(NSEQ,1)       : SYSTEM FLOW MATRIX
C   LM(NSSPE)       : SYSTEM DOF CORRESPONDING TO EACH ELEMENT DOF
C   V(NSNOD)  : NODAL VOLUMETRIC MASSES
C   FORM        : FORM OF [F]: 'FULL' OR 'BAND'
C------------------------------------------------------------------

      INCLUDE 'CNTCOM.INC'

      REAL*8 KIK(NSSPE,NSSPE), F(NSEQ,1), V(NSNOD), SCALE
      INTEGER LM(NSSPE), LN, KSEQ(NSNOD,NSSPE)
      CHARACTER FORM*4

      DO 100 N=1,NSSPE
  100 LM(N) = ABS(KSEQ(LN,N))
      SCALE = V(LN)
      CALL ADDA(KIK,NSSPE,F,NSEQ,MSBAN,LM,SCALE,FORM)

      RETURN
      END

C----------------------------------------------------------------- ADDA
      SUBROUTINE ADDA(AE,NEDOF,AS,NSDOF,MSBAN,LM,SCALE,FORM)
C--SUB:ADDA - ADDS SCALED ELEMENT ARRAY, SCALE*[AE], TO SYSTEM ARRAY, AS
C
      REAL*8 AE(NEDOF,NEDOF), AS(NSDOF,1), SCALE
      INTEGER LM(NEDOF)
      CHARACTER FORM*4

C-----VARIABLE---------DESCRIPTION-------------------------------------
C    AE(NEDOF,NEDOF)  : ELEMENT ARRAY
C   NEDOF          : NUMBER OF ELEMENT DOFS
C  AS(NSDOF,2*MSBAN-1)     : (COMPACTED) BANDED ASYM. SYSTEM ARRAY
C        -- OR --
C    AS(NSDOF,NSDOF)  : FULL ASYM. SYSTEM ARRAY
C   LM(NEDOF)        : SYSTEM DOF CORRESPONDING TO EACH ELEMENT DOF
C   SCALE          : SCALAR FACTOR
C   FORM       : FORM OF SYSTEM ARRAY 'FULL' OR 'BAND'
C
C---------------------------------------------------------------------
      DO 20 I=1,NEDOF
        II = LM(I)
        DO 10 J=1,NEDOF
          IF(FORM.EQ.'BAND') JJ = MSBAN - II + LM(J)
          IF(FORM.EQ.'FULL') JJ = LM(J)
          AS(II,JJ) = AS(II,JJ) + SCALE*AE(I,J)
   10   CONTINUE
   20 CONTINUE
      RETURN
      END

C--------------------------------------------------------------- FORMVM
      SUBROUTINE FORMVM(KSEQ,V,VCD,VM)
C--SUB:FORMVM - FORMS VOLUMETRIC MASS MATRIX (A DIAGONAL ARRAY)
C
      INCLUDE 'CNTCOM.INC'

      REAL*8 V(NSNOD),VM(NSEQ),VCD(NSNOD),VN
      INTEGER KSEQ(NSNOD,NSSPE)

      CALL ZEROR(VM,NSEQ,1)

      DO 10 N=1,NSNOD
        VN = V(N) + VCD(N)
      DO 10 M=1,NSSPE
        NEQ = ABS(KSEQ(N,M))
        IF(NEQ.NE.0) VM(NEQ) = VN
   10 CONTINUE

      RETURN
      END

C---------------------------------------------------------------GETTDT
      SUBROUTINE GETTDT(TDAT)
C--SUB: GETTD0 - UPDATES TIME DATA VALUES
C
      INCLUDE 'IOCOM.INC'

      REAL*8 TDAT(2)
C
C--1.0 UPDATE OLD VALUES
C
      TDAT(2) = TDAT(1)
C
C--2.0 READ NEW VALUE
C
C---- CHECK FOR END-OF-COMMAND "END"
      IF(EOC) THEN
        EOD = .TRUE.
        RETURN
      ENDIF
      IF(MODE.EQ.'INTER') CALL PROMPT('TIME>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
```

```
C---- CHECK FOR END-OF-COMMAND "END"
      IF(EOC) THEN
        EOD = .TRUE.
        RETURN
      ENDIF
      CALL FREER('E',TDAT(1),1)
C---- REPORT
      IF(ECHO) WRITE(NTW,2020) TDAT(1)
      WRITE(NOT,2020) TDAT(1)
 2020 FORMAT(/,'   == Time: ',G10.3)

      RETURN
      END

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C                                                                      C
C       C O M M A N D   P R O C E S S O R   U T I L I T I E S        C
C                                                                      C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C

C---------------------------------------------------------------- NOPEN
      SUBROUTINE NOPEN(LUN,FNAME,FRM)
C--SUB: NOPEN - OPENS A FILE AS A NEW FILE WHETHER IT EXISTS OR NOT
C                LUN = LOGICAL UNIT NUMBER
C                FNAME = FILENAME
C                FRM = FORM; 'UNFORMATTED' OR 'FORMATTED'

      INTEGER LUN
      CHARACTER FNAME*(*), FRM*(*)
      LOGICAL FOUND

      INQUIRE(FILE=FNAME,EXIST=FOUND)
      IF(FOUND) THEN
        OPEN(LUN,FILE=FNAME,STATUS='OLD',FORM=FRM)
        IF(FRM.EQ.'FORMATTED') THEN
          WRITE(LUN,2000) LUN
 2000     FORMAT(I6)
        ELSEIF(FRM.EQ.'UNFORMATTED') THEN
          WRITE(LUN) LUN
        ENDIF
        CLOSE(LUN,STATUS='DELETE')
        OPEN(LUN,FILE=FNAME,STATUS='NEW',FORM=FRM)
      ELSE
        OPEN(LUN,FILE=FNAME,STATUS='NEW',FORM=FRM)
      ENDIF

      RETURN
      END

C-----------------------------------------------------------------APPEND
      SUBROUTINE APPEND(LUN)
C--SUB: APPEND - POSITIONS 'OLD' FILE AT LAST RECORD SO ADDITIONAL
C                RECORDS MAY BE APPENDED
C                LUN = LOGICAL UNIT NUMBER
C
      INTEGER LUN

      REWIND LUN
   10 READ(LUN,*,END=20)
      GO TO 10
   20 BACKSPACE(LUN)
      RETURN
      END

C--------------------------------------------------------------- PROMPT
      SUBROUTINE PROMPT(STRING)
C--SUB:PROMPT - INLINE PROMPT

      INCLUDE 'IOCOM.INC'

      CHARACTER STRING*(*)

      WRITE(NTW,'(A)') STRING
      RETURN
      END

C---------------------------------------------------------------- PROMH
      SUBROUTINE PROMH(N)
C--SUB:PROMH - "HOLLERITH PROMPT"

      COMMON MTOT,NP,IA(1)

      INCLUDE 'IOCOM.INC'
      CHARACTER*1 NCMND,M
      COMMON /CMND/ NCMND(8),M(4,7)

C-----PROMPT FOR ARRAY NAMES
      IF(MODE.EQ.'BATCH') GO TO 900
      DO 200 I=1,N
  100 IF(M(1,N).NE.' ') GO TO 200
      CALL PROMPT('   ** Enter name of array # '//CHAR(N+48)//': ')
      CALL FREE
      CALL FREEC(' ',M(1,N),8,1)
      GO TO 100
  200 CONTINUE
C
  900 RETURN
      END

C---------------------------------------------------------------- PROMI
      SUBROUTINE PROMI(NR,NC)
C--SUB: PROMI - "INTEGER PROMPT"

      INCLUDE 'IOCOM.INC'

C-----ASK FOR NUMBER OF ROWS AND COLUMNS
      IF(MODE.EQ.'BATCH') GO TO 900
  100 IF(NR.GT.0) GO TO 200
      CALL PROMPT('   ** Enter number of rows: ')
      CALL FREE
      CALL FREEI(' ',NR,1)
      GO TO 100
C
  200 IF(NC.GT.0) GO TO 900
      CALL PROMPT('   ** Enter number of columns: ')
      CALL FREE
      CALL FREEI(' ',NC,1)
      GO TO 200
C
  900 RETURN
      END

C-----------------------------------------------------------------ABORT
      SUBROUTINE ABORT
C--SUB:ABORT - ABORTS COMMAND AND RETURNS TO INTERACTIVE MODE
C-----------------------------------------------------------------------
      INCLUDE 'IOCOM.INC'

      WRITE(NTW,2000)
      WRITE(NOT,*)
      WRITE(NOT,2000)
 2000 FORMAT(' **** COMMAND ABORTED')
      IF(MODE.EQ.'BATCH') CALL RETRN

      RETURN
      END

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C                                                                      C
C                  C A L S A P X   L I B R A R Y                       C
C                                                                      C
C        AN EXTENSION OF "CAL-SAP" LIBRARY OF SUBROUTINES              C
C              DEVELOPED BY ED WILSON, U.C. BERKELEY                   C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C 1.0 FREE-FIELD INPUT SUBROUTINES
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C----------------------------------------------------------------- FREE
      SUBROUTINE FREE
C--SUB:FREE - READ LINE OF FREE FIELD DATA
C             COMMENTS LINES ECHOED TO SCREEN

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'
C
C-0.0-INITIALIZE VARIABLES
C
      EOD = .FALSE.
      EOC = .FALSE.
      DO 5 I=1,160
    5 LINE(I)=' '
C
C-1.0 GET LINE OF DATA
C
   10 I = 1
      II= 80
      READ(NCMD,1000,ERR=100) (LINE(KK),KK=I,II)

C-----CHECK FOR ADDITIONAL LINE

      JJ = LENTRM(LLINE)
      DO 12 K=I,JJ
        IF(LINE(K).EQ.'\') THEN
          I = K
          II= K+79
          READ(NCMD,1000,ERR=100) (LINE(KK),KK=I,II)
 1000     FORMAT(80A1)
          GO TO 14
        ENDIF
   12 CONTINUE

C-----CHECK FOR COMMENT

   14 IF(LINE(1).EQ.'*') THEN
        IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
        CALL FREEWR(NOT)
        GO TO 10
      ENDIF
C
C-2.0 DETERMINE LENGTH-OF-INFORMATION
C
      JJ = LENTRM(LLINE)
C
C-3.0 DETERMINE LENGTH-OF-DATA AND CONVERT DATA TO UPPER CASE
C
      ISP = ICHAR(' ')
      IA  = ICHAR('a')
      DO 30 I=1,JJ
        IF(LINE(I).EQ.'<') GO TO 32
        NN = ICHAR(LINE(I))
        IF(NN.GE.IA) LINE(I) = CHAR(NN-ISP)
   30 CONTINUE
   32 II = I - 1
C
C-4.0 CHECK FOR END-OF-DATAGROUP & END-OF-COMMAND
C
      IF(LINE(1).EQ.'<') EOD = .TRUE.

      IF(LINE(1)//LINE(2)//LINE(3).EQ.'END') EOC = .TRUE.

      RETURN

C-----ERROR IN READ -----

  100 CALL ALERT('ERROR: Error in reading input line.','$','$')
      CALL ABORT
      RETURN

      END
```

```
C-------------------------------------------------------------------FREEWR
      SUBROUTINE FREEWR(LUN)
C--SUB:FREEWR - WRITE COMMAND/DATA LINE TO FILE LUN
C               LUN = LOGICAL UNIT NUMBER TO WRITE TO

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'

      WRITE(LUN,2000)  (LINE(I),I=1,JJ)
 2000 FORMAT (1X,80A1)

      RETURN
      END

C-------------------------------------------------------------------FREEFN
      SUBROUTINE FREEFN(SEP,NC,FOUND)
C--SUB:FREEFN - FINDS NEXT NC-CHARACTER SEPARATOR IN INPUT FILE
C               SEP(NC)*1 = CHARACTER STRING

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'

      CHARACTER*1 SEP(NC)
      LOGICAL FOUND

      FOUND = .FALSE.

   50 CALL FREE
      IF(NC.LE.II) THEN
        DO 60 N=1,NC
   60   IF(SEP(N).NE.LINE(N)) GO TO 50
        FOUND = .TRUE.
        RETURN
      ELSE
        GO TO 50
      ENDIF

      RETURN
      END

C-------------------------------------------------------------------- FREER
      SUBROUTINE FREER(IC,DATA,NUM)
C--SUB:FREER - FIND AND INTERPRET REAL DATA
C              IC*1 = DATA IDENTIFIER CHARACTER
C              DATA = REAL DATA RETURNED
C              NUM  = NUMBER OF DATA VALUES TO EXTRACT

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION DATA(10)
      CHARACTER IC*1

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'

      NODATA = .FALSE.
C-----FIND REAL STRING ------------------------------
   90 I=0
      IF(IC.EQ.' ') GO TO 250
      DO 100 I=1,II
      IF((LINE(I).EQ.IC).AND.(LINE(I+1).EQ.'=')) GO TO 250
  100 CONTINUE
      NODATA = .TRUE.
      RETURN
C---- EXTRACT REAL DATA -----
  250 DO 260 J=1,NUM
  260 DATA(J)=0.0
      DO 300 J=1,NUM
      JJ=0
  270 IF(I.GT.II) GO TO 300
      CALL FREER1(I,XX,NN)
      IF(JJ.NE.0) GO TO 275
      DATA(J) = XX
      GO TO 290
C-----ARITHMETRIC STATEMENT -----
  275 IF(JJ.EQ.1) DATA(J)=DATA(J)*XX
      IF(JJ.EQ.2) DATA(J)=DATA(J)/XX
      IF(JJ.EQ.3) DATA(J)=DATA(J)+XX
      IF(JJ.EQ.4) DATA(J)=DATA(J)-XX
      IF(JJ.NE.5) GO TO 290
C-----EXPONENTIAL DATA -----
      JJ = DABS(XX)
      IF(JJ.EQ.0) GO TO 290
      DO 280 K=1,JJ
      IF(XX.LT.0.0) DATA(J) = DATA(J)/10.
      IF(XX.GT.0.0) DATA(J) = DATA(J)*10.
  280 CONTINUE
C-----SET TYPE OF STATEMENT -----
  290 JJ=0
      IF(LINE(I).EQ.'*') JJ=1
      IF(LINE(I).EQ.'/') JJ=2
      IF(LINE(I).EQ.'+') JJ=3
      IF(LINE(I).EQ.'-') JJ=4
      IF(LINE(I).EQ.'E') JJ=5
      IF(LINE(I+1).EQ.'=') JJ=0
      IF(JJ.NE.0) GO TO 270
      IF(NN.GT.9) RETURN
  300 CONTINUE
      RETURN
      END

C-------------------------------------------------------------------FREER1
      SUBROUTINE  FREER1(I,XX,NN)
C-SUB:FREER1 - INTERPRETS A SINGLE REAL VALUE

      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE 'FRECOM.INC'

C-----CONVERT STRING TO REAL FLOATING POINT NUMBER --
      IF(LINE(I+1).EQ.'=') I=I+1
      Y=0
      IS=1
      XX=0.0
      IF(LINE(I+1).EQ.'-') THEN
        IS=-1
        I=I+1
      ELSEIF(LINE(I+1).EQ.'+') THEN
        IS=1
        I=I+1
      ELSE
        CONTINUE
      ENDIF
  267 IF(LINE(I+1).NE.' ') GO TO 270
      I=I+1
      IF(I.GT.II) GO TO 300
      GO TO 267
  270 I=I+1
      IF(I.GT.II) GO TO 300
      IF((LINE(I).EQ.' ').AND.(LINE(I+1).EQ.' ')) GO TO 270
      NN = ICHAR( LINE(I) ) - ICHAR('0')
      XN=ISIGN(NN,IS)
      IF(LINE(I).NE.'.') GO TO 275
      Y=1.0
      GO TO 270
  275 IF(LINE(I).EQ.' ') GO TO 300
      IF(LINE(I).EQ.',') GO TO 300
      IF((NN.LT.0).OR.(NN.GT.9)) GO TO 300
      IF(Y.EQ.0) GO TO 280
      Y=Y/10.
      XN=XN*Y
      XX=XX+XN
      GO TO 270
  280 XX=10.*XX+XN
      GO TO 270
  300 RETURN
      END

C-------------------------------------------------------------------- FREEI
      SUBROUTINE FREEI(IC,IDATA,NUM)
C--SUB:FREEI - FIND AND INTERPRET INTEGER DATA
C             IC*1  = DATA IDENTIFIER CHARACTER
C             IDATA = INATEGER DATA RETURNED
C             NUM   = NUMBER OF DATA VALUES TO EXTRACT
      CHARACTER*1 IC,LNE
      DIMENSION IDATA(72)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'

      NODATA = .FALSE.
C-----FIND INTEGER STRING ----------------------------
   90 I=0
      IF(IC.EQ.' ') GO TO 200
      DO 100 I=1,II
      IF((LINE(I).EQ.IC).AND.(LINE(I+1).EQ.'=')) GO TO 200
  100 CONTINUE
      NODATA = .TRUE.
      RETURN
C-----ZERO INTEGER STRING -----
  200 DO 210 J=1,NUM
  210 IDATA(J)=0
      IF(LINE(I+1).EQ.'=') I=I+1
      DO 250 J=1,NUM
      ISIGN = 1
C-----SKIP BLANKS BETWEEN INTEGERS -----
  215 IF(LINE(I+1).NE.' ') GO TO 220
      I=I+1
      IF(I.GT.II) GO TO 900
      GO TO 215
  220 I=I+1
      IF(I.GT.II) GO TO 230
C-----CHECK FOR SIGN -----
      LNE = LINE(I)
      IF(LNE.NE.'-') GO TO 225
      ISIGN = -1
      GO TO 220
C-----EXTRACT INTEGER -----
  225 IF(LNE.EQ.' ') GO TO 230
      IF(LNE.EQ.',') GO TO 230
      IF(LNE.EQ.'<') GO TO 230
      NN = ICHAR(LNE) - ICHAR('0')
      IF((NN.LT.0).OR.(NN.GT.9)) GO TO 900
      IDATA(J)=10*IDATA(J)+NN
      GO TO 220
C-----SET SIGN -----
  230 IDATA(J) = IDATA(J)*ISIGN
  250 CONTINUE
  900 RETURN
      END

C-------------------------------------------------------------------- FREEC
      SUBROUTINE FREEC(IC,IDATA,NC,NUM)
C--SUB:FREEC - FIND AND INTERPRET CHARACTER DATA
C             IC*1  = DATA IDENTIFIER CHARACTER
C             IDATA = CHARACTER DATA RETURNED
C             NC    = NUMBER OF CHARACTERS PER DATA VALUE
C             NUM   = NUMBER OF DATA VALUES TO EXTRACT
      CHARACTER*1 IC,IDATA
      DIMENSION IDATA(NC,NUM)

      INCLUDE 'IOCOM.INC'
      INCLUDE 'FRECOM.INC'

      NODATA = .FALSE.
C-----FIND DATA IDENTIFIER
   90 I=0
      IF(IC.EQ.' ') GO TO 200
      DO 100 I=2,II
      IF((LINE(I-1).EQ.IC).AND.(LINE(I).EQ.'=')) GO TO 200
```

```
  100 CONTINUE
      NODATA = .TRUE.
      RETURN
C-----EXTRACT CHARACTER DATA -----
  200 DO 210 J=1,NUM
      DO 210 N=1,NC
  210 IDATA(N,J)=' '
C
      DO 300 J=1,NUM
  260 I = I + 1
      IF(I.GT.II) GO TO 400
      IF(LINE(I).EQ.',') GO TO 260
      IF(LINE(I).EQ.' ') GO TO 260
      IF(LINE(I).EQ.CHAR(9)) GO TO 260
      DO 290 N=1,NC
      IF(LINE(I).EQ.'<') GO TO 300
      IF(LINE(I).EQ.' ') GO TO 300
      IF(LINE(I).EQ.',') GO TO 300
      IF(LINE(I).EQ.CHAR(9)) GO TO 300
      IDATA(N,J) = LINE(I)
      IF(N.EQ.NC) GO TO 290
      I = I + 1
  290 CONTINUE
  300 CONTINUE
  400 RETURN
      END

C-----------------------------------------------------------------LENTRM
      FUNCTION LENTRM(STRING)
C--FUN:LENTRM - DETERMINES LENGTH OF TRIMMED STRING - A STRING WITH
C               TRAILING BLANKS REMOVED
C
C
C        LENTOT  : THE TOTAL LENGTH OF THE STRING
C        LENTRM  : THE LENGTH OF THE TRIMMED STRING
C-----------------------------------------------------------------------
      CHARACTER STRING*(*)
      INTEGER LENTOT, LENTRM

      LENTOT = LEN(STRING)

      DO 10 I=LENTOT,1,-1
        IF(STRING(I:I).NE.' ') GO TO 20
   10 CONTINUE

   20 LENTRM = I

      RETURN
      END

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C 2.0 ERROR CHECKING AND ALERT ROUTINES
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C

C-----------------------------------------------------------------CKRRNG
      SUBROUTINE CKRRNG(STRING,RVALUE,NUM,RMIN,OPT,RMAX,ERR)
C--SUB:CKRRNG - CHECKS REAL VALUE RANGE
C               RETURN ERR=.TRUE. IF NOT O.K.
C               VALUE IS A VECTOR OF DIMENSION RVALUE(NUM)
C
C               OPT = 'LELE' : (RMIN <= RVALUE(N) <= RMAX) IS O.K.
C               OPT = 'LTLE' : (VMIN <  RVALUE(N) <= RMAX) IS O.K.
C               OPT = 'LELT' : (RMIN <= RVALUE(N) <  RMAX) IS O.K.
C               OPT = 'LTLT' : (RMIN <  RVALUE(N) <  RMAX) IS O.K.
C
C               STRING = SINGULAR NOUN DESCRIBING RVALUE
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      CHARACTER STRING*(*), OPT*4
      REAL*8 RVALUE(NUM),RMAX,RMIN,RVAL
      LOGICAL ERR

      DO 500 N=1,NUM
      RVAL = RVALUE(N)

      IF(OPT.EQ.'LELE') THEN
        IF(.NOT.((RMIN.LE.RVAL).AND.(RVAL.LE.RMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,RMIN,' <=  value  <= ',RMAX,RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,RMIN,' <=  value  <= ',RMAX,RVAL
          RETURN
        ENDIF
 2000 FORMAT(
     +' **** ERROR: The value of',1X,A,1X,'is limited to the range:',/,
     +12X,G11.4,A,G11.4,/,
     +'             The given or generated value is:',G11.4)

      ELSEIF(OPT.EQ.'LELT') THEN
        IF(.NOT.((RMIN.LE.RVAL).AND.(RVAL.LT.RMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,RMIN,' <=  value  < ',RMAX,RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,RMIN,' <=  value  < ',RMAX,RVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'LTLE') THEN
        IF(.NOT.((RMIN.LT.RVAL).AND.(RVAL.LE.RMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,RMIN,' <  value  <= ',RMAX,RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,RMIN,' <  value  <= ',RMAX,RVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'LTLT') THEN
        IF(.NOT.((RMIN.LT.RVAL).AND.(RVAL.LT.RMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,RMIN,' <  value  < ',RMAX,RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,RMIN,' <  value  < ',RMAX,RVAL
          RETURN
        ENDIF

      ELSE
        ERR = .TRUE.
        WRITE(NTW,2040)
        WRITE(NOT,*)
        WRITE(NOT,2040)
        RETURN
      ENDIF

 2040 FORMAT(' **** ERROR: Call to CKRRNG passed an undefined option.')

  500 CONTINUE

      RETURN
      END
C-----------------------------------------------------------------CKIRNG
      SUBROUTINE CKIRNG(STRING,IVALUE,NUM,IMIN,OPT,IMAX,ERR)
C--SUB:CKIRNG - CHECKS INTEGER VALUE RANGE
C               RETURN ERR=.TRUE. IF NOT O.K.
C               VALUE IS A VECTOR OF DIMENSION IVALUE(NUM)
C
C               OPT = 'LELE' : (IMIN <= IVALUE(N) <= IMAX) IS O.K.
C               OPT = 'LTLE' : (IMIN <  IVALUE(N) <= IMAX) IS O.K.
C               OPT = 'LELT' : (IMIN <= IVALUE(N) <  IMAX) IS O.K.
C               OPT = 'LTLT' : (IMIN <  IVALUE(N) <  IMAX) IS O.K.
C
C               STRING = SINGULAR NOUN DESCRIBING IVALUE
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      CHARACTER STRING*(*), OPT*4
      INTEGER IVALUE(NUM),IMAX,IMIN,IVAL
      LOGICAL ERR

      DO 500 N=1,NUM
      IVAL = IVALUE(N)

      IF(OPT.EQ.'LELE') THEN
        IF(.NOT.((IMIN.LE.IVAL).AND.(IVAL.LE.IMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,IMIN,' <=  value  <= ',IMAX,IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,IMIN,' <=  value  <= ',IMAX,IVAL
          RETURN
        ENDIF
 2000 FORMAT(
     +' **** ERROR: The value of',1X,A,1X,'is limited to the range:',/,
     +12X,I6,A,I6,/,
     +'             The given or generated value is:',I6)

      ELSEIF(OPT.EQ.'LELT') THEN
        IF(.NOT.((IMIN.LE.IVAL).AND.(IVAL.LT.IMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,IMIN,' <=  value  < ',IMAX,IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,IMIN,' <=  value  < ',IMAX,IVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'LTLE') THEN
        IF(.NOT.((IMIN.LT.IVAL).AND.(IVAL.LE.IMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,IMIN,' <  value  <= ',IMAX,IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,IMIN,' <  value  <= ',IMAX,IVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'LTLT') THEN
        IF(.NOT.((IMIN.LT.IVAL).AND.(IVAL.LT.IMAX))) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,IMIN,' <  value  < ',IMAX,IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,IMIN,' <  value  < ',IMAX,IVAL
          RETURN
        ENDIF

      ELSE
        ERR = .TRUE.
        WRITE(NTW,2010)
        WRITE(NOT,*)
        WRITE(NOT,2010)
        RETURN
      ENDIF
 2010 FORMAT(' **** ERROR: Call to CKIRNG passed an undefined option.')

  500 CONTINUE

      RETURN
      END

C-----------------------------------------------------------------CKRZER
      SUBROUTINE CKRZER(STRING,RVALUE,NUM,OPT,ERR)
C--SUB:CKRZER - CHECKS REAL VALUE RELATIVE TO ZERO
C               RETURNS ERR=.TRUE.IF NOT O.K.
C               WHERE VALUE IS A VECTOR OF DIMENSION RVALUE(NUM)
C
C               OPT = 'LT' : RVALUE(N) .LT. 0.0D0 IS O.K.
C               OPT = 'LE' : RVALUE(N) .LE. 0.0D0 IS O.K.
C               OPT = 'GE' : RVALUE(N) .GE. 0.0D0 IS O.K.
C               OPT = 'GT' : RVALUE(N) .GT. 0.0D0 IS O.K.
C               OPT = 'NE' : RVALUE(N) .NE. 0.0D0 IS O.K.
C
```

```
C              STRING = SINGULAR NOUN DESCRIBING RVALUE
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      CHARACTER STRING*(*), OPT*2
      REAL*8 RVALUE(NUM),RVAL
      LOGICAL ERR


      DO 500 N=1,NUM
      RVAL = RVALUE(N)

      IF(OPT.EQ.'LT') THEN
        IF(.NOT.(RVAL.LT.0.0D0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be < 0.',RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be < 0.',RVAL
          RETURN
        ENDIF
 2000 FORMAT(' **** ERROR: The value of',1X,A,1X,A,/,
     +'               The given or generated value is:',G11.4)

      ELSEIF(OPT.EQ.'LE') THEN
        IF(.NOT.(RVAL.LE.0.0D0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be <= 0.',RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be <= 0.',RVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'GE') THEN
        IF(.NOT.(RVAL.GE.0.0D0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be >= 0.',RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be >= 0.',RVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'GT') THEN
        IF(.NOT.(RVAL.GT.0.0D0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be > 0.',RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be > 0.',RVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'NE') THEN
        IF(RVAL.EQ.0.0D0) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must not = 0.',RVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must not = 0.',RVAL
          RETURN
        ENDIF

      ELSE
        ERR = .TRUE.
        WRITE(NTW,2010)
        WRITE(NOT,*)
        WRITE(NOT,2010)
        RETURN
      ENDIF
 2010 FORMAT(' **** ERROR: Call to CKRZER passed an undefined option.')

  500 CONTINUE

      RETURN
      END

C------------------------------------------------------------------CKIZER
      SUBROUTINE CKIZER(STRING,IVALUE,NUM,OPT,ERR)
C--SUB:CKIZER - CHECKS INTEGER VALUE RELATIVE TO ZERO
C              RETURNS ERR=.TRUE.IF NOT O.K.
C              WHERE VALUE IS A VECTOR OF DIMENSION IVALUE(NUM)
C
C              OPT = 'LT' : IVALUE(N) .LT. 0 IS O.K.
C              OPT = 'LE' : IVALUE(N) .LE. 0 IS O.K.
C              OPT = 'GE' : IVALUE(N) .GE. 0 IS O.K.
C              OPT = 'GT' : IVALUE(N) .GT. 0 IS O.K.
C              OPT = 'NE' : IVALUE(N) .NE. 0 IS O.K.
C
C              STRING = SINGULAR NOUN DESCRIBING IVALUE
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      CHARACTER STRING*(*), OPT*2
      INTEGER IVALUE(NUM),IVAL
      LOGICAL ERR


      DO 500 N=1,NUM
      IVAL = IVALUE(N)

      IF(OPT.EQ.'LT') THEN
        IF(.NOT.(IVAL.LT.0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be < 0.',IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be < 0.',IVAL
          RETURN
        ENDIF
 2000 FORMAT(' **** ERROR: The value of',1X,A,1X,A,/,
     +'               The given or generated value is:',I6)

      ELSEIF(OPT.EQ.'LE') THEN
        IF(.NOT.(IVAL.LE.0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be <= 0.',IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be <= 0.',IVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'GE') THEN
        IF(.NOT.(IVAL.GE.0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be >= 0.',IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be >= 0.',IVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'GT') THEN
        IF(.NOT.(IVAL.GT.0)) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must be > 0.',IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must be > 0.',IVAL
          RETURN
        ENDIF

      ELSEIF(OPT.EQ.'NE') THEN
        IF(IVAL.EQ.0) THEN
          ERR = .TRUE.
          WRITE(NTW,2000) STRING,'must not = 0.',IVAL
          WRITE(NOT,*)
          WRITE(NOT,2000) STRING,'must not = 0.',IVAL
          RETURN
        ENDIF

      ELSE
        ERR = .TRUE.
        WRITE(NTW,2010)
        WRITE(NOT,*)
        WRITE(NOT,2010)
        RETURN
      ENDIF
 2010 FORMAT(' **** ERROR: Call to CKIZER passed an undefined option.')

  500 CONTINUE

      RETURN
      END

C------------------------------------------------------------------- ALERT

      SUBROUTINE ALERT(LINE1,LINE2,LINE3)
C--SUB: ERRMSG - WRITES ALERT MESSAGE TO TERMINAL AND OUTPUT FILE
C                LINE1 IS ALWAYS WRITTEN
C                LINE2 IS WRITTEN IF LINE2(1:1).NE.'$'
C                LINE3 IS WRITTEN IF LINE3(1:1).NE.'$'
C-----------------------------------------------------------------------

      INCLUDE 'IOCOM.INC'

      CHARACTER LINE1*(*), LINE2*(*), LINE3*(*)

      WRITE(NTW,2001) LINE1
      WRITE(NOT,*)
      WRITE(NOT,2001) LINE1
 2001 FORMAT(' **** ', (A))

      IF(LINE2(1:1).NE.'$') THEN
        WRITE(NTW,2002) LINE2
        WRITE(NOT,2002) LINE2
      ENDIF
 2002 FORMAT(13X,(A))

      IF(LINE3(1:1).NE.'$') THEN
        WRITE(NTW,2003) LINE3
        WRITE(NOT,2003) LINE3
      ENDIF
 2003 FORMAT(13X,(A))

      RETURN
      END

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C 3.0 DYNAMIC ARRAY MANAGEMENT ROUTINES
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C

C------------------------------------------------------------------ DEFINR
      SUBROUTINE DEFINR(NAME,NA,NR,NC)
C--SUB:DEFINR - DEFINE DIRECTORY AND RESERVE STORAGE
C               FOR REAL ARRAY IN DATABASE
C               NAME = NAME OF ARRAY
C               NA   = BLANK COMMON POINTER TO ARRAY (RETURNED)
C               NR   = NUMBER OF ROWS
C               NC   = NUMBER OF COLUMNS
C-----------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      CHARACTER*1 NAME(4)
      NP = 2
      CALL DEFIN(NAME,NA,NR,NC)
      RETURN
      END

C------------------------------------------------------------------ DEFINI
      SUBROUTINE DEFINI(NAME,NA,NR,NC)
C--SUB:DEFINI - DEFINE DIRECTORY AND RESERVE STORAGE
C               FOR INTEGER ARRAY IN DATABASE
C               NAME = NAME OF ARRAY
C               NA   = BLANK COMMON POINTER TO ARRAY (RETURNED)
```

```
C             NR    = NUMBER OF ROWS
C             NC    = NUMBER OF COLUMNS
C-----------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      CHARACTER*1 NAME(4)
      NP = 1
      CALL DEFIN(NAME,NA,NR,NC)
      RETURN
      END

C----------------------------------------------------------------- DEFIN
      SUBROUTINE DEFIN(NAME,NA,NR,NC)
C-----DEFINE AND RESERVE STORAGE FOR ARRAY -----

      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'

      CHARACTER*1  NAME(4)
C----DEFIN VARIABLES-----------------------------------------------------
C       NAME = NAME OF ARRAY - 4 LOGICALS MAXIMUM
C       NA = LOCATION OF ARRAY IF IN BLANK COMMON
C       NR = NUMBER OF ROWS
C       NC = NUMBER OF COLUMNS
C       MTOT = END OF DIRECTORY
C       NUMA = NUMBER OF ARRAYS IN DATA BASE
C       NEXT = NEXT AVAILABLE STORAGE LOCATION
C       IDIR = START OF DIRECTORY IN BLANK COMMON
C       IP   = NUMBER OF LOGICALS CONTAINED IN DATA TYPE
C       LENR = NUMBER OF LOGICALS IN PHYSICAL RECORD
C       NP = TYPE OF DATA
C          = 1 INTEGER DATA
C          = 2 REAL DATA
C          = 3 LOGICAL DATA
C------DIRECTORY DEFINITION FOR CORE OR SEQUENTIAL FILES
C     IDIR(1,N) = NAME OF ARRAY  - INAME (4 CHAR.)
C     IDIR(5,N) = NUMBER OF ROWS     - NR
C     IDIR(6,N) = NUMBER OF COLUMNS - NC
C     IDIR(7,N) = TYPE OF DATA      - NP
C     IDIR(8,N) = INCORE ADDRESS    - NA
C               = -1 IF SEQUENTIAL FILE ON DISK
C               = -2 IF DIRECT ACCESS ON DISK
C     IDIR(9,N) = SIZE OF ARRAY
C     IDIR(10,N) = 0 IF IN CORE STORAGE
C------DIRECTORY DEFINITION FOR DIRECT ACCESS FILES -----
C     IDIR(5,N) = NUMBER OF INTEGERS
C     IDIR(6,N) = NUMBER OF REAL WORDS
C     IDIR(7,N) = NUMBER OF LOGICALS
C     IDIR(8,N) = NUMBER OF LOGICAL RECORDS
C     IDIR(9,N) = LOGICAL RECORD NUMBER
C     IDIR(10,N)  = LUN IF ON LOGICAL UNIT LUN
C-----------------------------------------------------------------------
C     EVALUATE STORAGE REQUIREMENTS
      NSIZE = (NR*NC*IP(NP) -1)/(IP(1)*2)
      NSIZE = NSIZE*2 + 2
      NA = NEXT
      NEXT = NEXT + NSIZE
C-----SET UP NEW DIRECTORY -----
      NUMA = NUMA + 1
      IDIR = IDIR - 10
      I = IDIR
C-----CHECK STORAGE LIMITS -----
      IF(I.GE.NEXT) GO TO 100
      I = NEXT - I + MTOT - 1
      WRITE(NTW,2000) I,MTOT
      WRITE(NOT,*)
      WRITE(NOT,2000) I,MTOT
      PAUSE
      STOP
  100 CALL ICON(NAME,IA(I))
      IA(I+4) = NR
      IA(I+5) = NC
      IA(I+6) = NP
      IA(I+7) = NA
      IA(I+8) = NSIZE
      IA(I+9) = 0
  900 RETURN
 2000 FORMAT(
     *' **** ERROR: Insufficient blank COMMON storage.',/,
     *'             Storage required  MTOT =',I7,/,
     *'             Storage available MTOT =',I7)
      END

C----------------------------------------------------------------DEFDIR
      SUBROUTINE DEFDIR(NAME,NR,NC,ISTR)
C--SUB:DEFDIR - DEFINE DIRECTORY FOR OUT-OF-CORE FILE
C             NAME = NAME OF ARRAY
C             NR   = NUMBER OF ROWS
C             NC   = NUMBER OF COLUMNS
C             ISTR = OUT OF CORE FLAG (= 1)
C-----------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'

      CHARACTER*1 NAME(4)
C-----EVALUATE STORAGE REQUIREMENTS ----------------
      IF(NP.EQ.0) NP = 2
C-----SET UP NEW DIRECTORY -----
      NUMA  = NUMA + 1
      IDIR = IDIR - 10
      I = IDIR
C-----CHECK STORAGE LIMITS -----
      IF(I.GE.NEXT) GO TO 100
      I = NEXT - I + MTOT - 1
      WRITE(NTW,2000) I,MTOT
      WRITE(NOT,*)
      WRITE(NOT,2000) I,MTOT
      PAUSE
      STOP
```

```
  100 CALL ICON(NAME,IA(I))
      IA(I+4) = NR
      IA(I+5) = NC
      IA(I+6) = NP
      IA(I+7) = ISTR
      IA(I+8) = 0
      IA(I+9) = 0
  900 RETURN
 2000 FORMAT(
     *' **** ERROR: Insufficient blank COMMON storage.',/,
     *'             Storage required  MTOT =',I7,/,
     *'             Storage available MTOT =',I7)
      END

C----------------------------------------------------------------LOCATE
      SUBROUTINE LOCATE(NAME,NA,NR,NC)
C--SUB:LOCATE - LOCATE ARRAY "NAME" AND RETURN
C             NA = POINTER TO LOCATION IN BLANK COMMON
C             NR = NUMBER OF ROWS
C             NC = NUMBER OF COLUMNS
C-----------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)

      CHARACTER*1 NAME
      DIMENSION NAME(4),INAME(4)
C-----LOCATE AND RETURN PROPERTIES ON ARRAY ---------
      NA = 0
      CALL ICON(NAME,INAME)
      I = IFIND(INAME,0)
      IF(I.EQ.0) GO TO 900
C-----RETURN ARRAY PROPERTIES -----
      NA = IA(I+7)
      NR = IA(I+4)
      NC = IA(I+5)
      NP = IA(I+6)
  900 RETURN
      END

C---------------------------------------------------------------- DELETE
      SUBROUTINE DELETE(NAME)
C--SUB:DELETE - DELETE ARRAY "NAME" FROM DATABASE

      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'
      INCLUDE 'IOCOM.INC'

      CHARACTER*1 NAME
      DIMENSION NAME(4),INAME(4)
C-----DELETE ARRAY FROM STORAGE --------------------
  100 CALL ICON(NAME,INAME)
      I = IFIND(INAME,0)
      IF(I.EQ.0) GO TO 900
C     CHECK ON STORAGE LOCATION
  200 NSIZE = IA(I+8)
C-----SET SIZE OF ARRAY -----
      NEXT = NEXT - NSIZE
      NUMA = NUMA - 1
      NA   = IA(I+7)
C-----CHECK IF OUT OF CORE OR DIRECT ACCESS -----
      IF(NA.GT.0) GO TO 500
      WRITE(NTW,1000) NAME
      WRITE(NOT,*)
      WRITE(NOT,1000) NAME
      GO TO 800
  500 IF(NA.EQ.NEXT) GO TO 800
C-----COMPACT STORAGE -----
      II   = NA + NSIZE
      NNXT = NEXT - 1
      DO 700 J=NA,NNXT
      IA(J) = IA(II)
  700 II = II + 1
C-----COMPACT AND UPDATE DIRECTORY -----
  800 NA = I - IDIR
      IDIR = IDIR + 10
      IF(NA.EQ.0) GO TO 900
      NA = NA/10
      DO 860 K=1,NA
      II = I + 9
      DO 850 J=1,10
      IA(II) = IA(II-10)
  850 II = II - 1
      IF(IA(I+7).LE.0) GO TO 860
      IF(IA(I+9).EQ.0) IA(I+7) = IA(I+7) - NSIZE
  860 I = I - 10
C
  900 RETURN
 1000 FORMAT('  -- Name ',4A1,' is being used for an',
     * ' OUT-OF-CORE file.',/)
      END

C------------------------------------------------------------------ICON
      SUBROUTINE ICON(NAME,INAME)
      CHARACTER*1 NAME(4)
      DIMENSION INAME(4)
C-----CONVERT LOGICALS TO INTEGER DATA -------------
      DO 100 I = 1,4
  100 INAME(I) = ICHAR( NAME(I) )
C
      RETURN
      END

C----------------------------------------------------------------- IFIND
      FUNCTION IFIND(INAME,LUN)
C--FUN:IFIND - FIND
      COMMON MTOT,NP,IA(1)
      INCLUDE 'ARYCOM.INC'

      DIMENSION INAME(4)
C-----FIND ARRAY LOCATION --------------------------
      I = IDIR
```

```
      DO 100 N=1,NUMA
      IF(LUN.NE.IA(I+9)) GO TO 100
      IF (INAME(1).NE.IA(I  )) GO TO 100
      IF (INAME(2).NE.IA(I+1)) GO TO 100
      IF (INAME(3).NE.IA(I+2)) GO TO 100
      IF (INAME(4).EQ.IA(I+3)) GO TO 200
  100 I = I + 10
      I = 0
  200 IFIND = I
C
      RETURN
      END

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C
C 4.0 MATRIX OPERATION UTILITIES
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++C

C-----------------------------------------------------------------ZEROR
      SUBROUTINE ZEROR(A,NR,NC)
C--SUB:ZEROR - SET ARRAY A(NR,NC) TO 0.0
      REAL*8 A(NR,NC)
      DO 10 I=1,NR
      DO 10 J=1,NC
      A(I,J) = 0.0D0
   10 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------ZEROI
      SUBROUTINE ZEROI(IA,NR,NC)
C--SUB:ZERORI - SET ARRAY IA(NR,NC) TO 0
      DIMENSION IA(NR,NC)
      DO 10 I=1,NR
      DO 10 J=1,NC
      IA(I,J) = 0
   10 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------ZEROC
      SUBROUTINE ZEROC(CA,NR,NC)
C--SUB:ZERORC - SET ARRAY CA(NR,NC) TO BLANK
      CHARACTER*1 CA(NR,NC)
      DO 10 I=1,NR
      DO 10 J=1,NC
      CA(I,J) = ' '
   10 CONTINUE
      RETURN
      END

C----------------------------------------------------------------FACTCA
      SUBROUTINE FACTCA(A,NEQ,MBAND,ERR)
C--SUB:FACTCA - FACTORS COMPACT ASYMMETRIC MATRIX
C                FACTORS [A] = [L][U]
C                [L][U] IS WRITTEN OVER [A]
C                [A] MAYBE SYM OR ASYM, POSITIVE DEFINITE
C                [A] HAS SEMI-BANDWIDTH MBAND & IS STORED COMPACTLY
C     FROM: HUEBNER & THORNTON "THE FINITE ELEMENT METHOD FOR ENGRS."
C----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE 'IOCOM.INC'

      DIMENSION A(NEQ,2*MBAND-1)
      LOGICAL ERR

      NCOLS = 2*MBAND-1
      KMIN  = MBAND + 1
      DO 50 N=1,NEQ
      IF(A(N,MBAND).EQ.0.0D0) GO TO 60
      IF(A(N,MBAND).EQ.1.0D0) GO TO 20
      C = 1.0D0/A(N,MBAND)
      DO 10 K=KMIN,NCOLS
      IF(A(N,K).EQ.0.0D0) GO TO 10
      A(N,K) = C*A(N,K)
   10 CONTINUE
   20 CONTINUE
      DO 40 L=2,MBAND
      JJ = MBAND - L + 1
      I = N + L - 1
      IF(I.GT.NEQ) GO TO 40
      IF(A(I,JJ).EQ.0.0D0) GO TO 40
      KI = MBAND + 2 - L
      KF = NCOLS + 1 - L
      J = MBAND
      DO 30 K=KI,KF
      J = J + 1
      IF(A(N,J).EQ.0.0D0) GO TO 30
      A(I,K) = A(I,K) - A(I,JJ)*A(N,J)
   30 CONTINUE
   40 CONTINUE
   50 CONTINUE
      RETURN
   60 ERR = .TRUE.
      WRITE(NTW,2000) N
      WRITE(NOT,*)
      WRITE(NOT,2000) N
      RETURN
 2000 FORMAT(' **** ERROR: SUB:FACTCA - Equations may be singular.',/,
     +'              Diagonal of equation number ',I5,' is zero.')
      END

C----------------------------------------------------------------SOLVCA
      SUBROUTINE SOLVCA(A,B,NEQ,MBAND,ERR)
C--SUB:SOLVCA -SOLVES COMPACT ASYMMETRIC FACTORED MATRIX
C             SOLVES [L][U]{X} = {B}
C             [L][U] IS WRITTEN OVER [A]
C             [L][U]=[A] HAS SEMI-BANDWIDTH MBAND & IS STORED COMPACTLY
C             SOLUTION IS WRITTEN OVER {B}
C     FROM: HUEBNER & THORNTON "THE FINITE ELEMENT METHOD FOR ENGRS."
C----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE 'IOCOM.INC'

      DIMENSION A(NEQ,2*MBAND-1), B(NEQ)
      LOGICAL ERR

      NCOLS = 2*MBAND-1

C--1.0 REDUCTION OF {B}

      DO 30 N=1,NEQ
      IF(A(N,MBAND).EQ.0.0D0) GO TO 60
      IF(A(N,MBAND).EQ.1.0D0) GO TO 10
      B(N) = B(N)/A(N,MBAND)
   10 CONTINUE
      DO 20 L=2,MBAND
      JJ = MBAND - L + 1
      I = N + L -1
      IF(I.GT.NEQ) GO TO 20
      IF(A(I,JJ).EQ.0.0D0) GO TO 20
      B(I) = B(I) - A(I,JJ)*B(N)
   20 CONTINUE
   30 CONTINUE

C--2.0 BACKSUBSTITUTION

      LL = MBAND + 1
      DO 50 M=1,NEQ
      N = NEQ + 1 - M
      DO 40 L=LL,NCOLS
      IF(A(N,L).EQ.0.0D0) GO TO 40
      K = N + L - MBAND
      B(N) = B(N) - A(N,L)*B(K)
   40 CONTINUE
   50 CONTINUE
      RETURN
   60 ERR = .TRUE.
      WRITE(NTW,2000) N
      WRITE(NOT,*)
      WRITE(NOT,2000) N
      RETURN
 2000 FORMAT(' **** ERROR: SUB:SOLVCA - Equations may be singular.',/,
     +'              Diagonal of equation number ',I5,' is zero.')
      END

C-----------------------------------------------------------------EIGEN2
      SUBROUTINE EIGEN2(A,T,N,TMX,EP)
C--SUB: EIGEN2 - Unsymmetric Eigen Analysis Routine
C         Based on code from:
C        Wilkinson, J.H. & Reinsch, C., Linear Algebra, Springer-
C        Verlag, 1971
C    Solves eigenproblem for real matrix A(N,N), sym. or unsym., by
C    a sequence of Jacobi-like transformations [T]-1[A][T] where [T]=
C    [T1][T2][T3] ....  Each [Ti] is of the form [Ri][Si] where;
C
C    R:   Rk,k = Rm,m = cos(x)  ;    Rm,k = -Rk,m = sin(x)
C         Ri,i = 1 ;    Ri,j = 0 ;  (i,j ≠ k,m)
C    S:   Sk,k = Sm,m = cosh(y) ;    Sm,k = Sk,m = -sinh(y)
C         Si,i = 1 ;    Si,j = 0 ;  (i,j ≠ k,m)
C
C    in which x,y are determined by the elements of [Ai].
C
C    In the limiting matrix real eigenvalues occupy the diagonal while
C    real and imaginary parts of complex eigen values occupy the
C    diagonal and off-diagonal corners of 2x2 blocks centered on diag.
C
C    Array T(N,N) must be provided to receive eigenvectors.
C         TMX=0     : eigenvectors not generated and A(N,N) may be
C                 passed as T(N,N)
C         TMX<0     : generate left, [T]-1, transformations
C         TMX>0     : generate right, [T], transformations
C    Eigenvectors of real eigenvalues occurr as rows (cols) of [T]-1
C    ([T]).  Eigenvectors for a complex eigenvalue pair aj,i ± iaj,j+1
C    may be formed by tj ± itj+1 where tj, tj+1 are the corresponding
C    rows (cols) of [T]-1 ([T])
C
C    Iterations are limited to 50 maximum. On exit from the procedure
C    TMX records the number of iterations performed. Failure to
C    converge is indicated by TMX=50 or, if all transformations in
C    one iteration are the identity matrix, by TMX<0.
C
C    The machine dependent variable EP is set to 1E-08 and should be
C    reset for machine precision available.
C----------------------------------------------------------------------
C---- D I C T I O N A R Y   O F   V A R I A B L E S -----------------
C
C-----VARIABLE-----------DESCRIPTION-----------------------------------
C----INPUT
C      A(N,N)      Array to be analyzed.
C      N           System size
C      TMS         Control parameter
C----OUTPUT
C      T(N,N)      Array to receive eigenvectors.
C      TMX         Iteration count/iteration flag
C----LOCAL
C      EP     Precision
C----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 A(N,N),T(N,N),EP
      INTEGER N,TMX
      LOGICAL MARK, LEFT, RIGHT
C
C--0.0 INITIALIZE CONTROL VARIABLES
C
      IF(EP.LE.0.0D0) EP = 1.0D-8
      EPS = SQRT(EP)
      LEFT = .FALSE.
```

```
      RIGHT = .FALSE.
      IF(TMX.LT.0) THEN
        LEFT = .TRUE.
      ELSEIF(TMX.GT.0) THEN
        RIGHT = .TRUE.
      ENDIF
      MARK = .FALSE.
C
C--1.0  INITIALIZE [T] AS IDENTITY MATRIX
C
      IF(TMX.NE.0) THEN
        DO 10 I=1,N
          T(I,I) = 1.0D0
        DO 10 J=I+1,N
          T(I,J) = 0.0D0
          T(J,I) = 0.0D0
   10   CONTINUE
      ENDIF
C
C--2.0 MAIN LOOP
C
C-MAC WRITE(*,'(5X,A,\)') ' '
      DO 26 IT=1,50
C-MAC WRITE(*,'(A,\)') '+'
C
C--2.1 IF MARK IS SET
C         TRANSFORMATIONS OF PREVIOUS ITERATION WERE OMITTED
C         PROCEDURE WILL NOT CONVERGE
C
      IF(MARK) THEN
        TMX = 1-IT
        RETURN
      ENDIF
C
C--2.2 COMPUTE CONVERGENCE CRITERIA
C
      DO 20 I=1,N-1
        AII = A(I,I)
      DO 20 J=I+1,N
        AIJ = A(I,J)
        AJI = A(J,I)
        IF((ABS(AIJ+AJI).GT.EPS).OR.
     +    ((ABS(AIJ-AJI).GT.EPS).AND.(ABS(AII-A(J,J)).GT.EPS))) THEN
          GOTO 21
        ENDIF
   20 CONTINUE
      TMX = IT -1
      RETURN
C
C--2.3 BEGIN NEXT TRANSFORMATION
C
   21 MARK = .TRUE.
      DO 25 K=1,N-1
      DO 25 M=K+1,N
        H = 0.0D0
        G = 0.0D0
        HJ = 0.0D0
        YH = 0.0D0
        DO 22 I=1,N
          AIK = A(I,K)
          AIM = A(I,M)
          TE  = AIK*AIK
          TEE = AIM*AIM
          YH  = YH + TE - TEE
          IF((I.NE.K).AND.(I.NE.M)) THEN
            AKI = A(K,I)
            AMI = A(M,I)
            H   = H + AKI*AMI - AIK*AIM
            TEP = TE + AMI*AMI
            TEM = TEE + AKI*AKI
            G   = G + TEP + TEM
            HJ  = HJ - TEP + TEM
          ENDIF
   22   CONTINUE
        H   = H + H
        D   = A(K,K) - A(M,M)
        AKM = A(K,M)
        AMK = A(M,K)
        C   = AKM + AMK
        E   = AKM - AMK
C
C------ COMPUTE ELEMENTS OF [Ri]
C
        IF(ABS(C).LE.EP) THEN
          CX = 1.0D0
          SX = 0.0D0
        ELSE
          COT2X = D/C
          SIG = SIGN(1.0D0,COT2X)
          COTX = COT2X + (SIG*SQRT(1.0D0 + COT2X*COT2X))
          SX = SIG/SQRT(1.0D0 + COTX*COTX)
          CX = SX*COTX
        ENDIF

        IF(YH.LT.0.0D0) THEN
          TEM = CX
          CX = SX
          SX = -TEM
        ENDIF

        COS2X = CX*CX - SX*SX
        SIN2X = 2.0D0*SX*CX
        D = D*COS2X + C*SIN2X
        H = H*COS2X - HJ*SIN2X
        DEN = G + 2.0D0*(E*E + D*D)
        TANHY = (E*D - H/2.0D0)/DEN
C
C------ COMPUTE ELEMENTS OF [Si]
C
        IF(ABS(TANHY).LE.EP) THEN
          CHY = 1.0D0
          SHY = 0.0D0
        ELSE
          CHY = 1.0D0/SQRT(1.0D0 - TANHY*TANHY)
          SHY = CHY*TANHY
        ENDIF
C
C------ COMPUTE ELEMENTS OF [Ti] = [Ri][Si]
C
        C1 =  CHY*CX - SHY*SX
        C2 =  CHY*CX + SHY*SX
        S1 =  CHY*SX + SHY*CX
        S2 = -CHY*SX + SHY*CX
C
C------ APPLY TRANSFORMATION IF WARRANTED
C
        IF((ABS(S1).GT.EP).OR.(ABS(S2).GT.EP)) THEN
          MARK = .FALSE.
C-------- TRANSFORMATION ON THE LEFT
          DO 23 I=1,N
            AKI = A(K,I)
            AMI = A(M,I)
            A(K,I) = C1*AKI + S1*AMI
            A(M,I) = S2*AKI + C2*AMI
            IF(LEFT) THEN
              TKI = T(K,I)
              TMI = T(M,I)
              T(K,I) = C1*TKI + S1*TMI
              T(M,I) = S2*TKI + C2*TMI
            ENDIF
   23     CONTINUE
C-------- TRANSFORMATION ON THE RIGHT
          DO 24 I=1,N
            AIK = A(I,K)
            AIM = A(I,M)
            A(I,K) = C2*AIK - S2*AIM
            A(I,M) = -S1*AIK + C1*AIM
            IF(RIGHT) THEN
              TIK = T(I,K)
              TIM = T(I,M)
              T(I,K) = C2*TIK - S2*TIM
              T(I,M) = -S1*TIK + C1*TIM
            ENDIF
   24     CONTINUE
        ENDIF
   25 CONTINUE
   26 CONTINUE
      TMX = 50

      RETURN
      END
```

Include Files

```
C-----------------------------------------------------------------------
C CALSAPX  A R R A Y   M A N A G E M E N T                   $ARYCOM.INC
C-----------------------------------------------------------------------
      COMMON /ARYCOM/ NUMA,NEXT,IDIR,IP(3)

C----VARIABLE--------DESCRIPTION----------------------------------------
C   MTOT      SIZE OF BLANK COMMON VECTOR IA
C   NP        CURRENT DATA TYPE: 1=INTEGER; 2=REAL; 3=CHAR.
C   IA(MTOT)  BLANK COMMON VECTOR
C   NUMA      NUMBER OF ARRAYS IN BLANK COMMON DATA BASE
C   NEXT      NEXT AVAILABLE STORAGE LOCATION IN BLANK COMMON
C   IDIR      START OF DIRECTORY IN BLANK COMMON
C   IP(3)          NUMBER OF BYTES IN INTEGER, REAL, CHARACTER DATA
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C CALSAPX  C O M M A N D   M A N A G E M E N T               $CMDCOM.INC
C-----------------------------------------------------------------------
      CHARACTER*1 NCMND,M1,M2,M3,M4,M5,M6,M7,NNCMND*8
      COMMON /CMND/ NCMND(8),M1(4),M2(4),M3(4),M4(4),M5(4),M6(4),M7(4)
      EQUIVALENCE (NNCMND,NCMND(1))

C----VARIABLE--------DESCRIPTION----------------------------------------
C   NCMND(8)*1    CURRENT COMMAND
C   NNCMND*8 CURRENT COMMAND
C   M1(4) to M7(4)     CURRENT ARRAY NAMES
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C CONTAM  C O M M O N   S T O R A G E                        $CNTCOM.INC
C-----------------------------------------------------------------------
      PARAMETER (MAXSPE=25)
      REAL*8 EP
      COMMON /CNTCM1/NSNOD,NSSPE,NSEQ,MSBAN,NFELM,NKINEL,NESTRT,EP,
     +MPV,MPVCD,MPVM,MPF,MPC,MPE,MPKSEQ,MPWE,MPEFF,MPDIFF,MPGENR,
     +MPKIK(9),MPCONT,MPTEMP
      CHARACTER SID(MAXSPE)*4, CDATA(MAXSPE)*1
      COMMON /CNTCM2/ SID, CDATA
C----VARIABLE--------DESCRIPTION----------------------------------------
C   MAXSPE         MAX NUMBER OF SPECIES CURRENTLY ALLOWED
C   NSNOD          NUMBER OF SYSTEM NODES
C   NSSPE          NUMBER OF SYSTEM SPECIES
C   NSEQ      NUMBER OF SYSTEM EQUATIONS
C             = NSNOD*NSSPE (CURRENT VERSION)
C   MSBAN          (HALF) BANDWIDTH OF SYSTEM EQUATIONS
C   NFELM          NUMBER OF FLOW ELEMENTS
C   NKINEL         NUMBER OF KINETICS ELEMS
C   NKIN      NUMBER OF KINETICS RATE EXPRESSIONS
C   NESTRT         (CURRENT) STARTING ELEMENT NUMBER
C   EP        MACHINE PRECISION
C
C   SID(MAXSPE)*4 SPECIES ID CODES
C   CDATA(MAXSPE)*1   GENERAL PURPOSE CHARACTER CODE DATA
C
C     P O I N T E R S   T O   B L A N K   C O M M O N   L O C A T I O N S
C
C------POINTER----ARRAY-----------DESCRIPTION---------------------------
C
C GLOBAL-----------------------: SHARED BY TWO OR MORE COMMANDS
C   MPKSEQ KSEQ(NSNOD,NSSPE): SYSTEM EQUATION NUMBERS
C               :   0    = UNDEFINED
C               :   NEG = CONCENTRATION PRESCRIBED DOF
C               :   POS = GENERATION PRESCRIBED DOF
C   MPV  V(NSNOD) : ZONE/NODE VOLUMETRIC MASSES
C   MPVCD     VCD(NSNOD)     : CNDF ELEM CONTRIBUTION TO VOL. MASS
C      MPKIK(9) KIK?(NSSPE,NSSPE): KIN RATE COEF. MATRICES: ? = 1..9
C   MPF  F(NSEQ,*)      : FLOW MATRIX (UNSYMMETRIC)
C LOCAL------------------------: USED LOCALLY BY A SINGLE COMMAND
C         MPVM VM(NSEQN)      : VOLUMETRIC MASS MATRIX (DIAG MATRIX)
C   MPC  C(NSEQ)        : CONTAMINANT CONCENTRATION
C   MPE  E(NSEQ)        : CONTAMINANT EXCITATION (CONC.OR GEN.)
C   MPWE WE(NFELM)      : ELEMENT MASS FLOW RATES
C   MPEFF     EFF(NSSPE)     : FLOW ELEM. FILTER EFFICIENCIES
C   MPDIFF    DIFF(NSSPE)    : FLOW ELEM. DIFFUSIVITY OR DISPERSAL COEF
C   MPGENR    GENR(NSSPE)    : FLOW ELEM. GENERATION RATES
C   MPCONT    CONT(NSNOD)    : ARRAY USED FOR CONTINUITY ACCOUNTING
C   MPTEMP              : POINTER FOR MISC. TEMPORARY STORAGE
C
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C CALSAPX  I/O   F I L E   M A N A G E M E N T                $IOCOM.INC
C-----------------------------------------------------------------------
      CHARACTER FNAME*12, EXT*3, MODE*5
      LOGICAL ECHO,EOD,EOC,NODATA
      COMMON /IOCOM1/NTR,NTW,NCMD,NIN,NOT,NPLT,ND1,ND2,ND3,ND4,
     +ECHO,EOD,EOC,NODATA,LFNAME
      COMMON /IOCOM2/ MODE,FNAME,EXT

C----VARIABLE--------DESCRIPTION----------------------------------------
C     /IOCOM/
C   NTR       LOGICAL UNIT NUMBER FOR TERMINAL-READ (KEYBOARD)
C   NTW       LOGICAL UNIT NUMBER FOR TERMINA-WRITE (SCREEN)
C   NCMD      LOGICAL UNIT NUMBER FOR COMMAND/DATA INPUT
C   NIN       LOGICAL UNIT NUMBER FOR INPUT DATA ASCII FILE
C   NOT       LOGICAL UNIT NUMBER FOR OUTPUT DATA ASCII FILE
C   NPLT      LOGICAL UNIT NUMBER FOR PLOT DATA ASCII FILE
C   ND1 thru ND4  LOGICAL UNIT NUMBERS FOR GENERAL USE
C   ECHO      WHEN .TRUE. ECHO RESULTS OUTPUT TO NTW (SCREEN)
C   EOD       END-OF-DATA LOGICAL
C   EOC       END-OF-COMMAND LOGICAL
C   NODATA         NO DATA (FOR DATA IDENTIFIER) LOGICAL
C   LFNAME         LENGTH OF FILENAME WITH TRAILING BLANKS REMOVED
C   FNAME*12 RESULTS OUTPUT FILE NAME
C   EXT*3          RESULTS OUTPUT FILE EXTENSION
C   MODE*4         COMMAND MODE: 'INTER'=INTERACTIVE, 'BATCH'=BATCH
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C CALSAPX  F R E E - F I E L D   I N P U T                  $FRECOM.INC
C-----------------------------------------------------------------------
      CHARACTER LINE*1, LLINE*160
      COMMON /CLINE1/ LINE(160)
      COMMON /CLINE2/ II,JJ
      EQUIVALENCE (LLINE,LINE(1))
      SAVE /CLINE1/,/CLINE2/

C----VARIABLE--------DESCRIPTION----------------------------------------
C   LINE(160)     COMMAND LINE BUFFER
C   II        END-OF-DATA IN LINE BUFFER
C   JJ        END-OF-INFORMATION IN LINE BUFFER
C-----------------------------------------------------------------------
```

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 88-3814	2. Performing Organ. Report No.	3. Publication Date JULY 1988

4. TITLE AND SUBTITLE

Progress Toward a General Analytical Method for Predicting Indoor Air Pollution in Buildings - Indoor Air Quality Modeling Phase III Report

5. AUTHOR(S)

James Axley

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

National Bureau of Standards
Building Environment Division 747
Center for Building Technology
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This interim report presents the results of PhaseIII of the NBS General Indoor Air Pollution Concentration Model Project. It describes; a)a general element-assembly formulation of multi-zone contaminant dispersal analysis theory that provides a general framework for the development of detailed (element) models of mass transport phenomena that may affect contaminant dispersal in buildings. b)an approach to modeling the dispersal of interactive contaminants involving contaminant mass transport phenomena governed by basic principals of kinetics and introduces a linear first order kinetics element to achieve this end, c)an approach to modeling the details of contaminant dispersal driven by convection-diffusion processes in one-dimensional flow situations (e.g., HVAC ductwork) and introduces a convection-diffusion flow element to achieve this end, and d) the features and use of CONTAM87, a program that provides a computational implementation of the theory and methods discussed. The theory and methods presented are based upon a slight generalization of the building idealization employed earlier (Axley, 1987). Here, building air flow systems are idealized as assemblages of mass transport elements, rather than simply flow elements as used previously, connected to discrete system nodes corresponding to well-mixed air zones within the building and its HVAC system. Equations governing contaminant dispersal in the whole building air flow system due to air flow and reaction or sorption mass transport phenomena are formulated by assembling element equations, that characterize a specific instance of mass transport in the building air flow system, in such a manner that the fundamental requirement of conservation of mass is satisfied in each zone

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

contaminant dispersal analysis, inverse contaminant dispersal analysis, tracer gas techniques, building simulation

13. AVAILABILITY

[X] Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
[X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

124

15. Price

$18.95

USCOMM-DC 6043-P80